以環境探索式 Deep Q Learning 決定代理人行為

張偉治 Q36111281
January 11, 2023

Abstract

以傳統 DQN 為骨幹，修改其中代表 Q table 的 memory，同時不再從環境中獲得獎勵，而是 Agent 自己給自己嘉獎，透過比較是否得到新的資訊決定獲得的獎勵，從而讓模型能夠從當下相對座標以及環境學習到如何探索環境。

1. Introduction

現實中地圖的探索不同於普通遊戲的遊玩。普通遊戲的環境是沙盒式的，同時玩家也知道這件事，因此發展出各種探索沙盒式地圖的策略，例如：先確認邊界或是靠著右邊的牆移動。但現實中地圖的探索是開放式的，若採用與傳統相同的策略，可能根本無法確認邊界，導致訓練無法收斂，無法得到能應用的模型。

但從人類的行為可以得知，人類可以從環境的改變以及各種經驗，總結出生存的策略和標準。而其中強化學習的 Q table 策略正好符合我們今天探索環境的目標。Q table 策略會記錄當下的狀態、動作、動作導致的狀態改變以及獎勵，將其錄至 Q table 中。其中獎勵的概念是 Q table 策略中最核心的部分，透過獎勵以及扣分的機制讓模型傾向於得到最高的分數。

為了符合探索環境這項目標，我們將座標加入至 Q table 中，並且若探索至新座標或是新的環境將給予模型獎勵。因此我們希望從 Deep Q learning (DQN) 模型出發，透過改變其中代表「memory」的內容與結構，使模型能達到探索新環境的目標。

2. Environment design

設計和現實生活中相同的環境是不切實際的。因為設計開放式的地圖不僅難度較高，同時若 Agent 一直朝同個方向探索，將使得負責環境生成的程式需要一不停的生成新的區塊，同時花費大量的記憶空間儲存過去生成環境的資料。而且若要從上帝視角去分析模型的優劣，上述的方法是非常不切實際的。

但若限制 Agent 對環境觀察的範圍，使得 Agent 從環境中得到的訊息不是全域性的，這樣就可以使得 Agent 和環境的相對關係與掃地機器人和真實環境的相對關係是相同的。

同時為了確保 Agent 沒辦法全域性的去探索環境，我們將 Agent 和環境分別設計成兩種 class。同一個 Agent 永遠只會指向同一個環境，直觀地設計時，每次 Agent 查訪環境都需要再指定一次查訪的環境，因為 python 不允許在類別(class)中呼叫其他類別。為了讓程式碼簡潔易維護，我們需要在 Agent 類別中添加一個用以紀錄環境記憶體位置的屬性。為了讓 Agent 能探索環境，我們需要呼叫全域副程式，將環境以及代理人的記憶體位置輸出至類別外，此時該副程式即可同時調用兩個或兩個以上的類別，最後將結果回傳至類別的副程式，即可完成不同類別互相參照的封裝。

在環境類別中，我們分別設計初始化、設定以及展示三種副程式，初始化用以設定環境大小與形狀；設定用以讓環境有更多的多樣性；展示則是方便我們從上帝視角分析模擬的狀況。為了處理 Agent 跑出邊界或是穿牆的問題，設計上我們直接使用包邊的方式解決。

Agent 的設計和環境類似，都有初始化、設定以及展示三種功能，而其最重要同時與環境類別不同的是 step( )和 go( )兩項。Agent 透過 go( )來移動自己在環境中的位置，step( )則是與 DQN 互動，負責 Agent 得到的獎勵以及其他重要資訊回傳至 DQN 的 memory 中。

```
    def  step(self,action):
        self.go(action)
        done  =  False

        #  reward設計
        if(self.flag==1):  reward  =  -1
        elif(self.flag==2):
            self.wall_error  +=  1
            reward  =  -1*(self.wall_error  if  self.wall_error<4  else  4)
#ver3#  遇到新環境的獎勵
        elif(self.locx  not  in  self.map[:,0]  or  self.locy  not  in  self.map[:,0]):  reward  =  1
        else:  reward  =  0.1


        #  取得下一階段state
        state  =  agent_get_env(self.locx,  self.locy,  self.env)
#ver3#  替state加上座標
        state  =  agent_get_env(self.locx,  self.locy,  self.env)
        state  =  np.hstack((state,  self.locx,  self.locy))

        #  是否end  game
        if(self.bound_error>10):    done  =  True
```

