

智慧化企業整合 Project3

利用強化學習(Reinforcement Learning)跑小迷宮比較 Q-learning

跟 Sarsa 演算法之差異

Individual Research Project

108034547 梁茲晴

摘要

強化學習 (Reinforcement Learning) 已應用於許多領域和應用程序，著名領域是 Q-learning 和 Sarsa 算法，但是它們具有不同的特徵。Sarsa 算法具有更快的收斂特性，而 Q-learning 算法具有更好的最終性能。但是，Sarsa 算法很容易陷入局部最小值，並且 Q-learning 需要更長的時間來學習。機器學習在從控制系統到數據挖掘的多個領域中非常重要。

本文以迷宮 Maze 求解從迷宮中尋找寶藏，是一種可視化模型方法，將迷宮之炸彈、寶藏加入獎勵懲罰制度，依據前次的結果使機器不斷訓練進步，而過程類似於動態規劃中的最小路徑規劃問題。利用最後 reward 分數及步數來比較 Q-learning 跟 Sarsa 演算法之差異。

關鍵字：強化學習、Q-learning、Sarsa、Maze

目錄

一、	主題介紹	4
(一)	5W1H.....	4
(二)	動機	4
(三)	目的	4
(四)	何謂強化學習	4
二、	流程與架構	5
(一)	Q-learning	5
(二)	Sarsa.....	6
三、	程式碼	7
(一)	Env 環境架設	8
(二)	Q-learning	10
(三)	Sarsa.....	12
四、	結果及比較 Q-learning 和 Sarsa	15
(一)	Q-learning	15
(二)	Sarsa.....	18
(三)	比較 Q-learning 和 Sarsa	21
五、	結論	21
六、	未來展望	22
七、	文獻探討	22

一、主題介紹

(一)5W1H

What：利用小迷宮遊戲結果比較出 Q-learning 跟 Sarsa 演算法之差異

Why：想清楚了解兩種方法之差異與使用效果。

Where：可將其運用於動態規劃、博議論，或任何強調如何基於環境而行動，以取得最大化的預期利益的情況。

When：需要做優化選擇時

Who：需要得到最大獎賞的對象

How：利用小迷宮遊戲的結果觀察並比較

(二)動機

強化學習的演算法有多種，其中透過價值選行為的方法中，最多人使用的是 Q-learning 和 Sarsa，但也因為每個方法特性不同，各有各的支持者，因此我想要探討兩者之差異和適用時機。

(三)目的

強化學習中最具代表的技術就是 TD-learning(時間差學習)，可再細分為 on-policy 的 SARSA 算法與 off-policy 的 Q-learning，希望藉由小迷宮模擬，比較出兩者之差異。

(四)何謂強化學習

我們可以將所有強化學習的方法分為 model-free、model-based。前者為假如我們不嘗試去理解環境，環境給什麼就是什麼，就把這種方法叫做 model-free，像 Q learning、Sarsa、Policy Gradients 都是從環境中

得到反饋然後從中學習；那理解環境也就是學會用一個模型來代表環境，就是 model-based，它只是多了一道程序，為真實世界建模打造一個虛擬環境。

強化學習其實也是機器學習的一個分支，但是它與我們常見監督學習和無監督學習又不太一樣。監督學習是已經有了數據和數據對應的正確標籤，這樣就能學習出那些對應哪種標籤是正確的，但強化學習還要更進步，一開始它並沒有數據和標籤，要通過一次次在環境中的嘗試，獲取這些數據和標籤，然後再學習通過哪些數據能夠對應哪些標籤，最後利用學習到的這些規律，盡可能選擇可以得到高分的行為，這也就證明了在強化學習中，分數標籤就是他的老師，和監督學習中的老師也差不多意思。

強化學習旨在選擇最優決策，在一系列的情景之下，通過多步的決策來達到一個目標，是一種時間序列多步決策的問題，以預測獎勵總金額的量度預計會在將來出現，並不需要出現正確的輸入和輸出對，也不需要精確校正次優化的行為。強化學習更加專注於在線規劃，需要在探索未知的領域和遵從現有知識之間找到平衡。

二、 流程與架構

強化學習又分成 on-policy 的 Sarsa 算法與 off-policy 的 Q-learning。他們都是基於價值且從環境中學習在反饋的架構。

(一)Q-learning

Q 是 Quality 的首字母，表示質量/優劣，Q-learning 是基於價值(Value-Based) 的一種決策過程，永遠都是想著 Q-value 最大化，使得這個 maxQ 變得貪婪，不考慮其他非 maxQ 的結果，因此可以理解成 Q-learning 是一種貪

婪、大膽、勇敢的算法，鎖定寶藏勇往直前，對於錯誤，死亡並不在乎。下圖為 Q-learning 之偽代碼：

```
Initialize  $Q(s, a)$  arbitrarily
Repeat (for each episode):
  Initialize  $s$ 
  Repeat (for each step of episode):
    Choose  $a$  from  $s$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
    Take action  $a$ , observe  $r, s'$ 
     $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 
     $s \leftarrow s'$ ;
  until  $s$  is terminal
```

圖一 Q-learning 偽代碼

首先建立一個 Q 表，將現在位置 state 轉成 action，接著利用策略選擇動作並且建立一環境，根據 ϵ -greedy 貪婪法來學習，而上述公式正是程式碼裡面 RL 的定義所要表達之架構。而 ϵ -greedy = 0.9 表示 90% 會按照 Q 表的最佳情況尋優 10% 隨機； α 為學習效率； γ 為衰減值。此方法雖用 $\max Q(s_2)$ 估算下一個 state，但沒有在 s_2 做出任何的行為， s_2 的行為決策要等到更新完再重新另外算，這就是 off-policy 的 Q-learning 的決策和優化功能。

