DQN LunarLander-v3(3 Different Modules)

1. Introduction

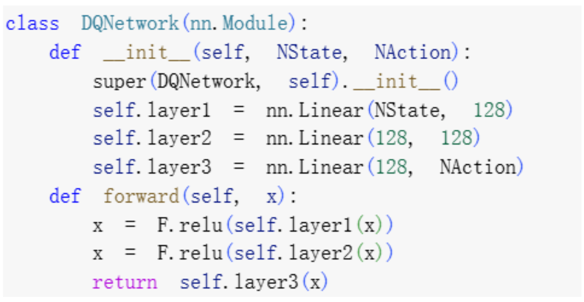
In this experiment, LunarLander-v3 is selected to design and implement a deep reinforcement learning agent to complete the control task in the environment. Deep Q-Network (DQN) is selected as the core algorithm to successfully control the spacecraft to land safely by training the agent, and to improve the cumulative reward by optimizing the strategy.

2. Code Implement

2.1 Environment and Parameters

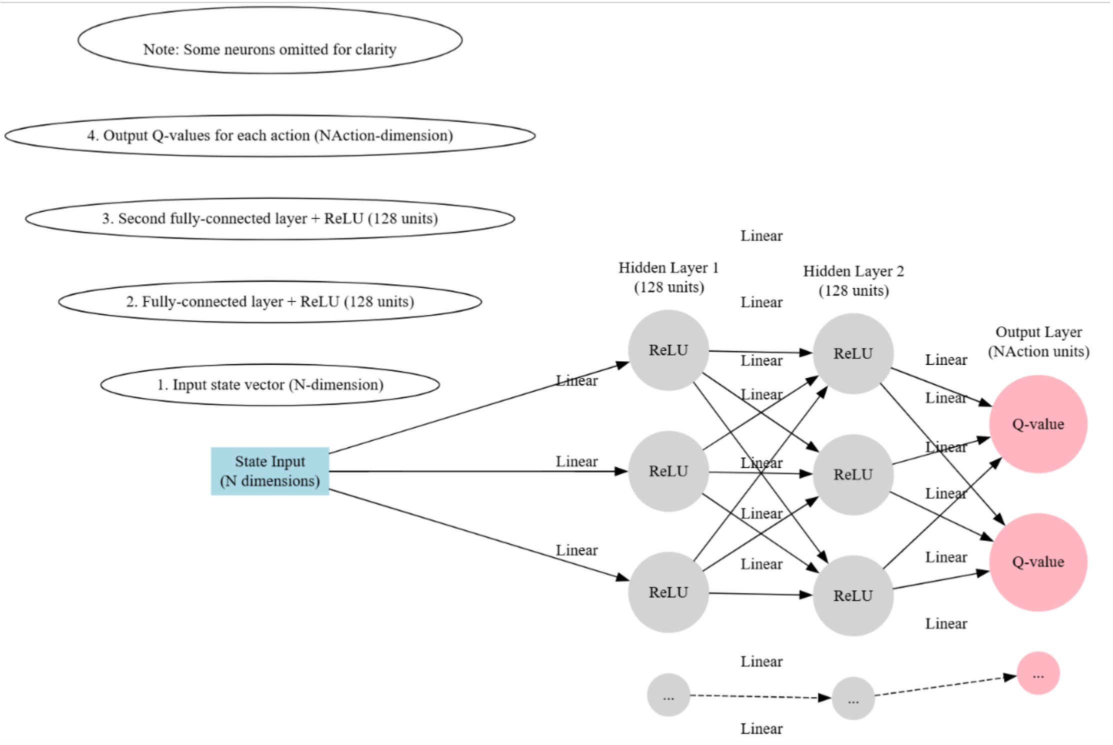
- **LunarLander-v3:** The state space is 8-dimensional continuous variables, and the action space is 4 discrete actions (main engine, small left propulsion, small right propulsion, no operation). Wind disturbances were disabled to stabilize test model performance.
- Setting **random seeds** ensures the repeatability of experiments
- **BATCHSIZE = 128:** The number of transitions sampled from the replay buffer in each training step is set to 128 to balance computational efficiency and stable gradient updates.
- **GAMMA = 0.99**： The valuation of future rewards as 0.99 indicates that future rewards are slightly discounted, encouraging the agent to prioritize long-term rewards.
- **EPSSTART = 1.0, EPSEND = 0.01, EPSDecay = 1000**： ε-Greedy policy parameter, controlling the transition from exploration to exploitation.
- **LR = 1e-4**： A small learning rate ensures a gradual update of the network weights, thus promoting stable learning.
- **TAU = 0.005**： Used to soft-update the coefficients of the target network to prevent violent fluctuations during training.
- According to the homework requirements, the training is 1000 episode.

