Practical_6_Reinforcement Learning-2

October 31, 2023

1 Practical 6: Reinforcement Learning

Author: CAMERON STROUD

Student Number: n11552123

1.0.1 Learning Outcomes:

- Implement the Q-learning and SARSA for Cliff Walking environment
- Compare the difference between these two algorithms

We will require the following library for this practical (Import all necessary libraries before running the codes):

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

import time

import gymnasium as gym


import os
from IPython.display import clear_output

from tqdm import tqdm
```

1.1 The Cliff Walking Environment

Cliff walking involves crossing a gridworld from start to goal while avoiding falling off a cliff.

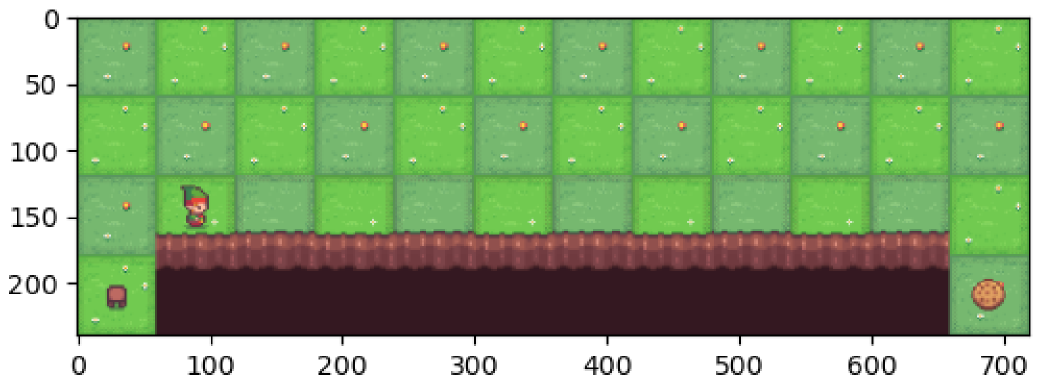
The game starts with the player at location [3, 0] of the 4x12 grid world with the goal located at [3, 11]. If the player reaches the goal the episode ends.

A cliff runs along [3, 1..10]. If the player moves to a cliff location it returns to the start location.

The player makes moves until they reach the goal.

The observation is the player's current position. The action space consists of "left, down, right, up". Each time step incurs -1 reward, unless the player stepped into the cliff, which incurs -100 reward.

```
[ ]: os.environ["SDL_VIDEODRIVER"] = "dummy"

     env = gym.make("CliffWalking-v0", render_mode="rgb_array")
     env.action_space.seed(42)

     state, info = env.reset(seed=42)

     for _ in range(20):
         action = env.action_space.sample()  # this is where you would insert your␣
      ↪policy
         observation, reward, terminated, truncated, info = env.step(action)

         if terminated or truncated:
             observation, info = env.reset()

         clear_output(wait=True)
         plt.imshow( env.render() )
         plt.show()
         env.render()

     env.close()
```



1.2 Part A: Q learning

1.2.1 Q1

Observe the Cliff Walking environment, and intuitively determine the optimal path. Explain why you choose this action.

[UP, RIGHT, RIGHT, RIGHT, RIGHT, RIGHT, RIGHT, RIGHT, RIGHT, RIGHT, RIGHT, RIGHT, DOWN]
-1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 = -13

Each step incurs a -1 reward so the least number of moves would be the optimal path. This would be ideally directly to the right but the cliffs incur -100, and a step upwards to skirt them (and downwards to reach the goal) is required.

1.2.2 Q2

Implement Q learning algorithm to derive an optimal policy for Cliff Walking environment.

