Lunar Lander using Continuous Control

Kirsten Odendaal
College of Computing
Georgia Institute of Technology

1 Introduction

Reinforcement Learning (RL) has shown incredible success in solving control problems that require sequential decision-making under uncertainty. However, a continuing challenge in RL is handling continuous action spaces efficiently. Therefore, the Lunar Lander environment from OpenAI Gym presents an excellent test case for RL algorithms due to its complex physics-based dynamics and the need for fine action control when set in continuous mode.

Traditional Q-learning methods, such as Deep Q-Networks (DQN), have proved successful in discrete environments like Atari games (Mnih et al., 2013). However, their extension to continuous spaces is non-trivial due to the need for infinite actions. Therefore, Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2019) is chosen because it can efficiently handle deterministic policies, making it practical for continuous action spaces. DDPG builds upon popular actor-critic architectures and utilizes an off-policy learning approach with experience replay and soft target updates to improve learning stability. This report presents the technical implementation of DDPG for solving the Lunar Lander problem, exploring the effects of hyperparameter tuning and assessing performance in different configurations. Relevant challenges such as gradient explosions and policy convergence issues are also highlighted.

2 Technical Background

Deep Reinforcement Learning (DRL) builds upon traditional RL by incorporating deep neural networks for function approximation, allowing agents to operate in high-dimensional state and action spaces. This section covers key concepts, including Deep Q-Networks (DQN), Policy Gradient methods, Actor-Critic frameworks, and the rationale behind selecting Deep Deterministic Policy Gradient (DDPG) for solving the Lunar Lander problem.

2.1 Q-Learning, Bellman Equations, and DQN

Q-learning is a fundamental RL algorithm that estimates the optimal action-value function $Q^*(s, a)$ using the Bellman optimality equation:

$$Q^*(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} Q^*(s', a') | s, a\right]$$

Where s, a represents the state-action pair, r is the immediate reward, $\gamma \in (0, 1]$ is the discount factor, and s', a' is the next state-action pair. The expectation $\mathbb{E}[\cdot]$ is taken over the environment's transition dynamics. DQN (Mnih et al., 2013) extends Q-learning to high-dimensional state spaces by using a deep neural network to approximate the Q-function:

$$Q(s, a; \theta) \approx Q^*(s, a)$$

where θ are the network parameters. The loss function for training the Q-network is given by:

$$L(\theta) = \mathbb{E}\left[(y - Q(s, a; \theta))^2\right]$$

Where the target Q-value can be represented as:

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

and θ^- represents the parameters of a target network, which is updated separately from the main Q-network to improve training stability by reducing divergence. Ultimately, DQN requires discretizing the action space to compute $\max_{a'} Q(s', a')$, which is computationally challenging in extremely high-dimensional continuous action spaces. Instead of searching over a discrete set of actions, a policy-

based approach can output continuous actions directly, motivating the need for policy gradient methods.

2.2 Policy Gradient Methods

Policy gradient methods directly optimize the policy $\pi_\theta(a|s)$, parameterized by θ, by maximizing the expected return:

$$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$$

using gradient ascent:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R_t\right]$$

where R_t is the cumulative reward from time t onward. Unlike value-based methods, the policy gradient approaches allow learning stochastic policies and are suitable for continuous control (Sutton & Barto, 2018). However, vanilla policy gradients usually struggle with high variance (Fujimoto et al., 2018). A common solution is to introduce a value function (critic) to guide policy updates, leading to Actor-Critic methods. These algorithms reduce high variance by combining policy-based and value-based approaches. The actor ($\pi_\theta(s)$) selects actions, while the critic ($Q_\phi(s, a)$) evaluates them. The critic is usually trained using Temporal Difference (TD) learning:

$$y_t = r_t + \gamma Q_\phi(s_{t+1}, \pi_\theta(s_{t+1}))$$

and the actor is updated via deterministic policy gradient:

$$\nabla_\theta J \approx \mathbb{E}\left[\nabla_\theta \pi_\theta(s) \nabla_a Q_\phi(s, a)|a = \pi\theta(s)\right]$$

Fortunately, this framework can be further extended to consider deep neural networks, making it well-suited for continuous action spaces.

