木棒台車平衡問題(cart-pole system)

王禹晴

國立清華大學工業工程與工程管理學系
yuching20131211@gmail.com

摘要

隨著時代的變遷，機器學習越來越熱門，使人們趨之若鶩，各種生活中的實務都想以機器學習的方式操作看看，而強化學習(Reinforcement learning)是一種機器學習方法，Q-learning 是強化學習中一個最知名的演算法，他能解決的是以下的問題：一個能「感知環境(獎勵、懲罰)」的無監督 agent，學習如何選擇「動作」 達到其最佳獎勵值。本專題說明如何利用 Q-learning 與類神經網路，在 OpenAI gym 的環境下模擬木棒台車平衡問題，分析不同條件下所獲得的獎勵機制，統整參數調整後的各個優缺點，了解其強弱項差異，找出最適合木棒台車平衡問題的情況。

關鍵字：Cart-pole system、Reinforcement learning、Q-learning

1 緒論

由於強化學習的普遍性，現今在許多領域皆有應用，Q-learning 為其中一個較常見的演算法，而木棒台車平衡問題(cart-pole system)是一個經典的 Q-learning 例子，為了解其內涵，本章以 5W1H 分析描述此專題。

1.1 5W1H 分析

- Why

  由於機器學習的熱門，而強化學習又是機器學習家族的一員，為一種目標導向(goal-oriented)的學習方法，旨在經由與環境互動過程中獲得的各種獎勵或懲罰，學會如何做決策，其中 Q-learning 又是強化學習的其中一個演算法，為了更了解這個演算法，便以木棒台車平衡問題為例子做研究，可以在一個較簡單的環境下學習。
- What

  建立一個木棒台車平衡系統，可以讓學習者快速上手。

- Who
  任何想藉著簡單的動作及狀態快速了解強化學習的學習者。
- When
  任何學習者想用簡單的例子學習 Q-learning 時。
- Where
  本專題所假想應用於實務上真正的木棒台車系統。
- How
  利用 4 個不同的方法及參數的改變快速明白木棒台車平衡問題的運作模式。

2 文獻回顧

本章回顧相關文獻。

2.1 OpenAI gym 下建構的 cart-pole 仿真環境

將世界表示為通過過渡（或動作）連接的狀態圖，這意味著要預測的未來狀態，只需考慮當前狀態和選擇執行的操作即可，這裡的關鍵是不需要考慮以前的狀態，這就是所謂的馬爾可夫模型。儘管過去確實會影響未來，但是此模型仍然有效，因為始終可以在當前狀態下編碼有關過去的信息。

有些狀態還包括了記憶，該記憶編碼了過去（或認為已經完成）的所有事情，但是並非所有狀態都是完全可觀察的，通常只能從可觀察狀態中猜測出真實狀態。例如，我不知道你在想什麼，只能通過你的行動來推斷，這就是所謂的“ 隱馬爾可夫模型”。

此專題的車桿世界由一個沿著水平軸移動的車和一個固定在車上的桿組成。在每個時間步長，都可以觀察其車位置、車速度、桿角度和桿尖速度，這些是這個世界可觀察的狀態，在任何狀態下，木棒台車只有兩種可能的操作：向左移動或向右移動。

換句話說，木棒台車桿的狀態空間具有四個連續值的維，而動作空間具有兩個離散值的一維。

2.2 Q-learning 及 DQN(Deep Q Network)

Q-learning 是強化學習的一種方法，其主要目的就是要記錄下學習過的政策，因而告訴 agent 什麼情況下採取什麼行動會有最大的獎勵值。Q-learning 不需要對環境進行建模，即使是對帶有隨機因素的轉移函數或者獎勵函數也不需要進行特別的改動就可以進行。「Q」這個字母在強化學習中表示一個動作的品質

（quality)。

在普通的 Q-learning 中，當狀態和動作空間是離散且維數不高時可使用 Q-Table 儲存每個狀態動作對的 Q 值，而當狀態和動作空間是高維連續時，使用 Q-Table 的動作空間和狀態太大十分困難，所以可以把 Q-table 更新轉化為函數擬合問題，通過擬合一個函數來代替 Q-table 產生 Q 值，使得相近的狀態得到相近的輸出動作，因此，可以想到深度神經網路對複雜特徵的提取有很好效果，所以將深度學習與強化學習結合，這就成為了深度 Q 網路(DQN)。

3 研究架構與方法

3.1 研究架構

選擇最適條件下的木棒台車平衡問題有許多因素需要考量，而每種條件下所考量的因素權重各有不同，為了更清楚的了解其運作模式，故本專題採取由淺入深的方式，分為 4 個不同的方法：Random Action、Hand-Made Policy、Q-table、DQN，並在下一個章節做進一步的結果分析。

3.2 研究方法實作

在 4 個方法開始前先查看需要的觀測數與各項範圍。

```
print(env.action_space)             # 查看這個環境中可用的 action 有多少個
print(env.observation_space)        # 查看這個環境中可用的 state 的 observation 有多少個
print(env.observation_space.high)   # 查看 observation 最高取值
print(env.observation_space.low)    # 查看 observation 最低取值
```

圖 3.2 觀測數及各項範圍

3.2.1 Random Action

首先用最簡單的例子體驗 gym 的使用，無論環境如何，都是採取隨機進行動作，也就是隨機決定要將小車左移或右移。

圖 3.2.1.1 程式碼意涵：每次嘗試都要到達終止狀態，一次嘗試結束後，agent 都要從頭開始，這就需要 agent 具有重新初始化的功能，函數 reset()就是這個作用。