```
[ ]: # Create an environment
     env = gym.make("CliffWalking-v0")
```

```
[ ]: def render_env(episode, iteration):
         clear_output(wait=True)
         plt.imshow( env.render() )
         plt.title(f'E{episode} k{iteration}')
         plt.show()
         env.render()
```

```
[ ]: num_episodes = 1000  # Number of training episodes
     discount_factor = 0.99  # Discount factor
     learning_rate = 1  # Learning rate
     exploration_rate = 1  # Probability of exploration
```

```
[ ]: # Establish a Q-table
     num_states = env.observation_space.n
     num_actions = env.action_space.n
     Q_table = np.zeros((num_states, num_actions))

     # Q-learning algorithm
     for episode in tqdm(range(num_episodes)):
         debug = False
         state,_ = env.reset()
         done = False
         k = 1

     #     learning_rate = 1  # Learning rate
     #     exploration_rate = 1  # Probability of exploration

         while not done:

             # Hint: choose an action based on epsilon-greedy policy and update␣
      ↪Q-value using the Q-learning update rule
             ### START CODE HERE ###
             # Choose an action based on epsilon-greedy policy
             if np.random.rand() < exploration_rate:
                 action = env.action_space.sample()
             else:
                 action = np.argmax(Q_table[state, :])
```

```
        # Perform the chosen action
        next_state, reward, terminated, truncated, _ = env.step(action)

        # Update Q-value using the Q-learning update rule
        Q_table[state, action] = Q_table[state, action] + learning_rate *␣
↪(reward + discount_factor * np.max(Q_table[next_state, :]) - Q_table[state,␣
↪action])
        state = next_state

        ### END CODE HERE ###

        k = k + 1
        learning_rate = 1 / k  # Learning rate
        exploration_rate = 1 / k  # Probability of exploration

        if terminated or truncated:
            done=True

        if not (k % 1000000):
            debug = True
        if debug:
            render_env(episode, k)
    # print(k)

env.close()
```

```
100%|     | 2000/2000 [00:01<00:00, 1342.11it/s]
```

[ ]:
```
print(Q_table)
```

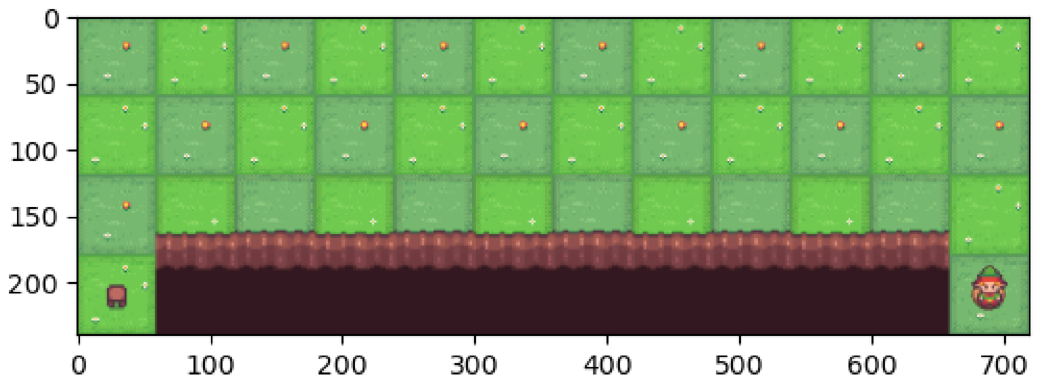
```
[[ -12.09284504  -11.96566475  -11.9541471   -12.20494975]
 [ -11.25950287  -11.1830121   -11.35899128  -11.73455116]
 [ -10.44725464  -10.38990905  -10.5377361   -10.62428201]
 [  -9.59497773   -9.58776478   -9.62202674   -9.64848448]
 [  -8.8542831    -8.76602627   -8.75430774   -9.00069367]
 [  -7.98744826   -7.94039478   -7.95498289   -8.06626977]
 [  -7.15233197   -7.09734559   -7.12959275   -7.40721666]
 [  -6.3006688    -6.25327885   -6.27274198   -6.43461023]
 [  -5.44567411   -5.41749001   -5.43753762   -5.45021089]
 [  -4.57940423   -4.56365881   -4.58522468   -4.69913067]
 [  -3.81225544   -3.73400283   -3.72992651   -3.74946604]
 [  -2.92829238   -2.93531252   -2.9122044    -3.00931919]
 [ -12.74804344  -12.24087032  -12.24766853  -13.09605283]
 [ -11.85853779  -11.35509375  -11.35869664  -13.02559833]
 [ -10.91497435  -10.46021978  -10.46145235  -12.00523479]
 [ -10.03158578   -9.55645962   -9.55659131  -10.38763856]
 [  -9.15199917   -8.64377591   -8.64381237   -9.86733883]
 [  -7.98667685   -7.7216592    -7.72249577   -8.43268599]
 [  -7.13338507   -6.79041076   -6.79061398   -7.05812948]
 [  -6.04456314   -5.8496456    -5.84969116   -6.78245126]
 [  -5.28009822   -4.89930703   -4.89945952   -5.23542314]
 [  -4.12997103   -3.93934441   -3.93937223   -4.3752134 ]
 [  -3.28614116   -2.96964536   -2.96964941   -3.24325105]
 [  -2.19779902   -2.2352524    -1.98999671   -2.20341596]
 [ -13.11788302  -11.36151283  -13.12541872  -12.2478977 ]
 [ -12.24025159  -10.46617457 -112.12541872  -12.2478977 ]
 [ -11.35332822   -9.5617925  -112.1254187   -11.36151272]
 [ -10.44887539   -8.64827525 -112.1251611   -10.46588082]
 [  -9.4960518    -7.72553056 -112.08297723   -9.56099948]
 [  -8.5746789    -6.79346521 -111.97602136   -8.64497567]
 [  -7.62954319   -5.85198506 -110.00593814   -7.70655723]
 [  -6.55649778   -4.90099501 -102.93848866   -6.72999639]
 [  -5.73375911   -3.940399    -98.17200564   -5.66962066]
 [  -4.39655601   -2.9701       -97.9930811    -4.63215226]
 [  -3.62917761   -1.99         -99.75739192   -3.60360955]
 [  -2.5739111    -1.79764977   -1.            -2.53335361]
 [ -12.2478977  -112.12541872  -13.1254187   -13.1254187 ]
 [   0.            0.            0.            0.        ]
 [   0.            0.            0.            0.        ]
 [   0.            0.            0.            0.        ]
 [   0.            0.            0.            0.        ]
 [   0.            0.            0.            0.        ]
 [   0.            0.            0.            0.        ]
 [   0.            0.            0.            0.        ]
 [   0.            0.            0.            0.        ]
 [   0.            0.            0.            0.        ]
 [   0.            0.            0.            0.        ]
 [   0.            0.            0.            0.        ]]
```

1.2.3 Q3

Evaluate the optimal policy obtained by Q learning in Cliff Walking environment.

