113學年度上學期強化學習DQN期末專題

201 05吳苡柔、17郭倢妤、21黃品薰

壹、問題敘述：

- **選用環境：**
  gymnasium CarRacing-v3
- **遊戲目標：**
  在灰色賽道格子上開車並避免超出車道。
- **狀態空間：**
  - **Box(0, 255, (96, 96, 3), uint8)**
  - Box ：連續範圍的值。
  - 0 是觀察空間的下限，255 是觀察空間的上限。
  - (96, 96, 3) 表示觀察的狀態是一個三維的影像，寬高為 96 像素，並且有三個顏色通道 (RGB) 。
  - uint8 表示觀察的資料型態是 8 位元的無符號整數。
- **動作空間：**
  - **Box([-1. 0. 0.], 1.0, (3,), float32)**
  - [−1.0, 0.0, 0.0] 是動作空間的下限，表示可以採取的動作的最小值。
  - 1.0 是動作空間的上限，表示可以採取的動作的最大值。
  - (3,) 表示動作是三維的，每個維度的範圍是從 −1.0 到 1.0 之間。
  - float32 表示動作的資料型態是 32 位元的浮點數。

連續動作空間	0：方向盤，−1 往左最大值，+1往右最大值 1： 加油門 2： 煞車
離散動作空間	0： 啥也不幹 1： 左轉 2： 右轉 3： 油門 4： 煞車

- **獎勵機制：**
  每一幀的獎勵是 −0.1，每踩一塊賽道格子，獎勵是 +1000/N，其中 N 是賽道上總共踩的格子數。
- **完成任務條件或死亡條件：**
  當 Agent 經過所有賽道格子就會結束遊戲 ，車子也可以偏離軌道，但會被扣100分並死亡。
- **可設定的環境參數：**
  - lap_complete_percent=y：必須訪問的格子百分比，以確定一圈賽道是否完成。

在這裡設置為 y，代表需要經過 100y% 的格子才算完成一圈。
- domain_randomize=False：這禁用了環境的隨機變異版本。在啟用隨機變異時，每次重設後的背景和賽道顏色會不同。
- continuous=False：這將環境轉換為使用離散動作空間。在離散動作空間中，有五個動作：[不採取動作, 左轉, 右轉, 加速, 煞車]。

貳、DQN原理介紹：

- **探索與利用**

小時候不知道以什麼方法在領紅包時能拿最多錢，經過幾年探索，以立正、躺著、翹二郎腿、盤腿、倒立、蹲著、跪著、劈腿、下腰、後空翻.....不同姿勢領紅包後，發現以土下座的姿勢最容易拿到 10000 元，因此長大後可能會較常利用土下座姿勢領紅包，這便是探索與利用的生活化例子。在深度 Q 網路（DQN）中，**探索（Exploration）與利用（Exploitation）**是非常重要的概念，用來平衡在學習過程中探索新策略與利用已知策略之間的關係，特別是在強化學習中面對未知環境或複雜任務時尤其關鍵。常用的演算法為 ε−greedy，它根據機率使智能體選擇目前已學到的最優動作（即利用當前知識）或進行探索（隨機採取動作）。這種策略透過平衡探索和利用，旨在在學習過程中逐步減少探索率，從而轉向使用最優策略。即使是學習的經驗也可以被重新利用，因此也能透過經驗回放（Experience Replay）以隨機抽樣過往學習資料來提供探索，以在不同的狀態下重新探索可能的動作。簡單來說，探索與利用就是一個避免原地踏步的機制。

- **DQN**

DQN 全稱是 Deep Q Network，是一種基於 Q−learning 再變化的深度學習＋強化學習的結合，由 DeepMind 研發，並且可以解決傳統 Q 學習在處理高維、複雜環境時的困難。本質上是運行 Q−Table，再反覆挑出最大 Q 值，但與傳統 Q−Table 不同的是，DQN 通過使用深度神經網絡來近似 Q 函數，可以解決 Q−Table 在狀態空間非常大或無法離散化時遇到的問題。

- **Replay Buffer**

Replay Buffer 的主要作用是儲存先前的經驗數據，以便後續的訓練可以隨機抽樣這些數據進行學習，提升樣本利用率，而不是僅僅依賴當前的經驗。儲存的資料包含狀態（state）、動作（action）、獎勵（reward）、下一個狀態（next state）等。另外，Replay Buffer 的其他功能包含打亂樣本關聯性。當連續的訓練資料有高度相關時，使用回放資料能讓訓練時不會陷入局部最優或導致過度擬合，就像某學生連續十天請假在家複習，前五天都讀化學，後五天都讀數學，考試時很可能會忘了大部分前五天讀的化學再說什麼，為了避免神經網路也出現類似情況而在 DQN 中加入 Replay Buffer 後，便能有效緩解高估。

