Smart Cab Q learning implementation **Michael Wartmann**
Udacity Nanodegree 10/31/2016
Machine Learning Engineer

Implementing a Basic Driving Agent

A non-learning agent takes random actions and moves almost Brownian motion-like, i.e., it carries out a random walk. If waiting long enough with no deadline enforcement active, the cab will eventually reach the destination purely by chance.

Inform the Driving Agent

I have identified the 5 states variables "next waypoint", "traffic light", "oncoming", "left" and "right" as they are sufficient to describe the environment out of the agent's perspective and decision-making space.

1. **Waypoint**: The waypoint direction is a key parameter for the agent to make progress towards its ultimate goal, i.e., reaching the destination. Since the agent does not have a detailed map or path forward, it needs to fully rely on the planner to indicate at each intersection on which decision to make and each correct decision delivers the maximum incremental reward of 2. There are 3 options for the agent: left, forward, right.

2. **Light**: Knowing the color of the traffic light is key to avoid collision with other agents and avoiding a penalty of 1 (reward of -1).

3. **Oncoming**: For a decision to take a left turn, while the traffic light is green, it is important for the agent to know if oncoming traffic turns right or goes straight versus going left to avoid a penalty of 1.

4. **Left**: Typically, the traffic light decides, if the agent can act or waits with traffic from the left, however, with one exception: If the light is red, it is possible to turn right, if there is no traffic from the left moving forward. For both traffic from left going right and left, the right turn on red can be taken.

5. **Right**: This state is modeled only for completeness but could be left out if reduction of computational complexity were required. As we are not dealing with an unsurmountable number of states, I have modeled the state for completeness and later evaluation. Reason for not needing the state is that all decisions are taken care of by the light.

6. **Deadline**: The deadline has been excluded as a state variable, since it is not useful as a state. The agent has no information about the total route and can hence not decide to choose the shortest path. In fact this is repeated one-round game, so planning future states is not needed and not possible as the agent has no knowledge of its spatial domain.

There are 3 options for waypoint, 2 for light, 4 for oncoming, 4 for left and 4 for right. If multiplied, this results in 384 states. If right as a state is eliminated, there are still 96 states. Many of the states will be degenerate, as they are going to be overruled by another state function, i.e., if the traffic light is red, one has to wait for most states and if the light is green, one can move mostly. If not for the exceptions of a right turn on red and waiting for oncoming traffic on green, there would be only 6 relevant states with each waypoint direction either green or red.

Implement a Q-Learning Driving Agent

After implementation of Q-Learning, the agent starts storing Q-values in the Q-table. If any same state is revisited, the agent acts based on its previous experience,i.e., the reward it had earlier received for a particular action would make it either choose for or against the action versus action state pairs it had not visited before. The more the agent learns, the better it has mapped out values for each state action pair and can make an informed choice on which is the best action. In the simulation, for a good set of parameters, the agent learns very fast. Within the first two trials, the agent learns enough to hardly make any more mistakes such that the only penalties occur when epsilon greedy forces the agent to take a random action. By that time, epsilon could be reduced to zero in such a predictable environment.

Improve the Q-Learning Driving Agent

The values used for epsilon, alpha, and gamma can be seen in the table below. Parameter tuning revealed that relatively low values for the learning rate are beneficial. A relatively low learning rate (little oscillation between optimal states) and a small epsilon (few random moves) create a situation in which, after sufficient learning, the agent quickly takes optimal action with few correction moves. Tuning of the gamma parameter has no impact, future states are not relevant for a repeated one-round game. The final agent performs nearly as good the optimal agent. An optimal policy can be shown to be to wait until the light is green and one has right of way, but then move directly to the next waypoint. The agent very quickly learns this from the first few trials, i.e., within the first 50 to 100 moves. The driving agent only fails to reach the destiny in time in one out of 100 trials and even less within the second 50 trials. To find the absolute optimal set of parameters, one could run simulations looping all three parameters and storing the key metrics (success rate, penalties, and average number of steps needed) and perform a search within the created space. This, however, seems an overkill for a simple problem like this in which a good set of parameters can be found with a bit of guided guesswork.