```
[ ]: """Evaluate the agent trained by Q-learning"""
     os.environ["SDL_VIDEODRIVER"] = "dummy"

     env = gym.make("CliffWalking-v0", render_mode="rgb_array")
     env.action_space.seed(42)

     state, info = env.reset(seed=42)
     done = False

     path = []
     action_space = ['up', 'right', 'down', 'left']
     while not done:
```

```
    # Hint: choose the action by the Q table and transit to the next state
    ### START CODE HERE ###
    action = np.argmax(Q_table[state, :])# this is where you would insert your␣
↪policy
    path.append(action_space[action])
    ### END CODE HERE ###
    state, reward, terminated, truncated, info = env.step(action)

    clear_output(wait=True)
    plt.imshow( env.render() )
    plt.show()
    env.render()

    if terminated or truncated:
        done = True

env.close()
```



[ ]:
```
print(path)
```

```
['up', 'right', 'right', 'right', 'right', 'right', 'right', 'right', 'right',
'right', 'right', 'right', 'down']
```

1.2.4 Q4

Does the optimal policy align with your initial expectation in Q1? Provide an explanation for the observed outcome.

Yes, the optimal policy aligns with my initial expectations from Q1. As the cliff has such a large cost, the Q-learning algorithm learns to avoid actions that result in those states by iterating the Q-table as it explores.

1.3 Part B: SARSA

```
[ ]: # Create an environment
     env = gym.make("CliffWalking-v0")
```

```
[ ]: num_episodes = 10000  # Number of training episodes
     discount_factor = 0.99  # Discount factor
     learning_rate = 1  # Learning rate
     exploration_rate = 0.1  # Probability of exploration
```

```
[ ]: # Establish a Q-table
     num_states = env.observation_space.n
     num_actions = env.action_space.n
     Q_table = np.zeros((num_states, num_actions))

     # SARSA algorithm
     for episode in tqdm(range(num_episodes)):
         debug = False
         state, _ = env.reset()
         action = env.action_space.sample()
         done = False
         k = 1

         while not done:

