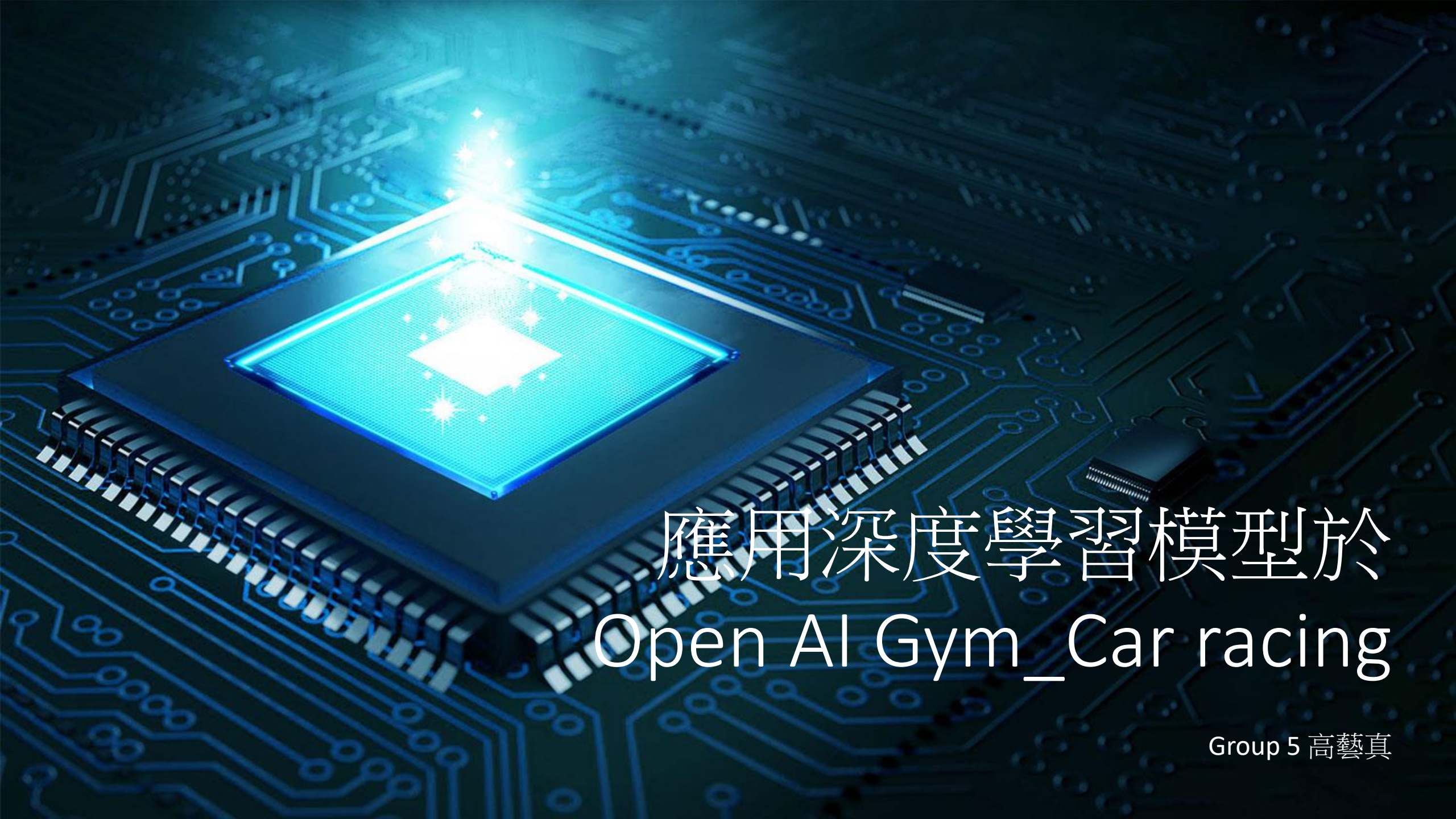


應用深度學習模型於 Open AI Gym_Car racing

Group 5 高藝真

Agenda Style

01 問題定義

02 基本介紹

Open AI gym, Car racing_v0, 資料預處理

03 模型介紹

CNN, Q-learning

04 結果驗證及總結

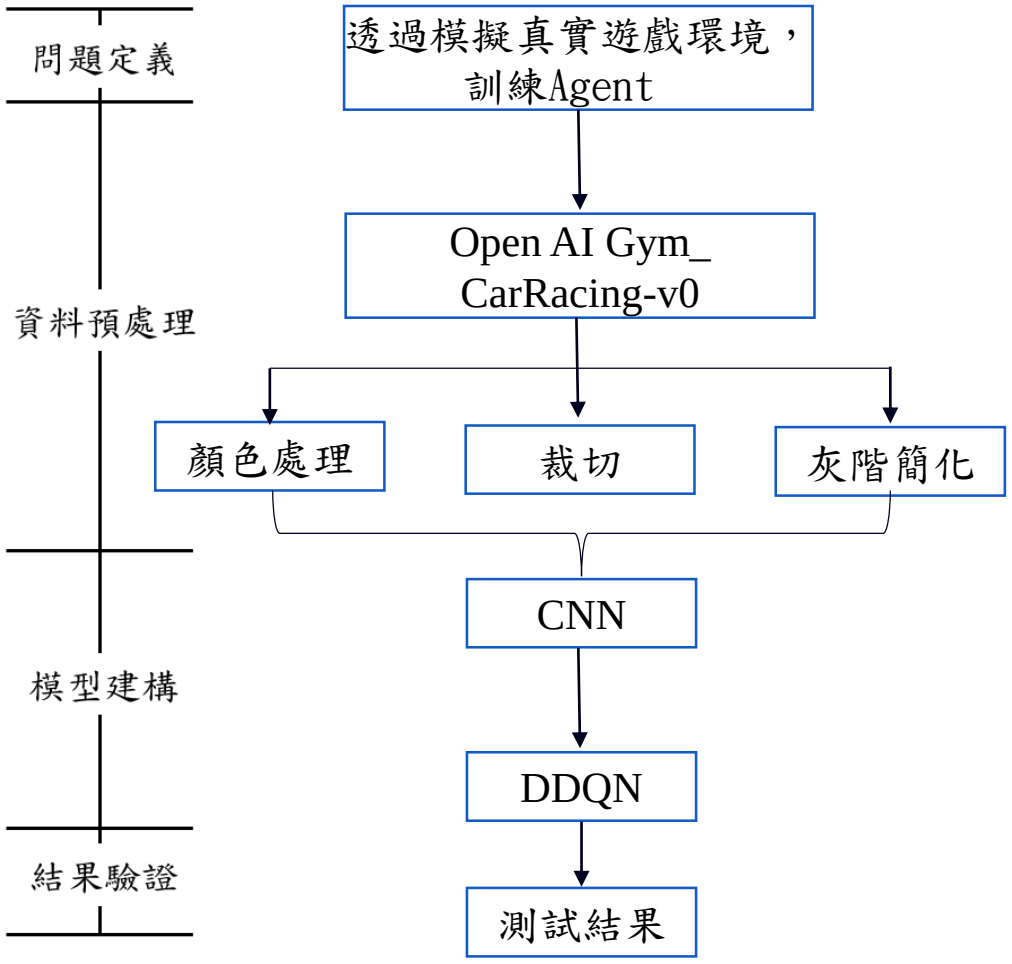
研究背景

研究動機

近年來，強化學習已被廣為流傳，各種優化的演算法不斷提出。因此，本研究期望透過實際處理一強化學習的問題，了解強化學習的模型建構和參數設計。而Open AI Gym 提供免費的環境，讓開發者能在此環境中進行研究開發和比較強化學習的效能。因此，本研究選定Open AI Gym中的CarRacing-v0環境進行研究。

5W1H	
When	當遊戲開始時
Where	Car racing V0每次所自動生成的新環境
Who	Agent
What	讓Agent自行判斷下一步該如何行進，才能使遊戲繼續進行
Why	使Agent能貼近人腦的經驗，甚至超過人類操作
How	資料預處理，CNN/RL學習

研究架構


問題定義
資料預處理
模型建構
結果驗證
透過模擬真實遊戲環境，訓練Agent
Open AI Gym_ CarRacing-v0
顏色處理
裁切
灰階簡化
CNN
DDQN
測試結果

