Project 4: Train a Smartcab

Training a smartcab using Reinforcement Learning

Task 1

Implement a basic Learning Agent

Actions are randomly choosen at each iteration using **`random.choice(available_actions)`**

```
if inputs['light'] == 'red':
            """
                Red Light means :
                    Left when no oncoming traffic
                    No action
            """
            if inputs['oncoming'] != 'left':
                available_actions = [None, 'right']
        else:
            """
                Green Light means :
                    Left only if no forward oncoming traffic
                    Perform atleast some action
            """
            available_actions = ['right', 'left', 'forward']
            if inputs['oncoming'] == 'forward':
                available_actions.remove('left')
        action = None
        if available_actions != []:
            action = random.choice(available_actions)
```

Q1. Mention what you see in the agent's behavior. Does it eventually make it to the target location?

The output as a result of the random movement algorithm (`/smartcab/test_random.txt`):

> **32 trips out of the 100, the cab reaches the destination.**

The agent is able to reach the destination more than 25% of the trips while still following all the traffic rules. It does eventually make it to the target location however since it wasn't behaving intelligently at any point of time, hence it has a very low success rate. This was when I set `enforce_deadline = True`
When the deadline is not enforced,

> **81 trips out of the 100, the cab reaches the destination.**

The behaviour was as expected because the randomised movements will take the cab to the destination however in majority of the cases not under the deadline.

Q2. Identify and Update State. Justify why you picked these set of states, and how they model the agent and its environment.

The state that I chose is represented by the input parameters, the next waypoint and the deadline.
I chose these parameters for my state because it represents the scenario of real life driving to a very good extent. In real life, while driving a cab, one considers oncoming traffic, lights etc. as well as in how much time one has to get there and the proposed route using waypoints. Another factor that I considered while choosing state variables was the availability of these in real life. The state variables that I chose are available through sensors in a self driving car.

Task 2

Implement Q Learning

The Q Learning algorithm I followed is exactly as mentioned in the video Lectures.
The state is represented by the input parameters **`oncoming`** and **`light`**, and **`the next waypoint`**. These three represent the exact state of the cab at a point of time. The primary aim is to train the cab for performing legal moves. Legal moves result in the highest possible reward. Even in real life, following the traffic rules is slightly more important than reaching before the deadline. Therefore in this case I haven't included the deadline as a state variable. The Q Learning Algorithm is explained below :

1. Construct the State Representation
2. Obtain Action for that state
    A. Random action is choosen with epsilon probability.
    B. Otherwise choose action using the policy of greatest q value gives best action.
3. Update Q Values according to reward by the action.

Q3. What changes do you notice in the agent's behavior ?

After running `agent.py` I found out that the agent is now considerably more accurate than the random agent. Tests revealed a success rate of ***97/100***(`/smartcab/test_q_100.txt`). In a seperate test with number of trials = 200, the agent reached the target ***195/200***(`/smartcab/test_q_200.txt`) times. The agent behaves intelligently at every decision point, based on the q values. Randomness has been minimised by setting a very low epsilon.

Q4. 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?

The parameters of the initial implementation of the Q Learning Algorithm were totally random at **alpha = 0.5**, **gamma = 0.5**, **epsilon = 0.5**, and **initial Q Value = 20.0**. After a host of testing with various parameters, I arrived at the optimal parameters:

- **alpha = 0.9**: High learning rate is desirable
- **epsilon = 0.001**: Provides sufficient amount of randomness in the selection of an action. Our aim is to

obtain maximum actions from the learning policy.

- **gamma = 0.35**: Brute Force
- **Initial Q Value = 19.75**: Brute Force

As mentioned above the algorithm performs better as compared to the random agent. **97 out of 100** is a