             # Hint: take action and observe next state and reward, then choose the␣
      ↪next action based on epsilon-greedy policy
             # and update Q-table using the SARSA update rule
             ### START CODE HERE ###

             # Take action and observe next state and reward
             next_state, reward, terminated, truncated, _ = env.step(action)

             # Choose the next action based on epsilon-greedy policy
             if np.random.rand() < exploration_rate:
                 next_action = env.action_space.sample()
             else:
                 next_action = np.argmax(Q_table[next_state, :])

             # Update Q-table using the SARSA update rule
             Q_table[state, action] = Q_table[state, action] + learning_rate *␣
      ↪(reward + discount_factor * Q_table[next_state, next_action] -␣
      ↪Q_table[state, action])

             state = next_state
             action = next_action
             ### END CODE HERE ###
```

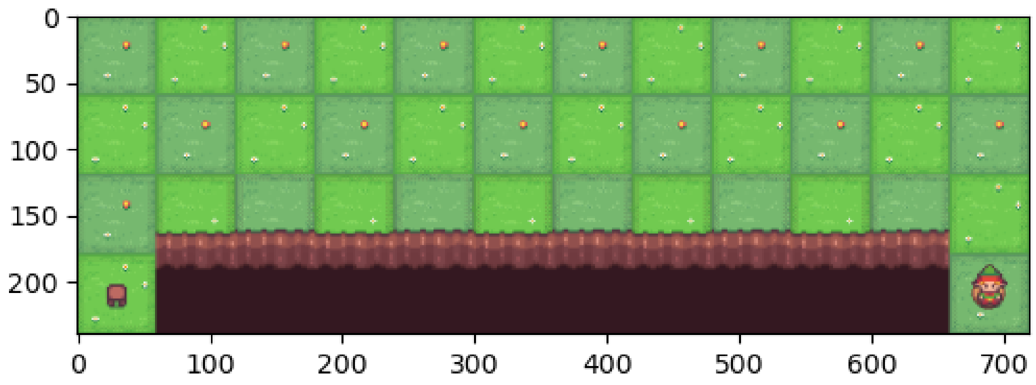
```
        k = k + 1
        learning_rate = 1 / k  # Learning rate
        # exploration_rate = 1 / k  # Probability of exploration

        if terminated or truncated:
            done=True

        if not (k % 1000000):
            debug = True
            break
        if debug:
            render_env(episode, k)