函數 render()在這裡扮演圖像引擎的角色，一個模擬環境必不可少的兩部分是物理引擎和圖像引擎，物理引擎類比環境中物體的運動規律；圖像引擎用來顯示環境中的物體圖像。對於強化學習演算法，該函數可以沒有，但是為了便於直觀顯示當前環境中物體的狀態，圖像引擎還是有必要的。

```python
import gym

if __name__ == '__main__':
    env = gym.make('CartPole-v0')

    # 跑 200 個 episode，每個 episode 都是一次任務嘗試
    for i_episode in range(200):
        observation = env.reset()  # 讓 environment 重回初始狀態
        rewards = 0                # 累計各 episode 的 reward
        for t in range(250):      # 設個時限，每個 episode 最多跑 250 個 action
            env.render()          # 呈現 environment
```

圖 3.2.1.1 agent 及匯入 gym 圖

圖 3.2.1.2 程式碼意涵：函數 step()在模擬器中扮演物理引擎的角色，其輸入是動作，輸出是下一步狀態，立即回報，是否終止。該函數描述了 agent 與環境交互的所有資訊，是環境檔中最重要的函數，一般利用 agent 的運動學模型和動力學模型計算下一步的狀態和立即回報，並判斷是否達到終止狀態。

```python
action = env.action_space.sample() # 在 environment 提供的 action 中隨機挑選
observation, reward, done, info = env.step(action) # 進行 action，environment 返回該 action 的 reward 及前進下個 state
rewards += reward# 累計 reward
```

圖 3.2.1.2 Random Action key section 圖

```python
if done: # 任務結束返回 done = True
    print('Episode finished after {} timesteps, total rewards {}'.format(t+1, rewards))

    break
```

圖 3.2.1.3 任務結束圖

```python
env.close() # 需要結束，不然錯誤會出現
```

圖 3.2.1.4 環境關閉圖

3.2.2 Hand-Made Policy

為了讓 agent 不會走得太隨意，再來引進一個簡單的策略，如果桿子向左傾（角度 < 0），則小車左移以維持平衡，否則右移。

```python
#定義 policy
def choose_action(observation):
    pos, v, ang, rot = observation #小車位置，小車速度，柱子角度，柱尖速度
    return 0 if ang < 0 else 1 # 僅基於角度的簡單規則 #柱子左傾則小車左移，否則右移
```

圖 3.2.2.1 Hand-Made Policy choose_action 圖

```
action = choose_action(observation) # 根據 hand-made policy 選擇動作
observation, reward, done, info = env.step(action) # 做動作，獲得獎勵
rewards += reward
```

圖 3.2.2.2 Hand-Made Policy key section 圖

3.2.3 Q-table

為了學習在某個狀態之下做出好的行為，我們定義所謂的 Q 函數 Q(s, a)，也就是根據身處的狀態(s)進行動作(a)所預期未來會得到的總獎勵。如果能求出最佳 Q 函數 Q*(s, a)，agent 在任何狀態之下，只要挑選能最大化未來總獎勵的動作，即 argmax_a Q*(s, a)，則能在任務中獲得最大獎勵，而習得 Q 函數的過程正是 Q-learning。

接著 agent 要藉由一次次跟環境互動中獲得的獎勵來學習 Q 函數，起初 agent 一無所知時，Q 函數的參數都是隨機的，再從跟環境互動的每一步，慢慢更新參數，逼近我們要的最佳 Q 函數，而把各個 state-action pair 的 Q 值存在 table 裡，直接查找或更新，即是所謂 Q-table。

圖 3.2.3.1 程式碼意涵：目標是學習到最佳 Q 函數，過程中以 ε-greedy 方法與環境互動，從中獲得獎勵以更新 Q-table 裡的 Q 值。ε-greedy 是一種在探索和開發間取得平衡的方法。探索是讓 agent 大膽嘗試不同動作，確保能夠吸收新知，而開發是讓 agent 保守沿用現有策略，讓學習過程收斂。方法很簡單：ε 是隨機選擇動作的機率，所以平均上有 ε 的時間 agent 會嘗試新動作，而(1 - ε)的時間 agent 會根據現有策略做決策。

```
def choose_action(state, q_table, action_space, epsilon):
    if np.random.random_sample() < epsilon: # 有 ε 的機率會選擇隨機 action
        return action_space.sample()
    else: # 基於 Q table 的 greddy action
        #其他時間根據現有 policy 選擇 action，也就是在 Q table 裡目前 state 中，選擇擁有最大 Q value 的 action
        return np.argmax(q_table[state])
```

圖 3.2.3.1 Q-table choose_action 圖

圖 3.2.3.1 程式碼意涵：狀態的表示。在 Cart-Pole 環境裡觀察到的特徵都是連續值，不適合作為一個表格的輸入，因此要將一個區間、一個區間的值包在一起用離散數值表示，也就是上面的存儲桶(bucket)。要使用 Q-Learning，必須將連續尺寸離散化為多個存儲桶，通常希望有更少的存儲桶，並保持狀態空間盡可能的小，更少的最佳策略來尋找意味著更快的培訓。

```python
def get_state(observation, n_buckets, state_bounds):
    state = [0] * len(observation)
    for i, s in enumerate(observation): # 每個 feature 有不同的分配
        l, u = state_bounds[i][0], state_bounds[i][1] # 每個 feature 值的範圍上下限
        if s <= l: # 低於下限，分配為 0
            state[i] = 0
        elif s >= u: # 高於上限，分配為最大值
            state[i] = n_buckets[i] - 1
        else: # 範圍內，依比例分配
            state[i] = int(((s - l) / (u - l)) * n_buckets[i])

    return tuple(state)
```

