

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

**CZ3005 Artificial Intelligence**

**Assignment 2: Reinforcement Learning**

Name: Loh Yi Xuan Renice

Matriculation No.: U1822247D

Group: TSP6

Introduction

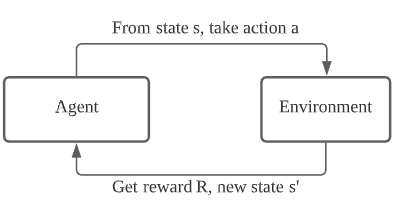


Figure 1: Reinforce Learning Framework

Reinforcement Learning(RL) is the science of making optimal decisions using experiences. In summary, the process of Reinforcement Learning involves these steps:

1. Observation of the environment
2. Deciding how to act using a certain strategy
3. Taking action accordingly
4. Receiving a reward or penalty
5. Learning from the experiences and refining our strategy
6. Iterate until an optimal strategy is found

The objective of RL is to maximise the reward of an agent by taking a series of actions in response to a dynamic environment.

There are two types of RL algorithms - ***model-based*** and ***model-free*.**

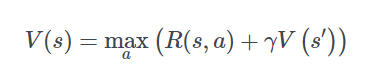
A model-based algorithm is an algorithm that estimates the optimal policy using or estimating the dynamics (transition and reward functions) of the environment while a model-free algorithm does not.

Q-Learning Algorithm

Overview

Q-learning is a model-free algorithm that solves for the optimal policy in an Markov Decision Process . It uses a value-based learning algorithm which updates the value function based on Bellman Equation. It is therefore considered model-free as Q-learning learns from actions that are outside the current policy, taking random actions which is known as “exploration”. Q-learning seeks to learn a policy that maximises the total reward, optimal in the sense that the expected value of the total reward over all successive steps is the maximum achievement. In other words, the goal of Q-learning is to find the optimal policy by learning the optimal Q-values for each state-action pair.

Bellman Equation



**s**: a particular state

**a**: action

**s’**: the next state agent goes from s

**γ**: discount factor

**R(s,a)**: a reward function which takes state s and action a and outputs a reward value

**V(s)**: value of being in a particular state

Algorithm Process

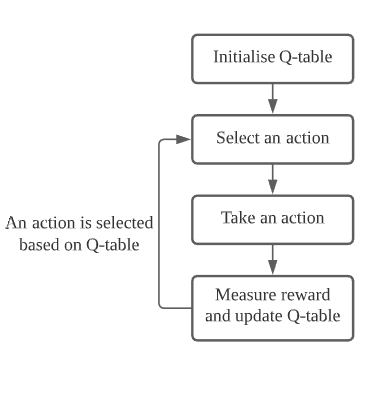


Figure 2: Overview process of Q-learning

**Initialise Q-table**

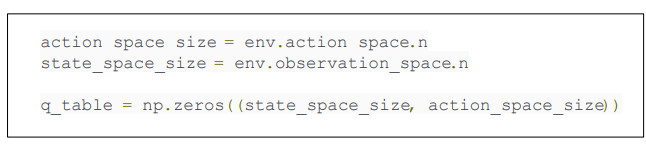


Figure 3: Initialise Q-table in Python

When we perform q-learning, a q-table[state, action] has to be built. There are n columns and m rows, where n = number of actions and m = number of states. We firstly initialise our values to zero. After each episode, the q-table will be updated. This q-table is used for our agent to select the best action based on the q-value

In our project, the action space is = [left, right, up, down, forward, back] and the state space is [0,0,0] to [3,3,3].

**Selection an action**

****

**Figure 4: Select an action in python**

For each time-step within an episode, exploration\_rate\_threshold is set to a random number between 0 and 1. This will be used to determine whether our agent will explore or exploit the environment. If the threshold is greater than the exploration\_rate, our agent will exploit the environment and choose the action that has the highest Q-value in the Q-table for the current state, otherwise, it will explore.

In this case, the exploration rate i.e. epsilon is set to ε = 0.01.

**Take an action**

****

**Figure 5: Take an action in python**

After our action is chosen based on the bellman equation, we then take that action by calling step() on our env object and passing our action to it. The function step() returns a tuple containing the reward for the action taken, whether or not the action ended the episode and the next state.

