Smart Cab – How to Drive

Implement a basic driving agent

Implement the basic driving agent, which processes the following inputs at each time step:

Next waypoint location, relative to its current location and heading, Intersection state (traffic light and presence of cars), and, Current deadline value (time steps remaining), And produces some random move/action (None, ‘forward’, ‘left’, ‘right’). Don’t try to implement the correct strategy! That’s exactly what your agent is supposed to learn.

Run this agent within the simulation environment with enforce_deadline set False (see run function in agent.py), and observe how it performs. In this mode, the agent is given unlimited time to reach the destination. The current state, action taken by your agent and reward/penalty earned are shown in the simulator.

Observe what you see with the agent's behaviour as it takes random actions. Does the smart cab eventually make it to the destination? Are there any other interesting observations to note?

Yes, the agent finally reached the destination. As it is the greedy approach where is is choosing the actions randomly, thus it took a lot of time and does not care whether there is an oncoming vehicle or whether that vehicle is at its right or left and what is the state of red light. Which leads to take a lot of trials and time. It takes so long that there is a significant difference from the deadline.

Identify and Update State

Identify a set of states that you think are appropriate for modeling the driving agent. The main source of state variables are current inputs, but not all of them may be worth representing. Also, you can choose to explicitly define states, or use some combination (vector) of inputs as an implicit state.

At each time step, process the inputs and update the current state. Run it again (and as often as you need) to observe how the reported state changes through the run.

What states have you identified that are appropriate for modelling the **smartcab** and environment? Why do you believe each of these states to be appropriate for this problem?

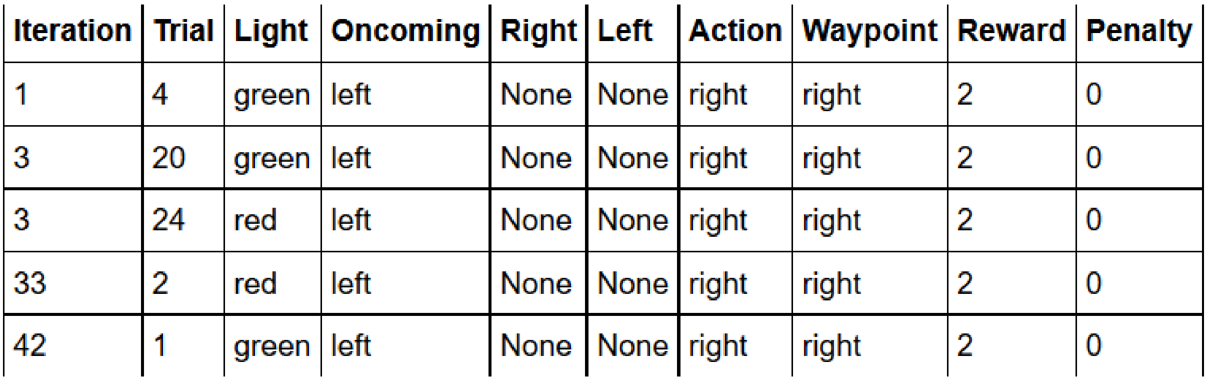
```
if(light=red):
        next_waypoint = [right,left(-),forward(-)]

        if(next_waypoint=forward)
                action=[none(+) or right (-)]
else if (light=green)
        next_waypoint = [right(+), left, forward(+)]

        if(oncoming=forward)
                action=[none(+) or left (-)]
```

And also the car seems to turn right when oncoming car is turning left and the light is red because the next_waypoint is right everytime.

It can be clearly noticed that the agent rewards such cased when the light is red and the oncoming is left the agent moves right. Thus, it makes no sense to take the right action into account when light is red and intersection is with oncoming left vehicle.



