Reinforcement Learning with Q-Learning

Lan H. Le
Department of Mechanical and Aerospace Engineering
University at Buffalo
Buffalo, NY 14226
hoanglan@buffalo.edu

Abstract

This project's purpose is to build a reinforcement learning agent to navigate the classic 4x4 grid-world environment. The agent will learn an optimal policy through Q-Learning.

1 Introduction

Unlike supervised learning and unsupervised learning where the goals are usually predicting future outputs or classifying objects, reinforcement learning's aim is to "teach" a machine to achieve a desired result. It does that by trying to figure out what course of action would result in the maximum reward.

In this project, we build a Q-learning agent that at the end of training would be able to navigate a 4x4 grid-world environment. The environment and agent will be built to be compatible with OpenAI Gym environments.

2 Environment

OpenAI Gym is a toolkit for developing reinforcement learning algorithms. The framework is easy to understand and establishes a standard environment for developer to build their reinforcement learning agent.

The environment we have is a 4×4 grid-world environment (the initial state for an 4×4 grid-world is shown in Figure 1) where the agent (shown as the green square) has to reach the goal (shown as the yellow square) in the least amount of time steps possible.

The environment's state space will be described as a 4×4 matrix with real values on the interval [0, 1] to designate different features and their positions. The agent will work within an action space consisting of four actions: down (0), up (1), right (2) , left (3). At each time step, the agent will take one action and move in the direction described by the action. The agent will receive a reward of $+1$ for moving closer to the goal and -1 for moving away or remaining the same distance from the goal.

3 Q-Learning Algorithm

Q-learning is a process in which we train some function that will tell us which action leads to maximum reward based on the current state the agent is in. Originally, Q-learning was done in a tabular fashion. Here, we use a 3D array as our Q-Table where the indices of two dimensions represent the state of the agent and the indices of the other dimension represents the action. Each value of the array is the reward corresponding to the state and action at that position. The Q-Table is updated as the agent explores the environment. The update rule is shown in Figure 2.

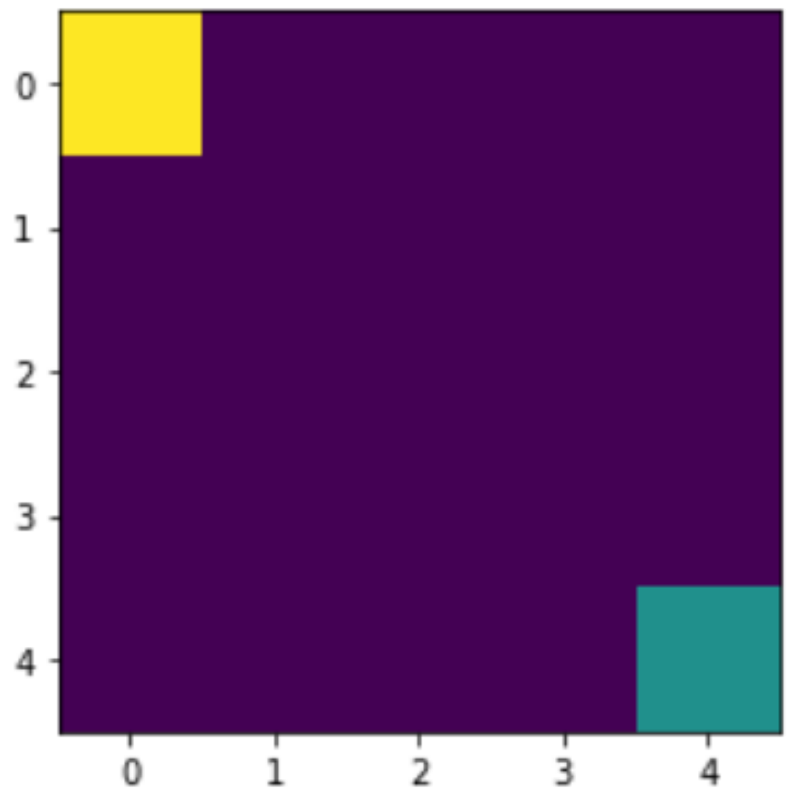


FIGURE 1: THE INITIAL STATE OF GRID-WORLD ENVIRONMENT

$$Q^{new}(s_t,a_t) \leftarrow (1-\alpha)\cdot \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}} \bigg)}^{\text{learned value}}$$

FIGURE 2: UPDATE RULE FOR Q-LEARNING

In order for our agent to "learn" the best path, exploration is essential. In this context, exploration means encouraging the agent to take random actions to gradually figure out which course of actions is the best. However, at some point, the agent also has to follow the optimal path. This is called exploitation. Otherwise, it would spend a lot of time wandering away from the end goal.