2.2 Network design



```
class DQNetwork(nn.Module):
    def __init__(self, NState, NAction):
        super(DQNetwork, self).__init__()
        self.layer1 = nn.Linear(NState, 128)
        self.layer2 = nn.Linear(128, 128)
        self.layer3 = nn.Linear(128, NAction)
    def forward(self, x):
        x = F.relu(self.layer1(x))
        x = F.relu(self.layer2(x))
        return self.layer3(x)
```

- A three-layer fully connected feedforward neural network designed for Deep Q-Learning in the context of the LunarLander-v3 environment.
- Each layer uses a ReLU activation function and outputs the Q value corresponding to each possible action.


Note: Some neurons omitted for clarity
4. Output Q-values for each action (NAction-dimension)
3. Second fully-connected layer + ReLU (128 units)
2. Fully-connected layer + ReLU (128 units)
1. Input state vector (N-dimension)
State Input (N dimensions)
Hidden Layer 1 (128 units)
Hidden Layer 2 (128 units)
Output Layer (NAction units)
ReLU
Q-value
Linear
...

DQN-Network Workflow

2.3 Agent Design

2.3.1 Initialization

- **NState, NAction:** record the dimensions of the state space and action space.
- **ReplayMemory:** set the experience pool size to 10000 to disrupt the sample distribution, reduce the correlation of training samples, and help stabilize training.
- **PolicyNet, TargetNet:** build two neural networks. PolicyNet is the current policy network; TargetNet is the target network, which is used to calculate the target Q value to avoid excessive oscillation during the training process.
- **load_state_dict():** copy the PolicyNet parameters to TargetNet, and keep them consistent initially.
- **eval():** set TargetNet to evaluation mode to prevent interference from BatchNorm or Dropout layers.
- **Adam optimizer:** use the Adam algorithm for weight update, the learning rate is 1e-4, and amsgrad=True for a more robust learning process.
- **self.StepDone:** used to record the number of steps to dynamically adjust the exploration probability in the ε-greedy strategy.

2.3.2 Action selection function(Core mechanism: ε-greedy strategy)

- Exploration vs. Exploitation: adjust the ε value through exponential decay, encourage exploration in the early stage of training, and converge to the optimal strategy in the later stage.

- Policy network action selection: In the non-exploration stage, use the Q value output by PolicyNet to select the action corresponding to the maximum value (argmax(Q)).
- Action selection in the exploration stage: randomly select actions with the probability of ε to promote comprehensive exploration of the state space.

2.3.3 Storage and transfer

- **Transition:** A namedtuple storing (state, action, next state, reward, done).
- **ReplayMemory:** A deque with fixed capacity (10,000), supporting **push** (store transitions), **sample** (random batch), and **__len__** (buffer size).
- Use the **StoreTransition** function to store a state transition into the experience pool for batch sampling during subsequent training.

2.3.4 Optimizer

- Randomly sample a batch of transfer samples from the experience pool and convert them into batch tensor format for network training.
- Then separate the terminal state and non-terminal state, and only calculate the Q value of the next state for the non-terminal state. The future return of the terminal state defaults to 0.
- Calculate the current Q value: **PolicyNet** gives the Q value of all actions. Use **.gather()** to extract the Q value corresponding to the executed action.
- Use **TargetNet** to predict the maximum Q value of the next state and use the Bellman equation to calculate the target Q value.
- Calculate loss and back propagate: Use Huber loss (SmoothL1Loss) to improve robustness. Gradient clipping avoids gradient explosion and controls model training stability.

2.3.5 Update target network（Soft Update）

- Use weighted average update and use **TAU=0.005** to control the update rate to make the target network smoother and reduce training instability.