Iteration	Trial	Light	Oncoming	Right	Left	Action	Waypoint	Reward	Penalty
1	4	green	left	None	None	right	right	2	0
3	20	green	left	None	None	right	right	2	0
3	24	red	left	None	None	right	right	2	0
33	2	red	left	None	None	right	right	2	0
42	1	green	left	None	None	right	right	2	0

States as per US Laws

1. *Light: red or green*
2. *Oncoming: forward, left, right or none*
3. *Left: Forward, left, right or none*
4. *Action: Take the next waypoint or none*
5. *Next waypoint: forward, left or right*

*It does not make sense to use deadline to fold into each state as the agent seems to learn the correct set of rules under **100** trials and reaching the destination safely, maximizing the rewards it got by following the next_waypoint thus deadline does not make any sense.*

Also, the deadline is being omitted as if we were to include the deadline into our current state, our state space would blow up, we would suffer from the curse of dimensionality and it would take a long time for the q-matrix to converge. Also note that including the deadline could possibly influence the agent in making illegal moves when the deadline is near.

Implement Q-Learning

Implement the Q-Learning algorithm by initializing and updating a table/mapping of Q-values at each time step. Now, instead of randomly selecting an action, pick the best action available from the current state based on Q-values, and return that.

Each action generates a corresponding numeric reward or penalty (which may be zero). Your agent should take this into account when updating Q-values. Run it again, and observe the behaviour.

What changes do you notice in the agent's behaviour when compared to the basic driving agent when random actions were always taken? Why is this behaviour occurring?

The agent reaches the destination much faster. As while exploiting the earlier trials the agent was also exploring while collecting the positive and negative rewards. Thus, it helped to collect more rewards in the later trials and reach the destination quickly i.e. in initial trials there are some negative rewards but later on there were no negative rewards. This behaviour is observed because agent gradually learning to follow the best next_waypoint. It is not random now and is learning to follow traffic rules as well though initial trials are being penalized.

Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best? How well does the final driving agent perform?

Q-Learning alpha and gamma parameters are being trained from values [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9] and reward_per_action and penalty_per_action is being calculated and analysed in excel to sort the data(qLearningTunning.csv). Thus alpha=0.9 and gamma=0.3 is being considered to the best value in this range of values. Though there is not much difference between gamma=0.3 and 0.5 but at the same time penalty_per_action is better, which is also analysed after repetitions. While sorting the following table with reward_per_action and penalty_per_action in decreasing order, one get to know the optimized value of alpha and gamma.

alpha	gamma	actions	rewards	penalty	reward_per_action	penalty_per_action
0.9	0.3	1306	2711.5	70	2.07618683	0.053598775
0.9	0.5	1444	2671.5	170	1.850069252	0.117728532
0.9	0.1	1233	2190	45	1.776155718	0.03649635
0.9	0.7	1631	2789	478	1.709993869	0.293071735
0.9	0.9	1584	2686	460	1.695707071	0.29040404
0.9	0.6	1363	2310	239	1.694790902	0.175348496
0.5	0.3	1299	2172.5	125	1.672440339	0.096227868
0.9	0.2	1298	2157	62	1.661787365	0.047765794
0.6	0.9	1569	2600	453	1.657106437	0.288718929
0.9	0.8	1546	2533.5	453	1.638745149	0.29301423
0.6	0.1	1321	2164.5	76	1.638531416	0.057532173
0.7	0.4	1372	2230.5	113	1.625728863	0.082361516
0.6	0.7	1586	2578	481	1.625472888	0.303278689
0.5	0.1	1334	2156.5	91	1.616566717	0.068215892

