Lab5 Report

Deep Learning

Value-Based Reinforcement Learning

姓名：陳科融

學號：314551010

July 30, 2025

1. Introduction

本實驗實作並強化深度強化學習中的 Deep Q-Network（DQN）演算法，首先於較為簡單的 CartPole 環境中驗證其基本功能，透過 Bellman Error 進行梯度更新，並成功達到最高分數。隨後將相同架構應用於難度較高的 Atari 遊戲 Pong，並逐步引入多項先進技術以提升訓練效率與模型表現，包括 Prioritized Experience Replay、Multi-step Return、Double DQN 等方法。

2. Implementation

2.1 How do you obtain the Bellman error for DQN?

在強化學習中，**Bellman error（貝爾曼誤差）**是 Q-learning 的核心概念。

- 它代表**目前對** Q 值的估計（預測）與實際目標值（TD target）的**差距**。
- 在 DQN 中，用這個誤差作為訓練，透過最小化它來更新神經網路參數。

Interpretation 1：Q-value 插值更新

這種寫法可以理解為舊的 Q 值與新的 TD 目標做加權平均。

$$\boldsymbol{Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha\left(r_{t+1} + \gamma \max_{a'} Q\left(s_{t+1}, a'\right)\right)}$$

Interpretation 2：誤差修正（Bellman Error）

這裡強調我們透過 TD 誤差（也就是 Bellman error）修正原本的 Q 值。

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left(r_{t+1} + \gamma \max_{a'} Q\left(s_{t+1}, a'\right) - Q(s_t, a_t)\right)$$

Bellman Error 定義

令 TD Target 為：

$$y_t = r_{t+1} + \gamma \max_{a'} Q\left(s_{t+1}, a'\right)$$

那麼 Bellman error 就是：

$$\delta_t = y_t - Q(s_t, a_t)$$

DQN 並不能直接更新 Q 值，而是透過最小化損失函數學習 Q 函數：

$$\mathcal{L} = \left(y_t - Q(s_t, a_t; \theta)\right)^2$$

$Q(s_t, a_t; \theta)$ 是使用 policy network 預測的 Q 值

y_t 是根據 target network 計算的 TD target

$\delta_t = y_t - Q(s_t, a_t)$ 即為 Bellman 誤差

```
self.policy_network.train() # 設定為訓練模式
self.target_network.eval() # 設定為評估模式

# 從回放緩衝區中採樣一個 mini-batch
batch = random.sample(self.replay_buffer, self.batch_size)
state_batch, action_batch, reward_batch, next_state_batch, done_batch = zip(*batch)

# 將採樣的數據轉換為 PyTorch 張量
state_batch = torch.from_numpy(np.array(state_batch)).float().to(self.device)
next_state_batch = torch.from_numpy(np.array(next_state_batch)).float().to(self.dev
action_batch = torch.tensor(action_batch, dtype=torch.int64).view(-1, 1).to(self.de
reward_batch = torch.tensor(reward_batch, dtype=torch.float32).view(-1, 1).to(self.
done_batch = torch.tensor(done_batch, dtype=torch.bool).view(-1, 1).to(self.device)

# 計算當前 Q 值 (Q(s,a))
current_q_values = self.policy_network(state_batch).gather(1, action_batch)

# 計算目標 Q 值 (r + gamma * max(Q_target(s',a')))
with torch.no_grad():
    next_q_values = self.target_network(next_state_batch).max(1)[0].view(-1, 1)
    target_q_values = reward_batch + self.gamma * next_q_values * (~done_batch)

# 計算損失 (MSE Loss)
loss = nn.MSELoss()(current_q_values, target_q_values)

# 優化器步驟：清零梯度、反向傳播、更新權重
self.optimizer.zero_grad()
loss.backward()
```

程式碼 1. Bellman Error 架構

2.2 How do you modify DQN to Double DQN?

Double Q-Learning 原本的想法如下：

- 使用兩個獨立的 Q 網路：Q_A和 Q_B
- 每次更新使用其中一個 Q 網路進行**動作選擇**，另一個 Q 網路進行**價值評估**
- 避免 max 運算時放大 noise 而產生高估

$$Q_A(s_t, a_t) \leftarrow Q_A(s_t, a_t) + \alpha \cdot \left(r_{t+1} + \gamma \cdot Q_B\left(s_{t+1}, \arg\max_a Q_A(s_{t+1}, a)\right) - Q_A(s_t, a_t) \right)$$

$$Q_B(s_t, a_t) \leftarrow Q_B(s_t, a_t) + \alpha \cdot \left(r_{t+1} + \gamma \cdot Q_A\left(s_{t+1}, \arg\max_a Q_B(s_{t+1}, a)\right) - Q_B(s_t, a_t) \right)$$

Double DQN 將這個想法應用在 DQN 上，使用：

- 使用 eval_net（即 policy network）選擇 下一狀態下最優動作：

$$a^* = \arg\max_{a'} Q_{\text{policy}}(s_{t+1}, a')$$

- 再由 target_net 評估該動作的 Q 值：

$$Q_{target}(s_{t+1}, a^*)$$

- 因此，Double DQN 的 TD 目標值為：

$$y_t = r_t + \gamma \cdot Q_{\text{target}}\left(s_{t+1}, \arg\max_{a'} Q_{\text{policy}}(s_{t+1}, a')\right)$$

- Loss function

$$L_{\text{DDQN}}(\boldsymbol{w}) = E_{(s,a,r,s')\sim\mathbb{D}}\left[\left(r + \gamma Q\left(s', \arg\max_{a'} Q(s', a'; \boldsymbol{w}); \bar{\boldsymbol{w}}\right) - Q(s, a; \boldsymbol{w})\right)^2\right]$$

```
current_q_values = self.policy_network(state_batch).gather(1, action_batch.unsqueeze(1)).squeeze(1)

with torch.no_grad():
    # Double DQN: 使用 online network 選擇動作
    best_actions = self.policy_network(next_state_batch).argmax(1)
    # Double DQN: 使用 target network 評估該動作的 Q-value
    next_q_values = self.target_network(next_state_batch).gather(1, best_actions.unsqueeze(1)).squeeze(1)
```

程式碼 2. Double DQN

2.3 How do you implement PER?

在傳統的經驗回放中，樣本是均勻隨機抽取，而 Prioritized Experience Replay (PER) 則根據每筆樣本的 TD 誤差(Bellman error) 來決定抽樣的優先權，讓 agent 更常練習「學習效果較大的經驗」，提升學習效率。

優先權計算

每筆樣本 i 的優先權p_i通常設定為該樣本的 TD 誤差的絕對值加上常數 ϵ，避免優先權為零：

$$\delta_i = r_i + \gamma \max_{a'} Q(s_i', a'; w) - Q(s_i, a_i; w), p_i = |\delta_i| + \epsilon$$

$$\text{抽樣的機率 } P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

Importance Sampling（IS）權重校正

由於優先抽樣會造成估計偏差，利用 IS 權重校正：

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

N 為 replay buffer 大小。

β 隨訓練進展由小逐漸升高到 1，減少偏差。

```python
def sample(self, batch_size):
    """根據優先級抽樣"""
    if len(self.buffer) == self.capacity:
        prios = self.priorities
    else:
        prios = self.priorities[:self.pos]

    probs = prios ** self.alpha
    probs /= probs.sum()

    indices = np.random.choice(len(self.buffer), batch_size, p=probs)
    samples = [self.buffer[idx] for idx in indices]

    # 計算重要性採樣 (IS) 權重
    total = len(self.buffer)
    weights = (total * probs[indices]) ** (-self.beta)
    weights /= weights.max()

    # 更新 beta
    self.beta = min(1.0, self.beta + self.beta_increment)

    return samples, indices, torch.from_numpy(weights).float()
```

