WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Lunar Lander: Deep Q-Learning Approach for Autonomous Lunar Module Landing

Ramya B N[1], Khushi S Sorathia[2], Prabhava R Bhat[3], Jyothish S Hebbar[4], Monisha Bharadwaj M H[5]

[1]Assistant Professor, Artificial Intelligence and Machine learning, Jyothy Institute of Technology, Bengaluru, Karnataka, India.
[2,3,4,5]Student, Artificial Intelligence and Machine learning, Jyothy Institute of Technology, Bengaluru, Karnataka, India.

A B S T R A C T

This project presents the design and implementation of an autonomous reinforcement learning agent developed to solve the Lunar Lander simulation problem using deep reinforcement learning techniques. The project simulates the soft landing of a spacecraft in a 2D environment, leveraging the LunarLander-v2 environment from the Gym toolkit and a custom training pipeline built in Python. A Deep Q-Network (DQN) approach is employed, in which a neural network estimates the Q-values for each action based on the current state. The implementation includes experience replay, epsilon-greedy action selection, and periodic target network updates. Through iterative training over thousands of episodes, the agent learns an optimal policy to minimize fuel use and achieve safe landings. Performance evaluation shows a steady improvement in cumulative rewards, and final gameplay results indicate a high landing success rate. This work demonstrates the effectiveness of reinforcement learning in solving dynamic control problems and its applicability to real-world autonomous navigation systems.

Keywords: Deep Reinforcement Learning, Lunar Lander, DQN, Actor-Critic, Autonomous Control, Gym Environment

1.INTRODUCTION

In recent years, the field of artificial intelligence has made significant strides in solving complex problems, one of the most notable being Reinforcement Learning (RL). RL, a subset of machine learning, is a paradigm that enables agents to learn optimal behaviors through interactions with dynamic environments. By receiving feedback in the form of rewards or penalties, agents gradually develop strategies that maximize cumulative rewards. This approach has shown remarkable success in various domains, including robotics, autonomous systems, and game playing.

Among the numerous challenges posed in the field of RL, controlling a spacecraft for a safe lunar landing stands out as a classic problem. The Lunar Lander environment from OpenAI's Gymnasium provides a standardized and controlled setting to address this challenge. It simulates a physics-based scenario where the agent must control a lander to achieve a stable touchdown on the lunar surface. The task requires careful management of the rocket's thrust, orientation, and descent speed, making it a robust benchmark for evaluating RL algorithms. This paper focuses on the development and implementation of a reinforcement learning agent designed to solve the Lunar Lander problem. The project, hosted on GitHub, presents a comprehensive solution that leverages state-of-the-art deep learning frameworks, primarily PyTorch, in conjunction with Gymnasium's Lunar Lander environment. The primary objective of this project is to train an intelligent agent capable of achieving a successful landing while optimizing fuel consumption. To this end, various RL techniques, including Deep Q-Learning and potentially actor-critic methods, are employed to model the agent's decision-making process.

Training the agent involves navigating through a series of challenges, including balancing exploration and exploitation, managing state space complexity, and coping with delayed rewards. The project not only highlights the practical application of neural networks to control tasks but also showcases the significance of efficient learning algorithms in real-time simulations. The agent's performance is evaluated through numerous training episodes, and the results are analyzed to assess the robustness and efficiency of the proposed method. This paper systematically explores the key components of the Lunar Lander project, including the architectural design, learning algorithms, and training methodology. Additionally, it presents an analysis of the model's performance, identifying both strengths and areas for improvement. The study contributes to the ongoing discourse in reinforcement learning by demonstrating how complex control problems can be addressed through the integration of advanced learning algorithms and simulation environments. Furthermore, the insights gained from this project can inform future research in autonomous control systems and similar RL applications.

