智慧化企業整合
Intelligent Integration of Enterprise

利用強化學習(Reinforcement Learning)
跑小迷宮比較Q-learning跟Sarsa演算法之差異

108034547 梁茲晴

Outline

- 主題介紹
- 流程與架構
- 程式碼
- 結果及比較Q-learning和Sarsa結論
- 未來展望
- Reference

主題介紹-5W1H

- What：利用小迷宮遊戲比較出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，或任何欲取得最大化的利益的情況。
- 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

主題介紹–動機與目的

- 動機

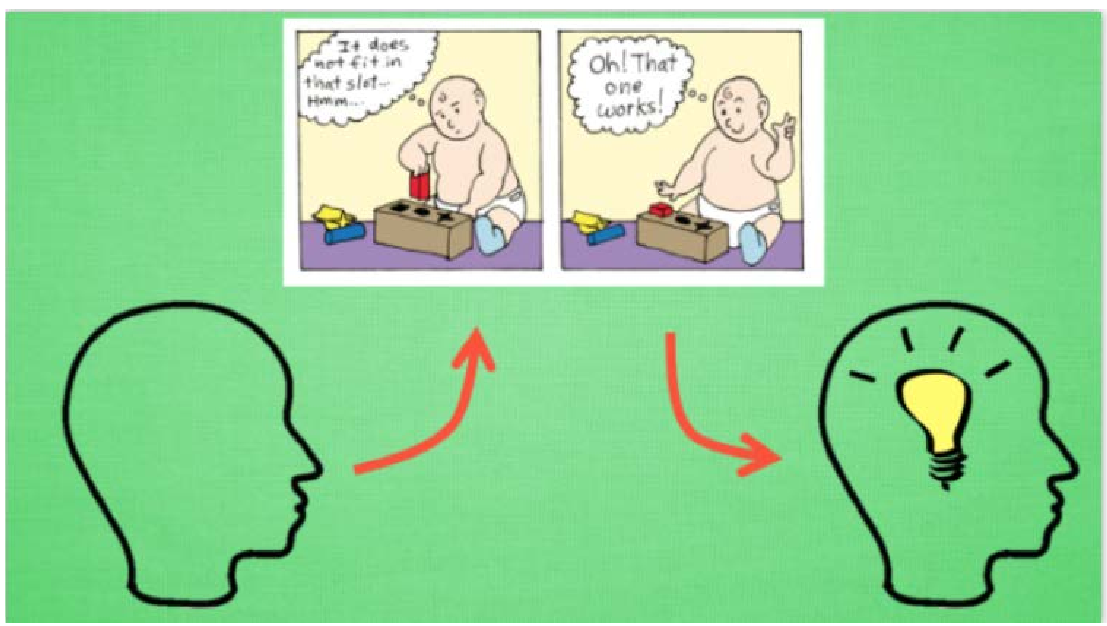
　　強化學習的演算法有多種，其中透過價值選行為的方法中，最多人使用的是Q-learning和Sarsa，但也因為每個方法特性不同，各有各的支持者，因此我想要探討兩者之差異和適用時機。

- 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。

何謂強化學習?

