# Comparison of Three Deep Reinforcement Learning Algorithms for Solving the Lunar Lander Problem

Dingli Shen

School of Software, North University of China, Taiyuan, Shanxi, 030051, China
631401120114@mails.cqjtu.edu.cn

**Abstract.** The Lunar Lander problem presents a formidable challenge in the realm of reinforcement learning, necessitating the creation of autonomous spacecraft capable of safe landings on the lunar surface. In this study, three prominent reinforcement learning algorithms, namely Deep Q-Network (DQN), Double Deep Q-Network (DDQN), and Policy Gradient, are investigated and examined to address this problem. Initially, DQN algorithm, combining neural networks and Q-Learning, is leveraged to learn an optimal landing policy. By approximating Q-Values through neural network training, the spacecraft learns to make informed decisions, leading to successful landings. Subsequently, DDQN algorithm, which mitigates overestimation bias, is used. By utilizing two neural networks - one for action selection and the other for evaluation - DDQN improves stability and convergence, resulting in refined landing policies. Furthermore, this work explores the application of Policy Gradient methods for this problem. By directly optimizing the policy using gradient ascent, the spacecraft maximizes cumulative rewards, achieving efficient and accurate landings. The performance of the algorithms is assessed through extensive simulations that encompass diverse lunar surface conditions. The results demonstrate the effectiveness of these methods, showcasing their capability to facilitate successful and fuel-efficient spacecraft landings. In conclusion, this study contributes to the understanding of DQN, DDQN, and Policy Gradient algorithms for the Lunar Lander problem. The findings highlight the unique strengths of each algorithm and their potential in autonomous spacecraft landing. The insights gained from this research have implications for the development of intelligent landing systems in future lunar missions, advancing the field of reinforcement learning in aerospace applications.

**Keywords:** Reinforcement Learning, Lunar Lander Problem, Deep Learning.

## 1 Introduction

The Lunar Lander problem is a simulated task that involves training an intelligent system to control a lander and achieve a safe landing on the Moon [1,2]. The lander has thrust control, and the goal is to adjust the thrust to achieve a smooth and accurate landing in each initial state within a target area. The challenge lies in finding the right


B. H. Ahmad (ed.), *Proceedings of the 2023 International Conference on Data Science, Advanced Algorithm and Intelligent Computing (DAI 2023)*, Advances in Intelligent Systems Research 180,
https://doi.org/10.2991/978-94-6463-370-2_21

thrust strategy to avoid accidents and landing failures in the low-gravity and complex terrain conditions of the Moon. Through interaction with the environment and guidance from reward signals, the intelligent system learns how to adapt to different states and make optimal decisions to successfully complete the landing task.

There are several key reasons for using reinforcement learning methods to solve the Lunar Lander problem [3,4]. Firstly, in the Lunar Lander problem, it involves a complex environment characterized by continuous state and action spaces, as well as unknown system dynamics. Traditional solution methods may struggle to effectively model and address such complexity. Secondly, reinforcement learning offers a flexible and highly adaptive approach to tackle such complex problems. Reinforcement learning algorithms can learn from interactions with the environment without requiring prior knowledge of the problem. In the case of the Lunar Lander problem, there is no need to have prior understanding of the complete physics model or the optimal strategy. Reinforcement learning algorithms can autonomously learn and improve through trial and error and optimization processes based on interactions with the environment. Thirdly, Reinforcement learning algorithms exhibit strong adaptability and generalization capabilities when faced with different tasks and environmental variations. For the Lunar Lander problem, there may be various initial conditions and task objectives. Reinforcement learning can adapt to these variations by learning from experience and finding optimal strategies to accomplish the tasks. Lastly, The Lunar Lander problem necessitates the intelligent system to make decisions without human intervention. By interacting with the environment, reinforcement learning methods empower the intelligent system to acquire an optimal strategy and autonomously make decisions without the need for predefined rules.

