Homework#3: Q-Learning for Maze Traversal

Submission date: 05/17/2023

2018312164

김석진

◆Research and Study

▶ What is Q-Learning?

Q-Learning is a model-free reinforcement learning algorithm used to solve Markov decision processes (MDPs). It is a form of value iteration learning that aims to learn the optimal action-value function, known as the Q-function. Q-Learning is a powerful technique for learning optimal policies in environments where the transition probabilities and rewards are unknown.

▶ Q-Table:

In Q-Learning, the agent maintains a Q-table, which is a lookup table that stores the action-value estimates for each state-action pair. The Q-table is initialized with arbitrary values and gets updated as the agent interacts with the environment. It enables the agent to make informed decisions by choosing actions with the highest Q-value for a given state.

▶ E-Greedy Strategy:

The E-Greedy strategy is used to balance exploration and exploitation in Q-Learning. During action selection, the agent chooses the action with the highest Q-value with a probability of (1 - epsilon). This exploitation allows the agent to choose the optimal action. However, with a probability of epsilon, the agent selects a random action, promoting exploration to discover new and potentially better actions.

▶ Q-Learning Update Rule:

The Q-Learning algorithm uses the Q-Learning update rule to iteratively update the Q-values in the Q-table. The update rule is as follows:

Q(s, a) = Q(s, a) + alpha \* (r + gamma \* max(Q(s', a')) - Q(s, a))

Here, Q(s, a) represents the Q-value for a particular state-action pair (s, a). r is the immediate reward received after taking action a in state s. gamma is the discount factor that determines the importance of future rewards. alpha is the learning rate that controls the weight of the new information compared to the existing Q-values. Q(s', a') represents the maximum Q-value among the possible actions in the next state s'.

▶ Parameters:

- Alpha (Learning Rate): It determines the extent to which newly acquired information overrides the existing Q-values. A higher value gives more weight to the new information, while a lower value makes the algorithm rely more on prior knowledge.

- Gamma (Discount Factor): It determines the importance of future rewards. A gamma value closer to 1 indicates that the agent values long-term rewards, while a value closer to 0 makes the agent focus more on immediate rewards.

- Epsilon (Exploration Rate): It controls the balance between exploration and exploitation. A higher epsilon promotes more exploration, while a lower epsilon leads to more exploitation of the learned Q-values.

- Episode: An episode is a complete run of the agent interacting with the environment, starting from the initial state until it reaches a terminal state or a maximum number of steps. The agent learns from each episode to improve its performance over time.

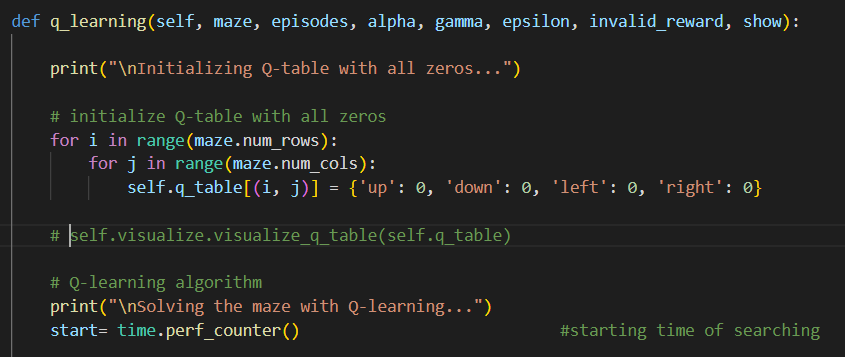
◆ MyQLearning class code review

This class contains two method to perform q-learning and search through the maze. As descripted in the assignment specification two methods name is q\_learning() and q\_learning\_path()

1. Implement the Q-Learning algorithm by adding a new function to the existing codebase. The function should be named q\_learning and should take as input a maze object, the number of episodes, the learning rate (α), the discount factor (γ), and the exploration rate (ε).
2. Create a function named q\_learning\_path that takes as input a maze object and a learned Q-table, and returns the found path from the starting position to the goal position, along with its cost

The following report is about how to implement these two methods.