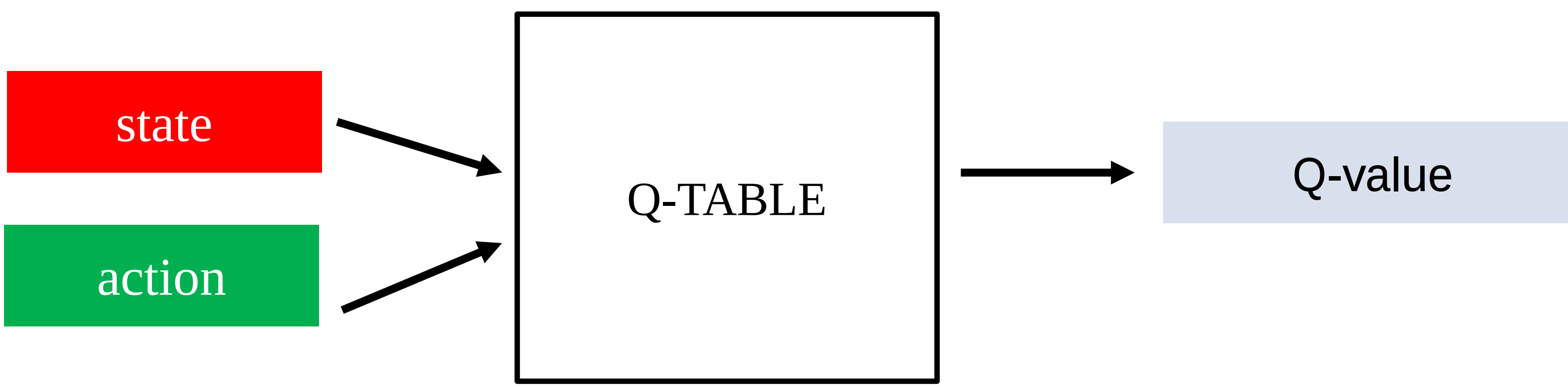
- 從無到有，不斷強化
- 強化學習的目標就是要尋找一個能使得到最大獎賞的策略。
  一套非通用得框架
- 強化學習中最具代表的技術就是TD-learning(時間差學習)
  可再細分為on-policy的SARSA算法與off-policy的Q-learning



何謂強化學習?

- 和其他機器學習不同之處
  1. 沒有大量標註數據進行監督，只能從每一步動作得到獎勵
  2. 時間序列的重要性，下一步仰賴於前一狀態
  3. 延遲獎勵，整局結束才會得到獎勵
- Q-learning的核心是Q-table

  Q-table的行和列分別表示state和action的值Q(s,a)

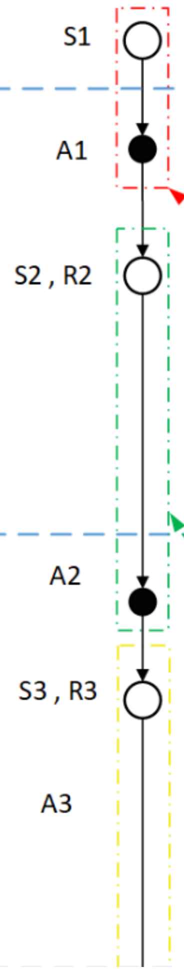


流程架構-Q-learning

- Q =Quality　質量/優劣
- 基於價值的一種決策過程，永遠都是想著Q-value最大化，使得maxQ變得貪婪

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., ε-greedy)
　　Take action a, observe r, s'
　　$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
　　$s \leftarrow s'$;
　until s is terminal

流程架構-Q-learning



Initialize S1

Repeat 1：

Choose A1 using policy derived from Q (e.g., ε-greedy)

Observe S2 , R2

Updating :

$$Q_{k+1}(S_1, A_1) = Q_k(S_1, A_1) + \alpha[R_2 + \gamma max_a Q_k(S_2, a_2) - Q_k(S_1, A_1)]$$

Note : We choose the optimal Q value, s.t $\{maxQ(S_2, a_2)|a_2 \in A_2\}$, to update without any ε-policy. And so far we haven't known which the next action is actually chose to interact with environment until

Repeat 2：

Choose A2 using policy derived from Q (e.g., ε-greedy)

Observe S3 , R3

Note : Again choose the optimal Q value, s.t $\{maxQ(S_3, a_3)|a_3 \in A_3\}$ to update without any ε-policy.

Updating :