As a result, epsilon is introduced to balance exploration and exploitation. During each time step, we pick a random single value on the uniform distribution over [0, 1] and compare this value to epsilon. If the value is less than epsilon the agent takes a random action. Otherwise, it takes an action following the Q-Table for optimal path. Since exploration is more important in the early stages we initially set epsilon to be 1 and gradually (exponentially) decrease it at every training step until it reaches 0.

The number of training steps (episodes) also needs to be set properly for efficient training. For this project, the number of episodes is 1000.

In order to train the Q-learning agent, we let it "play" the game multiple times to figure out the optimal strategy. At the beginning of each episode, we reset the environment. Next, we generate an action based on the policy that we set (exploration versus exploitation). We then use that action to find out the next state of the agent as well as the reward and whether the agent has reached the final destination at that point. Gradually, the G-Table is updated and later appropriately used by the agent to obtain an optimal path.

4 Challenges

For many cases, the Q-Table is two-dimensional which is convenient and slightly more intuitive for updating. However, in this project, the Q-Table is a three-dimensional array and the updating process has to be adjust accordingly.

In addition, the environment's state space is described as an 4×4 matrix with real values on the interval [0, 1] to designate different features and their positions. Therefore, before using the state's values to update Q-Table, we need to convert these real values to integers.

5 Results

Figure 3 is a graph showing how epsilon is changed over 1000 episodes. It decreases using exponential decay until it reaches 0.

