Train a Smart Cab to Drive with Reinforcement Learning

George Liu

Summary

A smart cab is a self-driving car from the not-so-distant future that ferries people from one arbitrary location to another. In this project, we will use reinforcement learning to train a smart cab drive. In particular, the Q-learning algorithm will be implemented in a simulated environment to learn the optimal actions to take for the cab to drive to a certain target location.

The simulated environment is a grid world comprising a primary driving agent (the smart cab) and several dummy agents as traffic. There are traffic lights at all intersections, the primary driving agent needs to learn to drive according to traffic and light situations.

Implement a basic driving agent

We first implement a basic driving agent. Although it does take inputs such as next waypoint location, traffic light and presence of cars, this basic driving agent produces only random moves. After running this agent for several times, we observed that although the agent is able to arrive at the target location all the time, due to the random nature of the moves, the time consumed varies wildly as shown below:

```
Simulator.run(): Trial 0
Environment.reset(): Trial set up with start = (8, 6), destination = (4, 6), deadline = 20
Environment.act(): Primary agent has reached destination!
LearningAgent.update(): deadline = 6, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = forward, reward = 12
Simulator.run(): Trial 1
Environment.reset(): Trial set up with start = (7, 2), destination = (6, 6), deadline = 25
Environment.act(): Primary agent has reached destination!
LearningAgent.update(): deadline = -99, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = forward, reward = 2
Simulator.run(): Trial 2
Environment.reset(): Trial set up with start = (7, 2), destination = (5, 4), deadline = 20
```

Environment.act(): Primary agent has reached destination!

LearningAgent.update(): deadline = -79, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = forward, reward = 2

Simulator.run(): Trial 3

Environment.reset(): Trial set up with start = (4, 6), destination = (7, 5), deadline = 20

Environment.act(): Primary agent has reached destination!

LearningAgent.update(): deadline = -5, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = right, reward = 2

Simulator.run(): Trial 4

Environment.reset(): Trial set up with start = (3, 3), destination = (6, 6), deadline = 30

Environment.act(): Primary agent has reached destination!

LearningAgent.update(): deadline = -23, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = right, reward = 2

Simulator.run(): Trial 5

Environment.reset(): Trial set up with start = (4, 3), destination = (8, 1), deadline = 30

Environment.act(): Primary agent has reached destination!

LearningAgent.update(): deadline = -7, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = forward, reward = 2

Simulator.run(): Trial 6

Environment.reset(): Trial set up with start = (3, 5), destination = (3, 1), deadline = 20

Environment.act(): Primary agent has reached destination!

LearningAgent.update(): deadline = -44, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = right, reward = 0.5

Simulator.run(): Trial 7

Environment.reset(): Trial set up with start = (3, 4), destination = (6, 3), deadline = 20

Environment.act(): Primary agent has reached destination!

LearningAgent.update(): deadline = -67, inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = right, reward = 2

Simulator.run(): Trial 8

Environment.reset(): Trial set up with start = (5, 6), destination = (1, 4), deadline = 30

Environment.act(): Primary agent has reached destination!

LearningAgent.update(): deadline = -237, inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = left, reward = 2

If we enforce the deadline, then the driving agent can hardly ever reach the destination within the prescribed time:

Simulator.run(): Trial 0

Environment.reset(): Trial set up with start = (2, 5), destination = (8, 4), deadline = 35

RoutePlanner.route_to(): destination = (8, 4)

Environment.reset(): Primary agent could not reach destination within deadline!

Simulator.run(): Trial 1

Environment.reset(): Trial set up with start = (5, 4), destination = (1, 6), deadline = 30

RoutePlanner.route_to(): destination = (1, 6)

Environment.reset(): Primary agent could not reach destination within deadline!

Simulator.run(): Trial 2

Environment.reset(): Trial set up with start = (4, 3), destination = (6, 1), deadline = 20

RoutePlanner.route_to(): destination = (6, 1)

Environment.reset(): Primary agent could not reach destination within deadline!

Simulator.run(): Trial 3

Environment.reset(): Trial set up with start = (5, 2), destination = (3, 4), deadline = 20

RoutePlanner.route_to(): destination = (3, 4)

Environment.act(): Primary agent has reached destination!

Simulator.run(): Trial 4

Environment.reset(): Trial set up with start = (1, 1), destination = (3, 4), deadline = 25

RoutePlanner.route_to(): destination = (3, 4)

Environment.reset(): Primary agent could not reach destination within deadline!