圖 3. 2. 3. 1 Q-table get_state 圖

```python
# 準備 Q table
## Environment 中各個 feature 的 bucket 分配數量
## 1 代表任何值皆表同一 state，也就是這個 feature 其實不重要
n_buckets = (1, 1, 6, 3) # Observation space: [小車位置，小車速度，桿子角度，桿尖速度 ]
```

圖 3. 2. 3. 2 Q-table buckets 圖

```python
# 離散 actions (數量)
n_actions = env.action_space.n

# state 範圍
state_bounds = list(zip(env.observation_space.low, env.observation_space.high))
state_bounds[1] = [-0.5, 0.5]
state_bounds[3] = [-math.radians(50), math.radians(50)]
```

圖 3. 2. 3. 3 Q-table action, state 圖

```python
# Q table，每個 state-action pair 存一值
q_table = np.zeros(n_buckets + (n_actions,))
```

圖 3. 2. 3. 4 Q-table state-action pair 圖

```python
# Q-learning
for i_episode in range(200):
    epsilon = get_epsilon(i_episode)
    lr = get_lr(i_episode)

    observation = env.reset()
    rewards = 0
    state = get_state(observation, n_buckets, state_bounds) #  將連續值轉成離散
    for t in range(250):
        env.render()
```

圖 3. 2. 3. 5 Q-table Q-learning 圖

圖 3. 2. 3. 6 程式碼意涵：學習過程中為了方便收斂，一些參數像 ε 和 learning rate 會隨著時間遞減，也就是 agent 從大膽亂走，到越來越相信已經學到的經驗。

```
# 學習相關的常數； 反複試驗決定的因素
get_epsilon = lambda i: max(0.01, min(1, 1.0 - math.log10((i+1)/25))) # epsilon-greedy;隨時間遞減
get_lr = lambda i: max(0.01, min(0.5, 1.0 - math.log10((i+1)/25))) # learning rate; 隨時間遞減
gamma = 0.99 # reward discount factor
```

圖 3.2.3.6 學習相關參數圖

```
# Agent takes action
action = choose_action(state, q_table, env.action_space, epsilon)
observation, reward, done, info = env.step(action)
rewards += reward
next_state = get_state(observation, n_buckets, state_bounds)

#更新 Q table
q_next_max = np.amax(q_table[next_state]) # 進入下一個 state 後，預期得到最大總 reward
q_table[state + (action,)] += lr * (reward + gamma * q_next_max - q_table[state + (action,)])

# 前進下一 state
state = next_state

if done:
    print('Episode finished after {} timesteps, total rewards {}'.format(t+1, rewards))
    break
```

圖 3.2.3.7 Q-table key section 圖

3.2.4 DQN

上一小節的 Q-table 方法的壞處是 table 大小有限，不適用於狀態和動作過多的任務，另一個方法是用 neural network 去逼近 Q 函數，即 Deep Q-Learning，如此一來就不會有容量限制了，而所謂 neural network 就是藉由不斷被餵食 input-output pair 後，最終逼近 input-output 對應關係的函數，亦即 f(input) = output。

由 neural network 取代 Q-table 的好處是，neural network 可以搭配不同變形，從龐大的狀態空間中自動提取特徵，例如經典的 Playing Atari with Deep Reinforcement Learning 即是以 Convolutional Neural Network 直接以遊戲畫面的 raw pixel 下去訓練，這是僵化的 Q-table 辦不到的。

以下會分為 3 個步驟說明整個 DQN 方法架構：建立 Network、建立 Deep Q-Network 及訓練。

(1)建立 Network：

首先建立一層隱藏層的 neural network，把狀態傳入後，得出每個動作的分數，分數越高的動作越有機會被挑選，目標是在當前狀態下，讓對未來越有利的動作分數能越高。

```python
class Net(nn.Module):
    def __init__(self, n_states, n_actions, n_hidden):
        super(Net, self).__init__()

        # 輸入層 (state) 到隱藏層，隱藏層到輸出層 (action)
        self.fc1 = nn.Linear(n_states, n_hidden)
        self.out = nn.Linear(n_hidden, n_actions)

    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x)
        actions_value = self.out(x)
        return actions_value
```

圖 3.2.4.1 DQN neural network 圖

(2)建立 Deep Q-Network：

DQN 當中的神經網路模式，將依據這個模式建立兩個神經網路，一個是現實網路(Target network)、一個是估計網路(Evaluation network)，有選動作機制，有存經歷機制，有學習機制。

圖 3.2.4.2 程式碼意涵：建立 target net、eval net 和記憶，其中 gamma 為 discount factor，代表未來獎勵的重要程度，0 為只看當前，短視其利；1 為看長期，而超過 1 就會發散。

```python
class DQN(object):
    def __init__(self, n_states, n_actions, n_hidden, batch_size, lr, epsilon, gamma, target_replace_iter, memory_capacity):
        self.eval_net, self.target_net = Net(n_states, n_actions, n_hidden), Net(n_states, n_actions, n_hidden)

        self.memory = np.zeros((memory_capacity, n_states * 2 + 2))
            # 每個 memory 中的 experience 大小為 (state + next state + reward + action)
        self.optimizer = torch.optim.Adam(self.eval_net.parameters(), lr=lr) #torch的優化器
        self.loss_func = nn.MSELoss() #誤差公式
        self.memory_counter = 0     #記憶庫記數
        self.learn_step_counter = 0 # 讓 target network 知道什麼時候要更新

        self.n_states = n_states
        self.n_actions = n_actions
        self.n_hidden = n_hidden
        self.batch_size = batch_size
        self.lr = lr
        self.epsilon = epsilon
        self.gamma = gamma
        self.target_replace_iter = target_replace_iter
        self.memory_capacity = memory_capacity
```