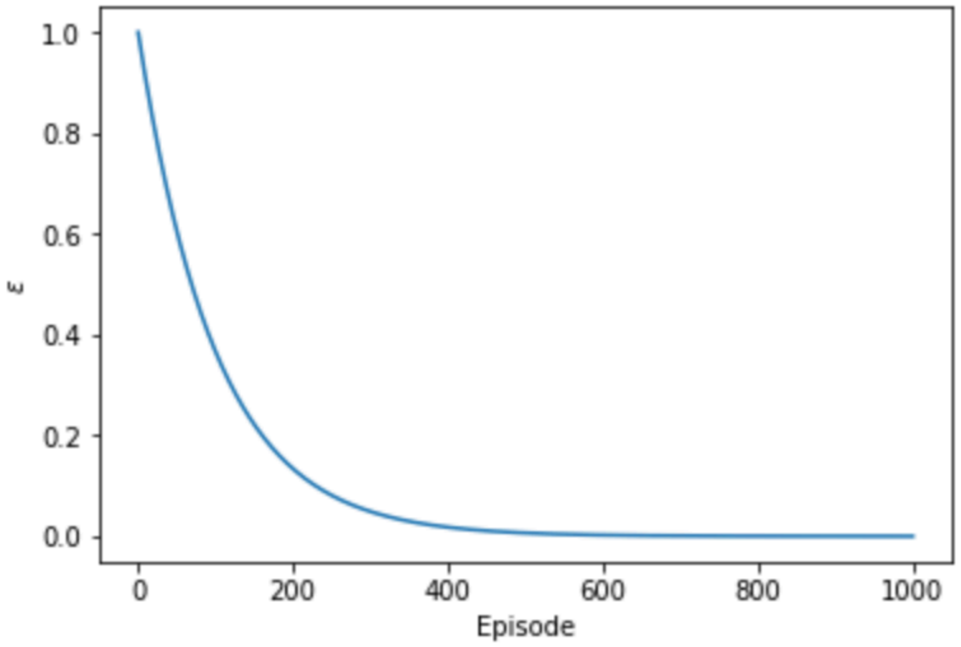


FIGURE 3: EPSILONS OVER EPISODES

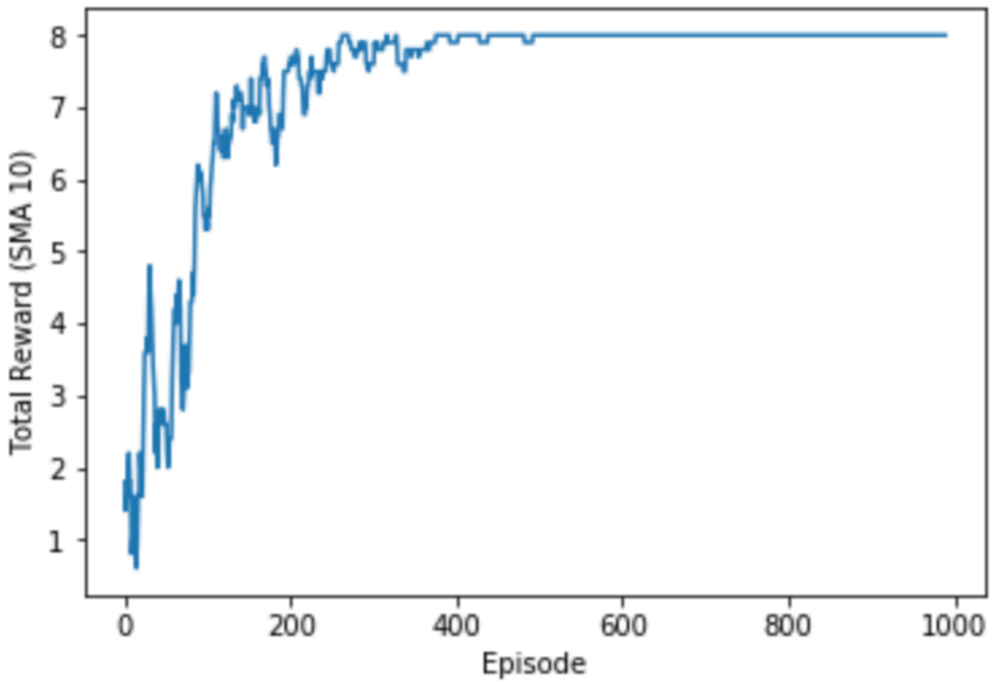


FIGURE 4: REWARDS OVER EPISODES

The total reward in each episode follows an increase trend and eventually reaches the maximum (8) as training concludes. At this point, the Q-Table is fully updated and the agent is ready to navigate to the end destination in the most efficient way.

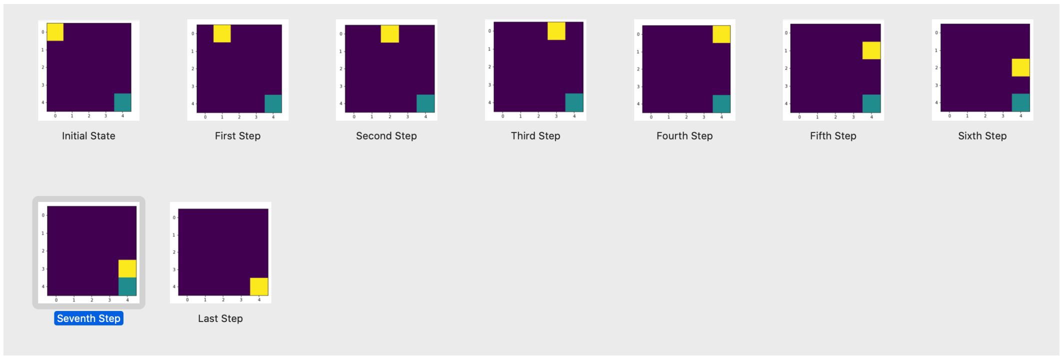
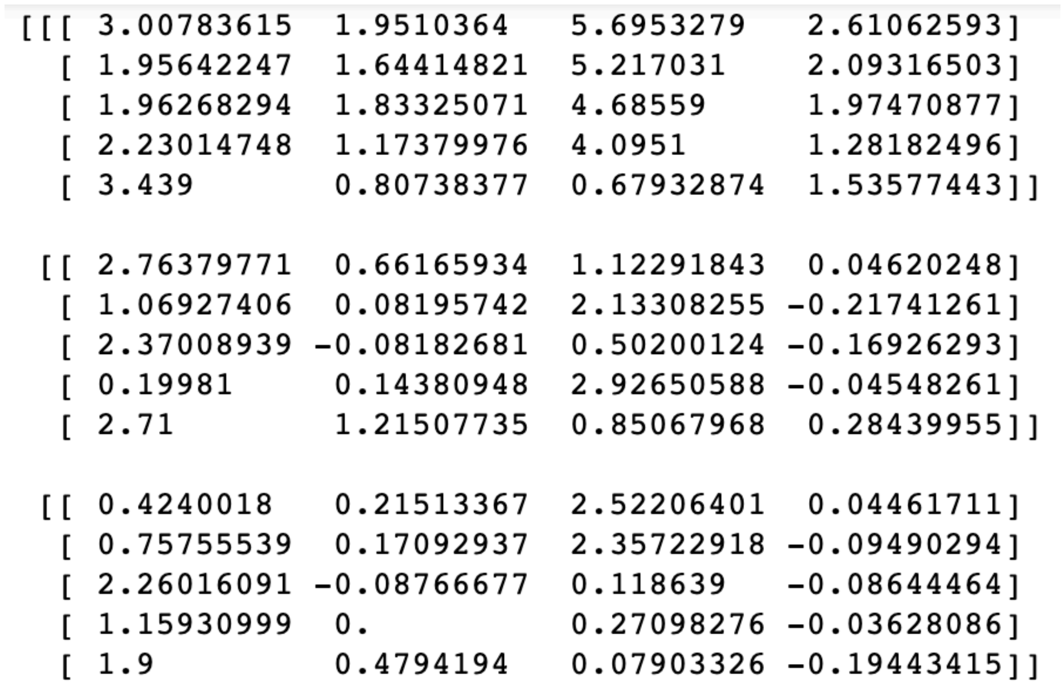