(二) Sarsa

SARSA 的首字母縮寫詞來自「狀態 (State) - 動作 (Action) - 獎勵 (Reward) - 狀態 - 動作」，基本前提是給定一個狀態 S 和動作 A，我們應該通過增加它來獲得獎勵 (R) 加上下一個狀態 - 動作對的 Q-value，利用 ϵ -greedy 貪婪算法來更新 Q 值。偽代碼如下圖所表示：

```

Initialize  $Q(s, a)$  arbitrarily
Repeat (for each episode):
  Initialize  $s$ 
  Choose  $a$  from  $s$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
  Repeat (for each step of episode):
    Take action  $a$ , observe  $r, s'$ 
    Choose  $a'$  from  $s'$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
     $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$ 
     $s \leftarrow s'; a \leftarrow a';$ 
  until  $s$  is terminal

```

圖二 Sarsa 偽代碼

Sarsa 的算法還是一直不斷更新 Q table 裡的值，再根據新的值來判斷下一個 state 要採取怎樣的 action，也就是在當前 state 已經想好對應的 action，且想好下一個 state 和下一個 action，更新 $Q(s, a)$ 時是基於的是下一個 $Q(s, a)$ ，而 Q-learning 是基於 maxQ，這也是與 Q-learning 差異之處，Sarsa 會實際採取行動。

三、 程式碼

程式碼的編寫主要分為:env、RL-brain、run 三個部分。其中環境架設和一樣的，因此以下統一解釋。

模型之參數設定:

- Learning rate=0.1 學習率
- Gamma=0.9 獎勵遞減值
- ϵ -greedy=0.9 貪婪度

(一)Env 環境架設

1. 套用 TKINTER(圖形程式設計介面)模組

```
import numpy as np
import time
import sys
if sys.version_info.major == 2:
    import Tkinter as tk
else:
    import tkinter as tk
```

2. 設定模型長寬

```
UNIT = 40 # pixels
MAZE_H = 5 # grid height
MAZE_W = 5 # grid width
```

3. 設定初始值(方向、位置、圖名、大小)

```
class Maze(tk.Tk, object):
    def __init__(self):
        super(Maze, self).__init__()
        self.action_space = ['u', 'd', 'l', 'r']
        self.n_actions = len(self.action_space)
        self.title('maze')
        self.geometry('{0}x{1}'.format(MAZE_H * UNIT, MAZE_H * UNIT))
        self._build_maze()
```

4. 建造一個背景是白色的迷宮(maze)

```
def _build_maze(self):
    self.canvas = tk.Canvas(self, bg='white',
                             height=MAZE_H * UNIT,
                             width=MAZE_W * UNIT)

    # create grids 劃格線
    for c in range(0, MAZE_W * UNIT, UNIT):
        x0, y0, x1, y1 = c, 0, c, MAZE_H * UNIT
        self.canvas.create_line(x0, y0, x1, y1)
    for r in range(0, MAZE_H * UNIT, UNIT):
        x0, y0, x1, y1 = 0, r, MAZE_W * UNIT, r
        self.canvas.create_line(x0, y0, x1, y1)

    # create origin
    # 零點(左上角) 往右是x增長的方向。往左是y增長的方向。
    # 因為每個方格是40畫素，20,20是中心位置。
    origin = np.array([20, 20])
```

5. 設定探索者(紅)、寶藏(黃)、炸彈(黑)*3

```
hell2_center = origin + np.array([UNIT * 2, UNIT * 1])
self.hell2 = self.canvas.create_rectangle(
    hell2_center[0] - 15, hell2_center[1] - 15,
    hell2_center[0] + 15, hell2_center[1] + 15,
    fill='black')

hell3_center = origin + np.array([UNIT * 1, UNIT * 3])
self.hell3 = self.canvas.create_rectangle(
    hell3_center[0] - 15, hell3_center[1] - 15,
    hell3_center[0] + 15, hell3_center[1] + 15,
    fill='black')

hell4_center = origin + np.array([UNIT * 4, UNIT * 2])
self.hell4 = self.canvas.create_rectangle(
    hell4_center[0] - 15, hell4_center[1] - 15,
    hell4_center[0] + 15, hell4_center[1] + 15,
    fill='black')

# create oval
oval_center = origin + np.array([UNIT * 2, UNIT * 3])
self.oval = self.canvas.create_oval(
    oval_center[0] - 15, oval_center[1] - 15,
    oval_center[0] + 15, oval_center[1] + 15,
    fill='yellow')

# create red rect
self.rect = self.canvas.create_rectangle(
    origin[0] - 15, origin[1] - 15,
    origin[0] + 15, origin[1] + 15,
    fill='red')

# pack all
self.canvas.pack()
```

6. 設定每回合遊戲開始時，探索者將會回到原點

```
# 重置 (遊戲重新開始，將機器人放到原處)
def reset(self):
    self.update()
    time.sleep(0.5)
    self.canvas.delete(self.rect)
    origin = np.array([20, 20])
    self.rect = self.canvas.create_rectangle(
        origin[0] - 15, origin[1] - 15,
        origin[0] + 15, origin[1] + 15,
        fill='red')
    # return observation
    return self.canvas.coords(self.rect)
```

