Neuroscience of Learning, Memory, Cognition

Instructor: Prof. Hamid Aghajan

Sharif University of Technology

Q-Learning

Author:

Parsa Palizian parsapalizian81@gmail.com

January 2024

Preface

Notes on the project:

- This project is a bonus to the final grade. Hence, it must be done individually.
- You must read the instructions carefully and implement the -greedy algorithm. The introduction to the algorithm and basic concepts are explained later.
- You may run the code multiple times and experiment with various parameters and training episodes. The goal is to achieve the best result provided in later sections. Your work will be graded based on the efficiency of your final answer along with the implementation of the desired concepts.
- The algorithm must be implemented from scratch. Utilization of third party libraries for the matter is prohibited. • You must submit your code along with a full report of the experiments carried out and the final plots and figures corresponding to the optimum result.
- The final figures are twofold: 1- a heat-map of the grid with the best route highlighted 2- the dynamics of the cumulative reward across all training episodes
- Any brainstorming and discussion with others is allowed and encouraged, but the final work must belong to you and your efforts specificall

Contents

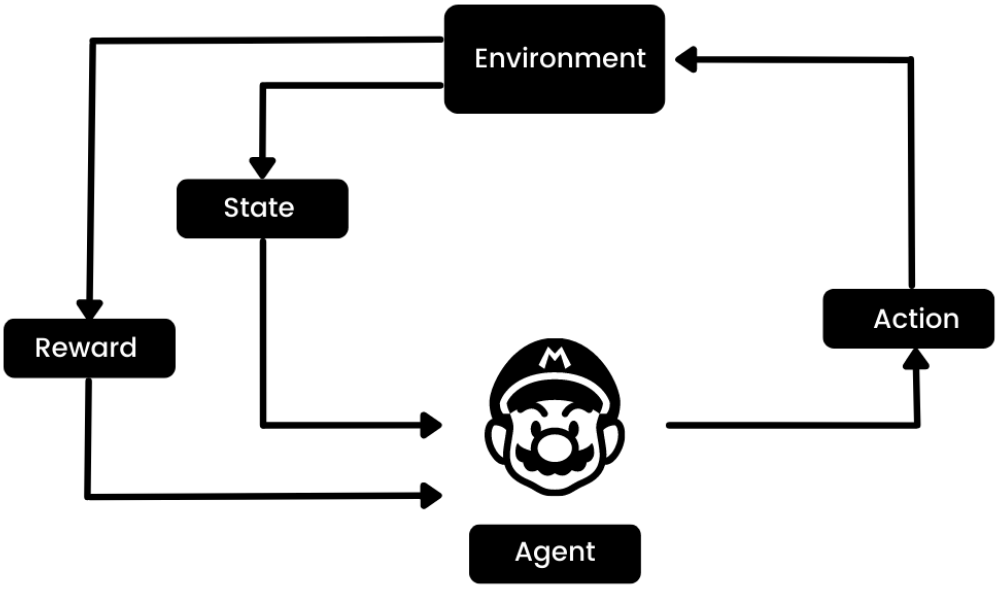
1 Introduction

1.1 RL-Learning

Reinforcement learning (RL) is the part of the machine learning ecosystem where the agent learns by interacting with the environment to obtain the optimal strategy for achieving the goals. It is quite different from supervised machine learning algorithms, where we need to ingest and process that data. Reinforcement learning does not require data. Instead, it learns from the environment and reward system to make better decisions.

For example, in the Mario video game, if a character takes a random action (e.g. moving left), based on that action, it may receive a reward. After taking the action, the agent (Mario) is in a new state, and the process repeats until the game character reaches the end of the stage or dies.

This episode will repeat multiple times until Mario learns to navigate the environment by maximizing the rewards.



We can break down reinforcement learning into five simple steps:

1. The agent is at state zero in an environment.
2. It will take an action based on a specific strategy.
3. It will receive a reward or punishment based on that action.
4. By learning from previous moves and optimizing the strategy.
5. The process will repeat until an optimal strategy is found.

2 Main

2.1 What is Q-Learning?

Q-learning is a model-free, value-based, off-policy algorithm that will find the best series of actions based on the agent's current state. The "Q" stands for quality. Quality represents how valuable the action is in maximizing future rewards.

The model-based algorithms use transition and reward functions to estimate the optimal policy and create the model. In contrast, model-free algorithms learn the consequences of their actions through the experience without transition and reward function.

The value-based method trains the value function to learn which state is more valuable and take action. On the other hand, policy-based methods train the policy directly to learn which action to take in a given state.