The Lunar Lander problem has practical applications in many aspects. Firstly, considering that spacecraft landing on the moon is devoid of any human intervention and relies solely on the capabilities of the machine itself, the Lunar Lander problem can be directly applied to the landing phase of space missions on Mars or the moon. By employing reinforcement learning methods, an intelligent system can learn to control the lander and achieve safe and precise landings in diverse landing scenarios. Secondly, the Lunar Lander problem can also be applied to the scenario of automated parking for self-driving cars. Through interactions with the environment, the intelligent system can learn appropriate control strategies, enabling the car to achieve accurate parking in various parking scenarios. This would greatly facilitate people's parking operations and help prevent accidents caused by human errors. Thirdly, the Lunar Lander problem can be utilized for robot navigation and manipulation tasks. By learning from reward signals in the environment, robots can acquire the skills to navigate and perform tasks in complex environments, such as avoiding obstacles and grasping objects. Lastly, the Lunar Lander problem can also be regarded as a game control problem, where reinforcement learning can be applied to flying or landing tasks within a game context. The intelligent system can learn the optimal strategies by interacting with the game environment, aiming to achieve high scores or complete specific objectives.

# 2 Method

## 2.1 Dataset

The environment implementation for the entire problem is based on the Gym environment, and the data used is sourced from Gym [5]. The specific data description of the Lunar Lander problem may be as follows:

Action Space: Push left, push down, push right, no thrust are four discrete actions in space.

Observation Space: lander's horizontal and vertical position, velocity, angle, angular velocity, and other relevant parameters are included in the observation space.These observations provide information about the lander's current state and the environment, enabling the agent to make appropriate decisions.

Rewards: The reward function is used to evaluate the agent's performance after taking specific actions at each time step. In the Lunar Lander problem, a common reward rule is to provide high rewards for successful landings and lower rewards or penalties for unsuccessful landings.

Starting date: The starting state refers to the initial state of the lander when the simulation environment begins. The choice of starting state can have an impact on problem-solving, and it should be set based on specific requirements.

Terminal States: Terminal states indicate the completion or failure of the lunar landing task. For example, the task may end when the lander successfully touches down in the target area, or when a landing failure occurs or the designated time limit is exceeded.

## 2.2 Models

**Deep Q-Network (DQN).** Before introduce this algorithm, the Q-Learning algorithm will be introduced, it is the fundamental of the DQN algorithm [6]. Q-Learning, a type of reinforcement learning, operates without a model and can be considered as an asynchronous dynamic programming approach. It equips agents with the ability to learn optimal actions in Markovian domains by directly experiencing the outcomes of their actions, eliminating the need for domain mapping.

During Q-Learning, the agent engages in interactions with the environment. It observes the current state, selects actions accordingly, and receives rewards as feedback. The primary objective is to determine an optimal policy that maximizes the total accumulated reward for each state. At the core of Q-Learning is the concept of the Q-Value function, which estimates the value of taking a specific action in a given state. This Q-Value denotes the expected cumulative reward for executing a particular action in a specific state. The Q-Value function is iteratively updated through the learning process in Q-Learning.

DQN is an improved algorithm that builds upon Q-Learning and employs a deep neural network estimate the Q-Values for different state-action pairs. [7]. Unlike traditional Q-Learning, DQN utilizes a deep neural network as a function approximator, with the state as input and the output being the corresponding Q-Values

for each action. By storing and randomly sampling previous experiences in an experience replay buffer, DQN can train the deep neural network stably and reduce the fluctuation of targets by using a target network. These improvements enable DQN to handle complex problems with high-dimensional state spaces and enhance the algorithm's convergence and stability.

Therefore, Q-Learning is the foundation of DQN, it addresses the limitations of traditional Q-Learning in high-dimensional state spaces and training instability. It achieves this by incorporating deep neural networks for function approximation and introducing techniques like experience replay. DQN demonstrates better performance and scalability in handling complex tasks and large state spaces.