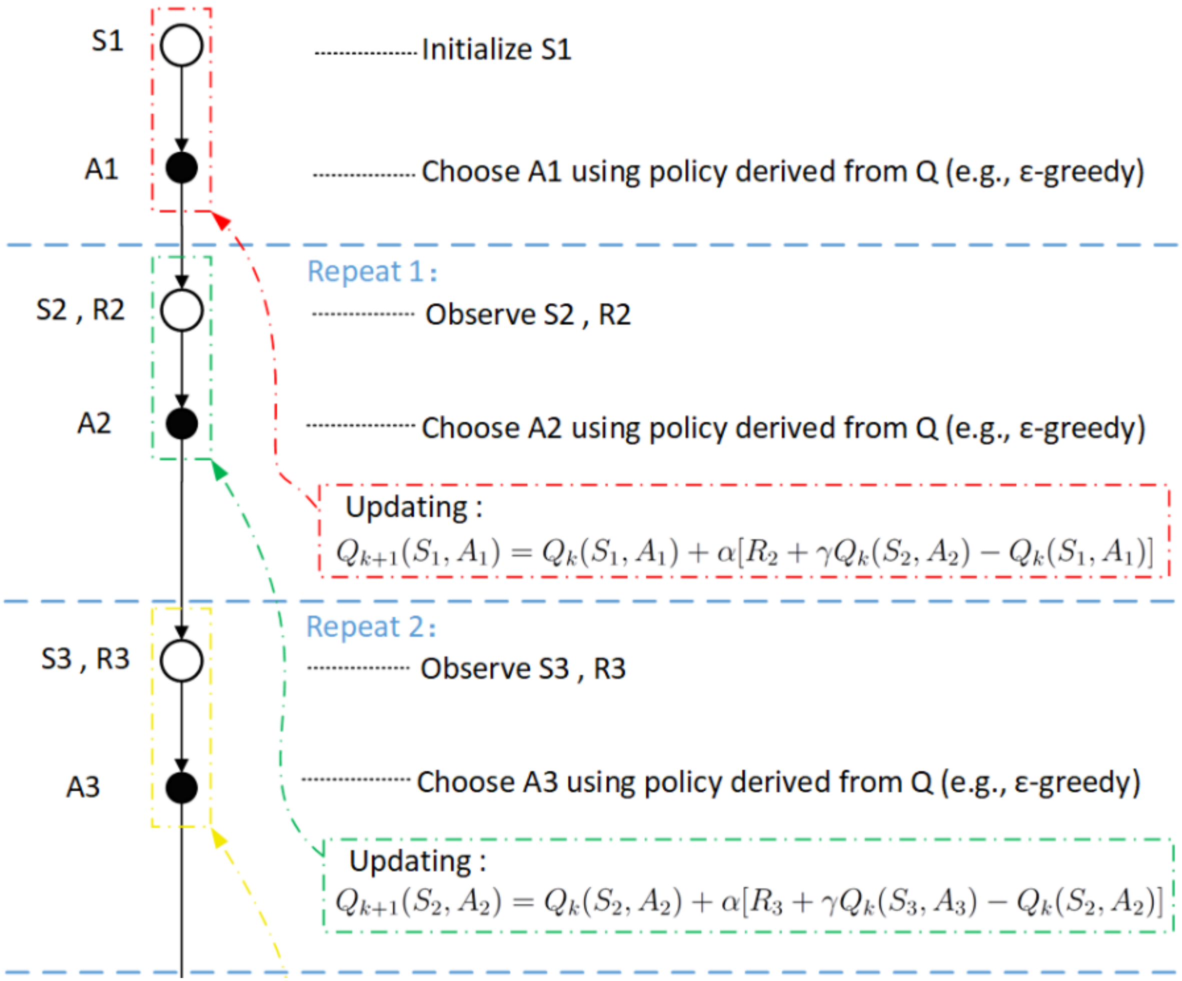
$$Q_{k+1}(S_2, A_2) = Q_k(S_2, A_2) + \alpha[R_3 + \gamma max_a Q_k(S_3, a_3) - Q_k(S_2, A_2)]$$

流程架構-Sarsa

狀態State - 動作Action - 獎勵Reward - 狀態 - 動作

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

流程架構-Sarsa



S1
Initialize S1
A1
Choose A1 using policy derived from Q (e.g., ε-greedy)
Repeat 1 :
S2 , R2
Observe S2 , R2
A2
Choose A2 using policy derived from Q (e.g., ε-greedy)
Updating :
$Q_{k+1}(S_1, A_1) = Q_k(S_1, A_1) + \alpha[R_2 + \gamma Q_k(S_2, A_2) - Q_k(S_1, A_1)]$
Repeat 2 :
S3 , R3
Observe S3 , R3
A3
Choose A3 using policy derived from Q (e.g., ε-greedy)
Updating :
$Q_{k+1}(S_2, A_2) = Q_k(S_2, A_2) + \alpha[R_3 + \gamma Q_k(S_3, A_3) - Q_k(S_2, A_2)]$