In the off-policy, the algorithm evaluates and updates a policy that differs from the policy used to take an action. Conversely, the on-policy algorithm evaluates and improves the same policy used to take an action.

2.2 Key Terminologies in Q-learning

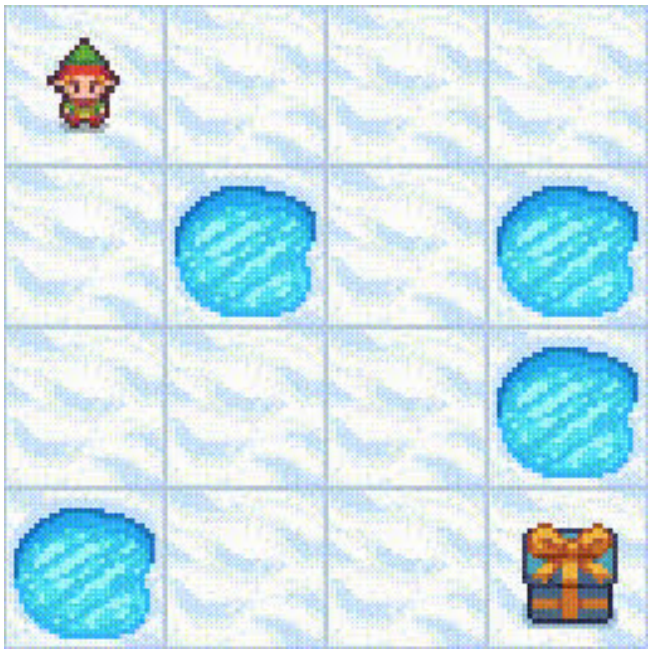
Before we jump into how Q-learning works, we need to learn a few useful terminologies to understand Q-learning's fundamentals.

- States(s): the current position of the agent in the environment.
- Action(a): a step taken by the agent in a particular state.
- Rewards: for every action, the agent receives a reward and penalty.
- Episodes: the end of the stage, where agents can't take new action. It happens when the agent has achieved the goal or failed.
- Q(St+1, a): expected optimal Q-value of doing the action in a particular state.
- Q(St, At): it is the current estimation of Q(St+1, a).
- Q-Table: the agent maintains the Q-table of sets of states and actions.
- Temporal Differences(TD): used to estimate the expected value of Q(St+1, a) by using the current state and action and previous state and action.

3 Approach

3.1 How Does Q-Learning Work?

We will learn in detail how Q-learning works by using the example of a frozen lake. In this environment, the agent must cross the frozen lake from the start to the goal, without falling into the holes. The best strategy is to reach goals by taking the shortest path.

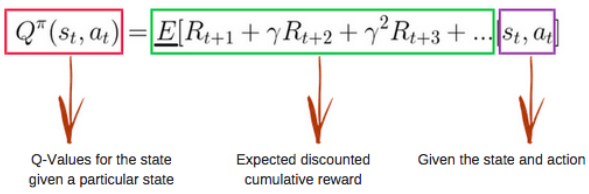


3.2 Q-Table

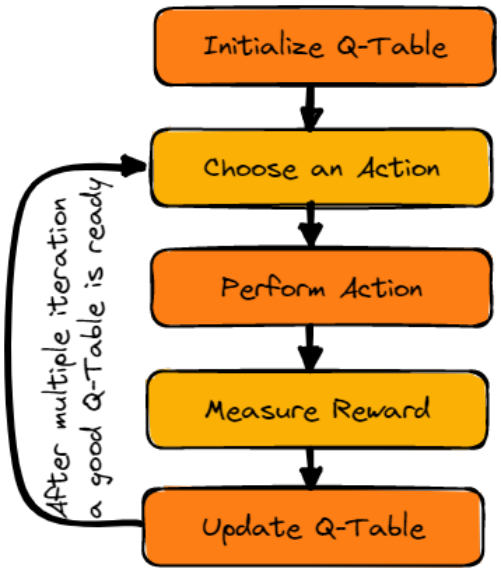
The agent will use a Q-table to take the best possible action based on the expected reward for each state in the environment. In simple words, a Q-table is a data structure of sets of actions and states, and we use the Q-learning algorithm to update the values in the table.

3.3 Q-Function

The Q-function uses the Bellman equation and takes state(s) and action(a) as input. The equation simplifies the state values and state-action value calculation.