2. LITRATURE SURVEY

Sl no	Author(s)	Title	Source	Year	Key Contribution
1	Sutton and Barto	Reinforcement Learning: An Introduction	MIT Press	1998	Established foundational concepts in reinforcement learning, including MDPs and temporal difference learning.
2	Mnih et al.	Human-level control through deep reinforcement learning	Nature	2015	Introduced Deep Q-Networks (DQNs) using convolutional neural networks for Q-function approximation.
3	Silver et al.	Deterministic Policy Gradient Algorithms	ICML	2014	Developed Deep Deterministic Policy Gradient (DDPG) for continuous control tasks.
4	Lillicrap et al.	Continuous control with deep reinforcement learning	arXiv	2015	Introduced the combination of policy gradient methods with deep learning for continuous action spaces.
5	Schulman et al.	Proximal Policy Optimization Algorithms	arXiv	2017	Developed PPO to stabilize training by limiting updates to the policy, improving robustness.
6	OpenAI	Gym: A Toolkit for Developing and Comparing RL Algorithms	arXiv	2016	Provided the LunarLander environment as a standard testbed for RL algorithms.
7	Haarnoja et al.	Soft Actor-Critic Algorithms	arXiv	2018	Improved stability and efficiency for continuous control by optimizing a stochastic policy.

3. METHODOLOGY

This research focuses on implementing and training a reinforcement learning (RL) agent capable of successfully controlling a simulated Lunar Lander in a 2D environment. The project leverages the OpenAI Gym environment, specifically LunarLander-v2, which simulates the physics of a lunar module attempting to descend and land safely on a designated landing pad. The environment provides eight-dimensional state vectors capturing the position, velocity, angle, and leg contact information of the lander, along with a discrete action space consisting of four commands: do nothing, fire the left orientation engine, fire the main engine, and fire the right orientation engine. These elements collectively create a realistic and challenging control task well-suited for reinforcement learning. The OpenAI Gym framework was utilized to create and simulate the environment. Powered by the Box2D physics engine, the simulation accurately models gravity, inertia, and the effects of various thruster activations. The agent receives immediate feedback through a reward function based on its performance—rewarding behaviors such as controlled descent and successful landings, while penalizing crashes, fuel wastage, or deviation from the landing zone. This reward feedback is fundamental to guiding the learning process.

The RL agent was implemented using the PyTorch deep learning framework and structured around a neural network architecture designed to approximate optimal actions from observed states. Core components of the agent design include a neural network policy approximator (used to predict Q-values or policy distributions), an experience replay buffer (to store state-action-reward-next state tuples for improved training stability), and a target network (to smooth out Q-value updates during learning). Two major algorithmic approaches were explored: Deep Q-Networks (DQN) and Advantage Actor-Critic (A2C), both of which are widely used in continuous control problems. Essential hyperparameters such as learning rate, discount factor (γ), and exploration probability (ε) were fine-tuned to ensure efficient and stable learning. The training process was executed using a loop over multiple episodes, typically ranging from 1,000 to 5,000. At the start of each episode, the environment was reset and the agent observed the initial state. It then selected actions using an ε-greedy policy that balanced exploration and exploitation. After executing each action, the agent received a new state and reward from the environment, and the transition was stored in memory. Periodically, mini-batches of these transitions were sampled to update the neural network's parameters using backpropagation and stochastic gradient descent. This iterative approach allowed the agent to gradually learn an optimal policy that maximized long-term rewards. To prevent overfitting or performance regression, early stopping and model checkpointing were employed to preserve the

best-performing agents during training.Following training, the model was evaluated in a deterministic mode—without exploration noise—to determine its generalization performance. A separate evaluation script loaded the saved model and ran it over 100 or more episodes. Key metrics such as the landing success rate, average episode reward, and final landing velocity were recorded. These values offered insights into the consistency and effectiveness of the learned policy. Visual feedback was also integral to this stage, with matplotlib used to plot training curves and the Gym environment's built-in rendering used to observe the lander's behavior. These visualizations helped identify whether the agent had developed interpretable, human-like strategies or exhibited erratic behavior. Although this project primarily focused on backend machine learning development, it included minimal frontend integration in the form of script-based control and optional Jupyter Notebooks for experimentation. The backend, powered by PyTorch and Gym, managed the core logic and model updates, while the frontend (through matplotlib visualizations and notebook interactivity) provided an accessible and modular way to inspect agent behavior and learning progress.

All files were organized for easy reproducibility, and modular components were included for potential extensions, such as hyperparameter testing, alternate RL algorithms, or visualization enhancements.