0.8	0.8	1661	2680	480	1.613485852	0.288982541
0.8	0.6	1391	2241.5	189	1.611430625	0.135873472
0.8	0.5	1331	2133	106	1.60255447	0.079639369
0.5	0.7	1579	2523.5	477	1.598163395	0.30208993
0.8	0.7	1453	2306	259	1.587061253	0.178251893
0.6	0.2	1358	2152	82	1.584683358	0.060382916
0.7	0.5	1444	2278.5	156	1.577908587	0.108033241
0.9	0.4	1398	2205.5	78	1.577610873	0.055793991
0.7	0.2	1401	2202	78	1.571734475	0.055674518
0.5	0.2	1393	2182	99	1.566403446	0.071069634
0.5	0.6	1649	2575	500	1.561552456	0.303214069
0.8	0.3	1370	2137.5	94	1.560218978	0.068613139
0.5	0.5	1359	2111.5	185	1.553715968	0.136129507
0.6	0.6	1405	2171	183	1.54519573	0.13024911
0.7	0.7	1640	2527.5	374	1.541158537	0.22804878
0.5	0.9	1684	2590.5	516	1.538301663	0.306413302
0.8	0.2	1330	2044.5	68	1.537218045	0.05112782
0.4	0.2	1431	2198.5	123	1.536338225	0.085953878
0.5	0.8	1652	2530.5	511	1.531779661	0.309322034
0.4	0.6	1675	2564.5	531	1.531044776	0.317014925
0.7	0.8	1721	2629	517	1.527600232	0.30040674
0.7	0.3	1428	2168.5	86	1.518557423	0.06022409
0.5	0.4	1493	2259.5	135	1.513395847	0.090421969
0.3	0.1	1367	2067.5	143	1.512435991	0.104608632
0.6	0.8	1718	2583.5	555	1.503783469	0.323050058
0.6	0.3	1453	2174	102	1.496214728	0.070199587
0.4	0.7	1669	2489.5	487	1.491611744	0.291791492
0.4	0.3	1421	2119.5	170	1.491555243	0.119634061
0.4	0.9	1725	2572	539	1.491014493	0.312463768
0.6	0.5	1441	2145.5	167	1.4888966	0.115891742
0.8	0.4	1512	2236.5	110	1.479166667	0.072751323
0.4	0.1	1421	2099.5	103	1.477480647	0.072484166
0.4	0.5	1482	2185	212	1.474358974	0.143049933
0.3	0.2	1416	2080	167	1.468926554	0.117937853
0.7	0.1	1514	2219	64	1.465653897	0.042272127
0.3	0.7	1769	2543	568	1.437535331	0.321085359
0.4	0.4	1500	2153	193	1.435333333	0.128666667
0.3	0.3	1453	2075	191	1.428079835	0.131452168
0.6	0.4	1501	2136	120	1.423051299	0.079946702
0.7	0.6	1540	2186	199	1.419480519	0.129220779
0.4	0.8	1730	2454.5	507	1.418786127	0.293063584
0.3	0.8	1579	2219	501	1.405319823	0.317289424
0.3	0.6	1728	2423	460	1.402199074	0.266203704
0.7	0.9	1695	2365	438	1.395280236	0.25840708
0.3	0.5	1502	2064	260	1.374167776	0.17310253
0.2	0.2	1467	1998	242	1.36196319	0.164962509

0.2	0.3	1477	2008	267	1.359512525	0.180771835
0.8	0.9	1706	2311.5	491	1.354923798	0.287807737
0.3	0.9	1939	2624	653	1.353274884	0.336771532
0.3	0.4	1517	2028	209	1.336849044	0.137771918
0.2	0.9	1858	2471.5	634	1.330193757	0.341227126
0.2	0.5	1640	2168.5	346	1.322256098	0.21097561
0.2	0.1	1548	2027	207	1.309431525	0.13372093
0.2	0.4	1613	2106.5	306	1.305951643	0.189708617
0.2	0.6	1792	2338	610	1.3046875	0.340401786
0.2	0.7	1884	2370.5	642	1.258227176	0.340764331
0.2	0.8	1915	2295	673	1.19843342	0.351436031
0.1	0.2	1588	1854	434	1.167506297	0.273299748
0.1	0.9	1982	2126.5	723	1.072906155	0.364783047
0.1	0.1	1719	1817	408	1.057009889	0.237347295
0.1	0.3	1773	1873.5	498	1.056683587	0.280879865
0.1	0.4	1902	1930.5	561	1.014984227	0.294952681
0.1	0.5	1989	1996	650	1.003519356	0.326797386
0.1	0.6	2013	1772	781	0.880278192	0.387978142
0.1	0.8	2247	1920	885	0.85447263	0.393858478
0.1	0.7	2154	1827.5	864	0.848421541	0.401114206
0.8	0.1	2337	1328	63	0.568249893	0.026957638