程式碼 3. 根據優先級進行抽樣

```
# PER: 計算 TD-error 並更新 priorities
errors = (current_q_values - target_q_values).abs().detach().cpu().numpy()
self.replay_buffer.update_priorities(indices, errors)

# PER: 使用 IS weights 加權 loss (改用 Smooth L1 Loss)
loss = (weights * nn.functional.smooth_l1_loss(current_q_values, target_q_values, reduction='none')).mean()
```

程式碼 4. PER 計算更新 priority

2.4 How do you modify the 1-step return to multi-step return

傳統 DQN 的 1-step TD Target：

這表示只考慮「下一步」的即時獎勵與估計的未來價值。

$$y_t^{(1)} = r_t + \gamma \cdot \max_{a'} Q\left(s_{t+1}, a'; \bar{w}\right)$$

Multi-step TD Target（展開形式）：

$$y_t^{(n)} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n \max_{a'} Q\left(s_{t+n}, a'; \bar{w}\right)$$

Multi-step TD Target（簡潔總和形式）：

$$y_t^{(n)} = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n \max_{a'} Q\left(s_{t+n}, a'; \bar{w}\right)$$

```python
def _get_multi_step_transition(self):
    """計算 n-step return"""
    reward = 0
    # 從 buffer 的最舊的經驗開始，計算 n-step 的折扣獎勵
    for i in range(self.args.multi_step):
        reward += (self.gamma**i) * self.multi_step_buffer[i][2]

    # 取出最舊的 state, action 和最新的 next_state, done
    state, action, _, _, _ = self.multi_step_buffer[0]
    _, _, _, next_state, done = self.multi_step_buffer[-1]

    return state, action, reward, next_state, done
```

程式碼 5. Multi step 計算架構

```python
with torch.no_grad():
    # Double DQN: 使用 online network 選擇動作
    best_actions = self.policy_network(next_state_batch).argmax(1)
    # Double DQN: 使用 target network 評估該動作的 Q-value
    next_q_values = self.target_network(next_state_batch).gather(1, best_actions.unsqueeze(1)).squeeze(1)

    # Multi-step return: gamma 要用 n 次方
    target_q_values = reward_batch + (self.gamma ** self.args.multi_step) * next_q_values * (1 - done_batch)
```

程式碼 6. Multi step 計算 n 步之後的獎勵價值

2.5 how you use Weight & Bias to track the model performance

在本實驗中，我們整合了 Weights & Biases (WandB) 作為訓練過程中的視覺化與監控工具，用來即時追蹤模型的學習進度與效能變化。

- Train/Loss（訓練損失）
- Evaluate Reward（測試平均回報）
- Progress/Epsilon（探索率變化）
- Progress/Learning Rate（學習率）

```python
# 修改：採用高效的「定期評估」策略
# 每 20 個回合評估一次模型
if (episode_index + 1) % 20 == 0:
    # 進行 20 個回合的評估
    current_eval_reward = self.evaluate_performance(episodes=20)
    self.eval_rewards_history.append(current_eval_reward)
    avg_eval_reward = np.mean(self.eval_rewards_history)

    # 更新學習率調度器
    self.scheduler.step(avg_eval_reward)

    print(f"\n--- Evaluation at Episode {episode_index+1} ---")
    print(f"Avg Eval Reward (last {len(self.eval_rewards_history)} eval

    wandb.log({
        "Evaluate Reward": current_eval_reward,
        "Progress/Learning Rate": self.optimizer.param_groups[0]['lr']
    }, step=self.env_step_count)

    # 如果平均獎勵創新高，儲存模型
    if avg_eval_reward > self.best_eval_reward:
        self.best_eval_reward = avg_eval_reward
        model_path = os.path.join(self.save_dir, f"LAB5_{self.student_i
        torch.save(self.policy_network.state_dict(), model_path)
        print(f"*** New best model saved to {model_path} with avg rewar
```

程式碼 7. 定期評估

```python
    with torch.no_grad():
        # Double DQN: 使用 online network 選擇動作
        best_actions = self.policy_network(next_state_batch).argmax(1)
        # Double DQN: 使用 target network 評估該動作的 Q-value
        next_q_values = self.target_network(next_state_batch).gather(1, best_

        # Multi-step return: gamma 要用 n 次方
        target_q_values = reward_batch + (self.gamma ** self.args.multi_step)

    # PER: 計算 TD-error 並更新 priorities
    errors = (current_q_values - target_q_values).abs().detach().cpu().numpy(
    self.replay_buffer.update_priorities(indices, errors)

    # PER: 使用 IS weights 加權 loss (改用 Smooth L1 Loss)
    loss = (weights * nn.functional.smooth_l1_loss(current_q_values, target_q

    self.optimizer.zero_grad()
    loss.backward()
    torch.nn.utils.clip_grad_norm_(self.policy_network.parameters(), 1.0)
    self.optimizer.step()

    if self.train_count % self.args.target_update_frequency == 0:
        self.target_network.load_state_dict(self.policy_network.state_dict())

    if self.train_count % 1000 == 0:
        wandb.log({
            "Train/Loss": loss.item(),
            "Progress/Epsilon": self.epsilon,
        }, step=self.env_step_count)
```