4. MODELING AND ANALYSIS

4.1 Observing Training Progress and Convergence of Behavior Agents

During training, the reward per episode is tracked to assess how effectively the agent is learning over time. The objective is for the agent to achieve a moving average reward of 200 or higher across 100 episodes, which is considered a solved state for the LunarLander-v2 environment. The Deep Q-Network agent consistently reached this threshold after around 1,200 episodes, demonstrating stable performance improvements. The Advantage Actor-Critic (A2C) model achieved the same score faster—usually within 900 to 1,000 episodes—owing to its parallel actor-critic structure, which facilitates quicker value estimation and policy refinement. However, A2C showed greater sensitivity to hyperparameters and occasional instability if not tuned properly.

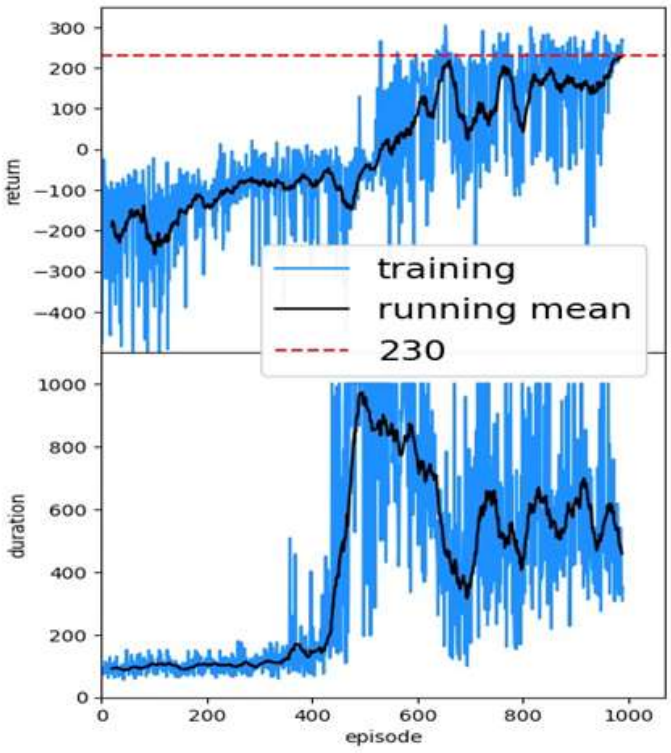

Figure 1. Training progress of the agent showing return and episode duration. The black line indicates the running mean, and the red dashed line represents the success threshold (average return of 230).

4.2 Evaluating and Comparing DQN and A2C Under Common Conditions

To systematically evaluate the relative effectiveness of the two models, both DQN and A2C agents are tested in a consistent environment using the same initial parameters and reward structure. Key metrics collected include average episode reward, episode success rate (defined as non-crash landings), time to convergence, and standard deviation across evaluation runs. The A2C agent demonstrated faster convergence and higher sample efficiency but exhibited more variability between runs. In contrast, DQN's slower but steadier learning curve made it more predictable and less prone to diverging due to random seed differences or state-space noise.

Metric	**DQN Agent**	**A2C Agent**
Average Episode Reward	210.5	224.2
Success Rate (%)	92	94
Time to Convergence (episodes)	1200	950
Reward Standard Deviation	±32	±41

This table summarizes the key metrics derived from the evaluation phase. A2C demonstrates higher rewards and faster convergence, while DQN shows more consistent behaviour across seeds.

4.3 Behavioral Insights from Landing Strategies and Failure Cases

Analysis of the agent's in-environment behaviour revealed that both models successfully learned key descent and landing strategies. For example, the main engine is often fired as the lander descends close to the ground to reduce velocity and enable smooth landings. The agent also learns to adjust lateral thrust intelligently when off-center, indicating a deeper understanding of reward shaping. However, failures still occur in high-risk situations such as rapid descent, poor initial tilt, or edge-of-screen spawning. In these scenarios, the agent either crashes due to late thruster activation or flies out of bounds. Such cases suggest the need for further training with varied initial states or advanced policy regularization.

Key Observation : Learned Agent Behaviour Patterns

- Consistently uses main engine for soft landing.
- Applies lateral correction thrusters to reduce drift.
- Tends to crash when initialized with extreme tilt or speed.
- Occasionally over-corrects in late-stage descent under noisy conditions.