**Double Deep Q-Network (DDQN).** It is an improvement over DQN aimed at mitigating a problem called overestimation in DQN [8]. In DQN, the selection of Q-Values is based on maximizing the Q-Value from the target network. However, this can lead to an overestimation of the value for certain actions. To address this issue, DDQN introduces an additional neural network to evaluate the value of the action chosen for maximization. In other words, DDQN uses the target network to select actions but uses the current main network to evaluate the value of that action. By doing so, DDQN can reduce the impact of the overestimation problem and obtain more accurate Q-Value estimates.

Therefore, DDQN is an improvement over DQN that mitigates the overestimation problem by using an additional neural network to evaluate action values. This improvement can enhance the performance and stability of the algorithm and, in some cases, result in more accurate strategies.

**Policy Gradient.** It is a gradient-based reinforcement learning algorithm used to learn the optimal policy [9]. Unlike value-based methods like Q-Learning, policy gradient directly optimizes the policy itself, avoiding the need for value function estimation and making it well-suited for problems that involve continuous action spaces.

The fundamental concept behind policy gradient is to optimize the policy through maximizing the expected value of cumulative rewards. To achieve this, a policy function $\pi(a|s)$ is defined, which entails choosing an action "a" by considering the current states according to a probability distribution. Then, the optimal policy is learned by updating the policy function based on its gradient. Specifically, the update rule for policy gradient algorithm is:

$$\theta = \theta + \alpha * \nabla_\theta J(\theta) \tag{1}$$

where the parameter $\theta$ corresponds to the policy function $\pi$, and $\alpha$ represents the learning rate, $J(\theta)$ is the expected value of cumulative rewards, and $\nabla_\theta J(\theta)$ is the gradient of $J(\theta)$ in terms of $\theta$. The update rule means that a series of actions are taken under the current policy, compute the expected value of cumulative rewards based on these actions, and then by iteratively adjusting the parameters of the policy function in the direction of the gradient to maximize the expected cumulative rewards.

**Reinforce Algorithm.** For Reinfoorce, it is a specific algorithm under Policy Gradient methods [10]:

At each iteration, an episode is sampled, and then the parameters are updated. In Reinforce algorithm, the return of the current episode is used

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t} R_k \tag{2}$$

This return, $G_t$, is used to update the parameters θ of the policy function qπ($S_t$, $A_t$). So, it can get the formula of Reinforce algorithm:

$$\theta_{t+1}=\theta_t + \alpha G_t \nabla ln\pi(A_t|S_t, \theta_{t+1}) \tag{3}$$

If $G_t$ is greater than zero, the direction of parameter update will enhance the likelihood of choosing the current action in the current state. This means that if the return is favorable, the probability of selecting that action will be increased. Moreover, the larger the return, the larger the magnitude of the gradient update, resulting in a greater increase in probability.

### 2.3 Evaluation Indicators

In this section, the effectiveness of algorithms are measured by using several metrics. Below are some relevant descriptions:

Learning speed: This metric measures the algorithm's processing speed in handling a problem within the same episodes. It indicates that among algorithms with a fixed number of episodes, the one that completes the learning process the fastest is considered to have the optimal learning speed.

The amount of reward: Rewards are generally divided into two types: average reward and maximum reward. Average reward refers to the average reward obtained in each episode, while maximum reward represents the highest reward achieved in a specific episode. It is generally considered better when the average reward value is larger and the maximum reward in a single episode is the highest.

Convergence rate: It refers to the speed or rate at which an algorithm or process approaches a desired or optimal solution. It measures how quickly the algorithm converges or reaches a stable state. A faster convergence rate indicates that the algorithm reaches the desired solution more quickly, requiring fewer iterations or computational steps. On the other hand, a slower convergence rate means that the algorithm takes longer to converge, requiring more iterations or computational effort to reach the desired solution.

Noise: It refers to random or irrelevant fluctuations, outliers, or errors present in the data. It can arise due to uncertainties in the measurement or data collection process, sensor noise, interference during data transmission, labeling errors, and other factors. In general, noise is measured by the magnitude of fluctuations in an image. If the fluctuations in an image are large, it indicates a higher level of noise.