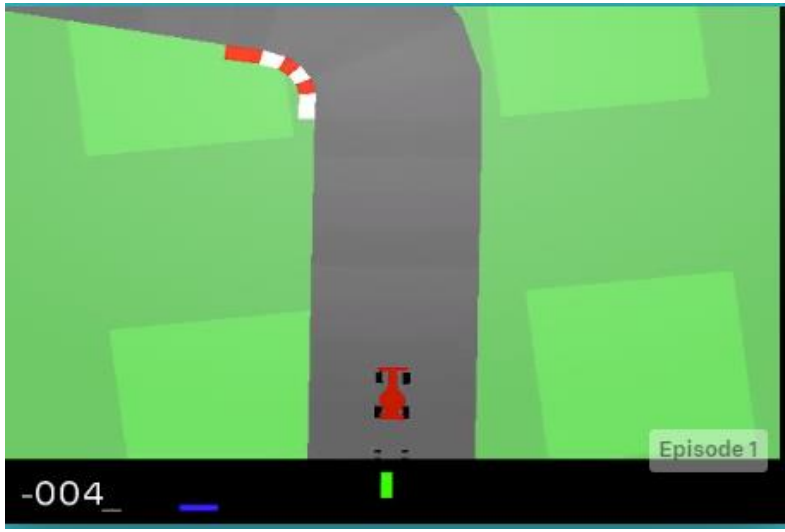
環境介紹

Open AI Gym_CarRacing v0

Open AI Gym 提供免費的環境，讓開發者能在此環境中進行研究開發和比較強化學習的效能。本次所選用的遊戲環境，是CarRacing-v0。
在這個環境中，我們預期車子能夠持續走在灰色的競賽場域，沒有在草地上打滑或撞到障礙的跡象，如此才能在競賽中得到更高的分數。其中，此競賽的分數計算共分為兩部份:

- 隨著時間的流逝，代價為-0.1/frame
- 經過賽道上的進度條所給予的獎勵，每經過一個進度調都給予Agent 1000/N的分數，N為賽道上所有的進度條。

$$\mathbf{score}(t) = 1000 - \frac{t}{10}$$ （t代表還差多少進度條車子可以抵達終點）



Gym

空間描述

狀態空間

本狀態空間由96*96畫素，在此強化學習演算法中，我利用了前三個連續的狀態空間來進行分析，因此總共是96*96*3個彩色網格。這代表了每一個分析的狀態空間的大小是$256^{3*96*96}$。

動作空間

本動作空間可以被描述成$(s,a,d) \in [-1,1] * [0,1] * [0,1]$。在CarRacing的遊戲裡，如果反推由人類來玩此遊戲，動作可被簡化為離散動作，舉例而言，按右鍵代表的是s=-1，下鍵代表d=1。這代表此遊戲有機會讓我們訓練出一個好的結果，因為所有動作都可以離散動作來識別和拆解。在此訓練模型，我將動作拆分為五個，如下圖所示:

```
def step(self, action):
    state_next = []
    info = []
    reward = 0
    done = False
#reduce action space (continuous->discrete)
    accelerate = [0.0, 1.0, 0.0]
    brake = [0.0, 0.0, 0.8]
    left = [-1.0, 0.0, 0.0]
    right = [1.0, 0.0, 0.0]
    none = [0.0, 0.0, 0.0]
```