◆ q\_learning() code review



<Q-Table Initialization>

The code initializes the Q-table with all zeros. It iterates over the rows and columns of the maze using nested for loops. For each coordinate (i, j) in the maze, the Q-table is updated with a dictionary containing the four possible actions ('up', 'down', 'left', 'right') as keys, and their corresponding initial Q-values set to 0.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

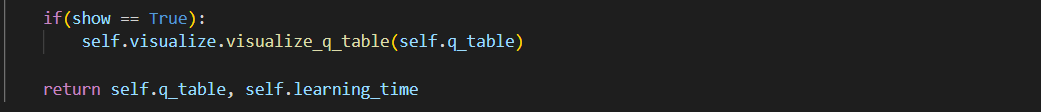
자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<Q-Learning Algorithm>

1. The episode loop iterates over the specified number of episodes.
2. The agent starts at the entry coordinate of the maze.
3. A while loop continues until the agent reaches a terminal state.
4. In each iteration of the while loop, the agent chooses an action based on an epsilon-greedy policy. With a probability of epsilon, a random action is selected, otherwise, the action with the maximum Q-value for the current state is chosen.
5. The agent takes the chosen action and observes the new state and the associated reward.
6. Based on the new state, the reward, and the Q-values, the agent updates the Q-table using the Q-Learning update rule. The Q-value for the current state-action pair is updated using the immediate reward and the maximum Q-value among the possible actions in the next state.
7. The loop continues until the agent reaches a terminal state.
8. The total learning time is calculated by measuring the elapsed time from the start of the Q-Learning process until completion of all episodes.



<Visualization>

If the 'show' parameter is set to True, the code visualizes the Q-table using a visualization function. This visualization can provide insights into the learned Q-values for each state-action pair.

And lastly returns the final q-table and the time it takes to update the q-table.

◆ q\_learning\_path() code review

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<Starting State and Initialization>

The function starts at the entry coordinate of the maze as the initial state. It initializes the path variable with a list containing the current state and a Boolean flag to indicate whether the state has been visited. The cost variable is initialized to 0, representing the total cost of the path.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<Path Generation Loop>

The function enters a loop that continues until the goal state (exit coordinate of the maze) is reached. Within each iteration of the loop, the function performs the following steps:

1. Selects the action with the highest Q-value for the current state. It retrieves the possible actions for the current state from the Q-table, calculates their corresponding Q-values, and chooses the action with the maximum Q-value. In case there are multiple actions with the same maximum Q-value, a random action is selected among them.
2. Prints the current state and the chosen action for informational purposes.
3. Calculates the next state and the cost associated with the chosen action. The next state is determined based on the current state and the chosen action, while the step cost is obtained from the maze's grid.
4. Adds the step cost to the total cost.
5. Appends the next state to the path list along with a Boolean flag indicating that it has not been visited yet.
6. Updates the current state with the next state.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<Path Length Limit & return value>

To prevent infinite loops or excessively long paths, the code includes a check to break the loop if the cost of the path exceeds a specified maximum path length (self.max\_path). This helps ensure that the path generation process terminates within a reasonable number of steps.

Finally, The function returns the generated path as a list of state-action pairs and the total cost of the path.

◆ get\_reward() code review (IMPORTANT)

Get\_reward function returns expected reward from next state. And as in the assignment description says the reward of exit is 100, is\_wall is -1, and else -0.1. For optimization I add a new parameter called wall\_reward in order to change the reward of going through a wall but I stick to -1 reward initially.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

◆ visualize\_q\_table () code review

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷이(가) 표시된 사진

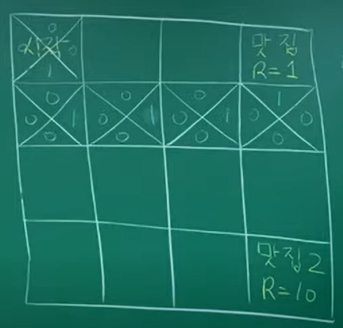
자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<Showing 4 expected reward in every cell and 4 actions>

visualize\_q\_table() code give a table with the same size of the maze. And each cell contains 4 values. 4 values looks like this.



In each position there are 4 actions to choose and each action has a expecting reward. By showing these value in a cell it is easy to know the q-learning is working as expected.

