Smartcab – Reinforcement learning

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?

When the deadline is active the smartcab has reached its destination in 194 of 1000 tries, which is about 20%. In 806 tries it didn´t reach its destination, because the deadline was exceeded.

When turning off the enforce deadline parameter the agent reached its destination more often (as expected) but also does hit the hard time limit sometimes. So in 674 tries it reached the desitnation and in 326 it didn´t.

This test was done by turning off the visuals and setting the trials to 1000, while counting the occurrences of successful tries in the log.

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?

I would consider using all of the inputs for the Q-Learning problem. As our smartcab is mostly legally correct, we don´t want to drive when lights are red. And furthermore we don´t want to crash into another car. That's why these actions are rewarded negative.

- next_waypoint, the waypoint suggested by the planner is needed because without this waypoint the algorithm is not able to determine the right direction to go. There is no way to determine where to drive, when we only have the traffic situation, but no idea of where the destination is.
- light, color of the traffic light
- oncoming, potential oncoming car driving to a specific direction
- left, potential car from left driving to a specific direction
- right, potential car from right driving to a specific direction

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?

We have the light variable (red and green) -> 2
We have all the other variables, which can always be None and one of 3 directions -> 4^4
So we have 2 * 4^4 -> 512 states

I think it is reasonable, because we need to manage 512 entries in a matrix, which is ok. If we would use the deadline, we would increase the complexity and add a dynamic value, which in the end does not improve the basic algorithm. unreal

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?

After implementing Q Learning there is a massive improvement of the success rate. In the first few tries the smartcab does also “trial and error” but improves very fast.

So in the first epoch the smartcab takes the trial and error stage, because the QTable is not filled with different values, so a random action is chosen. After very few trys the QTable is updated with the very common states (no traffic, red light + no traffic, green light) and learns that it is a good idea to follow the suggested waypoint, as the planner does show the direction to the destination. After that the smartcab reaches the destination in the most cases. But there is one thing to improve the, the agent could avoid red light waiting time. This behavior is more complex, and it takes more time to improve, because these states are more uncommon. When the smartcab follows the next_waypoint it’s a local minimum. Also the smartcab does behave legal.

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?

Testing with 1000 trials, because I think 100 are too few, when dealing with this small differences.

Learning rate	Discount	Exploration	Successrate
0.2	0.8	0	67.4%
0.2	0.8	0.2	75.9%
0.1	0.4	0.1	97.4%
0.1	0.1	0.1	96.4%
0.3	0.2	0	99.7%
0.5	0.01	0	99.9%

The final driving agent does almost succeed every try. I think the reason why the exploration is not necessary comes from the point, that there are some steps are very common and others rarely occur. In this case we don´t need to much exploration.

My first best results, where achieved with 0.55 and no discount, but as I did a bit research I found, that other students found the 0.5 / 0.01 parameters are the best results, and I tried them and it was a further improvement. The reason why exploration is not very promising, is simply that the accuracy is very close to 100% (the next_waypoint feature is very strong), every exploration lowers this score. Having 100% accuracy with 5% exploration will end up in something around 95% accuracy. A small discount on the other hand improves short term learning, so in our case it does very well.

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?

Currently the agent is optimized to reach the destination, which works very good (almost 100%). In order to get the best speed, I think it would be an idea adjust the rewards. While the agent still waits on the intersection until the traffic light is green, it could be an option to insert different rules to allow the smartcab to turn right, even if the light is red, to get faster to its destination. This would also be more appropriate to machine learning, as there is more uncertainty in the model. The question if this would be the optimal policy depends on the goal of the application. It could also be possible to ignore lights and vice versa or don´t drive directly and to the destination (to make some more money with the passenger). It depends on how the optimal policy is defined. In this case the optimal policy is to reach the destination and the agent does so.

Here is an example of the first try and the last one:

```
RoutePlanner.route_to(): destination = (6, 1)
state= 0, reward = 0, action = None, QValue = 0.0
state= 76, reward = 2.0, action = forward, QValue = 1.0
state= 320, reward = -0.5, action = right, QValue = -0.25
state= 384, reward = -0.5, action = right, QValue = -0.25
state= 192, reward = 0.0, action = None, QValue = 0.0
state= 192, reward = 2.0, action = right, QValue = 1.0
state= 192, reward = -0.5, action = left, QValue = -0.25
state= 320, reward = -1.0, action = forward, QValue = -0.5
state= 320, reward = -1.0, action = forward, QValue = -0.75
state= 320, reward = -1.0, action = left, QValue = -0.5
state= 320, reward = -1.0, action = left, QValue = -0.75
state= 64, reward = 2.0, action = forward, QValue = 1.0
state= 320, reward = 0.0, action = None, QValue = 0.005
state= 64, reward = 2.0, action = forward, QValue = 1.500025
state= 320, reward = 0.0, action = None, QValue = 0.002525
state= 320, reward = 0.0, action = None, QValue = 0.008762625
state= 64, reward = 2.0, action = forward, QValue = 1.757512625
state= 64, reward = 2.0, action = forward, QValue = 1.88754387563
state= 64, reward = 2.0, action = forward, QValue = 1.94377193781
state= 384, reward = -1.0, action = forward, QValue = -0.5
state= 384, reward = -1.0, action = forward, QValue = -0.75
state= 384, reward = -1.0, action = left, QValue = -0.5
state= 128, reward = 2.0, action = left, QValue = 1.00004381313
state= 320, reward = 0.0, action = None, QValue = 0.004425125625
state= 320, reward = 0.0, action = None, QValue = 0.0119314225016
state= 64, reward = 2.0, action = forward, QValue = 1.97194562602
Environment.step(): Primary agent ran out of time! Trial aborted.

RoutePlanner.route_to(): destination = (3, 3)
state= 0, reward = 12.0, action = None, QValue = 12.0086433837
state= 384, reward = 0.0, action = None, QValue = 0.00798104507454
state= 384, reward = 0.0, action = None, QValue = 0.00403042776264
state= 384, reward = 0.0, action = None, QValue = 0.0120837502803
state= 128, reward = 2.0, action = left, QValue = 2.00694223686
state= 320, reward = 0.0, action = None, QValue = 0.0189351684383
state= 64, reward = 2.0, action = forward, QValue = 2.01762261767
state= 64, reward = 2.0, action = forward, QValue = 2.01889942192
state= 64, reward = 2.0, action = forward, QValue = 2.01953766481
Environment.act(): Primary agent has reached destination!
```