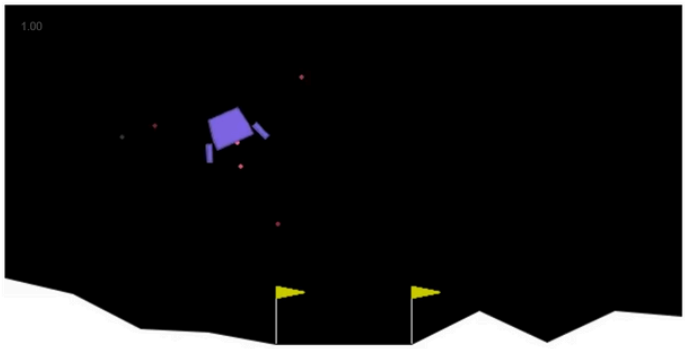
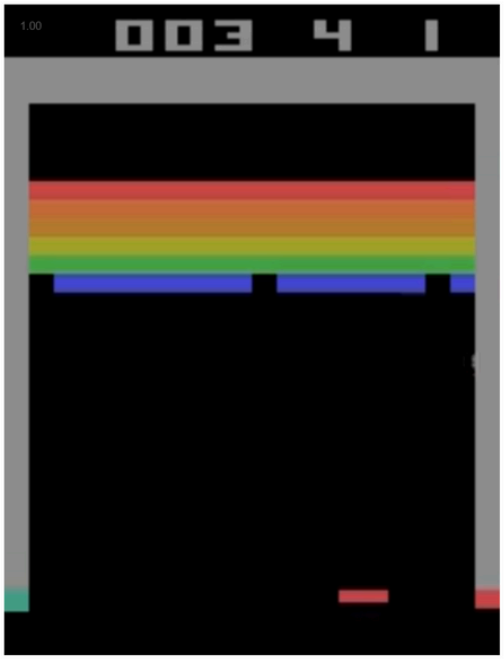
AI Capstone HW2 Report:
Gaming AI with Reinforcement Learning

Author: 司徒立中 (111550159) | GitHub Link





1. Introduction

The goal of this assignment is to explore reinforcement learning. The environments used are from OpenAI Gym, with the maintained fork being Gymnasium. This project involves two tasks/games: **Lunar Lander** from Box2D and **Breakout** from Atari.

For the implementation of RL algorithms, I applied both **value-based** methods and **policy-based** (including Actor-Critic) methods. In the value-based category, I experimented with DQN, Double DQN, Dueling DQN, and others. For policy and Actor-Critic approaches, I implemented A2C, PPO, and TRPO.

2. Environments

In the **LunarLander-v3** environment from the Box2D suite, the agent's task is to control a lunar module so that it touches down upright on a landing pad at the origin. The agent selects from four discrete actions: do nothing, fire the left orientation engine, fire the main engine, or fire the right orientation engine, and observes an 8-dimensional state vector consisting of its x and y positions, x and y velocities, pitch angle, angular velocity, and two boolean flags indicating leg contact with the ground.

The reward at each step is shaped to encourage precise, fuel-efficient landings: it includes a continuous bonus that grows as the lander approaches the pad and slows down, a penalty proportional to its tilt angle, and a frame-wise cost for each engine firing (–0.03 per frame for side engines and –0.3 per frame for the main engine). Each leg in contact with the ground grants +10 points, while a successful touchdown yields +100 and a crash incurs –100. An episode is considered solved once the agent achieves an average score of 200 over 100 consecutive runs.

Additionally, in the **ALE/Breakout-v5** environment from the Gymnasium Atari suite, the agent's task is to control a horizontally-moving paddle at the bottom of the screen to bounce a ball into and destroy a wall of bricks; it operates with a Discrete(4) action space (NOOP, FIRE, RIGHT, LEFT) and receives observations as raw RGB frames of size $210 \times 160 \times 3$ by default.

Rewards are issued for each brick destroyed, typically +1 point per brick, with minor variations based on brick color—and the episode continues until the agent clears the wall or exhausts its five lives, making the total brick-breaking score the primary learning signal.