Figure 1

```
def  go(self,do):
    if(do==0):
        flag  =  self.loc_test(self.locy-1,  self.locx)
        if(flag==0):    self.go_up()
    elif(do==1):
        flag  =  self.loc_test(self.locy+1,  self.locx)
        if(flag==0):    self.go_down()
    elif(do==2):
        flag  =  self.loc_test(self.locy,  self.locx-1)
        if(flag==0):    self.go_left()
    elif(do==3):
        flag  =  self.loc_test(self.locy,  self.locx+1)
        if(flag==0):    self.go_right()
    self.flag  =  flag
```

figure 2

3. Deep Q learning

DQN 分別有不斷迭代的 evaluation network，以及根據 'replace_target_iter' 更新的 target network，如此的設計是希望該模型是從長期的經驗學習而非根據短期經驗進行決策，同時這樣的設計能幫助模型更好的收斂。

```python
# 實際進行訓練的 evaluation network
class Eval_Q_Model(tf.keras.Model):
    # Evaluate Q Model 的模型架構
    def __init__(self, num_actions):
        super().__init__('mlp_q_network')
        self.layer1 = layers.Dense(10, activation='relu')
        self.logits = layers.Dense(num_actions, activation=None)

    def call(self, inputs):
        x = tf.convert_to_tensor(inputs)
        layer1 = self.layer1(x)
        logits = self.logits(layer1)
        return logits

# 久久更新一次的 evaluation network
class Target_Q_Model(tf.keras.Model):
    # Target Q Model 的模型架構(不更新layer權重)
    def __init__(self, num_actions):
        super().__init__('mlp_q_network_1')
        self.layer1 = layers.Dense(10, trainable=False, activation='relu')
        self.logits = layers.Dense(num_actions, trainable=False, activation=None)

    def call(self, inputs):
        x = tf.convert_to_tensor(inputs)
        layer1 = self.layer1(x)
        logits = self.logits(layer1)
        return logits
```

figure 3

Figure 4 中黃色的部分是 DQN 的核心，代表過去經驗的紀錄，其大小為 memory_size，memory_size 越大代表越多過去的經驗越多，同時需要的儲存資源也越大，但其中可能儲存很多不重要的甚至是誤導模型的經驗。若 memory_size 設計太小則會使得模型沒有足夠的經驗應付多變的狀況，但若環境變化不大則可以將 memory_size 設小。因此適當的 memory_size 可以讓模型有足夠的經驗去決策行為，同時也能適當淘汰掉不佳的經驗。

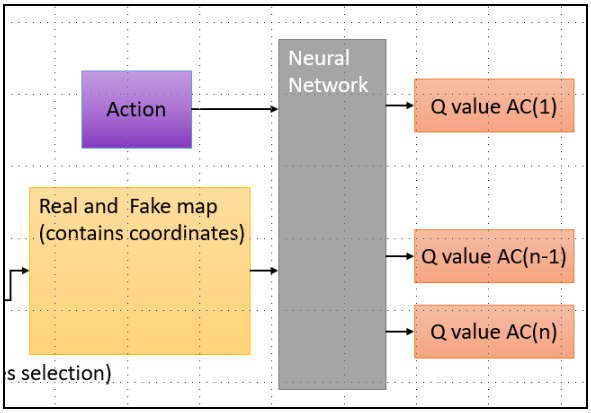

figure 4

3.1 Add Coordinate in Q Memory

與傳統 DQN 不同，我們在狀態中加入座標當作特徵，並且獎勵的部分並非是從環境中獲得，而是比較過去經驗，若有遇到新的經驗則獲得獎勵。

現在狀態	移動後的狀態	獎勵		行為
Agent周邊八格狀態+當下座標	移動後周邊八格狀態+移動後座標	撞邊界	次數(n)<=4	-n	上、下、左、右
			次數(n)<4	-4	
		遇到新環境		+1	
		遇到新座標		+1	
		移動		+0.1	

figure 5

4. Evaluation Model

我們知道 Memory_size 的大小決定了模型能記憶的過去經驗數量，因此我們設計 500、200、100 三種不同 Memory_size 的模型探索 10*10 的環境。可以觀察到 Memory_size = 200 時有最好的探索效果，這是因為比起 Memory_size = 500，Memory_size = 200 能夠淘汰掉不佳的經驗，同時比起 Memory_size = 100，Memory_size = 200 又能夠記錄下足夠多的經驗以供 DQN 做決策。