2.4 Training

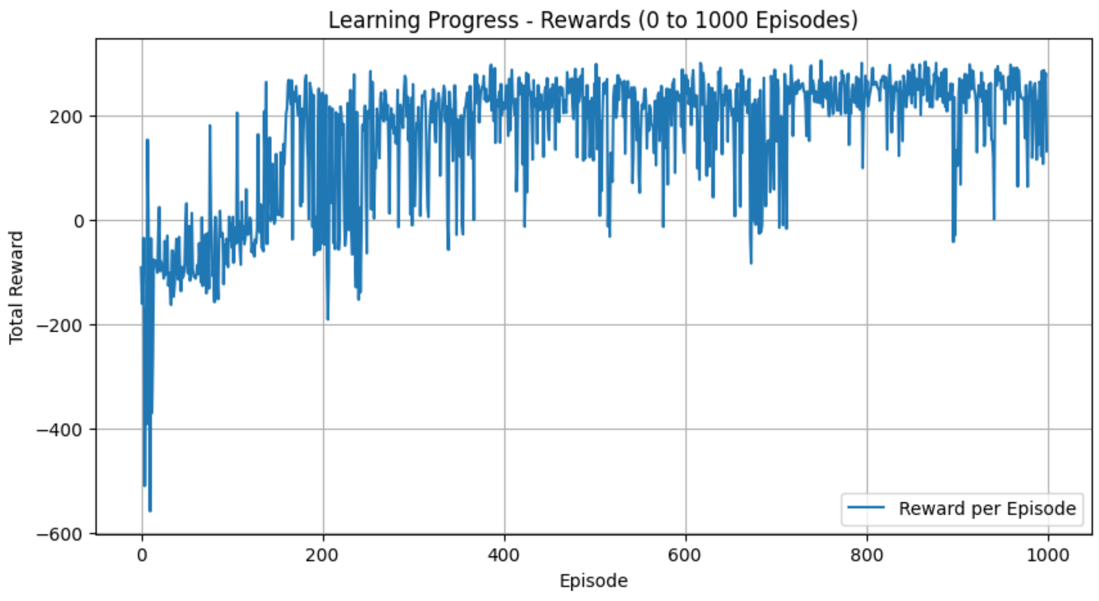
- Initialize the environment and agent.
- For each episode:
  1. Reset the environment
  2. Select an action at each step and record
  3. Save in replay buffer
  4. Call the optimizer to perform a parameter update

5. Update the target network parameters
6. Accumulate episode reward and loss
7. Save the average loss and total reward of each round