3. Methods

In our experiments we implemented both value-based and policy/actor-critic methods. The value-based suite builds on Q-learning with deep function approximators, while the policy and actor-critic family directly optimizes stochastic policies with gradient-based updates. We describe each class of algorithms below, along with key architectural and training choices.

Value-Based Methods

We begin with the Deep Q-Network (DQN) lineage, where agents learn action-value functions via temporal-difference updates and deep neural nets.

First of all, **Deep Q-Network (**DQN) uses a convolutional (for image inputs) or fully-connected (for low-dimensional states) network Q(s, a; θ) to approximate the action-value function. During training it employs an experience replay buffer and a separate, periodically updated target network to stabilize learning. Each update minimizes the mean-squared Bellman error where θ− are the target network parameters:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')}\left[r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right]^2$$

To counteract Q-value overestimation, **Double DQN** decouples action selection from evaluation by using the online network to pick the maximizing action and the target network to evaluate it. The update becomes:

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

In the other branch of DQN, **Dueling DQN** factors the Q-network into separate streams for the state-value V(s) and an advantage function A(s,a), recombining them as like this. It helps the agent learn which states are (or are not) valuable, independent of the action choice.

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

Next, in standard experience replay, the agent randomly samples past experiences to learn from. However, not all experiences are equally informative. **Prioritized Experience Replay (**PER) addresses this by assigning a priority to each experience based on how surprising or significant it is, typically measured by the agent's prediction error at that time. Experiences with higher errors are sampled more frequently, allowing the agent to focus on learning from its mistakes. This targeted approach can lead to faster and more efficient learning.

$$P(i) = \frac{|\delta_i|^{\alpha}}{\sum_k |\delta_k|^{\alpha}}$$

$$w_i = \left(\frac{1}{N \cdot P(i)}\right)^{\beta}$$

Traditional DQN estimates the average expected return for each action, which might not capture the full range of possible outcomes. **Quantile Regression DQN** (QR-DQN) enhances this by predicting multiple quantiles of the return distribution for each action. By understanding the distribution of potential returns, not just the average, the agent gains a more comprehensive view of the risks and uncertainties associated with each action. This richer information can lead to more robust and informed decision-making.

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} \rho_{\tau_i}^{\kappa}\left(\theta_j^{\text{target}} - \theta_i(s,a)\right)$$

Policy & Actor–Critic Methods

Policy-based methods directly parameterize a stochastic policy π(a | s ;θ) and adjust θ via gradient ascent on expected return. Actor-Critic methods augment this with a learned value function baseline.

To begin with, **Advantage Actor-Critic (A2C)** is the synchronous, batched variant of the classic A3C algorithm. Multiple parallel environments generate trajectories; the critic estimates the value V(s; ϕ), and the actor is updated via the advantage using the policy gradient, while the critic minimizes the squared error on V.

$$\hat{A}_t = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V(s_{t+n};\phi) - V(s_t;\phi)$$

$$\nabla_\theta J = \mathbb{E}[\nabla_\theta \log \pi(a_t|s_t;\theta)\, \hat{A}_t]$$

On top of that, **Proximal Policy Optimization (PPO)** maximizes a clipped surrogate objective. This clipping prevents overly large policy updates without requiring complex second-order solvers.

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

$$r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t)$$

Last but not least, **Trust Region Policy Optimization (TRPO)** enforces a hard constraint on the KL divergence between old and new policies, solving the formula below. A conjugate-gradient procedure and a line-search are used to approximate the constrained update, yielding stable monotonic policy improvement.

$$\max_\theta\ \mathbb{E}\big[r_t(\theta)\,\hat{A}_t\big] \quad \text{s.t.} \quad \mathbb{E}\big[\text{KL}[\pi_{\theta_{\text{old}}} \,\|\, \pi_\theta]\big] \le \delta.$$

Automated Action and Reward Shaping

In the Atari Breakout environment, it requires selecting action 0 to resume after losing a life. However, during training, the agent might not learn to do this promptly, or it may hesitate while waiting for the "right" moment, which unnecessarily prolongs the training time. To address this, I automated this action to ensure smoother progression. Additionally, I applied a simple form of reward shaping by subtracting 1 from the reward each time the ball is lost. According to other studies, this penalty tends to improve performance slightly by encouraging the agent to keep the ball in play longer.

