Udacity MLND P4: Train a Smartcab

Implement a Basic Driving Agent

To begin, your only task is to get the **smartcab** to move around in the environment. At this point, you will not be concerned with any sort of optimal driving policy. Note that the driving agent is given the following information at each intersection:

- The next waypoint location relative to its current location and heading.
- The state of the traffic light at the intersection and the presence of oncoming vehicles from other directions.
- The current time left from the allotted deadline.

To complete this task, simply have your driving agent choose a random action from the set of possible actions (`None`, `'forward'`, `'left'`, `'right'`) at each intersection, disregarding the input information above. Set the simulation deadline enforcement, `enforce_deadline` to `False` and observe how it performs.

Question: *Observe what you see with the agent's behavior as it takes random actions. Does the* ***smartcab*** *eventually make it to the destination? Are there any other interesting observations to note?*

Answer:

As the agent starts to take random action, it no longer stuck at the same point and starts to explore the environment. However, the agent's behavior is random. Sometimes it gets to the destination, sometimes it does not. As the number of iteration increases, the agent still does not get any better in reaching the destination. In short the agent does not learn over time.

Inform the Driving Agent

Now that your driving agent is capable of moving around in the environment, your next task is to identify a set of states that are appropriate for modeling the **smartcab** and environment. The main source of state variables are the current inputs at the intersection, but not all may require representation. You may choose to explicitly define states, or use some combination of inputs as an implicit state. At each time step, process the inputs and update the agent's current state using the `self.state` variable. Continue with the simulation deadline enforcement `enforce_deadline` being set

to False, and observe how your driving agent now reports the change in state as the simulation progresses.

Question: *What states have you identified that are appropriate for modeling the* ***smartcab*** *and environment? Why do you believe each of these states to be appropriate for this problem?*

Answer:

The states that are appropriate for modeling the smartcab and environment are as listed below:

1. Light: red or green
2. Oncoming: None, forward, left, or right
3. Left: None, forward, left, or right
4. Next waypoint: None, forward, left, or right

The state `light` is essential for modeling the smartcab and environment because many of rewards and penalties are dependent on the action taken at particular state of `light`. For instance, if the smartcab moves forward at a green light, it is considered a good move and hence is potential to get a reward (subjected that other criteria are fulfilled). On the other hand, if the smartcab moves forward at a red light, it will directly get a penalty of -1 as shown in the example below:

> LearningAgent.update(): deadline = 35, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, next_waypoint = forward, action = forward, reward= -1.0

The state `oncoming` is also crucial in the modeling. For the case where the smartcab were to turn left at a green light, but there is an oncoming vehicle that is moving forward or about to turn right. This is considered a bad move, and will be assigned a penalty of -1 as shown in the example below:

> LearningAgent.update(): deadline = 30, inputs = {'light': 'green', 'oncoming': 'right', 'right': None, 'left': None}, next_waypoint = left, action = left, reward= -1.0

Apart from that, the state `left` also encapsulates important information about the smartcab and the environment. The smartcab can also turn right when the light is red provided that the vehicle from the left intersection is not moving forward. Hence the smartcab can also be penalized by a score of -1, if it tries to turn right when the vehicle from the left is moving forward. An example is as shown below:

> LearningAgent.update(): deadline = 16, inputs = {'light': 'red', 'oncoming': 'right', 'right': None, 'left': forward}, next_waypoint = right, action = right, reward= -1.0

The next waypoint is also critical in the modeling. Given that all the conditions mentioned above is not violated, if the action taken coincides with the next_waypoint, then the smartcab will be rewarded a score of 2.0, like in the example below:

> LearningAgent.update(): deadline = 20, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, next_waypoint = forward, action = forward, reward = 2.0

Question: *How many states in total exist for the* ***smartcab*** *in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?*

Answer:

Apart from the four states above, there are two more states that are not included in the modeling. They are namely,

1. Right: None, forward, left, or right
2. Deadline: Distance between start and destination × 5. Maximum Manhattan distance being 12 from (1,1) to (8,6), hence the maximum deadline is 60.

The state `right` is not included in the modeling because it plays no role in determining the reward and penalty according to the right-a-way traffic rules. There are two special cases of right-a-way traffic rules, i.e. the one that does not allow left turn at green light if there is oncoming traffic moving forward or turning right, and the one that allows right turn at red light if there is no vehicle moving forward from the left intersection. Those cases only involve traffic conditions for oncoming intersection and left intersection. Hence adding the state for right intersection in the modeling will not be useful.

On the other hand, the state `deadline` is also excluded due to its high number of possible state, i.e. 60. If all six states were included, the total number of combinations for all states in this environment is

$$2 \times 4 \times 4 \times 4 \times 4 \times 60 = 30{,}720$$

Thus, there will be a high quantity of states to be learnt during Q-Learning, therefore more iterations are required for the model to be fully learnt.

However, if the state is reduced to the four states mentioned in the previous section, the total number of combinations for all states is reduced to

$$2 \times 4 \times 4 \times 4 = 128$$

Hence, the model can be fully learnt in lesser number of iterations.

On a side note, adding the state `deadline` would be a good idea if we are training an agent that is aggressive and willing to break some rules in order to reach the destination in time. Since, for states where the deadline is short, taking action that leads to the destination will be favored as the reward is bigger than the penalty. This is not the behavior we would want for the smartcab as safety during the trip is also very important. Hence, the exclusion of the state `deadline` seems to be a good choice here.

