AI Capstone HW2 report

111550151 徐嘉亨

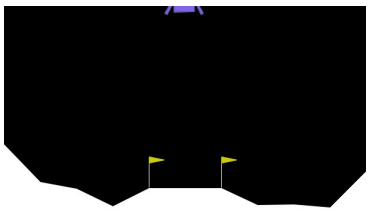
April 21, 2025

Abstract

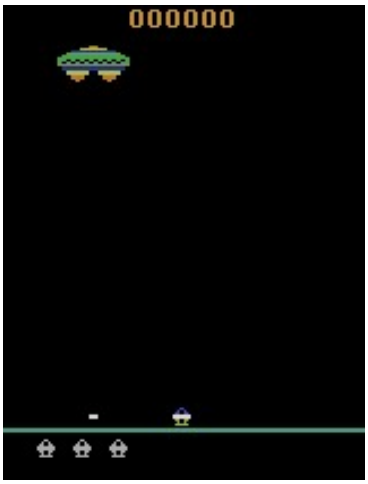
In this project, I explore two prominent reinforcement learning algorithms REINFORCE and Deep Q-Network (DQN)—for solving tasks in discrete action environments. Specifically, I apply REINFORCE, a Monte Carlo policy gradient method, to the Lunar Lander environment, and DQN, a value-based method, to the Atari Assault environment. While REINFORCE provides an unbiased gradient estimator, it suffers from high variance, which can hinder stable learning. To mitigate this, I progressively incorporate several variance-reduction and regularization strategies, including reward normalization, baseline value functions, generalized advantage estimation (GAE), and entropy regularization. For DQN, I adopt a parallel experimental design on Assault: beginning with a vanilla DQN agent and incrementally integrating widely used enhancements such as reward clipping, double Q-learning, dueling networks, and prioritized experience replay. In both settings, I perform controlled ablation studies, evaluating each technique's individual and combined effect on training performance and stability. All experiments follow a consistent configuration strategy to ensure fair and meaningful comparisons.

1 Environments & Tasks

- **Environment: Box2D / Task: Lunar Lander (Discrete version)**
  This environment is a classic rocket trajectory optimization problem. According to Pontryagin's maximum principle, it is optimal to fire the engine at full throttle or turn it off. This is the reason why this environment has discrete actions: engine on or off. For more detailed information, you can refer to Link.



- **Environment: Atari / Task: Assault**
  You control a vehicle that can move sideways. A big mother ship circles overhead and continually deploys smaller drones. You must destroy these enemies and dodge their attacks. For more detailed information, you can refer to Link.



2 RL Algorithms

In this project, I explore two fundamental reinforcement learning algorithms: **REINFORCE** and **Deep Q-Network (DQN)**. These algorithms represent two paradigms of reinforcement learning: policy-based and value-based, respectively. I apply REINFORCE to the **Lunar Lander** task and DQN to the **Atari Assault** task, both of which are discrete action environments from the Gymnasium benchmark suite.

REINFORCE Overview

The REINFORCE algorithm is a Monte Carlo policy gradient method that directly optimizes a stochastic policy $\pi_\theta(a|s)$ parameterized by θ. It belongs to the class of on-policy methods and is especially suitable for model-free environments.

The goal is to maximize the expected return $J(\theta)$ by updating the policy parameters in the direction of the gradient:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t$$

where:

- G_t is the total return from timestep t to the end of the episode.
- α is the learning rate.
- The log-probability term encourages actions with high returns.

Being a Monte Carlo method, REINFORCE requires full episode trajectories to compute the return and gradient, and does not involve bootstrapping.

DQN Overview

The Deep Q-Network (DQN) algorithm is a value-based, off-policy method that approximates the Q-function $Q(s, a)$ using a deep neural network. The update rule is based on minimizing the temporal difference (TD) error:

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

Key components of DQN include:

- A **target network** to stabilize learning by decoupling the target computation.
- An **experience replay buffer** to improve sample efficiency and break correlations.

The target network is updated periodically to track the evaluation network, and batches of transitions are sampled from the replay buffer to update the evaluation network.

Implementation Details

I implement both REINFORCE and DQN using PyTorch. For REINFORCE:

- The policy is parameterized by a fully-connected neural network that outputs action probabilities via a softmax layer.
- The agent collects full episode trajectories and computes the return G_t for gradient updates.
- I use the Adam optimizer with a constant learning rate.