圖 3.2.4.2 DQN 圖

圖 3.2.4.3 程式碼意涵：根據環境觀測值選擇動作的機制，會根據 ε-greedy policy 選擇動作，ε 表機率，訓練過程中有 ε 的機率 agent 會選擇亂（隨機）走，如此才有機會學習到新經驗。

```python
def choose_action(self, state):
    x = torch.unsqueeze(torch.FloatTensor(state), 0) #這裡只輸入一個 sample

    # epsilon-greedy
    if np.random.uniform() < self.epsilon: # 隨機 #選最優動作
        action = np.random.randint(0, self.n_actions)
    else: # 根據現有 policy 做最好的選擇
        actions_value = self.eval_net(x) # 以現有 eval net 得出各個 action 的分數
        action = torch.max(actions_value, 1)[1].data.numpy()[0] # 挑選最高分的 action

    return action
```

圖 3.2.4.3 DQN choose_action 圖

圖 3.2.4.4 程式碼意涵：DQN 需要存儲經驗。

```python
def store_transition(self, state, action, reward, next_state):
    # 打包 experience
    transition = np.hstack((state, [action, reward], next_state))

    # 存進 memory；舊 memory 可能會被覆蓋
    # 如果記憶庫滿了，就覆蓋老數據
    index = self.memory_counter % self.memory_capacity
    self.memory[index, :] = transition
    self.memory_counter += 1
```

圖 3.2.4.4 DQN store_transition 圖

圖 3.2.4.5 程式碼意涵：Target network 更新及學習記憶庫中的記憶。

```python
def learn(self):
    # 隨機取樣 batch_size 個 experience
    sample_index = np.random.choice(self.memory_capacity, self.batch_size)
    b_memory = self.memory[sample_index, :]
    b_state = torch.FloatTensor(b_memory[:, :self.n_states])
    b_action = torch.LongTensor(b_memory[:, self.n_states:self.n_states+1].astype(int))
    b_reward = torch.FloatTensor(b_memory[:, self.n_states+1:self.n_states+2])
    b_next_state = torch.FloatTensor(b_memory[:, -self.n_states:])

    # 計算現有 eval net 和 target net 得出 Q value 的落差
    q_eval = self.eval_net(b_state).gather(1, b_action) # 重新計算這些 experience 當下 eval net 所得出的 Q value
    q_next = self.target_net(b_next_state).detach()  # detach 才不會訓練到 target net #不進行反向傳遞誤差
    q_target = b_reward + self.gamma * q_next.max(1)[0].view(self.batch_size, 1)
    #計算這些 experience 當下 target net 所得出的 Q value
    loss = self.loss_func(q_eval, q_target)

    # Backpropagation # 計算更新 eval_net
    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()

    # 每隔一段時間 (target_replace_iter), 更新 target net，即複製 eval net 到 target net
    self.learn_step_counter += 1
    if self.learn_step_counter % self.target_replace_iter == 0:
        self.target_net.load_state_dict(self.eval_net.state_dict())
```

圖 3.2.4.5 DQN learn 圖

(3)訓練：

圖 3.2.4.6 程式碼意涵：訓練過程是先選擇動作、再存儲經驗，最後才是訓練，按照 Q-learning 的形式進行 off-policy 的更新，進行回合制更新，一個回合完了，進入下一回合，一直到將桿子立起來很久。

```
# 建立 DQN
dqn = DQN(n_states, n_actions, n_hidden, batch_size, lr, epsilon, gamma, target_replace_iter, memory_capacity)
# Collect experience # 學習
for i_episode in range(n_episodes):
    t = 0 # timestep
    rewards = 0
    state = env.reset()
    while True:
        env.render()
        # Agent takes action # 選擇 action
        action = dqn.choose_action(state) # choose an action based on DQN
        next_state, reward, done, info = env.step(action) # do the action, get the reward

        # 儲存 experience
        dqn.store_transition(state, action, reward, next_state)

        # 累積 reward
        rewards += reward

        # 有足夠 experience 後進行訓練
        if dqn.memory_counter > memory_capacity:
            dqn.learn()

        # 進入下一 state
        state = next_state
```

圖 3.2.4.6 DQN 的 dqn 圖

4 研究結果

本章將統整上一章節所述的方法，且加以評估調整參數後的情況，最後優化方法。

4.1 研究前觀測值查詢

由圖 4.1.1 可知，有兩個離散動作(向左及向右)，有四個狀態(小車位置，小車速度，桿子角度，桿尖速度)。

```
Discrete(2)
Box(4,)
```

圖 4.1.1 觀測 1 圖

由圖 4.1.2 可知每個觀測的最高取值及最低取值。

```
[4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38]
[-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38]
```