2.3 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2019) extends Deterministic Policy Gradient (DPG) (Silver et al., 2014) with deep neural networks, allowing for efficient learning in high-dimensional continuous action spaces. Unlike vanilla Actor-Critic methods, DDPG stabilizes training using experience replay and target networks, drawing similarities to DQN. Experience Replay is a training cache that stores past transitions (s, a, r, s') to break correlation between consecutive samples. Target networks further improve stability by using separate parameters θ^-, which are updated gradually to reduce variance in learning:

$$\theta^- \leftarrow \tau\theta + (1 - \tau)\theta^-$$

where $\tau \ll 1$ controls update speed.

The DDPG algorithm follows a structured approach, using an actor-critic architecture and experience replay to efficiently learn in continuous action spaces. However, due to it being a deterministic algorithm, conventional exploration techniques are not suitable. For a detailed step-by-step breakdown, refer to the pseudocode in Lillicrap et al. (2019).

2.4 Exploration Strategies for DDPG

Exploration is a fundamental challenge in RL, particularly in off-policy algorithms like DDPG. Since the policy learned by DDPG is deterministic, an independent noise process must be introduced to encourage exploration. The original paper (Lillicrap et al., 2019) proposed an Ornstein-Uhlenbeck (OU) process, but a simpler alternative is to inject zero-mean Gaussian noise directly into the action output (Morales, 2020):

$$\epsilon_t \sim N(0, \sigma^2)$$

where ϵ_t is the noise added at each timestep and σ controls the scale of randomness. The perturbed action is then computed as:

$$a_t = \pi_\theta(s_t) + \epsilon_t$$

Component	Lunar Lander Problem	Description
State Space (S)	$s = [x, y, v_x, v_y, \theta, \omega, c_1, c_2]$	An 8-dimensional vector representing: horizontal and vertical positions, horizontal and vertical velocity, angle and angular velocity, and two boolean ground contact indicators.
Action Space (A)	$a = [a_{\text{main}}, a_{\text{lateral}}]$	Continuous action space where the main engine throttle is controlled via $a_{\text{main}} \in [-1, 1]$, and the lateral boosters via $a_{\text{lateral}} \in [-1, 1]$.
Transition Dynamics	Physics-based simulation	The lander follows Newtonian mechanics, influenced by thrust, gravity, and collisions. Excessive impact forces can cause termination.
Reward Function ($R(s, a)$)	$R = R_{\text{landing}} + R_{\text{distance}} + R_{\text{fuel}} + R_{\text{crash}}$	(+) Reward for moving toward the landing pad; (−) Penalty for moving away. (+) 100 points for a successful landing (−) 100 points for a crash. (+) 10 points per leg-ground contact. (−) 0.3 points per main engine activation. - Landing outside the pad is allowed. - Infinite fuel enables multiple attempts.
Episode Termination	Defined by crash, out-of-bounds, or stability conditions	The episode ends if: - the lander crashes, - moves out of bounds, or - remains immobile for an extended period.
Environmental Parameters	Gravity, wind effects, turbulence	Adjustable stochastic parameters such as wind power and turbulence influence agent dynamics. Note these are maintained at the default provided values.

Table 1: Lunar Lander Problem Components

This latter method provides a simpler implementation by eliminating the need to tune additional parameters such as the mean reversion rate and proves effective in high-dimensional spaces, making it particularly well-suited for DDPG, which operates in continuous action domains.

3 Problem Definitions

The Lunar Lander environment, which is a part of OpenAI Gym, presents a solid benchmark problem. This case simulates the challenge of controlling a spacecraft to land safely on a designated pad. The following section details the problem dynamics, action and observation spaces, and reward structure.