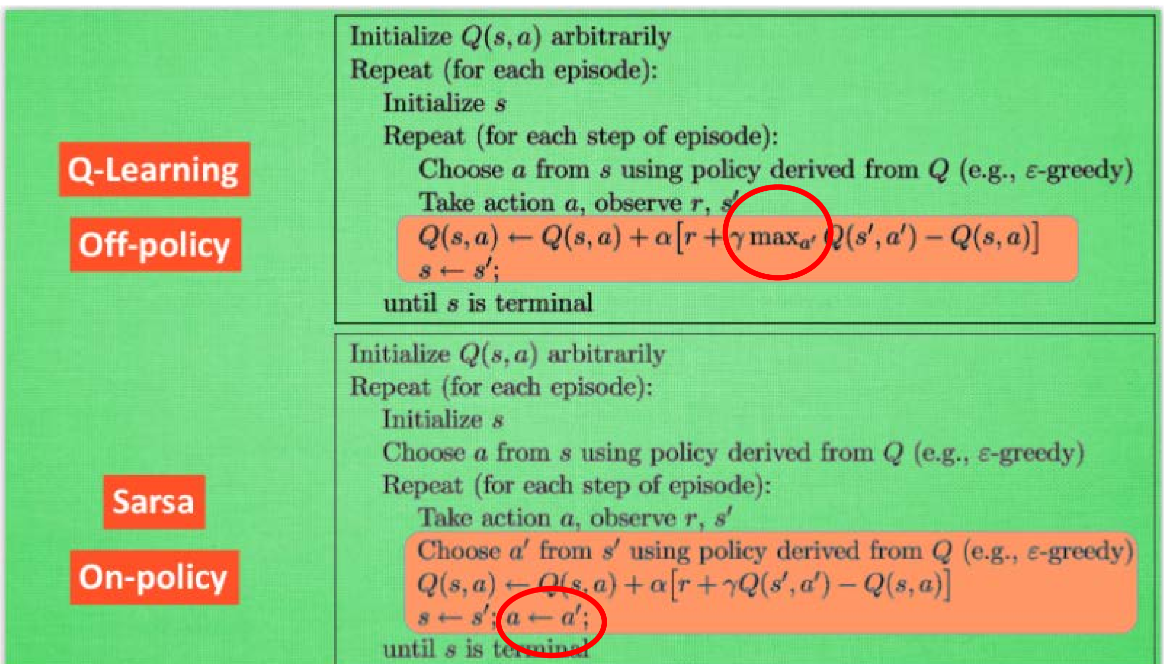
國立清華大學
National Tsing Hua University

INNOVATION ENGINEERING LAB

程式碼

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



程式碼-env

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
```

程式碼-env

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()
```

程式碼-env

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])
```

程式碼-env

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()
```

程式碼-env

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)
```

National Tsing Hua University 國立清華大學

INNOVATION ENGINEERING LAB

程式碼-env

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)
```

程式碼-env

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
```

程式碼-Q-learning

◆ RL_brain

1. 設置QLearningTable及初始值

```python
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。

```python
# 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
```

程式碼-Q-learning

- RL_brain

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,
            )
```

程式碼-Q-learning

- Run

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

更新制度，每回合皆會印出Q-Table及步數。

```python
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 : {}\n'.format(step_count))
            if done:
                step_list.append(step_count)
                break
```

```python
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()
```

程式碼-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,
            )
```

程式碼-Sarsa

- RL_brain

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
```

程式碼-Sarsa

- RL_brain

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)
```

程式碼-Sarsa

- Run

```python
import matplotlib.pyplot as plt

from maze_env_sarsa import Maze
from RL_brain_sarsa import SarsaTable
```

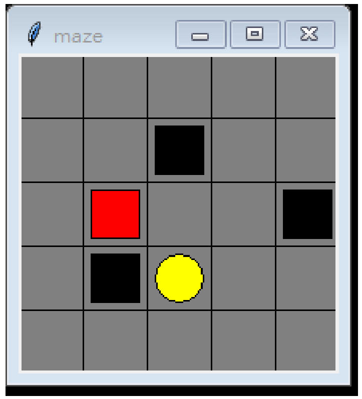
更新機制

```python
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_))
```

```python
            # 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 : {}\n'.format(step_count))
            #temp = format(step_count)
            if done:
                step_list.append(step_count)
                break

    print(step_list)

    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()
```