程式碼 8. 訓練次數達到 1000 的整數倍做紀錄

3. Analysis and discussions

3.1 Training curve

Task1

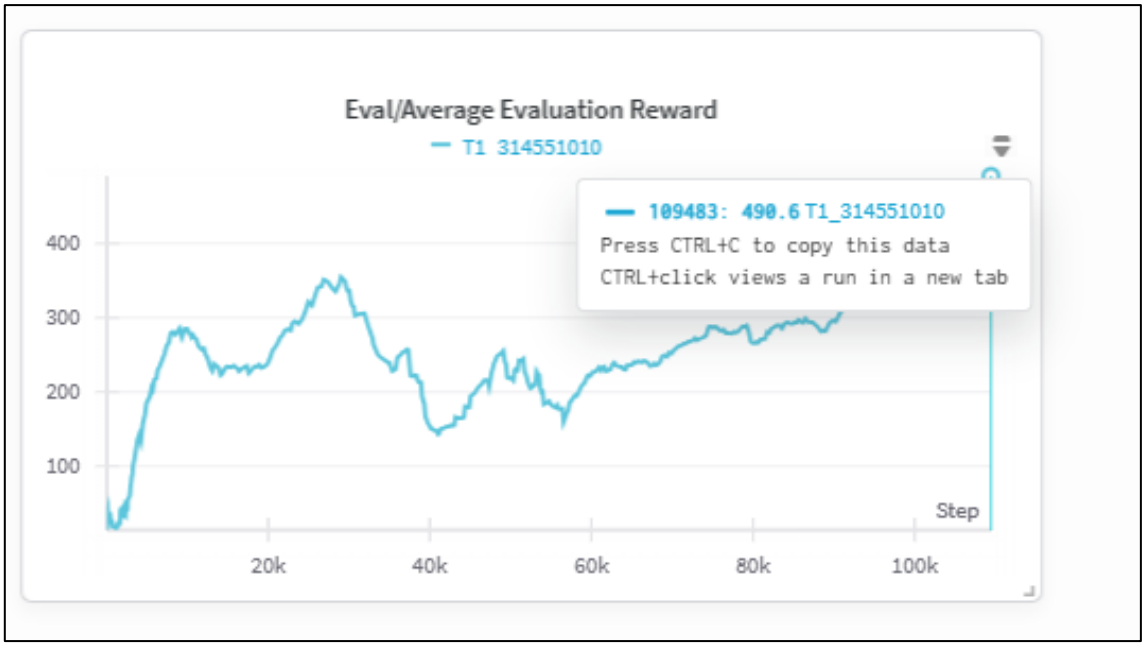


圖 1. Cartople avg reward

Task2

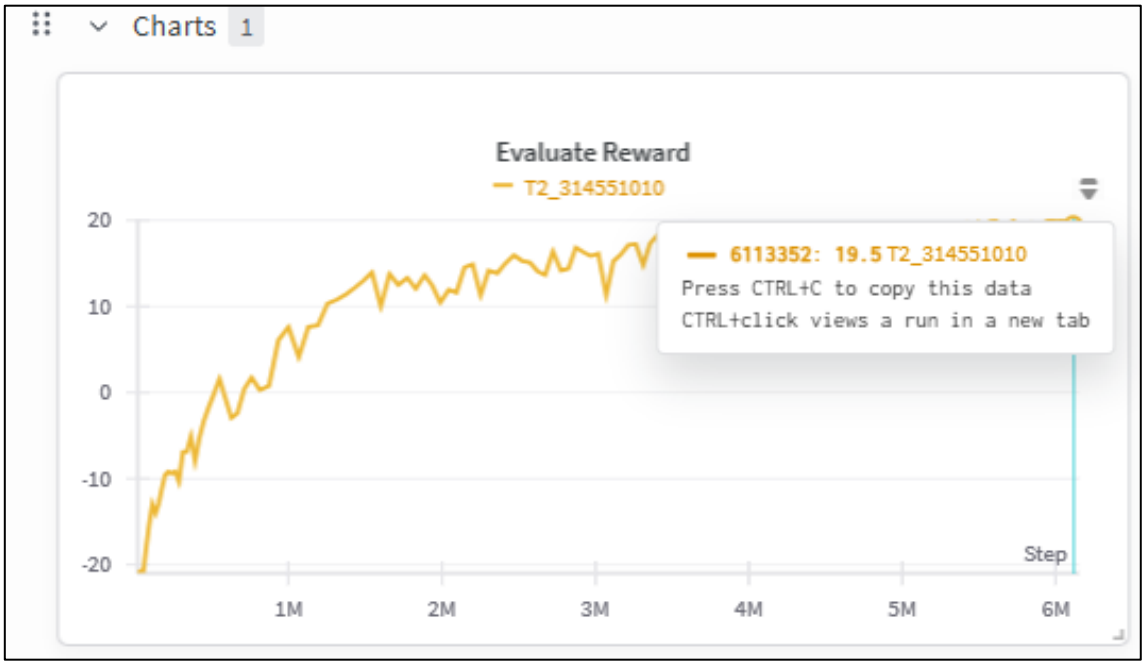


圖 2. Pong average reward

Task3

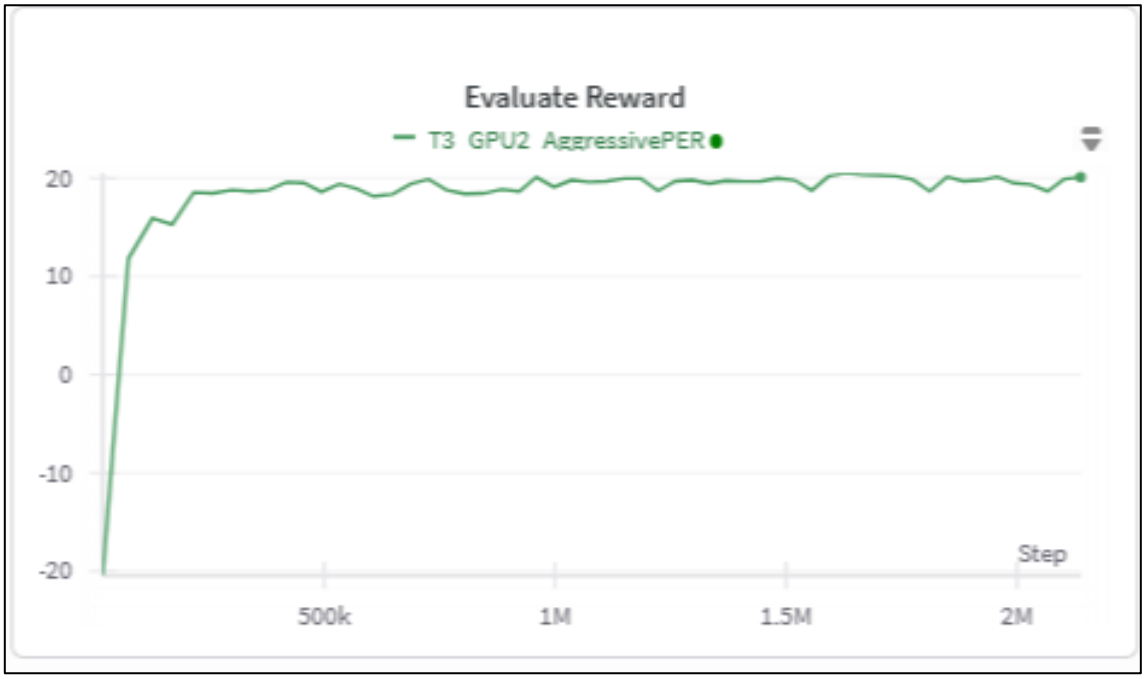
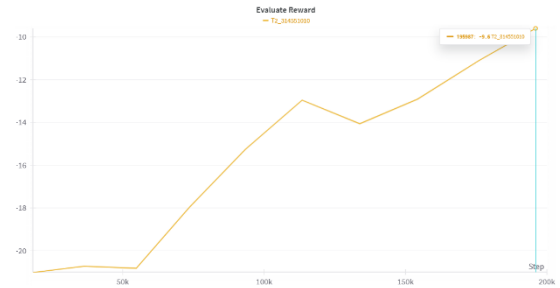
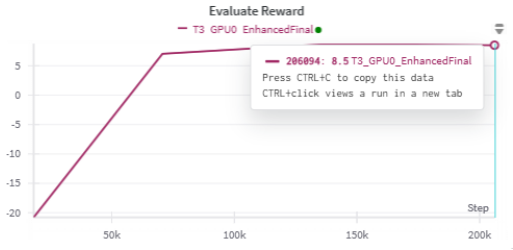