3.1 Environment Description

The environment consists of a physics-based trajectory optimization task where an agent must control a lander's thrusters to achieve a smooth and precise landing on a designated pad while minimizing velocity and excessive movement (Klimov, 2016; Brockman et al., 2016). The environment incorporates realistic physics, including gravity, velocity, angular momentum, and thrust forces, making it a challenging RL problem. Although fuel is unlimited, optimal policies must successfully land with minimal thruster use. This work focuses on the continuous action space, where thrusters allow smooth adjustments for precise landing control. As indicated by Klimov (2016), the problem is solved when an average score ≥ 200 over 100 consecutive runs is achieved. The key components of the environment, such as termination conditions and reward structures, are summarized in Table 1.

3.2 Overcoming Critical Challenges

Continuous control requires fine-grained thruster adjustments, making policy learning much more difficult than in discrete settings. Additionally, the reward structure is sparse, providing meaningful feedback only on successful or failed landings. This delayed feedback complicates credit assignment since early actions can significantly influence the outcomes. Therefore, a focus on long-term strategy optimization is crucial. Lastly, the deterministic state-action space increases the complexity of exploration, requiring an effective balance between exploration and exploitation strategies. To overcome these difficulties, the DDPG algorithm is implemented. DDPG is inherently suited for continuous control tasks (see Section 2.3), making it an appropriate choice for tackling the challenges presented by the Lunar Lander environment.

Hyperparameter	Defaults	Search Range
Discount Factor (γ)	0.99	[0.90, 0.99]
Soft Target Update (τ)	0.001	[0.0001, 0.01]
Batch Size (b)	32	{32, 64, 128}
Update Frequency (n_f)	1	{1, 2, 3, 4, 5}
Buffer Size (e_t)	100,000	-
Noise ratio (ϵ_s)	0.1	-
Total Episodes (n_{ep})	1,000	-

Table 2: Hyperparameter Optimization Variable Ranges

Parameter	Actor	Critic
Input Dimension	8 (state)	10 (state + action)
Hidden Layers	2	2
Neurons per Layer	256	256
Activation (Hidden)	ReLU	ReLU
Activation (Output)	Tanh	Linear
Optimizer	Adam	Adam
Learning Rate	10^{-4}	10^{-4}
Output Dimension	2 (action)	1 (Q-value)

Table 3: Neural Network Hyperparameters for Actor and Critic Networks

4 Methodology

This section outlines the implementation of the DDPG algorithm for solving a continuous action space. A description of the training and evaluation procedure, neural network architectures, and the hyperparameter optimization approach is detailed.

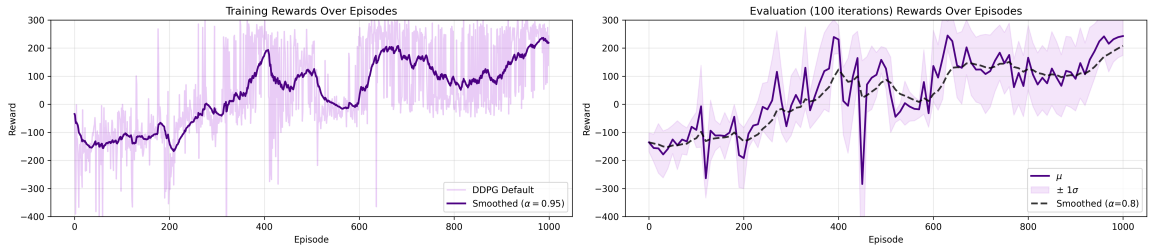
4.1 Training and Evaluation Process

The training and evaluation pipeline follows a structured approach to ensure stability and reproducibility. The process is outlined as follows:

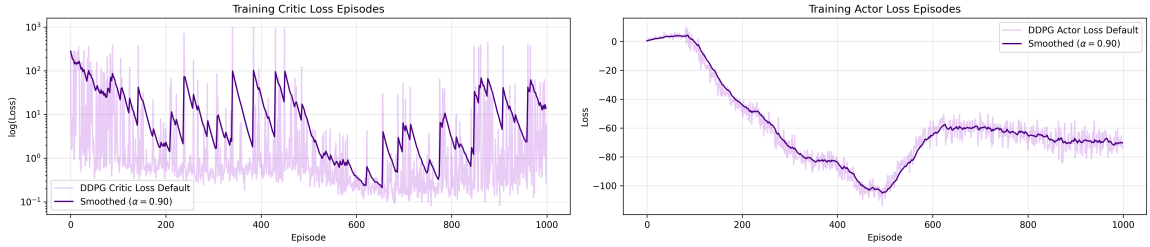
- A baseline model is first trained using default hyperparameters, inspired by prior research (Lillicrap et al., 2019). The default configuration is presented in Table 2.
- Training performance is monitored and logged at each step by measuring cumulative rewards and the loss functions of both the actor and critic networks for each consecutive episode.
- Every 10 episodes, the trained model is evaluated using a deterministic policy (a greedy strategy with no noise injection). Each evaluation consists of 100 simulations, including a final evaluation after the last training episode.
- Training was conducted for 1000 episodes, each with a maximum duration of 1000 timesteps. The average wall-clock training time per run is approximately 1.0 hours on a 4-core CPU (Lightning.AI infrastructure).
- After a baseline evaluation, the same training and evaluation process is repeated for 20 additional trials using the Optuna hyperparameter optimization framework (Akiba et al., 2019).

4.2 Hyperparameter Considerations

Training of a robust RL agent is generally highly sensitive to hyperparameters, thus a systematic tuning strategy is required. Therefore, Bayesian optimization via Optuna is applied to identify the optimal values for key parameters. This method is conventionally more efficient that that of either structured grid or random search approaches (Akiba et al., 2019) as it accounts for interaction effects. Ideally, such an optimization would include all attributes such as replay buffer characteristics, exploration noise, episode duration, and function approximation details. However, to ensure computational efficiency, this study focuses on four key DDPG parameters known to impact outcomes. The *Discount Factor* impacts the effects of long-term versus short-term rewards which can greatly vary depending on sparsity. *Soft Target Update* impacts how quickly the target model changes thus effecting the corresponding gradient flow. *Batch Size* considers the trade-off between learning stability and computational cost. Finally, *Parameter Update Frequency* considers the number of steps before any soft network updates are applied. The parameter defaults and hyperparameter search spaces are summarized in Table 4. A total of 20 unique optimization trials were conducted, where each configuration was evaluated based on the average cumulative reward over the final training episodes.



(a) Default Learning Curves for (Left) Training and (Right) Evaluation



(b) Default Loss Curves for (Left) Critic and (Right) Actor

Figure 1: Training reward and loss progression of DDPG with default hyperparameters. (Top-Left) The learning curve of the training is presented for 1000 epochs with an exponential smoothing curve to better visually identify the global trends. (Top-Right) The corresponding evaluation points every 10^{th} epoch is presented, including the mean evaluation and standard deviations. (Bottom-Left) The Critic losses over the 1000 training epochs. (Bottom-Right) The Actor losses over the 1000 training epochs.

4.3 Neural Network Architectures

The actor-critic framework in DDPG relies on deep neural networks to approximate the policy and value functions. The network architectures were designed to efficiently handle the 8-dimensional state space and 2-dimensional continuous action space while maintaining training stability. Table 3 summarizes the network parameters used in the study and experiments. The actor network outputs continuous action values normalized between -1 and 1 using the Tanh activation function. The critic network estimates the Q-value of state-action pairs using a linear activation in the output layer. Both networks were optimized using the Adam optimizer with the same learning rates. To reduce instability, gradient clipping in the range [-5, 5] was applied to both networks, as baseline results showed large gradient magnitudes causing spikes in the loss function. This helped prevent exploding gradients, ensuring smoother policy updates and stable convergence.

5 Results and Discussion

This section presents the experimental results of training a DDPG agent on the Lunar Lander problem, following the structure outlined in Section 4. In addition to model evaluation and optimization, insights into parameter influence, general challenges, and potential future work is highlighted.