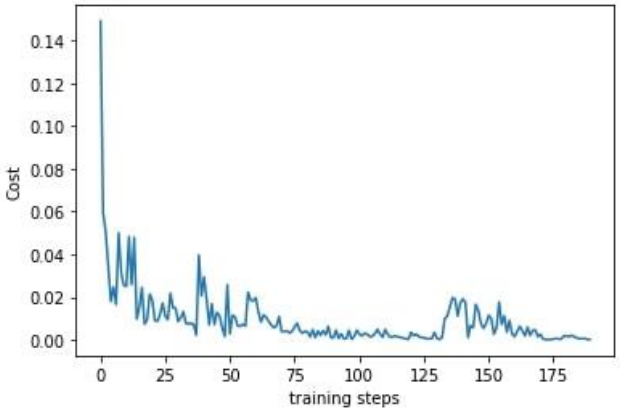

figure 6：Memory_size = 500

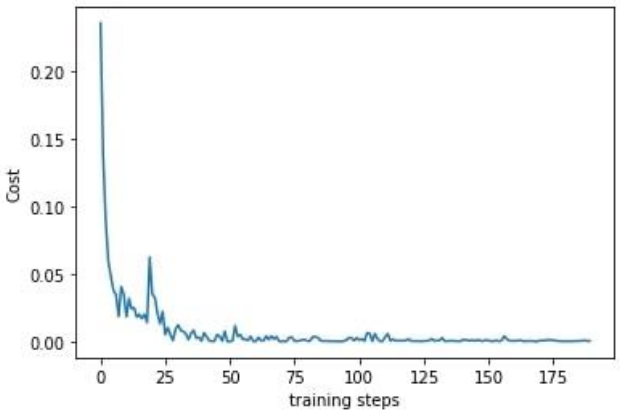

figure 7：Memory_size = 200

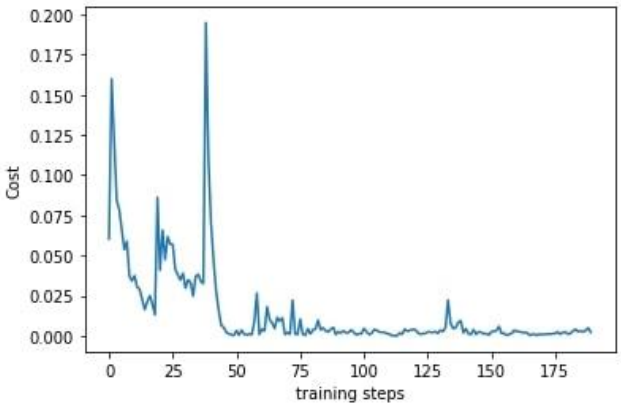

figure 8：Memory_size = 100

模型探索的環境大小也決定了模型探索的訓練結果。在最小的地圖中，模型移動沒幾步就會撞到牆壁或是邊界，因此前期 cost 較大，過了一段時期後 cost 就會收斂。而在最大的地圖中，因為模型碰到邊界或是牆的機率是三者最低，因此訓練一下就收斂，不過移動的決策幾乎是傾向隨機。