圖 3. Pong Enhance average reward

3.2 Sample efficiency 分析

在使用 DQN 方法的前提下，我比較了各版本在 200k 個環境互動步數（env steps）時的分數表現。結果顯示，僅僅在 200k 步之內，強化過的 DQN 版本（整合了 Prioritized Experience Replay、Double DQN 與 Multi-Step Return）即已展現出明顯優勢，分數差距可達十多分，顯示其在樣本效率（sample efficiency）上的提升十分顯著，這代表這些增強技術能夠讓 agent 更快速且有效地學習策略，提早收斂至更高的性能水準。

DQN	Enhanced DQN
	

3.3 Ablation study

Prioritized Experience Replay（PER）

使得 agent 更頻繁地回放高 TD-error 的關鍵經驗，聚焦於學習價值高、學習難度大的 transition，避免樣本浪費。

- **改回 uniform replay（均勻抽樣）後：**代理人將無法聚焦於「學習價值較高的 transition」。

Double DQN

解決了原始 DQN 中對 Q 值的高估問題，使得 Q-value 更新更加穩定，間接提升了策略學習的可靠性。

- **回到原始 DQN：**容易出現 **Q 值高估**（overestimation bias）。

Multi-Step Return

提供了更遠視的回饋，使得 agent 能夠更快地建立對長期獎勵的預期，進而加速價值函數的學習。

- **只使用單步 TD（n=1）更新：**學習會依賴較短期的回饋訊號，**遠期回報的影響減弱**。

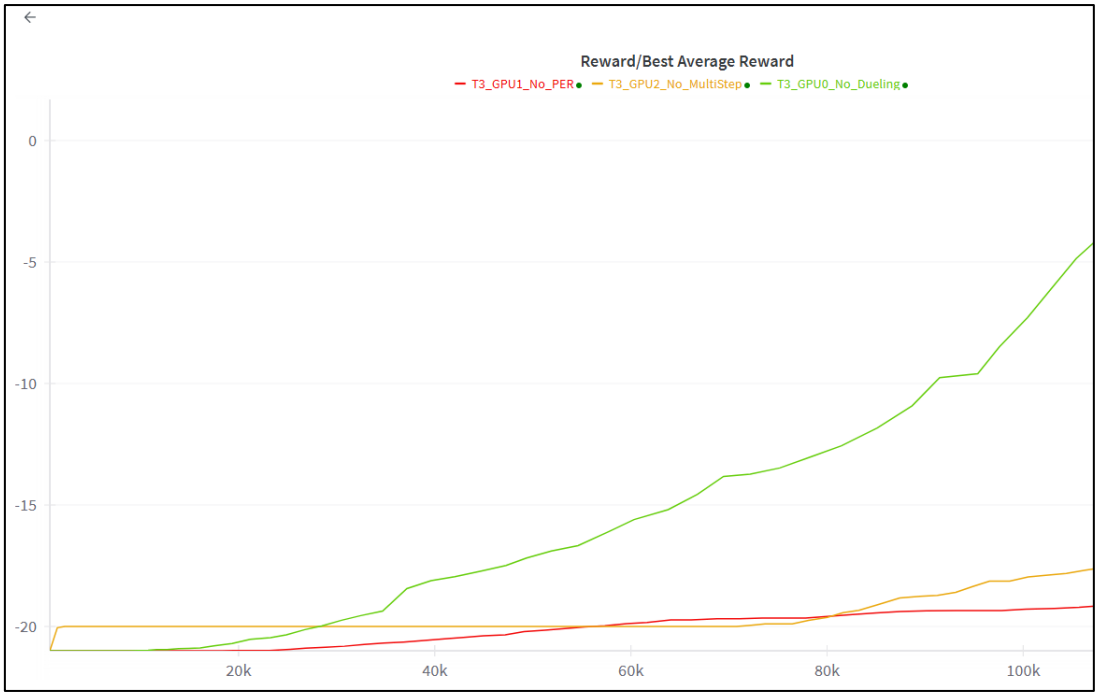


圖 4. Ablation study 圖表

3.4 Bonus：其他訓練策略分析

Dueling Network Architecture

傳統 DQN 中，網路直接預測每個動作對應的 Q-value，但這忽略了一個事實：狀態本身的價值與選擇哪個動作是可以分開學習的。因此，Dueling DQN 將 Q-value 分解為：

- 狀態價值函數 ：描述在某狀態下的整體價值。
- 動作優勢函數 ：評估在狀態下，採取動作 相較於其他動作的好壞。

這兩者結合成最終的 Q-value

Q-value 函數：

$$\boldsymbol{Q(s,a;w) = V(s;w) + A(s,a;w)}$$

為了解決「V 與 A 的唯一性問題」，實際實作使用如下公式聚合：

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

```python
def forward(self, x):
    """前向傳播, 計算 Q-value"""
    x = x / 255.0
    features = self.feature_extractor(x)
    value = self.value_stream(features)
    advantage = self.advantage_stream(features)
    q_values = value + (advantage - advantage.mean(dim=1, keepdim=True))
    return q_values
```

程式碼 9. Dueling Network 架構

Noisy Networks for Exploration（雜訊網路探索）

在 Task 3 中，我們採用了 Noisy Networks 來取代傳統的 ε-greedy 探索策略，主要目的是提升 agent 在訓練過程中的探索效率與學習表現。

傳統的 ε-greedy 策略雖然簡單易實作，但存在缺點：首先，其探索方式為純隨機，缺乏方向性，因此在訓練初期可能會造成大量無效探索；其次，ε 值隨訓練進度逐漸減少，導致模型在後期幾乎不再探索，容易陷入區域最佳解；此外，ε 的初始值、最小值與衰減曲線需人工設計，難以針對不同任務進行最佳化。