- Plot the trend

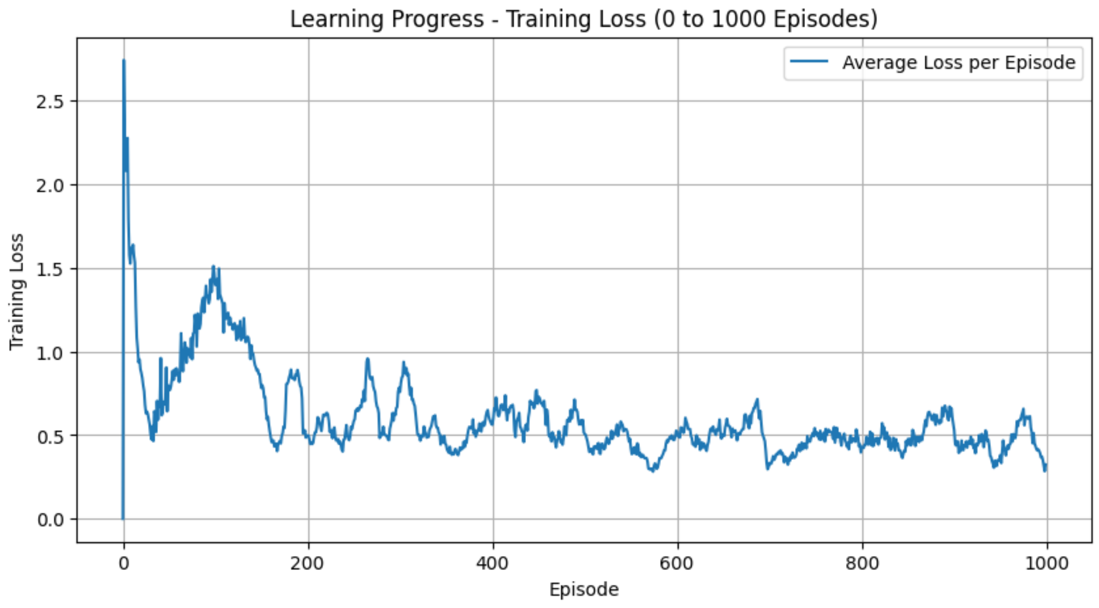
3. Results Discussion and Improvement

3.1.1 Pictures for No.1 Agent (1000 episodes)



- **Episode 0 to 100(Initial Exploration Phase)**
  1. The reward values are low, mostly below -500, and some even close to -800.
  2. This means that the agent cannot land safely most of the time, and even crashes frequently.
  3. At the beginning of the ε-greedy strategy, ε=1, it takes completely random actions.
- **Episodes 100 to 700(Improvement phase)**
  1. The reward gradually increases, and some episodes have rewards exceeding 100.
  2. This shows that the agent is learning quickly and has found some reasonable landing strategies.
  3. ε is decaying exponentially (gradually transitioning from exploration to exploitation).
- **Episode 700 to 1000(High-level fluctuation stage)**
  1. Most rewards are concentrated around 200 or even close to 300 (indicating that the agent successfully completed the task and landed smoothly).
  2. A few points of reward decline may be due to: A small number of states in

the environment are still unexplored. Although ε is close to EPSEND = 0.01, there are still a few exploration actions.



- **Loss Plot:** The training loss starts high (around 2.5), drops rapidly to around 0.5 within the first 100 episodes, and then fluctuates between 0 and 0.5 for the remaining episodes, showing no consistent downward trend.

3.1.2 Poor performance of the DQN agent11\1.mp4

- Phenomena Reflected by Results：
  1. Exploration-Exploitation Imbalance: The high variance in rewards and lack of improvement suggest the agent may be over-exploring or failing to exploit learned knowledge effectively.
  2. Training Instability: The fluctuating loss and reward variance point to instability in the Q-value updates.
  3. Lack of Learning Progress: The absence of an upward trend in rewards indicates the agent is not improving its policy over time.
- Reasons:
  1. Insufficient Training Episodes
  2. Hyperparameter Issues (Learning Rate, Epsilon Decay, Target Network Update)
  3. Network Architecture
  4. Exploration Strategy
  5. Reward Structure and Environment

3.2.1 Improved Agent2

Due to the large fluctuations in overall returns, the following measures are taken

to obtain a more stable model：

- **Improve the parameters：**
  1. Reduce the batch size to 64, which reduces memory usage and can lead to more stable updates by introducing more stochasticity, potentially improving convergence for tasks.
  2. Higher minimum epsilon to 0.05 ensures more exploration late in training, preventing the agent from becoming overly greedy and getting stuck in suboptimal policies.
  3. Larger replay memory to 100000 allows the agent to learn from a more diverse set of experiences, improving generalization and reducing overfitting to recent transitions.
  4. Introduce periodic hard updates of the target network every 50 steps, complementing soft updates.
  5. Other parameters remain unchanged to observe the changes
- **Neural Network Architecture:** The deeper and wider network enhances the agent's ability to model complex relationships between states and actions, leading to more accurate Q-value predictions and better decision-making.