7. 以當前狀態更新下一步動作及如何得獎勵

#當前狀態選擇動作後的下一狀態及其獎勵

```
def step(self, action):
    s = self.canvas.coords(self.rect)
    base_action = np.array([0, 0])
    #基本動作
    if action == 0: # up
        if s[1] > UNIT:
            base_action[1] -= UNIT
    elif action == 1: # down
        if s[1] < (MAZE_H - 1) * UNIT:
            base_action[1] += UNIT
    elif action == 2: # right
        if s[0] < (MAZE_W - 1) * UNIT:
            base_action[0] += UNIT
    elif action == 3: # left
        if s[0] > UNIT:
            base_action[0] -= UNIT
```

```
self.canvas.move(self.rect, base_action[0], base_action[1]) # move agent
```

#取得下一個state

```
s_ = self.canvas.coords(self.rect)
```

8. 獎勵機制

reward function獎勵機制

```
if s_ == self.canvas.coords(self.oval):
    reward = 3
    done = True
    s_ = 'terminal'
```

```
#elif s_ in [self.canvas.coords(self.hell1), self.canvas.coords(self.hell2)]:
```

```
elif s_ == self.canvas.coords(self.hell2):
    reward = -1
    done = True
    s_ = 'terminal'
```

```
elif s_ in [self.canvas.coords(self.hell3), self.canvas.coords(self.hell4)]:
    reward = -1 # 踩到炸彈2，獎勵為 -1
    done = True
    s_ = 'terminal' # 終止
```

```
else:
    reward = 0
    done = False
```

```
return s_, reward, done
```

(二)Q-learning

◆ RL_brain

1. 設置 QLearningTable 及初始值

```
import numpy as np
import pandas as pd

class QLearningTable:
    def __init__(self, actions, learning_rate=0.01, reward_decay=0.9, e_greedy=0.9):
        self.actions = actions # a list
        self.lr = learning_rate #學習率
        self.gamma = reward_decay #遲罰因子
        self.epsilon = e_greedy #貪婪度
        self.q_table = pd.DataFrame(columns=self.actions, dtype=np.float64) #Q表
```

2. 選擇 action。先檢查這步的 state 是否已經存在，利用 ϵ -greedy 進行學習，並選擇 Q-value 較大者。

```
# 根據 observation 來選擇 action
def choose_action(self, observation):
    self.check_state_exist(observation) # 檢測此 state 是否在 q_table 中存在
    # action selection 選擇行為，用 Epsilon Greedy 貪婪方法
    if np.random.uniform() < self.epsilon:
        # choose best action
        state_action = self.q_table.loc[observation, :] # 選擇QTABLE比較大的值
        # some actions may have the same value, randomly choose on in these actions
        action = np.random.choice(state_action[state_action == np.max(state_action)].index)
    else:
        # choose random action
        action = np.random.choice(self.actions)
    return action
```

3. 先確認 state 狀態，若下一步還沒終止，預測出 action，更新 Q-table

```
#學習
def learn(self, s, a, r, s_):
    self.check_state_exist(s_)
    q_predict = self.q_table.loc[s, a]
    if s_ != 'terminal':
        q_target = r + self.gamma * self.q_table.loc[s_, :].max() # next state is not terminal
    else:
        q_target = r # next state is terminal
    self.q_table.loc[s, a] += self.lr * (q_target - q_predict) # update
```

4. 檢查目前 state 是否已在 Q-table 中，若無則添加。

```
def check_state_exist(self, state):
    if state not in self.q_table.index:
        # append new state to q table
        self.q_table = self.q_table.append(
            pd.Series(
                [0]*len(self.actions),
                index=self.q_table.columns,
                name=state,
            ),
            ignore_index=True)
```

◆ Run

1. 放入套件

```
import matplotlib.pyplot as plt
from maze_env2 import Maze
from RL_brain2 import QLearningTable
```

2. 更新制度，每回合皆會印出 Q-Table 及步數。

```
def update():
    reward = 0
    reward_list_1 = []
    step_list = []
    for episode in range(30):
        # initial observation
        step_count = 0
        observation = env.reset()
        #step_count = 0 # 記錄走過的步數
        while True:
            # fresh env
            env.render()

            # RL choose action based on observation RL 大腦根據observation挑選 action
            action = RL.choose_action(str(observation))

            # RL take action and get next observation and reward
            # 探索者在環境中實施這個 action，並得到環境返回的下一個observation，reward 和 done (是否是踩到炸彈或者找到寶藏)
            observation_, reward, done = env.step(action)
            #step_count = 1 # 增加步數

            # RL Learn from this transition
            RL.learn(str(observation), action, reward, str(observation_))

            # swap observation 機器人移動到下一個observation
            observation = observation_
            reward_list_1.append(reward)
            step_count += 1
            reward += 0
            # break while loop when end of this episode
            print(RL.q_table)
            print('game over, 總步數: {}'.format(step_count))
        if done:
            step_list.append(step_count)
            break
```

3. 印出每回合之 Reward

```
print(step_list)
plt.plot(reward_list_1, label = " Q-learning ")
plt.legend(loc = 0)
plt.xlabel( ' episode ' )
plt.ylabel( ' reward sum per episode ' )
plt.xticks([])
plt.title( " sarsa " )
env.destroy()
```

(三)Sarsa

◆ RL_brain

1. 放入套件，設定初始值

```
import numpy as np
import pandas as pd

class RL(object):
    def __init__(self, action_space, learning_rate=0.01, reward_decay=0.9, e_greedy=0.9):
        self.actions = action_space # a list
        self.lr = learning_rate
        self.gamma = reward_decay
        self.epsilon = e_greedy

        self.q_table = pd.DataFrame(columns=self.actions, dtype=np.float64)
```

2. 檢查目前 state 是否已在 Q-table 中，若無則添加。

```
def check_state_exist(self, state):
    if state not in self.q_table.index:
        # append new state to q table
        self.q_table = self.q_table.append(
            pd.Series(
                [0]*len(self.actions),
                index=self.q_table.columns,
                name=state,
            )
        )
```