將探索行為內建於神經網路中，用帶雜訊的參數取代 ε-greedy，讓模型在不同狀態中自動調整探索程度，實現更智慧的策略探索。

$$W = \mu_W + \sigma_W \odot \epsilon_W$$

$$b = \mu_b + \sigma_b \odot \epsilon_b$$

$\boldsymbol{\mu}$：可學的平均權重參數

$\boldsymbol{\sigma}$：可學的標準差參數（控制雜訊幅度）

ϵ：每次 forward 時重新採樣的隨機變數

$\odot$：代表元素相乘（Hadamard product）

```python
def __init__(self, in_features, out_features, sigma_init=0.5):
    super(NoisyLinear, self).__init__()
    self.in_features = in_features
    self.out_features = out_features
    self.sigma_init = sigma_init
    # 可學習的權重和偏差的平均值 (mu) 和標準差 (sigma)
    self.weight_mu = nn.Parameter(torch.Tensor(out_features, in_features))
    self.weight_sigma = nn.Parameter(torch.Tensor(out_features, in_features))
    self.bias_mu = nn.Parameter(torch.Tensor(out_features))
    self.bias_sigma = nn.Parameter(torch.Tensor(out_features))
    # 用於生成雜訊的非學習參數 (epsilon)
    self.register_buffer('weight_epsilon', torch.Tensor(out_features, in_features))
    self.register_buffer('bias_epsilon', torch.Tensor(out_features))
    self.reset_parameters()
    self.reset_noise()

def reset_parameters(self):
    """初始化 mu 和 sigma 參數"""
    mu_range = 1 / np.sqrt(self.in_features)
    self.weight_mu.data.uniform_(-mu_range, mu_range)
    self.weight_sigma.data.fill_(self.sigma_init / np.sqrt(self.in_features))
    self.bias_mu.data.uniform_(-mu_range, mu_range)
    self.bias_sigma.data.fill_(self.sigma_init / np.sqrt(self.out_features))

def reset_noise(self):
    """生成新的雜訊樣本"""
    epsilon_in = torch.randn(self.in_features, device=self.weight_mu.device)
    epsilon_out = torch.randn(self.out_features, device=self.weight_mu.device)
    self.weight_epsilon.copy_(torch.outer(epsilon_out, epsilon_in))
    self.bias_epsilon.copy_(epsilon_out)

def forward(self, x):
    """前向傳播。訓練時使用帶有雜訊的權重，評估時只使用平均值。"""
    if self.training:
        return nn.functional.linear(x, self.weight_mu + self.weight_sigma * self.weight_epsilon, self
    else:
        return nn.functional.linear(x, self.weight_mu, self.bias_mu)
```

程式碼 10. Noisy Network 架構

強化影像預處理（Enhanced Frame Preprocessing）

Atari 遊戲原始畫面 為彩色，解析度約為 210x160，包含大量不必要的資訊（例如：遊戲分數、邊框、背景細節），對於強化學習模型來說，會增加學習負擔與計算資源消耗，因此目的就是簡化輸入資訊，降低維度，讓模型專注於遊戲核心元素（如球、球拍等動態物件），提升學習效率與收斂速度。

```python
# --- Preprocessing ---
class AtariPreprocessor:
    """Atari 遊戲畫面預處理器 (參考 Jayin 的版本，穩定有效)"""
    def __init__(self, frame_stack=4):
        self.frame_stack = frame_stack
        self.frames = deque(maxlen=frame_stack)

    def preprocess(self, obs):
        """預處理單一畫面：灰階 -> 裁切 -> 縮放 -> 二值化"""
        gray = cv2.cvtColor(obs, cv2.COLOR_RGB2GRAY)
        cropped = gray[34:194, :] # 裁切掉分數等不重要區域
        resized = cv2.resize(cropped, (84, 84), interpolation=cv2.INTER_AREA)
        _, thresholded = cv2.threshold(resized, 127, 255, cv2.THRESH_BINARY) # 強調特徵
        return thresholded

    def reset(self, obs):
        """重置時用第一個畫面填滿 frame buffer"""
        frame = self.preprocess(obs)
        self.frames = deque([frame for _ in range(self.frame_stack)], maxlen=self.frame_stack)
        return np.stack(self.frames, axis=0)

    def step(self, obs):
        """將新畫面加入並回傳堆疊後結果"""
        frame = self.preprocess(obs)
        self.frames.append(frame)
        return np.stack(self.frames, axis=0)
```

程式碼 11. 對遊戲影像處理

Huber Loss（平滑損失函數）

傳統均方誤差（MSE）損失對於離群值（大誤差）非常敏感，會造成梯度爆炸，影響模型穩定訓練。

Huber Loss 結合了 MSE 和 MAE 的優點：

- 當誤差較小時，表現像 MSE（平方誤差），利於快速收斂。
- 當誤差較大時，表現像 MAE（線性誤差），降低梯度對大誤差的影響，防止訓練不穩定。

Huber Loss 的數學定義：

$$Loss\left(y,\hat{y}\right)=\begin{cases}\frac{1}{2}\left(y-\hat{y}\right)^2, & \left|y-\hat{y}\right|\leq\delta\\ \delta(\left|y-\hat{y}\right|-\frac{1}{2}\delta), & O.W.\end{cases}$$

```python
self.replay_buffer.update_priorities(indices, errors)
loss = (weights * nn.functional.smooth_l1_loss(current_q_values, target_q_values, reduction='none'))
```

程式碼 11. Huber Loss 架構

4. Instrcution

4.1 Train

Task1	python LAB5_314551010_Code/dqn_task1.py
Task2	python LAB5_314551010_Code/dqn_task2.py
Task3	python LAB5_314551010_Code/dqn_task3.py --gpu 0 --student-id 314551010 --save-dir . --noisy --lr 1e-4 --batch-size 64 --memory-size 500000 --replay-start-size 5000 --target-update-frequency 500 --train-frequency 4 --train-per-step 4 --multi-step 4 --per-alpha 0.6 --per-beta-start 0.4 --total-steps 2500000 --no-wandb

4.2 Test(產生測試結果，與保存影片)

Task1	python LAB5_314551010_Code/test_model_task12.py --task Task1 --model-path ./LAB5_314551010_task1.pt
Task2	python LAB5_314551010_Code/test_model_task12.py --task Task2 --model-path ./LAB5_314551010_task2.pt
Task3 (400k)	python LAB5_314551010_Code/test_model_task3.py --task Task3 --model-path ./LAB5_314551010_task3_400000.pt
Task3 (800k)	python LAB5_314551010_Code/test_model_task3.py --task Task3 --model-path ./LAB5_314551010_task3_800000.pt
Task3 (1.2M)	python LAB5_314551010_Code/test_model_task3.py --task Task3 --model-path ./LAB5_314551010_task3_1200000.pt
Task3 (1.6M)	python LAB5_314551010_Code/test_model_task3.py --task Task3 --model-path ./LAB5_314551010_task3_1600000.pt
Task3 (2M)	python LAB5_314551010_Code/test_model_task3.py --task Task3 --model-path ./LAB5_314551010_task3_2000000.pt

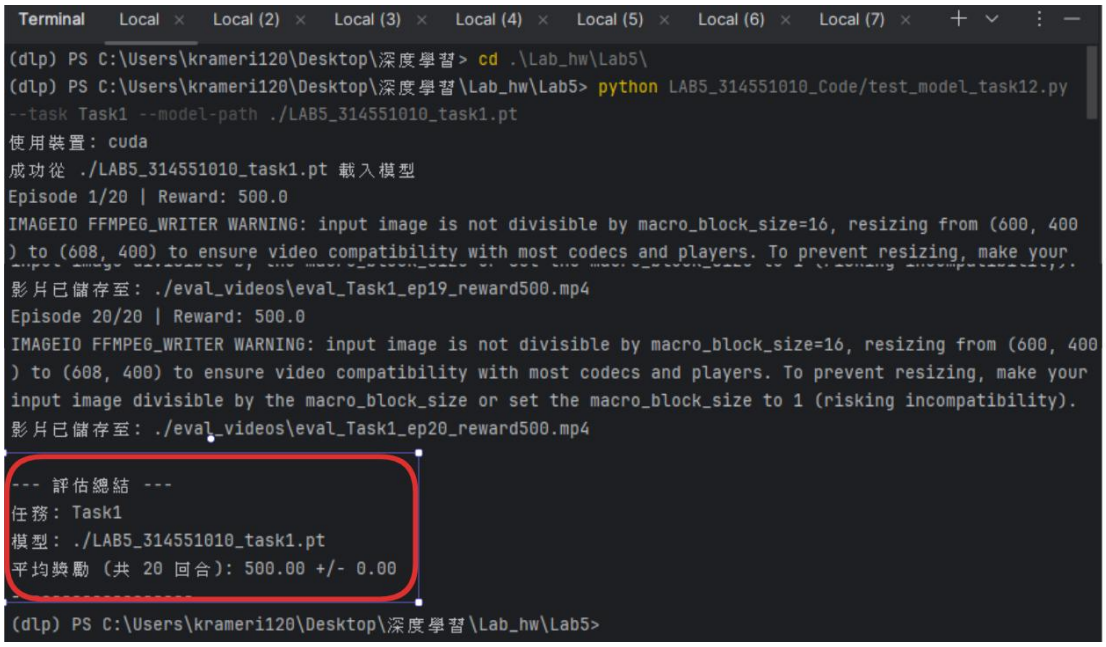
4.3 Test(針對 Task3 產生 seed 0~19 進行評估)