Q-Learning

- 5次




```
[17, 6, 3, 9, 34]
```


```
                                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
```

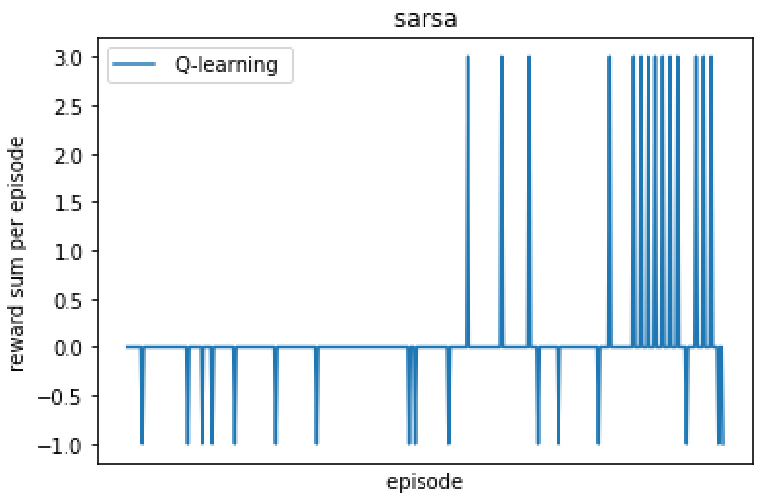
Q-Learning

◆ 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]

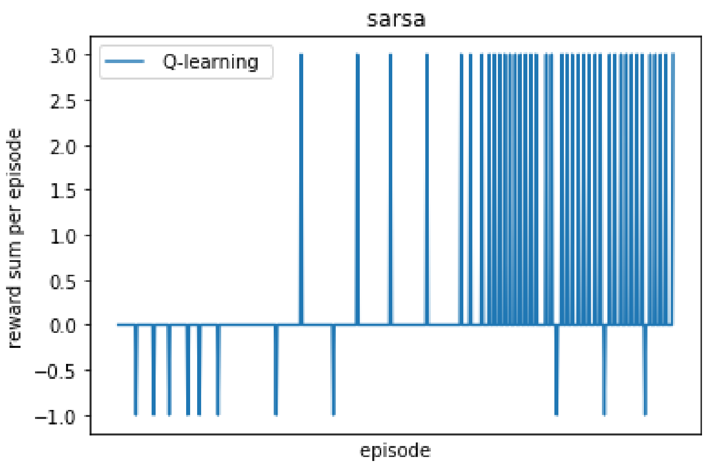


Q-Learning

◆ 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

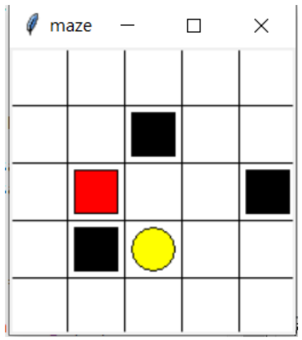
game over，總步數 : 7



[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, 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]

Sarsa

- 5次



[6, 3, 12, 13, 14]

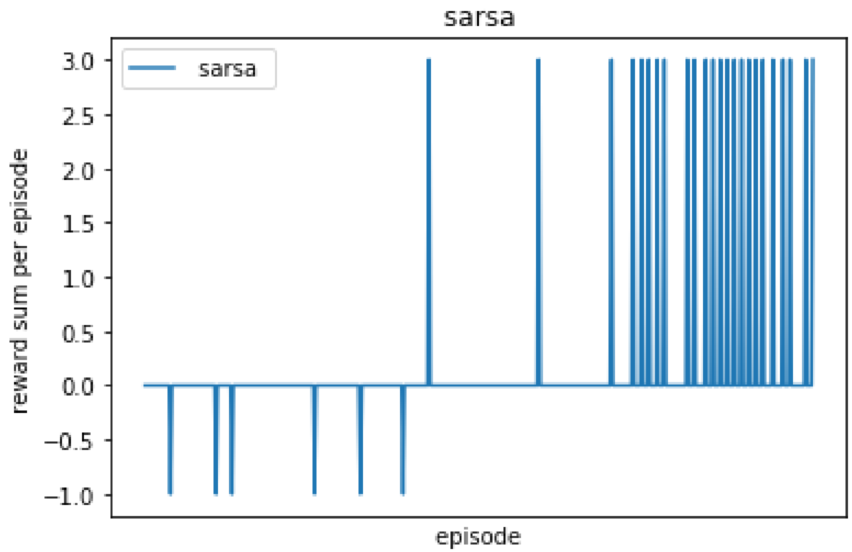

```
                                 0      1      2     3
[5.0, 5.0, 35.0, 35.0]        0.00   0.00   0.00   0.0
[5.0, 45.0, 35.0, 75.0]       0.00   0.00   0.00   0.0
[45.0, 45.0, 75.0, 75.0]      0.00   0.00   0.00   0.0
[45.0, 85.0, 75.0, 115.0]     0.00  -0.01   0.00   0.0
terminal                      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
```