- **fixed Q Target**

fixed Q Target 是為了避免使用同一個 Q 網路來決策目前以及目標，避免訓練不穩定或是過度估計。在 DQN 中，通常會設定兩個神經網路：目前 Q 網路（用於選擇動作和計算目前狀態下的 Q 值）與目標 Q 網路（用於計算目標 Q 值，會週期性更新網路參數），以使得目標 Q 值相對穩定。

- **Bootstrapping**

Bootstrapping，中文翻譯為自舉，在強化學習中的意思是用估計去更新同類的估計。具體來說，Bootstrapping 是結合已有的獎勵與未來的獎勵，以更新現在的獎勵，並且能夠在沒有先驗證假設分佈的情況下，透過反覆抽樣和重複計算來對統計量進行估計，是一種強大的統計工具。舉生活化的例子，假設你現在準備期末考了，因為時間很趕，沒辦法做太多事，很猶豫到底要先讀物理第五章、第六章還是做期末專題。一開始發現自己第五章小考成績較不理想，決定先複習第五章，但後來學姊告訴你每個選項對物理分數的重要性，分別是第五

章會有40分，第六章會有60分，做專題0分。同時你又發現第五章的內容可能會延伸到第六章，所以你調整複習計畫，把時間多分給第五章，未來可能會取得較好的時間。像這樣重新評估每件事的分數權重，這過程就像Bootstrapping，結合過去(學姊經驗)、當下(小考分數)、未來(期末考分數)，反覆更新當前的策略(複習計畫)。

- **Double DQN**

  Double DQN（Double Deep Q-Network）是對經典的 DQN 演算法的改進，旨在解決傳統 DQN 中存在的過度估計問題。在傳統的 Q-learning 中，用於評估下一個狀態的最大動作值時使用的 Q 函數與選擇動作時使用的 Q 函數是同一個，可能導致對動作價值的高估。 **Double DQN** 引入了兩個獨立的 Q 函數（Q-networks），分別用於選擇動作和評估動作值，透過這種方式減少了過度估計的發生，提高學習的穩定性跟效率。

- **Dueling DQN**

  Dueling DQN 是一種透過重新設計 Q 網路的結構來提升效能的方法。Dueling DQN 將 Q 函數分解為狀態值函數（State Value Function）和優勢函數（Advantage Function）。狀態價值函數估計狀態的基本價值，而優勢函數則估計每個動作相對於其他動作的優劣程度，也就是比較成本的概念。用種田概念來類推就是，原本只會預期收穫為多少，要施肥嗎？但是 Dueling DQN 可以學習狀態以及比較利益，比如在豐雨年，不論種法，可以預期有不錯的收成，也可以說此時狀態價值函數較高；而在蝗災年，就可以預期遭殃，不論種法如何，此時狀態價值函數較低。優勢函數則是比較分析動作間的相對可能造成的結果，比如說現在去睡覺而不是寫報告對於我的健康有更大的益處。一言以蔽之，優勢函數就是相對優秀解。

- **Nature DQN**

  Nature DQN 是最早亮相的 DQN，目的為解決複雜環境中的強化學習問題。它利用深度卷積神經網路（CNN）作為 Q 函數，能夠從像素層級的原始影像資料中學習狀態的表示和動作的價值。而 Nature DQN 和 DQN 最主要的差別，就是在 DQN 中，預測值和目標值是用同一個 Q 網路訓練，這樣兩者依賴性太強，不利收斂，舉生活化的例子，就像是你在考試前用這份考卷來訓練，在審核時又用這分考卷檢驗，這樣無法測出真正的實力。因此 Nature DQN 用兩個 Q 網路（PredictQ 和 TargetQ）來訓練，且 TargetQ 的參數不會迭代更新，而是隔一定時間從 PredictQ 複製過來。Nature DQN 在經典的 Atari 遊戲環境中展現了出色的效能，標誌著深度學習在強化學習領域的重要進展。

參、研究方法：

1. 敘述搭建的 DQN、Double DQN、Dueling DQN、Nature DQN 網路架構

DQN：

DQN最主要的特徵是用 Q 網路取代 Q-Table，由於我們所選環境是以影像呈現，因此我們搭建了一個用捲積神經網路（CNN）組成的 Q 網路。

網路架構

我們使用兩層隱藏層和兩層捲積層的捲積神經網路（CNN）及 ReLU 激活函數來搭建 Q 網路，以下為 Q 網路架構：

- **輸入層：**n_act，表示要預測價值的動作。

- **隱藏層：**
  - 第一層捲積層：輸出 16 個特徵圖，使用 8x8 的卷積核，步長為 4。
  - 第二層捲積層：輸出 32 個特徵圖，使用 4x4 的卷積核，步長為 2。
- **全連接層：**
  - 第一層：輸出 256 維的向量。
  - 第二層：輸出 n_act，即每個動作對應的 Q 值。