3.4 Q-learning algorithm



3.5 Initialize Q-Table

We will first initialize the Q-table. We will build the table with columns based on the number of actions and rows based on the number of states.

In our example, the character can move up, down, left, and right. We have four possible actions and four states(start, Idle, wrong path, and end). You can also consider the wrong path for falling into the hole. We will initialize the Q-Table with values at 0.

	→	←	↑	↓
Start	0	0	0	0
Idle	0	0	0	0
Hole	0	0	0	0
End	0	0	0	0

In the frozen lake example, the agent is unaware of the environment, so it takes random action (move down) to start. As we can see in the above image, the Q-Table is updated using the Bellman equation.

3.6 Measuring the Rewards

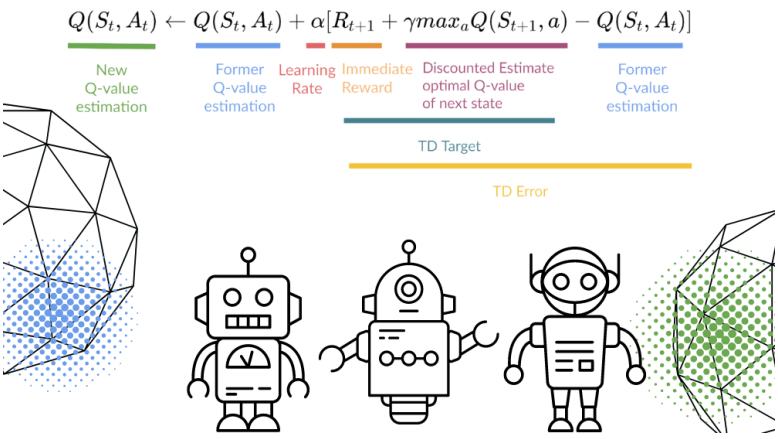
After taking the action, we will measure the outcome and the reward.

- The reward for reaching the goal is +1
- The reward for taking the wrong path (falling into the hole) is 0
- The reward for Idle or moving on the frozen lake is also 0.

3.7 Update Q-Table

We will update the function Q(St, At) using the equation. It uses the previous episode's estimated Q-values, learning rate, and Temporal Differences error. Temporal Differences error is calculated using Immediate reward, the discounted maximum expected future reward, and the former estimation Q-value.

The process is repeated multiple times until the Q-Table is updated and the Q-value function is maximized.



At the start, the agent is exploring the environment to update the Q-table. And when the Q-Table is ready, the agent will start exploiting and start taking better decisions.

	→	←	↑	↓
Start	0	1	0	0
Idle	2	0	0	3
Hole	0	2	0	0
End	1	0	0	0

In the case of a frozen lake, the agent will learn to take the shortest path to reach the goal and avoid jumping into the holes.

4 Explaining code

4.1 Libraries

In this section we imported 3 common libraries numpy , pandas and matplotlib because we was forced to implement this algorithm from scratch and we couldn't use advanced libraries.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

4.2 Parameters

As we discussed before we in section 3 we have some parameters like learning rate , discount factor , number of episode , epsilon(for implementing exploration probability) , possible actions , Q-table.

we loaded the data using pd.read_excel and found the size of data. Then we defined Q-table base on this size.start and end show the start and end point of the table.

```
env =  pd.read_excel("Grid.xlsx" , header = None).to_numpy()
env_rows = env.shape[0]
env_cols = env.shape[1]
actions = ['up', 'right']
q_table = np.zeros((env_rows, env_cols, len(actions)))
start = (29,0)
end = (0, 29)
epsilon = 0.05
discount_factor = 0.9
learning_rate = 0.05
n_runs = 1000
```

4.3 Functions

In this section we defined get_next_location to check the collision to walls.This function get current location(row and column) and action(up or right) then calculate next state if the next state is not in the table it will return current state but if the next state is in the table it will return the next state.

```
def get_next_location(row, col, action):
    new_row = row
    new_col = col
    if actions[action] == 'up' and row > 0:
        new_row -= 1
    elif actions[action] == 'right' and col < env_cols - 1:
        new_col += 1
    return new_row, new_col
```