Implementation Details (Hyper-parameters)

【Lunar Lander -v3】

Episode 1000, Batch Size 64, Buffer Size 1e5, Gamma 0.99, Learning Rate 1e-3, Epsilon 1.0~0.1, Epsilon Decay 1e-5, Random Seed 111550159, Device 'cuda'.

【ALE/Breakout -v5】

Valued-Based: Episode 1e5, Batch Size 32, Buffer Size 1e5, Gamma 0.99, Learning Rate 1e-4, Epsilon 1.0~0.1, Epsilon Decay 1e-6, Random Seed 42, Device 'cuda'.

Policy & Actor–Critic Methods: Total Timesteps 5e7, Num Env 8, Gamma 0.99, Learning Rate 1e-4, Epsilon 1.0~0.1, Epsilon Decay 1e-6, Random Seed 42, Device 'cuda'.

4. Experiments

LunarLander-v3

	Highest	Last Avg100
DQN	324.65	232.74
Double	309.58	234.50
Dueling	318.92	240.84
PER	56.09	-217.73
QR-DQN	179.78	-366.35
Shaping	-	-
Nstep	314.77	262.86
Noisy	314.40	248.78

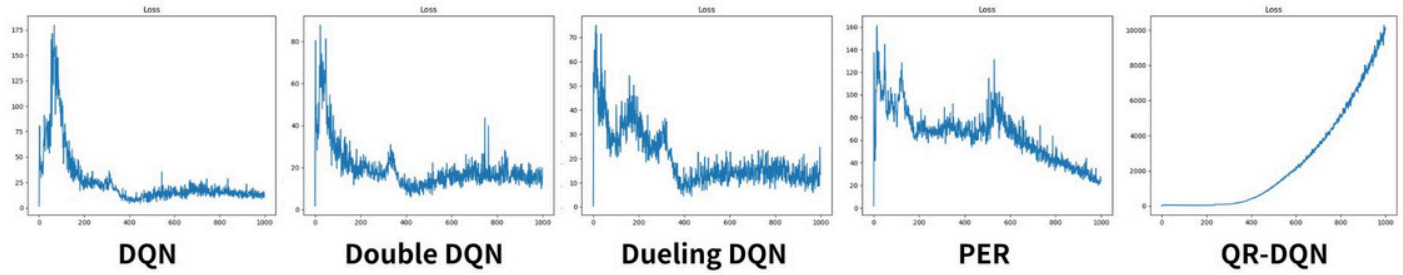
ALE/Breakout-v5

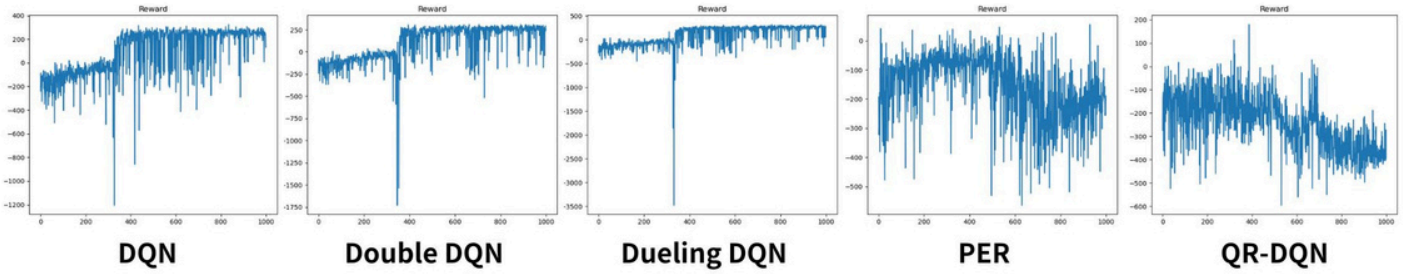
	Highest	Best Avg100
DQN	32.00	10.94
Double	39.00	14.53
Dueling	29.00	10.08
PER	35.00	11.51
QR-DQN	65.00	20.96
A2C	352.00	42.15
PPO	37.00	10.64
TRPO	27.00	9.28

In this section, we evaluate the performance of various RL methods and other strategies. The table below reports, for each algorithm and each task, both the highest score achieved and the average score over the last 100 episodes.