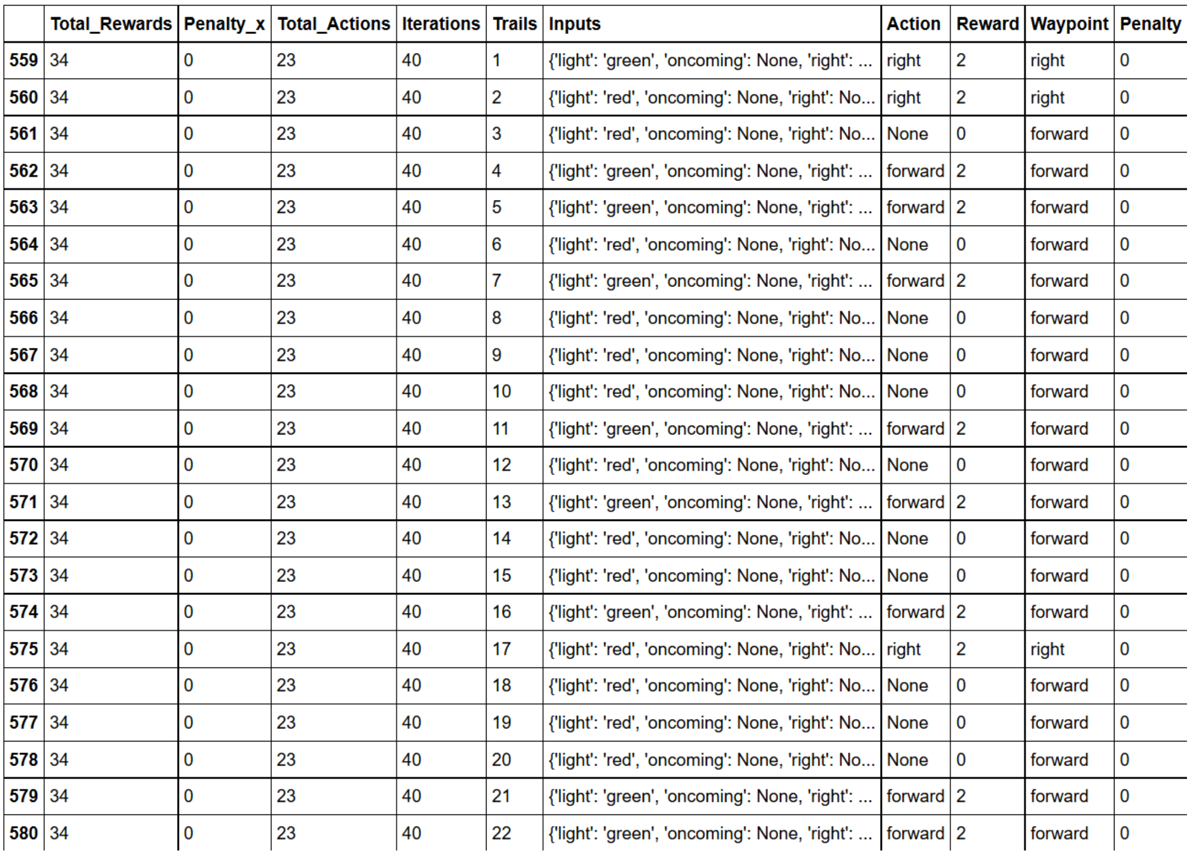
At alpha=0.9 and gamma=0.3 the agent reaches the destination 97/100 times thus the performance of the driving agent is really good. (alpha_0.9_gamma_0.3.csv)

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?

Yes, obviously the agent reached to the optimal policy. In 100 iterations, it consistently reaches the destination, and minimally incurs penalties (especially in the beginning). After iterations, it seems to consistently take correct actions. One thing I would add is that the "learning" could probably be accelerated by creating more dummy agents (i.e. it would speed up convergence for states that were happening less often by making those events happen more frequently). By increasing the number of dummy agents, the car would probably interact more with them, and would accumulate penalties to avoid earlier than later. This would then translate to less "training time", and would be a nice enhancement.