另外我們加入了經驗回放 (Replay Buffer) 及使用 Fixed Q Target 以緩減模型發生高估的狀況。

參數選擇

- 折扣因子 γ : 0.95
- 探索最低點 eps_low : 0.05
- 學習率 lr : 0.00025
- 訓練回合數 N_EPISODES : 1000

```
class DQN(torch.nn.Module):
    def __init__(self,n_act):
        super(DQN,self).__init__()
        self.conv1 = torch.nn.Conv2d(4, 16, kernel_size=8, stride=4)
        self.conv2 = torch.nn.Conv2d(16, 32, kernel_size=4, stride=2)
        self.fc1 = torch.nn.Linear(32 * 9 * 9, 256)
        self.fc2 = torch.nn.Linear(256, n_act)
    def forward(self,x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = x.view((-1, 32 * 9 * 9))
        x = self.fc1(x)
        x = self.fc2(x)
        return x
```

^ 圖一：DQN 神經網路架構

```
class ReplayBuffer:
    def __init__(self,max_size=int(1e5), num_steps=1):
        self.s = np.zeros((max_size,4,84,84), dtype=np.float32)
        self.a = np.zeros((max_size,), dtype=np.int64)
        self.r = np.zeros((max_size, 1), dtype=np.float32)
        self.s_ = np.zeros((max_size,4,84,84), dtype=np.float32)
        self.done = np.zeros((max_size, 1), dtype=np.float32)
        self.ptr = 0
        self.size = 0
        self.max_size = max_size
        self.num_steps = num_steps

    def append(self,s,a,r,s_,done):
        self.s[self.ptr] = s
        self.a[self.ptr] = a
        self.r[self.ptr] = r
        self.s_[self.ptr] = s_
        self.done[self.ptr] = done
        self.ptr = (self.ptr + 1) % self.max_size
        self.size = min(self.size+1,self.max_size)
    def sample(self, batch_size):
        ind = np.random.randint(0, self.size, batch_size)
        return torch.FloatTensor(self.s[ind]),torch.LongTensor(self.a[ind]),torch.FloatTensor(self.r[ind]),torch.FloatTensor(self.s_[ind]),torch.FloatTensor(self.done[ind])
```

^ 圖二：Replay Buffer

```
class DQNAgent():
    def __init__(self,gamma=0.9,eps_low=0.1,lr=0.00025):
        self.env = env
        self.n_act=self.env.action_space.n
        self.PredictDQN= DQN(self.n_act)
        self.TargetDQN= DQN(self.n_act)
        if Load_File>0:
            self.PredictDQN.load_state_dict(torch.load(Old_File))
            self.TargetDQN.load_state_dict(torch.load(Old_File))
        self.PredictDQN.to(device)
        self.TargetDQN.to(device)
        self.LossFun=torch.nn.SmoothL1Loss()
        self.optimizer=torch.optim.Adam(self.PredictDQN.parameters(),lr=lr)
        self.gamma=gamma
        self.eps_low=eps_low
        self.rb=ReplayBuffer(max_size=10000, num_steps=1)
    def PredictA(self,s):
        with torch.no_grad():
            return torch.argmax(self.PredictDQN(torch.FloatTensor(s).to(device))).item()
    def SelectA(self,a):
        return self.env.action_space.sample() if np.random.random()<self.EPS else a
```

