

CS 3120

Machine Learning and Neural Computing

Reinforcement Learning

Assignment

Name : D. R. R. Wijewardene

Index No. : S14245

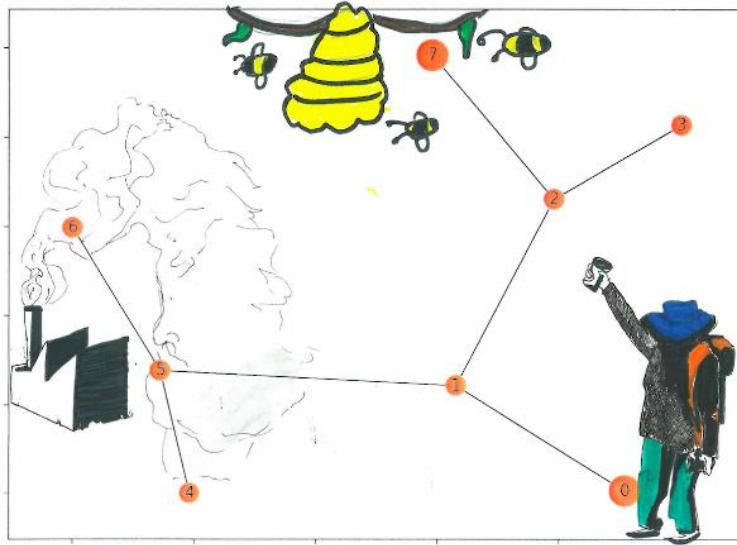
Reg. No. : 2018s16778

Faculty : Faculty of Science (Bioinformatics)

Contents

		Page No.
Part 1	Practice the code examples given in https://www.viralml.com/video-content.html?v=nSxaG_Kjw_w using Google Colab.	2
Part 2	Model the Example 1 problem discussed in the lecture note using Python and compare the outputs/results (Octave vs. Python)	3
Part 3	Give your answer by modeling the Exercise problem (maze solving) given in the lecture note.	7

Part 1 - Practice the code examples given in https://www.viralml.com/video-content.html?v=nSxaG_Kjw_w using Google Colab.



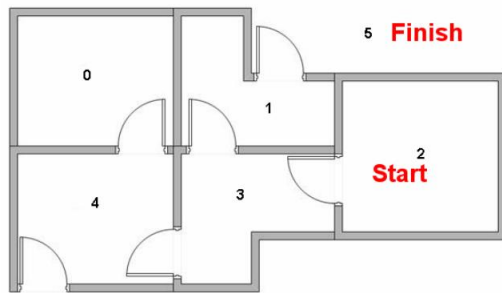
Colab Notebook Link:

https://colab.research.google.com/drive/1OOfacIVx6gZpFuMxNjskLtMk27KSz_Bf?usp=sharing

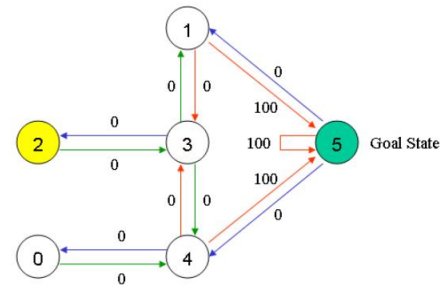
The “Reinforcement Learning - Simple Python Q-learning Example” given in the provided website was demonstrated, and the expected results were obtained.

Part 2 - Model the Example 1 problem discussed in the lecture note using Python and compare the outputs/results (Octave vs. Python)

An agent has to travel from one room (start state) to another (goal state) in the following house



Representing the goal using immediate rewards



Room Number	Python Index	Octave Index
0	0	1
1	1	2
2 - START	2	3
3	3	4
4	4	5
5 - END	5	6

Since the indexing of Python and Octave are different, the indexes used to indicate each room/state are also different. Therefore, the final output should be compared accordingly.

