智慧化企業整合

期末報告

應用深度學習模型於 Open AI Gym_Car racing 遊戲

第五組

110034551 高藝真

目錄

壹、 問題定義

一、 研究背景

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 OpenAI 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，此環境提供連續且複雜的動作空間(action space)及狀態空間(state space)，適合用於模擬真實環境下的運作情形。

二、 研究動機

在深度學習的學習中，我對於強化學習特別有興趣。因此我特別針對強化學習的領域，進行相關的論文研究搜尋。強化學習共分為兩類，分別是 Model-free 和 Model-based 兩種。此次的研究將採用 Model-free 的方式，亦即 Agent 事先並不知道任何的環境資訊，每次進入環境時環境都會有所改變，Agent 必須依照當下的狀況做立即的判斷。本次研究將採用 Q-learning 演算法，下面將根據 Q-learning 的演變從 DQN , Natural DQN, DDQN 進行介紹。本次將應用 DDQN 作為訓練模型的演算法。

三、 5W1H

我們利用 5W1H 來定義問題：

表 1 5W1H 問題定義表

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

四、　預計分析架構

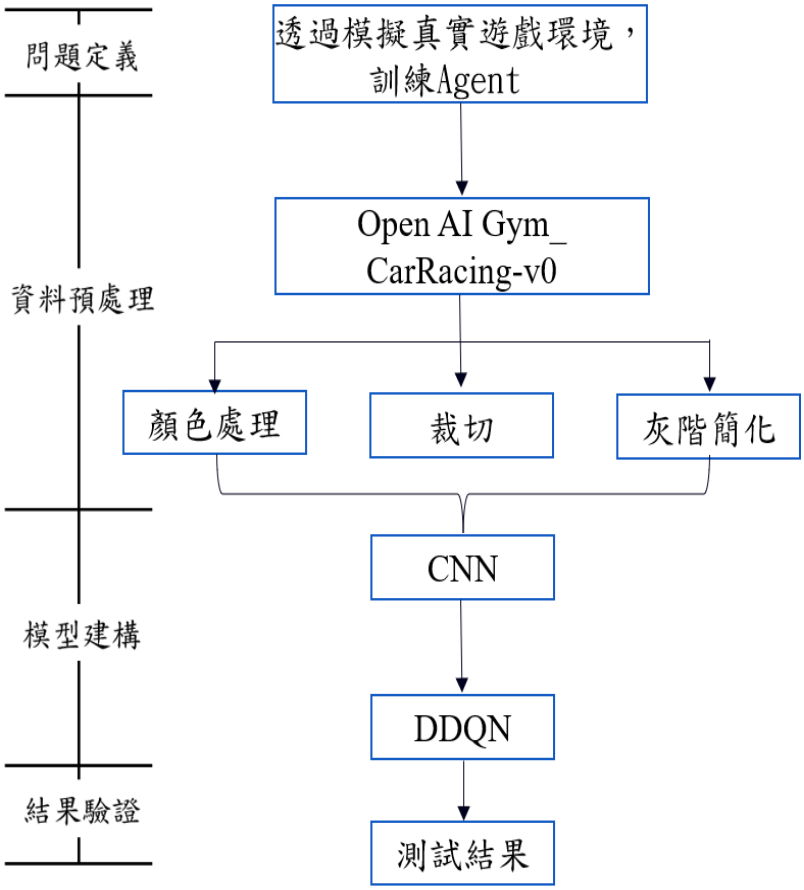


圖 1　分析架構圖

貳、環境介紹及預處理

一、 環境介紹

本次所使用的 CarRacing-v0 環境是一個 top-down 競賽環境，此環境中包含綠色的草地，灰色的競賽場域以及紅白色的障礙。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，其一隨著時間的流逝，會不斷付出代價，代價為-0.1/frame。第二部分是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。簡單的公式可列如下:

$$\mathtt{score}(t) = 1000 - \frac{t}{10}$$

t 代表還差多少進度條車子可以抵達終點。

二、 狀態空間及動作空間描述

本狀態空間由 96*96 畫素，在我的強化學習演算法中，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。這代表了每一個我們分析的狀態空間的大小是$256^{3*96*96}$。這個大小對電腦的處理而言過於龐大，因此接下來我們會進行預處理，設法縮小狀態空間的大小。

本動作空間可以被描述成$(s, a, d) \in [-1,1] * [0,1] * [0,1]$，其中，s 代表方向盤可從最左到最右邊的打圈，a 代表從無加速到最大加速，d 則代表從無煞車到最大的煞車數值。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，如下所示:

A := {left, right, accelerate, brake, none}.

```
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]
```