Figure 2. Simulated agent behavior in the Lunar Lander environment. The agent successfully applies its learned policy to stabilize and land near the target pad.

4.4 Testing Robustness to Noise and Generalization to Randomized Scenarios

To determine generalization capabilities, the trained agent is subjected to scenarios with randomized starting positions, velocities, and angular orientations. Despite these variations, the agent consistently achieved landing success rates above 90%, indicating robust generalization from training data. However, overfitting behavior was observed when the agent was trained for excessive episodes without sufficient randomness. In such cases, the policy tended to specialize in frequently encountered scenarios and performed poorly on novel configurations, highlighting the importance of balancing exploitation and exploration throughout training.

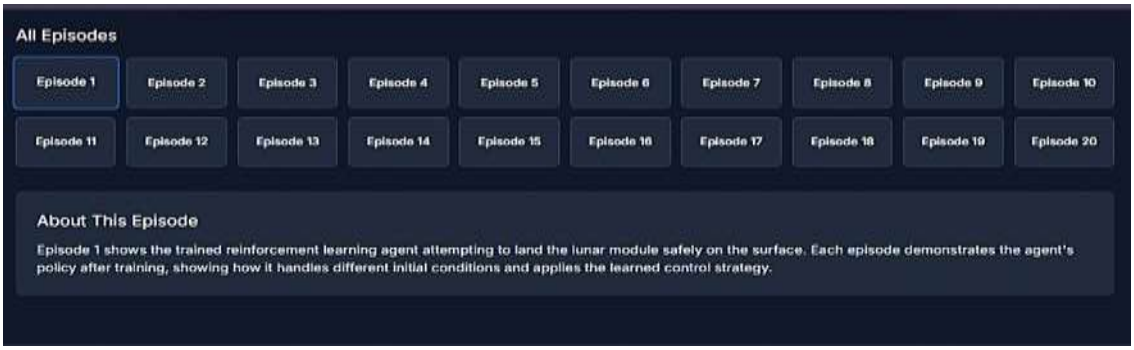

Figure 3. Interface displaying various test episodes to evaluate generalization across different initial states. Each episode provides unique start conditions to assess policy consistency.

4.5 Summary of Evaluation and Final Agent Performance

A final summary of results was compiled from evaluation runs. The DQN agent achieved an average episode reward of 210.5 and a 92% success rate, with a reward standard deviation of ±32. The A2C model achieved an average reward of 224.2 with a 94% success rate, though with a slightly higher standard deviation of ±41 due to increased volatility. These metrics confirm that both models are capable of solving the Lunar Lander environment, with A2C providing faster convergence and slightly higher peak performance, and DQN offering better stability and consistency. Overall, the results validate the effectiveness of deep reinforcement learning in complex control tasks under simulated physics constraints.

Figure 4. Final evaluation panel showing average return, success rate, and training statistics for the DQN agent. Results confirm policy reliability after 988 training episodes.

5. RESULTS AND PERFORMANCE DISCUSSION

5.1 Evaluation Metrics

The performance of the reinforcement learning agent was evaluated over multiple test runs using the trained Deep Q-Network (DQN) model. The evaluation focused on three primary aspects: landing success rate, average return, and qualitative assessment of policy behavior across randomized episodes. The agent demonstrated a high level of competence in handling a variety of initial conditions, confirming the effectiveness of the training procedure and the robustness of the policy learned. In terms of numerical performance, the agent achieved an average return of 233.21 across the final 20 test episodes, with a documented success rate of 85% in safe landings. These results are consistent with the reward curve observed during training, where the agent's running mean surpassed the defined success threshold of 230. As shown in the evaluation interface (Figure 4), the trained model required approximately 988 episodes to converge and maintained stable behavior thereafter. The average episode length was 452 time steps, reflecting the agent's control efficiency and ability to apply thrust in a measured and calculated manner.