# 3 Result

## 3.1 Environment and Experiment Design

In the OpenAI Gym environment, there is a Lunar Lander experiment environment called "LunarLander-v2" that is used for machine learning experiments and algorithm development. This environment is leveraged to do the experiment.

Task Objective: The objective of Lunar Lander is to control a lunar lander spacecraft to achieve a smooth landing on the lunar surface. The agent needs to learn how to adjust the lander's thrust and angle in the given environment to minimize fuel consumption and maintain stability during landing.

State Space: The state space in the LunarLander-v2 environment includes the lander's various position and velocity information, and whether it is in contact with the lunar surface. These state information is provided to the intelligent agent for decision-making and action selection.

Action Space: The agent can take the actions mentioned before. The target of agent is to keep studying to choose appropriate actions to control the lander and achieve the landing objective.

Reward Mechanism: The reward mechanism in the LunarLander-v2 environment evaluates the agent's actions by providing rewards. A high positive reward is given for a successful landing while penalizing unstable actions. Additionally, a significant negative reward is given if the lander crashes or leaves the game area.

Termination Condition: The experiment terminates when the agent completes the task or reaches the maximum number of steps. The completion condition is achieved when the lander successfully lands.

## 3.2 Comparison between DQN and DDQN

When using DQN and DDQN to solve the Lunar Lander problem, determining in choosing the hyperparameters, learning rate and discount factor are really significant. Here are some descriptions to these two hyperparameters:

Learning Rate: The learning rate determines the step size when updating the neural network weights. Choosing an appropriate learning rate balances the training speed and stability. If the network weights change too slowly or the training does not converge, the learning rate could be increased. Conversely, if the training process is unstable or diverges, the learning rate could be decreased.

Discount Factor: The significance of future rewards is measured by the discount factor. A higher discount factor (e.g., 0.99) implies that the agent assigns greater importance to rewards in the distant future. This is beneficial for learning long-term goals and planning. However, an excessively high discount factor can lead to unstable training or slow convergence. A lower discount factor (e.g., 0.95) focuses more on immediate rewards, which may make the agent inclined towards short-term gain strategies.

As shown in Table 1 and 2, considering these two factors, various values are tried. Here are two tables showing the comparison between DQN and DDQN when using different learning rate discount factor.

**Table 1.** The result of DQN

| | Learning rate | Discount factor | Average reward | Completion time(s) |
|---|---|---|---|---|
| 1 | 0.0005 | 0.99 | 195.55 | 423 |
| 2 | 0.001 | 0.99 | 209.69 | 396 |
| 3 | 0.0005 | 0.975 | 123.70 | 602 |
| 4 | 0.001 | 0.975 | 60.41 | 580 |
| 5 | 0.0005 | 0.95 | -55.19 | 640 |
| 6 | 0.001 | 0.95 | -69.33 | 648 |

**Table 2.** The result of DDQN

| | Learning rate | Discount factor | Average reward | Completion time(s) |
|---|---|---|---|---|
| 1 | 0.0005 | 0.99 | 234.88 | 450 |
| 2 | 0.001 | 0.99 | 243.36 | 410 |
| 3 | 0.0005 | 0.975 | 204.08 | 510 |
| 4 | 0.001 | 0.975 | 112.42 | 498 |
| 5 | 0.0005 | 0.95 | -46.69 | 715 |
| 6 | 0.001 | 0.95 | -63.35 | 727 |

From two tables, it could be observed that when choosing 0.001 in learning rate and 0.99 in discount factor, average reward is the biggest among them. Here is the reward comparison between DQN and DDQN, as shown in Fig. 1, 2 and 3 respectively.

