****

**CZ3005: Artificial Intelligence**

***Reinforcement Learning***

Lab 2

Submitted By:

Royce Ang Jia Jie

U1840416D

TSP2

**Problem Statement**

We are to solve/ find the optimal value for the given MDP formulation:

* States: a 3D coordinate, which indicates the current position where the agent is. The initial state is (0, 0, 0) and there is only one terminal state: (3,3,3).
* Action: The action space is (forward, backward, left, right, up, down). The agent needs to select one of them to navigate in the environment.
* Reward: The agent will receive 1 reward when it arrives at the terminal states, or otherwise receive -0.1 reward.
* Transition: The intended movement happens with probability 0.6. With probability 0.1, the agent ends up in one of the states perpendicular to the intended direction. The happened movement ”forward”/”backward” will add/subtract one to the 1st element of state.

**Key Features of Q-Learning**

To solve the given MDP formulation, I have decided to use Q-Learning.

* Q-Learning is a values-based learning algorithm.
* Q-Learning is an off-policy learner. Meaning itlearns the value of the optimal policy independent of the agent’s actions.
* Q\*(s,a), where, *s,* represents state, and, *a,* representing action, is the expected value( cumulative discounted reward) of doing a in state s and then following some optimal policy.
* Q-Learning estimates Q\* out of the best Q values (as shown in given diagram below).

A picture containing chart

Description automatically generated

Figure 1 Q-Learning selecting the best Q values. [1]

* Q-Learning uses Temporal Difference (TD) to estimate the value of Q\*(s,a). *Note: TD is an agent learning from an environment through episodes with no prior knowledge of the environment.*
* Definition of Temporal Difference:



Figure 2 Temporal Difference

**Implementation**

This will be how our Q-learning Algorithm process will look like:

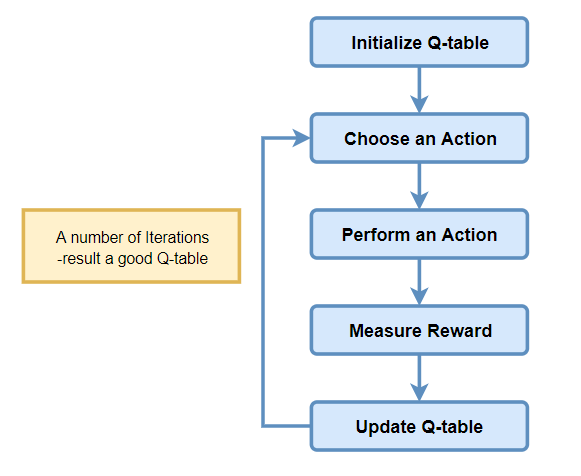


Figure 3 Q-Learning Process

Limitations Involved:

* + - Gamma (Discount Factor) = 0.99
    - Alpha (Learning-Rate) = 0.5
    - Epsilon (Exploration rate) = 0.01

*Step 1: Initialize the Q-Table:*

Given that, we have a 3x3x3 state, each with a total of 6 different actions (forward, backward, left, right, up, down).

1.1: *def createAllPossibleStates(self):* (Agent Class)

Graphical user interface, text, application

Description automatically generated

1.2: Initialize Q-Table in a Dictionary format to produce a Q(s,a).

**Graphical user interface, text

Description automatically generated**

*Step 2: Choosing an action*

Epsilon greedy strategy concept comes into play here. The agent takes greedy action for probability < ( 1 – ε), else a random action will be taken from the given action space. *Note: epsilon = 0.01*.

2.1*: def random\_greedy\_action(self, state\_key, eps=0.01)*

**Graphical user interface, text, application

Description automatically generated**

*Step 3: Perform an action*

Here, we would make use of the given environment method, def step(action), to return us the *reward*, *terminate* and *next\_state* values.

Text

Description automatically generated

*Step 4: Measuring the reward*

Every action taken would produce an observed outcome and reward. As such, the returned result would then be added to our overall *episode\_reward* bank, using a simple cumulative method.