On the LunarLander-v3 environment, vanilla DQN attains the single highest return (324.65), yet its 100-episode average (232.74) lags behind several of its variants. It indicates that, although DQN can occasionally discover very good policies, its learning trajectory is less stable than methods like Dueling DQN, N-step DQN, or Noisy DQN. Those extensions consistently push mean performance above 240, with N-step DQN even surpassing 260 on average, thanks to more reliable multi-step returns and improved exploration.

Turning to ALE/Breakout-v5, we observe a different hierarchy: synchronous policy gradients (A2C) dominate, reaching a best score of 352.00 and an average of 42.15, while distributional value methods (QR-DQN) lead among DQN variants. PPO and TRPO, despite their theoretical advantages, yield only limited gains over standard DQN in this discrete-action, high-variance setting.



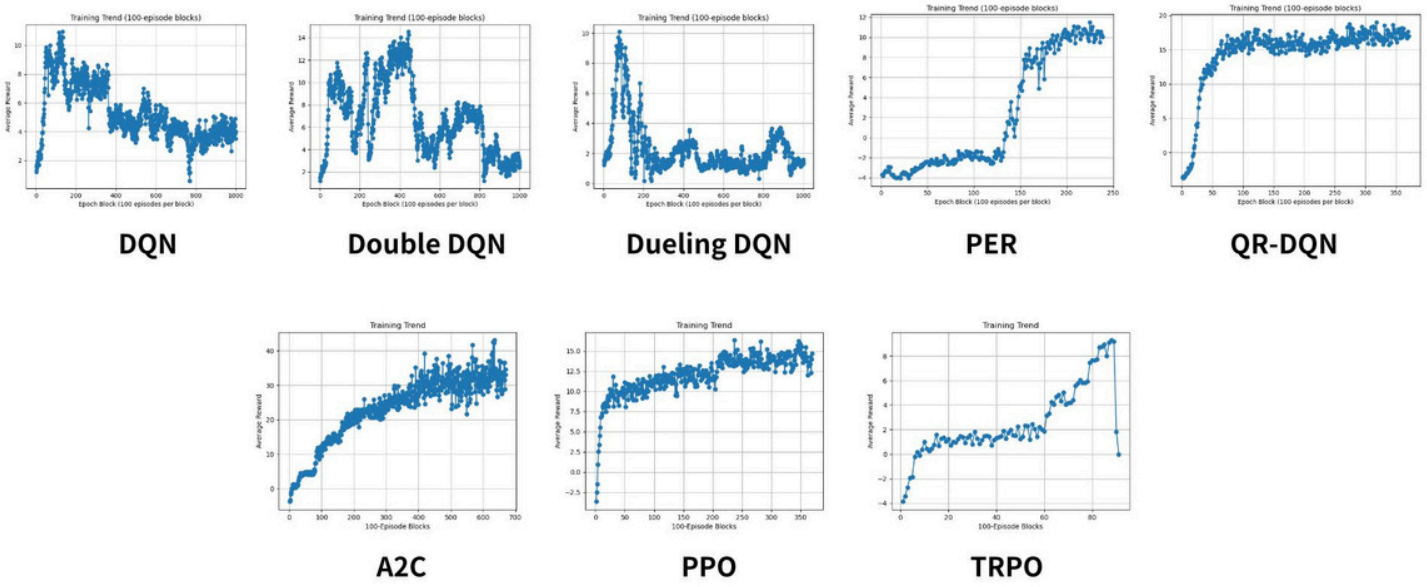


Those training curves are run-throughs of how each method's loss and reward curves evolve in the LunarLander-v3 environment.

Across the three classic value-based agents (DQN, Double DQN and Dueling DQN), the loss curves all drop quickly and the rewards rise steadily toward the solved region, with only minor differences. Dueling DQN shows a few small loss upticks deeper into training but otherwise tracks very closely with its siblings in both convergence speed and final reward.