圖 4.1.2 觀測 2 圖

4.2 研究結果

4.2.1 Random Action

agent 選擇並隨機進行一個動作，從環境中獲得獎勵，從圖 4.2.1 可以看到 agent 並沒有任何學習行為，所以整體獎勵並不高。

```
Episode finished after 13 timesteps, total rewards 13.0
Episode finished after 25 timesteps, total rewards 25.0
Episode finished after 35 timesteps, total rewards 35.0
Episode finished after 12 timesteps, total rewards 12.0
Episode finished after 27 timesteps, total rewards 27.0
Episode finished after 13 timesteps, total rewards 13.0
Episode finished after 10 timesteps, total rewards 10.0
Episode finished after 13 timesteps, total rewards 13.0
Episode finished after 24 timesteps, total rewards 24.0
Episode finished after 19 timesteps, total rewards 19.0
Episode finished after 34 timesteps, total rewards 34.0
Episode finished after 15 timesteps, total rewards 15.0
Episode finished after 12 timesteps, total rewards 12.0
Episode finished after 27 timesteps, total rewards 27.0
Episode finished after 13 timesteps, total rewards 13.0
Episode finished after 21 timesteps, total rewards 21.0
Episode finished after 27 timesteps, total rewards 27.0
Episode finished after 11 timesteps, total rewards 11.0
Episode finished after 27 timesteps, total rewards 27.0
```

圖 4.2.1 Random action 結果圖

4.2.2 Hand-Made Policy

引進一個簡單的策略後，由圖 4.2.2 可以看到 agent 所獲得的整體獎勵比 Random Action 高出許多，不過 agent 依然沒有根據經驗做學習。

```
Episode finished after 41 timesteps, total rewards 41.0
Episode finished after 36 timesteps, total rewards 36.0
Episode finished after 39 timesteps, total rewards 39.0
Episode finished after 36 timesteps, total rewards 36.0
Episode finished after 53 timesteps, total rewards 53.0
Episode finished after 39 timesteps, total rewards 39.0
Episode finished after 26 timesteps, total rewards 26.0
Episode finished after 47 timesteps, total rewards 47.0
Episode finished after 34 timesteps, total rewards 34.0
Episode finished after 36 timesteps, total rewards 36.0
Episode finished after 36 timesteps, total rewards 36.0
Episode finished after 27 timesteps, total rewards 27.0
Episode finished after 60 timesteps, total rewards 60.0
Episode finished after 36 timesteps, total rewards 36.0
Episode finished after 50 timesteps, total rewards 50.0
Episode finished after 51 timesteps, total rewards 51.0
Episode finished after 44 timesteps, total rewards 44.0
Episode finished after 40 timesteps, total rewards 40.0
Episode finished after 36 timesteps, total rewards 36.0
```

圖 4.2.2 Hand-Made Policy 結果圖

4.2.3 Q-table

首先嘗試將車位置、車速度、桿尖速度及桿子角度的存儲桶(bucket)設置非1，在圖 4.2.3.1 也可看出桿子角度只需 6 個 bucket，車位置、車速度、桿尖速度只需 3 個 bucket。

```
if (x < -0.8)                          box = 0;
else if (x < 0.8)                      box = 1;
else                                   box = 2;

if (x_dot < -0.5)                      ;
else if (x_dot < 0.5)                  box += 3;
else                                   box += 6;

if (theta < -six_degrees)              ;
else if (theta < -one_degree)          box += 9;
else if (theta < 0)                    box += 18;
else if (theta < one_degree)           box += 27;
else if (theta < six_degrees)          box += 36;
else                                   box += 45;

if (theta_dot < -fifty_degrees)         ;
else if (theta_dot < fifty_degrees)    box += 54;
else                                   box += 108;

return(box);
```

圖 4.2.3.1 bucket 需求圖[6]

當把車位置、車速度、桿尖速度、出桿子角度都調成圖 4.2.3.1 所需後，可由圖 4.2.3.2 看出其整體獎勵都偏低。

```
Episode finished after 11 timesteps, total rewards 11.0
Episode finished after 14 timesteps, total rewards 14.0
Episode finished after 24 timesteps, total rewards 24.0
Episode finished after 12 timesteps, total rewards 12.0
Episode finished after 13 timesteps, total rewards 13.0
Episode finished after 14 timesteps, total rewards 14.0
Episode finished after 10 timesteps, total rewards 10.0
Episode finished after 12 timesteps, total rewards 12.0
Episode finished after 13 timesteps, total rewards 13.0
Episode finished after 18 timesteps, total rewards 18.0
Episode finished after 12 timesteps, total rewards 12.0
Episode finished after 10 timesteps, total rewards 10.0
Episode finished after 10 timesteps, total rewards 10.0
Episode finished after 24 timesteps, total rewards 24.0
Episode finished after 34 timesteps, total rewards 34.0
Episode finished after 13 timesteps, total rewards 13.0
Episode finished after 28 timesteps, total rewards 28.0
Episode finished after 27 timesteps, total rewards 27.0
Episode finished after 11 timesteps, total rewards 11.0
```

圖 4.2.3.2 bucket 需求結果 1 圖

接著可以從 Cart-pole 動畫發現，在平衡桿位時，通常不會漂移那麼遠，所

以將車位置及車速度視為較不重要的特徵，存儲桶(bucket)設置為 1，而桿子角度及桿尖速度設置非 1，由圖 4.2.3.3 可以看出，在訓練後期，agent 已經學會如何最大化自己的獎勵，也就是維持住小車上的桿子了。

```
Episode finished after 113 timesteps, total rewards 113.0
Episode finished after 26 timesteps, total rewards 26.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
```

