

# CS420 ARTIFICIAL INTELLIGENT Q-LEARNING

## What is Q-LEARNING?

Q-LEARNING is an un-policy reinforcement learning algorithm that seeks to find the best action to take given the current state. Q-LEARNING is considered as off-policy because its function learns from actions that are not in the current policy, for instances: taking random actions or direct experimentation, and therefore a policy is redundant. By using rewards and punishment, it enables the agent have the ability to learn and seek a policy that maximize the total rewards, in order to solve problem in an optimal way.

## HOW DOES Q-LEARNING WORK?

The Q-Learning algorithm goes as follow:

1. Set the gamma parameter  $\gamma$ , and environment rewards in matrix R.
2. Initialize matrix Q to zero.
3. For each episode:
  - Select random initial state.
  - While not reach goal state:
    - Select one among all possible actions for the current state.
    - Using this possible action, consider to go to next state.
    - Get maximum Q-value of this next state based on all possible actions.
    - Compute Q-value:
$$Q(\text{state}, \text{action}) = R(\text{state}, \text{action}) + \gamma \cdot \text{Max}[Q(\text{next state}, \text{all actions})]$$
    - Set next state as current state.
  - End while loop
- End for loop
4. After many episode, return the values of Q matrix which is convergence.
5. Normalize Q matrix by dividing it by the highest value in the matrix.
6. Return sequences of next steps which has the highest value for each current state.

The gamma  $\gamma$  parameter has a range from 0 to 1. If gamma is closer to zero, the agent will tend to consider only immediate rewards. If gamma is closer to one, the agent will consider future rewards with greater weight, willing to delay the reward.

For example:

Best value for  $\gamma$  is 0.8: the range for values in normalized Q matrix give a good indication about which state is best.

$\gamma$  is 0.5: the values are near to each other and small, this make learning process for agent is slow.

$\gamma$  is 1: learning process not accurate because values for most states close to each other. This make selection of next state more complex for the agent.

Each episode is equivalent to one training session. In each training session, the agent explores the environment (represented by matrix R), receives the reward (if any) until it reaches the goal state. The purpose of the training is to enhance the “brain” of our agent, presented by matrix Q. More training results in a more optimized matrix Q. In this case, if the matrix Q has been enhanced, instead exploring around, and going back and forth the same rooms, the agent will find the fastest route to the goal state.

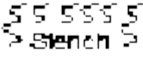



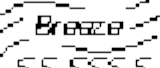
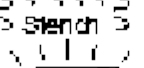
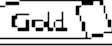

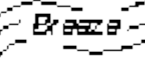
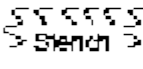


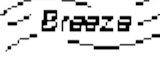

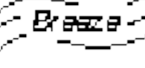
## WHY Q-LEARNING SOLVE WUMPUS PROBLEM?

Q-LEARNING can be used to solve Wumpus problem because:

- Wumpus has agent player, environment and find gold state.
- Wumpus has determined number of state and actions.
- Each action from one state to another has reward or punishment (if agent come to pit or wumpus will get score punishment, by contrast, if agent come to gold state, it will get reward).
- The basic idea is to treat problem as an input/output mapping (states  $\rightarrow$  actions), and to learn which actions produce the greatest rewards  $\Rightarrow$  Q-LEARNING has suitable to learn the shape of a function, the learning approach will need to try and fail many times before playing well.

### EXAMPLE OF SOLVING WUMPUS PROBLEM BY Q-LEARNING:

We have 4x4 map:

4				
3		  		
2				
1				
	1	2	3	4

From this map, we will have  $4*4 = 16$  states.

Each state has its position(i, j), which i and j has range from 1 to 4.

And we also have 4 actions: go down(D), go up(U), turn right(R), turn left(L).

Rewards and punishment:

If agent is at pit or wumpus position, it gets -100.

If agent is at gold position, it gets 100.

If agent goes outside map, it gets -100

Else, it gets 1.