**Update the Q value**

****

**Figure 6: Updating Q-table in python**

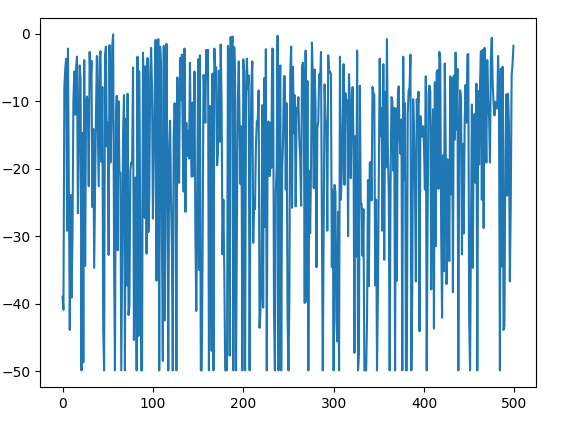
The Q(state,action) returns the expected future reward of that action at that state.

Once we have taken an action and observed an outcome and reward. We need to update the function Q(s,a). In this project, the agent will receive reward = 1 when it arrives at the terminal state [3,3,3]. Otherwise, reward = -0.1

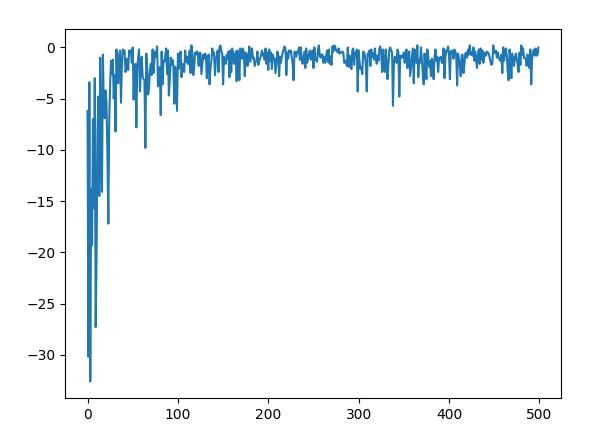
This process is repeated again and again until the maximum number of episodes. After many iterations, the value function Q is maximised. The new Q-value for this state-action pair is a weighted sum of our old value and the “learned value.”

Learning Progress

Graph Plot



**Figure 7: Episode rewards vs Episode for Random Agent**



**Figure 8: Episode rewards vs Episode for Q-learning Agent**

Interpretation

The graphs above plots episode rewards against episode. That means, the total reward received by the agent for the entire episode for each episode is plotted. Our agent played 500 episodes. At each time step within an episode, the agent receives a reward of 1 if it reaches the goal, otherwise, it receives a reward of -0.1. If the agent did indeed reach the goal, then the episode finishes at that time-step.

From the graph we can see that the episode rewards for the q-learning agent has significantly improved compared to the random agent. Furthermore, the episode rewards for the q-learning is more consistent and consistently high while the episode rewards for the random agent is fluctuating significantly. This shows that the Q-learning agent has learnt.

Final Value Q-table

The Q-learning agent updates the q-table with each episode and decides what action to take at each state based on the Q-table. Below is the final value q-table which is what the q-learning agent has learnt after 500 episodes:

| **State|Action** | **Left** | **Right** | **Forward** | **Backward** | **Up** | **Down** |
| --- | --- | --- | --- | --- | --- | --- |
| **[0, 0, 0]** | -0.1 | -0.1 | -0.1 | -0.1 | -0.1 | -0.1 |
| **[0, 0, 1]** | -0.0999609 | -0.0999338 | -0.0878706 | -0.0999968 | -0.0999197 | -0.0993207 |
| **[0, 0, 2]** | -0.0999991 | -0.0272083 | -0.0183941 | -0.0636044 | -0.0354980 | -0.0999999 |
| **[0, 0, 3]** | -0.0999970 | -0.0940663 | -0.0940663 | -0.0999785 | -0.0956243 | -0.0996109 |
| **[0, 1, 0]** | -0.0998882 | -0.0997077 | -0.0999694 | -0.0986394 | -0.0999999 | -0.1 |
| **[0, 1, 1]** | -0.0997480 | -0.0761362 | -0.0582504 | -0.0997650 | -0.0681844 | -0.0982008 |
| **[0, 1, 2]** | -0.0999992 | -0.0106019 | 0.06155108 | -0.0557342 | -0.0600829 | -0.0999265 |
| **[0, 1, 3]** | -0.0928902 | -0.0309486 | -0.0084619 | -0.0999755 | -0.0790592 | -0.0999755 |
| **[0, 2, 0]** | -0.0978149 | -0.0930960 | -0.0438017 | -0.0999998 | -0.0498612 | -0.0997829 |
| **[0, 2, 1]** | -0.0999999 | -0.0810745 | 0.01946560 | -0.0999999 | -0.0731363 | -0.0242006 |
| **[0, 2, 2]** | -0.0795841 | 0.1853828 | -0.0032998 | -0.0391226 | -0.0992187 | -0.096875 |
| **[0, 2, 3]** | -0.0998046 | -0.09375 | -0.1648499 | -0.0999023 | -0.09375 | -0.0984375 |
| **[0, 3, 0]** | -0.0970918 | -0.0364189 | -0.0695179 | -0.0999999 | -0.0577818 | -0.0999999 |
| **[0, 3, 1]** | -0.0999511 | -0.0508745 | -0.0024165 | -0.0999996 | -0.0999755 | -0.0999755 |
| **[0, 3, 2]** | -0.0992187 | -0.0992187 | 0.31010533 | -0.0523689 | -0.0875000 | -0.0999755 |
| **[0, 3, 3]** | -0.0875000 | -0.096875 | -0.09375 | -0.0750000 | -0.096875 | 0.0130004 |
| **[1, 0, 0]** | -0.0999999 | -0.0999999 | -0.0998614 | -0.1 | -0.0999999 | -0.0979382 |
| **[1, 0, 1]** | -0.0972891 | -0.0564308 | -0.0563054 | -0.0741623 | -0.0656239 | -0.0855212 |
| **[1, 0, 2]** | -0.0911822 | -0.0419558 | -0.0098774 | -0.0888167 | -0.0237084 | -0.0929310 |
| **[1, 0, 3]** | -0.0999511 | 0.3066394 | -0.0421633 | -0.0984375 | -0.0999969 | -0.0.09999 |
| **[1, 1, 0]** | -0.0989926 | -0.0748398 | -0.0527360 | -0.0999999 | -0.0517090 | -0.0999960 |
| **[1, 1, 1]** | -0.0022102 | -0.0079784 | -0.0796536 | -0.0993928 | 0.0848886 | -0.0587130 |
| **[1, 1, 2]** | -0.0504239 | -0.09375 | -0.0128308 | -0.0984375 | 0.1290975 | -0.0877029 |
| **[1, 1, 3]** | -0.09375 | 0.1085035 | 0.44259155 | -0.096875 | -0.0784786 | -0.0984375 |
| **[1, 2, 0]** | -0.0999999 | -0.0788590 | -0.0478695 | -0.0959223 | 0.1299948 | -0.0999755 |
| **[1, 2, 1]** | -0.0651244 | -0.0037787 | -0.0180812 | -0.0999999 | 0.1557527 | -0.0010844 |
| **[1, 2, 2]** | -0.0721165 | -0.0522391 | 0.17909041 | -0.0996093 | -0.0996093 | -0.0984375 |
| **[1, 2, 3]** | -0.0875000 | 0.0756312 | 0.37562845 | -0.0984375 | 0.1142592 | -0.0984375 |
| **[1, 3, 0]** | -0.0674438 | -0.0999023 | -0.0999023 | -0.0999755 | 0.0899584 | -0.0999877 |
| **[1, 3, 1]** | -0.0996093 | -0.0115550 | -0.0981545 | -0.0570228 | 0.2819927 | 0.0065998 |
| **[1, 3, 2]** | -0.09375 | 0.0507114 | 0.62378652 | -0.0750000 | 0.0550991 | -0.09375 |
| **[1, 3, 3]** | 0.01941629 | -0.0750000 | -0.05 | -0.0750000 | 0.096875 | -0.0875000 |