^ 圖三：**DQN Agent** 的參數設定與選擇動作

```
def Train(self,N_EPISODES):
    for i in tqdm(range(Load_File,N_EPISODES)):
        self.EPS=self.eps_low+(1-self.eps_low)*math.exp(-i*5/(N_EPISODES))
        total_reward=0
        s,_=self.env.reset()
        while True:
            a=self.SelectA(self.PredictA(s))
            s_,r,done,stop,_=self.env.step(a)
            self.rb.append(s,a,r,s_,done)
            if self.rb.size > 200 and i%self.rb.num_steps==0:self.Learn()
            if i % 20==0:    self.TargetDQN.load_state_dict(self.PredictDQN.state_dict())
            s=s_
            total_reward+=r
            if done or stop:break
        # print(f"\n{total_reward}")
        Log["TrainReward"].append(total_reward)
        if i % 10 == 9:
            test_reward=self.Test()
            print(f"\n訓練次數{i+1}，總回報{test_reward}")
            Log["TestReward"].append(test_reward)
            torch.save(self.PredictDQN.state_dict(), f"Model-{i+1}.pt")
            np.save(f"Log-{i+1}.npy", Log)
def Learn(self):
    self.optimizer.zero_grad()
    batch_s, batch_a, batch_r, batch_s_, batch_done=self.rb.sample(32)
    predict_Q = (self.PredictDQN(batch_s.to(device))*F.one_hot(batch_a.long().to(device),self.n_act)).sum(1,keepdims=True)
    with torch.no_grad():
        target_Q = batch_r.to(device)+(1-batch_done.to(device))*self.gamma*self.TargetDQN(batch_s_.to(device)).max(1,keepdims=True)[0]
    loss = self.LossFun(predict_Q, target_Q)
    Log["Loss"].append(float(loss))
    loss.backward()
    self.optimizer.step()
```

^ 圖四：**DQN Agent** 的訓練及學習

```
def Test(self,VIDEO=False):
    total_reward=0
    video=[]
    s,_=self.env.reset()
    while True:
        video.append(self.env.render())
        a=self.PredictA(s)
        s,r,done,stop,_=self.env.step(a)
        total_reward+=r
        if done or stop:break
    if VIDEO:
        patch = plt.imshow(video[0]) #產生展示圖形物件
        plt.axis('off') #關閉坐標軸
        def animate(i): #設定更換影格的函數
            patch.set_data(video[i])
            #plt.gcf()=>建新繪圖區 animate=>更換影格函數 frames=>影格數 interval=>影隔間距(毫秒)
        anim = animation.FuncAnimation(plt.gcf(),animate,frames=len(video),interval=200)
        anim.save('Car_Racing.mp4') #儲存為mp4檔
    return total_reward
```

^ 圖五：DQN Agent 的測試

Double DQN：

Double DQN是在DQN的基礎上延伸改進的演算法，目的是為了緩解 DQN 容易出現高估問題。Double DQN 將動作選擇及價值評估分成兩個網路，分別是預測網路（Predict Q）及目標網路（Target Q），但兩者的網路架構都和上述 DQN 的 Q 網路架構一模一樣。

在每次訓練中會先讓 Predict Q 選擇動作並將數據存入 Replay Buffer，並使用 Target Q計算 Q 值以更新 Predict Q，每隔 20 次同步 Target Q 的參數。

```
def Learn(self):
    self.optimizer.zero_grad()
    batch_s, batch_a, batch_r, batch_s_, batch_done=self.rb.sample(32)
    predict_Q = (self.PredictDQN(batch_s.to(device))*F.one_hot(batch_a.long().to(device),self.n_act)).sum(1,keepdims=True)
    with torch.no_grad():
        # DoubleDQN 與 DQN 差在以下兩行，將動作選擇與價值評估分開
        a_ = self.PredictDQN(batch_s_.to(device)).max(dim=1)[1]
        target_Q = batch_r.to(device)+(1-batch_done.to(device))*self.gamma*(self.TargetDQN(batch_s_.to(device))*F.one_hot(a_.long(),self.n_act)).sum(1)

    loss = self.LossFun(predict_Q, target_Q)
    Log["Loss"].append(float(loss))
    loss.backward()
    self.optimizer.step()
```

^ 圖六：DoubleDQN 與 DQN 的程式碼差異（將動作選擇與價值評估分開）

Dueling DQN：

Dueling DQN 是在 DQN 的基礎上延伸改進的演算法，目的是為了解決 DQN 容易出現高估問題。和 DQN 不同的是 Dueling DQN 在計算 Q 值時，將 Q 網路分成兩個分支，分別是與狀態有關的狀態價函數 v(s)，及與動作有關的優勢函數 A(s,a)，最後將兩者相加即可獲得 Q 值。

```python
class DQN(torch.nn.Module):
    def __init__(self,n_act):
        super(DQN,self).__init__()
        self.conv1 = torch.nn.Conv2d(4, 16, kernel_size=8, stride=4)    #[N,4,84,84]->[N,16,20,20]
        self.conv2 = torch.nn.Conv2d(16, 32, kernel_size=4, stride=2)    #[N,16,20,20]->[N,32,9,9]
        self.fc1 = torch.nn.Linear(32 * 9 * 9, 256)

        # 分兩路
        self.value1 = torch.nn.Linear(256, 128)
        self.value2 = torch.nn.Linear(128, 1)

        self.adv1 = torch.nn.Linear(256 , 64)
        self.adv2 = torch.nn.Linear(64, n_act)
    def forward(self,x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = x.view((-1, 32 * 9 * 9))
        x = F.relu(self.fc1(x))
        value = self.value2(F.relu(self.value1(x)))
        adv = self.adv2(F.relu(self.adv1(x)))
        return value + ( adv - adv.mean(dim=-1, keepdim=True))
```