FIGURE 5: FINAL RESULT

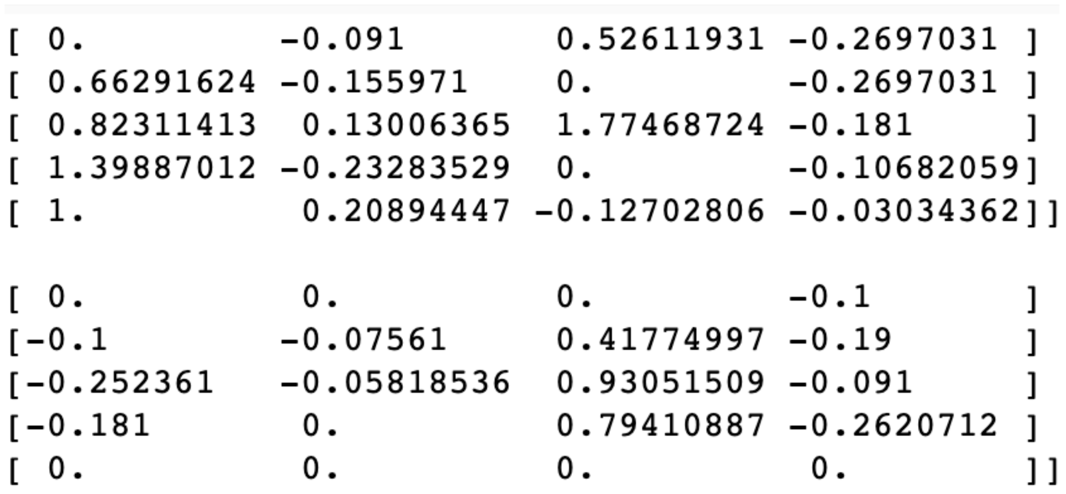


```
[[[ 3.00783615  1.9510364   5.6953279   2.61062593]
  [ 1.95642247  1.64414821  5.217031    2.09316503]
  [ 1.96268294  1.83325071  4.68559     1.97470877]
  [ 2.23014748  1.17379976  4.0951      1.28182496]
  [ 3.439       0.80738377  0.67932874  1.53577443]]

 [[ 2.76379771  0.66165934  1.12291843  0.04620248]
  [ 1.06927406  0.08195742  2.13308255 -0.21741261]
  [ 2.37008939 -0.08182681  0.50200124 -0.16926293]
  [ 0.19981     0.14380948  2.92650588 -0.04548261]
  [ 2.71        1.21507735  0.85067968  0.28439955]]

 [[ 0.4240018   0.21513367  2.52206401  0.04461711]
  [ 0.75755539  0.17092937  2.35722918 -0.09490294]
  [ 2.26016091 -0.08766677  0.118639   -0.08644464]
  [ 1.15930999  0.          0.27098276 -0.03628086]
  [ 1.9         0.4794194   0.07903326 -0.19443415]]
```

FIGURE 6: Q-TABLE FIRST THREE ROWS



```
[ 0.         -0.091       0.52611931 -0.2697031 ]
[ 0.66291624 -0.155971    0.         -0.2697031 ]
[ 0.82311413  0.13006365  1.77468724 -0.181     ]
[ 1.39887012 -0.23283529  0.         -0.10682059]
[ 1.          0.20894447 -0.12702806 -0.03034362]]

[ 0.          0.          0.         -0.1       ]
[-0.1        -0.07561     0.41774997 -0.19      ]
[-0.252361   -0.05818536  0.93051509 -0.091     ]
[-0.181       0.          0.79410887 -0.2620712 ]
[ 0.          0.          0.          0.        ]]
```

FIGURE 7: Q-TABLE LAST TWO ROWS

6 Conclusion

With reinforcement learning (Q-learning in this project), we are able to build an agent that could autonomously navigate in a grid-world environment in the most efficient way. The process of figuring out the optimal path by updating the Q-Table is simple yet effective. In the future, we would like to explore other algorithms of reinforcement learning so as to compare them with Q-learning. Within Q-learning, it would also be interesting to see how different ways of epsilon decay affect the speed and efficiency of Q-learning training.

Acknowledgments

The descriptions of the environment and algorithm are derived from the original project description.

References