Smartcab based on Q-learning Alogrithm

Tao Chen

September 2016

1 Implement a Basic Driving Agent

Question 1:
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:

Give longer time, the smartcab (the red cab) can have higher probability to reach the destination eventually. Sometimes, it hit the hard time limit before reaching the destination. If the deadline is set up, the probability of failure to reach the destination is higher than the case when there is no deadline. Since it proceeded in a random action at each intersection, it makes sense that it would take much longer time to reach the destination. Basically, the smartcab just randomly move across the whole streets, and it ran into the destination just by chance since no policy or strategy was learned till now. When there was no deadline, the smartcab reached the destination before hitting the hard time limit 58 times out of 100 trials (58%). When there was a deadline on each trial, the smartcab only succeeded 28 times out of 100 trials (28%).

2 Inform the Driving Agent

Question 2:
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:

I choose all the keys of the Environment.valid_inputs plus the self.next_waypoint as the states. That is to say, states are composed of 'light', 'oncoming', 'left', 'right', 'next_waypoint'. I chose all these states because I think that the each variable here will affect how the cab behave. For example, even though the 'light', 'left', 'right', 'next_waypoint' are same under two cases, the cab need to take different actions when there is an 'oncoming' car or there is no 'oncoming' car. The same analysis can be applied to 'light', 'left', 'right','next_waypoint'.

So I think it would be better to include all these variables to account for a state. And since “light” can have 2 possible values (’red’ and ’green’), and “oncoming” , “left”, “right” , ’next_waypoint’ all have 4 possible values (’None’, ’forward’, ’left’, ’right’), the number of total possible states is $2 \times 4 \times 4 \times 4 \times 4 = 512$. There is also another variable called ’deadline’ here which I do not include to form a state. The reason is that including deadline will dramatically increase the state space. Because ’deadline’ variable can take many possible values (in order of 10), which will enlarge the volume of the state space by an order of magnitude. Of course, by including ’deadline’, the agent will perform better because it will learn to find the shortest way given the time limit. So it is a balance between the agent’s performance and the volume of the state space. Since the agent worked well without including ’deadline’, I did not further include ’deadline’.

3 Implement a Q-Learning Driving Agent

Question 3:
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 Q-learning algorithm was implemented, the agent quickly learned a better strategy to reach the destination point. There are several parameters that need to be tuned for Q-learning, including the epsilon value, the initial q table values, the gamma value(discount factor), the learning rate. At this point, I just chose a small epsilon (0.20) to avoid explore the new path a lot. And the learning rate was set to 0.80, gamma = 0.10, initial q table values are 0. With these parameters being set, the agent succeeded 92 times out of 100 trials. The agent ran out of time in the rest 8 trials. It is a great improvement (from 28% to 92%) compared to the basic driving agent where no Q-learning algorithm was implement. Basically, what I implemented in the program can be summarized as follows:

1. set the learning rate, initial q values, gamma values, epsilon
2. get the current state
3. choose the best action with the largest q value under this state
4. perform the action and get the reward value
5. update the q table
6. repeat this process until the agent reach the destination or ran out of time

And a good strategy should make the agent obey the traffic rules, follow the suggestions of the planner, reach the destination far ahead the deadline.

4 Improve the Q-Learning Driving Agent

Question 4:

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 Table 1 shows the group of parameters I tuned for the agent. From the table, we can see that the success rate is most sensitive to the epsilon value. The learning rate and gamma won't affect the agent's behavior as much as the epsilon value do. Actually I cannot say which set of parameters gives the agent best performance since even if all the parameters are set, the running results fluctuate a little bit each time I ran the program since there is still some randomness. And as we can see from the table, most sets of parameters give very close success rate. Therefore, the agent has a wide range of choices as for these parameters. If one has to make a single choice, I would recommend that $\epsilon = 0.05, learning\_rate = 0.90, \gamma = 0.10$

Table 1: Parameter Tuning With 500 Trials each

Epsilon	Learning Rate	Gamma	Success Rate
0.05	0.80	0.10	98.60%
0.10	0.80	0.10	97.60%
0.15	0.80	0.10	96.00%
0.20	0.80	0.10	94.60%
0.25	0.80	0.10	90.20%
0.30	0.80	0.10	85.60%
0.35	0.80	0.10	83.40%
0.40	0.80	0.10	82.60%
0.45	0.80	0.10	73.60%
0.60	0.80	0.10	58.00%
1.00	0.80	0.10	17.80%
0.05	0.45	0.10	99.40%
0.05	0.50	0.10	98.60%
0.05	0.55	0.10	97.60%
0.05	0.60	0.10	98.00%
0.05	0.65	0.10	99.00%
0.05	0.70	0.10	98.60%
0.05	0.75	0.10	99.60%
0.05	0.80	0.10	98.60%
0.05	0.85	0.10	98.60%
0.05	0.90	0.10	99.80%
0.05	0.95	0.10	99.20%
0.05	1.00	0.10	98.60%
0.05	0.90	0.05	98.20%
0.05	0.90	0.10	99.20%

0.05	0.90	0.15	98.40%
0.05	0.90	0.20	99.00%
0.05	0.90	0.25	98.00%
0.05	0.90	0.30	98.20%
0.05	0.90	0.35	98.60%
0.05	0.90	0.40	99.60%
0.05	0.90	0.45	99.40%
0.05	0.90	0.50	98.60%
0.05	0.90	0.60	98.20%
0.05	0.90	0.65	98.40%
0.05	0.90	0.70	98.60%
0.05	0.90	0.75	97.80%
0.05	0.90	0.80	98.20%
0.05	0.90	0.85	98.20%
0.05	0.90	0.90	99.20%
0.05	0.90	0.95	99.20%
0.05	0.90	1.00	98.80%

Question 5:

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: My agent does get very close to finding an optimal policy. At the first few trials, the agent might some undesireable behaviors like circling around a square or go furthur away from the destination. But such behaviors soom disappered as the agent updated the Q matrix. And I ran 100 trials, and got about 80 penalty moves and 1400 total moves. And the penalty moves only occur in the first few trials.So as the agent took more trials, the probability that the agent would get negative rewards got much lower. An optimal policy should have a small amount of total penalty moves in 100 trials. Also it will make the agent choose a way that takes minimum number of move steps. Even better, it should reduce the waiting time (e.g. the agent takes the 'None' action) so that the agent can reach the destination as soon as possible.