For DQN:

- The Q-network is a convolutional neural network (CNN) that takes stacked Atari frames as input and outputs Q-values for each action.
- I use `AtariPreprocessing` and `FrameStack` wrappers to convert the visual input into a compact, stacked grayscale format.
- The target network is updated every fixed number of steps, and experience replay is used for learning.

Each algorithm is applied to one task:

- **Lunar Lander**: Trained using REINFORCE with a policy network.
- **Assault**: Trained using DQN with a value-based Q-network.

Both implementations serve as a baseline for subsequent experiments that incorporate additional techniques such as reward normalization, value baselines, advantage estimation, entropy regularization (for REINFORCE), and reward clipping, double Q-learning, dueling architectures, and prioritized experience replay (for DQN).

3 Experiments and Results

In this section, I systematically investigate the effectiveness of several improvement techniques for both the REINFORCE and DQN algorithms. While REINFORCE provides an unbiased estimate of the policy gradient, it is known to suffer from high variance. DQN, although more stable, is sensitive to overestimation bias and sample inefficiency. To address these challenges, I apply analogous enhancements to each method and conduct ablation studies to isolate their effects.

REINFORCE Improvements

I implement and compare the following modifications to the vanilla REINFORCE algorithm:

- **Reward Normalization**: Standardizing episode returns to have zero mean and unit variance to stabilize gradient updates. This heuristic is widely used in modern RL implementations, including PPO [10].
- **Baseline Function**: To reduce the variance of the gradient estimate without introducing bias, I introduce a state-dependent baseline $b(s_t)$ and subtract it from the return G_t. This follows the formulation by Williams [11]. Further theoretical justification was provided by Greensmith et al. [13], who derived the form of an optimal baseline that minimizes variance. I use a learned value function $V^{\pi_\theta}(s_t)$ as the baseline by Luo [12], trained via mean squared error:

$$\mathcal{L}_{\text{critic}} = \frac{1}{2}\left(V^{\pi_\theta}(s_t) - G_t\right)^2$$

- **Generalized Advantage Estimation (GAE)**: I replace the baseline with a smoothed advantage estimate computed via GAE [14], which introduces a tunable bias-variance trade-off through the λ parameter.
- **Entropy Regularization**: To encourage exploration, I add an entropy bonus to the policy loss. This technique was introduced in A3C [15] and prevents premature convergence to deterministic actions.

DQN Improvements

To improve the stability and sample efficiency of DQN, I incorporate several key techniques drawn from the Rainbow DQN architecture [16], which combines multiple improvements into a unified agent. While Rainbow includes components such as Noisy Nets, Categorical DQN, and N-step returns, in this work I focus on a subset of its core techniques that are most relevant and compatible with my experimental framework:

- **Reward Clipping**: Following the original DQN paper [17], I clip all rewards to the range $[-1, 1]$ to prevent large reward magnitudes from destabilizing training.
- **Double DQN**: To reduce the overestimation bias commonly observed in Q-learning, I apply the Double DQN method proposed by Hasselt et al. [18], which decouples action selection and target Q-value estimation.
- **Dueling DQN**: To improve value estimation, I adopt the dueling architecture from Wang et al. [19], which separates the estimation of state values and action advantages, and combines them as:

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

- **Prioritized Experience Replay (PER)**: Inspired by Schaul et al. [20], I replace uniform sampling in the replay buffer with a prioritized scheme based on the temporal-difference (TD) error, so that transitions with higher learning potential are sampled more frequently.

Experimental Design

To evaluate the contribution of each individual technique, I design a series of experiments for both REINFORCE and DQN. For REINFORCE, I define seven configurations. Beginning with the vanilla algorithm, I independently add one variance-reduction or regularization component at a time to isolate its effect. Finally, I construct two complete models that combine reward normalization and entropy regularization with one of two advantage estimators: baseline or GAE.

The REINFORCE configurations are as follows:

1. **Vanilla REINFORCE**
2. **+ Reward Normalization**
3. **+ Baseline Function (Value Function)**
4. **+ Generalized Advantage Estimation (GAE)**
5. **+ Entropy Regularization**
6. **Complete (R-Norm + Baseline + Entropy)**
7. **Complete (R-Norm + GAE + Entropy)**

Both the baseline function and GAE serve to approximate the advantage function and are evaluated separately. The two complete models allow for comparing their integration effectiveness when used with all enhancements.