三、 環境預處理

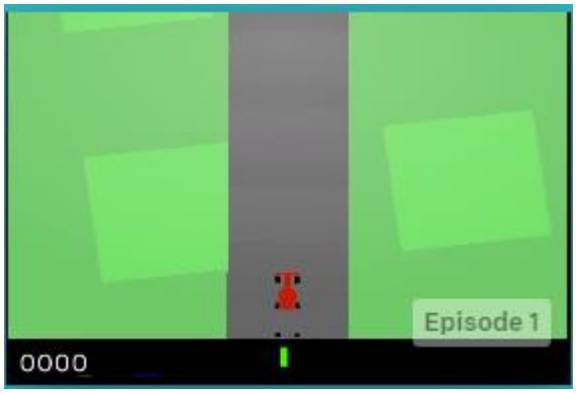
如前所述，為了降低資料大小過度龐大的問題，我先進行資料預處理，在不影響深度學習的效能下，降低資料維度和動作複雜性。

i. 顏色處理

因為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。

ii. 裁切

從下圖可以看到，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。



接著，我使用 cv2.INTER_AREA 進行圖像的縮小。

iii. 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。

做完所有的預處理後，我的狀態空間大小從$256^{3*96*96}$下降為$3^{3*85*96}$，大幅降低訓練的資料量。

參、模型介紹

一、Q-learning

下圖是基本的 Q-learning 架構，我們可以看到整個強化學習由五個部分所組成:狀態(s)，動作(a)，環境，獎勵和 Agent(Actor)。強化學習的目標主要是透過當前的環境觀察，建構狀態和動作動硬的 Q-table。其中，建立 Q-table 的公式如下:

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

其中，左邊的等式是更新後的 Q-value，右邊的等式第一項是更新前的 Q-value。α 是學習率，在此模型中我設為 0.0004，大括弧中的第一項是某個行動所獲得的 reward，而 gamma 是用來調整前後 reward 的重要性，這代表一個 action 在越後面所擁有的影響力將越小。在本次模型中，我設為 0.99。第三項則是下一個狀態時，Q-value 中的最大值。整個大括弧中所有項目就是此次學習的學習值(Learning value)。整個 Q learning 的目標就是可以收煉製最優的 Q 值。總結來說，在 CarRacing-v0 的環境中，我輸入了狀態 s，並輸出每一個 action 的 Q 值，並不斷進行更新。

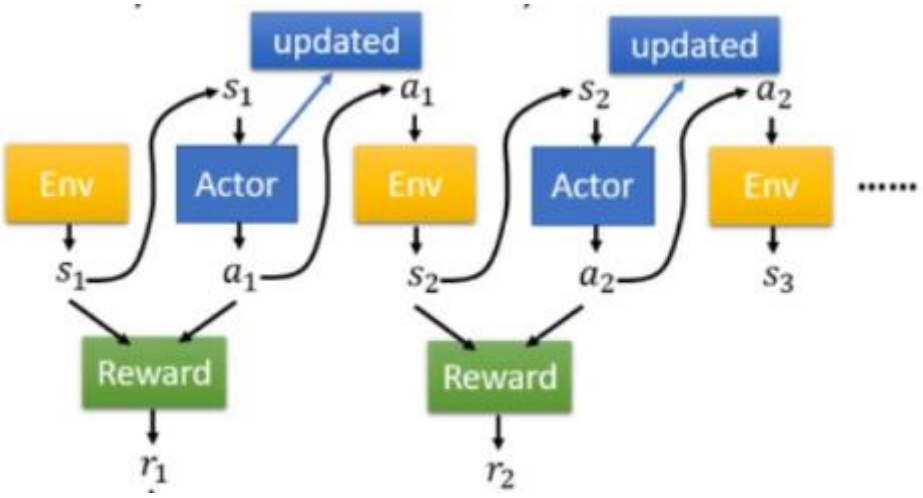


圖 1　超平面（Hyper Plane）示意圖

二、Q-value 神經網路化

在介紹完 Q-learning 的基本架構後，我將介紹如何結合神經網路與 Q-value。透過下圖的方法我們從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。