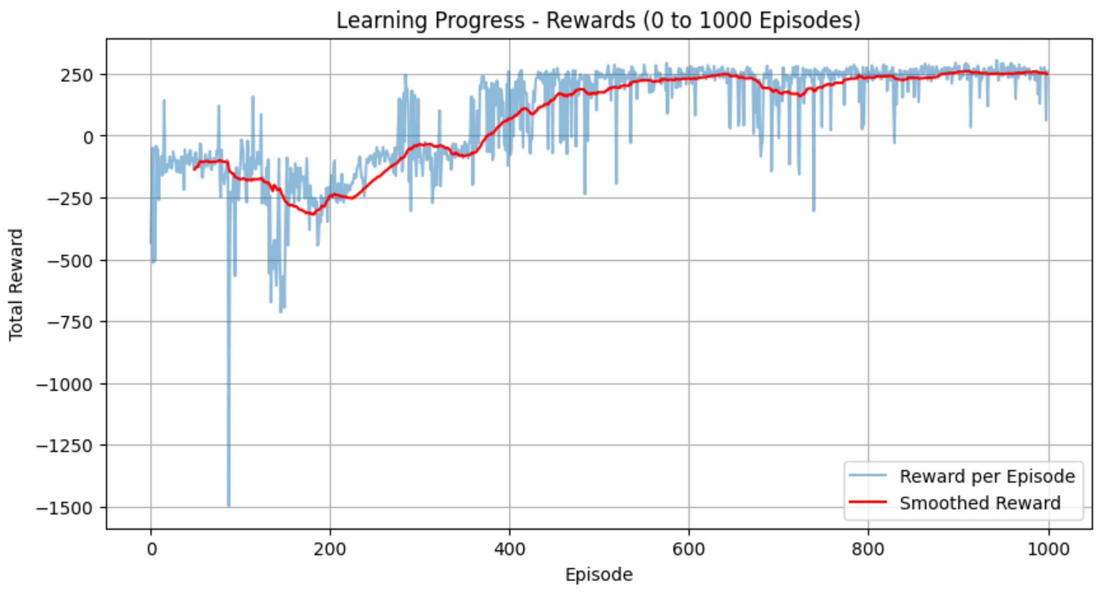
```
class DQNetwork(nn.Module):
    def __init__(self, NState, NAction):
        super(DQNetwork, self).__init__()
        self.layer1 = nn.Linear(NState, 512)
        self.layer2 = nn.Linear(512, 256)
        self.layer3 = nn.Linear(256, 128)
        self.layer4 = nn.Linear(128, NAction)
    def forward(self, x):
        x = F.relu(self.layer1(x))
        x = F.relu(self.layer2(x))
        x = F.relu(self.layer3(x))
        return self.layer4(x)
```

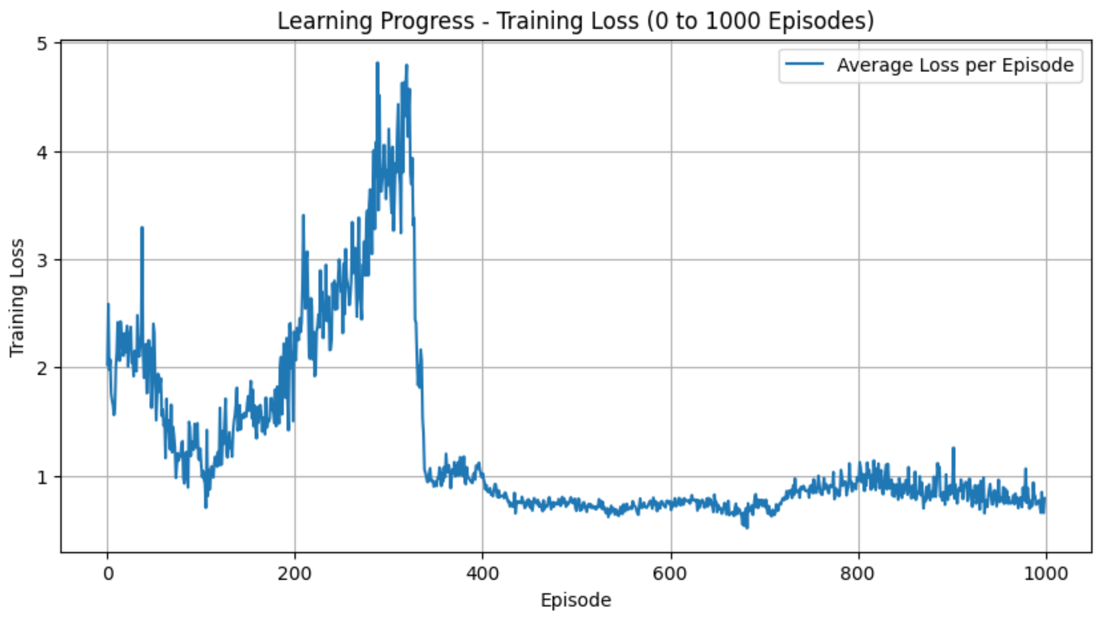
- **Training Logic Enhancements:**
  1. Adds bonuses: distance, speed, angle (Introduce reward shaping to guide the agent toward desirable states (closer to landing pad, lower speed, upright orientation), accelerating learning.)
  2. Early Stopping: Prevent overfitting and saves computational resources if performance degrades significantly.
  3. Soft update + hard update every 50 steps: Ensure the target network tracks the policy network more effectively, reducing divergence in Q-value estimates.
- **Optimization Improvements:**

    1. Gradient Clipping: Use norm-based clipping, which is more robust, preventing extreme updates.
    2. Q-Value Computation: Implement Double DQN, reducing overestimation of Q-values by decoupling action selection and evaluation, leading to more stable training.