The Python and Octave implementations and result comparisons are as follows;

Python Implementation

Colab Notebook Link:

https://colab.research.google.com/drive/1a0yd2u2932yQRogUi9d-ca_RfJFgHXLe?usp=sharing

Octave Implementation

```

1 % Reinforcement Learning Assignment - Part 2
2 % S14245 - D. R. R. Wijewardene
3
4 disp("Best path to travel from one room (start) to another (final) in a house")
5
6
7 % Reward table
8 disp("Reward Table")
9 R = [-inf, -inf, -inf, -inf, 0, -inf;
10      -inf, -inf, -inf, 0, -inf, 100;
11      -inf, -inf, -inf, 0, -inf, -inf;
12      -inf, 0, 0, -inf, 0, -inf;
13      0, -inf, -inf, 0, -inf, 100;
14      -inf, 0, -inf, -inf, 0, 100]
15
16 gamma = 0.8; % learning rate
17 goalState = 6;
18
19 % Q table
20 q = zeros(size(R));
21 disp("Initial Q-table")
22 q
23
24 % Exploration
25
26 for episode = 1:1000
27     % Select a random initial state
28     y = randperm(size(R,1));
29     state = y(1);
30
31     % Find all possible actions from the state
32     actions = find(R(state,:) >= 0);
33
34     if size(actions, 2) > 0
35         % Select one action randomly
36         i = randperm(size(actions,2));
37         action = actions(i(1));
38     end
39
40     % Return a column vector with the max values of each row
41     qMax = max(q, [], 2);
42
43     % Compute the q values
44     q(state, action) = R(state, action) + gamma * qMax(action);
45
46     % Translation to the next state
47     state = action;
48
49 end
50
51 disp("Q-table after 1000 episodes of training (not normalized)")
52 q
53
54 disp("Normalized Q-table after 1000 episodes of training")
55 q*100/max(max(q))
56
57 % Exploitation
58 disp("Starting room = 2 (state = 3)")
59 disp("Final goal = 5 (state = 6)")
60 disp("Most efficient path:")
61
62 state = 3; % Set the initial state
63
64 while state ~= goalState
65     % From current state, find the action with the highest Q value
66     [mx, action] = max(q(state,:));
67     % Take the action (transition to the next state)
68     state = action;
69 end
70
71 state

```

Comparison between Octave and Python tables and outputs/results;

Octave	Python
Rewards Table	
<pre>Reward Table R = -Inf -Inf -Inf -Inf 0 -Inf -Inf -Inf -Inf 0 -Inf 100 -Inf -Inf -Inf 0 -Inf -Inf -Inf 0 0 -Inf 0 -Inf 0 -Inf -Inf 0 -Inf 100 -Inf 0 -Inf -Inf 0 100</pre>	<pre>R matrix([[-1., -1., -1., -1., 0., -1.], [-1., -1., -1., 0., -1., 100.], [-1., -1., -1., 0., -1., -1.], [-1., 0., 0., -1., 0., -1.], [0., -1., -1., 0., -1., 100.], [-1., 0., -1., -1., 0., 100.]])</pre>
Initial Q Table	
<pre>Initial Q-table q = 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</pre>	<pre>Q matrix([[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.]])</pre>
Trained Q Table	
<pre>Normalized Q-table after 1000 episodes of training ans = 0 0 0 0 79.9999 0 0 0 0 63.9999 0 100.0000 0 0 0 64.0000 0 0 0 80.0000 51.2000 0 79.9999 0 63.9999 0 0 64.0000 0 99.9998 0 80.0000 0 0 79.9999 100.0000</pre>	<pre>Trained Q matrix: [[0. 0. 0. 0. 79.99995211 0.] [0. 0. 0. 64. 0. 100.] [0. 0. 0. 63.99996169 0. 0.] [0. 80. 51.19996935 0. 79.99995211 0.] [63.99996169 0. 0. 64. 0. 99.99994014] [0. 80. 0. 0. 79.99995211 99.99994014]]</pre>
Most Efficient Path from Room 2 to Room 5 Python : Index 2 -> 5 Octave : Index 3 -> 6	
<pre>Most efficient path: state = 3 state = 4 state = 2 state = 6</pre>	<pre>Most efficient path: [2, 3, 1, 5]</pre>

Comparison of Octave and Python trained Q tables;

		0	1	2	3	4	5
0	Octave	0	0	0	0	79.99	0
	Python	0	0	0	0	79.99	0
1	Octave	0	0	0	63.99	0	100.0
	Python	0	0	0	64.00	0	100.0
2	Octave	0	0	0	64.00	0	0
	Python	0	0	0	63.99	0	0
3	Octave	0	80.0	51.20	0	79.99	0
	Python	0	80.0	51.19	0	79.99	0
4	Octave	63.99	0	0	64.0	0	99.99
	Python	63.99	0	0	64.0	0	99.99
5	Octave	0	80.0	0	0	79.99	100.00
	Python	0	80.0	0	0	79.99	99.99

The trained Q-tables for this situation are approximately similar in both Octave and Python implementations. This is because both are trained until the end of the learning process. If this was done only for a few iterations, these tables might be different because they train for random episodes.

