Train a Smartcab to Drive

Yiying Wang

July, 2016

Task 1

Implement a basic driving agent

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

The agent randomly chooses an action from "stay put", "turn left", "turn right" or "go forward" at each time point. The agent's action most of the time is not the same with the one that the planner told it (as shown right next to the car). When the agent's action is randomly chosen to be "**forward**", it knows to wait for the traffic light to turn green before it makes a move (this new move changes, it is not necessarily the same as before). When the agent's action is randomly chosen to be "**left**", if the light is red, it knows to not make a move and choose again if there is another car coming straight from forward. If the light is green and there is either no cars coming from front or the oncoming car is about to take a left, agent takes a left. When the agent's action is "**right**", if the light is green, it will turn right right away. If the light is red, it will look at its left, if there is no car on its left or if the car on its left doesn't want to go straight, it will turn right.

After a lot of driving around, wandering back and forth, the agent can eventually reach the destination after a lot of steps (much larger than the required time frame).

Task 2

Identify and update sate

Identify a set of states that are appropriate for modeling the driving agent. Justify why you picked these set of states, and how they model the agent and its environment.

I pick the set of states as 6 variables and put them in a namedtuple data structure.

1) Relative_Location: this is the relative location of agent. It is represented by the sign of (Destination[0] – Location[0]) and (Destination[1] – Location[1]). It has 9 possible values:
    a. (0, 1), destination is on the south relative to the agent,
    b. (0, -1), destination is on the north relative to the agent,
    c. (0, 0), agent reaches the destination,
    d. (-1, 1), destination is on the southwest relative to the agent,
    e. (-1, 0), destination is on the west relative to the agent,
    f. (-1, -1), destination is on the northwest relative to the agent,
    g. (1, 1), destination is on the southeast relative to the agent,
    h. (1, 0), destination is on the east relative to the agent,
    i. (1, -1), destination is on the northeast relative to the agent.
2) Heading: this is the direction of the agent is facing. It has 4 possible values:
    a. (1, 0) east
    b. (0, -1) north
    c. (-1, 0) west
    d. (0, 1) south
3) Light: red or green. This is the traffic light for agent. It has 2 possible values.
4) Oncoming: this is to indicate if there is another car coming from the front at the same intersection with the agent. If the value is "None", it means there is no car coming from front. Otherwise, its value indicates the direction the other car is going, e.g. "left", "right", "forward". It has 3 possible values.

5) Left: this is to indicate if there is another car coming from the left at the same intersection with the agent. It has 3 possible values.
6) Right: this is to indicate if there is another car coming from the right at the same intersection with the agent. It has 3 possible values.

There are 1944 distinct states. And there are 4 possible actions ("None", "forward", "left", "right"). Thus, the Q-table would have a size at most 7776 elements.

I picked these variables because they are most relevant for agent to make the optimal decisions. **Relative_Location** is a general direction to which the agent should move so that it can get closer to the destination. For example, if the destination is on the southeast of the agent, the agent should be able to know that it can move either to the south or to the east, based on other conditions. The **heading** of the agent is important because, firstly, it determines which cars are on its left, right or front, and the next moves of those cars would potentially impact how the agent reacts; secondly, the direction which the agent is facing also impacts the next move even if the agent is going to move between two same points. **The rest variables** are the information for the surroundings. The light status is important because the agent needs to obey the traffic rules and not move or yield to traffic when the light is red. The information of other cars that are at the same intersection is important because our agent need to know if it should yield to other traffics. If it runs into other cars, it should get a big penalty so that it can learn that he should take none action in that case.

Task3

Implement Q-learning

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

At the beginning of the run, the agent was aimless and making many mistakes. It was not able to reach the destination within in the required time. But as it learns, it made less and less wrong choices and was able to reach the target in time.

Task4

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, (reach the destination in the minimum possible time, and not incur any penalties)?

I used learning rate of 1, discount of 0.8. I changed the reward policy in "act" function in "environment.py": instant reward = 5.0 if the new location gets closer to the destination (distance decreases); instant reward = -2 if the new location gets further to the destination (distance increases); instant reward = 0 if there is no action taken; instant reward = -3 if the agent runs into other cars (the agent moves when it shouldn't have); instant reward = 20 if the agent reaches the destination. I set 10 dummy agents to run on the road.

After the learning starts, the Q table grows quickly and almost reaches convergence after less than 100 trials. After a decent number of runs, the Q table has 1832 elements (out of size of 7776), and the agent is able to reach the destination in time with minimal or none penalties. Based on the Q-table size, the agent hasn't met all the possible (state, action) pairs, this might because the agent is able to make the right choice for the majority of the states and the algorithm is converged.