Task3 (400k)	python LAB5_314551010_Code/evaluate_task3.py --model_path ./LAB5_314551010_task3_400000.pt --noisy
Task3 (800k)	python LAB5_314551010_Code/evaluate_task3.py --model_path ./LAB5_314551010_task3_800000.pt --noisy
Task3 (1.2M)	python LAB5_314551010_Code/evaluate_task3.py --model_path ./LAB5_314551010_task3_1200000.pt --noisy
Task3 (1.6M)	python LAB5_314551010_Code/evaluate_task3.py --model_path ./LAB5_314551010_task3_1600000.pt --noisy
Task3 (2M)	python LAB5_314551010_Code/evaluate_task3.py --model_path ./LAB5_314551010_task3_2000000.pt --noisy

4.4 Test model

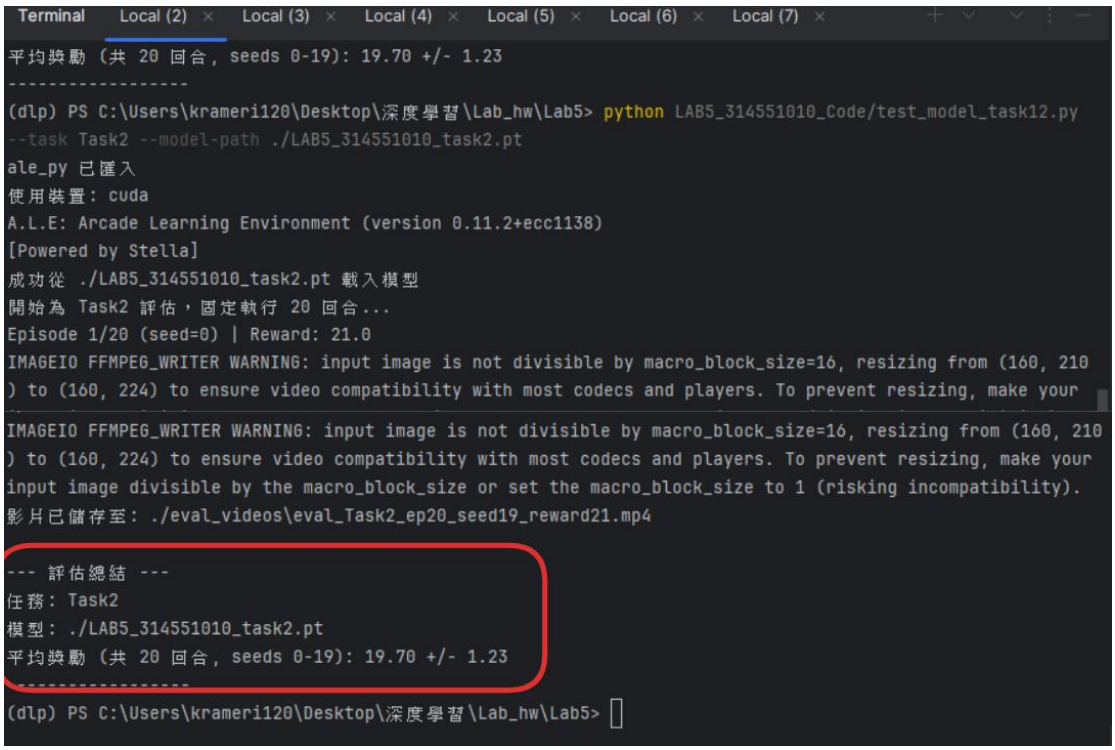
Task1

平均獎勵 (共 20 回合): 500.00 +/- 0.00



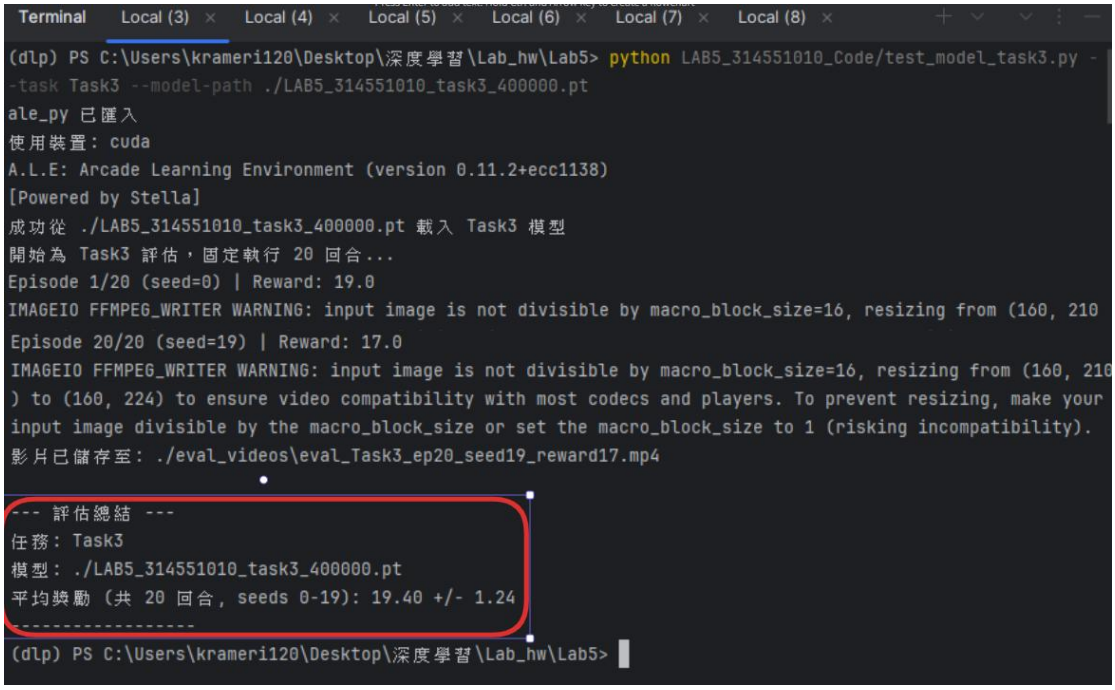
Task2

平均獎勵 (共 20 回合, seeds 0-19): 19.70 +/- 1.23



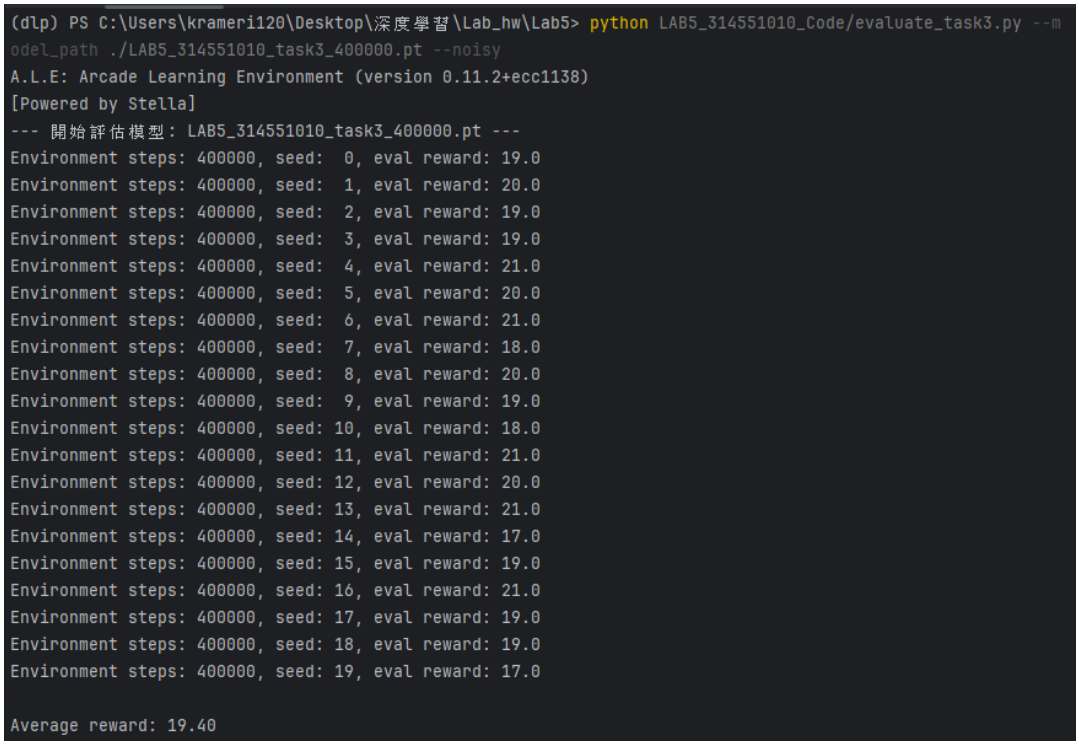
Task3(用產生影片的指令)

平均獎勵 (共 20 回合, seeds 0-19): 19.40 +/- 1.24



Task3(用評分指令)

400K (19.4 分)



800K(18.4 分)

```
Terminal  Local (4) ×  Local (5) ×  Local (6) ×  Local (7) ×  Local (8) ×
(dlp) PS C:\Users\krameri120\Desktop\深度學習\Lab_hw\Lab5> python LAB5_314551010_Code/evaluate_task3.py --m
odel_path ./LAB5_314551010_task3_800000.pt --noisy
A.L.E: Arcade Learning Environment (version 0.11.2+ecc1138)
[Powered by Stella]
--- 開始評估模型: LAB5_314551010_task3_800000.pt ---
Environment steps: 800000, seed:  0, eval reward: 19.0
Environment steps: 800000, seed:  1, eval reward: 17.0
Environment steps: 800000, seed:  2, eval reward: 21.0
Environment steps: 800000, seed:  3, eval reward: 20.0
Environment steps: 800000, seed:  4, eval reward: 17.0
Environment steps: 800000, seed:  5, eval reward: 20.0
Environment steps: 800000, seed:  6, eval reward: 19.0
Environment steps: 800000, seed:  7, eval reward: 21.0
Environment steps: 800000, seed:  8, eval reward: 17.0
Environment steps: 800000, seed:  9, eval reward: 18.0
Environment steps: 800000, seed: 10, eval reward: 19.0
Environment steps: 800000, seed: 11, eval reward: 21.0
Environment steps: 800000, seed: 12, eval reward: 17.0