圖 4.2.3.3 bucket 需求結果 2 圖

最一開始都是以學習率為 0.01 去訓練整個模型，現在更改學習率為 0.9，可以由 lambda 這個函數發現學習率幾乎不會隨時間遞減，圖 4.2.3.5 可看出所跑出的整體獎勵並不高。

```
# 學習相關的常數： 反複試驗決定的因素
get_epsilon = lambda i: max(0.01, min(1, 1.0 - math.log10((i+1)/25))) # epsilon-greedy;隨時間遞減
get_lr = lambda i: max(0.9, min(0.5, 1.0 - math.log10((i+1)/25))) # learning rate; 隨時間遞減
gamma = 0.99 # reward discount factor
```

圖 4.2.3.4 lambda 函數內調整圖

```
Episode finished after 9 timesteps, total rewards 9.0
Episode finished after 11 timesteps, total rewards 11.0
Episode finished after 8 timesteps, total rewards 8.0
Episode finished after 9 timesteps, total rewards 9.0
Episode finished after 9 timesteps, total rewards 9.0
Episode finished after 10 timesteps, total rewards 10.0
Episode finished after 18 timesteps, total rewards 18.0
Episode finished after 34 timesteps, total rewards 34.0
Episode finished after 16 timesteps, total rewards 16.0
Episode finished after 11 timesteps, total rewards 11.0
Episode finished after 9 timesteps, total rewards 9.0
Episode finished after 9 timesteps, total rewards 9.0
Episode finished after 9 timesteps, total rewards 9.0
Episode finished after 16 timesteps, total rewards 16.0
Episode finished after 13 timesteps, total rewards 13.0
Episode finished after 11 timesteps, total rewards 11.0
Episode finished after 18 timesteps, total rewards 18.0
Episode finished after 44 timesteps, total rewards 44.0
Episode finished after 28 timesteps, total rewards 28.0
```

圖 4.2.3.5 lambda 函數內調整結果圖

4.2.4 DQN

整個訓練結果可以由圖 4.2.4.1 發現，就算把 episode 調成 4000 次，學習過程也是看不出有收斂的。

```
Episode finished after 11 timesteps, total rewards 11.0
Episode finished after 9 timesteps, total rewards 9.0
Episode finished after 8 timesteps, total rewards 8.0
Episode finished after 10 timesteps, total rewards 10.0
Episode finished after 10 timesteps, total rewards 10.0
Episode finished after 10 timesteps, total rewards 10.0
Episode finished after 10 timesteps, total rewards 10.0
Episode finished after 9 timesteps, total rewards 9.0
Episode finished after 9 timesteps, total rewards 9.0
Episode finished after 12 timesteps, total rewards 12.0
Episode finished after 11 timesteps, total rewards 11.0
Episode finished after 10 timesteps, total rewards 10.0
Episode finished after 10 timesteps, total rewards 10.0
Episode finished after 11 timesteps, total rewards 11.0
Episode finished after 12 timesteps, total rewards 12.0
Episode finished after 10 timesteps, total rewards 10.0
Episode finished after 9 timesteps, total rewards 9.0
Episode finished after 9 timesteps, total rewards 9.0
Episode finished after 10 timesteps, total rewards 10.0
```

圖 4.2.4.1 DQN 原始結果圖

接下來回到 OpenAI gym 給的環境：

```
if not done:
    reward = 1.0
elif self.steps_beyond_done is None:
    # Pole just fell!
    self.steps_beyond_done = 0
    reward = 1.0
else:
    if self.steps_beyond_done == 0:
        logger.warn("You are calling 'step()' even though this environment has already returned done = True. You should always call
    self.steps_beyond_done += 1
    reward = 0.0
```

圖 4.2.4.2 OpenAI gym_reward 圖

桿子倒了之後獲得 0 分，其他情況下獲得 1 分，這麼缺乏資訊的獎勵，得有勞 agent 嘗試好幾回才能學會怎麼維持桿子平衡。

因此自己建立獎勵分配方法，讓獎勵提供更多資訊，很直覺的，柱子的角度越正，獎勵應該越大，小車保持在中間，那麼小車跟中間的距離越小，獎勵也應該越大。

```
if CHEAT:
    x, v, theta, omega = next_state
    r1 = (env.x_threshold - abs(x)) / env.x_threshold - 0.8 # reward 1: 小車離中間越近越好
    r2 = (env.theta_threshold_radians - abs(theta)) / env.theta_threshold_radians - 0.5 # reward 2: 柱子越正越好
    reward = r1 + r2
```

圖 4.2.4.3 獎勵分配方法圖

由圖 4.2.4.4 可以看出，當獎勵機制改變且把 episode 調到 400 次，已經能有不錯的結果了。由於方法的不同，這邊不能以整體獎勵值來比較，主要是看 timesteps，也就是桿子維持不倒的時間，從 Cart-pole 動畫也可以看出在訓練後期，桿子已經可以穩定地站立在小車上了。

```
Episode finished after 310 timesteps, total rewards 176.47080394926073
Episode finished after 201 timesteps, total rewards 112.57292323476638
Episode finished after 25 timesteps, total rewards 10.474243736333788
Episode finished after 35 timesteps, total rewards 16.50747909161164
Episode finished after 482 timesteps, total rewards 198.8238345028323
Episode finished after 628 timesteps, total rewards 355.42020145843276
Episode finished after 1045 timesteps, total rewards 380.44824649249875
Episode finished after 774 timesteps, total rewards 254.32957692366915
Episode finished after 593 timesteps, total rewards 251.21114123671276
Episode finished after 290 timesteps, total rewards 81.71939433045574
Episode finished after 101 timesteps, total rewards 21.828462049137904
Episode finished after 239 timesteps, total rewards 95.1841632159812
Episode finished after 52 timesteps, total rewards 25.52748891375813
Episode finished after 79 timesteps, total rewards 41.26481874299716
Episode finished after 198 timesteps, total rewards 73.517600954537
Episode finished after 239 timesteps, total rewards 95.71996255103255
Episode finished after 169 timesteps, total rewards 27.158818000381196
Episode finished after 123 timesteps, total rewards 10.605630061985527
Episode finished after 41 timesteps, total rewards 18.35730630605925
```