環境預處理

顏色處理

此環境的顏色相對單純，只有綠、紅、白、灰三色，因此我將原先的彩色轉為灰階

裁切

原先的環境中有計算分數的進度條，然而這對於RL並沒有幫助，因此在訓練時將他裁切，從原先96*96到85*96。並使用cv2.INTER_AREA進行圖像的縮小。

灰階簡化

首先，我將顏色從[0-255] -> [0-1]，接著，我將顏色只拆分為三個區塊，分別是0,0.5,1

狀態空間大小從$256^{3*96*96}$下降為$3^{3*85*96}$。

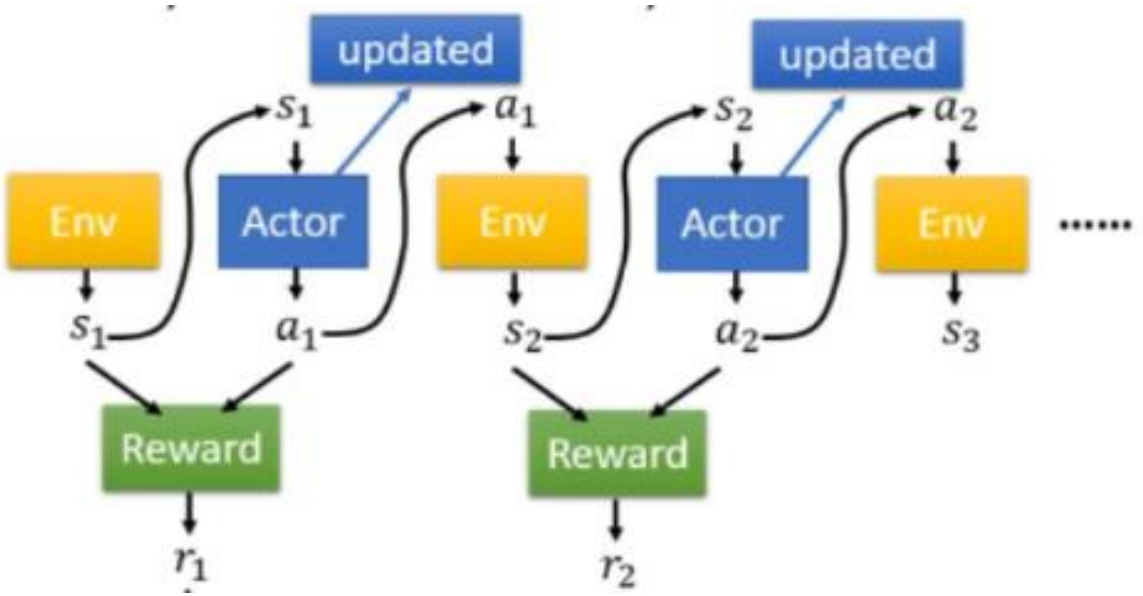
模型介紹 1

Q-learning

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha_t(s_t, a_t) \times \left[R_{t+1} + \gamma \times \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)\right]$$

參數設定

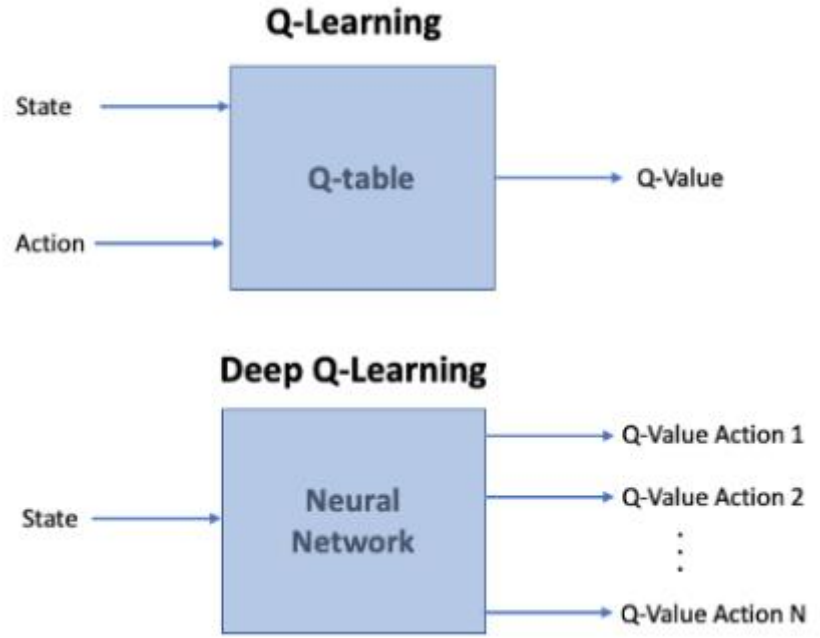
```
GAMMA = 0.99
BATCH_SIZE = 32
REPLAY_SIZE = 10000
LEARNING_RATE = 1e-4
SYNC_TARGET_FRAMES = 1000
REPLAY_START_SIZE = 10000

EPSILON_DECAY_LAST_FRAME = 10**5
EPSILON_START = 1.0
EPSILON_FINAL = 0.02
```



模型介紹 2

Q-learning 神經網路化

傳統的一次只能產生一個Q值，轉變為透過神經網路的方式，一次輸出所有Q-value針對所有可能的action。透過這個方式，能夠有效地加快學習速度，並且能從所有的Q-value中，選擇最大的Q值作為參考。



模型介紹 2

Q-learning 神經網路化

```
    self.conv = nn.Sequential(
        nn.Conv2d(input_shape[0], 32, kernel_size=8, stride=4),
        nn.ReLU(),
        nn.Conv2d(32, 64, kernel_size=4, stride=2),
        nn.ReLU(),
        nn.Dropout(0.5),
        nn.Conv2d(64, 64, kernel_size=3, stride=1),
        nn.ReLU()
    )
```

使用stride代替池化，縮減運算量。另外，我採用Relu激活函數，使的計算更快速。

```
    conv_out_size = self._get_conv_out(input_shape)
    self.fc = nn.Sequential(
        nn.Linear(conv_out_size, 512),
        nn.ReLU(),
        nn.Linear(512, n_actions)
    )
```

經過convolutional layers後，有一個全連接層，產生動作的向量。其中，conv_out_size是從convolutional layers所output的值。

```
def _get_conv_out(self, shape):
    o = self.conv(torch.zeros(1, *shape))
    return int(np.prod(o.size()))
```

