

Smart Cab Report

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?

When take action in a random manner, the agent basically just move around randomly. It could sometimes make to the target location, but only because of coincidence.

Identify and update state

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

I chose "next_waypoint" and "light" as states. The agent has two goals: 1. follow the traffic rules; 2. get to the target location before deadline.

In order to achieve the first goal, information need to be given are: traffic lights, other cars in the intersection. The "inputs" variables in the agent.update() has all these information. However, the reward system in the enviroment.act() doesn't take the traffic collision into account . So I also ignored traffic collision part for the agent for now.

For the second goal, although the agent has access to the deadline variable, I don't think it is useful to achieve the goal. If the agent learned to follow the "next_waypoint", it can always arrive at the target location within the time limit.

Above all, I combined "next_waypoint" and "light" as states to let the agent learn to follow the waypoint meanwhile obey the traffic light. If the reward system considers traffic collision. "oncoming", "left", "right" should be added to the state variable.

Implement Q-Learning

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

The learning rate of the Q-learning is set to 1 because it is a deterministic environment here. The discount factor is set to 0.2 for the basic Q-learning.

One of the problem in implementing Q-learning is the Q value for last action (the action get to the destination). The following state will be arrival the destination and end of the trail. So for the last action, the Q value is updated using the reward directly.

After implementing Q-learning and only select action with the highest Q-value, the agent starts to learn the two goals. It learns the traffic lights very quickly. Because it achieves the negative reward if it violates the traffic lights. However, it cannot always learn to get to the target within time. The agent can easily get stuck on the road. The reason is it can get positive reward even it doesn't follow the next_waypoint but obey the traffic rules or it just doesn't move. When I initialize all Q value as 0, the action will become positive as long as the agent obey the traffic rules or not move. So if the agent first entered one state and

randomly picked a None or other action which follows the traffic rules but not the next_waypoint, this action will have the biggest value in this state and the agent will never learn to follow the next_waypoint in that state.

This problem can be solved in many ways, includes using different initial Q values. But I chose to add some random selection in the process to add some exploration. It will be discussed in the next part.

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?

As I mentioned above, I added a random rate when select the actions. So when the agent enters a state, it has 0.8 probability that it will follow the Q value to select action and the other 0.2 probability it will select action randomly. After training, the agent will test without random selection, so it will always follow the trained policy, because we don't need exploration when the agent has been trained. In the testing process, the Q-table will not be updated. With the original parameter settings (learning rate=1, gamma=0.2), the random selection process works very well that the agent will find the optimal policy eventually.

However, how long it takes to find the optimal policy depends on the simulation. Sometimes, the agent will always reach the destination without any penalties even after few iterations. This means the agent was able to enter and update every states and actions in the Q-table in the training process.

One of the thing need to be noticed is that if we set the discount factor to 0, the Q values will be the reward and it will actually lead to the optimal policy. So if we set the learning rate =1 and discount factor=0, after the agent entering all states and actions, the training is complete.

In order to test effects of different discount factor. The agent was then trained with 50 trails with discount factor=[0.1,0.2...0.9] and with the random rate active (random_rate=0.2). Then, each agent is tested with 50 trials and without the random rate. The count of successful arrival and the number of violating the traffic lights are reported for each agent.

γ \ times	1	2	3	4	5
0.1	(50, 0)	(49, 0)	(50, 0)	(50, 0)	(50, 0)
0.2	(50, 0)	(49, 0)	(50, 0)	(50, 0)	(47, 0)
0.3	(49, 0)	(50, 0)	(20, 0)	(50, 0)	(50, 0)
0.4	(50, 0)	(50, 0)	(50, 0)	(5, 0)	(50, 0)
0.5	(50, 0)	(50, 0)	(2, 0)	(50, 0)	(50, 0)
0.6	(2, 0)	(40, 0)	(47, 0)	(7, 424)	(38, 0)
0.7	(22, 0)	(45, 0)	(39, 0)	(0, 0)	(28, 0)
0.8	(6, 0)	(5, 796)	(1, 27)	(20, 0)	(27, 0)
0.9	(25, 246)	(27, 298)	(21, 0)	(11, 0)	(2, 648)

*(number of successful arrival, number of violating traffic lights)

The table shows that when the discount factor becomes larger, the agent becomes non-stable. It is consistent with the above discussion that the agent should be more emphasis on the current reward instead of long term utility. So for the final program, I choose the discount factor as 0.