^ 圖七：DuelingDQN 與 DQN 的程式碼差異（改變神經網路模型架構，分成狀態函數與價值函數兩路去運算）

Nature DQN：

Nature DQN 是由 DeepMind 團隊提出的經典深度強化學習算法，發表於 2015 年的《Human-level control through deep reinforcement learning》。大致上皆與 Double DQN 相同，不同的是 Nature DQN 更注重處理高維度輸入數據（如遊戲畫面）。

網路架構

Nature DQN 的網路架構大致上與 Double DQN 相同，不同的是 Double DQN 使用的損失函數是 SmoothL1Loss()，但 Nature DQN 使用的是 MSELoss()（見圖九），此外 Q 網路的架構也仿照了論文中提及的架構，此網路由三層隱藏層及兩層全連接層組成（見圖八）。以下為 Nature DQN 架構：

- **輸入層：**
  - n_act，表示要預測價值的動作。
- **隱藏層：**
  - 第一層捲積層：32 個特徵圖，8x8 卷積核，步長為 4。
  - 第二層捲積層：64 個特徵圖，4x4 卷積核，步長為 2。
  - 第三層捲積層：64 個特徵圖，3x3 卷積核，步長為 1。
- **全連接層：**
  - 第一層：輸出 512 維向量。
  - 第二層：n_act，對應每個動作的 Q 值。

搭建DQN神經網路的類別

```python
class DQN(torch.nn.Module):
    def __init__(self,n_act):
        super(DQN,self).__init__()
        self.conv1 = torch.nn.Conv2d(4, 32, kernel_size=8, stride=4)    #[N,4,84,84]->[N,32,20,20]
        self.conv2 = torch.nn.Conv2d(32, 64, kernel_size=4, stride=2)    #[N,32,20,20]->[N,64,9,9]
        self.conv3 = torch.nn.Conv2d(64, 64, kernel_size=3, stride=1)    #[N,64,9,9]->[N,64,7,7]
        self.fc1 = torch.nn.Linear(64 * 7 * 7, 512)
        self.fc2 = torch.nn.Linear(512, n_act)
    def forward(self,x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = F.relu(self.conv3(x))
        x = x.view((-1, 64 * 7 * 7))
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

^ 圖八：Nature DQN 與 DQN 的經網路架構不同
^ 圖九：Nature DQN 的損失函數改用 MSELoss()

