Reinforcement Learning Project 2 Report
Lunar Lander

Xingyan Liu (202264690069@mail.scut.edu.cn), Shiyi Wang (202230055267@mail.scut.edu.cn)
Wolin Liang (202264690304@mail.scut.edu.cn), Haoming Jia (202264690267@mail.scut.edu.cn)
Instructor: Ou Shiqi

Abstract

This experiment focuses on the lunar landing mission in the OpenAI Gym environment. A variety of methods, including DQN, DDQN, DDPG, PPO, were used to adjust and optimize their parameters. The results show that different algorithms show different advantages and limitations in the lunar landing mission. Among them, DQN and PPO have the best stability.

I. Introduction and related work

In the Apollo program, engineers used a variety of classical control theories and algorithms to adjust the lunar module to achieve a safe landing. These methods include linear control systems[1], feedback control[2], proportional-integral-derivative (PID) control[3], state-space methods[4], Kalman filtering[5], and optimal control theory[6].However, these methods are still insufficient in adaptability and still rely heavily on manual operations in actual operations. Therefore, the problem of using reinforcement learning methods to control the landing capsule for soft landing is proposed[7].

The LunarLander environment in OpenAI Gym presents a complex control problem involving high-dimensional state spaces and requires sophisticated strategies for successful landings. Several algorithms have emerged as particularly effective in tackling this challenge.

One of the pioneering approaches is the Deep Q-Network (DQN) [8], which combines deep learning with Q-learning to approximate the Q-function using neural networks. DQN has demonstrated success in handling high-dimensional state spaces and complex decision-making tasks, including those presented by the LunarLander environment.

Variants of DQN, such as Double DQN [9] and Dueling DQN [10], have further enhanced performance by addressing overestimation biases and improving learning efficiency. Proximal Policy Optimization (PPO) is another prominent algorithm, favored for its simplicity and robustness [11]. PPO utilizes a novel approach to policy gradient methods by constraining the policy updates, which results in more stable training processes. Furthermore, Deep Deterministic Policy Gradient (DDPG), typically applied to continuous action spaces, has been successfully adapted to the continuous version of LunarLander (LunarLanderContinuous-v2) [12].

II. Problem Definition

`LunarLander-v2` is a reinforcement learning environment from OpenAI Gym designed to simulate the problem of landing a spacecraft. The agent's task is to land the spacecraft safely on the ground within a limited time and avoid collisions.

The state space and the action space is shown as follows:

TABLE I
Environment State Space and Action Space

Space Type	Physical Symbol	Details
State Space	$\vec{r}$	Spacecraft's position
	$\vec{v}$	Velocity
	θ	Angle
	$\vec{\omega}$	Angular velocity
	C (Contact status)	Whether it is in contact with the ground
Action Space	F_{left}	Left rocket
	F_{right}	Right rocket
	F_{up}	Up rocket
	F_{none}	No operation

III. Method Selection and Modeling

In this problem, we selected four reinforcement learning methods to model and solve the task, each leveraging its specific characteristics to handle the challenges posed by the `LunarLander-v2` environment.

A. DQN (Deep Q-Network)

DQN is a reinforcement learning method based on Q-learning that uses deep neural networks to approximate the Q-value function, thereby learning an optimal action-value function for each state. The core idea of DQN is to update the Q-function by minimizing the difference between the estimated Q-values and the actual return.

Reason for Selection:

- The action space in the `LunarLander-v2` environment is discrete, making DQN a good fit for this type of problem. By using deep neural networks to approximate the Q-value function, it can effectively learn an action strategy in complex environments.
- DQN can handle high-dimensional state spaces and improve stability through experience replay and target networks, making it well-suited for environments like `LunarLander-v2`, which contain significant noise and nonlinear relationships.

The Q-function for DQN is defined as:

$$Q(s,a) = \mathbb{E}\left[R_t + \gamma \cdot \max_{a'} Q(s',a')\right]$$

where s is the current state, a is the action taken, R_t is the immediate reward, and γ is the discount factor.

B. DDQN (Double Deep Q-Network)

DDQN is an improvement over DQN that aims to reduce the overestimation bias found in Q-learning. DDQN solves this by introducing two separate Q-networks: one for action selection and another for calculating the target Q-values.