- Add a dedicated testing phase to evaluate the trained agent's performance, providing a quantitative measure of success (average reward).
- Add smoothed rewards make it easier to visualize training trends, reducing noise in the reward plot.

3.2.2 Pictures for No.2 Agent2\2.mp4





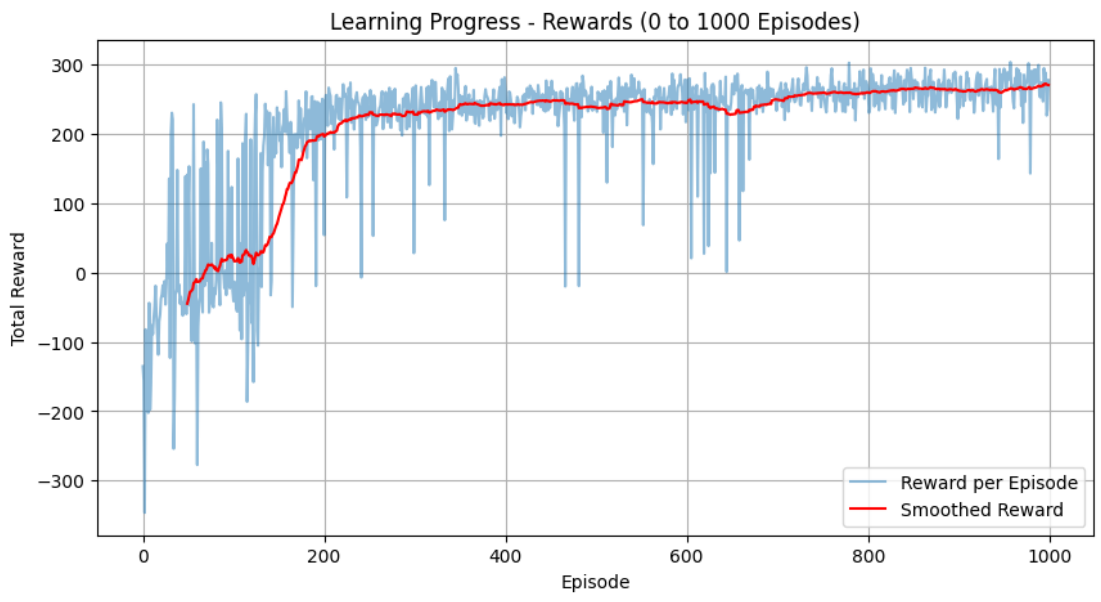
Average Test Reward over 100 episodes: 95.61

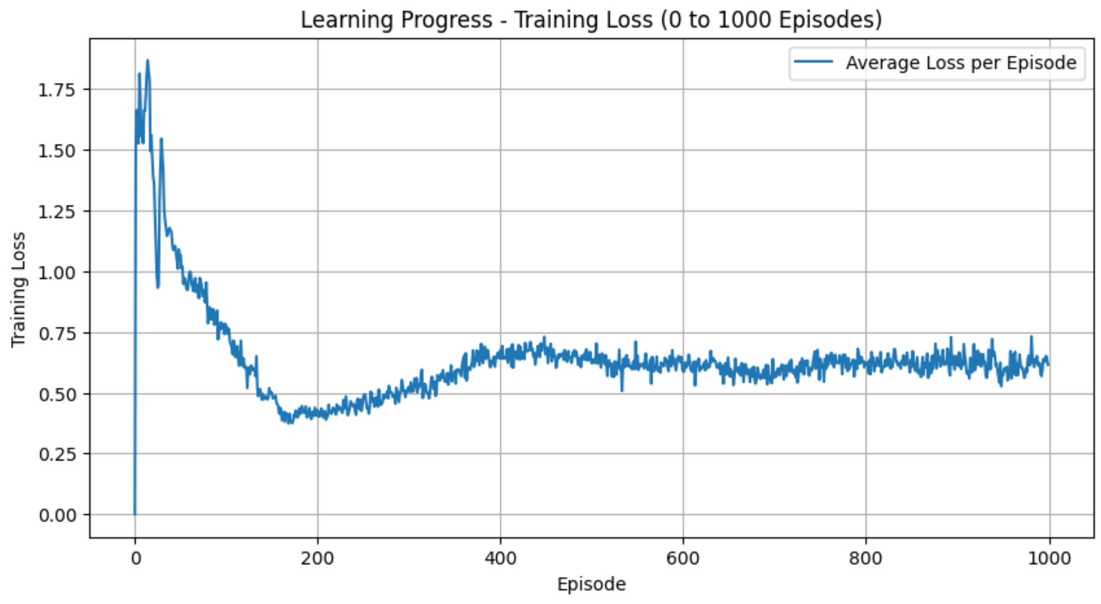
- **Converging Loss:** The loss drops to near 0, showing effective Q-value learning, but there are occasional small fluctuations, and it will be unstable when encountering difficult situations.
- **Improved Rewards:** Training rewards rise from -1000 to ~50, and the test reward reaches 95.61, indicating the agent learns meaningful behaviors.
- The plateauing reward and residual loss spikes indicate the agent is stuck in a suboptimal policy, likely due to insufficient exploration, overly harsh reward shaping, or sensitivity to challenging transitions.
- However, spikes in loss (e.g., around episodes 300 and 900) suggest occasional instability, possibly due to exploration introducing new experiences that temporarily disrupt Q-value estimates.
- It has learned to avoid major failures and can do things according to the guidance of rewards, but it is unstable, which may be due to insufficient exploration, too strict rewards and punishments, and immature strategies. The rewards did not increase at the end, indicating that there is still room for improvement.