For DQN, I apply a parallel set of six configurations. I adopt commonly used techniques from Rainbow DQN to improve performance and stability. The configurations include:

1. **Vanilla DQN**
2. **+ Reward Clipping**
3. **+ Double DQN**
4. **+ Dueling DQN**
5. **+ Prioritized Experience Replay**
6. **Complete (Double + Dueling + PER)**

Each technique is introduced incrementally to observe its isolated contribution. While reward clipping is a standard technique in many Atari-based DQN implementations, initial experiments showed that it significantly degraded performance in the Assault environment. As a result, I excluded reward clipping from the final combined configuration to better highlight the synergistic effects of the remaining enhancements.

A summary of both experiment configurations is shown in Table 1 2.

By applying a consistent methodology to both REINFORCE and DQN, I am able to fairly compare how each technique influences learning performance, variance reduction, and overall training stability.

Table 1: Overview of experiment configurations and included techniques for REINFORCE

Setting	R-Norm	Baseline	GAE	Entropy
1. Vanilla	✗	✗	✗	✗
2. + R-Norm	✓	✗	✗	✗
3. + Baseline	✗	✓	✗	✗
4. + GAE	✗	✗	✓	✗
5. + Entropy	✗	✗	✗	✓
6. Complete (Baseline)	✓	✓	✗	✓
7. Complete (GAE)	✓	✗	✓	✓

Table 2: Overview of experiment configurations and included techniques for DQN

Setting	R-Clip	Double	Dueling	PER
1. Vanilla	✗	✗	✗	✗
2. + R-Clip	✓	✗	✗	✗
3. + Double	✗	✓	✗	✗
4. + Dueling	✗	✗	✓	✗
5. + PER	✗	✗	✗	✓
6. Complete	✗	✓	✓	✓

Evaluation Protocol

To ensure the robustness and reproducibility of the experimental results, each configuration is trained using three different random seeds. During training, I periodically evaluate the current policy on a fixed number of test episodes. For the **Lunar Lander** task, training is conducted for a total of 4000 episodes, and I perform testing every 20 episodes. For the **Assault** task, training is conducted for 10000 episodes, with testing every 50 episodes. In both cases, each test evaluates the agent on 10 episodes using a separate environment with a fixed random seed. The model checkpoint that achieves the highest average test return is saved and later used for final evaluation. This approach ensures that model selection is based on generalization performance rather than potentially noisy or optimistic training returns.

After training, the best-performing checkpoint for each seed is selected for final evaluation. These selected models are then tested over 100 episodes in the environment to obtain the final performance statistics. I report the mean and standard deviation of the returns across the three seeds. In addition to final performance, I also consider the speed of convergence during training as an important metric, as it reflects how quickly a given method is able to achieve satisfactory performance.

This evaluation strategy allows me to account for the high variance inherent in reinforcement learning and focus on the most performant policy achieved during training.

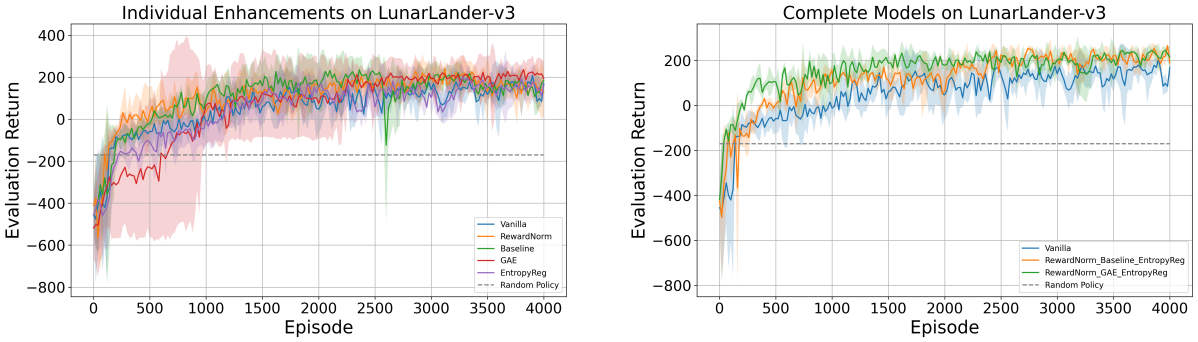
REINFORCE on LunarLander-v3

From the result table 3 and figure 1, I observe the following:

- Most enhancements improve performance over the vanilla REINFORCE, demonstrating their utility in stabilizing training and increasing sample efficiency.

