IIE Final Project

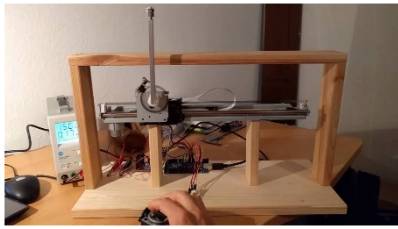
Cart-Pole System
木棒台車平衡問題

Presenter：王禹晴

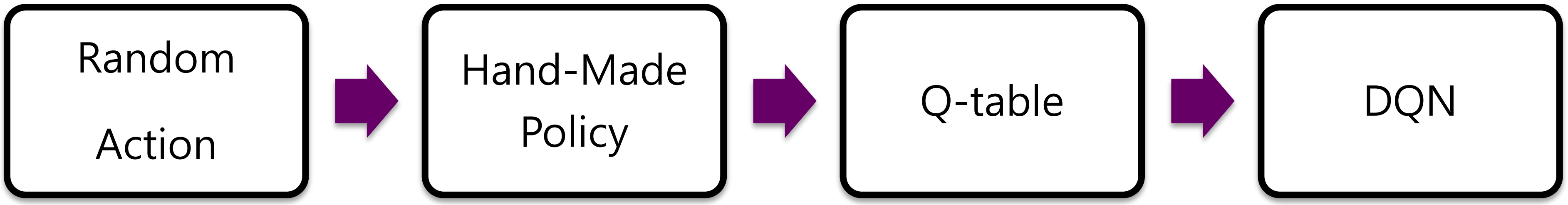
2020

5W1H分析

Why	為了更了解強化學習裡的Q-learning演算法，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。
Where	本專題所假想應用於實務上真正的木棒台車系統。
When	學習者想用簡單的例子學習Q-learning時。
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。
How	利用4個不同的方法(Random Action, Hand-Made Policy, Q-table, DQN)及參數的改變快速明白木棒台車平衡問題的運作模式。

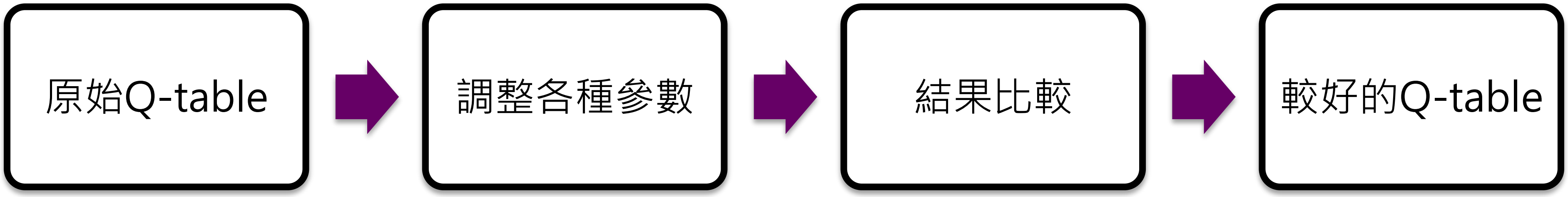


Process(1/2)

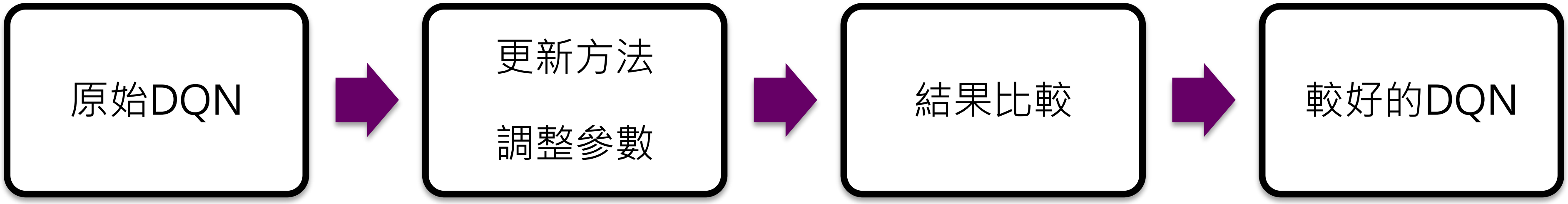


Process(2/2)

Q-table:

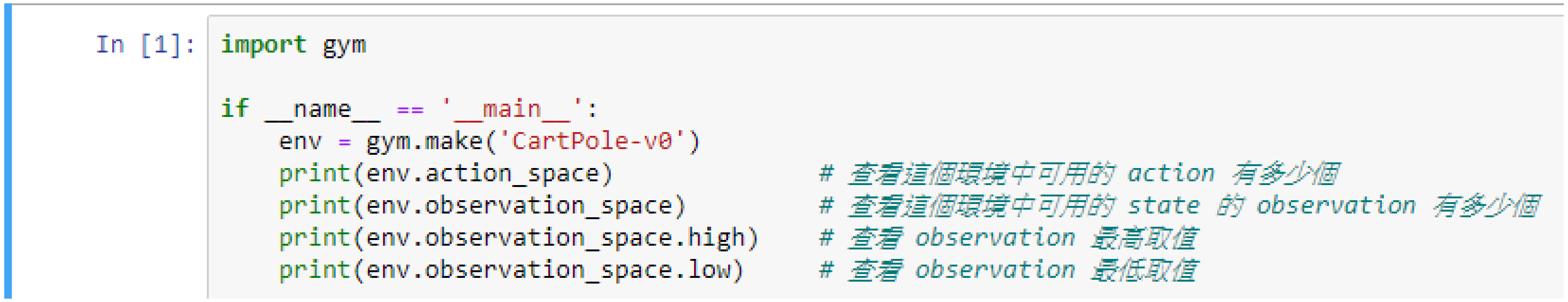


DQN:



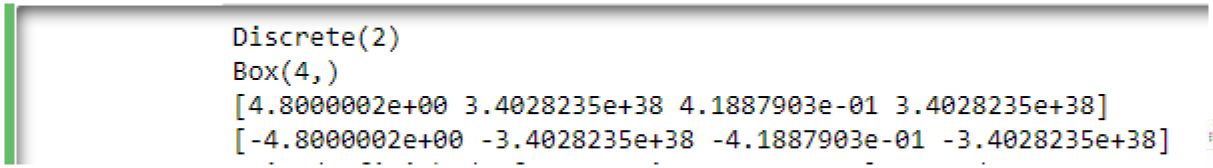
查詢

- 先查看需要的觀測數與各項範圍。

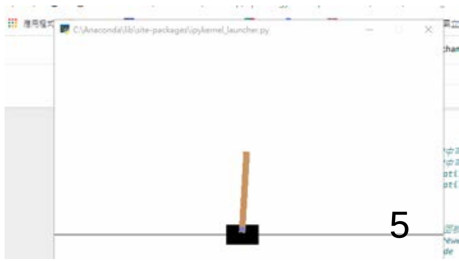


```
In [1]: import gym

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

- 有兩個離散值動作(向左或向右)
  有四個輸入的狀態變量(車位置、車速度、桿子角度、桿尖速度)

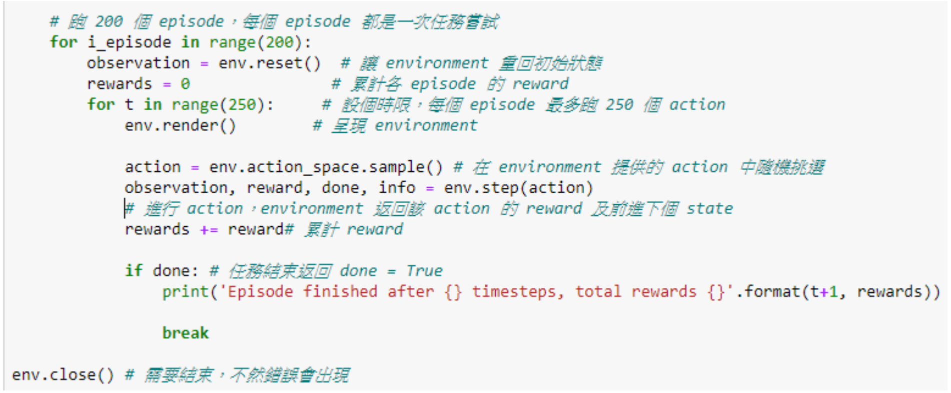


```
Discrete(2)
Box(4,)
[4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38]
[-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38]
```

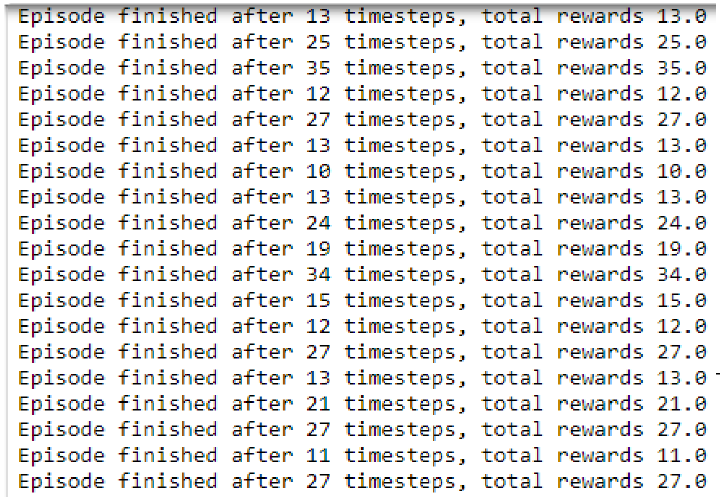


Random Action

- 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。

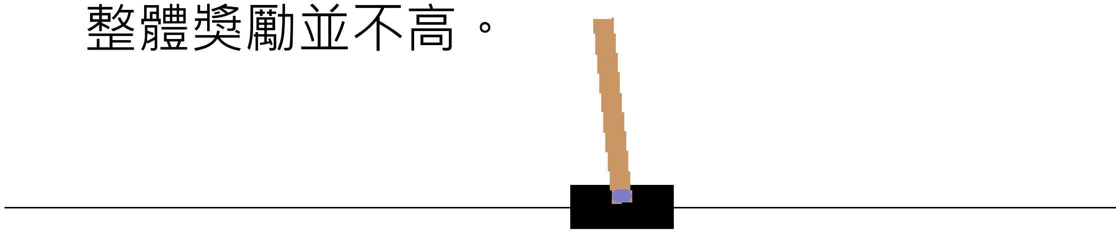


```
    # 跑 200 個 episode，每個 episode 都是一次任務嘗試
    for i_episode in range(200):
        observation = env.reset()  # 讓 environment 重回初始狀態
        rewards = 0                # 累計各 episode 的 reward
        for t in range(250):      # 設個時限，每個 episode 最多跑 250 個 action
            env.render()          # 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

            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

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

                break

env.close() # 需要結束，不然錯誤會出現
```



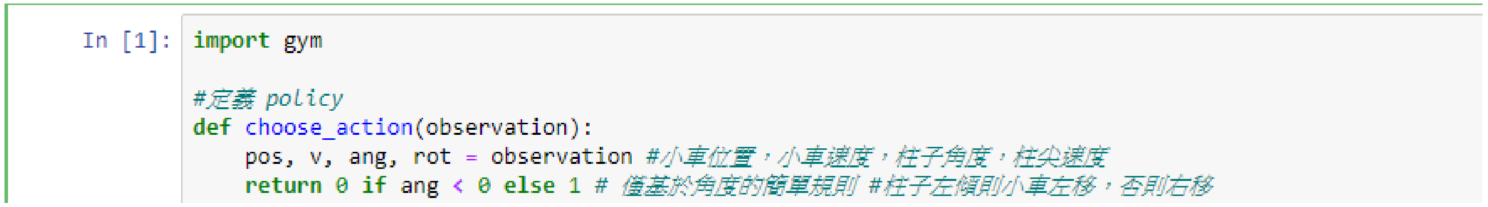
```
Episode finished after 13 timesteps, total rewards 13.0
Episode finished after 25 timesteps, total rewards 25.0
Episode finished after 35 timesteps, total rewards 35.0
Episode finished after 12 timesteps, total rewards 12.0
Episode finished after 27 timesteps, total rewards 27.0
Episode finished 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
```

- Agent並沒有任何學習行為，所以整體獎勵並不高。

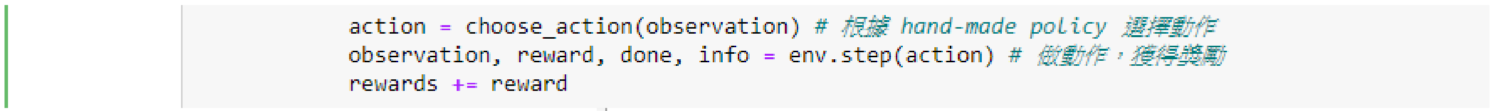


Hand-made Policy

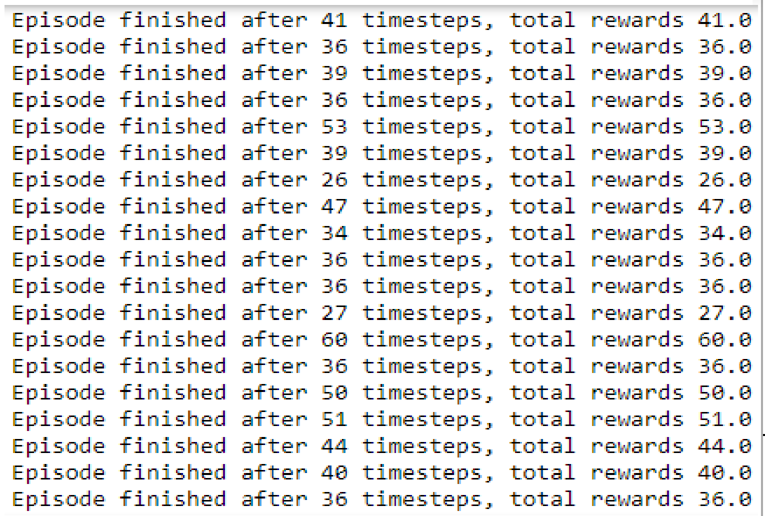
- 為了讓agent不會走得太無腦，所以引進一個簡單的策略：
  如果桿子向左傾（角度 < 0），則小車左移以維持平衡，否則右移。