3. 選擇 action

```
def choose_action(self, observation):
    self.check_state_exist(observation)
    # action selection
    if np.random.rand() < self.epsilon:
        # choose best action
        state_action = self.q_table.loc[observation, :]
        # some actions may have the same value, randomly choose on in these actions
        action = np.random.choice(state_action[state_action == np.max(state_action)].index)
    else:
        # choose random action
        action = np.random.choice(self.actions)
    return action

def learn(self, *args):
    pass
```

4. 設定 SarsaTable，其中從 def learn 那行可以看出此方法的學習包括下一個的 state(s_)、action(a_)，且是利用(實際-估計)*學習率去更新 Q-table。這亦是兩種方法差異之處。

```
# on-policy
class SarsaTable(RL):

    def __init__(self, actions, learning_rate=0.01, reward_decay=0.9, e_greedy=0.9):
        super(SarsaTable, self).__init__(actions, learning_rate, reward_decay, e_greedy)

    def learn(self, s, a, r, s_, a_):
        self.check_state_exist(s_)
        q_predict = self.q_table.loc[s, a]
        if s_ != 'terminal':
            q_target = r + self.gamma * self.q_table.loc[s_, a_] # next state is not terminal
        else:
            q_target = r # next state is terminal
        self.q_table.loc[s, a] += self.lr * (q_target - q_predict) # (實際-估計)*學習率 更新機制
        print(self.q_table)
```

◆ Run

1. 放入套件

```
import matplotlib.pyplot as plt

from maze_env_sarsa import Maze
from RL_brain_sarsa import SarsaTable
```

2. 更新機制

```
def update():
    reward=0
    reward_list_1 = []
    step_list = []
    for episode in range(30):
        # initial observation
        step_count = 0
        observation = env.reset()
        # RL choose action based on observation
        action = RL.choose_action(str(observation))

        while True:
            # fresh env
            env.render()

            # RL take action and get next observation and reward
            observation_, reward, done = env.step(action)

            # RL choose action based on next observation
            #選擇下一個狀態
            action_ = RL.choose_action(str(observation_))

            # Learning the Q-value==> Sarsa
            RL.learn(str(observation), action, reward, str(observation_), action_)

            # swap observation and action
            observation = observation_
            action = action_
            reward_list_1.append(reward)
            step_count +=1
            reward +=0
            # break while loop when end of this episode
            print('game over, 總步數 : {}'.format(step_count))
            #temp = format(step_count)
            if done:
                step_list.append(step_count)
                break

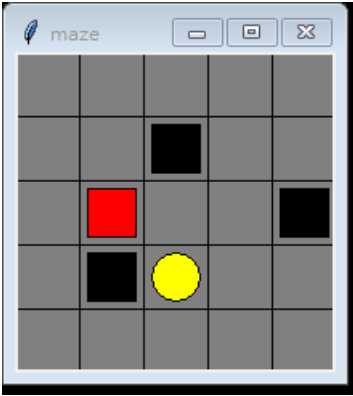
    print(step_list)
```

3. 印出每回合之 Reward

```
plt.plot(reward_list_1, label = " sarsa ")
plt.legend(loc = 0) #'best'表示自动分配最佳位置
plt.xlabel( ' episode ' )
plt.ylabel( ' reward sum per episode ' )
plt.xticks([])
plt.title( " sarsa " )
env.destroy()
```

四、結果及比較 Q-learning 和 Sarsa

(一)Q-learning

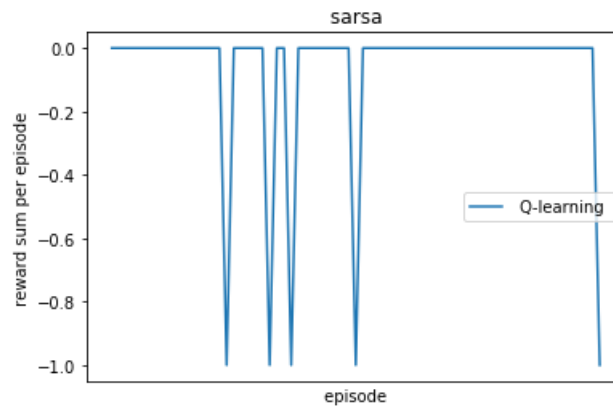


圖三 Q-learning 小迷宮示意圖

1. 5 次(學習次數太少)

	0	1	2	3
[5.0, 5.0, 35.0, 35.0]	0.00	0.00	0.00	0.0
[45.0, 5.0, 75.0, 35.0]	0.00	0.00	0.00	0.0
[45.0, 45.0, 75.0, 75.0]	0.00	0.00	-0.01	0.0
[5.0, 45.0, 35.0, 75.0]	0.00	0.00	0.00	0.0
[5.0, 85.0, 35.0, 115.0]	0.00	0.00	0.00	0.0
[5.0, 125.0, 35.0, 155.0]	0.00	0.00	0.00	0.0
[45.0, 85.0, 75.0, 115.0]	0.00	-0.01	0.00	0.0
[85.0, 85.0, 115.0, 115.0]	-0.01	0.00	0.00	0.0
terminal	0.00	0.00	0.00	0.0
[85.0, 5.0, 115.0, 35.0]	0.00	-0.01	0.00	0.0
[125.0, 5.0, 155.0, 35.0]	0.00	0.00	0.00	0.0
[165.0, 5.0, 195.0, 35.0]	0.00	0.00	0.00	0.0
[125.0, 45.0, 155.0, 75.0]	0.00	0.00	0.00	0.0
[125.0, 85.0, 155.0, 115.0]	0.00	0.00	-0.01	0.0
game over，總步數：34				