泛化此模型，可輸入不同的圖片大小。

```
def forward(self, x):
    conv_out = self.conv(x).view(x.size()[0], -1)
    return self.fc(conv_out)
```

模型介紹 3

Natural DQN and DDQN

Natural DQN在基本的Deep Q-learning model加入了一個Target Q網路，也就是在計算目標Q值時使用這個Target Q網路計算，目的為減少Target Q值與當前Q值的關聯性。

Target Q

Original Q

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For t = 1,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

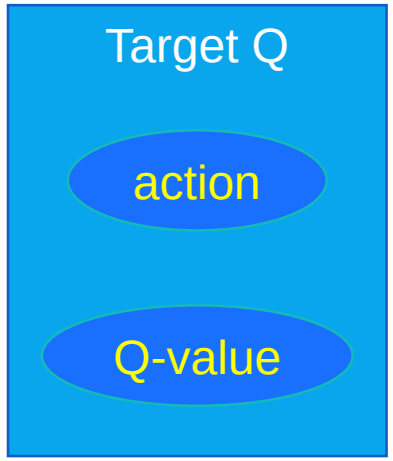
End For

模型介紹 3

Natural DQN and DDQN

Natural DQN在基本的Deep Q-learning model加入了一個Target Q網路，也就是在計算目標Q值時使用這個Target Q網路計算，目的為減少Target Q值與當前Q值的關聯性。

DDQN的唯一不同在於獲取action的方式。目的是用於解決過估計的問題。



Algorithm 1: deep Q-learning with experience replay.

(1) Initialize replay memory D to capacity N

(2) Initialize action-value function Q with random weights θ

(3) Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

(4) **For** episode = 1, M **do**

(5) Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1$,T **do**

(6) With probability ε select a random action a_t
otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

