MLND - P4 - Vishakh Rayapeta

1) Implement a basic driving agent

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

Using a random action gets the agent to the destination eventually. In most cases, the agent reaches the target location after the deadline value has expired. The agent's performance over consecutive trips is random - in some cases the target location is reached early and in other cases much later. The agent does not learn to avoid traffic violations (negative rewards).

2) Identify and update state

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

The goal of the learning agent is to reach the destination by following the directions from the planner without violating any traffic rules.

The Q-state selected comprises of {next_waypoint, light, oncoming, left}.

- *The first input 'next_waypoint' must be learnt by the agent to successfully reach the assigned destination by the planner. This input represents the next waypoint that the cab must move to in order to reach its destination.*
- *The next three inputs {light, oncoming, left} must be learnt by the agent in order to obey traffic rules. The traffic on the right side of the waypoint does not impact any traffic rule and can be disregarded.*
- *The input 'light' indicates the state of the traffic light. This input must be learnt to avoid jumping a red light when going forward or left.*
- *The input 'oncoming' indicates if there is any oncoming traffic. This input must be learnt to avoid running into traffic when making a left turn.*
- *The input 'left' indicates if there is any facing traffic waiting to make a left turn. This input must be learnt when making a right turn.*

*I am using one binary bit (0 or 1) to represent the state of the light (red or green). Two binary bits (00, 01, 10, 11) are used to represent the state of next_waypoint, oncoming & left (None, right, left, forward). This translates to a total of (2 * 4 * 4 * 4 = 128) states in the Q-matrix .*

3) Implement Q-learning

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

The initialization of the Q-matrix elements follows the optimistic strategy. All elements are set to a positive value (3) that is slightly above the most common reward value (2). This encourages the learning agent to explore as many states as possible at least once. Rewards are derated by an 'alpha value' and added to the existing state value.

The initial trial(s) provide the most learning opportunity for the agent. During the initial trials the agent explores many states and begins to learn to follow the planner and the traffic rules.

After implementing Q-learning, the agent manages to reach the target location before the deadline value has expired. The agent's performance over consecutive trips shows improvement that can be measured both in terms of the ability to avoid traffic violations (negative rewards) and to reach the destination before the deadline.

Here are some Q-state examples (examined using the log messages):

Q-state 80: *next_waypoint = left, light = green, oncoming = None, left = None*

In this state the planner requires the cab to turn left. Also, the light is green and there is no other traffic at the intersection. The agent learns the correct action of turning left after 4 attempts. Once the correct action is learnt, the agent continues to apply it every time the state is entered.

Q-state 96: *next_waypoint = forward, light = red, oncoming = None, left = None*

In this state the planner requires the cab to go forward. However the light is red and this would be a traffic violation. The agent learns the correct action of 'None' after 5 attempts. Once the correct action is learnt, the agent continues to apply it every time the state is entered.

NOTE: Since the simulator uses randomization in generating the environment, training curve of the agent will vary a little bit between consecutive runs.

4) Enhance the driving agent

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?

Learning Rate (alpha)

I selected an alpha value of 0.6 to encourage a gradual but quick learning rate.

Table 1: Number of negative rewards accumulated in 100 trips as a function of alpha (learning rate)

Alpha (Learning Rate)	*Trial 1* *num of negative rewards*	*Trial 2* *num of negative rewards*	*Trial 3* *num of negative rewards*	*Average (lower is better)*
0	*3045*	*skip*	*skip*	***3045***
0.1	*49*	*44*	*45*	***46***
0.2	*33*	*30*	*27*	***30***
0.5	*20*	*15*	*25*	***20***
0.6	*15*	*14*	*16*	***15***
0.9	*17*	*13*	*16*	***15.3***
1.0	*19*	*12*	*14*	***15***

Using low alpha values (0.1, 0.2) caused the learning process to require lot more attempts before settling to the proper Q-state values. Higher alpha values (0.6 → 1.0) resulted in faster learning of the Q-state values.

Table 2: Number of trips (100 trials) completed within the deadline as a function of learning rate

Alpha (Learning Rate)	*Trial 1* *trips completed before deadline*	*Trial 2* *trips completed before deadline*	*Trial 3* *trips completed before deadline*	*Average (higher is better)*
0	*14*	*skip*	*skip*	***14***
0.1	*97*	*97*	*98*	***97.3***

0.2	*97*	*97*	*97*	***97***
0.5	*97*	*99*	*98*	***98***
0.6	*99*	*99*	*98*	***98.7***
0.9	*99*	*99*	*98*	***98.7***
1.0	*99*	*100*	*100*	***99.7***

'Table:2' lists the number of trips completed before the deadline as a function of the learning rate. The agent learnt to follow the planner directions with all learning rates > 0. However, higher learning rates (0.6 → 1.0) achieved a slightly better trip completion rate (~98+%).

Decaying Learning Rate (not implemented)

One approach to consider when selecting a learning rate is to start with a high rate (say 0.9) and then gradually reduce the learning rate as more states are explored. This will ensure that the state values do not change by a lot once they have been learnt. Setting the learning rate as some function of the inverse of the number of trials will achieve this. This technique is not implemented.

In this project, setting a static learning rate (0.6 or above) achieves good results over 100 trials -

- *Agent is able to reach the destination before the deadline ~98% of the time.*
- *Total number of negative rewards is ~15.*

Epsilon Greedy Exploration (not implemented)

An alternative exploration approach introduces randomization while selecting the next Q-state. In this approach, the action with the highest Q value is selected with a probability of (1 - ε) and a random action is selected with a probability of 'ε'.

I have instead chosen to initialize the Q-matrix (for fast convergence & exploration) and used the optimistic greedy approach. This approach assumes that the task of learning is static and does not change over time. So, it is focused on learning the task during the first few trials and then simply implementing the learnt actions. If the task itself is changing (with changes to the reward structure), using the epsilon greedy exploration will allow 're-learning' and adjusting to the change.