```
In [1]: import gym

        #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
```

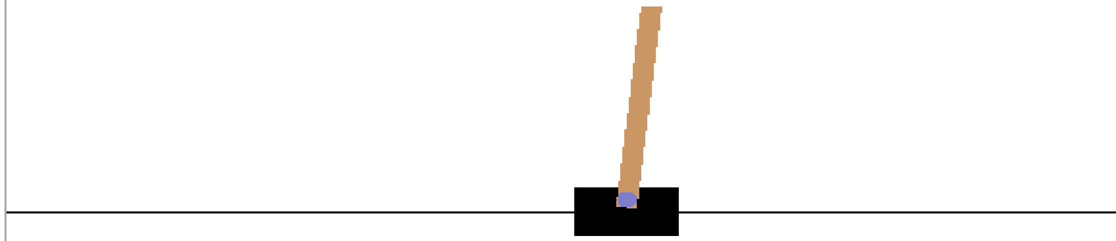


```
            action = choose_action(observation) # 根據 hand-made policy 選擇動作
            observation, reward, done, info = env.step(action) # 做動作，獲得獎勵
            rewards += reward
```



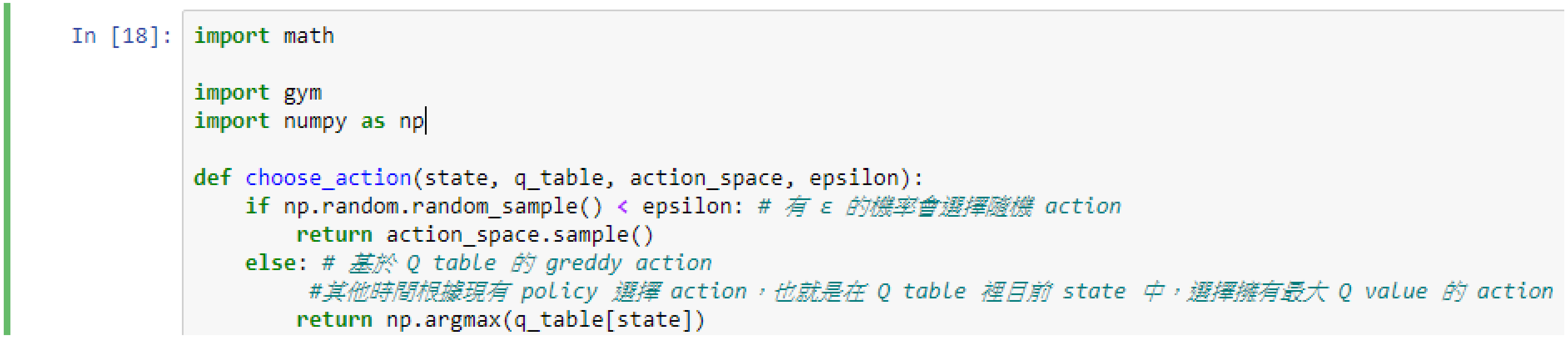
```
Episode finished after 41 timesteps, total rewards 41.0
Episode finished after 36 timesteps, total rewards 36.0
Episode finished after 39 timesteps, total rewards 39.0
Episode finished after 36 timesteps, total rewards 36.0
Episode finished after 53 timesteps, total rewards 53.0
Episode finished 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
```

- 整體獎勵比Random Action高一些，
  但是agent依然沒有根據經驗做學習。



Q-table(1/4)

- Agent要藉由一次次跟環境互動中獲得的獎勵來學習 Q 函數。
- 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。

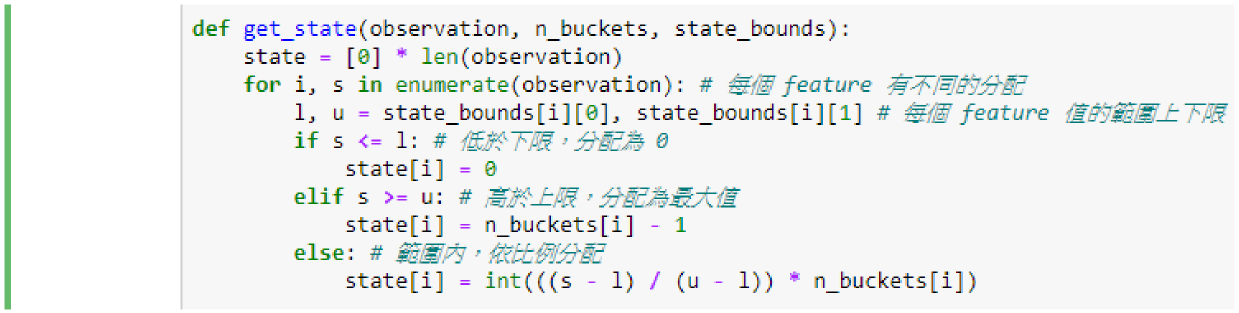


```
In [18]: import math

         import gym
         import numpy as np

         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])
```

- ε-greedy是一種在探索和開發間取得平衡的方法。
- ε的機率會嘗試新動作，而(1 - ε)的機率會根據現有策略做決策。

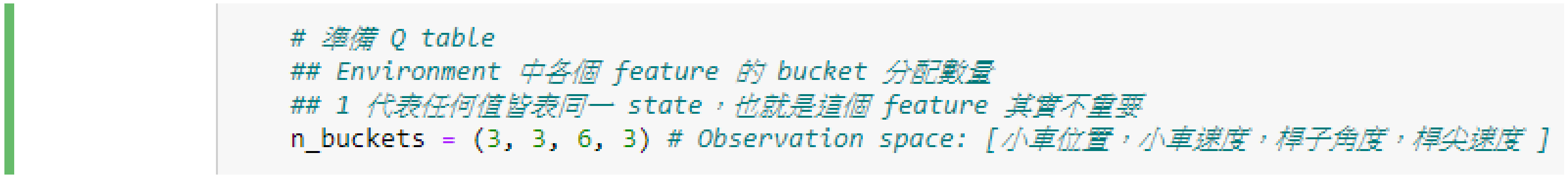
Q-table(2/4)

- 特徵是連續值，不適合作為一個表格的輸入，所以離散化為多個存儲桶。



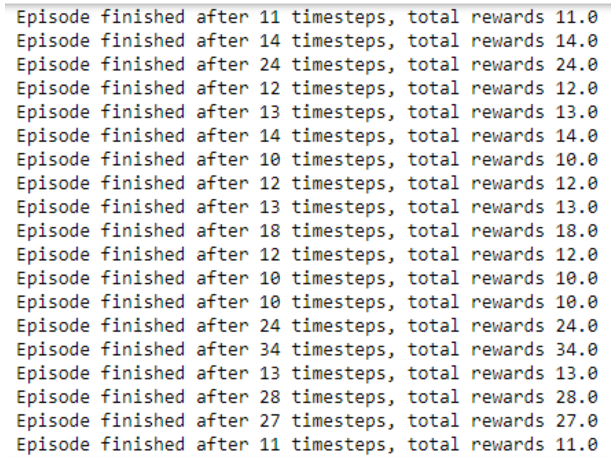
```python
def get_state(observation, n_buckets, state_bounds):
    state = [0] * len(observation)
    for i, s in enumerate(observation): # 每個 feature 有不同的分配
        l, u = state_bounds[i][0], state_bounds[i][1] # 每個 feature 值的範圍上下限
        if s <= l: # 低於下限，分配為 0
            state[i] = 0
        elif 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])
```

- 從 Bucket需求可知車位置、車速度、桿尖速度只需3個bucket，桿子角度需要6個bucket。



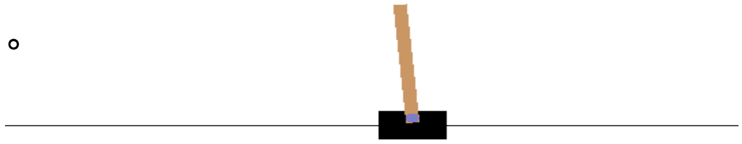
```python
# 準備 Q table
## Environment 中各個 feature 的 bucket 分配數量
## 1 代表任何值皆表同一 state，也就是這個 feature 其實不重要
n_buckets = (3, 3, 6, 3) # Observation space: [小車位置，小車速度，桿子角度，桿尖速度 ]
```

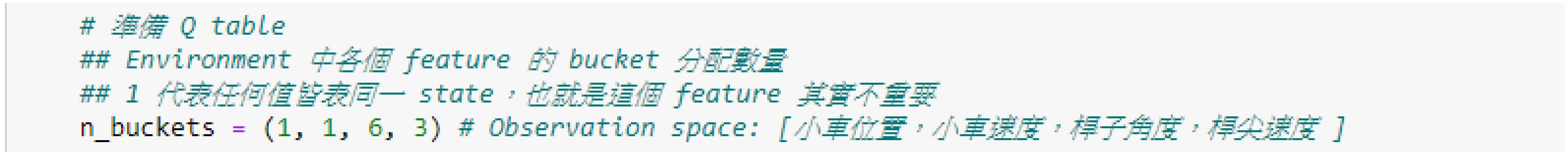
Q-table(3/4)