The evolution of cumulative rewards over each training episode and the corresponding loss curves are illustrated in Figure 1. During the early phase (Episodes 0–300), the agent displayed highly unstable behaviour, with large fluctuations in reward. This instability is likely due to the limited policy exploration. Since DDPG relies on off-policy learning with a replay buffer, sufficient experience accumulation is required before effective learning can occur. As training moved into the mid-phase (Episodes 300–700), the agent developed improved policies, as demonstrated by the increasing trend in cumulative rewards. However, the variance in rewards remained relatively high. Interestingly, a noticeable dip in cumulative rewards was observed during this phase. This phenomenon was likely due to instability in the critic network, which evaluates state-action pairs. Inspection of the loss curves reveals large spikes and an increasing trend in actor and critic loss instead of the expected de-

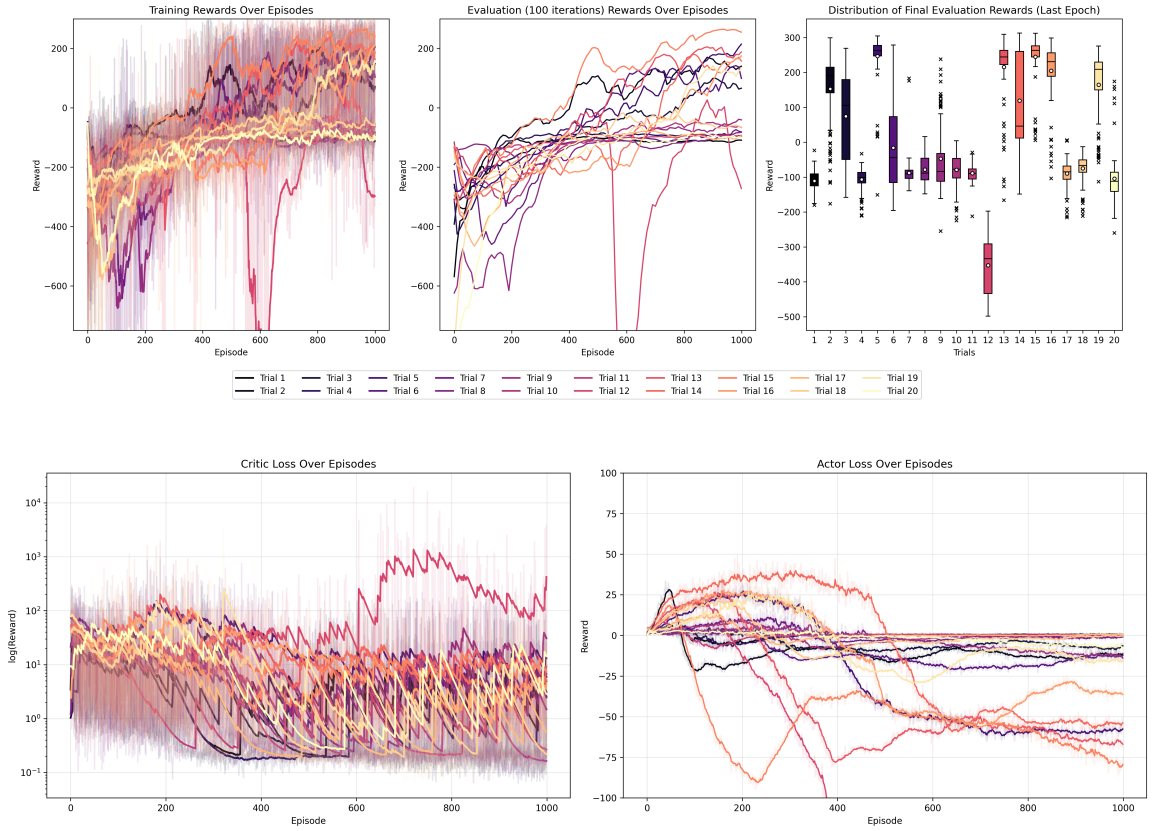


Figure 2: Top: Training reward progression of DDPG with default and optimized hyperparameters. (Left) Learning curves over 1000 epochs with exponential smoothing ($\alpha = 0.95$) for trend identification. (Middle) Evaluation points every 10^{th} epoch with smoothing ($\alpha = 0.8$), excluding standard deviation for clarity. (Right) Final evaluation (last epoch) box plot comparisons across all trials. Bottom: Trial Loss curves for (Left) Critic and (Right) Actor