圖 4.2.4.4 獎勵分配方法結果圖

將學習率調成 0.9，可由圖 4.2.4.5 看出整體獎勵並不高。

```
Episode finished after 10 timesteps, total rewards 2.6259487210414036
Episode finished after 9 timesteps, total rewards 2.14708708572077
Episode finished after 10 timesteps, total rewards 1.3304717192306754
Episode finished after 9 timesteps, total rewards 1.552073451328278
Episode finished after 16 timesteps, total rewards 6.631195305460702
Episode finished after 14 timesteps, total rewards 5.228845570004568
Episode finished after 10 timesteps, total rewards 3.1756375808706254
Episode finished after 10 timesteps, total rewards 3.0551744209276546
Episode finished after 16 timesteps, total rewards 6.845023733841264
Episode finished after 10 timesteps, total rewards 2.8672335524491332
Episode finished after 12 timesteps, total rewards 3.885800032277818
Episode finished after 11 timesteps, total rewards 2.918503973237407
Episode finished after 19 timesteps, total rewards 4.580812112248001
Episode finished after 26 timesteps, total rewards 5.798626282646678
Episode finished after 12 timesteps, total rewards 3.3947356540740516
Episode finished after 10 timesteps, total rewards 2.991036368022514
Episode finished after 9 timesteps, total rewards 2.7328471196123276
Episode finished after 9 timesteps, total rewards 1.264859470776846
Episode finished after 11 timesteps, total rewards 2.7559828637791455
```

圖 4.2.4.5 DQN 學習率 0.9 結果圖

將學習率調成 0.0001，可由圖 4.2.4.6 看出整體獎勵也是偏低，從 Cart-pole 動畫可以發現桿子及車的速度都動得非常快。

```
Episode finished after 10 timesteps, total rewards 2.213562820968987
Episode finished after 11 timesteps, total rewards 2.9075629411021424
Episode finished after 10 timesteps, total rewards 2.96046033779109
Episode finished after 9 timesteps, total rewards 2.5848233723375964
Episode finished after 9 timesteps, total rewards 2.063705771991575
Episode finished after 9 timesteps, total rewards 1.4290311887770049
Episode finished after 9 timesteps, total rewards 1.3218921850725756
Episode finished after 10 timesteps, total rewards 2.390368333312675
Episode finished after 9 timesteps, total rewards 2.225043307032972
Episode finished after 9 timesteps, total rewards 2.464841089370724
Episode finished after 10 timesteps, total rewards 2.476343025661195
Episode finished after 10 timesteps, total rewards 2.0510191710156924
Episode finished after 10 timesteps, total rewards 2.1776017847577727
Episode finished after 10 timesteps, total rewards 2.9154091702826657
Episode finished after 11 timesteps, total rewards 2.752275973671329
Episode finished after 9 timesteps, total rewards 2.517859200460433
Episode finished after 9 timesteps, total rewards 0.9472033923648783
Episode finished after 9 timesteps, total rewards 1.1881364448851783
Episode finished after 8 timesteps, total rewards 1.4046355676705642
```

圖 4.2.4.6 DQN 學習率 0.0001 結果圖

從上一章節可以知道 ε 代表 agent 會選擇亂（隨機）走的機率，將 ε 調成 0.9，agent 學習新知的機率大過原本的 0.1，可以由圖 4.2.4.7 看出整體獎勵偏低，表示 agent 花了大部分的時間都在學習新知。

```
Episode finished after 13 timesteps, total rewards 4.5263418835972296
Episode finished after 20 timesteps, total rewards 5.20442620415754
Episode finished after 34 timesteps, total rewards 12.834720753429728
Episode finished after 39 timesteps, total rewards 4.008596082043514
Episode finished after 46 timesteps, total rewards 13.75422653813691
Episode finished after 48 timesteps, total rewards 16.59772943358283
Episode finished after 24 timesteps, total rewards 7.947290347480627
Episode finished after 67 timesteps, total rewards 23.53113972365001
Episode finished after 13 timesteps, total rewards 3.3761423028764717
Episode finished after 28 timesteps, total rewards 7.319743000716445
Episode finished after 21 timesteps, total rewards 7.084532190904521
Episode finished after 70 timesteps, total rewards 16.55385337326199
Episode finished after 27 timesteps, total rewards 5.822776579180058
Episode finished after 21 timesteps, total rewards 6.78711575733506
Episode finished after 11 timesteps, total rewards 1.5170323784233877
Episode finished after 27 timesteps, total rewards 5.261511706820194
Episode finished after 10 timesteps, total rewards 1.6498494912643624
Episode finished after 47 timesteps, total rewards 18.126641905955225
Episode finished after 15 timesteps, total rewards 3.028032503694039
```