```
class DQNAgent():
    def __init__(self, gamma=0.9, eps_low=0.1, lr=0.001):
        self.env=env
        self.n_act=self.env.action_space.n
        self.PredictDQN= DQN(self.n_act)
        self.TargetDQN= DQN(self.n_act)
        if Load_File>0:
            self.PredictDQN.load_state_dict(torch.load(Old_File))
            self.TargetDQN.load_state_dict(torch.load(Old_File))
        self.PredictDQN.to(device)
        self.TargetDQN.to(device)
        self.LossFun=torch.nn.MSELoss()      # Nature DQN 的損失函數改用 MSELoss()
        self.optimizer=torch.optim.Adam(self.PredictDQN.parameters(),lr=lr)
        self.gamma=gamma
        self.eps_low=eps_low
        self.rb=ReplayBuffer(max_size=2000, num_steps=4)
```

2. 探索率衰減快慢對 DQN 訓練的比較

我們利用 研究方法1 的 DQN 模型，將折扣因子 γ 固定為 0.95 、 探索最低點 eps_low 固定為 0.05、學習率 lr 固定為 0.00025、訓練次數 N_EPISODES 固定為 5000，並且將下方計算探索率程式的 x 分別以 12、9、7、6、5.5 五個值作為替代。訓練完模型後畫出不同衰減率的訓練分數和訓練回合的關係圖並分析結果。

計算探索率：`self.EPS=self.eps_low+(1-self.eps_low)*math.exp(-i*x/n_episodes)`

3. DQN、Double DQN、Dueling DQN、Nature DQN訓練的比較

訓練模型時，我們將折扣因子 γ 固定為 0.95 、 探索最低點 eps_low 固定為 0.05、學習率 lr 固定為 0.00025、訓練次數 N_EPISODES 固定為 1000，並使用四種在 研究方法1 介紹的不同 DQN 演算法，執行 Car Racing 環境並訓練，最後畫出不同 DQN 的訓練分數和訓練回合關係圖並分析結果。