(7) Execute action a_t in emulator and observe reward r_t and image x_{t+1}

(8) Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

(9) Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

(10) Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

(11) Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

(12) Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

(13) Every C steps reset $\hat{Q} = Q$

End For

End For

結果驗證

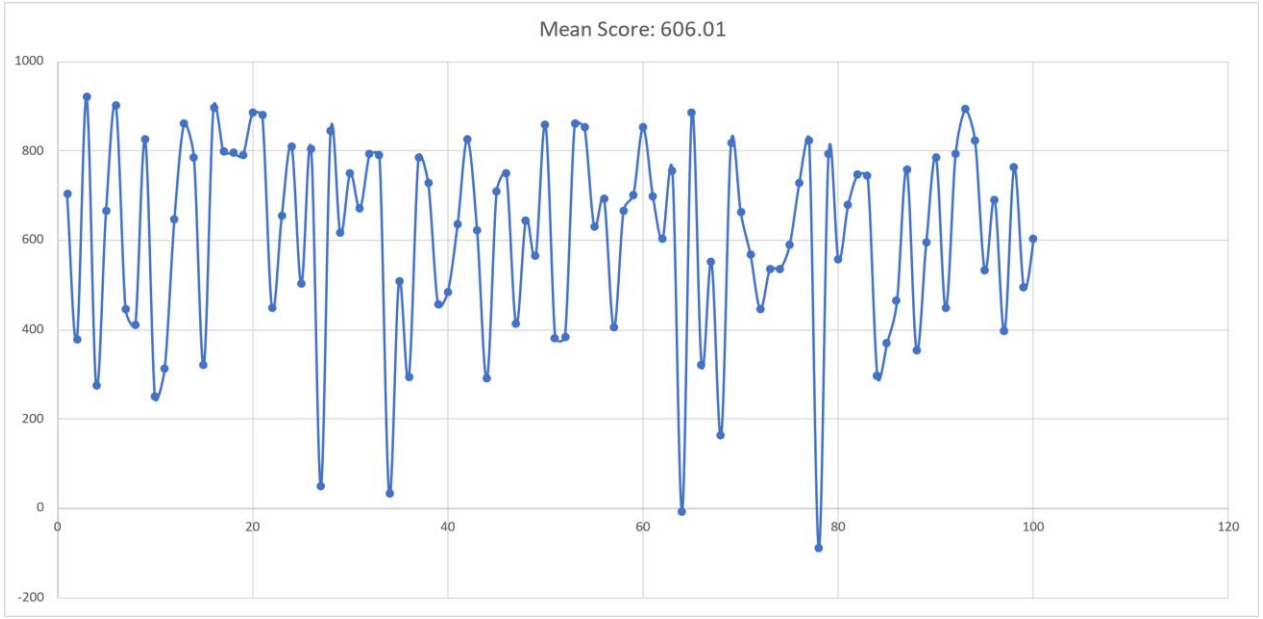
訓練結果

以下是此次的訓練近3340次的訓練結果，可以觀察到在前期有顯著的上升，之後則是緩慢的幅度上下震盪逐漸爬升。選擇不再繼續訓練的原因第一是訓練次數已經夠多，再訓練並不一定會獲得較好的訓練結果，可能只是在附近不斷震盪。

```
  warnings.warn(str(err))
Track generation: 1163..1458 -> 295-tiles track
334: done 1 games, mean reward -40.000, eps 1.00, speed 81.98 f/s
Track generation: 1304..1634 -> 330-tiles track
668: done 2 games, mean reward -34.286, eps 0.99, speed 83.87 f/s
Best mean reward updated -40.000 -> -34.286, model saved
Track generation: 1148..1447 -> 299-tiles track
1002: done 3 games, mean reward -38.967, eps 0.99, speed 82.61 f/s
Track generation: 1059..1328 -> 269-tiles track
1336: done 4 games, mean reward -44.158, eps 0.99, speed 84.15 f/s
Track generation: 1204..1511 -> 307-tiles track
retry to generate track (normal if there are not manyinstances of this messag
Track generation: 1156..1449 -> 293-tiles track
1670: done 5 games, mean reward -40.401, eps 0.98, speed 90.09 f/s
Track generation: 1152..1444 -> 292-tiles track
2004: done 6 games, mean reward -45.197, eps 0.98, speed 86.82 f/s
Track generation: 1155..1448 -> 293-tiles track
2338: done 7 games, mean reward -43.208, eps 0.98, speed 85.53 f/s
Track generation: 1152..1444 -> 292-tiles track
2672: done 8 games, mean reward -38.749, eps 0.97, speed 88.95 f/s
Track generation: 1170..1467 -> 297-tiles track
3006: done 9 games, mean reward -39.063, eps 0.97, speed 87.49 f/s
Track generation: 1327..1663 -> 336-tiles track
3340: done 10 games, mean reward -38.400, eps 0.97, speed 88.31 f/s
Track generation: 1195..1498 -> 303-tiles track
3674: done 11 games, mean reward -40.472, eps 0.96, speed 82.99 f/s
```

