Smart Cab

Basic driving agent

Agent accepts inputs

Agent accepts next waypoint, deadline and state of the intersection as its inputs.

Produces a valid output

Agent produces one of None, forward, left and right as its output. The output is a random action from the list of the valid actions from the environment.

Runs in simulator

Agents runs successfully in the simulator.

Overall response

Since the agent produces random action in every step, it could take valid action in each step but fails to reach the destination.

Identify and update state

Reasonable states identified

Out of the inputs - next waypoint, deadline and intersection state, I chose next waypoint and intersection state as the state of the environment. Next waypoint indicates the direction the agent must follow to reach the destination, without which no reasonable decision can be taken. The state of the intersection encodes the traffic signal, cars on the other sides of the intersection. These are necessary to observe in order to not violate the traffic signal. Deadline doesn't have to be included in the state space. It doesn't help in making the optimal decision because deadline is relevant only in the notion of a distance to travel. Since, the agent doesn't get any information about the distance to travel and all the information it has is to make correct decision based on traffic lights and other cars to not violate traffic rules and next_waypoint to get to the destination on time, these are enough to make optimal decisions.

From above discussions, it follows that we can't skip next_waypoint, traffic lights and position of other cars to make optimal decisions and not violate traffic conditions. Also, the reward structure entails that these are valid components of the state space - as the agent receives positive reward for following waypoint given by planner and negative reward for every mistake it makes.

Agent updates state

The agent reads the input for every step and updates its state corresponding to the inputs it receives.

Implement Q Learning

Agent updates Q-values

Agent updates the Q-values for the state action pairs, based on the following formula:

reward + gamma * Q_max(for the available actions)

Picks the best action

As can be seen above, choosing a random action won't help much in long run. But choosing a random action is only available option during the start of the training. This helps the agent in exploring the states and action pairs and learn a lookup table of actions-values pair. Effective learning needs a mixture of exploration-exploitation using the epsilon greedy algorithm. With a small probability epsilon, we choose to explore the available valid actions and with probability (1-epsilon) we choose to exploit what had been learnt in the past.

Changes in behavior

Compared to the agent choosing a random action in each state, the agent that implements basic Q -learning is smart enough to choose correct actions and hence reaches destinations in most of the trials and reduced the mistakes it commits. But, since the learning strategy doesn't involve anything that lets improve over time, it took really long times to train and failed to reach destination on time consistently most of the times.

Enhance the driving agent

The basic Q-learning algorithm doesn't contain means to update the rewards based on the changes that occur with time and also it doesn't weigh the long and short term rewards properly. In order to implement what is called as Temporal Difference Learning, we need to use enhanced Q-learning that incorporates the notions of learning rate and discount factor.

So, the agent is updated to implement the following enhanced version of Q-learning so that it learns to choose correct action in every step as time progresses. The expectation is that the agent learns to consistently reach destination with positive net reward.

(1-alpha)* Q_old + alpha * (self.reward + gamma * Q_max(for all available actions))

Agent learns a feasible policy in 100 trials

With the enhanced Q-learning algorithm, and with proper values for the learning rate and discount factor the agent could learn a policy that helped it to reach the destination most of the times consistently with net positive rewards. One observation is that agent made a lot of mistakes during the beginning of the iterations and as time passed, the errors reduced and it could reach the destination consistently on time. The stats are reported in the following section. One thing to note however, is the agent produced different results for same parameters due to inherent randomness in the system.

Improvements reported

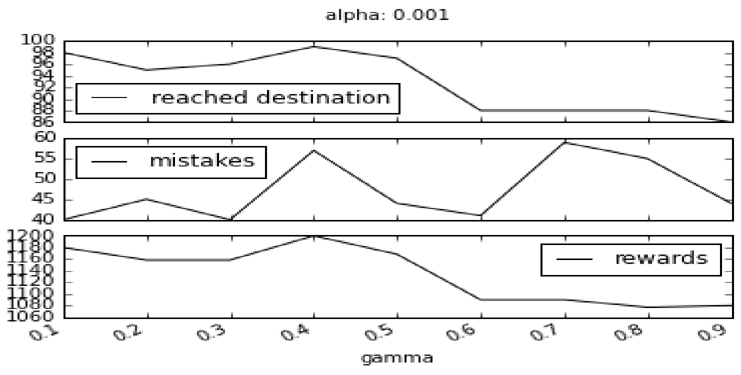
alpha	gamma	#times reached on time	#trials where agent made mistake
1	1	22 of 60 trials	

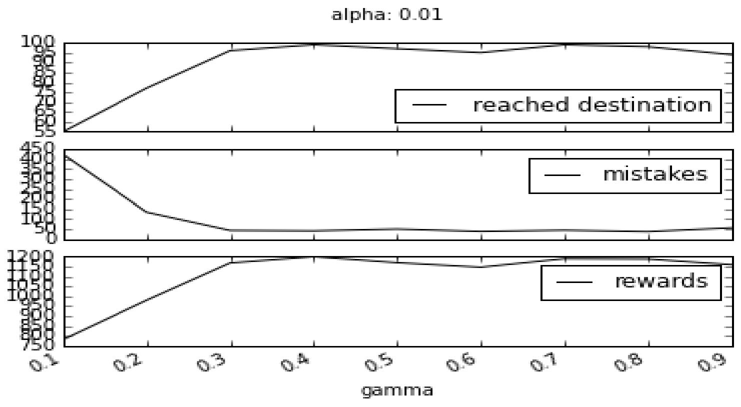
Basic Q-learning algorithm without learning rate and discount factor.