3.3.1 Improved Agent3

- Increasing the batch size allows the model to process more samples per optimization step, which can lead to more stable gradient updates and potentially faster convergence.
- Increasing the Epsilon decay to 2000 means the epsilon value decays more slowly. This allows the agent to explore more thoroughly in the early stages, potentially finding better strategies.
- According to the previous results, reduce the penalty weights for distance, speed, and angle. This makes the reward signal less harsh, allowing the agent to learn more effectively without being overly penalized for small deviations.
- Removing the early stopping mechanism helps to better train and prevent the learning potential from being limited. At the same time, data is printed every 100 episodes to clearly monitor the learning situation.

3.3.2 Pictures for No.3 Agent3\3.mp4





Average Test Reward over 100 episodes: 130.20

- The reward per episode is more stable than the previous agent, ranging mostly between -300 and 300, with the smoothed reward steadily increasing to around 250–300 after episode 400. This suggests the agent learns a more consistent and effective policy.
- The training loss starts lower (around 1.75), decreases steadily to 0.5–0.75, and is less noisy, reflecting more stable training dynamics.
- By comparison, we can see that the previous experiment had a smaller batch size, so the fluctuations were larger each time, while the smoothed reward was only stable around 0-50. Increasing the batch size directly improves the convergence speed and stability of the model, making the training process more

predictable and the reward curve smoother.

- Although various modifications have a great impact on the final result, the most critical one should be the increase in Epsilon decay. The decay rate determines the transition speed from exploration to exploitation, and a higher ESPDecay prolongs the exploration phase. This is especially important in the early stages of training because it allows the agent to explore the environment more fully and avoid falling into a local optimum too early. As can be seen from the reward graph, the smoothed reward of Agent3 is higher in the later stages of training, reaching 250-300, while the previous experiment only reached 250 at 600 episodes and then fluctuated greatly, which show that Agent3 has found a better strategy.

4 Explain exploration and exploitation for deep reinforcement learning

From the above training analysis：

Early stage of training

- At the beginning of the learning phase, the rewards are generally low and fluctuate greatly.
- This is the exploration-oriented phase, where the agent tries various actions, even if they do not seem "smart".
- The loss may fluctuate greatly at this stage because the experience samples are not stable and the strategy is constantly being updated.

Middle of training

- The reward starts to rise and stabilize, indicating that the agent has found a better strategy during exploration.
- ε gradually decreases, exploration decreases, and exploitation begins to work.
- Q-Loss gradually stabilizes, indicating that the Q network has a good convergence effect.

Late training

- The agent is mainly in exploitation, and the reward is more stable and close to convergence.
- ε is close to the lowest, and most of the time the action with the largest Q value is selected.
- If the reward still fluctuates greatly at this stage, it may mean that the training is not enough or the strategy is still unstable. At this time, you can increase the quality of the replay buffer or the number of training episodes.