Step 1: Set the gamma parameter  $\gamma$  is 0.8 and rewards matrix (R):

State	Position	Up	Down	Right	Left
1	1, 1	1	-100	1	-100
2	1, 2	1	-100	-100	1
3	1, 3	1	-100	1	1
4	1, 4	1	-100	-100	-100
5	2, 1	-100	1	1	-100
6	2, 2	100	1	1	1
7	2, 3	-100	-100	1	1
8	2, 4	1	1	-100	1
9	3, 1	1	1	100	-100
10	3, 2	1	1	-100	-100
11	3, 3	1	1	1	100
12	3, 4	-100	1	-100	-100
13	4, 1	-100	-100	1	-100
14	4, 2	-100	100	1	1
15	4, 3	-100	1	-100	1
16	4, 4	-100	1	-100	1

For example:

At state 1 (1, 1), if we go up (2, 1), we get 1 or else go down (outside map), we get -100.

At state 10 (3, 2) if we go right (3, 3), there is a pit => we get -100.

Or else we go left (3, 1), there is a wumpus => we get -100.

At state 6 (2, 2), if we go up (3, 2), there is gold => we get 100.

Step 2: Initialize Q matrix:

State	Position	Up	Down	Right	Left
1	1, 1	0	0	0	0
2	1, 2	0	0	0	0
3	1, 3	0	0	0	0
4	1, 4	0	0	0	0
5	2, 1	0	0	0	0
6	2, 2	0	0	0	0
7	2, 3	0	0	0	0
8	2, 4	0	0	0	0
9	3, 1	0	0	0	0
10	3, 2	0	0	0	0
11	3, 3	0	0	0	0
12	3, 4	0	0	0	0
13	4, 1	0	0	0	0
14	4, 2	0	0	0	0
15	4, 3	0	0	0	0
16	4, 4	0	0	0	0

Step 3: For loop episode = 1 until 300:

    Select random initial state 1 (1, 1)

    Since not at goal state 10 (3, 2), loop until current state is goal state.

        At current state, we can go up or go right (which has reward 1), therefore we pick random action “Go up”, which go to state 5 (2, 1).

        At state 5 (2, 1), we consider its next steps, which has highest Q-values. From the Q-matrix we can see that, at state 5, all Q-values is 0  $\Rightarrow \text{Max}(Q(\text{next state, all actions})) = 0$ .

$$\begin{aligned}\text{Compute: } Q(1, \text{“Up”}) &= R(\text{state, action}) + \gamma * \text{Max}(Q(\text{next state, all actions})) \\ &= R((1, \text{“Up”}) + \gamma * \text{Max}(Q(5, \text{all actions}))) \\ &= 1 + 0.8 * 0 = 1\end{aligned}$$

    Update Q-matrix:

State	Position	Up	Down	Right	Left
1	1, 1	1	0	0	0
2	1, 2	0	0	0	0
3	1, 3	0	0	0	0
4	1, 4	0	0	0	0
5	2, 1	0	0	0	0
6	2, 2	0	0	0	0
7	2, 3	0	0	0	0
8	2, 4	0	0	0	0
9	3, 1	0	0	0	0
10	3, 2	0	0	0	0
11	3, 3	0	0	0	0
12	3, 4	0	0	0	0
13	4, 1	0	0	0	0
14	4, 2	0	0	0	0
15	4, 3	0	0	0	0
16	4, 4	0	0	0	0

    Set state 5 (2, 1) as current state

    End while

In next loop: because state 5(2, 1) is not goal state, we consider:

    At current state, we can go down or go right (which has reward 1), therefore we pick random action “Go right”, which go to state 6 (2, 2).

    At state 6 (2, 2), we consider its next steps, which has highest Q-values. From the Q-matrix, at state 6, all Q-values is 0  $\Rightarrow \text{Max}(Q(\text{next state, all actions})) = 0$ .

$$\begin{aligned}\text{Compute: } Q(5, \text{“Down”}) &= R(\text{state, action}) + \gamma * \text{Max}(Q(\text{next state, all actions})) \\ &= R((5, \text{“Right”}) + \gamma * \text{Max}(Q(5, \text{all actions}))) \\ &= 1 + 0.8 * 0 = 1\end{aligned}$$

Update Q-matrix:

State	Position	Up	Down	Right	Left
1	1, 1	1	0	0	0
2	1, 2	0	0	0	0
3	1, 3	0	0	0	0
4	1, 4	0	0	0	0
5	2, 1	0	0	1	0
6	2, 2	0	0	0	0
7	2, 3	0	0	0	0
8	2, 4	0	0	0	0
9	3, 1	0	0	0	0
10	3, 2	0	0	0	0
11	3, 3	0	0	0	0
12	3, 4	0	0	0	0
13	4, 1	0	0	0	0
14	4, 2	0	0	0	0
15	4, 3	0	0	0	0
16	4, 4	0	0	0	0

Set state 6 (2, 2) as current state

End while

In next loop: because state 6(2, 2) is not goal state, we consider:

At current state, we can go up/down/right/left, therefore we pick random action “Go up”, which go to state 10 (3, 2).