Implement a Q-Learning Driving Agent

With your driving agent being capable of interpreting the input information and having a mapping of environmental states, your next task is to implement the Q-Learning algorithm for your driving agent to choose the *best* action at each time step, based on the Q-values for the current state and action. Each action taken by the **smartcab** will produce a reward which depends on the state of the environment. The Q-Learning driving agent will need to consider these rewards when updating the Q-values. Once implemented, set the simulation deadline enforcement `enforce_deadline` to `True`. Run the simulation and observe how the **smartcab** moves about the environment in each trial.

Question: *What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken? Why is this behavior occurring?*

Answer:

After the implementation of Q-Learning, the success rate of the agent reaching the destination in time seems to be improving as the number of iteration increases.

Initially, the agent may fail to reach the destination and may take some wrong moves along the way, for the first few iterations. But over time, the number of wrong moves made by the agent decreases. We see the agent either not taking any action and receives zero reward (i.e. waiting for its turn at the intersection), or taking the right action and receives positive reward.

This behavior occurs because as we go through more and more iterations with Q-Learning, the Q values are updated for each state-and-action pair, which reflect the utility / long term reward. When the agent is at a particular state, it chooses the action with the highest Q value. As the number of iterations increases, the Q values for all state-and-action pairs eventually converge. Hence the action chosen in this way corresponds to the optimal policy given the rewards and penalties defined in this environment.

Improve the Q-Learning Driving Agent

Your final task for this project is to enhance your driving agent so that, after sufficient training, the **smartcab** is able to reach the destination within the allotted time safely and efficiently. Parameters in the Q-Learning algorithm, such as the learning rate (alpha), the discount factor (gamma) and the exploration rate (epsilon) all contribute to the driving agent's ability to learn the best action for each state. To improve on the success of your **smartcab**:

- Set the number of trials, n_trials, in the simulation to 100.
- Run the simulation with the deadline enforcement enforce_deadline set to True (you will need to reduce the update delay update_delay and set the display to False).
- Observe the driving agent's learning and **smartcab's** success rate, particularly during the later trials.
- Adjust one or several of the above parameters and iterate this process.

This task is complete once you have arrived at what you determine is the best combination of parameters required for your driving agent to learn successfully.

Question: *Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best? How well does the final driving agent perform?*

Answer:

The different sets of parameters considered for the fine tuning are as listed below:

1. Learning rate, alpha: 0.1, 0.5, 0.9
2. Discount rate, gamma: 0.1, 0.5, 0.9
3. Exploration rate, epsilon: 0.1

We would like to explore the effect of different combinations of learning rate and discount rate, while keeping the exploration rate constant. The exploration rate is kept at a low value of 0.1 as we would like the agent to explore new actions and states a small portion of the time.

The result of the parameter grid search with success rate for reaching the destination in time as the optimization criteria is as shown below:

Learning Rate (Alpha)	Discount Rate (Gamma)	Exploration Rate (Epsilon)	Success Rate
0.1	0.1	0.1	1.00
0.1	0.5	0.1	0.89
0.1	0.9	0.1	0.93
0.5	0.1	0.1	0.99
0.5	0.5	0.1	0.88
0.5	0.9	0.1	0.75
0.9	0.1	0.1	0.98
0.9	0.5	0.1	0.93
0.9	0.9	0.1	0.58

Table 1 – Parameter grid search for Q-Learning

The agent with the parameters alpha = 0.1, gamma = 0.1, has the highest success rate for reaching the destination in time, i.e. 100% out of 100 iterations. In contrast, the agent with high alpha and gamma value of both 0.9 performs poorly and has a success rate of only 58% out of 100 iterations.

Question: *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? How would you describe an optimal policy for this problem?*

Answer:

An optimal policy for this problem would be as listed below:

1. If moving forward or turning left at the intersection, go when light is green and stop when light is red.
2. If turning right at the intersection, go when light is green.
3. If turning right at the intersection when light is red, go when vehicle from left of the intersection is not moving forward, and stop otherwise.
4. If turning left at the intersection when light is green, stop when there is oncoming vehicle moving forward or turning right.
5. Not to move outer the boundaries of the environment.
6. Take action that is consistent with the next waypoint.

The final agent tuned with parameters alpha =0.1, gamma = 0.1, and epsilon = 0.1 is getting close to the optimal policy listed above. At later trials of the 100 iterations, it managed to take the correct actions most of the time. However, we observed that some time along the way, the agent takes some incorrect actions which lead to penalties. This is due to the exploration rate of 0.1 which causes the agent to take random actions instead of optimal ones based on Q-Learning.

Below are some of the examples that the agent takes non-optimal actions during different trials:

Trial	Deadline	Input	Next waypoint	Action	Reward
93	32	{'light': 'red', 'oncoming': None, 'right': None, 'left': None}	forward	left	-1.0
94	16	{'light': 'green', 'oncoming': None, 'right': None, 'left': None}	forward	right	-0.5
96	19	{'light': 'red', 'oncoming': None, 'right': None, 'left': None}	forward	left	-1.0
98	18	{'light': 'green', 'oncoming': None, 'right': None, 'left': None}	forward	left	-0.5