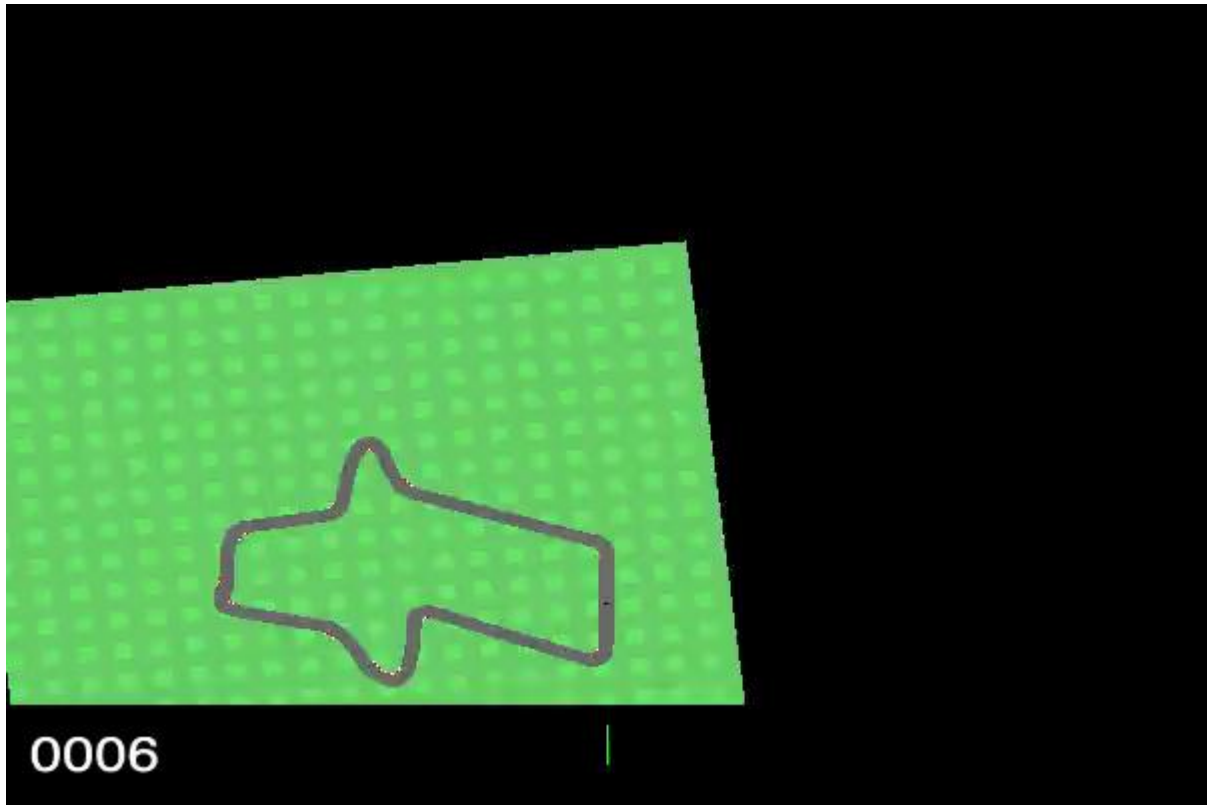
結果驗證

下圖則是訓練後實際測試100次的結果，平均分數是606分。可以看到有些分數異常低的部分，多數是因為車子在前期就打滑的結果。我認為此次的訓練還算是有所收穫，但未來仍有持續進步的空間。平均值如果能達到900分會是一個較好的結果。

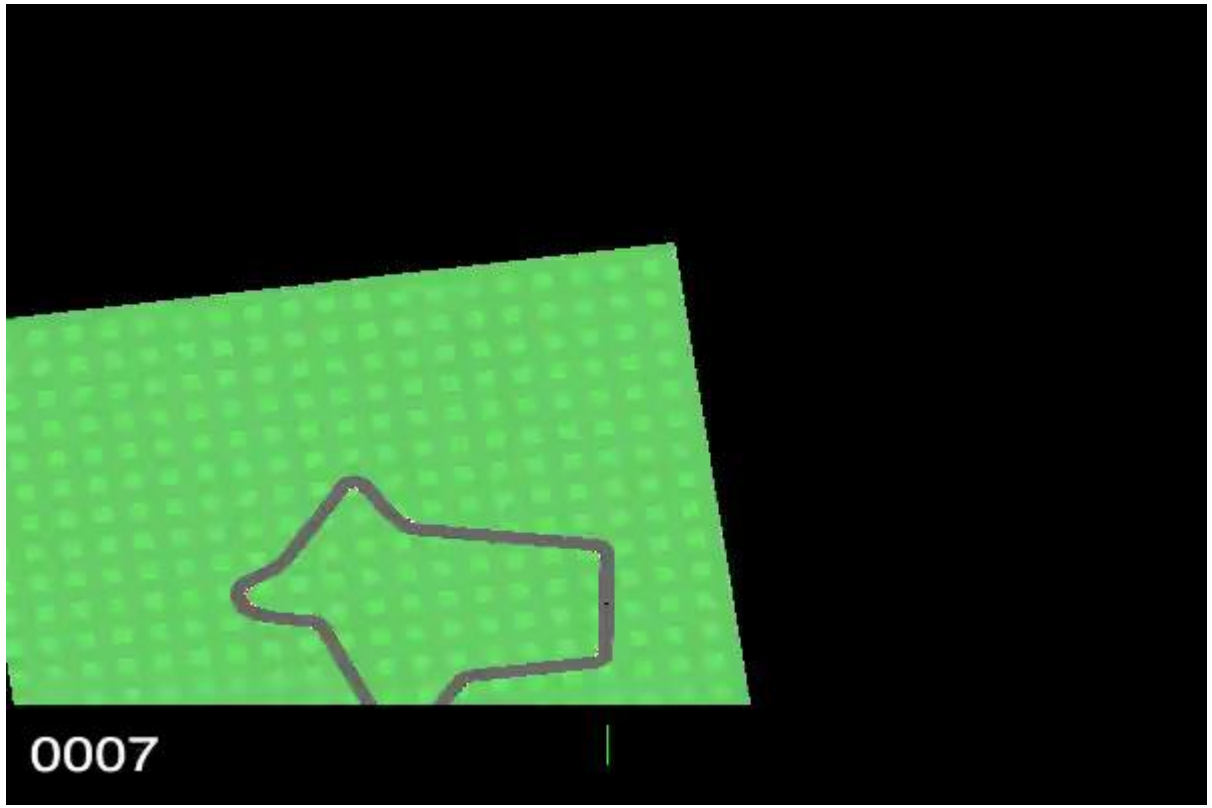


Demo

低分(311)