4.4 Training

Here's the most important part of the code.In this section we trained and updated our Q-table base on reward that we achieve in each action and the formula which was given to us in project's pdf.Also for choosing the action we have a policy which we make random number in [0,1] uniformly and if it is in [0,epsilon] we look at Q-table and find best action and otherwise we choose our action randomly for better exploration.

```
[ ] #TRAIN
    for episode in range(n_runs):
      row, col = start
      while not (row == end[0]  and col == end[1]):
        if np.random.uniform(0,1) < epsilon:
            action =  np.argmax(q_table[row, col])
        else:
            action =  np.random.randint(len(actions))

        old_row, old_col = row, col
        row, col = get_next_location(row, col, action)

        reward = env[row, col]
        old_q_value = q_table[old_row, old_col, action]
        temporal_difference = reward + (discount_factor * np.max(q_table[row, col])) - old_q_value

        new_q_value = old_q_value + (learning_rate * temporal_difference)
        q_table[old_row, old_col, action] = new_q_value
```

4.5 Best Way

According to what we achieved in previous section and the last updated Q-table we find best way by this code.Directions array save best action from start to end and then we save best path base on directions in way array. score also is the whole reward we achieved in our path.

```
armax = np.argmax(q_table,axis=2)
directions = []
direction_slice = []
for j in range(1,armax.shape[1]):
    direction_slice.append('right')
direction_slice.append('finish')
directions.append(direction_slice)
for i in range(1,armax.shape[0]):
    direction_slice = []
    for j in range(armax.shape[1]-1):
        if armax[i,j] == 0:
            direction_slice.append('up')
        else:
            direction_slice.append('right')
    direction_slice.append('up')
    directions.append(direction_slice)

#CREATE WAYS MATRIX
way = np.zeros(np.array(directions).shape,dtype='int16')
(x,y) = start
score = 0
while(True):
    way[x,y] = 1
    score = score + env[x,y]
    if (x == end[0] and y == end[1]):
        break
    elif (directions[x][y] == 'right'):
        y = y + 1
    else:
        x = x - 1
```

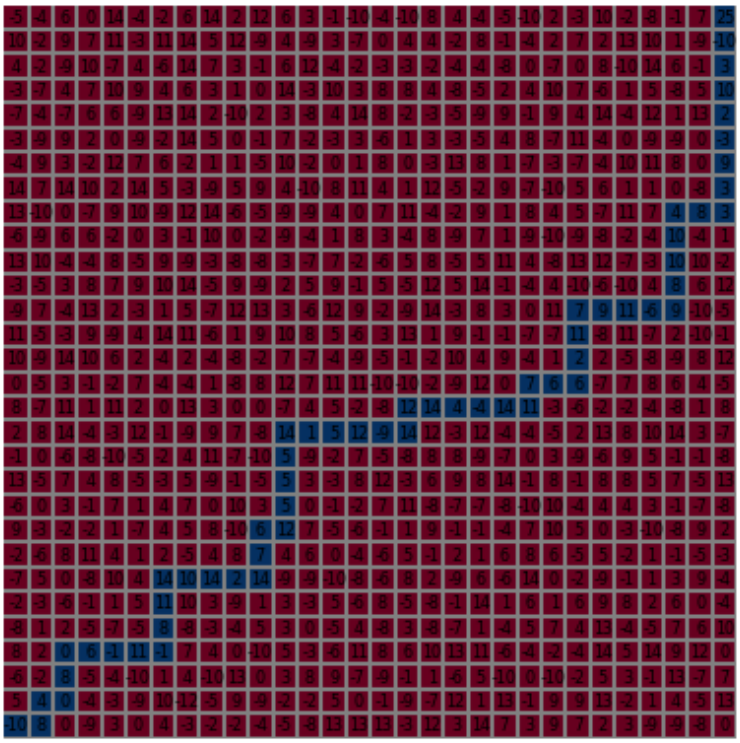
4.6 Plotting result

At the end we plotted our best path in the table by using this code :

```
plt.matshow(way,cmap='RdBu',resample=True)
n = way.shape[0]

plt.hlines(y=np.arange(0, n)+0.5, xmin=np.full(n, 0)-0.5, xmax=np.full(n, n)-0.5, color="gray")
plt.vlines(x=np.arange(0, n)+0.5, ymin=np.full(n, 0)-0.5, ymax=np.full(n, n)-0.5, color="gray")
plt.axis('off')
for (i, j), z in np.ndenumerate(env):
    plt.text(j, i, '{:.0f}'.format(z), ha='center', va='center', fontsize='xx-small',color='k')

plt.figtext(0.5, 0.01, f'SCORE: {score}', ha="center", fontsize=18)

plt.show()
```

5 Conclusion

We ran our code several times and the best result we achieved as follows:



SCORE: 370

6 Refrences

https://www.datacamp.com/tutorial/introduction-q-learning-beginner-tutorial