```
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
```

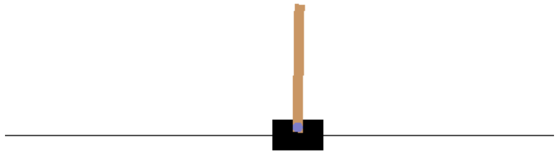
- 發現整體獎勵並不是太好，所以從Cart-Pole的動畫可以看出在平衡桿位時，通常不會漂移那麼遠，所以將車位置及車速度視為較不重要的特徵。

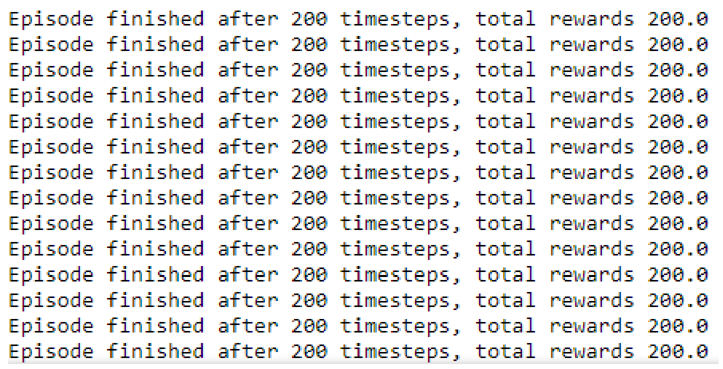




```
# 準備 Q table
## Environment 中各個 feature 的 bucket 分配數量
## 1 代表任何值皆表同一 state，也就是這個 feature 其實不重要
n_buckets = (1, 1, 6, 3) # Observation space: [小車位置，小車速度，桿子角度，桿尖速度 ]
```

- 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。

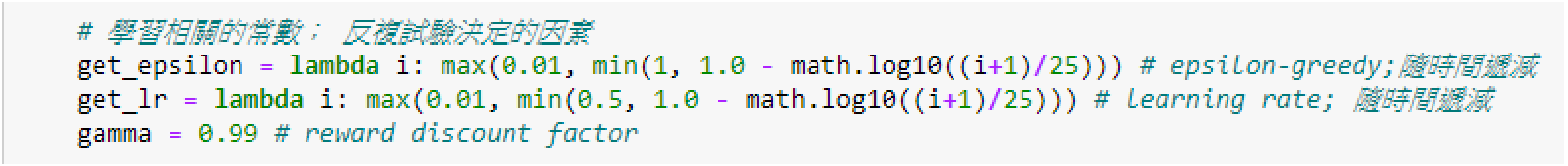




```
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode finished after 200 timesteps, total rewards 200.0
Episode 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
```

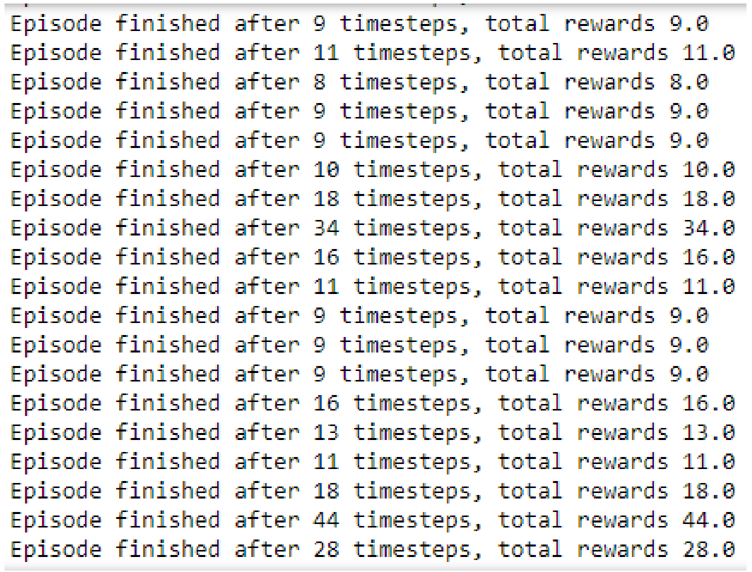
Q-table(4/4)

- 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。



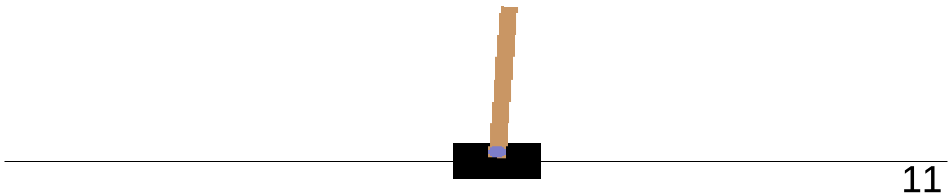
```
# 學習相關的常數： 反複試驗決定的因素
get_epsilon = lambda i: max(0.01, min(1, 1.0 - math.log10((i+1)/25))) # epsilon-greedy;隨時間遞減
get_lr = lambda i: max(0.01, min(0.5, 1.0 - math.log10((i+1)/25))) # learning rate; 隨時間遞減
gamma = 0.99 # reward discount factor
```

- 由lambda這個函數更改學習率為0.9，造成其幾乎不會隨時間遞減。



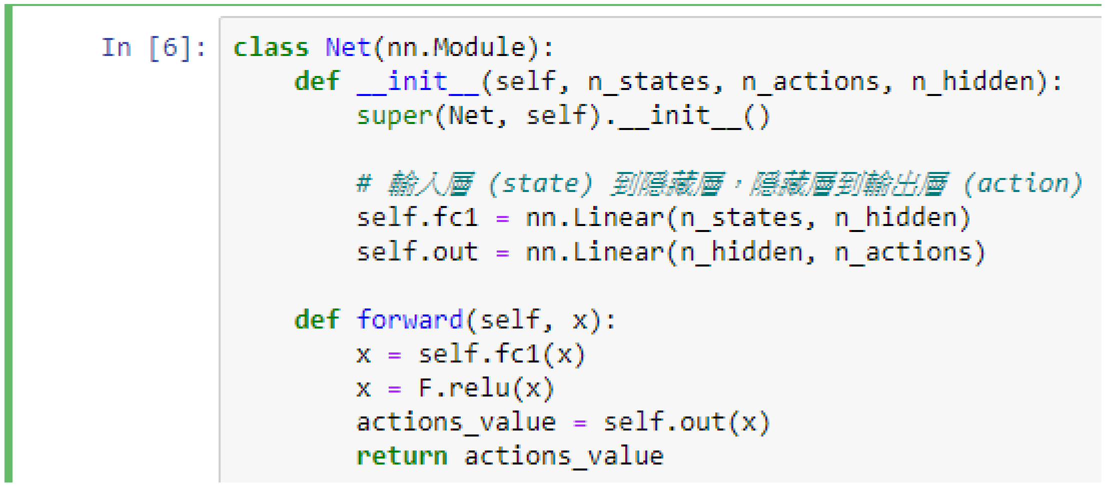
```
Episode finished after 9 timesteps, total rewards 9.0
Episode finished after 11 timesteps, total rewards 11.0
Episode finished after 8 timesteps, total rewards 8.0
Episode finished after 9 timesteps, total rewards 9.0
Episode finished after 9 timesteps, total rewards 9.0
Episode finished after 10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