print(Q_table)
```

```
100%|     | 10000/10000 [00:03<00:00, 2801.02it/s]
```

```
[[ -15.32836873  -14.41427061  -16.39720059  -15.54664308]
 [ -14.34365982  -13.51046835  -14.62287453  -16.02483109]
 [ -13.37514349  -12.47110662  -13.52416806  -14.57993202]
 [ -12.57969857  -11.48280437  -12.74654126  -13.70155259]
 [ -11.3891133   -10.34107641  -10.76436796  -13.01702098]
 [ -10.43552243   -9.37038869  -10.14352148  -11.62448658]
 [  -9.44435314   -8.35807878   -9.10065122  -10.56162819]
 [  -8.50724814   -7.32051187   -7.67907344   -9.53135849]
 [  -7.48792499   -6.3777397    -6.73452225   -8.68423067]
 [  -6.4354745    -5.39219693   -5.75518635   -7.5568578 ]
 [  -5.41265525   -4.65577871   -4.49343343   -6.52585751]
 [  -4.32261665   -4.45281771   -3.36228254   -5.52168215]
 [ -15.41592901  -14.34898622  -17.25036868  -15.40160554]
 [ -14.434962    -13.67060372  -17.92168925  -15.72357013]
 [ -13.63491102  -12.13874926  -14.99112227  -14.60663426]
 [ -12.47901163  -11.01570738  -13.62653275  -13.4291368 ]
 [ -11.38276158  -10.16147424  -11.96817411  -12.36549321]
 [ -10.408342     -9.23893913  -14.42942247  -11.4213103 ]
 [  -9.44783711   -8.34656298  -10.0459888   -10.25310377]
 [  -8.33591292   -7.02651222   -9.54926454   -9.05583555]
 [  -7.51202384   -5.75073513   -8.12343761   -8.16454308]
 [  -6.33389332   -4.32950415   -6.83855481   -6.96012644]
 [  -5.63753816   -3.21304208   -7.48182222   -5.51604565]
 [  -4.43665449   -3.30740954   -2.08649347   -4.44331138]
 [ -15.23457814  -17.12715921  -17.6793631   -16.69536931]
 [ -17.86355684  -23.60597125 -117.21141594  -16.0896002 ]
 [ -13.12057819  -15.96728624  -94.59160496  -16.54081987]
 [ -12.44508044  -14.12790474  -66.21876388  -16.13364698]
 [ -11.23045201  -10.98595542  -51.13620656  -11.86562823]
 [ -10.36034542  -12.77582374  -69.10577096  -10.50760421]
[  -8.88350292  -10.53726635  -66.93578094   -9.59686284]
[  -7.70423639   -7.7080079   -81.99183951   -7.91676366]
[  -6.58687177   -9.76545738  -46.40356026   -7.88709145]
[  -5.63610411   -9.25399242  -54.69602356   -5.85497102]
[  -3.72766948   -2.20308411  -82.65637455   -4.92380462]
[  -3.55397508   -2.40417418   -1.            -4.55735921]
[ -16.72351745 -116.337911    -43.33871583  -17.56021019]
[   0.            0.            0.            0.        ]
[   0.            0.            0.            0.        ]
[   0.            0.            0.            0.        ]
[   0.            0.            0.            0.        ]
[   0.            0.            0.            0.        ]
[   0.            0.            0.            0.        ]
[   0.            0.            0.            0.        ]
[   0.            0.            0.            0.        ]
[   0.            0.            0.            0.        ]
[   0.            0.            0.            0.        ]
[   0.            0.            0.            0.        ]]
```

[ ]:
```
"""Evaluate the agent trained by Q-learning"""
os.environ["SDL_VIDEODRIVER"] = "dummy"

env = gym.make("CliffWalking-v0", render_mode="rgb_array")
env.action_space.seed(42)

state, info = env.reset(seed=42)
done = False

while not done:

    # Hint: choose the action by the Q table and transit to the next state
    ### START CODE HERE ###
    action = np.argmax(Q_table[state, :])# this is where you would insert your␣
 ↪policy
    # path.append(action_space[action])
    ### END CODE HERE ###
    state, reward, terminated, truncated, info = env.step(action)

    clear_output(wait=True)
    plt.imshow( env.render() )
    plt.show()
    env.render()

    if terminated or truncated:
        done = True
```

```
env.close()
```



1.3.1 Q5

Does the optimal policy align with your intuition? Provide an explanation for the observed outcome.

No, the optimal policy does not align with my intuition, as it takes seemingly two additional steps upwards, resulting in another 2 steps downwards once its aligned with the goal.

1.3.2 Q6

Do Q-learning and SARSA choose the same paths to the goal? If not, try to increase the number of SARSA's episodes and observe the result. Explain why.

Q-learning is an off-policy algorithm, while SARSA is on-policy, resulting in different explorations of the grid. This means SARSA takes an action before updating its Q-table, while Q-learning updates based on the maximum Q-value of the next state. Increasing the number of SARSA episodes was not able to course-correct at 100,000 episodes. As SARSA applies its currently-optimal action before updating it often gets stuck taking inefficient actions (ie. switching between down and left in the starting square) and a threshold was required in the iteration count. Increasing the episode count resets the exploration rate more, which should give it ample opporutinity to discover new paths, but only within the first few iterations of an episode before it decays, and only if the exploration is initially set high enough

1.3.3 Q7

In the code of SARSA, the exploration probability is $1/k$ as in the lecture note. Now, use a constant exploration probability such as 0, 0.1, 0.2, ..., and observe the optimal policy. How will modifying this parameter impact the optimal path of SARSA? Explain the results.

Setting the exploration rate to a constant mitigates the amount to which SARSA algorithm exploits suboptimal actions, and enables it to learn other state costs. At 10,000 episodes and an exploration rate of 0.2, SARSA no longer goes to the top, and instead takes the middle route, an improvement over the prior suboptimal policy.