SMART CAB PROJECT

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?*

Giving the smart cab random actions to take allows it to move around the city with no consideration for traffic or rules. Given enough time, I did observe that the smart cab could make it to its destination, but not without acquiring penalties, most of the time, violating traffic laws and causing accidents.

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?*

A state for the smartcab is a four parameter tuple, consisting of 1) the current traffic light status, 2) the next waypoint 3) direction of traffic coming from the left and 4) the direction of oncoming traffic straight ahead. It was determined that the traffic coming from the right is not required since the dummy agents will always obey the traffic laws. This also helps reduce the feature space. The traffic light status is required since it is an important traffic law not to violate. The next waypoint is required since our agent needs to know the direction it should be taking instead of randomly guessing. The last two parameters are required because we would like our agent to learn to avoid collisions, maybe even more so than traffic light violations. I also experimented with including the deadline parameter as part of the state space. However, if became apparent very quickly that the agent could not possibly learn enough within 100 trials. This is because the feature space of the states greatly increases with the addition of the deadline parameter. Generally, we would like 10 times the amount of data as dimensions and with the additional deadline parameter our agent will suffer from the curse of dimensionality. Therefore, it was decided to exclude this parameter from the state space so it more reasonable for the agent to visit each state within 100 trials.

OPTIONAL: *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?*

Given the four different state parameters as listed in the question above, there are a total of 96 possible different states. Two options for the traffic light (red or green), three options for the next waypoint (forward, left, right), and four options for the two remaining parameters (None, Left, Right or Forward) results in 96 different states$2 \times 3 \times 4^2 = 96$. This does seem to be a reasonable number of states for the given problem. It is, theoretically, possible to hit all states within 100 or 1000 different trials and be able to learn from them. However, if we had included the traffic oncoming from the right we would have increased our feature space from 96 to 384 ($2 \times 3 \times 4^3 = 384$), which seems more unreasonable and might be harder to hit all possible states. This is the same reason why I did not include the time remaining into the state.

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?*

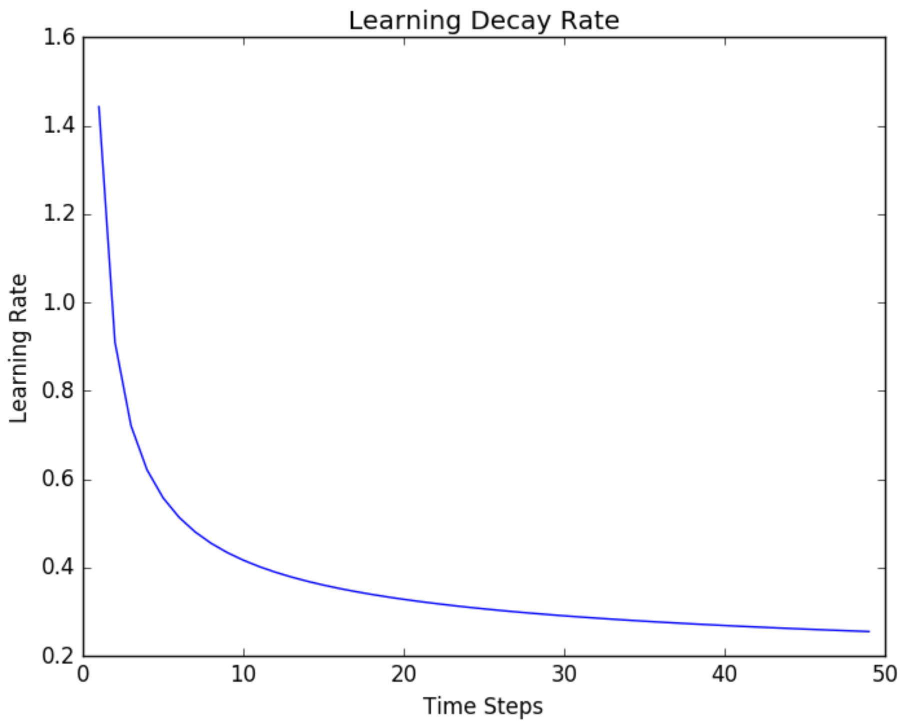
The biggest change noticed after the implementation of Q-learning was the driving agent's ability to quickly learn the traffic rules. After about the second trial the agent was able to avoid most traffic violations and accidents. Shortly after that the agent became faster at arriving at its destination. Since the initial value of epsilon (greedy exploration) is set to 1.0 the first several actions chosen are random and therefore appear similar to the first implementation. Shortly after though the agent begins to see states it has previously seen and is more likely to choose a "better" action than before unless that initial random action was the correct choice. Also, initially, with a larger alpha the reward received by the agent for an action is more heavily weighted than it is later in the process.

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?*

Initially, all values from 0.1 to 1.0 were tested for the learning rate (α) the driving agent began with a constant value for the learning rate, an optimal value of 0.2 was used and the agent was able to reach its destination roughly 97% of the time. Then I implemented a learning decay that decreased the learning rate logarithmically proportional to the number of times the update function was called. After each update the learning rate was decreased by the equation:

$$\alpha = \frac{1}{\ln(\text{steps} + 2)}$$

Where, α is the learning rate and steps is the number of times the update function was called. It is important to note that the steps counter was not reset after each trial ended. This prevented the model from resetting the learning rate back to 1. The graph below shows the rate at which the learning rate decayed.