A picture containing text

Description automatically generated

*Step 5: Evaluate*

We need to update the function Q(s,a), via the following equation to give us the **New Q(s,a).**

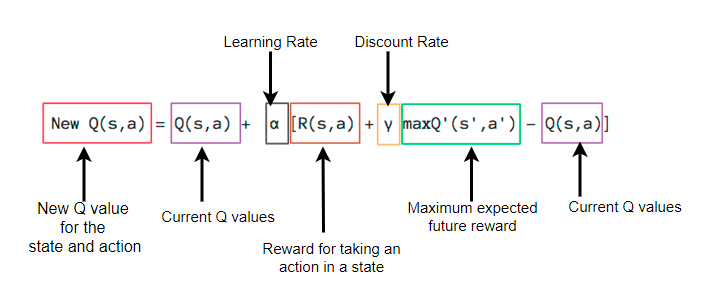


Figure 4 Equation Explanation

5.1: *def max\_dict(self, d*): this method would be used to return the maximum expected future reward.

**Text

Description automatically generated**

5.2: *def train(self, state, action, next\_state, reward):* this is a direct implementation of the given formula stated above to obtain the new Q value.

Text

Description automatically generated

**Additional Implementation**

*Additional 1: def print\_values(self, v):*  this would give a proper UI for printing Q-Values.

**Graphical user interface, text, application

Description automatically generated**

*Additional 2: def print\_policy(self, P): this would give a proper UI for printing the best policies.*

*Text

Description automatically generated*

Additional 3: print method to denote the proportional of time spent on updating each part of Q:

A picture containing text

Description automatically generated

**Outcome/ Overall Results**

In order to properly evaluate the final results, I have implemented 4 different print statements

1. *Q-Table Full*
2. *Q-Table-Max-Value*
3. *Policy Table*
4. *Proportion of time spent updating each part of Q*);
5. Plot episode rewards vs episodes.

These are evaluated against –max\_episode 500 && --max\_step 500.

Note: Q-Table-Max-Value and Policy Tables are states ordered in rows from (0,0,0) 🡪 (0,0,3), etc. with each value representing the max of values for each of the states respectively.

**Case 1 (Best-Case)**

*Total Steps: 9*

******

**A picture containing text

Description automatically generated**

**A picture containing text

Description automatically generated**

**Chart

Description automatically generated**

Figure 5 Step\_9 episode\_rewards vs episodes

**Case 2:**

*Total Steps: 10 version1*



Text

Description automatically generated

A picture containing text

Description automatically generated

Chart

Description automatically generated

Figure 6 Step\_10 episode\_rewards vs episodes

**Case 3:**

*Total Steps 10 version2*

**

*A picture containing window, sitting, curtain, person

Description automatically generated*

*A picture containing text

Description automatically generated*

*Chart, line chart

Description automatically generated*

Figure 7 Step\_10\_version2 episode\_rewards vs episodes

**Case 4:**

*Total Steps 11*



Text

Description automatically generated

A close up of text on a black surface

Description automatically generated

Text

Description automatically generated

A picture containing text

Description automatically generated

Chart

Description automatically generated

Figure 8 Step\_11 episode\_rewards vs episodes

**Final Observations**

Do the values eventually converge?

|  |  |  |  |
| --- | --- | --- | --- |
| **Step 9** | **Step 10 version 1** | **Step 10 version 2** | **Step 11** |
|  |  |  | A close up of text on a black surface  Description automatically generated |
|  |  |  | Text  Description automatically generated |

Theoretically, with increasing iterations, the Q-Table values should eventually converge. This is especially so for majority of the values from Q-Tables with similar Step values. However, there are edge cases for some states with large diverging values (also shown in the policy table with ‘*backward’ action*), and I would attribute this to the random chance of action occurring in the environment file, which may, by random chance, disrupt the sub-optimal value needed to compute for the subsequent most optimal value.

****

**Text

Description automatically generated**

Figure 9 Random Occurring Chance

**Further Possible Optimization**

There are varying methods to improve the performance of Q-learning, such as using simultaneous Q-values updating [1]. It’s possible to increase the number of episodes and further decrease the epsilon (exploration rate) from 0.01 to 0.0001, and toggling *stochastic to false*. These would produce a more consistent and converging values.

Text

Description automatically generated

Figure 10 Stochastic = False

|  |  |
| --- | --- |
| **Iteration 1 @ eps 0.0001** | **Iteration 2 @ eps 0.0001** |
|  |  |
|  |  |
|  |  |

# Reference

|  |  |
| --- | --- |
| [1] | M. Pouyan, "Improving the performance of Q-learning using simultaneous Q-values updating," *International Congress on Technology, Communication and Knowledge (ICTCK),* vol. 10, pp. 1-6, 2014. |
| [2] | C. J. Watkins, "Technial Note Q-Learning," *Kluwer Academic,* vol. 8, pp. 279-292, 1992. |