The final output which gave the most efficient path to reach the goal (room 5) when the agent began from room 2 is;

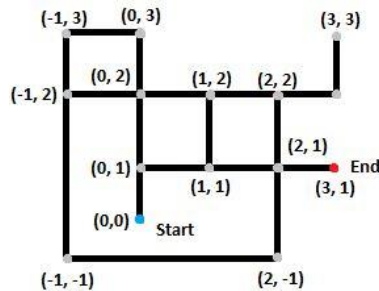
Python : 2, 3, 1, 5 \Rightarrow Room 2 \rightarrow 3 \rightarrow 1 \rightarrow 5

Octave : 3, 4, 2, 6 \Rightarrow Room 2 \rightarrow 3 \rightarrow 1 \rightarrow 5

It is evident that both implementations gave the same result after receiving the same training for this situation, as both programs have completed their learning processes.

Part 3 - Give your answer by modeling the Exercise problem (maze solving) given in the lecture note.

You are required to build a maze solving robot who should be able to reach the end point when it's placed at any starting point. The layout of the maze (paths) is given below.



Give the resulting q-tables with calculations after the following state transitions during the exploration.

- Start = (0, 0); Actions: UP, RIGHT, RIGHT, RIGHT
- Start = (0, 2), Actions: DOWN, RIGHT, UP, RIGHT, DOWN, RIGHT
- Start = (1, 2); Actions: DOWN, RIGHT, RIGHT

Now, what would be the sequence of states if the robot exploits to reach the goal starting from (0, 0)? Explain your answer.

Colab Notebook Link:

<https://colab.research.google.com/drive/1O2MDw1sZv1ONNy4uht2JI8p2c0US14zF?usp=sharing>

State Transition Table

Next State	Action	UP	DOWN	LEFT	RIGHT
State	Index	0	1	2	3
(0,0)	0	1	-1	-1	-1
(0,1)	1	2	0	-1	7
(0,2)	2	3	1	-1	8
(0,3)	3	-1	2	6	-1
(-1,-1)	4	5	-1	-1	9
(-1,2)	5	6	4	-1	-1
(-1,3)	6	-1	5	-1	3
(1,1)	7	8	-1	1	10
(1,2)	8	-1	7	2	11
(2,-1)	9	10	-1	4	-1
(2,1)	10	11	9	7	14
(2,2)	11	-1	10	8	12
(3,2)	12	13	-1	11	-1
(3,3)	13	-1	12	-1	-1
(3,1)	14	-1	-1	10	-1

```
matrix([[ 1, -1, -1, -1],
        [ 2,  0, -1,  7],
        [ 3,  1, -1,  8],
        [-1,  2,  6, -1],
        [ 5, -1, -1,  9],
        [ 6,  4, -1, -1],
        [-1,  5, -1,  3],
        [ 8, -1,  1, 10],
        [-1,  7,  2, 11],
        [10, -1,  4, -1],
        [11,  9,  7, 14],
        [-1, 10,  8, 12],
        [13, -1, 11, -1],
        [-1, 12, -1, -1],
        [-1, -1, 10, -1]])
```

-1 represents an impossible transition (null link)

Reward for reaching each state

State	Index	Reward
(0,0)	0	0
(0,1)	1	0
(0,2)	2	0
(0,3)	3	0
(-1,-1)	4	0
(-1,2)	5	0
(-1,3)	6	0
(1,1)	7	0
(1,2)	8	0
(2,-1)	9	0
(2,1)	10	0
(2,2)	11	0
(3,2)	12	0
(3,3)	13	0
(3,1)	14	100

```
matrix([[ 0],
        [ 0],
        [ 0],
        [ 0],
        [ 0],
        [ 0],
        [ 0],
        [ 0],
        [ 0],
        [ 0],
        [ 0],
        [ 0],
        [ 0],
        [ 0],
        [ 0],
        [100]])
```