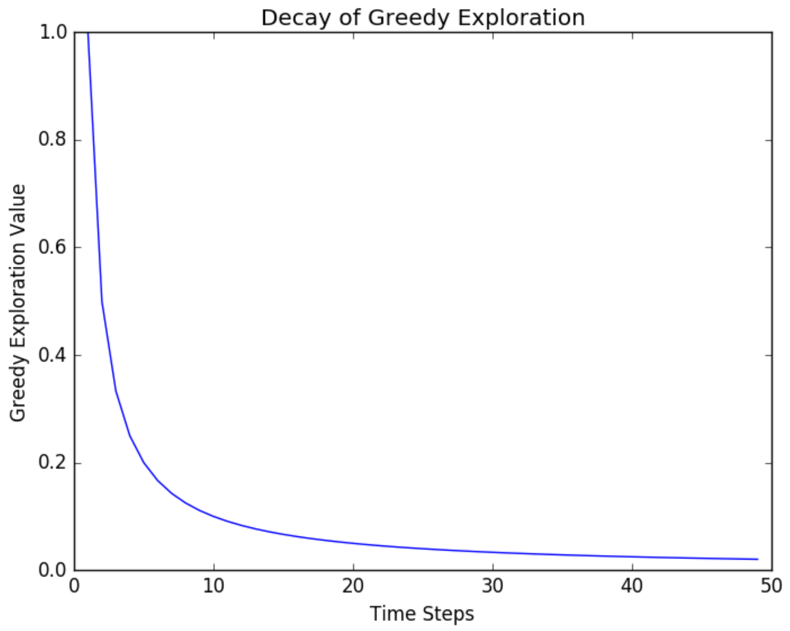


Since the model did prove to be learning relatively quickly the initial weight of alpha needed to be large so the reward value was weighted heavier. Then the learning rate could approach the optimal value initially found earlier of 0.2. As more time went on the reward value did not have as much effect on the learning rate as the previous Q-value.

The agent does also implement an epsilon value or greedy exploration (ε). This can help prevent the agent from getting stuck in a local minima or holding onto a poor Q-value when a better option does exist. Initially the value of epsilon is set to 1.0 and this forces the agent to make a random choice on the next action. However, since the agent appeared to be learning very quickly it was important that this value decayed faster than the learning rate. The equation below shows the decay rate for epsilon.

$$\varepsilon = \frac{1}{\text{steps} + 1}$$

Where, epsilon is the greedy exploration value and steps in the number of times the update function was called. The graph below shows the decay rate of the greedy exploration value.



As seen by the graph, after the first few runs the probability of choosing a random action greatly decreases and will continue to decrease just at a slower rate. The value for gamma was optimally found to be around 0.85 or 0.9 and was left constant since the decay of the learning rate put less weight on the value of gamma.

The driving agent learned very well by this model and appeared to not have a problem discovering an optimal policy. The agent was able to reach its destination 99.8% of the time or 998 trials out of 1000.

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?*

The designed agent is able to find an optimal policy, toward the latter part of the 100 trials the agent is able to arrive at the destination every time while not incurring any traffic violations or collisions. The more trials the agent has to learn the better it becomes. The optimal policy for this problem is to 1) always guarantee the agent makes it to the destination in the allocated time, 2) the agent always avoids collisions with other agents and 3) the agent obeys the traffic laws. This is done by minimizing travel time and maximizing reward. The final optimal policy does not normally perform unusual actions. However, it is possible that the agent would violate a traffic law in order to make a deadline. The agent would rather receive a small penalty for the violation and make the deadline (a large reward). Also, since it is possible that not every one of the 96 states is visited by the end of the 100 trials it is still possible for the agent to make a mistake. A log capture file has been added to the agent.py file to report any penalties that are received in the last 10 trials. Ideally, and in most of the different runs performed, there were no penalties reported. It is possible to view areas where the agent did not find the optimal policy by viewing this log file after the completion of the run. It is important to note though, that with more trials ran the chances of receiving a penalty decreases, since the agent had more trials to learn from.

The table below shows an example where there were three penalties were received between trial 90 and 100.

Inputs	**Action**	**Next Waypoint**
'light': 'green', 'oncoming': 'right', 'right': None, 'left': None	Right	Forward
'light': 'red', 'oncoming': None, 'right': None, 'left': 'forward'	Left	Forward
'light': 'green', 'oncoming': None, 'right': None, 'left': 'forward'	Right	Forward

By the 90th trial the agent had only seen a total of 27 different states. The three states above had either never been seen by the agent before or the agent had not yet had a chance to learn the optimal policy. Meaning that the Q-value for the different actions was most likely the same value and a random choice was made for selecting the "best" action. Again, the more trials ran the less penalties the agent received.