

Implement a basic driving agent

In your report, mention what you see in the agent's behavior. Does it eventually make it to the target location?

The basic driving agent achieves the target in around 20% of the times. So it is just luck dependence if the car could reach the destination point.

Identify and update state

Justify why you picked these set of states, and how they model the agent and its environment.

The main work is done by planner, so if agent sticks to navigation that suggested by next_waypoint it will easily reach the destination. But in this case the idea is to be able gather more rewards and as a result needs to find a different ways. So just the next_waypoint is not enough:

1. Planner navigation – **next waypoint** is the important anchor in the agent state;
2. There are a lot of states' components could be used like time, location, sense, but we are limited by exercise for relative fast learning that should be accomplished in 100 trials. The best candidate is **sense**.

The state like (agent location, time to deadline, destination) very attractive but to learn all states requires a lot of resources.

On the other hand, sense + next_waypoint will very promise and optimistic to learn it faster.

Implement Q-Learning

What changes do you notice in the agent's behavior?

- Car started reach the destination frequently.
- Started follow the rules;
- Amount of rewards was increasing;
- Amount of moves increased as well;
-

Enhance the driving agent

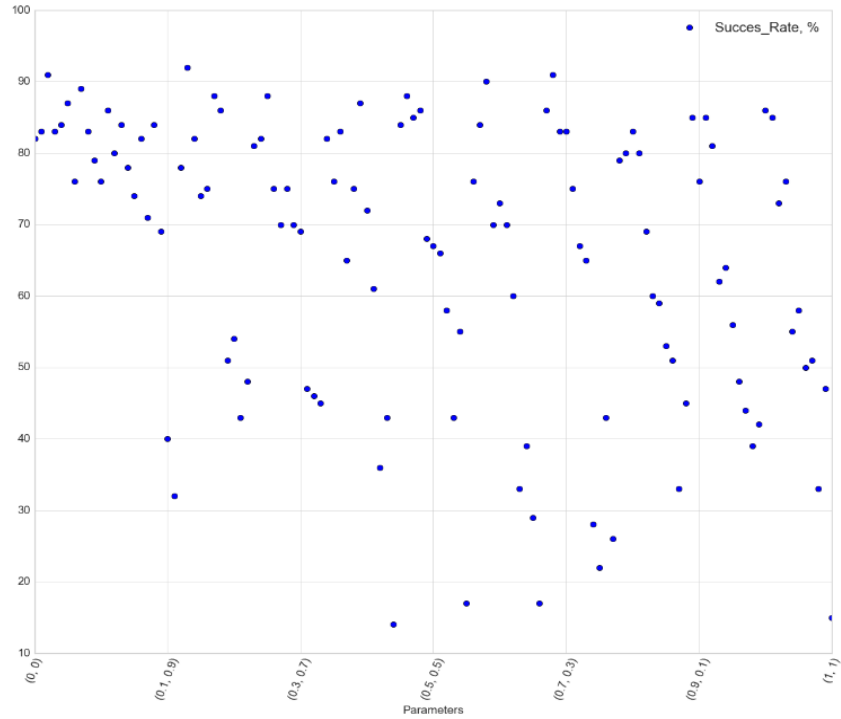
Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does it perform?

1. The sense of dummy cars are coming from the right was removed from the state. It is not playing any role according to US traffic rules.

- Counting the actions that following the rules, otherwise Environment prevents the agent move (chooseAction in a code), it reduced amount of penalties also;
- Ran the different Alpha and Gamma (see appx below), but in actual there are a lot of combinations:

Alpha	Gamma	Success Rate, %	Rewards
0.2	0.1	92	3228
0	0.2	91	3270
0.7	0.1	91	3149.5
0.6	0.2	90	3203
0	0.7	89	2960
0.2	0.5	88	3023.5
0.3	0.2	88	3147
0.5	0.1	88	3323.5
0	0.5	87	3209
0.4	0.5	87	3101.5
0.1	0	86	3159
0.2	0.6	86	3311
0.5	0.3	86	3129
0.7	0	86	3144
1	0	86	3115.5
0.5	0.2	85	3177.5
0.9	0	85	3210
0.9	0.2	85	3100.5
1	0.1	85	3207

Rewards should be dividing by 100



Success rate (reached/trials) is not a good criteria, because the because it takes a time to learn. Wiki says to increase learn rate Alpha should be high. Success Criteria in this exercise is 7 out of last ten moves and a lot of combinations are possible.

So it's hard to say what the best, a lot is of depends on starting point, but it gave the idea of the good Alpha-Gamma pairs, and the bad also:

Alpha	Gamma	Success Rate, %	Rewards
0.9	0.8	44	3256.5
0.2	0.9	43	3143.5
0.4	0.9	43	2952
0.5	0.8	43	3005.5
0.7	0.9	43	2979
0.9	1	42	2877.5
0.1	0.9	40	2982
0.6	0.8	39	3053
0.9	0.9	39	3063.5
0.4	0.8	36	3257.5
0.6	0.7	33	2957.5
0.8	0.9	33	3008
1	0.8	33	2888.5
0.1	1	32	2982
0.6	0.9	29	2765
0.7	0.7	28	3065.5
0.7	1	26	2949.5
0.7	0.8	22	2764
0.5	1	17	2641.5
0.6	1	17	2734
1	1	15	2885
0.4	1	14	2580

As well the starting Q-table value is very important, I found that '-1' is doing great job. Epsilon value important also - too big, will lead to a lot of random movements, too small – algo will sticky to next_waypoint then amount of rewards will decrease. Finally, following Wiki recommendation, alpha was chosen 0.7 and GAMMA = 0.1. I didn't see any benefits of using dynamic (time dependence) Alpha and Gamma.