Simulator.run(): Trial 5

Environment.reset(): Trial set up with start = (3, 3), destination = (2, 6), deadline = 20

RoutePlanner.route_to(): destination = (2, 6)

Environment.reset(): Primary agent could not reach destination within deadline!

Simulator.run(): Trial 6

Environment.reset(): Trial set up with start = (6, 4), destination = (2, 2), deadline = 30

RoutePlanner.route_to(): destination = (2, 2)

Environment.reset(): Primary agent could not reach destination within deadline!

Simulator.run(): Trial 7

Environment.reset(): Trial set up with start = (6, 3), destination = (2, 2), deadline = 25

RoutePlanner.route_to(): destination = (2, 2)

Environment.reset(): Primary agent could not reach destination within deadline!

Simulator.run(): Trial 8

Environment.reset(): Trial set up with start = (2, 1), destination = (3, 5), deadline = 25

RoutePlanner.route_to(): destination = (3, 5)

Environment.reset(): Primary agent could not reach destination within deadline!

Simulator.run(): Trial 9

Environment.reset(): Trial set up with start = (2, 5), destination = (6, 4), deadline = 25

RoutePlanner.route_to(): destination = (6, 4)

Environment.reset(): Primary agent could not reach destination within deadline!

In short, the behavior of the agent is totally unpredictable and doesn't improve over time, which is expected in the absence of a reinforcement learning algorithm providing directions.

Identify and update state

For environment state choices, we have the following candidates:

- location
- heading (direction of movement)
- traffic light status
- traffic on each of the incoming roadways (oncoming, left and right)
- waypoint given by the planner
- time available

Since the self-driving agent only has an egocentric view of the world, location is not an option. Although time seems to be a plausible choice, it is not necessary as best action taken combined with waypoint given will naturally lead to optimized utility for the agent. Furthermore, time generally is not included in reinforcement learning as a state, because time is already incorporated when we define states as states at certain moments. Heading can be helpful, especially when the next waypoint is one block back, the agent needs to learn to navigate to get back instead of backing up. However, since the waypoint is assigned by the planner, it is assumed that no waypoint is at the opposite direction of heading, so we'll leave out heading as a choice for state. For the traffic status on three roadways, we can drop "right". This is because no matter it is red or green light, the traffic on the right roadway does not affect the agent's decision making.

We are left with traffic light status, traffic on oncoming and left roadways and waypoint given by the planner, which will be the states to use for this project since the smart cab's condition at any point in time can be determined by these factors. The following is a list of all the possible states and corresponding possible values:

- traffic light: red, green
- waypoint: None, forward, left, right
- oncoming: None, forward, left, right
- left: None, forward, left, right

In total, we have $2 \times 4 \times 4 \times 4 = 128$ different states, combined with 4 different actions of None, forward, left, right, we can build a 128×4 Q-value matrix to obtain the policy for the game.

Implement Q-Learning

We then implement the Q-Learning algorithm by initializing and updating the above mentioned Q-value matrix (Q-table) at each time step. Instead of randomly selecting an action, the best action available from the current state based on the Q-table is selected and used by the smart cab.

With this change, the smart driving agent's behavior exhibits a much different pattern than before. First, there is rarely negative reward (penalty) given since the agent is always trying to follow the optimal action. Second, although it varies from trial to trial, overall, the agent's ability to reach the destination has greatly improved – instead of taking several minutes to get to the target location, now a steady percentage of trials can reach target location within the prescribed time. Below is the simulation result:

Simulator.run(): Trial 0

Environment.reset(): Trial set up with start = (4, 1), destination = (6, 3), deadline = 20

RoutePlanner.route_to(): destination = (6, 3)

Environment.act(): Primary agent has reached destination!

Simulator.run(): Trial 1

Environment.reset(): Trial set up with start = (1, 2), destination = (4, 3), deadline = 20

RoutePlanner.route_to(): destination = (4, 3)

Environment.act(): Primary agent has reached destination!

Simulator.run(): Trial 2

Environment.reset(): Trial set up with start = (7, 3), destination = (3, 2), deadline = 25

RoutePlanner.route_to(): destination = (3, 2)

Environment.reset(): Primary agent could not reach destination within deadline!

Simulator.run(): Trial 3

Environment.reset(): Trial set up with start = (1, 4), destination = (3, 2), deadline = 20

RoutePlanner.route_to(): destination = (3, 2)

Environment.reset(): Primary agent could not reach destination within deadline!

