MLND Project 4: Smartcab

Matthew Peyrard

August 7, 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 for each one. The most successful set of parameters on average was $\alpha = 1$, $\gamma = 0$, and $\epsilon = 0.2$. These parameters produced an average success rate of 93.2%.

That is a significant improvement over the initial set of parameters: $\alpha = 0.5$, $\gamma = 0.5$ and $\epsilon = 0.05$, which gave an average success rate of 85.7%.

Tables 1 – 6 show the full set of results for all of the permutations of parameters. The values in the tables are the ratios of successful runs to the number of attempted runs.

Table 1: $\alpha = 0$

		γ				
		0	0.2	0.4	0.6	0.8
	0	0.060	0.058	0.050	0.057	0.058
	0.2	0.174	0.180	0.195	0.182	0.198
ϵ	0.4	0.221	0.233	0.242	0.236	0.223
	0.6	0.239	0.234	0.226	0.222	0.237
	0.8	0.225	0.214	0.217	0.226	0.231

Table 2: $\alpha = 0.2$

		γ				
		0	0.2	0.4	0.6	0.8
	0	0.411	0.560	0.513	0.505	0.526
	0.2	**0.917**	0.690	0.870	0.802	0.767
ϵ	0.4	0.793	0.369	0.753	0.746	0.690
	0.6	0.579	0.746	0.590	0.572	0.551
	0.8	0.376	0.802	0.357	0.360	0.344

Table 3: $\alpha = 0.4$

		γ				
		0	0.2	0.4	0.6	0.8
	0	0.429	0.545	0.511	0.589	0.514
	0.2	**0.909**	**0.921**	0.877	0.850	0.827
ϵ	0.4	0.791	0.769	0.772	0.730	0.699
	0.6	0.583	0.582	0.572	0.543	0.547
	0.8	0.366	0.366	0.356	0.369	0.358

Table 4: $\alpha = 0.6$

		γ				
		0	0.2	0.4	0.6	0.8
	0	0.435	0.498	0.447	0.484	0.425
	0.2	**0.922**	0.893	0.861	0.857	0.812
ϵ	0.4	0.783	0.758	0.742	0.733	0.681
	0.6	0.576	0.571	0.562	0.541	0.528
	0.8	0.370	0.375	0.370	0.372	0.327

Table 5: $\alpha = 0.8$

		γ				
		0	0.2	0.4	0.6	0.8
	0	0.439	0.525	0.401	0.487	0.411
	0.2	**0.926**	0.896	0.873	0.850	0.808
ϵ	0.4	0.775	0.767	0.753	0.690	0.652
	0.6	0.572	0.561	0.573	0.510	0.487
	0.8	0.383	0.368	0.346	0.338	0.337

Table 6: $\alpha = 1$

		γ				
		0	0.2	0.4	0.6	0.8
	0	0.402	0.579	0.343	0.481	0.377
	0.2	**0.932**	**0.903**	0.844	0.816	0.761
ϵ	0.4	0.797	0.764	0.705	0.677	0.634
	0.6	0.587	0.547	0.517	0.509	0.486
	0.8	0.372	0.366	0.342	0.334	0.316

4.2

I would describe an optimal policy as one where the epsilon value decays over time as the algorithm becomes more successful. I would also say that the algorithm should be capable of deviating from the navigated plan in select scenarios. The primary scenario for this rule would be if the destination is such that the vehicle could just as easily move right as it could move forward. If the vehicle is stuck at a red light, then it could prematurely move right and hope for better luck on the green lights when it needs to move forward.

The current solution is for the epsilon value to be static, which allows the algorithm to continue making mistakes after the utility values have reached a highly optimal state.