Reason for Selection:

- In DQN, maximizing the Q - value may cause overestimation during action selection, leading to suboptimal actions. DDQN solves this by using two independent networks for more accurate state-action value estimation and reduced instability from overestimation.
- Given the relatively complex state and action spaces in the `LunarLander-v2` environment, DDQN provides a more robust optimization approach, mitigating potential biases in the estimation process.

The Double Q-learning objective in DDQN is:

$$y = r + \gamma Q_{\text{target}}(s', \arg\max_a Q_{\text{policy}}(s', a))$$

where Q_{policy} is the current policy network and Q_{target} is the target network.

C. DDPG (Deep Deterministic Policy Gradient)

DDPG is an actor-critic-based reinforcement learning algorithm designed for continuous action spaces. It uses stochastic gradient ascent to maximize the policy and stochastic gradient descent to optimize the value function.

Reason for Selection:

- Although the action space in `LunarLander-v2` is discrete, DDPG performs exceptionally well on high-dimensional, complex control tasks
- DDPG benefits from combining both actor and critic networks, which helps stabilize the learning process and avoid the limitations of value iteration methods.

The DDPG update rules are as follows:

- **Actor**: Updates the policy network parameters using policy gradients.
- **Critic**: Updates the value network parameters by minimizing the mean squared error.

D. PPO (Proximal Policy Optimization)

PPO is a policy gradient method designed to optimize the policy function while maintaining stability by limiting the range of policy updates. This method is particularly suitable for high-dimensional continuous spaces.

Reason for Selection:

- The main advantage of PPO is its ability to train the policy in a stable manner when dealing with complex continuous action spaces. It ensures that the policy update step does not lead to excessive policy changes, preventing instability.
- PPO has been widely used in reinforcement learning tasks, especially in complex control problems like `LunarLander-v2`, where it balances exploration and exploitation effectively.

The objective function for PPO is:

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min\left(\rho_t(\theta)\hat{A}_t, \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

where $\rho_t(\theta)$ is the ratio of the current policy to the old policy, $\hat{A}_t$ is the advantage function, and ϵ is a small hyperparameter controlling the range of policy updates.

IV. Experiment and Results

The DQN algorithm is based on off-policy Q-learning and combines function approximation with experience replay and target network stabilization techniques. Specifically, experience replay helps to decorrelate observations, while the use of a target network mitigates the risk of feedback loops during training, thus improving stability.

Similarly, DDPG is also an off-policy approximation model. It is based on the coordinated operation of the actor and the critic. While estimating the value of actions, it maximizes the value of actions. Meanwhile, it uses the buffer and the soft update method to stabilize the training process.

TABLE II
Hyperparameter Values for PPO Training

Hyperparameter	Value
Learning Rate (α)	0.0003
Discount Factor (γ)	0.99
Clip Range (ϵ)	0.2
Entropy Coefficient (β)	0.01
GAE Lambda (λ)	0.95
Batch Size	64
Number of Epochs (K)	10
Replay Buffer Size	100,000

A. Network Architecture and Hyperparameters

The neural network used for Q-value approximation has two hidden layers, each with 64 units, and ReLU activation functions. The hyperparameters are selected based on empirical results and prior studies on DQN performance in similar environments. The chosen values are as follows:

TABLE III
Hyperparameter Values for DQN Training

Hyperparameter	Value
Learning Rate (α)	0.001
Discount Factor (γ)	0.99
Exploration Rate (ϵ)	1.0 (initial)
Decay Rate	0.995
Replay Memory Size (N)	100000
Target Update Frequency (C)	1000 steps
Batch Size	64

Algorithm 1 DQN Algorithm for Lunar Lander

Require: Environment $\mathcal{E}$, learning rate α, discount factor γ, exploration rate ϵ, decay rate, target update frequency C

Ensure: Trained Q-network with optimal action-value function

Initialize memory D with N, primary Q-network Q with weights θ, and target Q-network $\hat{Q}$ with weights $\theta^- \leftarrow \theta$

for each episode **do**

Reset environment, get initial state s_0

while not done **do**

Select action a_t:

$$a_t = \begin{cases} \text{random action}, & \text{with probability } \epsilon \\ \arg\max_a Q(s_t, a; \theta), & \text{with probability } 1 - \epsilon \end{cases}$$

Execute a_t, observe reward r_t and next state s_{t+1}

Store transition (s_t, a_t, r_t, s_{t+1}) in D

if D has enough samples **then**

Sample mini-batch (s_j, a_j, r_j, s_{j+1}) from D

Compute target y_j:

$$y_j = \begin{cases} r_j, & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-), & \text{otherwise} \end{cases}$$