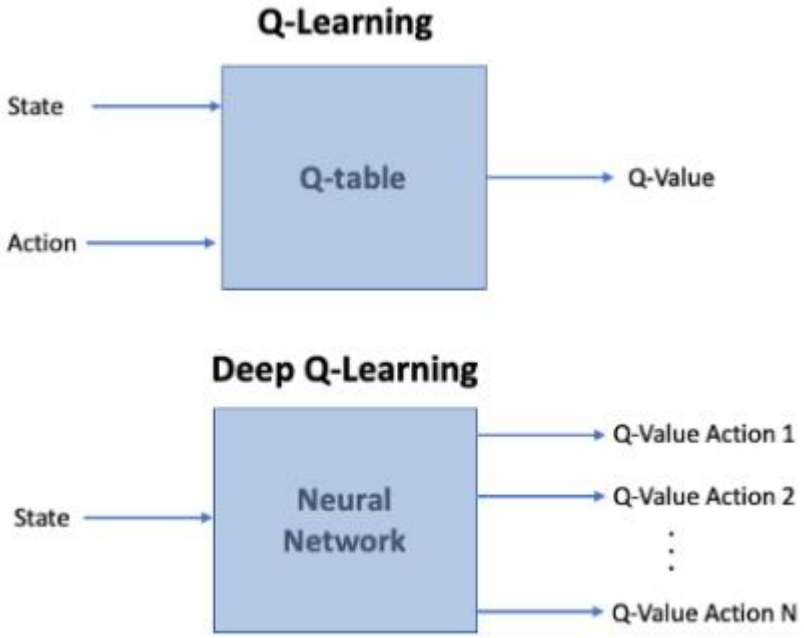


圖 2　隨機森林（Random Forest）示意圖

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。

接著，我將介紹我這次所運用的神經網路架構。首先，圖片將會經過三個 convolutional layers。此次在搭建神經網路時，我沒有採用池化，而是使用 stride 的方式，縮減運算量。另外，我採用 Relu 激活函數，使的計算更快速。詳細程式如附圖所示

```
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()
)
```

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，這個數字是被固定的，為了得到一個更泛化的模型，使其可被應用在不同的遊戲環境中，可適應不同的 input size，需要針對模型做一些變更。

```
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)
)
```

所加入的變更如下圖所示。

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

接著，透過 view() function，可以將數據按照順序排成一維的數據。舉例來說，本來有一 tensor of shape(2,3,4,6)，透過 view 函數的轉換，我們可以轉成 2D 的 tensor，兩列 72 行 view(2,72)。在此次的環境中我們將 4D tensors(batchsize, color channel, 圖片的長和寬)轉換為 2D tensor 至全連接層並獲得每一個 input batch 的 Q-value。

三、Natural DQN and DDQN

我此次的訓練主要以 DDQN model 為主，在介紹 DDQN 前，首先介紹與 DDQN 類似的 Natural DQN model。首先，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。這個 Target Q 網路的參數是由當前的 Q 網路在一段時間更新後，定期複製到 Target Q 網路。以下是 Natural DQN 的演算法步驟:

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 $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
(13) Every C steps reset $\hat{Q} = Q$
End For
End For

然而，無論是原本的 DQN 或是 Natural DQN 都存在 Q-value 過估計的問題，因此，本次的 model 我使用 Double DQN，以解決此問題。在 DDQN 中，唯一的差別是 action 是先由不斷更新的 Q 網路中獲得，因此選擇的依據是當前不斷更新的參數，再去 Target Q 網路獲得該動作對應的值。這樣可以有效降低過估計的情況發生。其餘的部分皆與 DQN 一樣。程式碼部分如下:

```
state_action_values = net(states_v).gather(1, actions_v.unsqueeze(-1)).squeeze(-1)
next_state_acts = net(next_states_v).max(1)[1]
next_state_acts = next_state_acts.unsqueeze(-1)
next_state_vals = tgt_net(next_states_v).gather( 1, next_state_acts).squeeze(-1)
next_state_vals[done_mask] = 0.0
next_state_values = next_state_vals.detach()

expected_state_action_values = next_state_values * GAMMA + rewards_v
return nn.MSELoss()(state_action_values, expected_state_action_values)
```

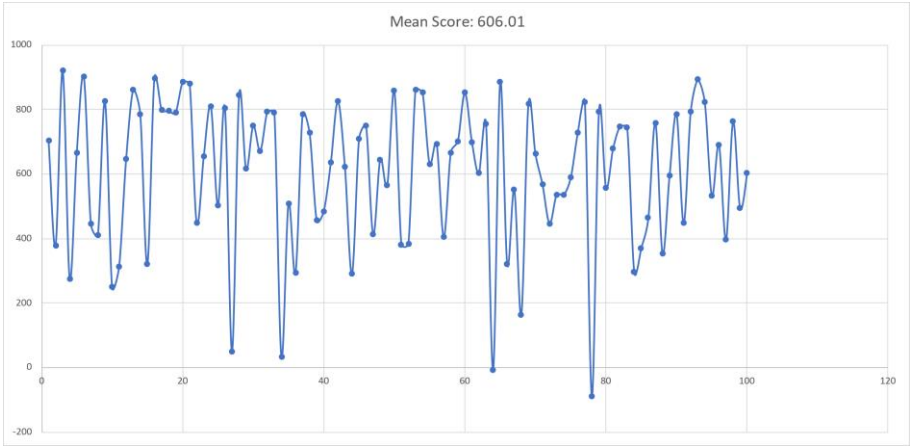
肆、結果驗證

一、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。

二、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。



伍、 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。

陸、 參考資料

1. (2) (PDF) Applying a Deep Q Network for OpenAIs Car Racing Game (researchgate.net)
2. final150.pdf (stanford.edu)
3. 李宏毅 DRL Lecture 3: Q-learning (Basic Idea) - HackMD
4. 卷積神經網路(Convolutional neural network, CNN) — CNN 運算流程 | by Tommy Huang | Medium