crease. This behaviour implies potential issues in gradient propagation within the neural networks. Since no gradient clipping or regularization strategies were implemented, the optimization process may have struggled from inconsistent gradient updates, leading to large performance oscillations. In the final phase (Episodes 700–1000), the policy tends towards convergence and improved stability, with rewards routinely approaching the 200-point success threshold. However, the observed oscillatory behaviour suggests that while the agent has learned, it might not have fully converged into an optimal policy. This reinforces hyperparameter optimization's need to refine the learning process, particularly in tuning the soft target update parameter (τ), discount factor (γ), batch size (b), and update frequencies (n_f) to ensure more stable training dynamics.

The primary optimization objectives were to improve training stability, reduce reward variance, enhance sample efficiency, and identify key parameters affecting policy performance. Figure 2 presents the optimization results across various trials, showing that hyperparameter tuning can improve convergence and stability. As shown in the smoothed reward curves, some policies clearly outperformed those trained with the default DDPG configuration in specific trials. These optimized agents reached stable performance more quickly and exhibited reduced variance, as reflected in the final evaluation box plots. Notably, $Trial - 4$ and $Trial - 15$ showed impressive improvements, with the best trials exceeding 200 cumulative reward points. Furthermore, optimized configurations led to more monotonic reward growth, thus indicating structured and stable learning across the entire training range. Table 4 summarizes the final collected results, comparing performance over the final epoch, the last

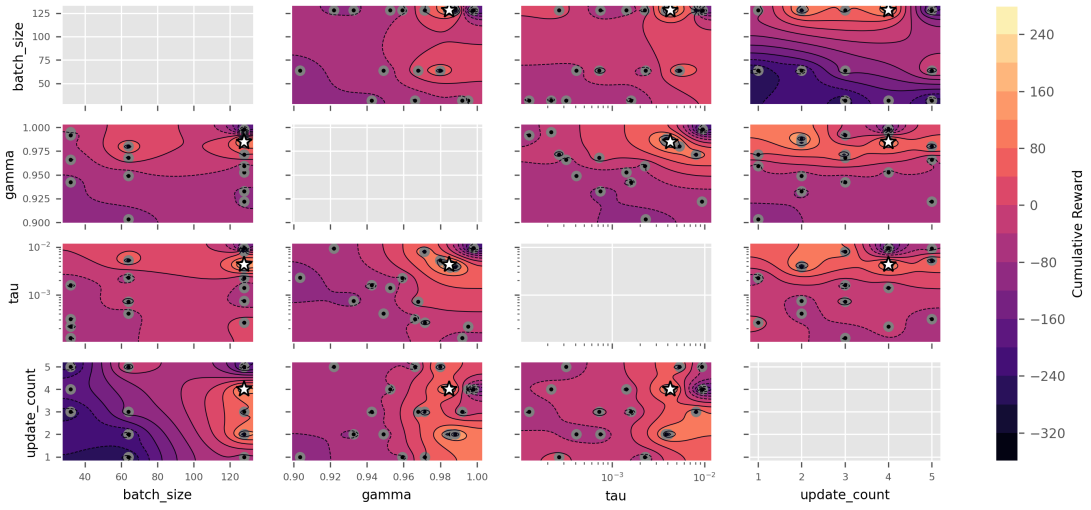


Figure 3: Contour maps illustrating the impact of hyperparameter choices and their interactions on cumulative reward. The colour gradient represents the cumulative reward achieved across different hyperparameter settings. The white star ($\star$) denotes Trial-4, which attained the highest cumulative reward during the final epoch evaluation

100 epochs, and the entire training process. The results show that hyperparameter optimization outperformed the default configuration across all metrics. However, the heuristic approach significantly outperformed all cases, suggesting that while reinforcement learning (RL) is powerful, it may not always be the optimal solution in all settings.