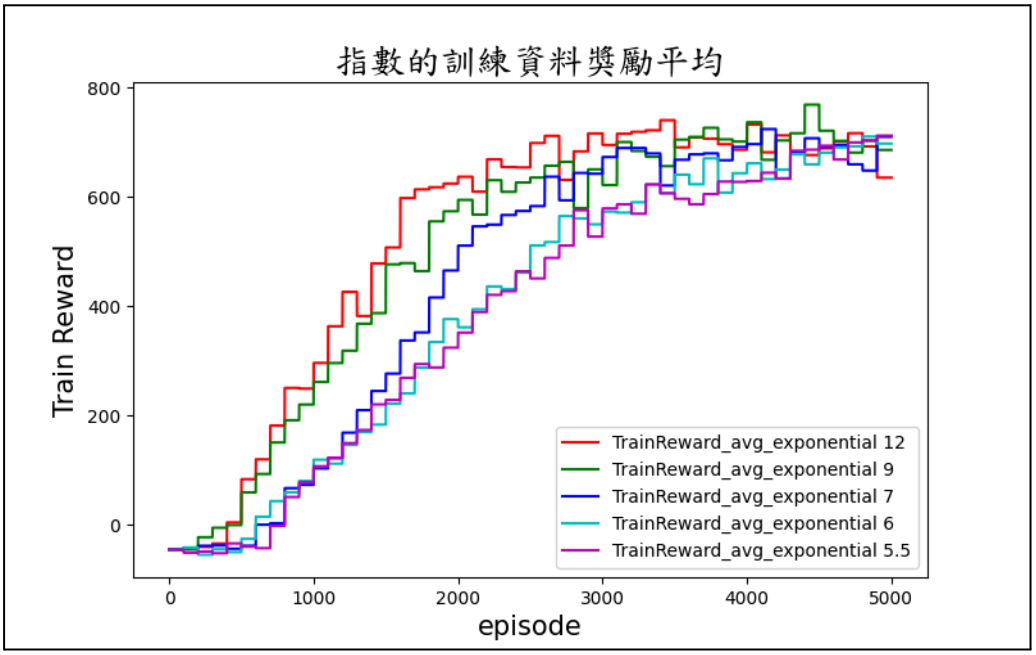
4. DQN、Double DQN、Dueling DQN、Nature DQN測試的比較

承接 研究方法3 的程式，我們每十次訓練就會進行一次測試，最終進行 100 次測試，以檢測訓練模型的成效，並繪出四種 DQN 的測試獎勵折線圖與比較最高分獎勵、最低分獎勵、獎勵分數平均、獎勵標準差等四項參數的直方圖。

肆、研究結果與分析：

1、探索率衰減快慢對 DQN 訓練的比較

(劃出不同衰減率的訓練分數和訓練回合的關係圖並分析結果)



< 圖表一

- 由上圖表可看出指數越高，五種參數一開始分數上升的速度越快，但到後面的震盪也比較大。
- 以前 2000 次回合的訓練獎勵來看，當計算探索率算式中的參數 x 以 12 代入時，訓

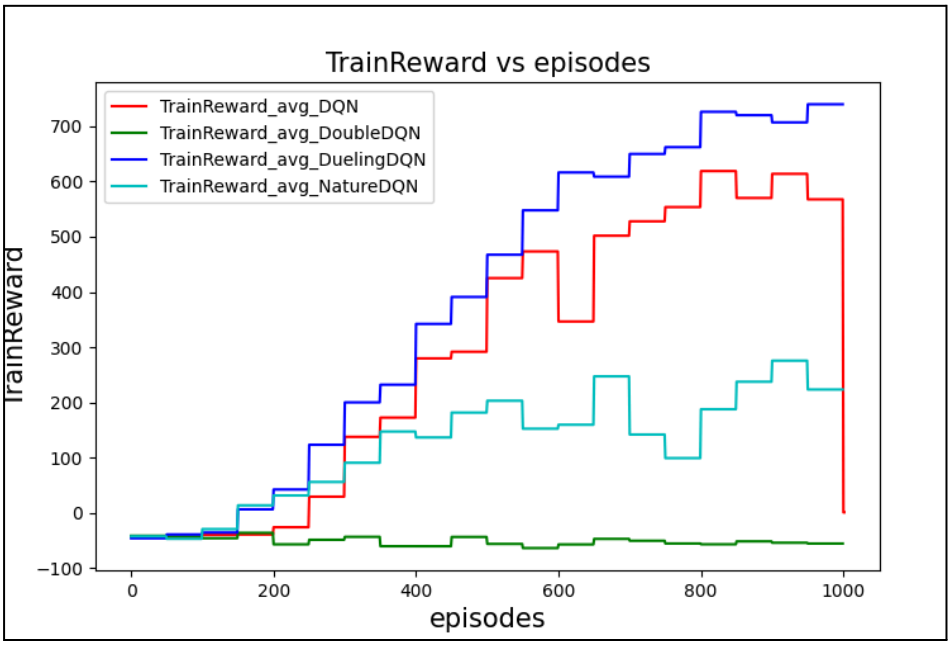
練獎勵成長速度最快，第二、第三快分別為代入 9、7，代入 6 和 5 的差距不明顯。

- 約訓練 4000 次後獎勵較接近，無明顯差距，顯示參數 x 對於前期學習速度影響較大，對後期訓練成果影響有限。

計算探索率：`self.EPS=self.eps_low+(1-self.eps_low)*math.exp(-i*x/n_episodes)`

2、DQN、Double DQN、Dueling DQN、Nature DQN訓練的比較

(劃出不同DQN的訓練分數和訓練回合的關係圖並分析結果)

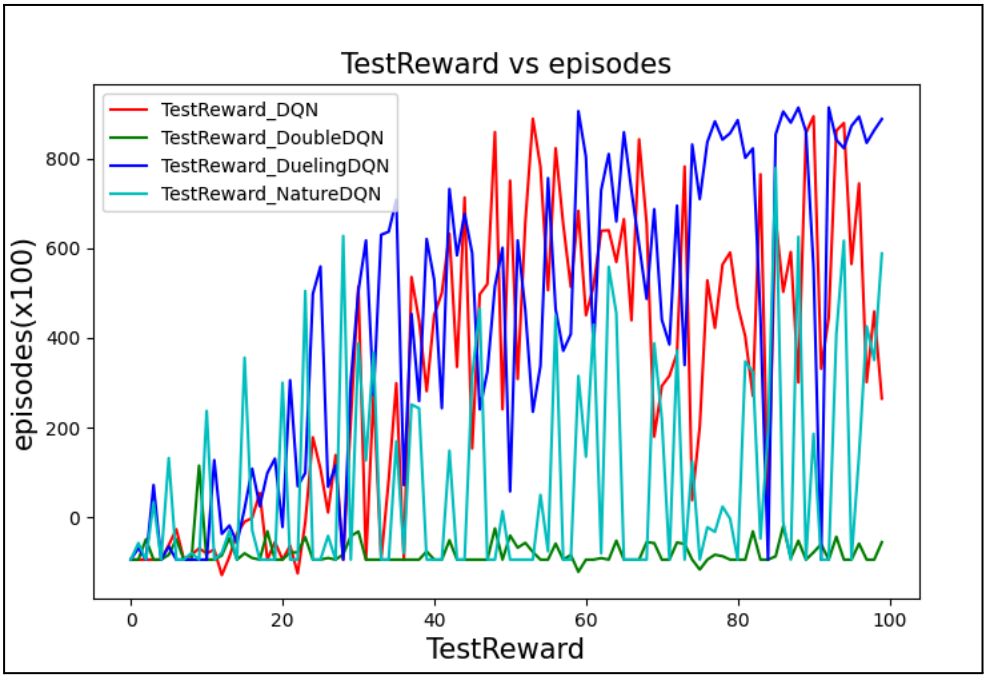


< 圖表二

- DuelingDQN 的效果最好，其次是 DQN，接著是 NatureDQN，最後一名則是 DoubleDQN。
- DQN 到 1000 回合就 overfitting。
- DoubleDQN 完全沒有成長，有可能是訓練回合不夠多，或是不適合處理 Car Racing 環境。

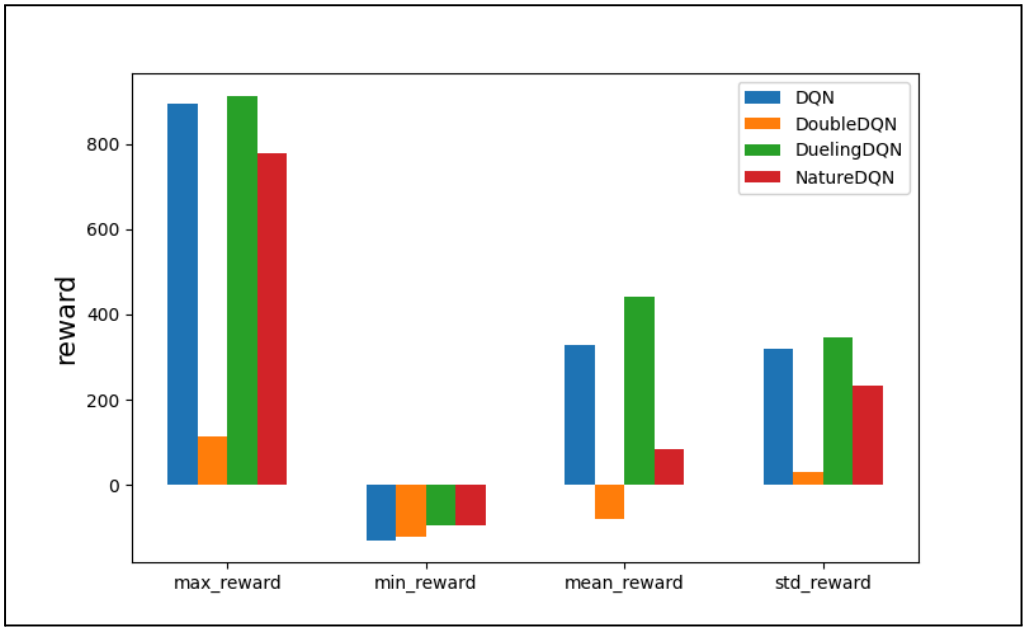
3、DQN、Double DQN、Dueling DQN、Nature DQN測試的比較

(劃出不同DQN測試100次的分數分布，比較最高分、最低分、平均分數、標準差)



< 圖表三

- DQN 與 DuelingDQN 在測試獎勵中效果最好，但 DuelingDQN 後期會出現明顯震盪。
- NatureDQN 效果排第三，但持續有震盪的情況。
- DoubleDQN 在 Car Racing 環境中表現最差，沒有明顯成長。



< 圖表四

