Tasks

Implement a Basic Driving Agent:

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: The Driving Agent's behavior as it takes random action within the simulation is that it does not care whether the light is red or green, whether there is an oncoming vehicle. It randomly goes "right, left, forward or none" without regards to the target location. It does not arrive at the target location. Every reward is equal to float(zero). The agent does not seem to reach the destination. Here is a sample after over hundred iterations as follows:

```
LearningAgent.update(): deadline = -100, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = None, reward = 0.0

Environment.step(): Primary agent hit hard time limit (-100)! Trial aborted.
```

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 I have identified are:

inputs[‘light’], inputs[‘left’], inputs[‘oncoming’], inputs[‘right’], self.next_waypoint. The state helps the agent to find out about the different aspects of its environment, to determine when it is safe to move or not, based on U.S traffic rules, as to whether the light is green or red, whether there is an oncoming vehicle or not, whether there is a vehicle on the left or right side trying to cross the agent’s direction or turn left or right, and based on the destination, which direction should the agent follow. Taking all these aspects into account make the agent more aware of its environment as to decide whether to navigate safely or not. Therefore, the state is a key factor of the agent movements in terms of reaching its destination safely with minimum penalties.

OPTIONAL: 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:

The states to consider are the following:

1. Light: Red or Green, (2)

2. Oncoming: forward, left, right, or None, (4)

3. Left: forward, left, right, or None, (4)

4. Next waypoint: forward, left, or right, (3)

With the above five folded states, the state-space in the q-learning is the following size: 2*4*4*3 = 96

Yes, this number is reasonable because if you follow the U.S right-of-way rules, it does not matter much whether there is a vehicle at your right unless it is a stop and go intersection or you are in the middle of the intersection trying to turn left while light has turned red after yielding for oncoming vehicle. But with this environment, the light turns either red on green, no other stop signs or blinking red light for

stop and go situation. Therefore, omitting the state “right “reduces the state size and the process speed without considerable impact.

If I were to add the state Right: forward, left, or right, or None, the state size would have been:

2*4*4*3*4 = 386 instead of 96.

Similarly, if I were to include the deadline=self.env.get_deadline(self), that could increased the size as well.

Implement a Q-Learning Driving Agent:

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: The agent follows the U.S right-of-way and traffic rules. It also reaches its destination within the deadline. It receives more positive rewards and very few negative rewards. This is because I implemented the agent's Q-Learning to exploit, to choose best actions which are the actions of the maximum Q-Learning table values 99% of the time and to exploit, to choose the other actions about 1% of the time randomly and the exploitation keep decreasing at a rate of 1/t as the agent increasingly gains knowledge of its environment.

Improve the Q-Learning Driving Agent

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?

First, to measure success when comparing different parameters, I thought that considering average reward over total moves, average reward / total moves might be a good metric to use. For example, let’s assume that there exist two sets of 100 trials A and B. If both sets have the same number of moves, and if the ratio is higher for A, than B, then A must be incurring more rewards than B. Likewise, if A and B got the same total reward but if B took more moves, then A is more efficient than B. This metric is also great because when the learning agent makes mistakes, the environment prevents it from moving forward, which means that the number of actions it takes increases without getting closer to the goal. Thus, when it makes a lot of mistakes, the average reward / total moves, average reward / total moves ratio (RPM) will be lower than when it does not make a lot of mistakes, assuming everything else equal.

By testing different learning rate and discount factor values, and calculating the average reward / total moves metric for each combination. I set the possible values for the learning rate and alpha as follows: alpha values [0.1,0.3,0.5,0.7,0.9], gamma values [0.1,0.3,0.5,0.7,0.9]. Thus, there are 25 combinations total. First, I ran the 100 trials for each of the 25 combination. The sample is in pdf file called: fifteen_alpha_gamma_sets

It seems that the best learning rate to discount factor pair is when learning rate is 0.9 and discount factor is 0.5, while the worst one is when the learning rate is 0.9 and the discount factor was 0.3, resulting in RPM of 2.4462.446 and 2.1492.149 respectively.

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?

The agent gets close to finding an optimal policy by using Q-Learning values and choosing the actions that not only lead to the targeted destination on time but also to incur no penalties, but the agent also, randomly explores other actions that may lead it to incur penalties, getting close to finding optimal policy is observed when the agent is almost done exploring other actions, in another word when the epsilon is almost zero the agent eventually stops exploring and continue exploiting. The exploitation phase is the phase that lead the agent to finding an optimal policy.