Environment steps: 800000, seed: 13, eval reward: 17.0
Environment steps: 800000, seed: 14, eval reward: 20.0
Environment steps: 800000, seed: 15, eval reward: 18.0
Environment steps: 800000, seed: 16, eval reward: 16.0
Environment steps: 800000, seed: 17, eval reward: 16.0
Environment steps: 800000, seed: 18, eval reward: 14.0
Environment steps: 800000, seed: 19, eval reward: 21.0

Average reward: 18.40
```

1.2M(19.7 分)

```
Terminal  Local (4) ×  Local (5) ×  Local (6) ×  Local (7) ×  Local (8) ×
(dlp) PS C:\Users\krameri120\Desktop\深度學習\Lab_hw\Lab5> python LAB5_314551010_Code/evaluate_task3.py --m
odel_path ./LAB5_314551010_task3_1200000.pt --noisy
A.L.E: Arcade Learning Environment (version 0.11.2+ecc1138)
[Powered by Stella]
--- 開始評估模型: LAB5_314551010_task3_1200000.pt ---
Environment steps: 1200000, seed:  0, eval reward: 18.0
Environment steps: 1200000, seed:  1, eval reward: 17.0
Environment steps: 1200000, seed:  2, eval reward: 20.0
Environment steps: 1200000, seed:  3, eval reward: 20.0
Environment steps: 1200000, seed:  4, eval reward: 21.0
Environment steps: 1200000, seed:  5, eval reward: 17.0
Environment steps: 1200000, seed:  6, eval reward: 18.0
Environment steps: 1200000, seed:  7, eval reward: 20.0
Environment steps: 1200000, seed:  8, eval reward: 20.0
Environment steps: 1200000, seed:  9, eval reward: 21.0
Environment steps: 1200000, seed: 10, eval reward: 20.0
Environment steps: 1200000, seed: 11, eval reward: 21.0
Environment steps: 1200000, seed: 12, eval reward: 20.0
Environment steps: 1200000, seed: 13, eval reward: 21.0
Environment steps: 1200000, seed: 14, eval reward: 19.0
Environment steps: 1200000, seed: 15, eval reward: 20.0
Environment steps: 1200000, seed: 16, eval reward: 21.0
Environment steps: 1200000, seed: 17, eval reward: 20.0
Environment steps: 1200000, seed: 18, eval reward: 19.0
Environment steps: 1200000, seed: 19, eval reward: 21.0

Average reward: 19.70
```