- 發現整體獎勵不高，代表新資訊重要程度的權重太高，導致agent一直在學習新知而忽略經驗。



DQN(1/9)

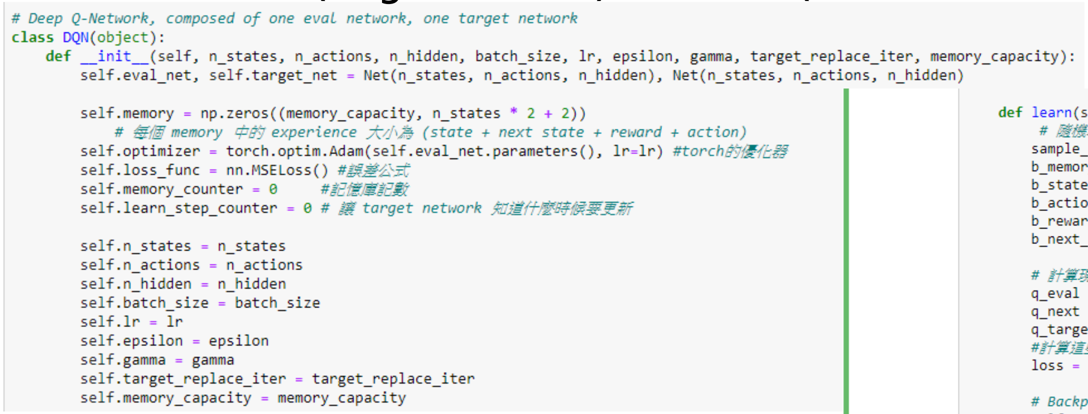
- 建立Network：把狀態傳入後，得出每個動作的分數，分數越高的動作越有機會被挑選。



```
In [6]: 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
```

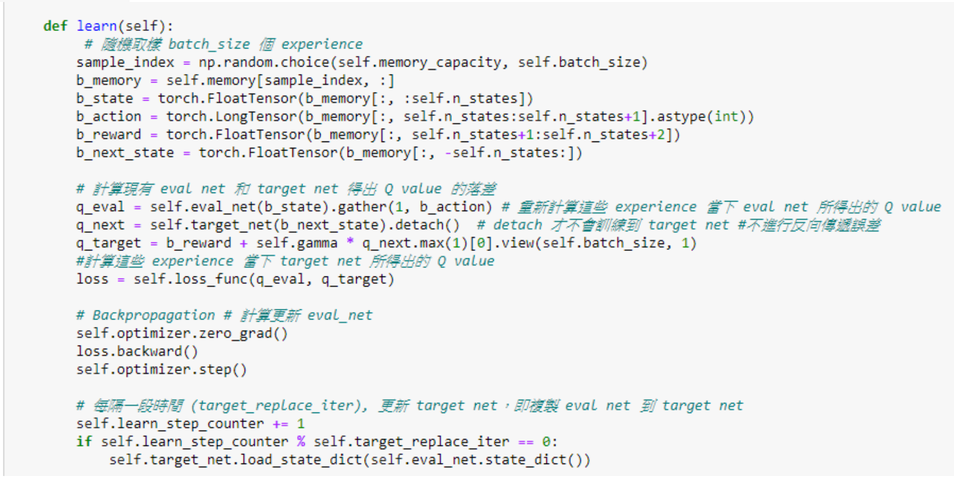
DQN(2/9)

- 建立Deep Q-Network：
  - **現實網路(Target network)、估計網路(Evaluation network)**



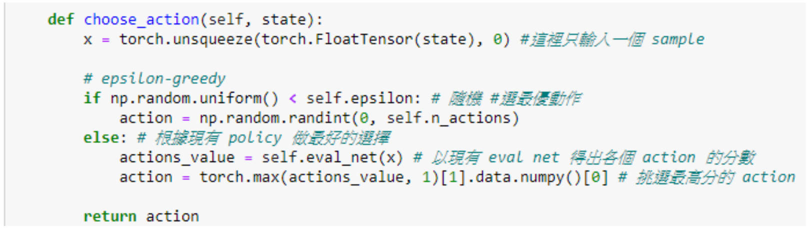
```python
# Deep Q-Network, composed of one eval network, one target network
class DQN(object):
    def __init__(self, n_states, n_actions, n_hidden, batch_size, lr, epsilon, gamma, target_replace_iter, memory_capacity):
        self.eval_net, self.target_net = Net(n_states, n_actions, n_hidden), Net(n_states, 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
```

- **學習機制**



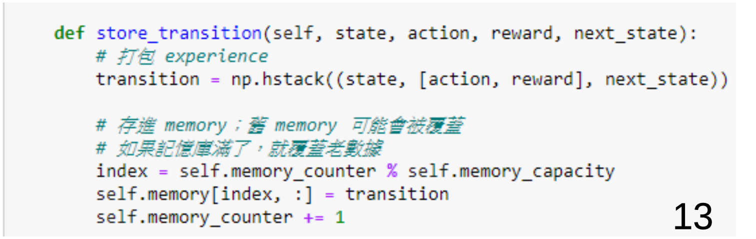
```python
    def learn(self):
         # 隨機取樣 batch_size 個 experience
        sample_index = np.random.choice(self.memory_capacity, self.batch_size)
        b_memory = self.memory[sample_index, :]
        b_state = torch.FloatTensor(b_memory[:, :self.n_states])
        b_action = torch.LongTensor(b_memory[:,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

- **選動作機制**