Among them, the stronger the exploration, the more diverse the behaviors in training, the larger the variance but the higher the potential; the stronger the exploitation, the more consistent the behaviors, the smaller the variance but the easier it is to fall into the local optimum. Balancing the two can ensure that the strategy is fully learned and the training process converges and stabilizes.

There are several important factors that affect exploration and exploitation:

- Epsilon Decay:
    1. ε decays too quickly → insufficient early exploration → the agent may fall into a local optimum, stable training but poor final performance.
    2. ε decays too slowly → excessive exploration → large reward fluctuations and unstable training.
- batch size: A larger batch size helps improve the stability of policy evaluation and enables the agent to more stably utilize existing experience. However, a large batch size may also reduce the sensitivity to recent exploration experience, indirectly weakening the short-term policy adjustment ability brought by exploration.
- Target Network Update and Learning Rate: High learning rate + more frequent target updates + stronger reward shaping = faster convergence, but more prone to instability; while low learning rate + stable target network + conservative exploration strategy = more stable but slower.

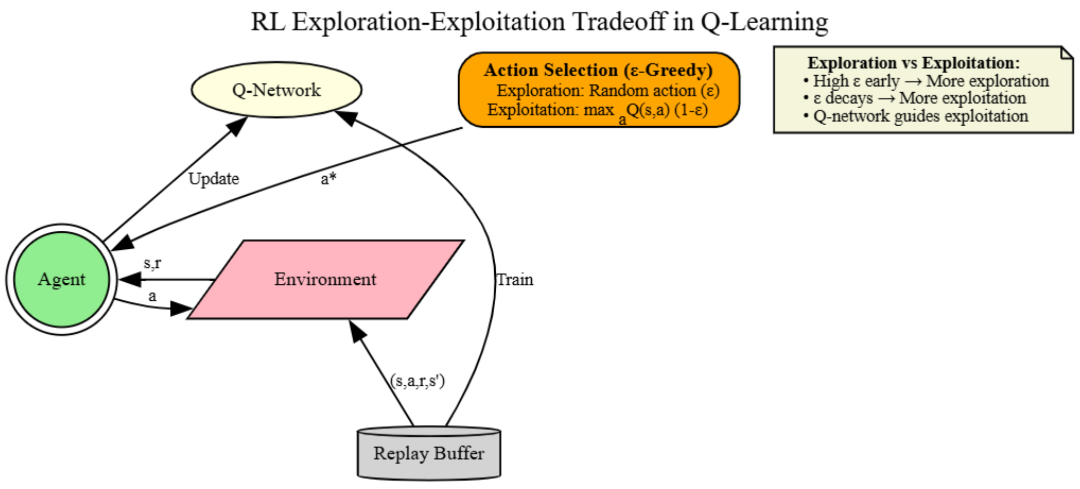
Overall, in Deep Reinforcement Learning, an agent learns a strategy to maximize cumulative rewards by interacting with the environment. One of the core challenges in this process is to strike a balance between exploration and exploitation.

In the LunarLander environment, the agent might try landing at different angles or speeds to learn which behaviors are safe and effective, which is a process of exploration. When the agent, after multiple rounds of training, tends to choose actions that are known to bring high rewards, which is the exploitation process.

Over-exploration: may lead to slow and unstable training.

Over-exploitation: may lead to being stuck in a local optimum, lack of innovation or being unable to escape from inefficient strategies.

ε decreases over time, that is, the early stage of training focuses on exploration, and the later stage focuses on utilization. The decay process of ε controls the transition rhythm from exploration to exploitation. It is necessary to allow the agent to fully learn to master the estimated values and of different actions in the early stage so that it can fully utilize them to achieve efficient and stable strategies in the later stage.


RL Exploration-Exploitation Tradeoff in Q-Learning
Q-Network
Action Selection (ε-Greedy)
Exploration: Random action (ε)
Exploitation: max_a Q(s,a) (1-ε)
Exploration vs Exploitation:
• High ε early → More exploration
• ε decays → More exploitation
• Q-network guides exploitation
Update
a*
Agent
s,r
a
Environment
Train
(s,a,r,s')
Replay Buffer

Exploration allows the agent to discover new possibilities, while utilization allows it to make the best decision based on known knowledge. The balance between the two is the key to the success of reinforcement learning.