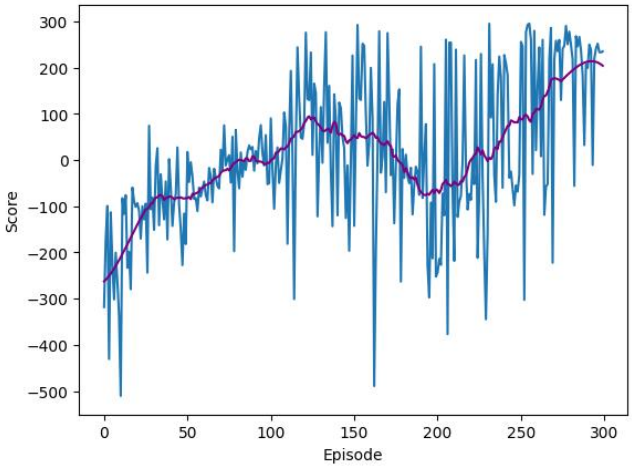


**Fig. 1.** The reward of DQN (Figure credit: Original).

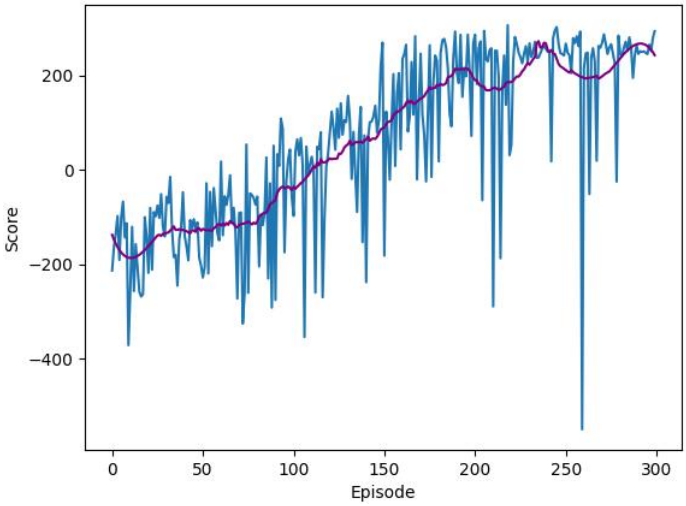


**Fig. 2.** The reward of DDQN (Figure credit: Original).

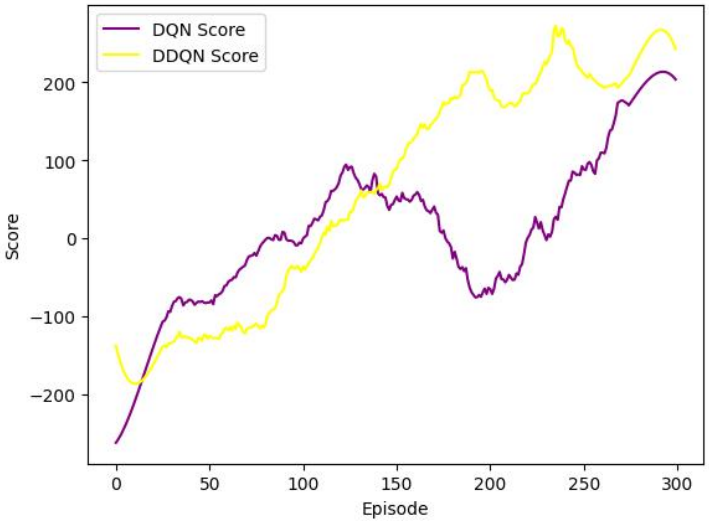


**Fig. 3.** The comparison of the reward between DQN and DDQN (Figure credit: Original).

### 3.3 Performance Comparison of Different Vales in Policy Gradient

Considering the two factors: learning rate and discount factor. This work tries to change the two values to find the best situation. Here is a table showing the comparison when using different learning rate discount factor, as shown in Table 3.