Model	**Hyperparameters**				**Results**		
	γ	τ	b	n_f	**Epoch (1)**	**Epoch (100)**	**Epoch (All)**
Default	0.99	0.001	32	1	249.03 ± 56.58	199.59 ± 104.63	44.72 ± 166.15
HypOpt (T-4)	0.985	0.0043	128	4	$\mathbf{264.41 \pm 23.29}$	197.03 ± 112.70	-22.84 ± 205.416
HypOpt (T-15)	0.988	0.0038	128	2	245.20 ± 66.42	$\mathbf{260.45 \pm 46.023}$	$\mathbf{81.43 \pm 194.83}$
Heuristic	-	-	-	-	289.72 ± 14.22	-	-

Table 4: Hyperparameter Optimization Results Summary and Final Comparison

Despite these improvements, the default configuration performed well. Ultimately, this was to be expected given that the initial hyperparameters were chosen based on well-established parameter ranges from prior work (Lillicrap et al. (2019); Mnih et al. (2013); Morales (2020)).

5.1 Parameter Influence Analysis

The relationship between hyperparameters and cumulative rewards was analyzed to assess their impact on performance. Figure 3 presents contour plots for various interactions of batch size (b), discount factor (γ), soft update rate (τ), and update frequency (n_f). Globally, higher discount factor (γ) values (≥ 0.98) generally improve performance and stability by prioritizing long-term rewards. However, the effect is nonlinear, with localized minima indicating strong interactions with other hyperparameters, such as batch size. A sharp performance drop is observed beyond $\gamma > 0.99$, which is reasonable, as some degree of short-term guidance is necessary to effectively control complex actions in a continuous space. The soft update rate (τ) exhibits an optimal range around 0.004-0.005, where cumulative rewards peak. This suggests that moderate updates strike a balance between stability and learning efficiency. A τ-value that is too high leads to excessive target network fluctuations, destabilizing learning, while too low results in slow adaptation and inefficient policy updates. The observed range aligns with findings from Lillicrap et al. (2019), where maintaining a smoothly evolving target network reduces variance in Q-value estimation and prevents divergence in off-policy learning. A batch size of 128 gives the highest rewards. However, as the value lies at the edge of the tested range, further exploration of larger batch sizes might show further performance gains. Larger batch sizes generally improve learning stability by providing more accurate gradient estimates,

reducing variance in updates. However, too large batches can slow convergence. Update frequencies between 2 and 4 are associated with high-reward regions, suggesting that moderately spaced updates enhance stability and performance. Slower updates reduce the risk of propagating noisy gradient estimates, allowing the policy and value functions to converge more smoothly. Frequent updates, while potentially accelerating learning, can introduce instability by amplifying errors in the critic network. These outcomes clearly demonstrate the importance of systematic hyperparameter tuning to balance stability, efficiency, and convergence speed in reinforcement learning.

5.2 General Challenges

Despite improvements through hyperparameter tuning, several challenges remain. Training deep RL models remained computationally expensive, with each DDPG agent requiring approximately 1.0 hours for 1000 episodes. The high computational demand was caused by large batch sizes, frequent network updates, and extended training durations necessary for policy convergence. Gradient-related instabilities are another concern. While gradient clipping helped reduce fluctuations in critic loss, instances of vanishing gradients in the actor network were still observed and likely due to the Tanh activation function. Tanh squashes outputs into $[-1, 1]$ ranges, causing gradients to diminish for saturated neurons, especially in deeper networks. Applying batch normalization or experimenting with alternative activation functions like Leaky ReLU could improve gradient flow and network stability. Despite improvements through hyperparameter tuning, convergence instability was still observed in trials. One factor contributing to this is Q-value overestimation in the critic network, a well-known issue in DDPG due to its use of bootstrapping and function approximation (Fujimoto et al., 2018). Overestimation bias occurs when the critic systematically assigns overly optimistic Q-values, leading the actor to favour suboptimal policies. This results in erratic learning curves and policy divergence, even when other hyperparameters are well-tuned.