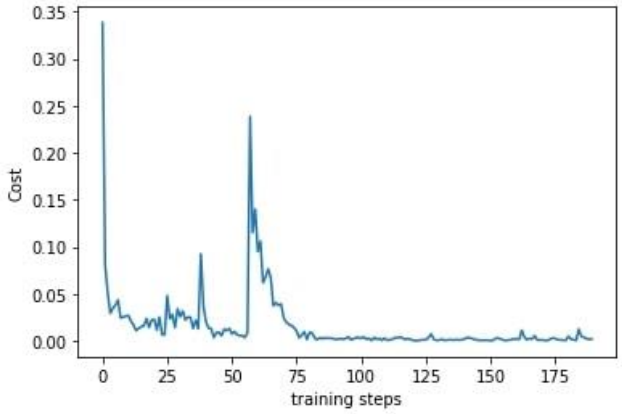

figure 9：env = 5*5

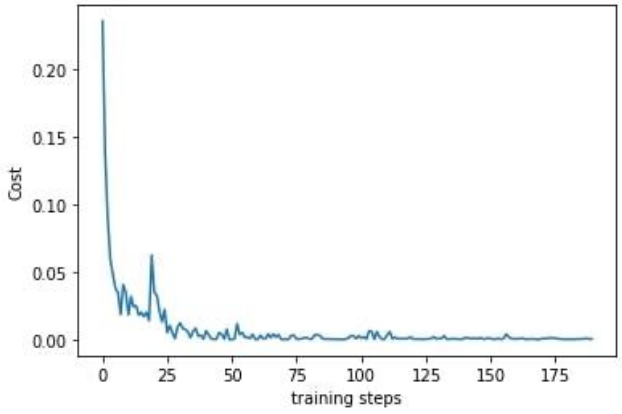

figure 10：env = 10*10

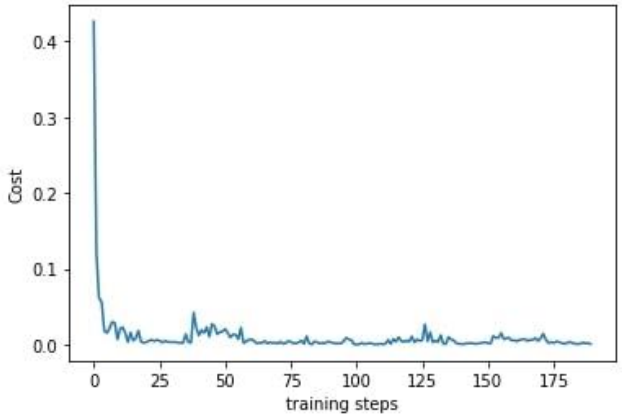

figure 11：env = 20*20

為了模擬在實際環境中的效果，因此我們設計了一張複雜的地圖，該地圖的特色是初始位置狹窄，只有向右或向下可以離開初始位置的侷限。模型參數中 memory_size=200、e_greedy=0.9。從訓練的圖形中可以發現，在前 25 次訓練 cost 非常不穩定，但之後馬上就收斂，然而該結果並非模型訓練狀況良好，在實際使用模型決定探索路徑時，模型僅學會如何從起始位置的窄空間離開，離開該空間後就一直於下方遊蕩或是回到起始位置。

figure 12：複雜地圖

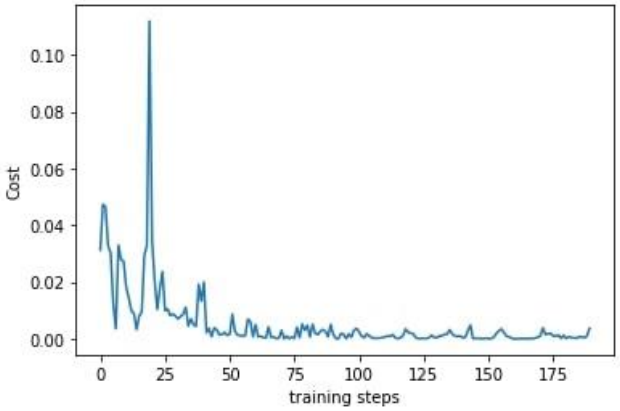

figure 13：模型於複雜地圖的訓練狀況

5. Conclusion

從實驗過程可以發現，對於強化學習這種基於環境而學習的模型，訓練結果良好並不代表實際效果很好，有可能只有從多次的經驗中學到皮毛而已，需要我們額外設計全域的評估方式來評估模型的好壞，例如環境探索率。

不過透過實驗，我們證實加入座標和使用新的獎勵評估方式的 DQN 可以使用在簡單的環境中，但對於複雜環境卻會因為 e_greedy 的設計而使得模型在一定次數後就不具有隨機行走的能力，因此未來需要設計 e_greedy 會隨著遇到新環境而更新，使得模型在遇到新環境時能更勇於探索。

參考文獻

[1] Henry Charlesworth, Giovanni Montana: “PlanGAN: Model-based Planning With Sparse Rewards and Multiple Goals”, 2020; arXiv:2006.00900.

[2] Bradly C. Stadie, Sergey Levine, Pieter Abbeel: “Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models”, 2015; arXiv:1507.00814.

[3] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, Mohammad Norouzi: “Dream to Control: Learning Behaviors by Latent Imagination”, 2019; arXiv:1912.01603.

[4] 莫烦 Python 网,(https://mofanpy.com/)