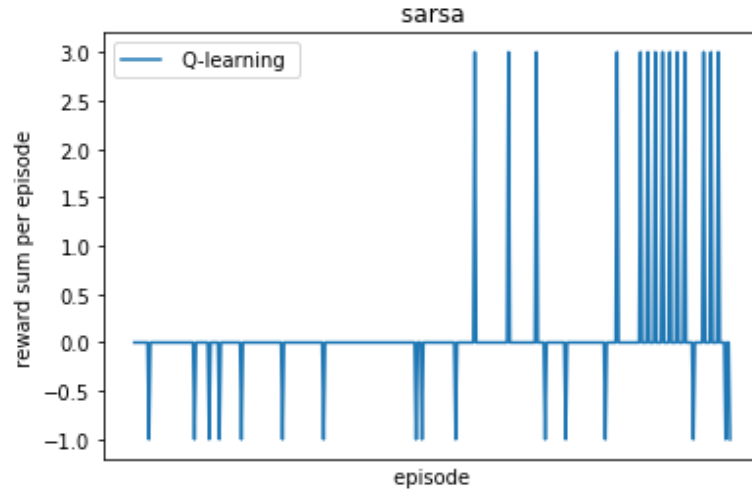
[17, 6, 3, 9, 34]



2. 30 次

	0	1	2	3
[5.0, 5.0, 35.0, 35.0]	0.000000e+00	0.000000	0.000001	0.000000e+00
[45.0, 5.0, 75.0, 35.0]	0.000000e+00	0.000048	0.000000	0.000000e+00
[5.0, 45.0, 35.0, 75.0]	0.000000e+00	0.000000	0.000000	0.000000e+00
[45.0, 45.0, 75.0, 75.0]	1.291337e-07	0.001126	-0.039404	0.000000e+00
terminal	0.000000e+00	0.000000	0.000000	0.000000e+00
[5.0, 85.0, 35.0, 115.0]	0.000000e+00	0.000000	0.000128	0.000000e+00
[45.0, 85.0, 75.0, 115.0]	0.000000e+00	-0.029701	0.026000	2.187000e-08
[85.0, 5.0, 115.0, 35.0]	0.000000e+00	-0.010000	0.000000	0.000000e+00
[125.0, 5.0, 155.0, 35.0]	0.000000e+00	0.000000	0.000000	0.000000e+00
[125.0, 45.0, 155.0, 75.0]	0.000000e+00	0.000000	0.000000	-1.000000e-02
[85.0, 85.0, 115.0, 115.0]	-1.990000e-02	0.393763	0.000000	0.000000e+00
[5.0, 125.0, 35.0, 155.0]	0.000000e+00	0.000000	-0.019900	0.000000e+00
[5.0, 165.0, 35.0, 195.0]	0.000000e+00	0.000000	0.000000	0.000000e+00
[45.0, 165.0, 75.0, 195.0]	0.000000e+00	0.000000	0.000000	0.000000e+00
[125.0, 85.0, 155.0, 115.0]	0.000000e+00	0.000000	-0.010000	0.000000e+00
[165.0, 45.0, 195.0, 75.0]	0.000000e+00	-0.010000	0.000000	0.000000e+00
[125.0, 125.0, 155.0, 155.0]	0.000000e+00	0.000000	0.000000	0.000000e+00
[125.0, 165.0, 155.0, 195.0]	0.000000e+00	0.000000	0.000000	0.000000e+00
[165.0, 125.0, 195.0, 155.0]	-1.000000e-02	0.000000	0.000000	0.000000e+00
game over , 總步數 : 3				

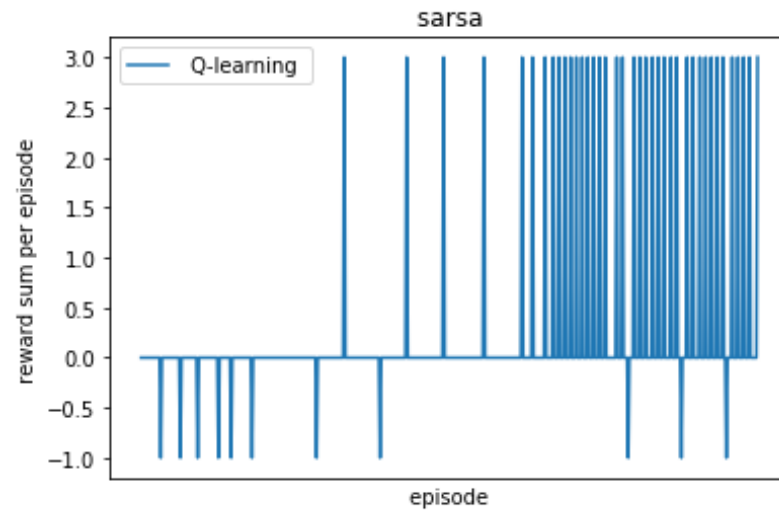
[11, 31, 10, 7, 15, 28, 28, 63, 4, 23, 13, 23, 19, 6, 14, 27, 8, 16, 5, 5, 5, 5, 5, 5, 6, 7, 5, 5, 5, 3]



