Long Nguyen
Udacity Machine Learning
Project: Train a SmartCab to drive
April 23, 2016

Train a SmartCab to drive

Random Strategy

If we use a random strategy for the Learning Agent, the agent (almost) **never** reaches the destination (with constraint distance from starting point to destination >5). This is also intuitive to understand because if you do not have the destination (end) in mind, taking random actions will never get you there.

Assume there is only 1 correct route to the destination (with the right Left, Right, Forward sequence), with minimum distance of 5, taking random action will give the random agent a chance of 0.33 ^ 5 = 0.39% to get there. With distance = 10, the chance of getting to destination is 0.001%. And assume there are 100 correct routes to get to destination, the chance of getting to destination is still only 0.1%.

With the “shortest path in the grid” strategy implemented by the provided RoutePlanner, the cab **always** get to the destination within the deadline. The only information that this agent does not take into account is the wait time for the traffic light. Taking around the traffic light would not help, however, since the wait time of the traffic light is 3 time unit, and going around and come back will take at least 3 time units (assume the other traffic lights are all green, which has only 50% chance for each light). Therefore, the provided “simple route planner” already has an optimal algorithm, which is difficult for the learning agent to beat.

Begin learning: state selection

The following states are defined by the Environment for all agents in the simulation

- Destination: final goal in grid coordinate.
- Deadline: number of time units left.
- Location: current current of the agent in grid coordinate.
- Heading: current direction of the cab.

The following states are defined by the “inputs”:

- Light: “red”, “greed” at current intersection
- Next way points of vehicles (if exist) from opposite (oncoming), left, and right position.

The following information are considered for the state information in the first version of Q-learning algorithm:

- Distance in East-West direction and distance in North-South direction: this is critical to help with to train the agent on taking the appropriate action. This is needed to calculate the next state after taking the action (forward, right, left).
- Current vehicle heading information (direction). This is needed to calculate the next state (location) after taking the action (forward, right, left).

Furthermore standard information like Destination and current Location are stored in the state map of the agent, but do not use for the learning (the delta distance is deduced from destination and current location). Current location by itself should not give input for learning, since it is highly dependent on where the destination is.

Q learning implementation

The first implementation of the Q-learning algorithm produced a rather “dumb” cab. The following parameters are being tweaked, resulting in a smarter cab over time.

- The “wrong” reward system is chosen: i chose to use the “reward” value return by Environment.Act. The reward calculated by Environment simply gives a reward for every successful move, even if we are heading in the wrong direction. We should give a reward when the vehicle is closer to the destination, and subtract when the vehicle is further away.

- Did not “explore” enough in the beginning. Our state map consists of 8 * 5 (grid size) * 3 * 3 (vehicle heading) = 360 possibilities. To converge the Q value, each state should be visited multiple times.

- When a Q value is not found for the current state for all possible actions, we should choose to explore (random action). However, exploration factor should not be set too high because of this simple city grid pattern.

- Discount factor was chosen at the beginning very high (0.8). Low discount factor (0) means we favor short term reward. High discount factor (1) means we favor long term reward. For our simple city grid, we should prefer “short term” reward, since one step closer to the goal is (almost) always the good way to go.

After tweaking the parameters, the cab has become smarter. It can make the right decision most of the time.

In 500 trials, it reaches the target in 477 times. Most of the failures are in “childhood”.

In 100 trials, it reaches the target 72 times. The number of failure since “adulthood” (trial 50) is 9.

When max exploration factor is decreased from 10 to 5%, cab reaches destination within deadline 81 out of 100, with failure since adulthood is 2.

Does it produce optimal result (compared to the shortest path planner)? **Yes, it does** (most of the time). The reason it does not produce a real optimal solution is probably due to higher exploration factor I choose.

An ideal policy for this city grid, as discussed in previous section, would be to implement the simple “shortest path” like the simple route planner. But this would not scale well when more states are involved. Our Q learning algorithm shall adapt to those situation better.

Having something similar to GridSearchCV would help tune all the parameters. But it can take a considerable amount of time.