**Table 3.** The result of Policy Gradient with Different Parameters.

| Policy Gradient | Learning rate | Discount factor | Average reward | Completion time(s) |
|---|---|---|---|---|
| 1 | 0.015 | 0.99 | -74.68 | 16 |
| 2 | 0.01 | 0.99 | 3.45 | 76 |
| 3 | 0.005 | 0.99 | 76.25 | 130 |
| 4 | 0.001 | 0.99 | -66.32 | 47 |
| 5 | 0.015 | 0.97 | -88.72 | 152 |
| 6 | 0.01 | 0.97 | 89.23 | 251 |
| 7 | 0.005 | 0.97 | 59.86 | 171 |
| 8 | 0.001 | 0.97 | -75.34 | 56 |

Results in the table shows that when selecting 0.01 and 0.97 as the learning rate and discount factor respectively, average reward is the biggest, as shown in Fig. 4.

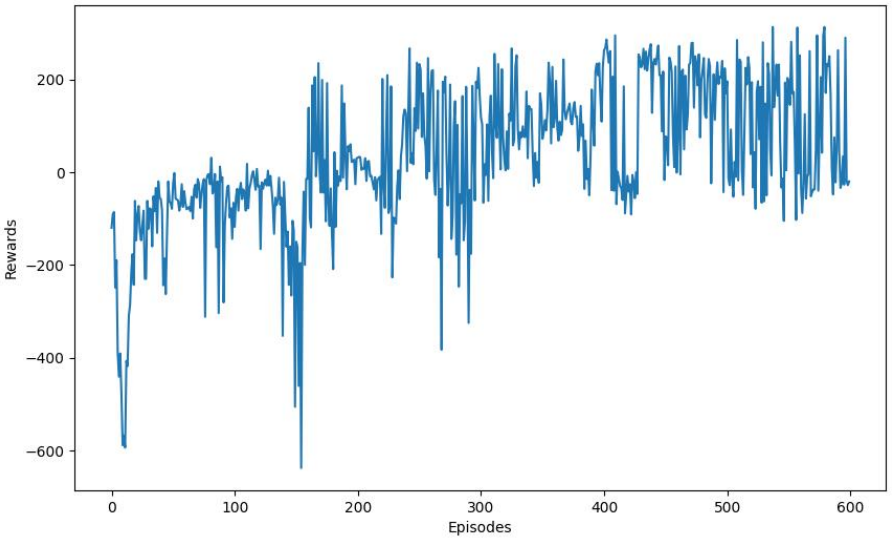


**Fig. 4.** The reward of Policy Gradient with 0.01 learning rate and 0.97 discount factor (Figure credit: Original).

Here is the reward when selecting 0.005 and 0.99 as the learning rate and discount factor respectively, as shown in Fig. 5.

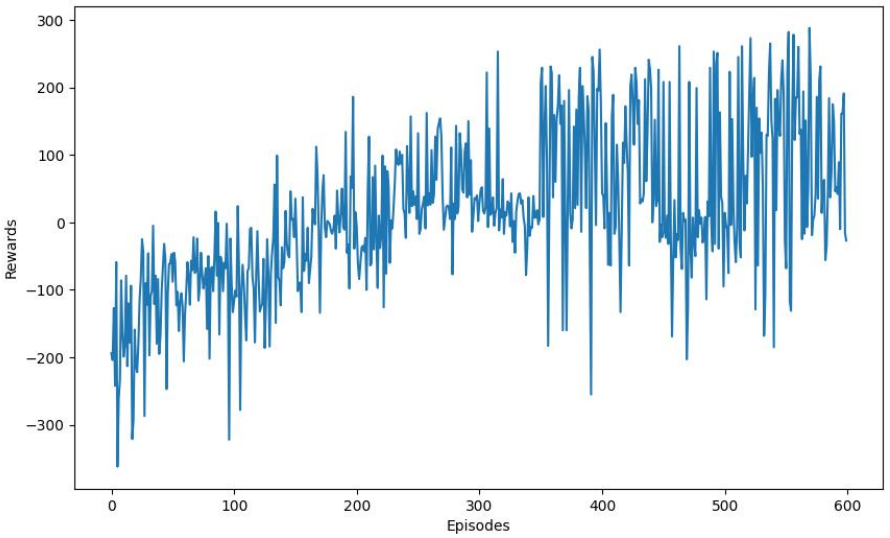


**Fig. 5.** The reward of Policy Gradient with 0.005 learning rate and 0.99 discount factor (Figure credit: Original).

When selecting 0.015 and 0.97 as the learning rate and discount factor respectively, average reward is the smallest, as shown in Fig. 6.