Sarsa

- 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, 7, 7, 5, 11, 5]
```

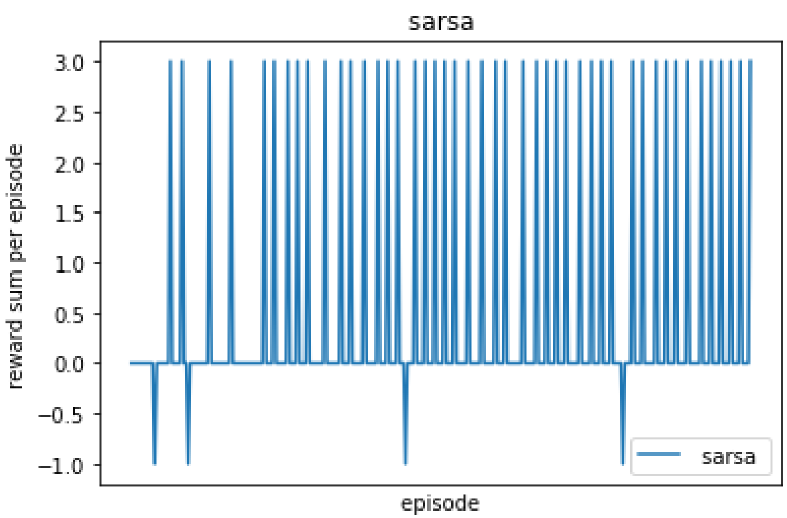
Sarsa

◆ 50次

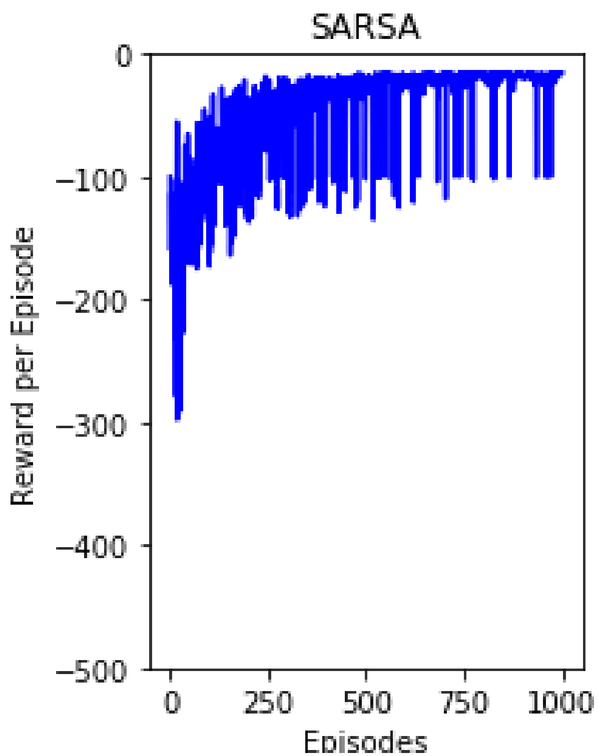
	0	1	2	3
[5.0, 5.0, 35.0, 35.0]	3.442650e-08	0.000180	7.267862e-10	7.076418e-07
[45.0, 5.0, 75.0, 35.0]	0.000000e+00	0.000000	0.000000e+00	1.782417e-07
[5.0, 45.0, 35.0, 75.0]	5.550029e-08	0.002424	9.330180e-07	4.838749e-06
[5.0, 85.0, 35.0, 115.0]	4.239968e-06	0.000000	2.535388e-02	0.000000e+00
[5.0, 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

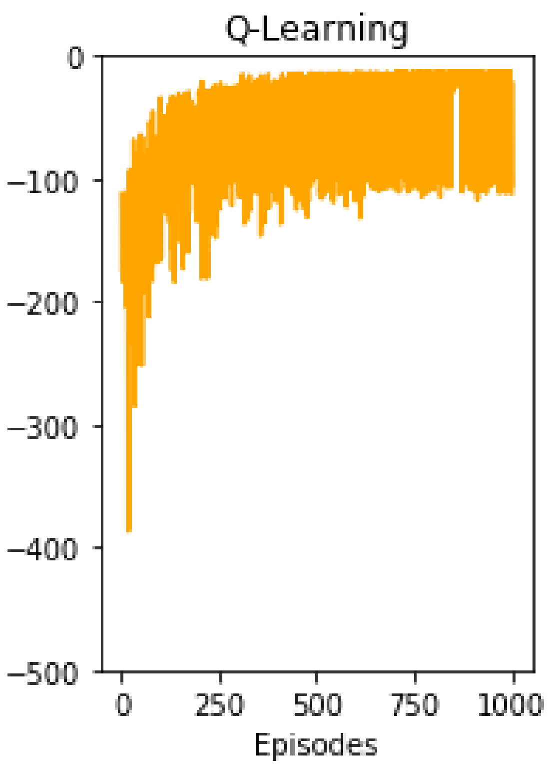
game over，總步數 : 5

[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]



Q-Learning v.s Sarsa



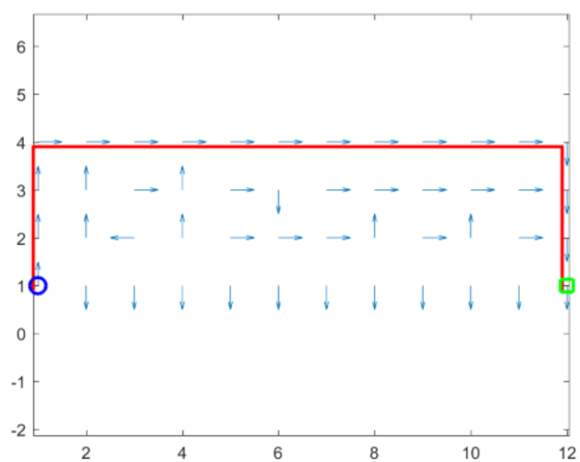


