Machine Learning Smart cab drive project

In your report, mention what you see in the agent's behavior. Does it eventually make it to the target location?

In section, I set the policy of the agent as following. If the agent gets a reward larger than its previous rewards, it will go forward if the traffic environment allows. If the rewards are less than or equal to the previous reward, it will generate a random action in all possible actions that traffic environment allows for. The agent was able to reach the destination. However, the success rate is low. In 100 trails only 4 ~7 times the agent reaches the destination (please see the agent.txt).

Identify a set of states that you think are appropriate for modeling the driving agent. The main source of state variables are current inputs, but not all of them may be worth representing. Also, you can choose to explicitly define states, or use some combination (vector) of inputs as an implicit state.
At each time step, process the inputs and update the current state. Run it again (and as often as you need) to observe how the reported state changes through the run.
Justify why you picked these set of states, and how they model the agent and its environment.

There are five variables in states: light, oncoming, right, left and waypoint. Their possible values are as following:

Light:{'green','red'}

Oncoming:{'forward','left','right',None}

Left: {'forward','left','right',None}

Right{'forward','left','right',None}

Waypoint{'forward','left','right',None}

Therefore, the agent can have 2*4*4*4*4=512 different kinds of status. However, not all of the status is necessary. The ‘Left’ and ‘Right’ are not relevant for the agent to reach the destination and therefore can be deleted.

Deadline is also not considered here because it will make the Q matrix too sparse and thus a long time to converge.

The state here will be combination of the Light, Oncoming and Waypoint variables. It tells the agent the environment. A typical state will be {‘light’:’green’,’Oncoming’:’forward’,’Waypoint’:’left’}. The agent can take four actions: right, left, forward and none. So the Q matrix will have 2*4*4*4=128 values. The values are filled up when the agent roams around the world.

Implement Q-Learnings

What changes do you notice in the agent’s behavior?

The success rate of the agent has been greatly improved. 85 out of 100 times, the agent reached the destination. In the first few trails, the agent fails to reach the destination because it is roaming around the world to fill up the Q matrix. As the Q matrix is more filled up, the agent if much more like to reach the destination. Also the possible steps required are also reduced as the agent has more trails.

The following parameters are initialized in order to implement the Q-learnings:

1 The discount rate: set to be 0.33.

2 The learning rate: set to be 0.6

3 The epsilon: set to be 0.1. Measures how likely the agent will take a random action. A higher value indicates that the agent will take a random action.

4 All default Q values are set to be 0.

The action with the highest Q value is selected and the Q matrix is updated each iteration with rewards+discount*max Q values of next state.

Apply the reinforcement learning techniques you have learnt, and tweak the parameters (e.g. learning rate, discount factor, action selection method, etc.), to improve the performance of your agent. Your goal is to get it to a point so that within 100 trials, the agent is able to learn a feasible policy - i.e. reach the destination within the allotted time, with net reward remaining positive.

Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does it perform?

Does your agent get close to finding an optimal policy, i.e. reach the destination in the minimum possible time, and not incur any penalties?

Change the default Q value from 0 to 20, which improve the success rate from 85% to 93%.

The intuitions behind the change is that, with a larger default Q value, the agent is more likely to try a new path and thus the Q values converge sooner, which improve the performance of the Q learning algorithm.

To make a conclusion, since the penalty rate of the Q-learning algorithm is 0.05 and it reaches a success rate more than 90%, the agent is close to finding an optimal policy.