圖 4.2.4.7 DQN ε 為 0.9 結果圖

當把 ε 調成 0.0001，可以從 Cart-pole 動畫發現一直到訓練後期，車子還是會往旁邊飄移，雖然從圖 4.2.4.8 看出整體獎勵還是比 ε 為 0.9 高很多，但是由於新知不足，所以沒有辦法一值讓桿子維持不倒的時間最大化。

```
Episode finished after 2550 timesteps, total rewards 673.1408683484697
Episode finished after 20 timesteps, total rewards 6.434723694033325
Episode finished after 11 timesteps, total rewards 3.0293895649818636
Episode finished after 26 timesteps, total rewards 1.6570338437450791
Episode finished after 23 timesteps, total rewards 1.4655084282650974
Episode finished after 37 timesteps, total rewards 9.373405134097913
Episode finished after 77 timesteps, total rewards 6.766916623999093
Episode finished after 8 timesteps, total rewards 1.1432287677601858
Episode finished after 36 timesteps, total rewards 5.61015606542991
Episode finished after 10 timesteps, total rewards 2.279845857628803
Episode finished after 58 timesteps, total rewards 3.3947801132912763
Episode finished after 10 timesteps, total rewards 2.63084016048811
Episode finished after 146 timesteps, total rewards 3.6342047231428483
Episode finished after 32 timesteps, total rewards 12.01149202687337
Episode finished after 215 timesteps, total rewards 49.96555143174603
Episode finished after 310 timesteps, total rewards 99.71439761591515
Episode finished after 507 timesteps, total rewards 205.61940885496833
Episode finished after 2093 timesteps, total rewards 754.0736000182288
Episode finished after 1007 timesteps, total rewards 353.4537934889371
```

圖 4.2.4.8 DQN ε 為 0.0001 結果圖

5 結論與未來展望

本章節將統整前面章節的方法及結果，做一個精確地比較，且對於其限制會在本章節提到，以及其未來方向。

5.1 結論

表 5.1.1 為統整 4 個方法的優缺點之比較。

表 5.1.1 4 個方法比較表

方法	優點	缺點
Random Action	程式碼較簡單，易懂，且速度較快。	Agent並沒有任何的學習，所獲得的整體獎勵非常低。
Hand-Made Policy	相比於Random Action，程式加入了一些簡單的策略，使整體獎勵比前一個方法高。	Agent依然沒有根據經驗做學習。
Q-table	在訓練後期，agent已經學會如何最大化自己的獎勵，也就是維持住小車上的桿子了。	Table的大小有限，有容量限制。
DQN	Neural network可以搭配不同變形，從龐大的狀態空間中自動提取特徵，是僵化的Q-table做不到的。	較複雜，且速度較慢。

表 5.1.2 為 Q-table 的各項調整結果分析。

表 5.1.2 Q-table 結果表

Q-table	結果
車位置、車速度、桿子角度、桿尖速度的bucket設置分別為[3, 3, 6, 3]。	整體獎勵都偏低。
觀察Cart-pole動畫可知道車位置、車速度這兩種特徵較不重要，將其bucket設置為1。	在訓練後期，可以看到agent已經最大化自己的獎勵。
更改學習率為0.9，可以由lambda這個函數發現學習率幾乎不會隨時間遞減。	結果所跑出的整體獎勵並不高。

表 5.1.3 為 DQN 的各項調整結果分析。

表 5.1.3 DQN 結果表

DQN	結果
最原始未做任何修改的DQN。	整體獎勵偏低，學習過程也看不出有收斂的，
查看OpenAI gym給的環境，自己建立一個獎勵分配機制。	方法的不同，以timestep比較，從Cart-pole動畫也可以看出在訓練後期，桿子已經可以穩定地站立在小車上了。
將學習率調成0.9。	可以發現整體獎勵非常低。
將學習率調成0.0001。	整體獎勵也是偏低，從Cart-pole動畫以可以發現桿子及車的速度都動得非常快。。
將ε調成0.9。	整體獎勵偏低，表示agent花了大部分的時間都在學習新知。
將ε調成0.0001。	從Cart-pole動畫發現一直到訓練後期，車子還是會往旁邊飄移，雖然整體獎勵還是比ε為0.9高很多，但是由於新知不足，所以沒有辦法一值讓桿子維持不倒的時間最大化。

5.2 限制

此專題的限制在於學生對程式碼的不熟悉，造成學習緩慢，且有些程式碼修改做的並不到位，或是有想法卻做不出來的種種困境，只能多方參考網路上的各種前人寫的程式碼。

5.3 未來展望

即使桿子在整個情節中都保持相對直立，但小車仍然朝著右側傾斜，這意味著神經網絡還利用了-由於我們的評估時間窗口足夠短，以至小車不能在水平方向上漂移太遠，如果想無限期地將其保持在中心，那麼將不得不懲罰水平位移，即除了獎勵機制還需加上懲罰機制，使 agent 意識到整個環境並不如一開始的單純，環境的複雜化將導致 agent 有更多學習的機會。

誌謝

謝謝 IIE 全體師生，特別感謝助教的幫忙，教導許多實用的觀念及程式。

參考資料

[1] https://gym.openai.com/

[2] https://en.wikipedia.org/wiki/Markov_model

[3] https://en.wikipedia.org/wiki/Hidden_Markov_model

[4] https://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/

[5] https://morvanzhou.github.io/tutorials/machine-learning/torch/4-05-DQN/

[6] http://pages.cs.wisc.edu/~finton/qcontroller.html