<Labeling start and end coordinate>

Also, I colored the starting coordinate and exiting coordinate. Green as entry, red as exiting.

◆ Main.py code review  
Main.py file is for executing the q-learning algorithm. Executing two task provided which are preceeding q-learning in 3 20x20 maze and show the path, and change the parameter alpha, gamma to optimize the result.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Initially I choose the parameter to be 1000~5000 iteration, learning rate of 0.1, decay factor of 0.8, epsilon to be 0.

And the reward of breaking through a wall is initially set to -1 as in the assignment.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

<Task1>

In 3 different 20x20 maze I preformed q-learning and see the result

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<Task 2>

To optimize the learning process and see the impact of changing the parameters I tried 16 different combinations of alpha, and gamma parameters.

Range of Alpha as [0.2, 0.4, 0.6, 0.8] ,and Gamma as [0.2, 0.4, 0.6, 0.8].

◆ Result

Task 1 Result

< Maze1 q-table>

20x20 table with each cell having value of reward by action it takes. Red indicates exit Green indicates start.

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명<Maze1 Solution Path>

도표, 직사각형, 패턴, 사각형이(가) 표시된 사진

자동 생성된 설명

< Maze2 q-table>

20x20 table with each cell having value of reward by action it takes. Red indicates exit Green indicates start.

텍스트, 스크린샷, 평행, 번호이(가) 표시된 사진

자동 생성된 설명

<Maze2 Solution Path>

도표, 직사각형, 사각형, 패턴이(가) 표시된 사진

자동 생성된 설명

< Maze3 q-table>

20x20 table with each cell having value of reward by action it takes. Red indicates exit Green indicates start.

텍스트, 스크린샷, 번호, 라인이(가) 표시된 사진

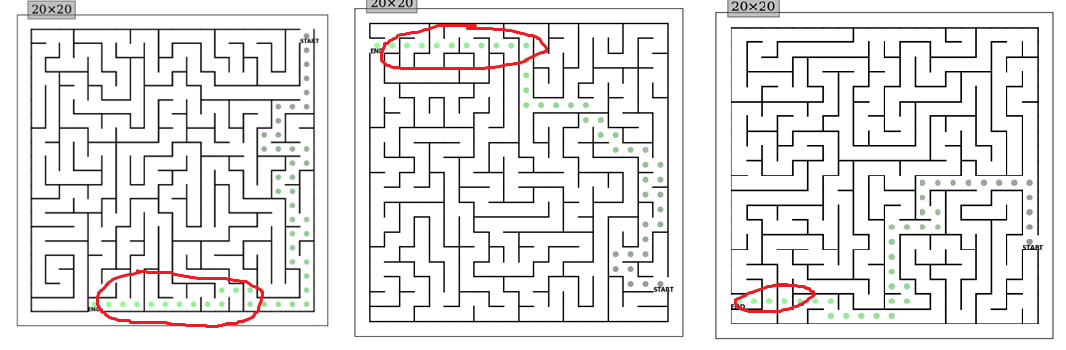
자동 생성된 설명

<Maze3 Solution Path>

직사각형, 도표, 사각형, 패턴이(가) 표시된 사진

자동 생성된 설명

<result analysis>



q-learning is successfully implemented but it seems that the agent is breaking through a wall when the goal is too close like the agent can not wait to get to the goal. About this problem I wrote in optimization part how to twick with the parameter.

Task 2 Result

In a same 20x20 maze I tried 16 different parameter combinations and analysis the performance. These are the result.

텍스트, 도표, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

<result analysis>

In the right graph we can see a tendency of learning speed increase when alpha is lager. But when alpha is less then 0.2 there is no performance diffence when gamma changes. So we can assume alpha needs to be less than 0.2 and gamma gives no good impact in time performance.

In the left graph we can see that gamma lager then 0.2 always finds the optimal path. And value less than 0.2 finds a suboptimal path or even didn’t finds the exit and stops in max iteration.

◆ Optimization

I tried many ways to optimize the result. My goal was to never break through a wall and give a optimal complete path.