Initial Q table

	Action	UP	DOWN	LEFT	RIGHT
State	Index	0	1	2	3
(0,0)	0	0	0	0	0
(0,1)	1	0	0	0	0
(0,2)	2	0	0	0	0
(0,3)	3	0	0	0	0
(-1,-1)	4	0	0	0	0
(-1,2)	5	0	0	0	0
(-1,3)	6	0	0	0	0
(1,1)	7	0	0	0	0
(1,2)	8	0	0	0	0
(2,-1)	9	0	0	0	0
(2,1)	10	0	0	0	0
(2,2)	11	0	0	0	0
(3,2)	12	0	0	0	0
(3,3)	13	0	0	0	0
(3,1)	14	0	0	0	0

```
matrix([[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.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]])
```


Exploration*Learning rate*

Gamma = 0.8

Q update rule
actions)] $Q(\text{state}, \text{action}) = R(\text{state}, \text{action}) + \text{Gamma} * \max[Q(\text{next state}, \text{all actions})]$ **Episode 1**

Start = (0, 0); Actions: UP, RIGHT, RIGHT, RIGHT (up = 0 , right = 3)

(0,0) → ((0,1) → ((1,1) → (2,1) → (3,1)

0 → 1 → 7 → 10 → 14

- $Q(0, \text{up}) = R + 0.8 * \max[Q(1,0), Q(1,1), Q(1,3)]$
State 0 to 1 reward $R = 0$
 $Q(0, 0) = 0 + 0.8 * 0 = 0$
- $Q(1, \text{right}) = R + 0.8 * \max[Q(7,0), Q(7,2), Q(7,3)]$
State 1 to 7 reward $R = 0$
 $Q(1, 3) = 0 + 0.8 * 0 = 0$
- $Q(7, \text{right}) = R + 0.8 * \max[Q(10,0), Q(10,1), Q(10,2), Q(10,3)]$
State 7 to 10 reward $R = 0$
 $Q(7, 3) = 0 + 0.8 * 0 = 0$
- $Q(10, \text{right}) = R(x,0) + 0.8 * \max[Q(14,2)]$
State 10 to 14 reward $R = 100$
 $Q(10, 3) = 100 + 0.8 * 0 = 100$

Updated Q table after episode 1

	Action	UP	DOWN	LEFT	RIGHT
State	Index	0	1	2	3
(0,0)	0	0	0	0	0
(0,1)	1	0	0	0	0
(0,2)	2	0	0	0	0
(0,3)	3	0	0	0	0
(-1,-1)	4	0	0	0	0
(-1,2)	5	0	0	0	0
(-1,3)	6	0	0	0	0
(1,1)	7	0	0	0	0
(1,2)	8	0	0	0	0
(2,-1)	9	0	0	0	0
(2,1)	10	0	0	0	100
(2,2)	11	0	0	0	0
(3,2)	12	0	0	0	0
(3,3)	13	0	0	0	0
(3,1)	14	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.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0. 100.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
```

Episode 2

Start = (0, 2); Actions: DOWN, RIGHT, UP, RIGHT, DOWN, RIGHT

(0,2) → (0,1) → (1,1) → (1,2) → (2,2) → (2,1) → (3,1)
 2 → 1 → 7 → 8 → 11 → 10 → 14

- $Q(2, \text{down}) = R + 0.8 * \max[Q(1,0), Q(1,1), Q(1,3)]$
 State 2 to 1 reward $R = 0$
 $Q(2, 1) = 0 + 0.8 * 0 = 0$
- $Q(1, \text{right}) = R + 0.8 * \max[Q(7,0), Q(7,2), Q(7,3)]$
 State 1 to 7 reward $R = 0$
 $Q(1, 3) = 0 + 0.8 * 0 = 0$
- $Q(7, \text{up}) = R + 0.8 * \max[Q(8,1), Q(8,2), Q(8,3)]$
 State 7 to 8 reward $R = 0$
 $Q(7, 0) = 0 + 0.8 * 0 = 0$
- $Q(8, \text{right}) = R + 0.8 * \max[Q(11,1), Q(11,2), Q(11,3)]$
 State 8 to 11 reward $R = 0$
 $Q(8, 3) = 0 + 0.8 * 0 = 0$
- $Q(11, \text{down}) = R + 0.8 * \max[Q(10,0), Q(10,1), Q(10,2), Q(10,3)]$
 State 11 to 10 reward $R = 0$
 $Q(11, 1) = 0 + 0.8 * 100 = 80$
- $Q(10, \text{right}) = R(x,0) + 0.8 * \max[Q(14,2)]$
 State 10 to 14 reward $R = 100$
 $Q(10, 3) = 100 + 0.8 * 0 = 100$

Updated Q table after episode 1 and 2

	Action	UP	DOWN	LEFT	RIGHT
State	Index	0	1	2	3
(0,0)	0	0	0	0	0
(0,1)	1	0	0	0	0
(0,2)	2	0	0	0	0
(0,3)	3	0	0	0	0
(-1,-1)	4	0	0	0	0
(-1,2)	5	0	0	0	0
(-1,3)	6	0	0	0	0
(1,1)	7	0	0	0	0
(1,2)	8	0	0	0	0
(2,-1)	9	0	0	0	0
(2,1)	10	0	0	0	100
(2,2)	11	0	80	0	0
(3,2)	12	0	0	0	0
(3,3)	13	0	0	0	0
(3,1)	14	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.]
  [ 0.  0.  0.  100.]
  [ 0.  80.  0.  0.]
  [ 0.  0.  0.  0.]
  [ 0.  0.  0.  0.]
  [ 0.  0.  0.  0.] ]
```