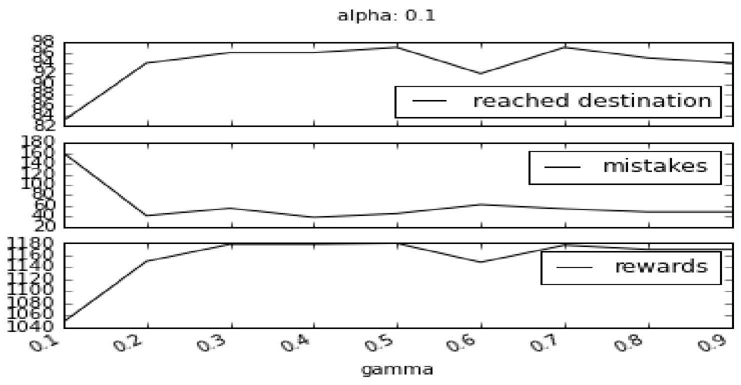
Runs	alpha	gamma	reached destination	mistakes	rewards
0	1.000	0.1	93	101	1118.0
1	1.000	0.2	94	77	1150.0
2	1.000	0.3	92	81	1115.5
3	1.000	0.4	91	69	1118.5
4	1.000	0.5	90	61	1117.0
5	1.000	0.6	86	45	1078.5
6	1.000	0.7	83	72	1079.5
7	1.000	0.8	80	100	1025.5
8	1.000	0.9	57	273	789.5
9	0.100	0.1	83	161	1048.5
10	0.100	0.2	94	41	1150.0
11	0.100	0.3	96	55	1178.5
12	0.100	0.4	96	38	1178.5
13	0.100	0.5	97	45	1180.0
14	0.100	0.6	92	62	1148.5

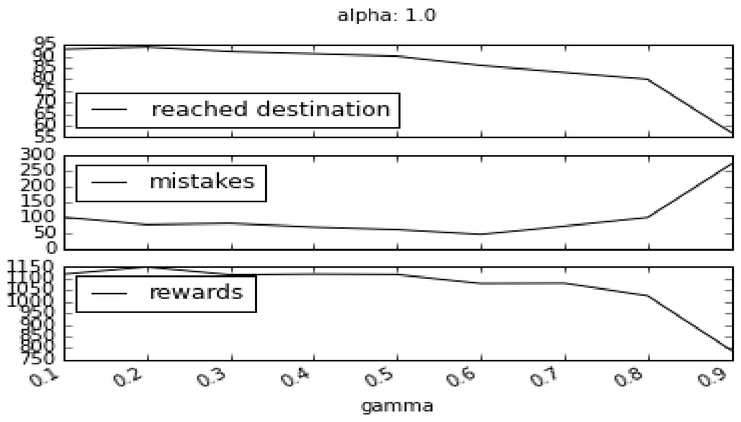
15	0.100	0.7	97	54	1177.0
16	0.100	0.8	95	48	1170.0
17	0.100	0.9	94	48	1170.0
18	0.010	0.1	55	425	781.0
19	0.010	0.2	77	135	979.5
20	0.010	0.3	96	43	1168.5
21	0.010	0.4	99	41	1200.0
22	0.010	0.5	97	50	1170.0
23	0.010	0.6	95	38	1147.0
24	0.010	0.7	99	44	1190.0
25	0.010	0.8	98	37	1188.5
26	0.010	0.9	94	56	1160.0
27	0.001	0.1	98	40	1180.0
28	0.001	0.2	95	45	1158.5
29	0.001	0.3	96	40	1158.5
30	0.001	0.4	99	57	1200.0
31	0.001	0.5	97	44	1168.5
32	0.001	0.6	88	41	1090.0
33	0.001	0.7	88	59	1090.0
34	0.001	0.8	88	55	1077.0
35	0.001	0.9	86	44	1080.0

Plotting the number of times the agent reached the destination on time and the number of mistakes it made in 100 trials for every value of gamma below.









As can be seen above, with the basic Q-learning algorithm, the agent failed to reach the destination on time most of the times. With enhanced Q-learning and different combinations of parameters alpha and gamma, the agent had varying levels of performance in terms of the time it takes to reach the destination and the number of trials in which it makes mistakes. Latter measure is important because it should be minimum and is a reasonable expectation from a driving agent.

Looking at the graphs above, I feel that alpha = 0.01 and gamma = 0.4 as the ideal parameters for the agent as it looks like it reaches the destination in time the maximum number of times and makes lesser number of mistakes compared to other parameters. It also earns the maximum reward.

Final agent performance

The optimal agent should reach the destination without committing any mistakes consistently on time over time. In optimal conditions, it follows the directions given by planner all the times if the traffic conditions are totally favorable which is not the case always. But when the traffic conditions are not favorable, the agent should take the long routes and reach the destination.

So, making sure that the agent reaches the destination on time consistently is a reasonable measure of the performance of the agent. Looking at the results above, with optimal parameters the agent could reach the destination 99% of the times. The agent is sub-optimal as it can be seen that it makes mistakes and also doesn’t reach destination on time 100% of times. Looking at the rewards, the agent doesn’t seem to loop around just to collect more rewards. In fact in most cases, it is the opposite. Whenever the agent made more mistakes, the reward was less because of the penalty incurred. We won't arrive at a completely optimal agent because the system is inherently stochastic. But, we can be sure that the agent will behave reasonably optimal given a chance to visit all the states infinite/large number of times.