Udacity Machine Learning Nanodegree

Project 4: Reinforcement Learning
Train a Smartcab to Drive

Jingzhe Hu

1 Goal of the project

The major project goal is to train an AI driving agent to learn traffic rules. A planner (Class RoutePlanner) will handle the route assignment and specify an action at each time point while **the smart driving agent would be responsible to decide whether it's ok to perform the action**.

2 Implement a Basic Driving Agent

Through the **env.sense()** method, the basic driving agent (**agent_random.py**) can detect the status of traffic light as well as the presence and moving direction of other cars at the intersection. The deadline can be obtained through the **env.get_deadline()** method. At each time point, the basic driving agent will produce a random action (one of None, 'forward', 'left', 'right').

Without enforcing the deadline, the basic agent can reach the destination eventually, albeit usually with much longer time compared to the deadline. In a representative test run (**n_trials=10**), the basic agent was able to reach the destination 1 out of 10 times before the deadline. (Figure 1) In trial 0, for example, the basic AI took 236 steps while the deadline is 24 steps.

3 Identify and Update State

Three variable categories are relevant for the driving agent to learn traffic rules: **a)** traffic light status: **green** or **red**, **b)** action determined by the planner: **None**, **forward**, **left** or **right**, and **c)** presence of any car near the intersection: **True**, **False**.

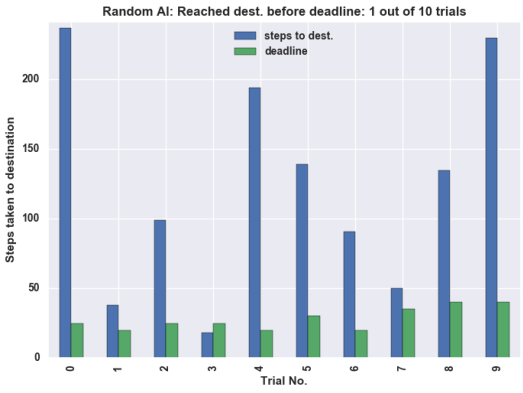

Figure 1: Basic driving agent eventually reaches the destination but usually needs a lot more time compared to deadline.

Combination of the above three variable categories produces a total of **18 states**, which can be constructed via **itertools.product()**. This particular set of states offers a reasonable trade-off, capturing the most crucial information of about the environment while avoiding the complexity of specifying the full status of other cars at the intersection including their relative positions and driving directions. Based on current **inputs**, the state of the smart agent would be represented by the following 3 member tuple: **(inputs['light'], next_waypoint, any_car_nearby)**.

4 Implement Q-Learning

Implemented as a pandas.DataFrame, the Q-table has rows referring to the states of the agents and the columns corresponding to two possible actions (ok to perform current action or not). Entries in the Q-table are the Q-values.

The initial parameters chosen for the Q-update rule (Eqn 1) are listed in Table 1.

discount factor	γ	0.85
learning rate	α	0.15
error rate	ϵ	0.15

Table 1: Parameters for ϵ-greedy Q-update rule

Given current state, the action with the highest Q-value will be picked with a probability of (1-ϵ) while an random action will be picked to facilitate state exploration with a probability of ϵ. Alternatively, softmax action selection can be implemented to mitigate the probability of selecting disastrous actions. With the Q-learning and action selection rules in place, the smartcab can now move towards the destination through the guidance of the planner.

$$\hat{Q}(s_t, a_t) = (1 - \alpha_t)\,\hat{Q}(s_t, a_t) + \alpha_t\,[r_{t+1}(s_t, a_t) + \gamma\,\max_{a'}\hat{Q}(s_{t+1}, a')]$$

5 Enhance the Driving Agent

To make the situation a bit harder for the smartcab, the deadline was decreased from 5 to 4 times the computed distance between the starting position and destination. With current implementation of the ϵ-greedy Q-learning algorithm, smartcab can reach the destination more than 97% of the time on testing trials after 100 training trials. During the testing trials, the ϵ is set to zero taking full advantage of the Q-table to select the optimal policy.

In order to tune the learning rate, 50 independent experiments (100 training trials + 10 testing trials for each exp) were run for each value of α (n_α=5). After the tuning process, an α value of 0.2 was found to produce the highest success rate of 99.4%. (Table 2 and Figure 2) After setting the learning rate to 0.2, a representative testing trial is shown in Figure 3, where the smartcab made it to the destination 10 out 10 times.

α	success rate
0.1	97.8
0.2	99.4
0.3	97.6
0.4	98.2
0.5	97.8

Table 2: An α value of 0.2 leads to the best success rate of 99.4%

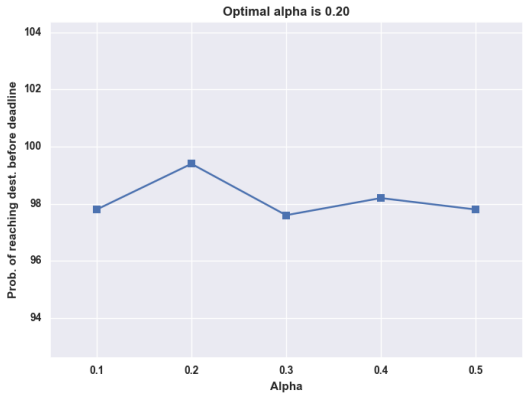

Figure 2: Tuning for the learning rate α

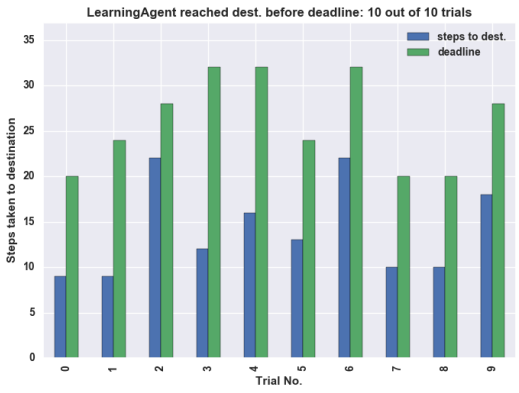

Figure 3: The learning agent can consistently get to the destination in testing trials.

As shown in Figure 4, the cumulative rewards for the learning agent increases monotonically, meaning no penalties incurred during all

of the 10 trials and near optimal policy achieved during the training trials.

6 Conclusion

A smartcab has been successfully trained to learn traffic rules with an ϵ-greedy Q-learning algorithm. After 100 training trials, the smart cab can consistently reach the destination without incurring penalties.

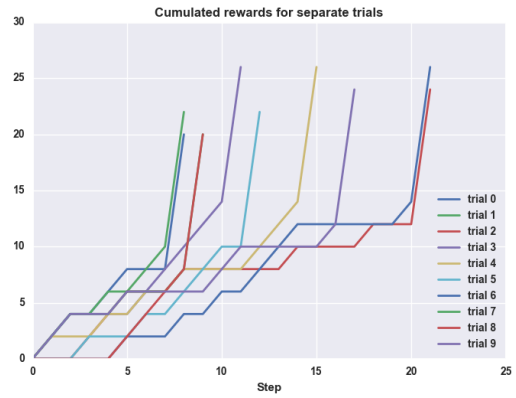

Figure 4: Cumulative rewards for the learning agent during 10 test trials.