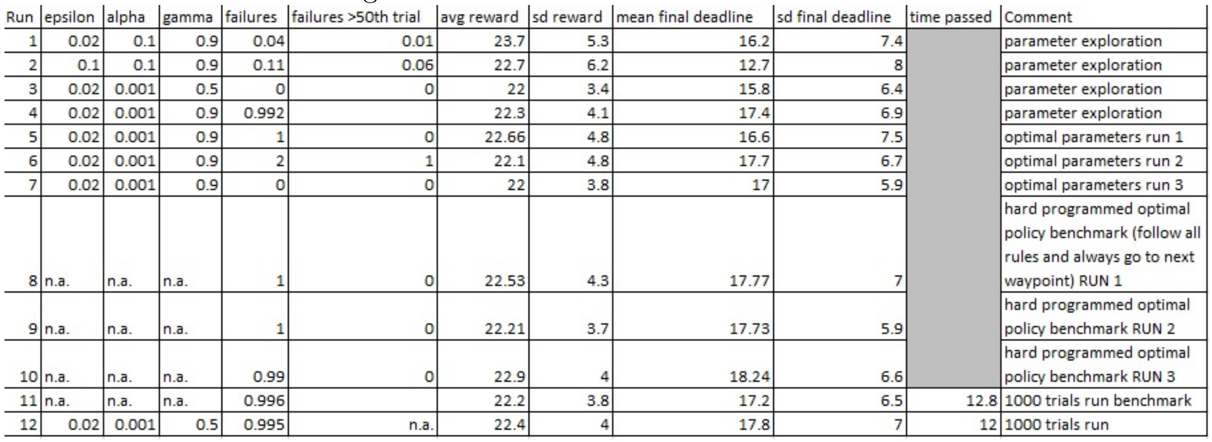
Figure 1: Table of simulations carried out.

Run	epsilon	alpha	gamma	failures	failures >50th trial	avg reward	sd reward	mean final deadline	sd final deadline	time passed	Comment
1	0.02	0.1	0.9	0.04	0.01	23.7	5.3	16.2	7.4		parameter exploration
2	0.1	0.1	0.9	0.11	0.06	22.7	6.2	12.7	8		parameter exploration
3	0.02	0.001	0.5	0	0	22	3.4	15.8	6.4		parameter exploration
4	0.02	0.001	0.9	0.992		22.3	4.1	17.4	6.9		parameter exploration
5	0.02	0.001	0.9	1	0	22.66	4.8	16.6	7.5		optimal parameters run 1
6	0.02	0.001	0.9	2	1	22.1	4.8	17.7	6.7		optimal parameters run 2
7	0.02	0.001	0.9	0	0	22	3.8	17	5.9		optimal parameters run 3
8	n.a.	n.a.	n.a.	1	0	22.53	4.3	17.77	7		hard programmed optimal policy benchmark (follow all rules and always go to next waypoint) RUN 1
9	n.a.	n.a.	n.a.	1	0	22.21	3.7	17.73	5.9		hard programmed optimal policy benchmark RUN 2
10	n.a.	n.a.	n.a.	0.99	0	22.9	4	18.24	6.6		hard programmed optimal policy benchmark RUN 3
11	n.a.	n.a.	n.a.	0.996		22.2	3.8	17.2	6.5	12.8	1000 trials run benchmark
12	0.02	0.001	0.5	0.995	n.a.	22.4	4	17.8	7	12	1000 trials run

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?

The agent gets very close to an optimal policy. I have carried out a simulation with 1000 trials to extend the window towards more measurable statistics. For comparison, I have hard-coded the optimal policy, namely, to follow all traffic rules and always choose the next waypoint. The average time to destination is ca. 12 steps, with 6 steps required traffic waiting time, while having to take an average of 6 steps to reach the destination. The optimal benchmark has roughly the same numbers, while waiting time is ca. 1 step longer even (total of 7 steps waiting time in average). The only improvement that could still be made is to have epsilon and alpha converge to zero to avoid changing a working q-value table and taking random steps not anymore needed for learning. The agent takes wrong steps with a penalty only, if taking a random step forced by epsilon greedy.
The optimal policy is simple: Respect all traffic rules and follow the waypoint planner.