1. Sarsa獎勵值比較好，有較多接近0(在線學習效果比較好)
2. sarsa收斂快速，直觀簡單且較保守，走安全路徑路線但很容易陷入局部最小值
3. Q-learning 直接用最大值來估計，因此有學習到全局最優的能力，

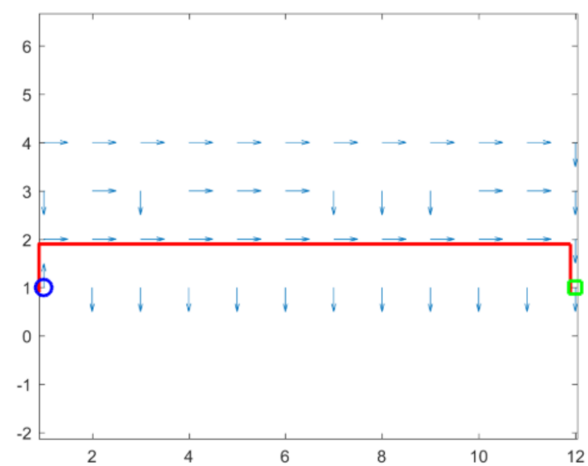
所以有更好的最終性能，但是收斂慢，需要更長的時間來學習，

結論

- 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Reward結果較低。
- 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
- 對於欲快速找到最佳路徑的問題，則可以使用Q-learning。



Sarsa



Q-learning

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。

References

- 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

- Sarsa

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

References

- 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.

National Tsing Hua University

國立清華大學

INNOVATION ENGINEERING LAB

謝謝聆聽!

新年快樂~