Final Trained Q-table after the 3 given episodes of training;

	Action	UP	DOWN	LEFT	RIGHT
State	Index	0	1	2	3
(0,0)	0	0	0	0	0
(0,1)	1	0	0	0	0
(0,2)	2	0	0	0	0
(0,3)	3	0	0	0	0
(-1,-1)	4	0	0	0	0
(-1,2)	5	0	0	0	0
(-1,3)	6	0	0	0	0
(1,1)	7	0	0	0	80
(1,2)	8	0	0	0	0
(2,-1)	9	0	0	0	0
(2,1)	10	0	0	0	100
(2,2)	11	0	80	0	0
(3,2)	12	0	0	0	0
(3,3)	13	0	0	0	0
(3,1)	14	0	0	0	0

Q table after 3 given episodes :

```
[[ 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.  80.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0. 100.]
 [ 0. 80.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
```

Exploitation

*The “Most Efficient Path” when **Start** = (0,0) → Index 0*

When a starting state is provided, and the most efficient path to reach the goal state is asked, the robot tries to use the knowledge gained during its learning process to give a result.

Even though it is not completely trained, it is likely to give,

0, 3, 3, 3

(Up, Right, Right, Right)

as the most efficient path to reach the end state (3,1) when starting from (0,0). This is because such a path was among the training episodes and the robot is aware of that. It sees it as a path with high rewards.

```
Indexes of ACTIONS :
UP      = 0
DOWN    = 1
LEFT    = 2
RIGHT   = 3


Goal state = (3,1) -> Index 14

Trained Episodes :

Episode 1 :   Start = (0,0) -> Index 0 ;   Actions: UP, RIGHT, RIGHT, RIGHT
Episode 2 :   Start = (0,2) -> Index 2 ;   Actions: DOWN, RIGHT, UP, RIGHT, DOWN, RIGHT
Episode 3 :   Start = (1,2) -> Index 8 ;   Actions: DOWN, RIGHT, RIGHT

Exploitation :

when the Start State is (0,0) -> Index 0 ,
    The Actions to follow the Most Efficient Path towards the Goal State :   [0, 3, 3, 3]
```



However, it may produce inefficient results with longer pathways as well. This is because this system is not trained enough to learn about the optimum path. It only went through 3 episodes of learning, which is insufficient for it to produce accurate results.

When the training is not sufficient, it is still trying to learn more about this system. For that, it has to try out other possible pathways as well. Until it is well-trained, the results provided might not be as accurate as possible.

Therefore, during the exploration period it is important to let the learning process continue until it is well-trained, before exploitation.

References

- Amunategui, M. (n.d.). *Reinforcement Learning - A Simple Python Example and A Step Closer to AI with Assisted Q-Learning*. Reinforcement Learning. Retrieved August 23, 2021, from https://www.viralml.com/video-content.html?v=nSxaG_Kjw_w
- M. (n.d.). *Getting AI smarter with Q-learning: a simple first step in Python*. The Beginner Programmer. Retrieved August 23, 2021, from <http://firsttimeprogrammer.blogspot.com/2016/09/getting-ai-smarter-with-q-learning.html>
- Lecture Note – Reinforcement Learning

~ THE END ~