Update θ by minimizing $(y_j - Q(s_j, a_j; \theta))^2$

end if

if step mod $C = 0$ **then**

Update target network: $\theta^- \leftarrow \theta$

end if

end while

Decay ϵ according to decay rate

end for

Algorithm 2 DDPG Algorithm

Require: Environment $\mathcal{E}$, learning rates α_μ and α_Q, discount factor γ, buffer capacity N, soft update parameter τ, batch size M

Ensure: Trained actor and critic networks

Initialize replay buffer D and networks μ, μ', Q, and Q' with weights θ^μ, $\theta^{\mu'} \leftarrow \theta^\mu$, θ^Q, $\theta^{Q'} \leftarrow \theta^Q$

for each episode **do**

Reset environment, get initial state s_0

while not done **do**

Generate $a_t = \mu(s_t; \theta^\mu)$, execute a_t, observe r_t, s_{t+1}

Store (s_t, a_t, r_t, s_{t+1}) in D

if D has enough samples **then**

Sample mini-batch from D

Target: $y_j = r_j + \gamma Q'(s_{j+1}, \mu'(s_{j+1}; \theta^{\mu'}))$

Loss: $L(\theta^Q) = \frac{1}{M} \sum_j (y_j - Q(s_j, a_j))^2$

Update actor using policy gradient $\nabla_{\theta^\mu} J$

end if

Soft update target networks $\theta^{\mu'}$ and $\theta^{Q'}$

end while

end for

Algorithm 3 PPO Algorithm for LunarLander-v2

Require: Environment $\mathcal{E}$, learning rate α, discount factor γ, clipping parameter ϵ, entropy coefficient β, epochs K, batch size M

Ensure: Trained Actor and Critic networks

Initialize Actor and Critic networks with θ and ϕ

for each episode **do**

Reset environment, obtain initial state s_0

while not done **do**

Predict action distribution $\pi_\theta(s_t)$ with Actor

Sample action a_t, observe reward r_t, next state s_{t+1}

Store (s_t, a_t, r_t, s_{t+1}) in buffer

Update $s_t \leftarrow s_{t+1}$

end while

Compute advantages and discounted rewards

for K epochs **do**

Sample mini-batch from buffer

Update Actor using PPO clipped loss

Update Critic with value loss

end for

end for

function PPO_LOSS(y_{true}, y_{pred})

Compute clipped objective and entropy loss

Return total loss

end function

TABLE IV
HYPERPARAMETER VALUES FOR DDPG TRAINING

Hyperparameter	Value
Number of Episodes	1000
Buffer Capacity (N)	100000
Batch Size (M)	64
Discount Factor (γ)	0.99
Soft Update Parameter (τ)	0.001
Learning Rate of Actor Network (α_μ)	0.0001
Learning Rate of Critic Network (α_Q)	0.001

B. Results

The training results show clear progress in the agent's performance. Figure 1, early scores are highly variable and often negative, reflecting the agent's initial struggles. Over time, scores trend upward, frequently exceeding 200, indicating improved skill. Figure 2 confirms this, showing a steady rise in average scores, reflecting the agent's growing consistency. By the end, the agent reaches near-target scores, proving that the DQN, DDQN, DDPG, and PPO algorithm and fine-tuning enabled it to learn an effective landing strategy.

Additional experiments were performed to assess the impact of different hyperparameters, including epsilon decay, discount factors, and learning rates. The outcomes are presented in Figures 3, 4, and 5.

V. RESULT ANALYSIS

In this section, we analyze the experimental results in depth, with a focus on the effects of various hyperparameters such as epsilon decay, discount factor γ, and learning rate α.