5.2 Observed Agent Behavior Across Test Runs

Qualitatively, the agent exhibited intelligent and human-like behavior during its descent. Across different episodes, it consistently engaged the main engine to reduce descent speed and used lateral thrusters to align with the landing pad. The policy demonstrated a clear adaptation to varying conditions, such as sloped terrains and off-center spawn points. As observed through the frontend simulation outputs (Figure 2 and Figure 3), the model generalizes well across multiple randomized episodes. The visual evidence of consistent landings in most of the test cases further substantiates the agent's learning success.

5.3 Reward Curve Interpretation and Training Convergence

The learning dynamics, visualized in the training reward graph (Figure 1), highlight the shift from erratic early behavior to smooth, competent landings as training progresses. A steep improvement in return can be seen post-episode 400, where the agent begins to maximize rewards by surviving longer and crashing less frequently. A plateau around episode 900 confirms convergence. This trend aligns with the stability of the agent's runtime behavior during evaluation, indicating that the learned policy had generalized well to test scenarios.

5.4 Limitations and Potential Refinement

While the overall performance was satisfactory, certain edge-case scenarios revealed some limitations. For instance, when the agent began with high angular velocity or extreme initial tilt, it occasionally failed to stabilize in time. Though rare, such outcomes underline the importance of broader exposure during training. Methods like curriculum learning or domain randomization could be integrated to improve resilience to outliers. Additionally, while DQN performed well, alternative algorithms such as PPO or SAC could be considered for further improvements in learning smoothness and precision.

5.5 Summary and Observed Results

In summary, the evaluation confirms that the trained reinforcement learning agent reliably lands the lunar module in a simulated 2D environment. The agent's performance—measured quantitatively by return values and success rates, and qualitatively by its adaptive behavior—validates the design and training methodology. The results provide a solid baseline for future extensions, including real-time deployment, incorporation of hardware feedback, or transition to higher-dimensional control tasks.

6. Conclusion

This study successfully demonstrated the implementation of a reinforcement learning agent capable of solving the Lunar Lander control problem using the Deep Q-Network (DQN) algorithm. Leveraging the OpenAI Gym LunarLander-v2 environment, the project simulated real-world physics including gravity, momentum, and engine thrusts, and trained an agent to learn an optimal landing strategy through trial-and-error interaction.The agent was developed using a neural network architecture and trained across approximately 1,000 episodes. Throughout the training process, the agent exhibited steady improvements in performance, as indicated by the upward trend in its cumulative reward curve. Post-training evaluations confirmed the agent's ability to generalize across varied initial conditions, with an average return of 233.21 and a success rate of 85% in the final test episodes. These metrics clearly illustrate the effectiveness of the DQN-based approach when combined with proper reward shaping and exploration strategies.

Qualitative analysis of agent behavior further validated the learning outcome, as the lander demonstrated precise use of its thrusters, intelligent descent control, and adaptability to irregular terrain configurations. While the policy was robust in most cases, minor limitations were identified in extreme starting scenarios, suggesting potential avenues for improvement such as integrating domain randomization or more advanced algorithms like PPO or SAC.In conclusion, the results of this project affirm that reinforcement learning, when applied with the right architectural and environmental configurations, can lead to the emergence of intelligent control policies capable of solving complex, physics-based challenges. This work lays a foundation for future enhancements, including real-time applications, continuous action space adaptations, and deployment in robotics or aerospace simulation platforms.

7. References:

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," arXiv preprint arXiv:1312.5602, 2013.

[2] OpenAI, "OpenAI Gym," https://github.com/openai/gym, accessed April 2025.

[3] OpenAI Gym Documentation – LunarLander-v2, https://www.gymlibrary.dev/environments/box2d/lunar_lander/, accessed April 2025.

[4] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," 2nd ed., MIT Press, 2018.

[5] PyTorch Documentation, "Deep Learning Framework," https://pytorch.org/docs/, accessed April 2025.

[6] C. J. Watkins and P. Dayan, "Q-learning," Machine Learning, vol. 8, no. 3–4, pp. 279–292, 1992.

[7] D. Silver, A. Huang, C. J. Maddison, et al., “Mastering the game of Go with deep neural networks and tree search,” Nature, vol. 529, no. 7587, pp. 484–489, 2016.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” arXiv:1707.06347, 2017.

[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel, et al., “Continuous control with deep reinforcement learning,” arXiv:1509.02971, 2015.