```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
```

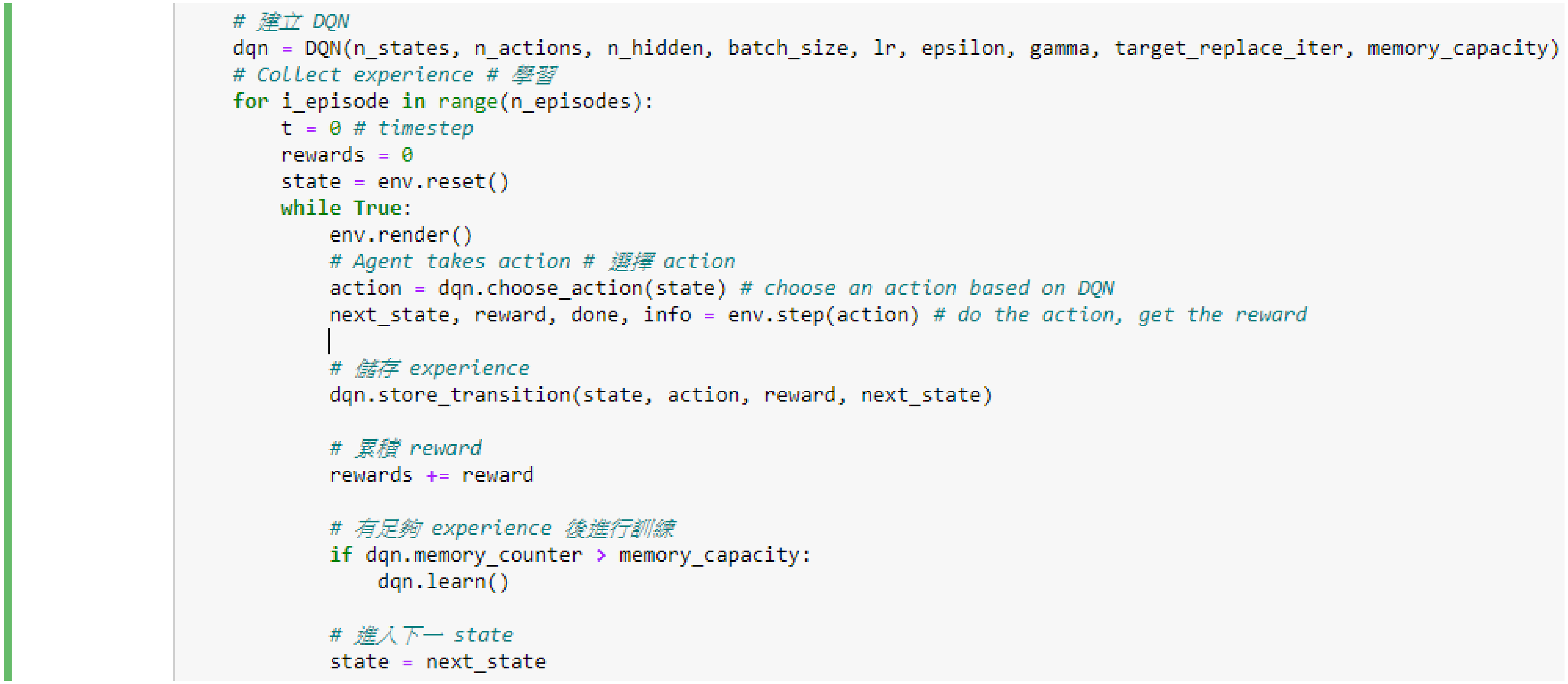
- **存經歷機制**



```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
```

DQN(3/9)

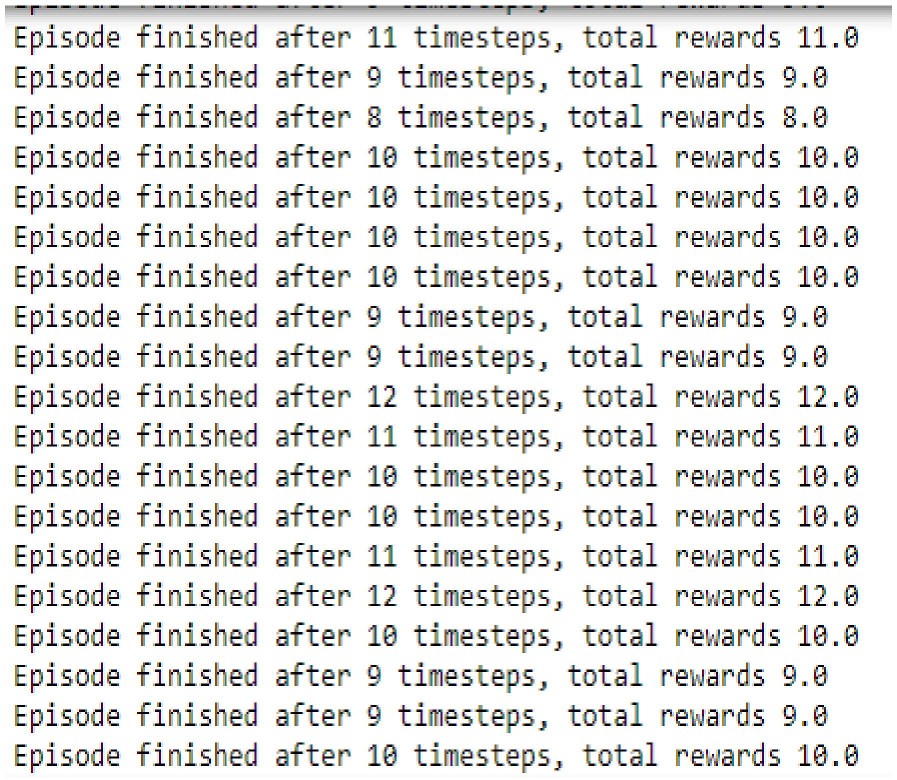
- 訓練：先選擇動作、再存儲經驗，最後才是訓練。



```
# 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
```

DQN(4/9)

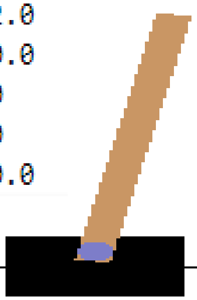
- 最後訓練完的結果。