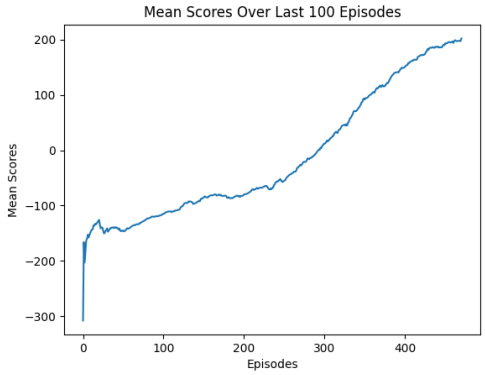

Fig. 1. Mean Scores over the Last 100 Episodes

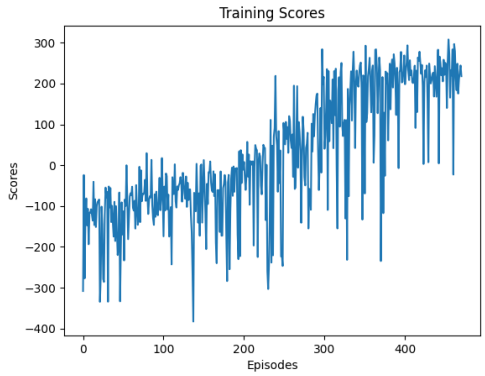

Fig. 2. Training Scores over Episodes

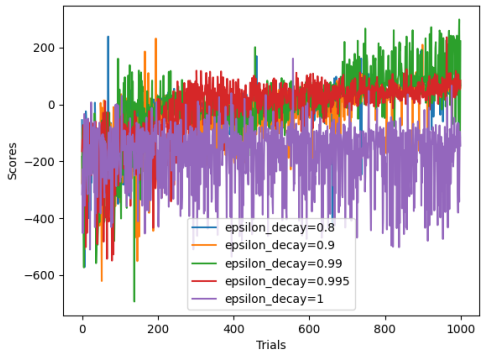

Fig. 3. Epsilon Decay Over Time for Different Decay Rates

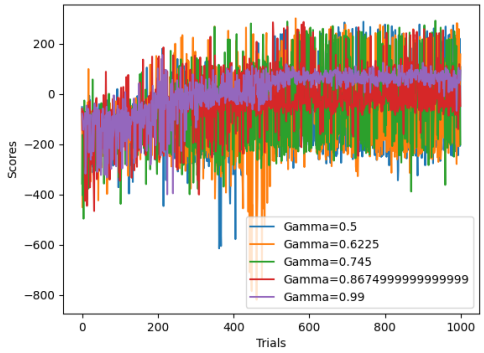

Fig. 4. Scores with Different Discount Factors

A. Epsilon Decay Over Episodes

The analysis in Figure 3 shows the discount factor (γ) strongly affects the agent's stability and performance. Higher values (e.g., $\gamma = 0.99$) lead to more consistent and higher scores as they make the agent prioritize long-term rewards, crucial in complex tasks. Lower values (e.g., $\gamma = 0.5$) cause greater score variability and lower performance, reflecting a short-sighted strategy. Mid-range values improve but lack the stability of the highest discount factor. Thus, a high discount factor is effective for a stable, high-performing policy.

B. Impact of Discount Factor γ

The discount factor γ determines the value of future rewards relative to immediate ones. A high γ (e.g., 0.99) makes the agent prioritize long - term rewards, crucial in the Lunar Lander environment with optimal strategies involving controlled maneuvers. Figure 4 shows epsilon decay rate affects performance. A slower decay rate (0.995) promotes balanced exploration, leading to higher and more stable scores during training, helping the agent develop a good policy. Faster decay rates (0.8 and 0.9) make the agent exploit too soon, resulting in lower and inconsistent scores. So, a decay rate of 0.995 is optimal for stable learning.

C. Impact of Learning Rate α

The learning rate α impacts the speed and stability of network weight updates during training. As Figure 5 shows, an optimal rate of 0.0005 enables efficient convergence and stability. Lower rates (e.g., 5×10^{-5}) slow convergence as Q-network updates are too small for effective environment dynamics capture. High rates (e.g., 0.05 or 0.5) cause instability, with the agent diverging from optimal policies and having large performance fluctuations, consistent with standard DQN practice where moderate rates are key for balanced convergence without sacrificing stability.

Figure 6 shows how different epsilon decay rates affect the transition from exploration to exploitation. Faster decay rates, like 0.8, reduce epsilon quickly, pushing the agent into exploitation early. Moderate rates (0.9 and 0.99) offer a more balanced shift. The slowest rate, 0.995, allows prolonged exploration, helping the agent learn a more stable policy. A rate of 1.0, with no decay, keeps the agent in constant exploration, hindering effective learning. Thus, a decay rate of 0.99 best supports a gradual and beneficial transition.

