Implement a Basic Driving Agent

Although the driving agent is able to move around in the environment, it does so randomly by choosing one of the possible actions (`None`, `'forward'`, `'left'`, `'right'`). With `enforce_deadline` set to `False`, it's correctly disregarding the deadline.

It is sometimes able to reach the destination (purely by chance) during some trials, but there's no pattern i.e. it's able to reach the destination during some of the early trials (w/message: *Primary agent has reached destination!*), but not during some of the later ones (*Primary agent hit hard time limit (-100)! Trial aborted.*) – indicating the agent is not "learning" from its movement in the environment.

Also even with seemingly correct actions, it's collecting negative rewards:

> *inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = right, reward = -0.5*
>
> *inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = left, reward = -0.5*

With certain actions, it's collecting positive rewards as well.

> *inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = right, reward = 2.0*
>
> *inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = right, reward = 2.0*

Inform the Driving Agent

Based on the environment the driving agent is operating in, the following variables were used to model the states/actions:

States:

- Traffic Light: *['green', 'red']*
- Next waypoint: *[None, 'forward', 'left', 'right']*
- Oncoming Traffic: *[None, 'forward', 'left', 'right']*
- Traffic to the Left: *[None, 'forward', 'left', 'right']*

Actions: *[None, 'forward', 'left', 'right']*

With the above variables, the agent is able to learn *Right-of-way* rules under 100 trials and reach the destination - safely, within time and reasonably maximizing the rewards received during the course of actions it took.

The following were considered, but not included to restrict size of the state space (minimally sufficient):

- Deadline: Including this variable may have an undesirable effect of forcing the agent to take some actions compromising safety of passengers. Also including this variable would expand the state space largely.

- Traffic to the Right: Turns out that the agent is able to learn and take actions as effectively without accounting for this variable, given the nature of the environment/problem it's operating in.

Implement a Q-Learning Driving Agent

The initial model was setup as follows –

- All states/actions were started with a value of 0
- 'Gamma' value of 0.5 (50% discount)
- Of the various possible powers for alpha=1/t, it was initialized with power 1 i.e. alpha=1/t
- To start with, no random actions were taken (i.e. 'epsilon' ignored)

With the above setting, the agent was able to reach the destination within deadline little over 50% of time in 100 trials. It accumulated more negative rewards during initial runs with almost all positive rewards in the last 10 trials. With only about 50% success rate, results indicated that the agent is possibly getting stuck in local minima, although it was not accumulating much negative rewards as the number of trials increased.

So to take care of the local minima issue, some randomness with some small 'epsilon=5%' was introduced to force the agent to take some random actions and explore more states/actions, as one of the critical points about Q-Learning is that the agent needs to visit all or as many as possible states/actions pairs many times for it to properly understand & exploit the tradeoffs.

With the above randomness introduced, the agent was able to reach the destination over 90% of the time within deadline, especially during the last 10 or so trials – indicating it's able to get unstuck & has learnt how to navigate the rules and take appropriate actions trying to maximize positive rewards.

Compared with the agent without Q-Learning implementation, the agent now is able to exploit its learnings by trying to receive as many positive rewards as it can. The earlier agent was just taking random actions without taking into account the rewards or inputs it received.

Now that the agent was operating in Q-Learning environment, with enough learning i.e. trials, it should converge to a policy in line with an optimal policy.

Improve the Q-Learning Driving Agent

My "grid-search" to uncover optimal parameters involved tuning the following parameters:

- An appropriate power/exponent for the learning rate 'alpha' in the range {0.4, 0.6, 0.8}
- Restricting 'epsilon' value, so the agent does not take too many unwanted random actions {5%, 10%, 20%}
- Toying with discount factor 'gamma' (to see how it might affect learning)

As expected, of the above, the first 2 parameters had an impact on agent behavior.

The table below shows performance trade-off of the agent in the last 20% of trials (of 100) - so the optimal parameters are gauged on a reasonably learned agent:

	alpha power = .4				alpha power = .6				alpha power = .8			
	% success	avg steps	pos rwrds	neg rwrds	% success	avg steps	pos rwrds	neg rwrds	% success	avg steps	pos rwrds	neg rwrds
epsilon=.2	80	11.2	18	3.3	85	14.05	20.15	3.7	90	11.65	20.82	3.05
epsilon=.1	95	13.2	25.07	3.2	90	12.55	18.95	1.75	85	13.25	19.97	1.35
epsilon=.05	95	12.5	22.12	1.6	85	10.85	18	0.8	100	11.55	22.32	1.45

As shown highlighted above, with an alpha (learning rate) of 0.8 and epsilon (randomness) of 5% seems to produce the best outcome. Clearly 'avg steps', 'pos rewards' & 'neg rewards' are not the best for the highlighted, but to me, consistent '% success' is more important than the other metrics (as long as the other metrics are not way too off). Also the negative rewards the agent is picking up seems to be due to its random actions and it's not picking up more than -0.5 in the last 5 trials.

Although the above results with the said settings were obtained on a single run of the agent with 100 trials - with more trials, the results would be consistent (if not exactly the same).

Also it's clear that the agent still needs to visit far more state/action pairs to be able to find the optimal policy and that it has not yet converged, but is in-line with an optimal policy – given it's able to achieve a good '% success' value consistently.

Optimal Policy

In theory, some of the important characteristics of an optimal policy would be where an agent:

- Follows traffic rules correctly and does not accumulate <u>any</u> negative rewards
- <u>Always</u> reaches the destination within deadline

In view of the above, in my implementation with the identified optimal settings (discussed above), the agent is consistently able to

- ❖ Collect minimal negative rewards as trials increase - indicating it's learned the traffic rules (the ratio of positive to negative rewards in the last 20 trials with sample 3 runs was roughly 20:1 and in last 10 trials it's ~25:1). The negative rewards it's collecting is possibly due to the random actions it's taking, especially towards the end of trials
- ❖ Reach the destination consistently as number of trials increase; in fact, of the 100 trials, it invariably reaches the destination in the last 10 trials

The above behavior is seen with a number of sample runs, not just a few.

Another metric for optimal policy would be - an agent reaching the destination in lowest possible time by taking the best possible route. I have not given much importance to this metric, because to me, the first two metrics are very important - that the agent is able to consistently reach the destination within deadline and obeys traffic rules to the most part.

Q-value convergence **(vis a vis Optimal Policy convergence)**

In theory, the Q-value being converged to comes from a huge number of inputs:

- Possible rewards for a state/action pair

- Possible sequences of states to come after
- Recursive combination of these things

In practice, the actual Q-values do not converge before a very large number of trials. Also, in order to get convergence, *alpha* needs to decay over time - so that as time goes by, the agent is more & more confident in its learning/actions and relies less & less on its environment to learn.

So more often the focus is finding an optimal policy (i.e. a policy that is in line with an optimal one) than to make sure that Q-value is converged.