```
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
```

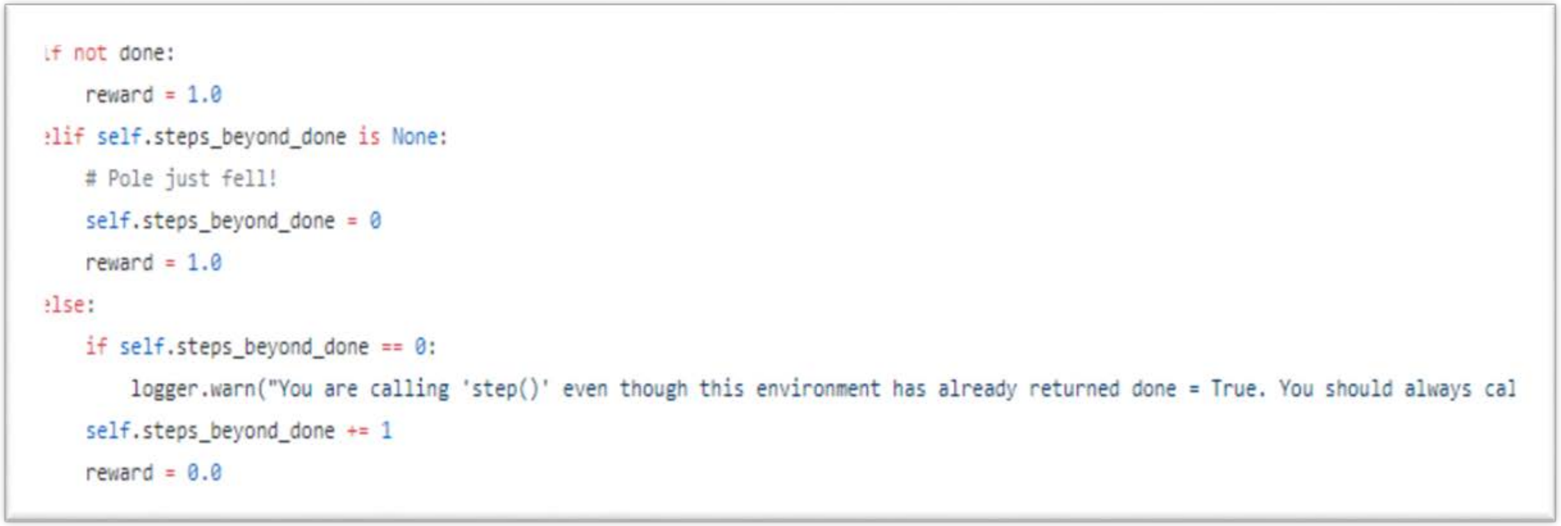
- 太糟了吧!!!!!!!

- 不管episode調成幾次，也看不出學習過程有收斂。



DQN(5/9)

- 回到OpenAI gym給的環境：

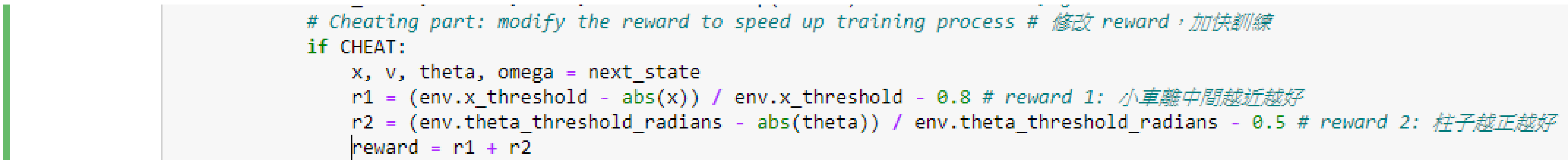


```
if not done:
    reward = 1.0
elif self.steps_beyond_done is None:
    # Pole just fell!
    self.steps_beyond_done = 0
    reward = 1.0
else:
    if self.steps_beyond_done == 0:
        logger.warn("You are calling 'step()' even though this environment has already returned done = True. You should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

- 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，agent嘗試好幾回才能學會怎麼維持桿子平衡。

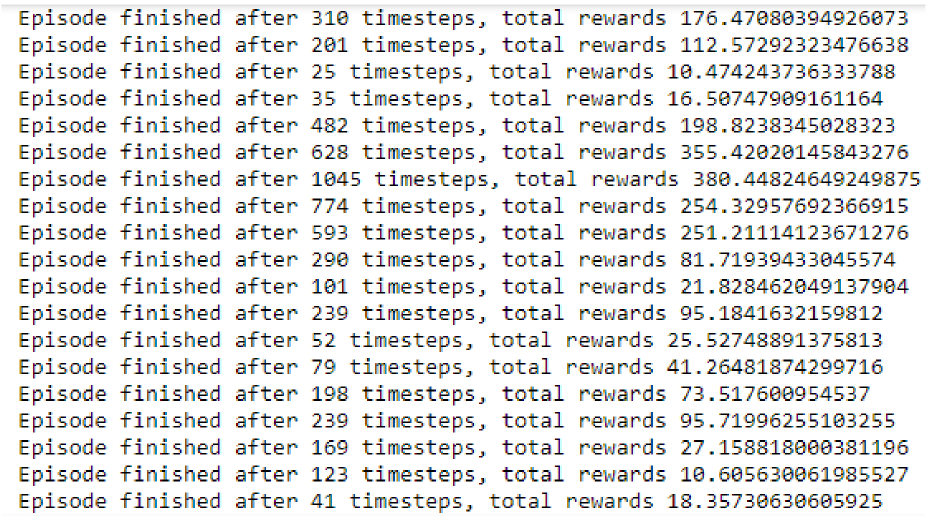
DQN(6/9)

- 建立獎勵分配方法，讓獎勵提供更多資訊



```
# Cheating part: modify the reward to speed up training process # 修改 reward，加快訓練
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
```

- 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。

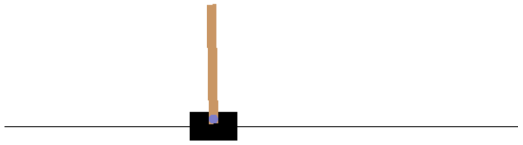


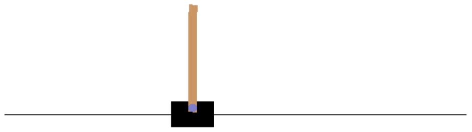
```
Episode finished after 310 timesteps, total rewards 176.47080394926073
Episode finished after 201 timesteps, total rewards 112.57292323476638
Episode finished after 25 timesteps, total rewards 10.474243736333788
Episode finished after 35 timesteps, total rewards 16.50747909161164
Episode finished 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 18.357306306059925
```

- 由於方法的不同，不能以整體獎勵值來比較，主要是看timesteps。

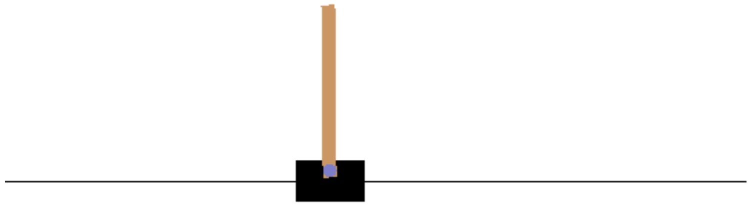
DQN(7/9)