Configuration	Average Return
1. Vanilla REINFORCE	195.32 ± 86.72
2. + Reward Normalization	224.21 ± 66.70
3. + Baseline	**241.01 ± 62.06**
4. + GAE	223.65 ± 77.37
5. + Entropy Regularization	195.77 ± 87.28
6. Complete (R-Norm + Baseline + Entropy)	223.08 ± 96.24
7. Complete (R-Norm + GAE + Entropy)	215.85 ± 92.95
Random Policy	-170.15 ± 98.33

Table 3: Performance of REINFORCE with different configurations on LunarLander-v3. **Boldface** denotes best performance



(a) Individual reinforcement techniques

(b) Combined configurations

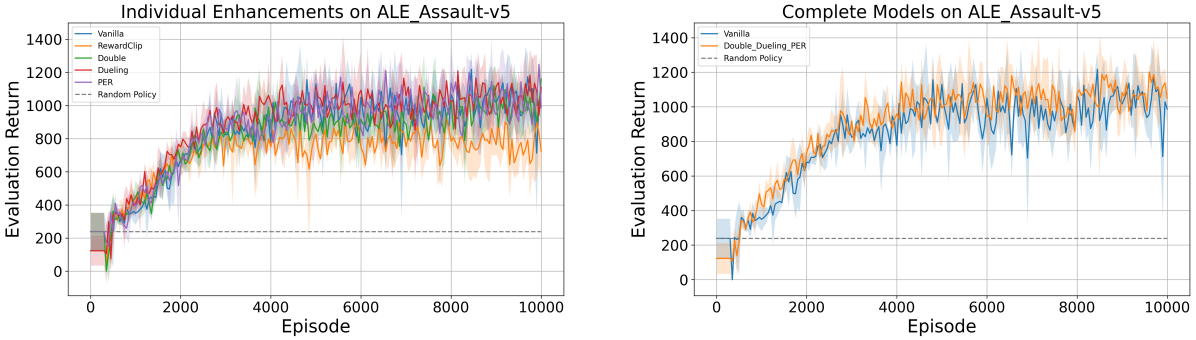
Figure 1: Performance comparison on LunarLander-v3 across different configurations.

- The configuration using a **baseline value function alone** (Configuration 3) achieves the best mean return and lowest variance among all settings. This supports the classical theory that baseline subtraction effectively reduces gradient variance without introducing bias.
- **Reward normalization** (Configuration 2) also shows a clear improvement over vanilla, although its benefit diminishes when combined with other techniques (Configurations 6 and 7), potentially due to interactions with entropy or advantage estimation.
- Surprisingly, the **complete settings** (Configurations 6 and 7) underperform compared to some simpler configurations, suggesting that combining all techniques does not necessarily lead to additive benefits. These configurations exhibit larger variance and slightly lower average returns, indicating possible interference among techniques.
- The **random policy** performs significantly worse than all learning-based configurations, with returns near -200, highlighting the necessity of learning a structured policy.
- Additionally, the use of **GAE** leads to unstable learning in this environment, likely due to inaccurate early value estimates. **Entropy regularization** contributes modestly by encouraging exploration but offers limited performance gain.

DQN on Atari Assault-v5

Configuration	Average Return
1. Vanilla DQN	1694.14 ± 759.49
2. + Reward Clipping	1137.80 ± 423.62
3. + Double DQN	1314.24 ± 577.37
4. + Dueling DQN	1468.01 ± 654.55
5. + Prioritized Replay	**1760.13 ± 716.01**
6. Complete (Double + Dueling + PER)	1583.43 ± 713.20
Random Policy	238.00 ± 69.30

Table 4: Performance of DQN variants on Assault. **Boldface** indicates best performance.



(a) Individual DQN techniques

(b) Combined configuration

Figure 2: Performance comparison on Atari Assault across different DQN configurations.

From the result table 4 and figure 2, I observe the following:

- While most enhancements yield some benefit, not all consistently outperform the vanilla DQN. Surprisingly, the **vanilla configuration** achieves strong performance, potentially due to lucky initialization or favorable early trajectories.
- **Prioritized Experience Replay (PER)** provides the most consistent improvement, achieving the highest average return. It focuses learning on transitions with large TD errors, improving sample efficiency and convergence.
- **Double DQN** and **Dueling DQN** do not individually surpass vanilla DQN. While theoretically they help reduce overestimation and improve value estimation, their standalone contributions in this environment may be limited due to interaction effects or sensitivity to hyperparameters.
- **Reward clipping**, though a widely used stabilization technique, significantly impairs performance in Assault. This supports concerns raised in the original DQN paper that clipping can obscure meaningful reward structures in games with varied reward magnitudes.
- The **complete configuration** (Double + Dueling + PER) underperforms compared to PER alone, indicating that combining enhancements does not always yield additive benefits and may introduce conflicting dynamics.