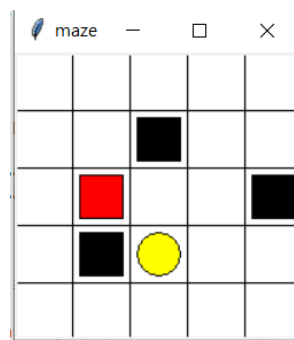
3. 50 次

	0	1	2	3
[5.0, 5.0, 35.0, 35.0]	2.294879e-07	0.000125	0.000000	9.270430e-08
[5.0, 45.0, 35.0, 75.0]	1.015560e-06	0.000000	0.001592	1.456489e-06
[45.0, 45.0, 75.0, 75.0]	0.000000e+00	0.017388	-0.019900	0.000000e+00
[45.0, 5.0, 75.0, 35.0]	0.000000e+00	0.000000	0.000000	0.000000e+00
[5.0, 85.0, 35.0, 115.0]	4.279697e-07	0.000000	0.000000	0.000000e+00
terminal	0.000000e+00	0.000000	0.000000	0.000000e+00
[5.0, 125.0, 35.0, 155.0]	0.000000e+00	0.000000	-0.010000	0.000000e+00
[45.0, 85.0, 75.0, 115.0]	6.377194e-07	-0.019900	0.152892	0.000000e+00
[85.0, 85.0, 115.0, 115.0]	-1.990000e-02	0.952336	0.000000	0.000000e+00
[85.0, 5.0, 115.0, 35.0]	0.000000e+00	-0.010000	0.000000	0.000000e+00
[5.0, 165.0, 35.0, 195.0]	0.000000e+00	0.000000	0.000000	0.000000e+00
[45.0, 165.0, 75.0, 195.0]	-1.000000e-02	0.000000	0.000000	0.000000e+00
[85.0, 165.0, 115.0, 195.0]	0.000000e+00	0.000000	0.000000	0.000000e+00
[125.0, 85.0, 155.0, 115.0]	0.000000e+00	0.000000	0.000000	2.700000e-04
[125.0, 45.0, 155.0, 75.0]	0.000000e+00	0.000000	0.000000	-1.000000e-02
[125.0, 165.0, 155.0, 195.0]	0.000000e+00	0.000000	0.000000	0.000000e+00
[85.0, 165.0, 115.0, 195.0]	0.000000e+00	0.000000	0.000000	0.000000e+00
[125.0, 85.0, 155.0, 115.0]	0.000000e+00	0.000000	0.000000	2.700000e-04
[125.0, 45.0, 155.0, 75.0]	0.000000e+00	0.000000	0.000000	-1.000000e-02
[125.0, 165.0, 155.0, 195.0]	0.000000e+00	0.000000	0.000000	0.000000e+00
[125.0, 125.0, 155.0, 155.0]	0.000000e+00	0.000000	0.000000	3.000000e-02
[125.0, 5.0, 155.0, 35.0]	0.000000e+00	0.000000	0.000000	0.000000e+00
[165.0, 45.0, 195.0, 75.0]	0.000000e+00	-0.010000	0.000000	0.000000e+00
[165.0, 5.0, 195.0, 35.0]	0.000000e+00	0.000000	0.000000	0.000000e+00
game over, 總步數 : 7				

[18, 17, 15, 18, 11, 18, 56, 24, 31, 23, 32, 35, 33, 9, 11, 7, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 10, 5, 5, 5, 5, 5, 6, 5, 5, 6, 5, 4, 5, 5, 6, 5, 5, 5, 5, 3, 5, 5, 5, 5, 7]



(二)Sarsa



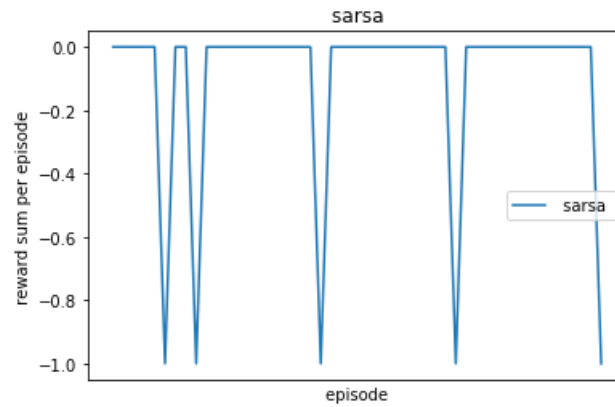
圖四 Sarsa 小迷宮示意圖

1. 5 次(學習次數太少)

	0	1	2	3
[5.0, 5.0, 35.0, 35.0]	0.00	0.00	0.00	0.0
[5.0, 45.0, 35.0, 75.0]	0.00	0.00	0.00	0.0
[45.0, 45.0, 75.0, 75.0]	0.00	0.00	0.00	0.0
[45.0, 85.0, 75.0, 115.0]	0.00	-0.01	0.00	0.0
terminal	0.00	0.00	0.00	0.0
[45.0, 5.0, 75.0, 35.0]	0.00	0.00	0.00	0.0
[85.0, 5.0, 115.0, 35.0]	0.00	-0.01	0.00	0.0
[125.0, 5.0, 155.0, 35.0]	0.00	0.00	0.00	0.0
[165.0, 5.0, 195.0, 35.0]	0.00	0.00	0.00	0.0
[165.0, 45.0, 195.0, 75.0]	0.00	-0.01	0.00	0.0
[5.0, 85.0, 35.0, 115.0]	0.00	0.00	0.00	0.0
[85.0, 85.0, 115.0, 115.0]	-0.01	0.00	0.00	0.0
[5.0, 125.0, 35.0, 155.0]	0.00	0.00	-0.01	0.0

game over，總步數：14

[6, 3, 12, 13, 14]

