SmartCab project report

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: It seems, the agent could not reach to the given destination within a deadline by taking random actions. But rarely, it could. As observed, it could reach the destination usually at the beginning of simulation.

The reward was between -0.5 and 2.0, it's like

- 0.0 reward for the 'None' action,
- 2.0 reward for valid move
- -0.5 reward for valid but other direction

The smartcab was jumping from one edge to another, it was very annoying and weird.

The traffic light stays about 3 seconds (updates) at one state. So detouring time over the traffic light (at least 3 moves) was almost same as waiting time at the traffic light.

Inform the Driving Agent

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:

This is a navigation problem with a deadline. The mission is to reach a given destination with a given time in a given environment (grid space).

The current location of the agent is the most important metric to complete the task. But in this problem the agent doesn't know it's location and destination. So the agent should follow the waypoint, it may be punished if it goes in other direction.

So I think, the following states are appropriate (ordered by importance):

- `Next waypoint` - The next waypoint location relative to its current location and heading. Since current location and distance are unknown for the agent, the waypoint should be the most important information to navigate in the environment. Just by following the 'next_waypoint', the agent could reach the destination successfully, except traffic light rule. But the agent should learn other optimal moves while following the waypoint.

  So I created a implicit state, which is a mix of last action and last waypoint. I think, this mixed state is appropriate for the problem. Because a mixed state will be better for learning than a single waypoint state.

- `Deadline` - The current time left from the allotted deadline. The agent has to select a correct action according to deadline. It represents how many moves the agent has left, at maximum. The deadline must be enough value for the task, otherwise it would be a mission impossible. So we may skip this information and follow the waypoint.

- `Light` - traffic lights: red, green

- `Oncoming cars` - oncoming vehicles from other directions

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:

I think, in total, there are 4 usefull states. The 'Next waypoint' and 'Deadline' states are the most important for this task. But, we could ignore 'Deadline', 'Light', 'Oncoming car' states, because we could maintain it by traffic policy.

We need to keep state combinations as few as possible, because many states (combinations) will take longer time to "learn".

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?

Success rate is increased. Because it's learning from the past history.

I used some code snippet from this blog: https://studywolf.wordpress.com/2012/11/25/reinforcement-learning-q-learning-and-exploration/

It was very well explained and easy to understand. And saved a lot time to implement Q-Learning part.

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?

ANSWER

I tuned the parameters as below:

- alpha = 0.5 # learning rate
- gamma = 0.9 # discount
- epsilon = 0.7 # exploration. The epsilon increases when the number of iteration increased. Then, the agent selects the greedy action with probability of this epsilon.

At beginning of simulation, the agent makes a lot round movement and takes long stop at intersections. Over time these mistakes lowered and success rate is increased.

The success rate was greater than 90% for 100 simulation run.

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

I described the optimal policy as the following. And the goal is to maximize an average reward.

```
Average reward = Total_reward / Total_time_or_move
```

After several simulation my agent's average reward was more than 1.32. I don't know if it's optimal or not, but based on reward values the average reward could be 2.0 at maximum. In other words, the agent's every movement is correct and perfect, and get 2.0 reward on every movement. But it may be not ideal.

By observation, the agent was very close to an optimal policy.