- As expected, the **random policy** performs poorly, reinforcing the necessity of learning structured Q-value approximations in complex environments like Atari.

Demonstration Videos and Code Repository

- **LunarLander (REINFORCE)**: Watch demo
- **Assault (DQN)**: Watch demo
- **Code Repository**: GitHub Link

4 Discussion

Through this project, I deepened my understanding of reinforcement learning algorithms by applying them to both low-dimensional (LunarLander-v3) and high-dimensional (Atari Assault) environments. Although I have studied reinforcement learning for over a year, this was my first time tackling Atari-based tasks, which introduced new challenges and insights.

Initially, I attempted to solve the Assault environment using the classic REINFORCE algorithm. However, due to its reliance on full-episode rollouts and high-variance gradient estimates, I observed frequent instability and unproductive policy learning. In particular, REINFORCE often converged to degenerate behaviors, such as dying early to avoid negative reward accumulation. This confirmed a known limitation of REINFORCE: its vulnerability to gradient explosion and its inefficiency in sparse or delayed-reward environments like Atari.

Recognizing these limitations, I pivoted to value-based methods and implemented a DQN agent. To explore the trade-offs and strengths of modern extensions, I integrated several techniques from the Rainbow DQN architecture, including reward clipping, Double DQN, Dueling networks, and Prioritized Experience Replay (PER). This hands-on exploration not only helped me appreciate the stabilizing effects of these improvements but also introduced me to practical concerns like sampling bias and memory efficiency. For me, PER stood out as a particularly interesting and impactful technique, offering both conceptual appeal and empirical benefit.

Remaining Questions:

- How would REINFORCE perform on high-dimensional tasks if used with advanced variance reduction techniques like trust region policy optimization (TRPO) or proximal policy optimization (PPO)?
- Could REINFORCE be made viable for Atari tasks through hybrid approaches, such as incorporating actor-critic structures with replay buffers or truncated episodes?
- What is the long-term impact of using PER on learning dynamics and convergence, especially with respect to overfitting to high-error transitions?

Overall, this project not only reinforced foundational concepts but also gave me hands-on experience with modern deep RL engineering. It highlighted the importance of algorithm-environment fit and inspired further curiosity in scalable policy gradient methods and exploration strategies.

References

[1] https://gymnasium.farama.org/environments/box2d/lunar_lander/

[2] https://ale.farama.org/environments/assault/

[3] https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63

[4] https://paperswithcode.com/method/reinforce

[5] https://medium.com/@shogulomkurganov73/atari-games-with-proximal-policy-optimization-ed28c7fafa3f

[6] https://github.com/lucaslingle/pytorch_ppo_atari

[7] https://github.com/DLR-RM/stable-baselines3/blob/master/stable_baselines3

[8] https://github.com/yyc0314/DQN-Atari-Breakout/blob/master/dqn_atari_breakout.ipynb

[9] https://github.com/rlcode/per/blob/master/prioritized_memory.py

[10] John Schulman et al. *Proximal Policy Optimization Algorithms*. arXiv preprint arXiv:1707.06347, 2017.

[11] Ronald J. Williams. *Simple statistical gradient-following algorithms for connectionist reinforcement learning*. Machine Learning, 1992.

[12] Ruotian Luo. *A Better Variant of Self-Critical Sequence Training*

[13] Lex Weaver, Nigel Tao. *The Optimal Reward Baseline for Gradient-Based Reinforcement Learning*. arXiv preprint arXiv:1301.2315, 2013.

[14] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. ICLR, 2016.

[15] Volodymyr Mnih et al. *Asynchronous Methods for Deep Reinforcement Learning*. ICML, 2016.

[16] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, David Silver. *Rainbow: Combining Improvements in Deep Reinforcement Learning*. arXiv preprint arXiv:1710.02298, 2017.

[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. *Human-level control through deep reinforcement learning*. Nature, 518(7540):529–533, 2015.

[18] Hado van Hasselt, Arthur Guez, David Silver. *Deep Reinforcement Learning with Double Q-learning.*. arXiv preprint arXiv:1509.06461, 2015.

[19] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, Nando de Freitas. *Dueling Network Architectures for Deep Reinforcement Learning*. arXiv preprint arXiv:1511.06581, 2015.

[20] Tom Schaul, John Quan, Ioannis Antonoglou, David Silver. *Prioritized Experience Replay*. arXiv preprint arXiv:1511.05952, 2015.