```
max_reward  :  [894.9999999999907, 114.9644128113926, 914.62727272726, 779.9999999999861]
min_reward  :  [-129.65849056603784, -122.20780669144986, -94.99999999999903, -94.99999999999906]
mean_reward  :  [328.61881660097293, -79.922995885568, 441.5763023594238, 82.3036624094752]
std_reward  :  [319.68548055524593, 28.813822761920804, 346.494622932658, 232.31322193541786]
```

^ 上圖為圖表四直方圖的細部數據，第 0 項為 DQN，第 1 項為 DoubleDQN，第 2 項為 DuelingDQN，第 3 項為 NatureDQN

- 從「最高分獎勵」與「平均獎勵」的直方圖可以看出，DuelingDQN 表現最好（分數最高），其次為 DQN，再者是 NatureDQN，DoubleDQN 則表現最差（分數最低）。
- 「最低分獎勵」的直方圖中，最高至最低分的模型分別為：DuelingDQN、NatureDQN、DoubleDQN、DQN。
- 從「標準差」的直方圖中，可以看到標準差由低至高排序為：DoubleDQN、NatureDQN、DQN、DuelingDQN，這也顯示了資料由集中至疏散的程度。

伍、討論與心得：

1. 透過這次報告及了解相關研究，更具體了解 DQN 及其相關演算法運作原理與組成
2. 透過「調整探索衰減率對 DQN 訓練的比較」實驗，了解調整參數對 DQN 訓練會造成何種影響
3. 透過四種 DQN 演算法的訓練、測試的獎勵比較實驗，更明白各種 DQN 演算法的運作原理及其在 Car Racing 環境中的表現情形
4. 目前只學了最基礎的DQN，還有更多延伸DQN（例如:Prioritized Replay DQN、Rainbow DQN等等）值得我們去探討。

陸、組員與工作分配：

座號	姓名	負責工作
5	吳苡柔	訓練 NaturalDQN 模型、研究結果與分析、繪製圖表、研究方法、討論與

		心得
17	郭倢妤	撰寫問題敘述、DQN原理介紹
21	黃品薰	訓練 DQN、DoubleDQN、DuelingDQN 模型、研究結果與分析、研究方法、討論與心得

柒、參考資料

- Car Racing 環境：Oleg klimov. (n.d.). Car Racing. Gymnasium Documentation. https://gymnasium.farama.org/environments/box2d/car_racing/
- Replay Buffer 舉例參考：Mini-Batches in RL. (2018, December 20). Stackoverflow. https://stackoverflow.com/questions/53864434/mini-batches-in-rl
- Bootstrapping：

Amit Yadav. (2024, October 9). N-Step Bootstrapping in Reinforcement Learning. Medium. https://medium.com/@amit25173/n-step-bootstrapping-in-reinforcement-learning-e4f70f264933

- Nature DQN : https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
- DQN系列: https://www.cnblogs.com/jiangxinyang/p/10112381.html