MLND Project 4: Smartcab

Matthew Peyrard

August 6, 2016

1 Implement a Basic Driving Agent

By using a random action strategy the smartcab can usually get to its destination *eventually*. As expected, it often violates traffic laws, gets into accidents, and takes extremely sub-optimal routes to get to the destination.

2 Inform the Driving Agent

2.1

I have defined the state as a 5-tuple: (L, T_L, T_R, T_F, W). L is a boolean flag that is true if the light is green, otherwise false. T_L, T_R and T_F are boolean flags that are true in the absence of other cars in the intersection, where the subscripts L, R and F refer to *left*, *right* and *forward* respectively. And finally, W refers to the next waypoint, and can take on the values *left*, *right* or *forward*.

I have excluded the time limit from the state because I feel it is not necessary. The waypoints should guide us optimally to the destination, meaning the time limit grants us no additional value.

2.2

The first four items in the tuple can take on two values, and the final value can take on three. Therefore there are $2^4 \cdot 3 = 48$ possible states. Furthermore, there are 4 possible actions, meaning that our utility table needs to have $48 \cdot 4 = 192$ entries. The Q-Learning algorithm requires that we iterate over

the table every time to we want to ajust our utilities, and having 192 values to look at is virtually negligible given today's speeds. It also means our table will take up very little memory. The state can be stored in 6 bits, and the actions stored in 2, meaning that we need at most 192 bytes of memory to store our utility table. This is also a negligible amount.

3 Implement a Q-Learning Driving Agent

At first, the agent still drives randomly. However, after some time, the agent begins to show a bias towards behaviors that give it positive rewards. This is happening because are now tracking the utility of each action at each state (a function of the reward), and choosing the actions based on that information. As the agent gains experience over the entire range of states and actions, it demonstrates a greater and greater bias towards the optimal (highest reward) actions. As a result, over a relateivly small number of trials, the agent is capable of precisely following the directions from the waypoints directly to the goal.

4 Improve the Q-Learning Driving Agent

4.1

I used Python to generate several permutations of the three parameters, and iterated over each one twenty times, storing the average accuracy. The most successful set of parameters on average was $\alpha = 0.6$, $\gamma = 0$, and $\epsilon = 0.2$. These parameters produced an average success rate of 92.95%. This is curious, because it means that we are using the rewards directly as the utility.

4.2

The agent not only peforms extremely well with these parameters, but it also seems to learn much faster than with other configurations that I have observed. It very quickly learns to follow the waypoints and to obey the traffic laws. However, the relatively high epsilon value does allow it to make some mistakes on a relatiely frequent basis. I suspect that it might be wise to

have the value of epsilon decay over time while the agent is receiving positive feedback.

I also believe that the policy is not entirely optimal. It seems that the agent tends to spend a lot of time waiting at red lights. If the agent knew its position and the destination, then it might be possible for it to learn that turning right at a red light might allow it to reach the goal faster. However, in order for it to be able to accomplish this, it would be required to have a dynamic reward based on its time remaining, and possibly not be punished for deviating from the assigned route.