- 前期
- 中期
- 後期



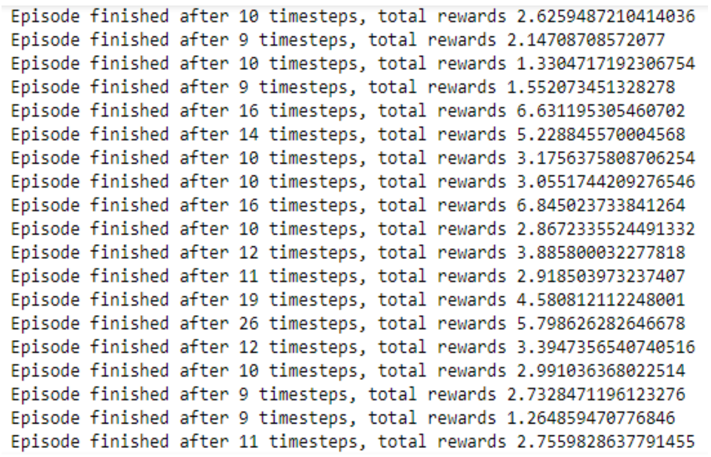


- 在訓練後期的後期，桿子已經可以穩定地站立在小車上了



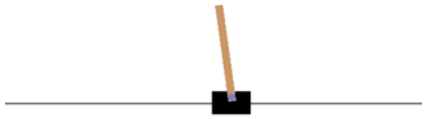
DQN(8/9)

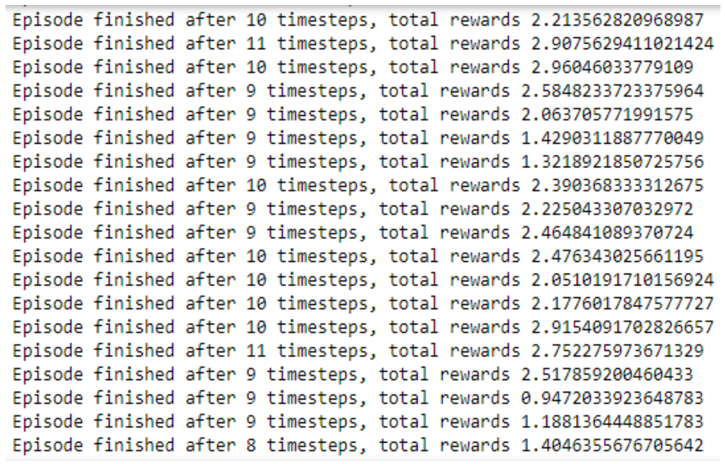
- 學習率→0.9



```
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
```

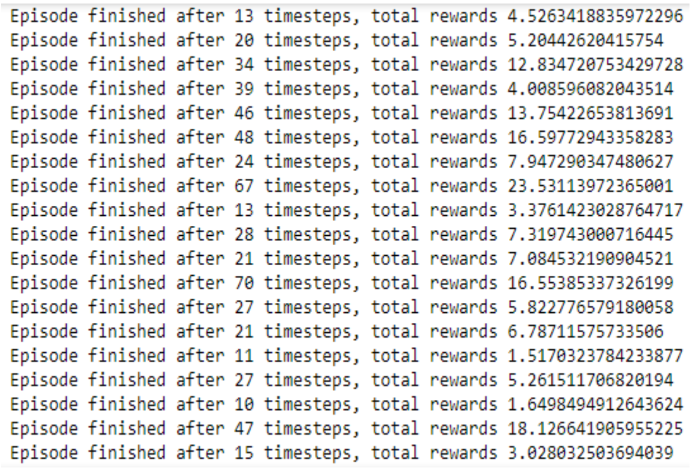
- 學習率→0.0001





```
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
```

- ε→0.9

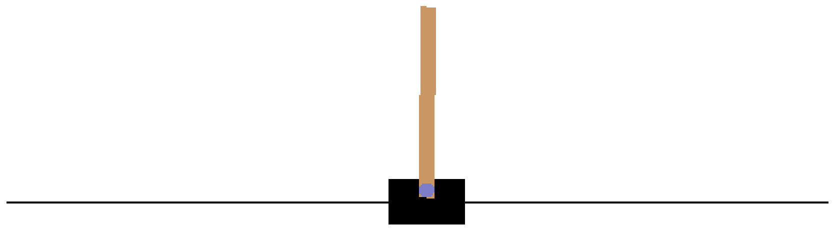


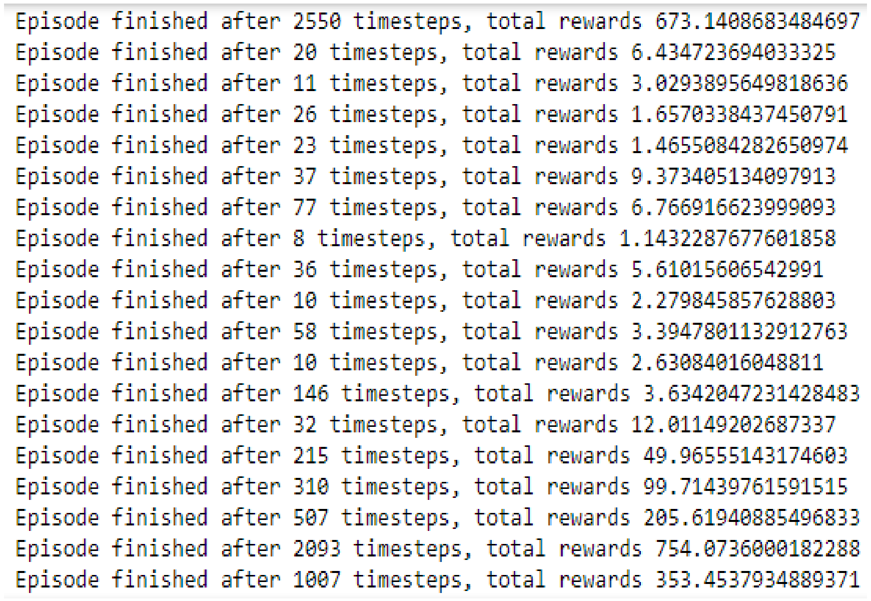
```
Episode finished after 13 timesteps, total rewards 4.5263418835972296
Episode finished after 20 timesteps, total rewards 5.20442620415754
Episode finished after 34 timesteps, total rewards 12.834720753429728
Episode finished after 39 timesteps, total rewards 4.008596082043514
Episode finished after 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
```

- Agent學習新知的機率大過原本的0.1，花了大部分的時間都在學習新知。

DQN(9/9)

- ε→0.0001





```
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
```

- 一直到訓練後期，車子還是會往旁邊飄移。

- 整體獎勵比ε為0.9高很多，但是由於新知不足，所以沒有辦法一直讓桿子維持不倒的時間最大化。

Conclusion(1/3)

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。

Conclusion(2/3)

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。

Conclusion(3/3)

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。

Future work

- 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，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。

Discussion

- 遇到的困難

對程式碼的不熟悉，造成學習緩慢。

程式碼修改不到位，或是有想法卻做不出來的種種困境。

- 如何克服

多方參考網路上的各種前人寫的程式碼。

Reference

- OpenAI gym Cart-Pole環境：
  https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py?fbclid=IwAR0_pDPRxQi4GjcSmQqXVENyUKQYihXYNFqfWOfi0fzE2OPow1AUByJDS3M

- Bucket查詢：
  http://pages.cs.wisc.edu/~finton/qcontroller.html

- 各式定理：
  https://en.wikipedia.org/wiki/Markov_model
  https://en.wikipedia.org/wiki/Hidden_Markov_model
  https://morvanzhou.github.io/tutorials/machine-learning/torch/4-05-DQN/
  https://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/

國立清華大學
National Tsing Hua University

INNOVATION ENGINEERING LAB

Thank You for Your Listening