6ELEN018W - Tutorial 5 Exercises

Exercise 1

Consider the gridworld problem shown in Figure 1. An agent needs to move to the target goal position **G** starting from any cell (arbitrary position) by following the optimum policy which gathers the largest reward.

The immediate rewards $r(s, a)$ for the transition from one state to another for this problem are also shown in Figure 1. For example, the reward received by moving from state s_1 to s_2 by taking the *right* action is 0. The reward for moving from state s_5 to state G (i.e. taking the *north* action) is 100.

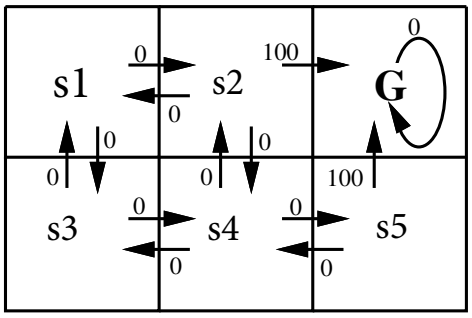


Figure 1: $r(s, a)$ (immediate reward) values.

Implement in a programming language of your choice (Python or Java) the Q-learning applied to the problem of the previous Figure, until the Q values converge and they do not change any more.

Hint: You need to consider multiple episodes, i.e. after the terminal (final, goal state) is reached, a new episode is started. The initial values of the Q values for the new episode will be the ones that the previous episode had calculated.

Check your derived Q values with the Figure 2 in the lecture slide 21.

Do not look at the solution provided until you spend sufficient time working out your own solution.

Exercise 2

Install the gymnasium environment in your computer (if you are using a lab computer, do this within the Anaconda environment)

Exercise 3

Implement the lunar lander code given in the lecture which uses the random policy.

Exercise 4

Implement the code for the taxi driver problem given in the lecture which follows a random policy.

Exercise 5

Can you improve the performance of the random policy by using simple logical operations. E.g. if a condition is met, then take a specific action.

Hint: For example, choose the pick up action when passing through the location of the passenger. What about moving towards the passenger, can you think of a simple way of doing so?

Sample Solution to Exercise 1

The following sample solution code can be also found as a Python notebook format in:

`https://ddracopo.github.io/DOCUM/courses/6elen018w/gridworld.ipynb`

```
import random

#*************************************************************
#************ author: Dimitris C. Dracopoulos *****************
#*************************************************************

# delta (see lecture slides formulas) is a dictionary containing
# the states of the dynamic system and mapping them to a dictionary
# {a: s'} where s' is the new state that we arrive when we apply
# action a when in state s

# We assume that states, s_1, s_2 to s_6 are assigned by row traversing:
# ------------
# |s1 |s2 |s3|
# |s4 |s5 |s6|
# ------------

delta = {}
Q = {}

# 'N' is go North action
# 'E' is go East action
# 'S' is go South action
# 'W' is go West action

delta['s1'] = {'N':'s1', 'E':'s2', 'S':'s4', 'W':'s1'}

delta['s2'] = {'N':'s2', 'E':'s3', 'S':'s5', 'W':'s1'}

delta['s3'] = {'N':'s3', 'E':'s3', 'S':'s6', 'W':'s2'}

delta['s4'] = {'N':'s1', 'E':'s5', 'S':'s4', 'W':'s4' }

delta['s5'] = {'N':'s2', 'E':'s6', 'S':'s5', 'W':'s4'}

delta['s6'] = {'N':'s3', 'E':'s6', 'S':'s6', 'W':'s5'}

# all rewards are zero, except those leading to state s3

# initialise all Q with 0
Q['s1'] = {'N':0, 'E':0, 'S':0, 'W':0}
Q['s2'] = {'N':0, 'E':0, 'S':0, 'W':0}
Q['s3'] = {'N':0, 'E':0, 'S':0, 'W':0}
Q['s4'] = {'N':0, 'E':0, 'S':0, 'W':0}
```

```
Q['s5'] = {'N':0, 'E':0, 'S':0, 'W':0}
Q['s6'] = {'N':0, 'E':0, 'S':0, 'W':0}


# action 1 is North
# action 2 is East
# action 3 is South
# action 4 is West
def num2Action(action):
    if action == 1:
        a = 'N'
    elif action == 2:
        a = 'E'
    elif action == 3:
        a = 'S'
    else:
        a = 'W'

    return a


# find and return the action which will lead to the
# maximum Q, given the current state s
def findMaxAction(s):
    max_so_far = Q[s]['N']
    max_action = 'N'

    if Q[s]['E'] > max_so_far:
        max_so_far = Q[s]['E']
        max_action = 'E'

    if Q[s]['S'] > max_so_far:
        max_so_far = Q[s]['S']
        max_action = 'S'

    if Q[s]['W'] > max_so_far:
        max_so_far = Q[s]['W']
        max_action = 'W'

    return max_action


# display Q values
def displayQ():
    print('s1: -> ', Q['s1'])
    print('s2: -> ', Q['s2'])
    print('s3: -> ', Q['s3'])
    print('s4: -> ', Q['s4'])
    print('s5: -> ', Q['s5'])
    print('s6: -> ', Q['s6'])
```

```
# apply Q-learning
def Q_learning():
    gamma = 0.9

    # Run many episodes
    for e in range(10000):
        # Run 1 episode of the problem (i.e. until state 'G' is reached)

        # Start with from random state s
        s = random.randint(1, 6)
        s = 's' + str(s)  # current state

        while s != 's3':
            # 1. Select an action and execute it
            a = random.randint(1, 4)

            a = num2Action(a)

            # 3. Observe the new state s'
            s_new = delta[s][a]

            if __debug__:
                print(s, '(', a, ')', '->', s_new)

            # 3. Receive immediate reward r
            # if next state is the goal state G then the reward is 100,
            # otherwise it is 0
            if s_new == 's3':
                r = 100
            else:
                r = 0

            # 4. Update the table entry for \hat{Q}(s,a) as follows:
            #\Qhat(s,a) -> r + gamma*max_{a'}[Qhat(s',a')]
            # where s' is the next state after applying action a in state s
            if s_new != s:
                a_next = findMaxAction(s_new)
                Q[s][a] = r + gamma*Q[s_new][a_next]

                # 5. s <- s'
                # The new state becomes the next current state
                s = s_new

    displayQ()


# the starting point of program execution
def main():
```

```
    Q_learning()


if __name__ == "__main__":
    main()
```

Exercise 2

Consider the grid in Figure 2. An agent can move in either of the four directions starting from S and finishing in the goal state G.

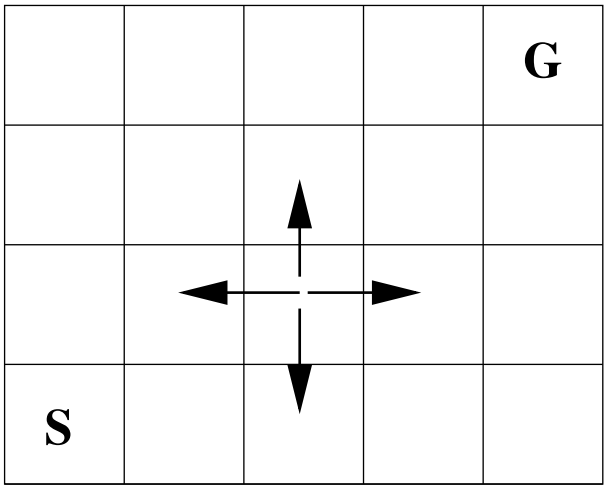


Figure 2: The grid world example for the application of reinforcement learning techniques.

1. If the reward on reaching the goal is 100, and all other rewards between state transitions are 0, write a program in Python which uses Q learning to learn the optimal policy. Assume that $\gamma = 0.9$.
2. What are the actions of the optimal policy?

Solution

This should be based on the previous Exercise code, the only changes is in the initialisation of the dictionaries for the different states. The number of states is larger here, but the Q-learning code is the same.

Exercise 3

Consider the previous exercise: how does the optimal policy change if another goal state is added to the lower right corner with reward -100.

Hint: To see this, rerun your implementation with the additional goal.

Solution

Just a modification of the calculation of the reward.