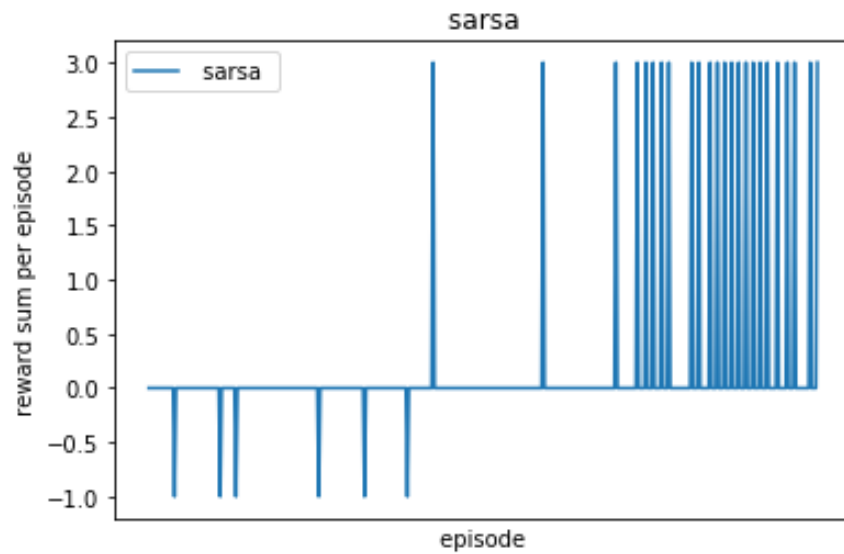


2. 30 次

	0	1	2	3
[5.0, 5.0, 35.0, 35.0]	0.00	0.000000	0.000005	8.831737e-10
[5.0, 45.0, 35.0, 75.0]	0.00	0.000000	0.000000	0.000000e+00
[5.0, 85.0, 35.0, 115.0]	0.00	0.000000	0.000000	0.000000e+00
[45.0, 85.0, 75.0, 115.0]	0.00	0.000000	0.054619	0.000000e+00
[85.0, 85.0, 115.0, 115.0]	0.00	0.595108	0.000000	0.000000e+00
[125.0, 85.0, 155.0, 115.0]	0.00	0.000000	-0.010000	0.000000e+00
terminal	0.00	0.000000	0.000000	0.000000e+00
[45.0, 5.0, 75.0, 35.0]	0.00	0.000139	0.000000	0.000000e+00
[85.0, 5.0, 115.0, 35.0]	0.00	-0.010000	0.000000	0.000000e+00
[125.0, 5.0, 155.0, 35.0]	0.00	0.000000	0.000000	0.000000e+00
[5.0, 125.0, 35.0, 155.0]	0.00	0.000000	-0.010000	0.000000e+00
[5.0, 165.0, 35.0, 195.0]	0.00	0.000000	0.000000	0.000000e+00
[45.0, 165.0, 75.0, 195.0]	-0.01	0.000000	0.000000	0.000000e+00
[125.0, 45.0, 155.0, 75.0]	0.00	0.000000	0.000000	-1.000000e-02
[165.0, 45.0, 195.0, 75.0]	0.00	-0.010000	0.000000	0.000000e+00
[45.0, 45.0, 75.0, 75.0]	0.00	0.003245	-0.010000	0.000000e+00
[165.0, 5.0, 195.0, 35.0]	0.00	0.000000	0.000000	0.000000e+00
[85.0, 165.0, 115.0, 195.0]	0.03	0.000000	0.000000	0.000000e+00

game over, 總步數 : 5

[19, 32, 11, 58, 32, 30, 18, 77, 51, 15, 6, 5, 6, 5, 16, 5, 8, 5, 5, 5, 5, 5, 5, 5, 5, 5, 7, 7, 5, 11, 5]