5.3 Future Work

Several directions for future work could enhance the robustness and efficiency of the following investigation. A critical area for improvement is exploring alternative algorithms such as Twin Delayed Deep Deterministic Policy Gradient (TD3), Proximal Policy Optimization (PPO), or Soft Actor-Critic (SAC). TD3 shrinks Q-value overestimation, SAC enhances exploration via entropy regularization, and PPO provides stable, on-policy optimization. A comparative study of these methods against DDPG would provide deeper insights into the trade-offs between stability and sample efficiency in continuous control tasks. TD3 is a natural extension to DDPG, involving only minor structural modifications. One modification already explored is adjusting the number of update frequencies. The outcomes support the benefits of TD3, which has been shown to greatly outperform DDPG (Fujimoto et al., 2018). Further tuning of network architectures presents another opportunity for improvement. Investigating deeper networks, alternative activation functions (Leaky ReLU), and normalization techniques (Batch Normalization) could improve gradient flow and stability. Reproducibility remains a key concern in reinforcement learning research (Fujimoto et al., 2018). To address this, future work should incorporate experiments across multiple random seeds with standardized evaluation metrics to verify the generalizability of the results. Finally, while the current study systematically explores hyperparameter optimization, the depth of the exploration is likely too shallow to extract the maximum benefits. The study should be expanded to consider more parameters related to both DDPG and function approximation architectures.

6 Conclusion

This study implemented and analyzed the DDPG algorithm for solving the Lunar Lander problem in continuous space. The primary objective was to enable precise thrust control for stable and efficient landings. Default characteristics led to high variance and instability, but hyperparameter optimization significantly improved training performance, reducing variance and accelerating convergence in some cases. Despite these improvements, challenges such as high computational cost and gradient instability remained. These outcomes highlight the importance and necessity of hyperparameter tuning in RL. Future work should explore alternative algorithms like TD3 and SAC,

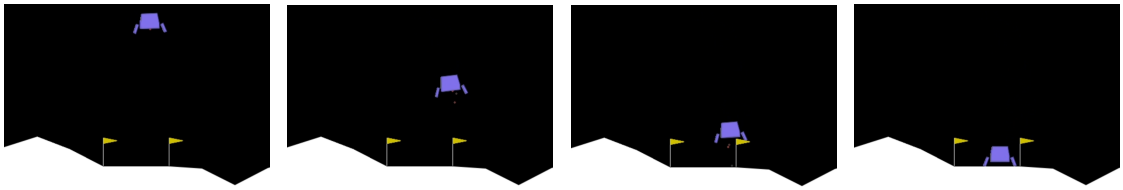


Figure 4: Successful environment snap-shots of a Lunar Landing scenario

improved network architectures, and advanced exploration strategies to enhance learning stability and efficiency. While conventional heuristics perform in this case, the complexity of the real-world environments cannot always be fully captured through handcrafted control strategies. As such, this study highlights the effectiveness of deep RL learning for continuous control while emphasizing the critical role of hyperparameter selection in achieving optimal performance.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. URL https://arxiv.org/abs/1606.01540.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018. URL https://arxiv.org/abs/1802.09477.

Oleg Klimov. Lunarlander-v2 environment for openai gym. https://www.gymlibrary.dev/environments/box2d/lunar_lander/, 2016. Accessed: 2025-02-15.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019. URL https://arxiv.org/abs/1509.02971.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. URL https://arxiv.org/abs/1312.5602.

M. Morales. *Grokking Deep Reinforcement Learning*. Manning, Shelter Island, NY, 1st edition, 2020.

David Silver, Guy Lever, Nicolas Manfred Otto Heess, Thomas Degris, Daan Wierstra, and Martin A. Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, 2014. URL https://api.semanticscholar.org/CorpusID:13928442.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 2nd edition, 2018.

A Available Repository

All code and supplementary materials used in this research are available in a GitHub repository. The repository can be accessed at: https://github.gatech.edu/gt-omscs-rldm/7642RLDMSpring2025kodendaal3 where the latest commit hash is: