Implement a Basic Driving Agent

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

ANSWER:
hen set the agent to take random actions, the SmartCab would drive around randomly. Most of the time, it does not reach to the destination. But very rarely, eventually, it would get lucky and it would make it to the destination. But it appears to be completely random and due to luck. Out of 100 trial simulation runs, the success rate of delivering customers due to random actions varies in the range of 0.16 ~ 0.24. It also appears that the SmartCab would sometimes break transportation rules and run into accidents, but it does not seem to learn from mistakes. Its behavior does not seem to improve even after receiving penalty, nor does it appear to responding to rewards.

Inform the Driving Agent

QUESTION: What states have you identified that are appropriate for modeling the SmartCab and environment? Why do you believe each of these states to be appropriate for this problem?

ANSWER:
The self.state [self.state=(inputs['light'], inputs['oncoming'], inputs['left'], inputs['right'], self.next_waypoint, deadline_info)] attribute for LearningAgent is set to be taking the following four conditions into consideration as for the purpose of modeling the SmartCab and its environment: 1) the transportation light (green or red); 2) the traffic situation at the intersection (whether or not there is traffic in the oncoming, left, and right directions); 3) the planned next way point; 4) a categorized parameter indicating how close it is for SmartCab to run out of time and hit deadline. The overall goal of one SmartCab trip is to get to the destination ***without violating transportation rules nor running into accidents as fast as possible****, hence the state of transportation light of an intersection is important state information to inform the LearningAgent so that it could learn not to violate transportation rule (would receive negative rewards if it violates). The information regarding to traffic state around the SmartCab and its next way point is crucial for teaching the LearningAgent to drive safely. The agent's variable [deadline] is another factor which is potentially useful state information, however it was not added into forming the LearningAgent's self.state attribute. Because adding [deadline] expands the number of self.state by a factor of 20~30, which will make it more difficult for the LearningAgent to learn. The solution to the MDP problem asks the algorithm to visit every states infinitely often, given the setting of the number of simulation trial runs (~100), a relatively smaller size of self.state is more desirable. Hence the original deadline variable (with a possible value range of -1 ~ 30) is categorized into three categories: urgent - coded as 1, representing deadline < 10; okay - coded as 2, representing 10 < deadline < 20; not urgent - coded as 3, representing deadline >= 20.*

QUESTION: How many states in total exist for the SmartCab in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?

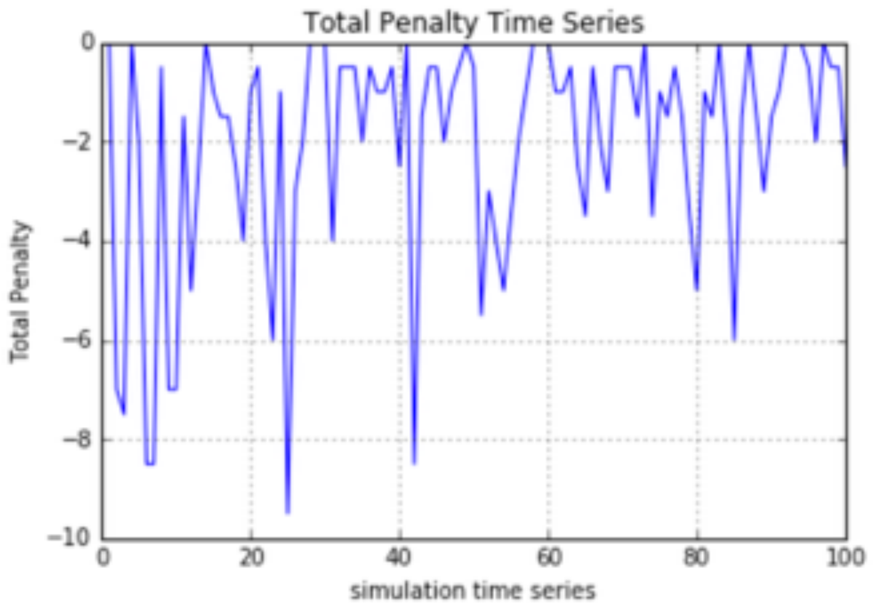
ANSWER:
There are 96 states in total exist for the SmartCab in this environment. This number seems reasonable for Q-Learning to learn and make informed decisions about each state. If set the trial run number of the simulation to be 100, and assume that for each trial simulation run, the SmartCab takes 20 actions and experiences 20 states on average before it terminates (either because of successfully reach the destination or trip aborts due to run out of time), so the SmartCab is expected to experience ~2000 states in total. This number seems to be reasonably large enough to ensure that all 96 states would be appeared/seen so that Q-Learning could learn about.

Implement a Q-Learning Agent

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

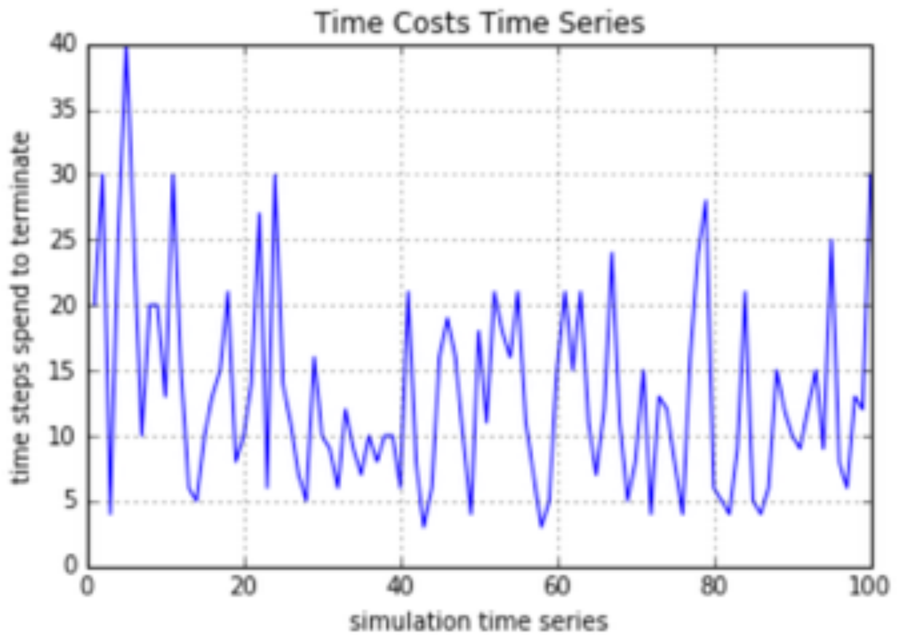
ANSWER:
*After implementing Q-Learning, the agent appears to be no longer randomly driving around, and it seems to be responding to the rewards and penalties that its receiving after taking certain actions while its at certain states. Its rate of delivering customer to destination before time runs out obviously has improved as compared to when it's taking random behaviors. The reason for this difference is that Q-Learning algorithm [Q(state, action) = Q(state, action) + learning _rate * (Immediate_Reward + discount_factor * argmax (Q(future_state, future_action)) - Q(state, action))] is teaching the agent by rewarding agent's correct behavior (positive rewards for taking the right action at the right state), and punishing its incorrect behavior (negative rewards if it violates transportation rule or runs into accidents).*

Specifically, it is observed that the SmartCab is more obeying traffic rules and makes less mistakes as compared to taking random actions. This is confirmed through plotting the time series of the total received penalty (negative reward) of the 100 simulation run.



As you can observe from the figure above, the total received penalty for each ride has a decreasing trend, and there are less and less strong negative penalty.

In addition to the that SmartCab is violating less and less traffic rules, it also appears to delivery customers faster and faster. See the following figure depicting each ride's time costs time series for 100 simulation rides.



Improve the Q-Learning Driving Agent

QUESTION: 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?

ANSWER:

- *Tuned omega values: [0.7, 0.8, 0.9, 1.0, 2.0]. The learning rate is constructed using omega value: apha = 1/(t^(omega)). This parameter adjusts how much does the LearningAgent to learn: It is intuitive that the LearningAgent learns more from the rewards/penalties in the very beginning, then the agent learns less and less along the training evolves. Setting alpha/ learning rate to be too low may cause it takes too long for the agent to learn; setting the value to be too high may cause the agent to "forget" what it has learnt.*
- *Tuned gamma/discount factor values: [0.5, 0.6, 0.7, 0.8, 0.9]. This is a factor for controlling how much long term/future utility is taken into account for current state and action pair. If this factor sets to be too low, then the agent would only care about the immediate/current reward that its receiving. The consequence of taking too little consideration of long term/future utility*

could be that the learning agent would end up drive around too long to deliver the customer to its destination.

- *Tuned epsilon/exploration factor values: [0.1, 0.2, 0.3, 0.4, 0.5]. Exploration factor is set for aid with finding the global *optimal*/*maximum*. In plain language,along the training goes, it is possible that the learning agent finds out that it always gets a good reward by taking one specific action, hence it will always taking that action, which may cause it to stuck at a local optimal. Properly exploring some unexperienced options/actions, would allow the learning agent to move out of local optimal and then has a chance move to the global optimal. If this factor sets to be too big, then the agent may act a lot like the initial setting behavior, where let the agent take random actions on all conditions. The parameter tuning performance data indicates that when epsilon is set to be 0.5 or higher, the success rate drops significantly to below 0.3. (details see Report.ipynb)*

The performances regarding to different settings of tuned parameters are stored in data.csv. It is loaded and analyzed in the Q-Learning Parameter Tuning-Performance Analysis section presented in the Report.ipynb.

Since there were a lot of combinations (125) of parameters examined in parameter tuning, not all of their performances are reported here. A more detailed investigation regarding to all 125 sets of parameters are discussed in Report.ipynb. There are only 11 sets of parameters resulted relatively good performances are reported here.

Parameters (omega, gamma, epsilon)	**average success rate**	**average total penalty**	**average net reward**	**average time costs**
(0.7, 0.7, 0.1)	0.96	-2.690	23.035	13.99
(0.7, 0.8, 0.2)	0.95	-2.615	23.700	13.18
(0.7, 0.9, 0.1)	0.90	-3.145	24.450	17.57
(0.7, 0.9, 0.3)	0.90	-3.155	23.365	15.13
(0.7, 0.9, 0.4)	0.96	-3.210	23.585	13.70
(0.8, 0.5, 0.4)	0.96	-2.830	20.960	11.97
(0.8, 0.8, 0.3)	0.96	-2.785	21.990	13.23
(0.9, 0.8, 0.1)	0.97	-2.050	22.965	13.11
(0.9, 0.9, 0.1)	0.95	-1.790	21.210	12.66
(0.9, 0.9, 0.4)	0.88	-3.345	23.120	14.61
(1.0, 0.9, 0.2)	0.92	-2.830	23.355	15.07

Among all the 11 reported parameters' performances, there are two rows highlighted in shades appearing to performance better than the rest.

- *(0.9, 0.8, 0.1): achieved a slightly higher success rate and a slightly higher mean net reward score, however on average it takes slightly longer to delivery customers and its average total penalty is slightly higher*

- *(0.9, 0.9, 0.1): achieved a lower success rate, and its received mean net reward is smaller than the previous parameter setting; however it obeys traffic rules more and it takes shorter time to delivery customers on average.*

Given that obeying traffic rule and delivery customers as fast as possible is the criteria setting for optimal policy, pick the optimal parameter setting as ***omega = 0.9, gamma (discount factor) = 0.9, epsilon (exploration rate) = 0.1****.*
Using this parameter setting, the SmartCab achieved a success customer deliver rate of 0.95; it received an average penalty of -1.79, an average net reward of 21.21 during the 100-trial-run simulation. Its average time costs for delivering customer is 12.66 time step.*

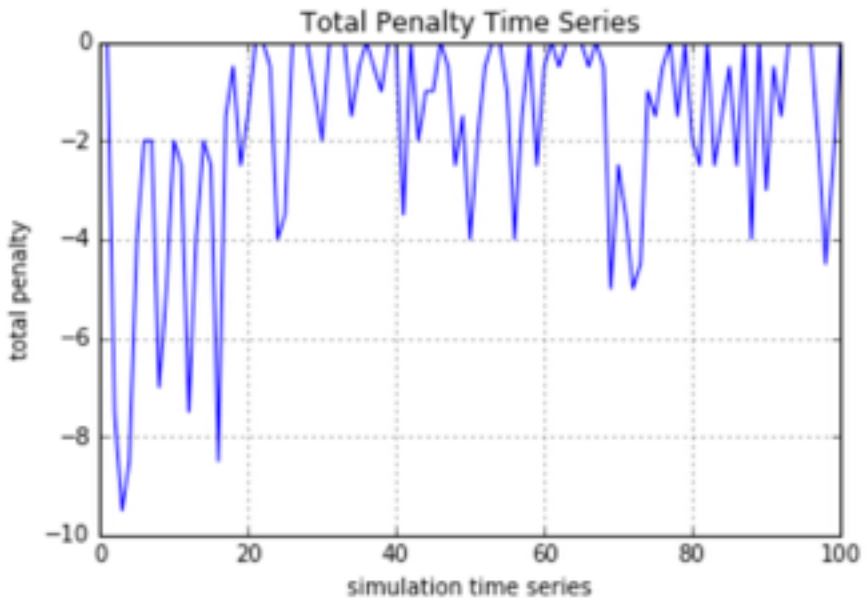
*(*Note that the performance of the optimal parameter setting reported here is the output based on one specific parameter tuning simulation run. Due to the stochasticity of some other factors in the game, different runs would result slightly different output. Various results of simulation runs using above optimal tuned parameter setting suggest that the success rate regarding to customer delivery before time runs out varies from 0.85 ~ 0.95, with a commonly observed success rate in range of 0.88 ~ 0.93)*

QUESTION 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?

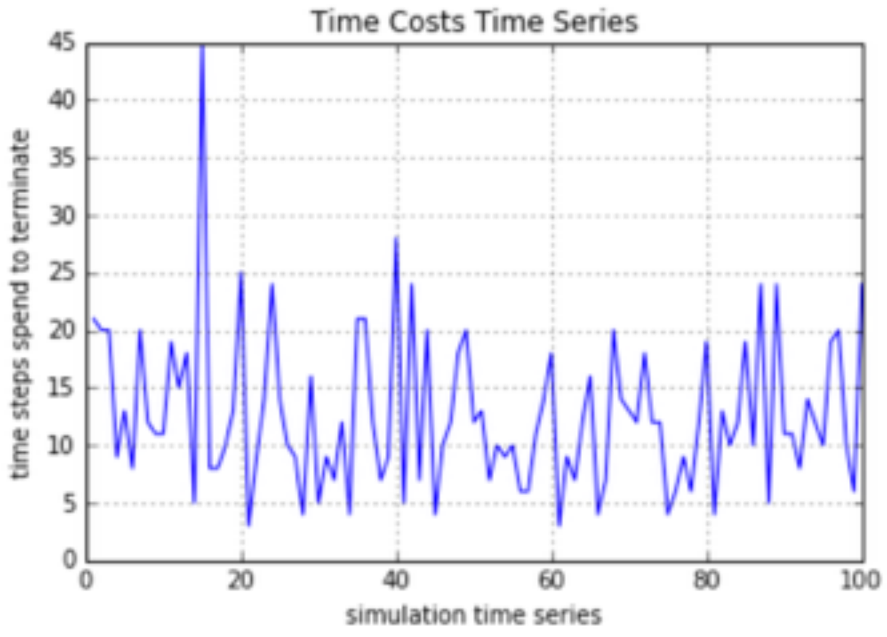
ANSWER:
An optimal policy for this problem would be a policy which produces a series of optimal actions for each state that the SmartCab is in, and this series of optimal actions would guide the SmartCab to accomplish the mission (delivery the customer to its destination) in the shortest time step possible with no violation of traffic rule and no occurrence of accidence (hence results in low penalties and high rewards).