| **[2, 0, 0]** | -0.0944519 | -0.0951654 | -0.0849835 | -0.0909743 | -0.0277439 | -0.0985228 |
| **[2, 0, 1]** | -0.0375779 | -0.0124250 | -0.0421374 | -0.0994669 | 0.0970488 | -0.0998751 |
| **[2, 0, 2]** | -0.0999023 | 0.1285231 | -0.0201298 | -0.0130886 | 0.0384554 | -0.0957483 |
| **[2, 0, 3]** | -0.0999877 | -0.0975944 | 0.34479873 | -0.0999877 | -0.0999023 | -0.0999023 |
| **[2, 1, 0]** | -0.0999998 | -0.0348215 | -0.0396481 | -0.0992361 | -0.0827668 | -0.0957623 |
| **[2, 1, 1]** | -0.0031549 | -0.0845237 | 0.24060362 | -0.0999998 | -0.0048469 | -0.0980990 |
| **[2, 1, 2]** | -0.0858909 | 0.0675260 | -0.0882328 | -0.099975 | 0.2212651 | -0.0960097 |
| **[2, 1, 3]** | -0.0194637 | 0.6300217 | -0.0208573 | -0.096875 | -0.0875000 | -0.0992187 |
| **[2, 2, 0]** | -0.0897444 | 0.0775810 | -0.0418226 | -0.0982044 | -0.0902391 | -0.0993338 |
| **[2, 2, 1]** | -0.0999969 | -0.0371968 | -0.0403594 | -0.0171286 | 0.1508732 | -0.0369790 |
| **[2, 2, 2]** | -0.0616902 | -0.0875000 | 0.50078672 | -0.0984375 | 0.2614698 | 0.0504620 |
| **[2, 2, 3]** | -0.05 | 0.8296758 | -0.0767008 | 0.30421545 | -0.096875 | 0.2091794 |
| **[2, 3, 0]** | -0.0999511 | -0.0875000 | -0.0154523 | -0.0998046 | 0.2923318 | -0.0998046 |
| **[2, 3, 1]** | 0.09575876 | -0.0750000 | -0.0931007 | -0.0984375 | 0.3848001 | -0.0173589 |
| **[2, 3, 2]** | -0.096875 | -0.0750000 | 0.80366465 | -0.0875000 | 0.3042046 | 0.22030112 |
| **[2, 3, 3]** | -0.0750000 | 0.0531269 | 1 | -0.0875000 | -0.05 | 0.3421744 |
| **[3, 0, 0]** | -0.0492547 | -0.0491148 | -0.0999999 | -0.0647527 | -0.0633954 | -0.0999877 |
| **[3, 0, 1]** | -0.09999961853027345 | -0.05258990539485607 | -0.09998894305598599 | -0.0999999761581421 | 0.23133748924390765 | -0.09999847412109375 |
| **[3, 0, 2]** | -0.09999389648437501 | -0.09655382660127947 | -0.0999755859375 | -0.09981247112131594 | 0.4560072754201042 | -0.0998046875 |
| **[3, 0, 3]** | -0.0999023 | 0.5952619 | -0.0992187 | -0.0999938 | -0.0440639 | -0.0796924 |
| **[3, 1, 0]** | -0.0999992 | -0.0842177 | -0.0999969 | -0.0999755 | 0.0648898 | -0.0999996 |
| **[3, 1, 1]** | -0.0999969 | -0.0998379 | -0.0995266 | -0.0990742 | 0.1859993 | -0.0999877 |
| **[3, 1, 2]** | -0.09375 | 0.4880216 | -0.0998046 | 0.14177240 | -0.0352466 | 0.2431023 |
| **[3, 1, 3]** | -0.096875 | 0.5701884 | -0.0750000 | -0.0677359 | -0.0898056 | -0.0875000 |
| **[3, 2, 0]** | -0.0128660 | -0.0693106 | -0.0999969 | -0.0752325 | -0.0083020 | -0.0999755 |
| **[3, 2, 1]** | -0.0300044 | -0.0183981 | -0.0720940 | 0.39100084 | -0.0994691 | -0.0998046 |
| **[3, 2, 2]** | -0.0988180 | 0.8367618 | 0.36742555 | 0.22226460 | -0.0753549 | -0.0619345 |
| **[3, 2, 3]** | 0 | 1 | 0.42214365 | -0.05 | 0.42 | -0.0372417 |
| **[3, 3, 0]** | -0.0753494 | -0.0999984 | -0.0987115 | -0.0148243 | -0.0252250 | -0.0999938 |
| **[3, 3, 1]** | -0.0984375 | -0.0378200 | -0.0999511 | 0.37471336 | -0.0843728 | -0.0993529 |
| **[3, 3, 2]** | 0.08579910 | -0.0875000 | -0.0414132 | -0.0622658 | 1 | -0.0875000 |
| **[3, 3, 3]** | 0 | 0 | 0 | 0 | 0 | 0 |

Code Implementation

\*Attached source code in python file\*