At state 10 (3, 2), we consider its next steps, which has highest Q-values. From the Q-matrix, at state 10, all Q-values is 0 =>  $\text{Max}(Q(\text{next state, all actions})) = 0$ .

$$\begin{aligned}
 \text{Compute: } Q(5, \text{“Down”}) &= R(\text{state, action}) + \gamma * \text{Max}(Q(\text{next state, all actions})) \\
 &= R((5, \text{“Right”})) + \gamma * \text{Max}(Q(5, \text{all actions})) \\
 &= 1 + 0.8 * 0 = 1
 \end{aligned}$$

Update Q-matrix:

State	Position	Up	Down	Right	Left
1	1, 1	1	0	0	0
2	1, 2	0	0	0	0
3	1, 3	0	0	0	0
4	1, 4	0	0	0	0
5	2, 1	0	0	1	0
6	2, 2	0	0	1	0
7	2, 3	0	0	0	0
8	2, 4	0	0	0	0
9	3, 1	0	0	0	0
10	3, 2	0	0	0	0
11	3, 3	0	0	0	0
12	3, 4	0	0	0	0
13	4, 1	0	0	0	0
14	4, 2	0	0	0	0
15	4, 3	0	0	0	0
16	4, 4	0	0	0	0

Set state 10 (3, 2) as current state

End while

In next loop: because state 10(3, 2) is goal state, stop while loop and new episode.

Step 4: After 300 episodes, we get a convergence matrix:

State	Position	Up	Down	Right	Left
1	1, 1	270	0	270	0
2	1, 2	0	0	0	0
3	1, 3	0	0	0	0
4	1, 4	0	0	0	0
5	2, 1	0	90	240	0
6	2, 2	210	60	60	30
7	2, 3	0	0	0	0
8	2, 4	0	100	0	0
9	3, 1	0	0	0	0
10	3, 2	0	0	-100	0
11	3, 3	210	0	0	0
12	3, 4	0	0	0	0
13	4, 1	0	180	-100	0
14	4, 2	-200	0	0	120
15	4, 3	0	0	0	0
16	4, 4	0	0	0	0

Step 5: Normalize Q-matrix with max-value is 270

State	Position	Up	Down	Right	Left
1	1, 1	1	0	1	0
2	1, 2	0	0	0	0
3	1, 3	0	0	0	0
4	1, 4	0	0	0	0
5	2, 1	0	0.3	0.8	0
6	2, 2	0.7	0.2	0.2	0.1
7	2, 3	0	0	0	0
8	2, 4	0	0.37	0	0
9	3, 1	0	0	0	0
10	3, 2	0	0	-0.37	0
11	3, 3	0.7	0	0	0
12	3, 4	0	0	0	0
13	4, 1	0	0.6	-0.37	0
14	4, 2	-0.74	0	0	0.4
15	4, 3	0	0	0	0
16	4, 4	0	0	0	0

Step 6: Path we can go:

Initial state (1,1) we pick go Up to go to (2, 1)

At (2, 1), (2, 2) will be next step because "Right" has highest Q-value = 0.8

At (2, 2), (3, 2) will be next step because "Up" has highest Q-value = 0.7

Because (3, 2) is goal state => stop.

Path we have: state 1 -> state 5 -> state 6 -> state 10.

## **COMPARING Q-LEARNING AND LOGICAL SEARCH:**

### **In logical search:**

- We cannot pick random next step, because we can know which step is safe or unsafe using logical formulation.
- Process may be faster than q-learning because we don't need to calculate the max q-value for next step, and don't have to loop episode.
- Not find optimal path or highest outcome, because we just run once, don't try to learn.
- However, with a determined number of steps we allow to go, logical search is more suitable because we can turn back initial state after determined steps rather than q-learning.

### **In q-learning:**

- We can random go to possible state
- Process is slower than logical search because we have many episode and each episode, at every current state, we need to find max q-value.
- Sometimes, it stuck at infinite loop when there are more than 1 goal state.
- However, we can find optimal path.