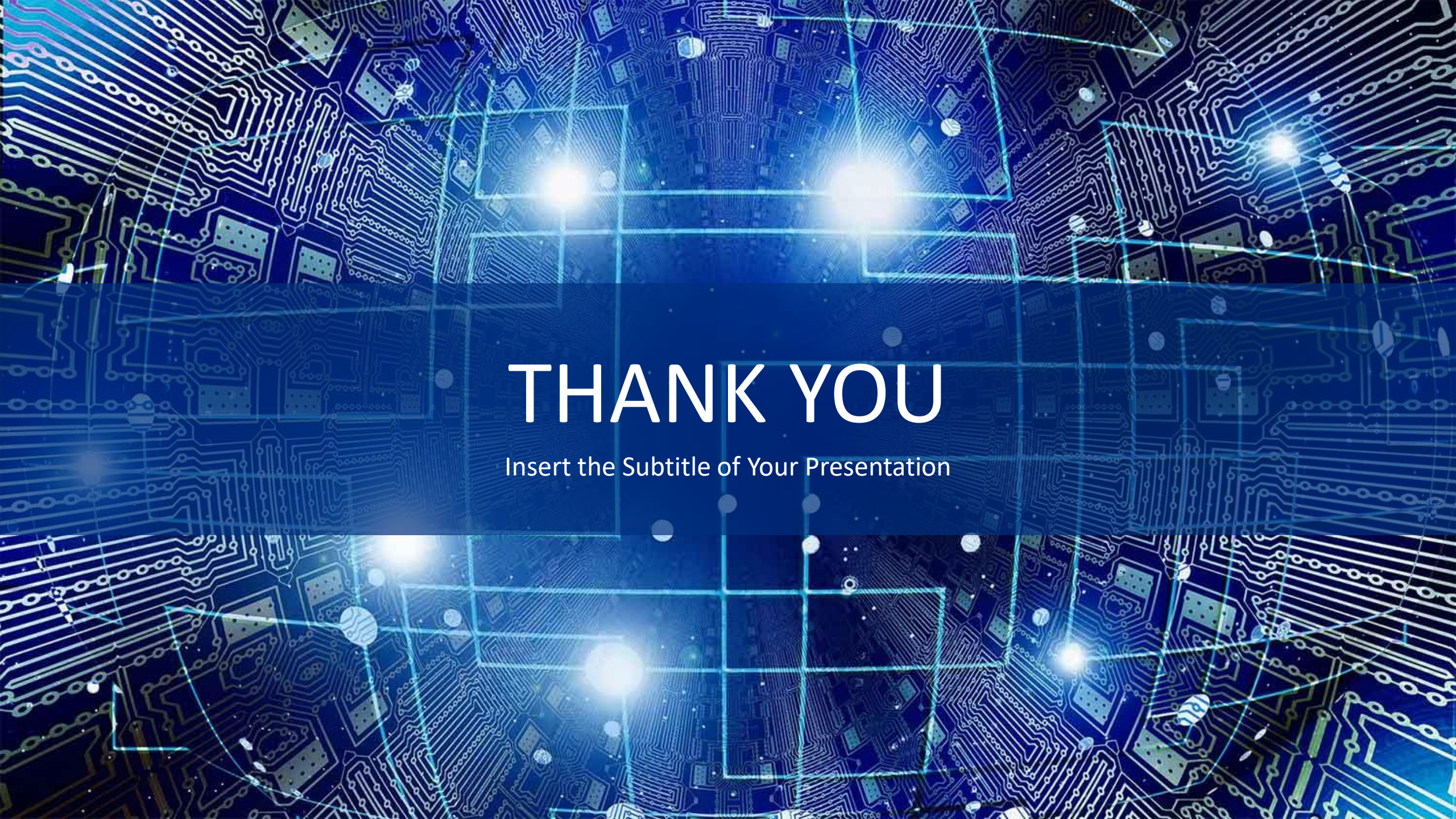


高分(889)



結論

透過此次的練習，我對於強化學習有了一個更全面的認識，也藉此機會學習了Q-learning的演進，包含從最早的Q-table到DQN再到DDQN。未來這個model還有許多可以持續改善的地方，包含套用Dueling DQN或者其他強化學習的方法，來提升它的遊戲訓練分數，以及一個model是否能夠更加泛化於各種Open AI gym的遊戲中等等。


THANK YOU
Insert the Subtitle of Your Presentation