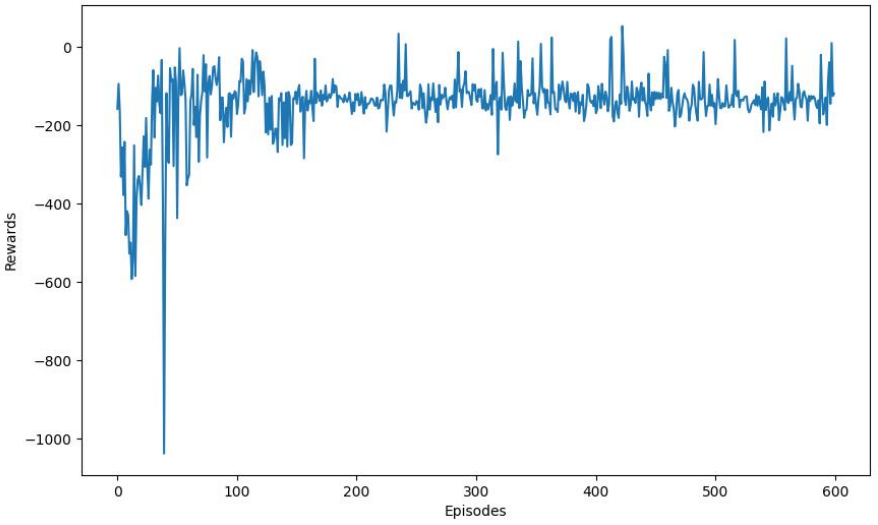


**Fig. 6.** The reward of Policy Gradient with 0.015 learning rate and 0.97 discount factor (Figure credit: Original).

## 4 Discussion

There are advantages and disadvantages of methods. For DQN, firstly, it utilizes experience replay and target networks to improve training stability and sample efficiency. Secondly, It exhibits the capacity to effectively manage high-dimensional state spaces, such as image inputs. However, it is less suitable for problems that involve continuous action spaces. as it operates and learns in discrete action spaces. And it can suffer from training instability, leading to difficulties in convergence during the training process.

As for DDQN, it is an improvement over DQN that addresses the issue of overestimation of action values, enhancing training stability. And it employs two Q-Networks, one for action selection and another for action value estimation, mitigating the instability of training targets. However, it still operates in discrete action spaces and is not applicable to problems with continuous action spaces.

As for Policy Gradient it is a direct policy learning method that can handle problems with continuous action spaces. Moreover, it learns the probability distribution of actions and can generate more continuous and smooth policies. Additionally, it directly optimizes the expected return without the need for indirect value estimation like value-based methods. However, it typically requires more samples and longer training time as it relies on sampling and Monte Carlo methods for training. And it can encounter training instability, especially in the initial stages.

The result shows that the impact of parameters on these two methods is nearly identical. When choosing 0.001 and 0.99 as two parameters, both methoods perform best. They get the highest average reward and the completion time is the shortest. The average rewards are 209.69 and 243.36 respectively, and the completion times are 396s and 410s respectively.

While keeping the learning rate constant, the discount factor progressively decreases, it could be observed that a decrease in average rewards and an increase in completion time. When keeping the discount factor constant and change the learning rate, result shows that the average rewards are generally higher when the learning rate is 0.001 compared to when it is 0.0005.