Attempt1: increase iteration<Fail>

<1000 iteration>

도표, 직사각형, 사각형, 패턴이(가) 표시된 사진

자동 생성된 설명

<2000 iteration>

도표, 직사각형, 패턴, 평면도이(가) 표시된 사진

자동 생성된 설명

<5000 iteration>

직사각형, 도표, 사각형, 패턴이(가) 표시된 사진

자동 생성된 설명

Attempt2: restrict an agent to go through a wall<Fail>

After second TA office meeting I found out that it is forbidden to change the reward instead we need to change the visualize\_q\_table() so that the agent choose an action that is not going through the wall and with the biggest reward. So I decided to add a new fuction that filter the action. Only return action that is not going through the wall. Then existing code can evaluate which action has the most expected reward.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

도표, 스크린샷, 텍스트, 직사각형이(가) 표시된 사진

자동 생성된 설명도표, 스크린샷, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명

It is because the reward around the wall is all pointing toward the wall. So the agent is just moving around the wall stuck.

Attempt4: restrict an agent go through a wall in learning<Fail>

After second TA office meeting I found out that it is forbidden to change the reward instead we need to change the visualize\_q\_table() so that the agent choose an action that is not going through the wall and with the biggest reward. So I decided to make a agent go back to the current position in learning when the agent is going through a wall.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

This code prevent the agent to go path the wall in learning porcess. And the result was disappointing. The learning time was so long about 30min per 100 iteration and it also break through the wall.

Attempt5: reducing gamma parameter<Success>

Gamma parameter determines the decay factor so less gamma will se the short reward. And the short reward must be not going through the wall so I thought reducing gamma will give a change.

<gamma 0.8>

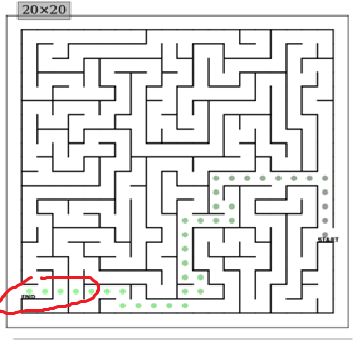
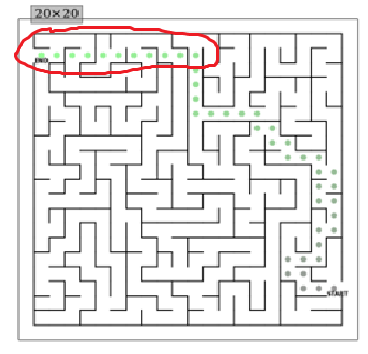
도표, 직사각형, 사각형, 패턴이(가) 표시된 사진

자동 생성된 설명

<gamma 0.6>

직사각형, 도표, 사각형, 패턴이(가) 표시된 사진

자동 생성된 설명



We can see the length of breaking through a wall decreases! But still when the goal is too close it seems the agent can’t resist the erge of breaking through a wall.

Attempt6: gradually decreasing gamma parameter<Success>

There is a method of decreasing gamma parameter to prevent the agent acting weird as it approaches the goal. So I tried that method.

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

패턴, 도표, 직사각형, 사각형이(가) 표시된 사진

자동 생성된 설명

We can see that the agent is successfully solve the maze.

직사각형, 패턴, 도표, 사각형이(가) 표시된 사진

자동 생성된 설명

But in some maze. A small problem is still left. Just before the goal it break through.

Attempt7: changing the reward of breaking through<Success>

Get\_reward function returns expected reward from next state. And as in the assignment description says the reward of exit is 100, is\_wall is -1, and else -0.1. For optimization I changed the value so that the agent can know breaking through a wall will cost a lot.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

<wall\_reward = -10000>

With a ridiculously large wall\_reward the agent returns a optimal path.

패턴, 도표, 직사각형, 사각형이(가) 표시된 사진

자동 생성된 설명

도표, 평면도, 직사각형, 사각형이(가) 표시된 사진

자동 생성된 설명

◆ Optimization Conclusion

Using decaying gamma method we can optimize the agent to prevent going through a wall and giving an optimal path. Or increase the cost of going through the wall.

<Final task 1 result>

These are the maze solved.

도표, 평면도, 직사각형, 사각형이(가) 표시된 사진

자동 생성된 설명 도표, 직사각형, 사각형, 평면도이(가) 표시된 사진

자동 생성된 설명