D. Impact of Model Selection in DDPG Training

In this DDPG experiment, the team designed 3 actor networks (pure MLP (Linear), MLP with LayerNorm (Lin-

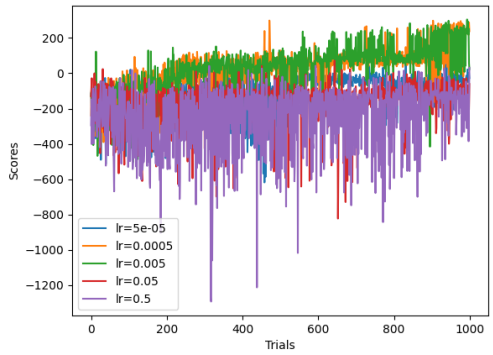

Fig. 5. Score Variability for Different Learning Rates

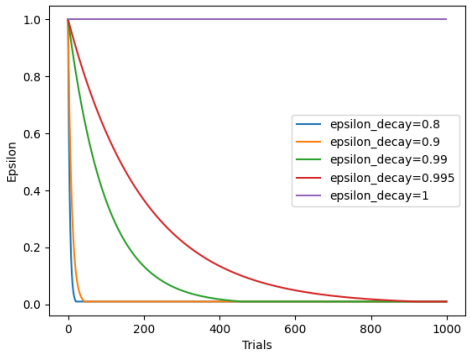

Fig. 6. Epsilon Per Training Trial Given Different Epsilon Decay Rates

earNorm), deep network with residual blocks (Residue)) and 4 critic networks (Linear1, Linear2, Residue1, Residue2 based on input combination and complexity) for the task. There are 12 networks in total. After 12 independent experiments, the results are as Fig 7.

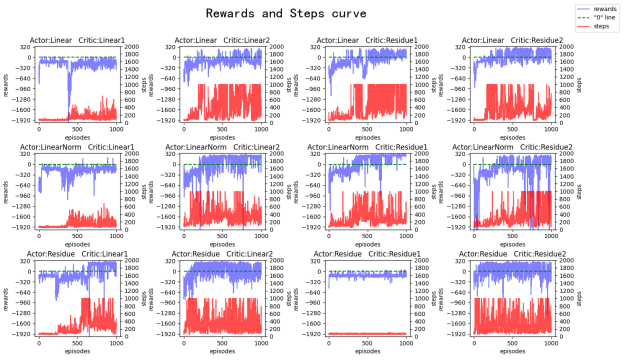

Fig. 7. Rewards' and Steps' curves in DDPG Training

E. Experiment Summary for PPO on LunarLander-v2

The experiment aimed to train a Proximal Policy Optimization (PPO) agent on the `LunarLander-v2` environment. The graph below shows the agent's performance over the training cycle, measured by the episode score.

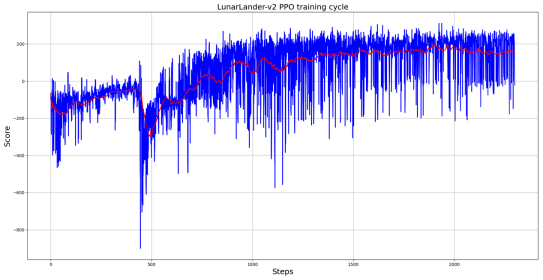

Fig. 8. Training Performance of PPO on LunarLander-v2

Observations

During training, the agent's score showed significant fluctuations. In the initial phase, the score often dropped into negative values, reflecting difficulty in landing. As training progressed, the score improved, indicating better control and fewer crashes. In the later phase, the score stabilized around positive values, showing that the agent had learned a stable landing strategy, with only minor fluctuations remaining.

Conclusion

The PPO agent successfully learned to control the lander over time, as indicated by the increasing and stabilizing average score. Despite the variability in individual episode scores, the overall trend shows effective learning and improved performance in the `LunarLander-v2` environment.

REFERENCES

[1] Katsuhiko Ogata. *Modern Control Engineering*. Prentice Hall, 2010.

[2] Gene F Franklin, J David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*. Pearson, 2015.

[3] Karl J Åström and Richard M Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2010.

[4] Thomas Kailath. *Linear Systems*. Prentice Hall, 1980.

[5] Rudolf Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[6] Donald E Kirk. *Optimal Control Theory: An Introduction*. Dover Publications, 2004.

[7] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[9] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[10] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003. PMLR, 2016.

[11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[12] 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. *arXiv preprint arXiv:1509.02971*, 2015.