Fig. 1, 2, and 3 display the reward plots for the best-case scenario. Fig. 1 show that the rewards show an overall upward trend, but the noise is relatively high, with noticeable fluctuations. Moreover, the convergence rate is slow, as it takes approximately 270 episodes for the rewards to converge to nearly 200. Fig. 2 demonstrates that, firstly, its average reward is slightly higher than that of DQN. Secondly, the noise is relatively smaller, which makes its performance more stable compared to DQN. Lastly, its convergence rate is relatively faster, reaching around 200 rewards at approximately 180 episodes. Finally, the reward comparison plot in Fig. 3 shows that DDQN exhibits a consistently increasing trend in rewards, while DQN shows larger fluctuations. This indicates that DDQN is more stable than DQN.

The findings in Table 3 illustrate the outcomes when either the learning rate is unchanged and the discount factor is adjusted, or the learning rate remains constant while modifying the discount factor. The results indicate that the best performance is observed when the learning rate is set to 0.005 and the discount factor is set to 0.99,

as well as when the learning rate is set to 0.01 and the discount factor is set to 0.97. the average rate is relatively high, which means it has a good performance.

From Fig. 4, it could be observed that reward coverage to 200 at about 250 episodes and the noise is small, and the completion time is 251s. Fig. 5 shows that reward coverage to 200 at about 350 episodes and the noise is relatively bigger, however, the completion time is 130s, which is smaller than Fig. 4. Fig. 6 displays that in this scenario, the reward performance is the worst, as it remains consistently stable around -180 after approximately 200 episodes. This represents the poorest performance among the several scenarios examined.

## 5 Conclusion

In this study, three different methods are applied, namely DQN, DDQN, and Policy Gradient, to the Lunar Lander problem. Each method brought unique insights and performance characteristics to the table. Methodology-wise, DQN is a value-based approach that learns the value function between states and actions. DDQN, an improvement over DQN, addresses the issue of overestimation of action values by utilizing two Q-Networks. Policy Gradient, on the other hand, is a direct policy learning method that optimizes the expected return without relying on value estimation.

The results obtained from experiments indicate that in terms of average rewards, both DQN and DDQN outperformed Policy Gradient. They demonstrated higher rewards and more stable performance throughout the training process. However, it is worth noting that Policy Gradient exhibited a faster completion time, achieving twice the number of episodes in less time compared to DQN and DDQN. This indicates its superior processing speed.

In conclusion, The selection of method relies on the distinctive objectives and practical considerations of the Lunar Lander problem. DQN and DDQN are suitable for maximizing rewards and ensuring stability, making them reliable choices when high performance is desired. On the other hand, Policy Gradient offers faster processing time, which can be advantageous in time-sensitive applications. Future research could explore hybrid approaches that combine the strengths of these methods to achieve even better results in Lunar Lander or similar reinforcement learning tasks.

## References

1. Furfaro, R., Bloise, I., Orlandelli, M., Di Lizia, P., Topputo, F., & Linares, R. Deep learning for autonomous lunar landing. Advances in the Astronautical Sciences, 167, 3285-3306 (2018).
2. Gadgil, S., Xin, Y., & Xu, C. Solving the lunar lander problem under uncertainty using reinforcement learning. In 2020 SoutheastCon, 2, 1-8 2020.
3. Moerland, T. M., Broekens, J., Plaat, A., & Jonker, C. M. Model-based reinforcement learning: A survey. Foundations and Trends® in Machine Learning, 16(1), 1-118 (2023).

4. Levine, S., Kumar, A., Tucker, G., & Fu, J. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. arXiv preprint arXiv:2005.01643 (2020).
5. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. Openai gym. arXiv preprint arXiv:1606.01540 (2016).
6. Watkins, C. J., & Dayan, P. Q-learning. Machine learning, 8, 279-292 (1992).
7. Fan, J., Wang, Z., Xie, Y., & Yang, Z. A theoretical analysis of deep Q-learning. In Learning for dynamics and control, 486-489 (2020).
8. Van Hasselt, H., Guez, A., & Silver, D. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence, 30(1), 2094-2100 (2016).
9. Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. Advances in neural information processing systems, 12, 1057-103 (1999).
10. Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 8, 229-256 (1992).