Some visualization regarding to the time series performances of SmartCab trained by the picked optimal policy is shown as following:

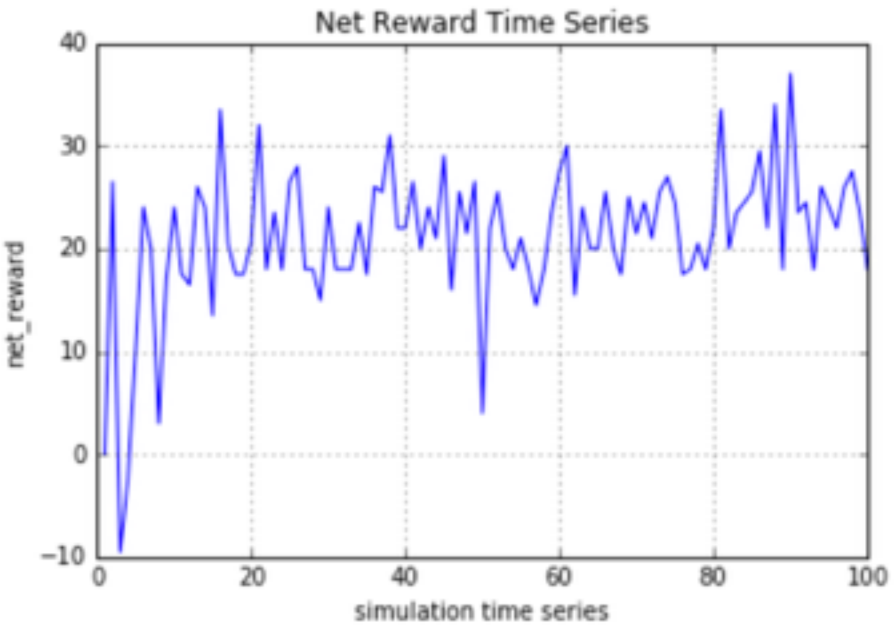
- *Total Penalty of Each Ride Time Series for 100 Simulation Trial Rides/Runs*



- *Time Costs of Each Ride Time Series for 100 Simulation Trial Rides/Runs*



- *Net Reward of Each Ride Time Series for 100 Simulation Trial Rides/Runs*



As shown in above three plots, the LearningAgent appears to ***get close*** *to finding an optimal policy which has a high customer delivery rate in relatively small time costs with high net reward and low penalty punishments. When the simulation runs just started, the LearningAgent violets traffic rules a lot and hence its total penalty per ride oscillates in the range of -2 ~ -8. After about 20 rides of learning, it's becoming better and the total penalty per ride oscillates around 0 ~ -4. A*

lot of times it even does not make mistakes and results a penalty of 0. The time cost time series only shows ***a mild decreasing trend****. The net reward time series pattern tells a consistent story with the total penalty time series: such that the SmartCab started with poor performance, but it began to perform well, make a few mistakes and receiving high rewards after about 20 rides of training.*
However, given that at the end of the 100 simulation rides, the SmartCab still makes mistakes occasionally/violates traffic rules occasionally, and there isn't an obvious decreasing trend in time costs to deliver a customer, there are potentials in further improving the SmartCab's performance. For instance, with allowing to modify reward distribution and adding reward weights on "reach the destination" may encourage the LearningAgent try to deliver the customer faster. Increasing penalty weights for behaviors which break traffic rule, could possibly further improve SmartCab's performance regarding to obeying traffic rules. Although there are caveats in adjusting reward system, cause inappropriate rewarding system could still result bad SmartCab performances as well.