Following represents the optimal policy which has been chosen based on the following criteria i.e ***Total_Reward_Earned, Total_Penalty_Earned, Total_Actions***

```
sort_values(by=['Reward_x', 'Penalty_x','Total_Actions'], ascending=[0,1,1])
```



	Total_Rewards	Penalty_x	Total_Actions	Iterations	Trails	Inputs	Action	Reward	Waypoint	Penalty
559	34	0	23	40	1	{'light': 'green', 'oncoming': None, 'right': ...	right	2	right	0
560	34	0	23	40	2	{'light': 'red', 'oncoming': None, 'right': No...	right	2	right	0
561	34	0	23	40	3	{'light': 'red', 'oncoming': None, 'right': No...	None	0	forward	0
562	34	0	23	40	4	{'light': 'green', 'oncoming': None, 'right': ...	forward	2	forward	0
563	34	0	23	40	5	{'light': 'green', 'oncoming': None, 'right': ...	forward	2	forward	0
564	34	0	23	40	6	{'light': 'red', 'oncoming': None, 'right': No...	None	0	forward	0
565	34	0	23	40	7	{'light': 'green', 'oncoming': None, 'right': ...	forward	2	forward	0
566	34	0	23	40	8	{'light': 'red', 'oncoming': None, 'right': No...	None	0	forward	0
567	34	0	23	40	9	{'light': 'red', 'oncoming': None, 'right': No...	None	0	forward	0
568	34	0	23	40	10	{'light': 'red', 'oncoming': None, 'right': No...	None	0	forward	0
569	34	0	23	40	11	{'light': 'green', 'oncoming': None, 'right': ...	forward	2	forward	0
570	34	0	23	40	12	{'light': 'red', 'oncoming': None, 'right': No...	None	0	forward	0
571	34	0	23	40	13	{'light': 'green', 'oncoming': None, 'right': ...	forward	2	forward	0
572	34	0	23	40	14	{'light': 'red', 'oncoming': None, 'right': No...	None	0	forward	0
573	34	0	23	40	15	{'light': 'red', 'oncoming': None, 'right': No...	None	0	forward	0
574	34	0	23	40	16	{'light': 'green', 'oncoming': None, 'right': ...	forward	2	forward	0
575	34	0	23	40	17	{'light': 'red', 'oncoming': None, 'right': No...	right	2	right	0
576	34	0	23	40	18	{'light': 'red', 'oncoming': None, 'right': No...	None	0	forward	0
577	34	0	23	40	19	{'light': 'red', 'oncoming': None, 'right': No...	None	0	forward	0
578	34	0	23	40	20	{'light': 'red', 'oncoming': None, 'right': No...	None	0	forward	0
579	34	0	23	40	21	{'light': 'green', 'oncoming': None, 'right': ...	forward	2	forward	0
580	34	0	23	40	22	{'light': 'green', 'oncoming': None, 'right': ...	forward	2	forward	0

It can be very well seen that the initial trials do incur a fair amount of penalty but gradually as the agent learns the penalty goes to zero.

Trial	Penalty
1	-13.5
2	-12
3	-11.5
4	-5
10	-3
22	-2
5	-1.5
11	-1
6	-0.5
12	-0.5
15	-0.5
17	-0.5
75	0
74	0
73	0
68	0
72	0
71	0
70	0
69	0

63	0
66	0
65	0
64	0
76	0
62	0
61	0
60	0
59	0
58	0
67	0
77	0
83	0
79	0
98	0
97	0
96	0
95	0
94	0
93	0
92	0
91	0
90	0
89	0
88	0
87	0
86	0
85	0
84	0
57	0
82	0
81	0
80	0
78	0
56	0
50	0
54	0
30	0
29	0
28	0
27	0
26	0
25	0
24	0
23	0
31	0

21	0
19	0
18	0
16	0
14	0
13	0
9	0
8	0
7	0
20	0
32	0
33	0
34	0
53	0
52	0
51	0
99	0
49	0
48	0
47	0
46	0
45	0
44	0
43	0
42	0
41	0
40	0
39	0
38	0
37	0
36	0
35	0
55	0
100	0

Also, the initial values were not going the correct way which can be seen through the following table.

Trails	Inputs	Action	Reward	Waypoint	Penalty
33	{'light': 'red', 'oncoming': None, 'right': No...	forward	-1.0	left	-1.0
34	{'light': 'red', 'oncoming': None, 'right': No...	right	-0.5	left	-0.5
35	{'light': 'red', 'oncoming': None, 'right': No...	right	-0.5	forward	-0.5
36	{'light': 'green', 'oncoming': None, 'right': ...	forward	-0.5	left	-0.5
38	{'light': 'green', 'oncoming': None, 'right': ...	left	-0.5	forward	-0.5