By contrast, Prioritized Experience Replay (PER) stands out for its persistent oscillations: after an initial decline, the loss flattens into a noisy plateau and the reward trace swings widely up and down even late in training. This instability points to PER's over, emphasis on high, error transitions.

Most dramatically, QR-DQN behaves entirely differently. Its quantile, regression loss grows without bound, and its reward curve remains flat and noisy, never catching up to the other methods. This exploding loss signals that, without proper clipping or normalization, distributional updates in a high-variance continuous task can undermine learning altogether.



In the value-based group (DQN, Double DQN and Dueling DQN), all three agents inch their average Breakout reward into the single digits after 1K*100 episodes, but none break much past 8–10 points. Double DQN and Dueling DQN exhibit a bit more jitter as they chase down overestimated Q-values, yet their final plateaus still sit well below what you'd want for high-score play.

By contrast, QR-DQN shoots straight up into the mid-teens almost as soon as it can, settling around 16–18 points, a clear sign that modeling the full return distribution helps capture those rare high-reward sequences that standard DQN misses. PER likewise lags early on but then jumps into the low tens once its prioritized sampling home in on especially informative frames; its delayed spike suggests it simply needed enough "surprising" transitions to accumulate before exploitation really kicks in.

On the policy-gradient side, A2C towers over everything else, steadily climbing into the 30s and even nudging 40 points by the end, thanks to its on-policy, synchronous updates that suit the highly stochastic Breakout dynamics. PPO warms up quickly into the low teens, better than vanilla DQN but nowhere near A2C, before flatlining, a behavior that reflects its conservative clipping mechanism preventing overly bold policy swings.

TRPO, meanwhile, shows a respectable but slow rise into single digits and then abruptly collapses at the very end, hinting that its strict KL-constraint can sometimes hamper sustained improvement in environments where rapid adaptation is key.

	Speed up approach	Training Time	Performance (Best Avg)
Value-based (DQN)	Automated Actions	57h → 40h (1e5 episodes)	12.48 → 10.94
Actor–Critic (A2C)	Num env (=8)	119h → 21h (5e7 episodes)	X→ 42.15

This table compares two reinforcement learning approaches, value-based (DQN) and actor-critic (A2C), in terms of training speed-up strategies, training time, and performance.

For DQN, automated actions were used as a method to speed up training. This reduced training time from 57 hours to 40 hours over 100,000 episodes. However, this came at a cost to performance, where the best average score dropped from 12.48 to 10.94.

On the other hand, A2C used multiple environments (set to 8) to accelerate training. This significantly reduced training time from 119 hours to 21 hours over 50 million episodes. Interestingly, performance improved, reaching a best average score of 42.15. The original score (denoted as "X") is not provided, so the improvement can't be compared numerically, but the final performance is notably strong.

5. Discussion

Based on the experimental results, we observed a few key points. **Advanced RL algorithms do not always achieve the highest scores**, but they tend to produce more stable results. This was evident when PPO, which is typically stronger than A2C, did not perform as well—possibly due to insufficient hyperparameter tuning, which was too time-consuming to fine-tune properly during these tests.

We also found that environments with **sparse rewards and image-based observations**, like Atari games, are particularly challenging. In these cases, QR-DQN and A2C performed better, likely due to their better handling of complex input and reward sparsity.

Lastly, while some **automated settings** can speed up training, they often result in lower final performance, showing a clear **trade-off** between efficiency and optimal outcomes.

That said, some questions remain, particularly why certain **advanced methods underperform without extensive tuning**. This highlights the importance of better tuning strategies and understanding how different algorithms interact with specific environments.

6. Reference

[1] Gymnasium

[2] Box2D - Lunar Lander

[3] Atari - Breakout

[4] YouTube - Reinforcement Learning by Hung-Yi Lee

[5] Medium - OpenAI Gym

[6] GitHub - Atari DRL

[7] GitHub - DQN Atari Breakout

[8] GitHub - Atari Breakout Reinforcement Learning

7. Video Clip Links

[1] Box2D - Lunar Lander (DQN)

[2] Atari - Breakout (A2C)