1.6M(20.4 分)

```
Terminal  Local (4)  Local (5)  Local (6)  Local (7)  Local (8)
(dlp) PS C:\Users\krameri120\Desktop\深度學習\Lab_hw\Lab5> python LAB5_314551010_Code/evaluate_task3.py --m
odel_path ./LAB5_314551010_task3_1600000.pt --noisy
A.L.E: Arcade Learning Environment (version 0.11.2+ecc1138)
[Powered by Stella]
--- 開始評估模型: LAB5_314551010_task3_1600000.pt ---
Environment steps: 1600000, seed:  0, eval reward: 21.0
Environment steps: 1600000, seed:  1, eval reward: 21.0
Environment steps: 1600000, seed:  2, eval reward: 21.0
Environment steps: 1600000, seed:  3, eval reward: 21.0
Environment steps: 1600000, seed:  4, eval reward: 21.0
Environment steps: 1600000, seed:  5, eval reward: 21.0
Environment steps: 1600000, seed:  6, eval reward: 21.0
Environment steps: 1600000, seed:  7, eval reward: 21.0
Environment steps: 1600000, seed:  8, eval reward: 21.0
Environment steps: 1600000, seed:  9, eval reward: 21.0
Environment steps: 1600000, seed: 10, eval reward: 20.0
Environment steps: 1600000, seed: 11, eval reward: 20.0
Environment steps: 1600000, seed: 12, eval reward: 20.0
Environment steps: 1600000, seed: 13, eval reward: 19.0
Environment steps: 1600000, seed: 14, eval reward: 20.0
Environment steps: 1600000, seed: 15, eval reward: 21.0
Environment steps: 1600000, seed: 16, eval reward: 20.0
Environment steps: 1600000, seed: 17, eval reward: 20.0
Environment steps: 1600000, seed: 18, eval reward: 20.0
Environment steps: 1600000, seed: 19, eval reward: 18.0

Average reward: 20.40
```

2M(19.75 分)

```
(dlp) PS C:\Users\krameri120\Desktop\深度學習\Lab_hw\Lab5> python LAB5_314551010_Code/evaluate_task3.py --m
odel_path ./LAB5_314551010_task3_2000000.pt --noisy
A.L.E: Arcade Learning Environment (version 0.11.2+ecc1138)
[Powered by Stella]
--- 開始評估模型: LAB5_314551010_task3_2000000.pt ---
Environment steps: 2000000, seed:  0, eval reward: 19.0
Environment steps: 2000000, seed:  1, eval reward: 17.0
Environment steps: 2000000, seed:  2, eval reward: 21.0
Environment steps: 2000000, seed:  3, eval reward: 20.0
Environment steps: 2000000, seed:  4, eval reward: 21.0
Environment steps: 2000000, seed:  7, eval reward: 20.0
Environment steps: 2000000, seed:  8, eval reward: 20.0
Environment steps: 2000000, seed:  9, eval reward: 21.0
Environment steps: 2000000, seed: 10, eval reward: 21.0
Environment steps: 2000000, seed: 11, eval reward: 21.0
Environment steps: 2000000, seed: 12, eval reward: 19.0
Environment steps: 2000000, seed: 13, eval reward: 17.0
Environment steps: 2000000, seed: 14, eval reward: 19.0
Environment steps: 2000000, seed: 15, eval reward: 19.0
Environment steps: 2000000, seed: 16, eval reward: 20.0
Environment steps: 2000000, seed: 17, eval reward: 20.0
Environment steps: 2000000, seed: 18, eval reward: 17.0
Environment steps: 2000000, seed: 19, eval reward: 21.0

Average reward: 19.75
```

4.5 Parameter

參數名稱	數值 / 設定	說明
batch_size	64	每次訓練所使用的樣本數
lr（learning_rate）	0.0001	學習率
memory_size	100,000	Replay Buffer 最大容量
multi_step	4	N-step TD 訓練中的 n 值
noisy	TRUE	是否啟用 Noisy Networks 作為探索策略
per_alpha	0.6	Prioritized Replay 中的 α 值，決定 TD error 的重要性
per_beta_start	0.4	Prioritized Replay 中的初始 β 值，用於權重修正
replay_start_size	5,000	開始訓練前，Replay Buffer 至少需要儲存多少筆資料
target_update_frequency	500	每隔多少步同步 target network
total_steps	3,000,000	總訓練步數
train_frequency	4	每幾步進行一次訓練
train_per_step	4	每次訓練幾個 batch（為加速訓練）

5. 參考資料

[1] J. Yin, “NYCU Deep Learning Course Code,” GitHub Repository, [Online]. Available: https://github.com/jayin92/NYCU-deep-learning

[2] alu98753, “NYCU Deep Learning 2025,” GitHub Repository, [Online]. Available: https://github.com/alu98753/NYCU-Deep-Learning-2025

[3] Part-time-Ray, “DLP: Deep Learning Practice,” GitHub Repository, [Online]. Available: https://github.com/Part-time-Ray/DLP

[4] c1uc, “2025 Spring Deep Learning Labs,” GitHub Repository, [Online]. Available: https://github.com/c1uc/2025_Spring_Deep-Learning-Labs

[4] 微信用戶_37958272，〈深入理解 DQN 演算法〉，《CSDN 部落格》，網址：https://blog.csdn.net/weixin_37958272/article/details/121882249

[5] Anyscale，〈使用深度 Q 網路進行強化學習〉，《Anyscale 官方部落格》，網址：https://www.anyscale.com/blog/reinforcement-learning-with-deep-q-networks

[6] 微信用戶_46133643，〈DQN 深度強化學習實作詳解〉，《CSDN 部落格》，

網址：https://blog.csdn.net/weixin_46133643/article/details/121863216

[7] LENG_Lingliang，〈強化學習：從 DQN 到進階演算法〉，《CSDN 部落格》，網址：https://blog.csdn.net/LENG_Lingliang/article/details/136241424

[8] hehedadaq，〈DQN 演算法解析與實作〉，《CSDN 部落格》，網址：https://blog.csdn.net/hehedadaq/article/details/100127962

[9] qq_45889056，〈從零開始實現 DQN 深度強化學習〉，《CSDN 部落格》，網址：https://blog.csdn.net/qq_45889056/article/details/130621187

[10] qq_40206371，〈DQN 演算法與實踐技巧總結〉，《CSDN 部落格》，網址：https://blog.csdn.net/qq_40206371/article/details/124994956

[11] qq_40206371，〈DQN 改進技巧實作細節〉，《CSDN 部落格》，網址：https://blog.csdn.net/qq_40206371/article/details/124993882

[12] 〈深度強化學習 DQN 模型分析〉，《知乎專欄》，網址：https://zhuanlan.zhihu.com/p/138504673

[13] 〈淺談強化學習與 DQN 應用〉，《知乎專欄》，網址：https://zhuanlan.zhihu.com/p/554735911

使用的 AI 工具：

✧ **ChatGPT-4o**（由 OpenAI 提供）

用於問答、報告草稿撰寫、英文轉中文及論文說明。

✧ **Google AI Studio**

作為補充問答工具，用於模型概念、架構或函式撰寫思路輔助。

✧ **VSCode + Google Gemini 2.5 Pro**

協助撰寫與補全 Python 程式碼，並支援簡易 debug。

✧ **VSCode GitHub Copilot（GPT-4o）**

用於實作過程中進行即時程式補全、語法建議與除錯輔助。