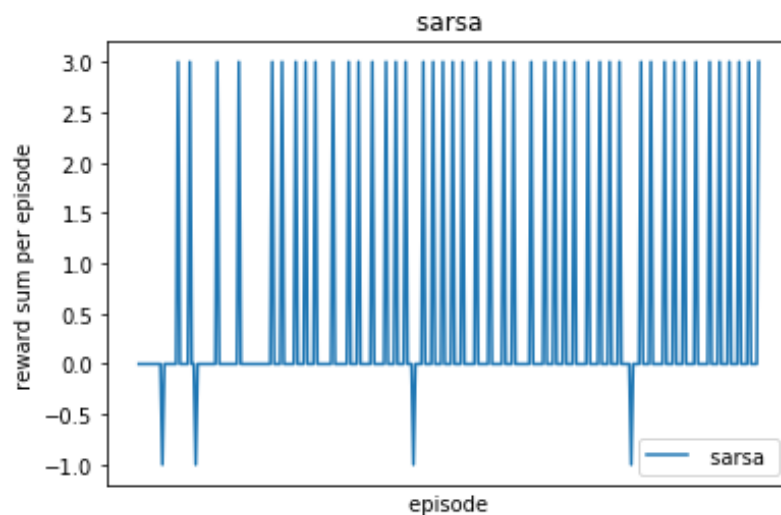


3. 50 次

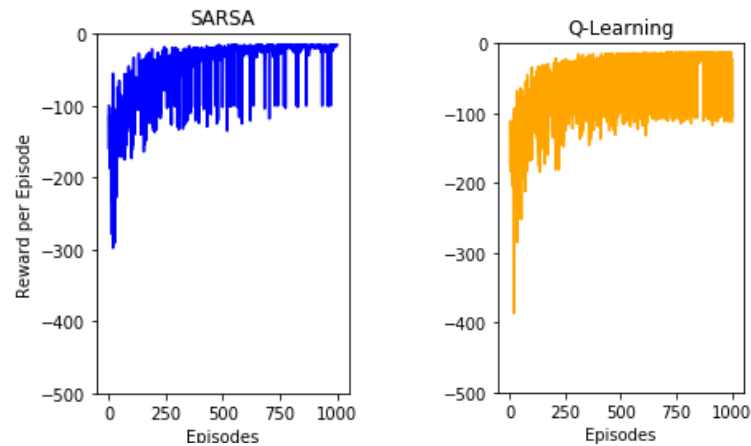
	0	1	2	3
[5.0, 5.0, 35.0, 35.0]	3.442650e-08	0.000180	7.267862e-10	7.076418e-07
[45.0, 5.0, 75.0, 35.0]	0.000000e+00	0.000000	0.000000e+00	1.782417e-07
[5.0, 45.0, 35.0, 75.0]	5.550029e-08	0.002424	9.330180e-07	4.838749e-06
[5.0, 85.0, 35.0, 115.0]	4.239968e-06	0.000000	2.535388e-02	0.000000e+00
[5.0, 125.0, 35.0, 155.0]	0.000000e+00	0.000000	-1.000000e-02	0.000000e+00
terminal	0.000000e+00	0.000000	0.000000e+00	0.000000e+00
[45.0, 45.0, 75.0, 75.0]	1.939655e-12	0.002394	-1.000000e-02	0.000000e+00
[45.0, 85.0, 75.0, 115.0]	1.074664e-05	-0.010000	2.065990e-01	2.830187e-05
[85.0, 85.0, 115.0, 115.0]	0.000000e+00	1.110529	0.000000e+00	1.596812e-03
[125.0, 85.0, 155.0, 115.0]	0.000000e+00	0.000000	-1.000000e-02	0.000000e+00

game over, 總步數 : 5

[13, 8, 6, 3, 11, 11, 17, 5, 7, 5, 5, 9, 8, 5, 7, 7, 5, 5, 4, 5, 5, 5, 5, 5, 7, 7, 7, 5, 9, 7, 5, 5, 5, 7, 6, 5, 5, 6, 5, 5, 7, 5, 5, 6, 7, 5, 5, 5, 5, 5]



(三)比較 Q-learning 和 Sarsa



上兩張圖為將兩者都跑 1000 次後繪製之圖形分析，可以看出 Q-learning 學習過程中獎勵值多小於 sarsa 學習方法，Sarsa 在線學習效果比較好。

得知 sarsa 對策略的探索更加高效，收斂快速，直觀簡單且較保守，走安全路徑路線但很容易陷入局部最小值；而 Q-learning 對下一狀態是直接用最大值來估計的，具有學習到全局最優的能力，所以有更好的最終性能，但是其收斂慢，需要更長的時間來學習。

五、 結論

SARSA 會根據其遵循的策略來學習操作值，而 Q-Learning 則相對於貪婪策略來學習操作值。

Q-learning 取得的是最大值，但是實際不一定會執行，每次使用 epsilon-greedy 貪婪方法來學習 Q 值，選擇當前狀態的 action，而 SARSA 使用當前策略執行的操作來學習 Q 值，選擇 next_state 的 next_action，也就是下一個狀態的動作，這個動作是下一次一定要做的，在更新 Q 值的時候已經為未來規劃好了動作，對錯誤和死亡比較敏感。

- ◆ Sarsa 選擇的是一條最安全的道路，遠離陷阱，因此容易來回踱步，不敢靠近寶藏，導致步數增加。

- ◆ Q-learning 選擇的是一條最快的道路，儘快到達出口，在決策過程中較為大膽，踩地雷也無妨。

因此，對於成本較高或風險較大，不允許失敗的情境，適合適用 Sarsa 方法；而對於欲快速找到最佳路徑的問題，則可以使用 Q-learning。

六、 未來展望

這兩種算法都屬於不連續決策的問題，可能會導致動作值函數空間中缺少固定點，因此可靠性較差，在決策過程中是以一步結束後對結果進行學習，對此並無法明確的知道哪一步是真正要的，若需要可靠性高一點之的方法可以考慮採用 Sarsa(λ)，是將每一局結束後下去分析，可以更清楚的知道哪一步，是正確對拿到寶藏有利的。

七、 文獻探討

(一) Q-learning

- <https://medium.com/@skywalker0803r/%E8%AA%8D%E8%AD%98%E4%B8%A6%E8%A7%A3%E9%8E%96reinforce-learning%E7%9A%84%E7%AC%AC%E4%B8%80%E6%AD%A5-q-learning%E7%AE%97%E6%B3%95-712045b890d3>
- <https://www.itread01.com/content/1526976299.html>
- <https://morvanzhou.github.io/tutorials/machine-learning/reinforcement-learning/2-2-tabular-q1/>

(二) Sarsa

- <https://ithelp.ithome.com.tw/articles/10207744>
- <https://morvanzhou.github.io/tutorials/machine-learning/reinforcement-learning/3-1-tabular-sarsa1/>

(三) 論文

- [1] Wang, Y., Shi, Z. R., Yu, L., Wu, Y., Singh, R., Joppa, L., & Fang, F. (2019, July). Deep reinforcement learning for green security games with real-time information. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 33, pp. 1401-1408).
- [2] Osmanković, D., & Konjicija, S. (2011, May). Implementation of Q—Learning algorithm for solving maze problem. In *2011 Proceedings of the 34th International Convention MIPRO* (pp. 1619-1622). IEEE.
- [3] Tijisma, A. D., Drugan, M. M., & Wiering, M. A. (2016, December). Comparing exploration strategies for Q-learning in random stochastic mazes. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)* (pp. 1-8). IEEE.
- [4] Wang, Y. H., Li, T. H. S., & Lin, C. J. (2013). Backward Q-learning: The combination of Sarsa algorithm and Q-learning. *Engineering Applications of Artificial Intelligence*, 26(9), 2184-2193.