Simulator.run(): Trial 4

Environment.reset(): Trial set up with start = (7, 4), destination = (6, 1), deadline = 20

RoutePlanner.route_to(): destination = (6, 1)

Environment.reset(): Primary agent could not reach destination within deadline!

Simulator.run(): Trial 5

Environment.reset(): Trial set up with start = (7, 2), destination = (4, 5), deadline = 30

RoutePlanner.route_to(): destination = (4, 5)

Environment.reset(): Primary agent could not reach destination within deadline!

Simulator.run(): Trial 6

Environment.reset(): Trial set up with start = (7, 6), destination = (8, 2), deadline = 25

RoutePlanner.route_to(): destination = (8, 2)

Environment.reset(): Primary agent could not reach destination within deadline!

Simulator.run(): Trial 7

Environment.reset(): Trial set up with start = (3, 4), destination = (7, 5), deadline = 25

RoutePlanner.route_to(): destination = (7, 5)

Environment.act(): Primary agent has reached destination!

Simulator.run(): Trial 8

Environment.reset(): Trial set up with start = (7, 1), destination = (1, 2), deadline = 35

RoutePlanner.route_to(): destination = (1, 2)

Environment.act(): Primary agent has reached destination!

Simulator.run(): Trial 9

Environment.reset(): Trial set up with start = (4, 1), destination = (8, 1), deadline = 20

RoutePlanner.route_to(): destination = (8, 1)

Environment.reset(): Primary agent could not reach destination within deadline!

Enhance the driving agent

Next, we tune different parameters to improvement the smart driving agent's performance. In particular, the following parameters are optimized:

1. alpha – the learning rate that determines to what extent new information learned will override old information
2. gamma – the discount factor that controls the importance of future rewards
3. epsilon – probability of exploration through random actions

The following table includes the changes made and the corresponding results:

Alpha	**Gamma**	**Epsilon**	**Performance Result (successes out of 100 trials)**
0.9	0.9	0.2, no decay	63/100
0.5	0.5	0.2, no decay	79/100
0.5	0.5	Decayed from 1 to 0	66/100, in about 35 trials, the agent starts to reach destination consistently, time used keeps dropping
0.5	0.5	Decayed from 0.6 to 0	79/100, for final 50 trials, the agent almost always reaches destination
0.5	0.8	Decayed from 0.6 to 0	57/100
0.5	0.6	Decayed from 0.6 to 0	74/100
0.5	0.5	Decayed from 0.5 to 0	82/100, total reward: 3205.5
0.5	0.5	Decayed from 0.2 to 0	90/100, total reward: 3039.5

Initially, we have relatively higher alpha and gamma (both 0.9), and a low epsilon with no decay, i.e., to keep the same rate of random actions across different trials. This parameter setting tells the agent to learn almost only the most recent information, to strive for high long-term reward, and to explore random actions 20% of the time. This configuration immediately gives us a good result – the agent reached destination within allotted time for 63 out of 100 trials.

In the following round of optimization, both alpha and gamma are lowered to 0.5, giving the agent a balanced motivation for learning and reward assessment. This immediately brings the performance up to 79 successful trials.

Next, while keeping both alpha and gamma unchanged, we start experimenting with epsilon decaying. Essentially, we let the agent start by full exploration (taking 100% random actions), and set the agent to take fewer and fewer random actions as we get closer to 100 trials. This gives the agent an opportunity to learn a lot at the beginning and to exploit the learned knowledge at later stages. While decaying from 1 drives down the performance, decaying from 0.6 brings it back to 79 trials.

After that, higher gamma settings are tested. However, neither configuration gives higher performance. Then, different epsilon decay starting points including 0.5 and 0.2 are experimented. With the setting of 0.2 as the decay starting value, the smart agent achieves the highest performance with 90 successful trials! The total reward is 3039.5. I believe this is very close to the optimal policy since the agent exhibits a very "intelligent" driving pattern conforming to the preset traffic rules. Plus, the agent is able to reach the destination consistently after learning. The agent seems to have learned how to drive within as few as 10 trials since no clustering of failed trials happens after that. The last failed trial is 65.

Reference

1. Reinforcement learning on Wikipedia
2. Q-learning on Wikipedia
3. Reinforcement Learning
4. Artificial Intelligence: Foundations of Computational Agents
5. Q-LEARNING AND EXPLORATION
6. Q-Learning Tutorial
7. Optimal epsilon (ϵ-greedy) value
8. How to implement epsilon greedy strategy / policy