Long Nguyen
Udacity Machine Learning
Project: Train a SmartCab to drive
April 25 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”, “green” 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:

- Vehicle’s heading information provided by the simple Route Planner. This essentially gives the vehicle the “gps for one step ahead”. The heading information shall give good input for the training algorithm in trade-off made with deviating from the “shortest path” (which is planned by the Route planner.
- The light information at the current intersection. The light information shall have a high impact on training the vehicle to obey traffic law.

Furthermore, the follow information could influence the learning algorithm but chosen not to include in the first version:

- Heading information of other vehicles at the interaction. Shall learn the vehicle to obey traffic light even more. But considering the “sparse” nature of the grid, the chance 2 vehicles are present at the same interaction at the same time is relatively small.

- Current deadline information: shall learn the vehicle to make trade-off of going around versus going to the destination too soon.

However, it is chosen to begin simple and focus on the state variables that have the highest impact for the first version. Also because of the short deadline and learning iteration, it is better to keep the state space small (3 (lights) * 4 headings = 12).

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.

- 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.

- If a higher default value is Q_value is chosen (5.0 instead of 0.0), the success rate increases to 98-99% from ~90%. The average reward per trip also increases to ~22 from 20.5.

After tweaking the parameters, the cab has become smarter. It can make the right decision most of the time. The success rate of reaching destination within deadline is ~98-99% with average reward per trip is 22 points.

An optimal policy would be a policy which:

- Let the agent arrive at the destination within the deadline.

- Earn maximum reward.

- (obeying traffic law has less priority because of the low penalty: -1).

Does it produce optimal result ? **Yes, it does.** The agent reaches destination most of the time (98-99%) and obey traffic rules. However, it tends to take the shortest path (it only sometimes circles around to earn more reward.). This is probably because the deadline is not included in the state. However, if deadline is included in the state, the state space shall increases considerably (x30-x40 folds). Also i would prefer a cab that is more efficient (e.g. rather arrive

sooner than later, since you will never know which the light/traffic condition closer to destination is).