So, as a result the code produces Q-table that successfully passing the last 10 moves in average 9 wins.

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?

No, because planner gives the minimum time and Q-table in a good chance can achieve it in a same time. In general time is higher, because of randomness that implemented. But amount of rewards has been increased almost by factor '2'.

Reference:

<https://github.com/studywolf/blog/tree/master/RL>

Appendix – TEST of different Alpha and Gamma:

Alpha	Gamma	Success Rate, %	Rewards
0.2	0.1	92	32.28
0	0.2	91	32.7
0.7	0.1	91	31.495
0.6	0.2	90	32.03
0	0.7	89	29.6
0.2	0.5	88	30.235
0.3	0.2	88	31.47
0.5	0.1	88	33.235
0	0.5	87	32.09
0.4	0.5	87	31.015
0.1	0	86	31.59
0.2	0.6	86	33.11
0.5	0.3	86	31.29
0.7	0	86	31.44
1	0	86	31.155
0.5	0.2	85	31.775
0.9	0	85	32.1
0.9	0.2	85	31.005
1	0.1	85	32.07
0	0.4	84	31.885
0.1	0.2	84	32.835
0.1	0.7	84	32.72
0.5	0	84	31.7
0.6	0.1	84	30.915
0	0.1	83	33.24
0	0.3	83	32.8
0	0.8	83	32.7
0.4	0.2	83	31.26
0.7	0.2	83	31.665
0.7	0.3	83	32.28
0.8	0.2	83	31.505
0	0	82	31.725
0.1	0.5	82	31.96
0.2	0.2	82	33.3
0.3	0.1	82	32.755

0.4	0	82	31.76
0.3	0	81	32.6
0.9	0.3	81	32.26
0.1	0.1	80	32.465
0.8	0.1	80	29.99
0.8	0.3	80	32.355
0	0.9	79	30.12
0.8	0	79	32.16
0.1	0.3	78	30.81
0.2	0	78	32.285
0	0.6	76	32.5
0	1	76	31.96
0.4	0.1	76	30.435
0.6	0	76	32.61
0.9	0.1	76	31.895
1	0.3	76	32.035
0.2	0.4	75	32.5
0.3	0.3	75	31.68
0.3	0.5	75	32.325
0.4	0.4	75	32.765
0.7	0.4	75	32.52
0.1	0.4	74	33.11
0.2	0.3	74	31.9
0.6	0.4	73	31.185
1	0.2	73	32.385
0.4	0.6	72	33.66
0.1	0.6	71	32.545
0.3	0.4	70	29.755
0.3	0.6	70	30.165
0.6	0.3	70	32.015
0.6	0.5	70	31.89
0.1	0.8	69	33.48
0.3	0.7	69	32.19
0.8	0.4	69	32.81
0.5	0.4	68	30.865
0.5	0.5	67	31.795
0.7	0.5	67	32.805
0.5	0.6	66	33.4

0.4	0.3	65	32.62
0.7	0.6	65	31.475
0.9	0.5	64	31.985
0.9	0.4	62	30.765
0.4	0.7	61	31.605
0.6	0.6	60	30.72
0.8	0.5	60	32.075
0.8	0.6	59	30.97
0.5	0.7	58	32.425
1	0.5	58	30.97
0.9	0.6	56	31.685
0.5	0.9	55	32.555
1	0.4	55	30.445
0.2	0.8	54	30.465
0.8	0.7	53	33.43
0.2	0.7	51	31.99
0.8	0.8	51	30.855
1	0.7	51	30.44
1	0.6	50	31.21
0.2	1	48	28.72
0.9	0.7	48	29.55
0.3	0.8	47	30.815
1	0.9	47	31.025
0.3	0.9	46	29.355
0.3	1	45	29.32
0.8	1	45	30.13
0.9	0.8	44	32.565
0.2	0.9	43	31.435
0.4	0.9	43	29.52
0.5	0.8	43	30.055
0.7	0.9	43	29.79
0.9	1	42	28.775
0.1	0.9	40	29.82
0.6	0.8	39	30.53
0.9	0.9	39	30.635
0.4	0.8	36	32.575
0.6	0.7	33	29.575
0.8	0.9	33	30.08

1	0.8	33	28.885
0.1	1	32	29.82
0.6	0.9	29	27.65
0.7	0.7	28	30.655
0.7	1	26	29.495
0.7	0.8	22	27.64
0.5	1	17	26.415
0.6	1	17	27.34
1	1	15	28.85
0.4	1	14	25.8