Project 4: Train a Smart Cab to Drive

Sumedha Swamy

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?

Solution:
As seen in Figure 1, when the agent takes random actions, it sometimes reaches its destination (in ~40% of the trials). The reward is frequently negative and the algorithm frequently runs out of time before the agent reaches it destination.

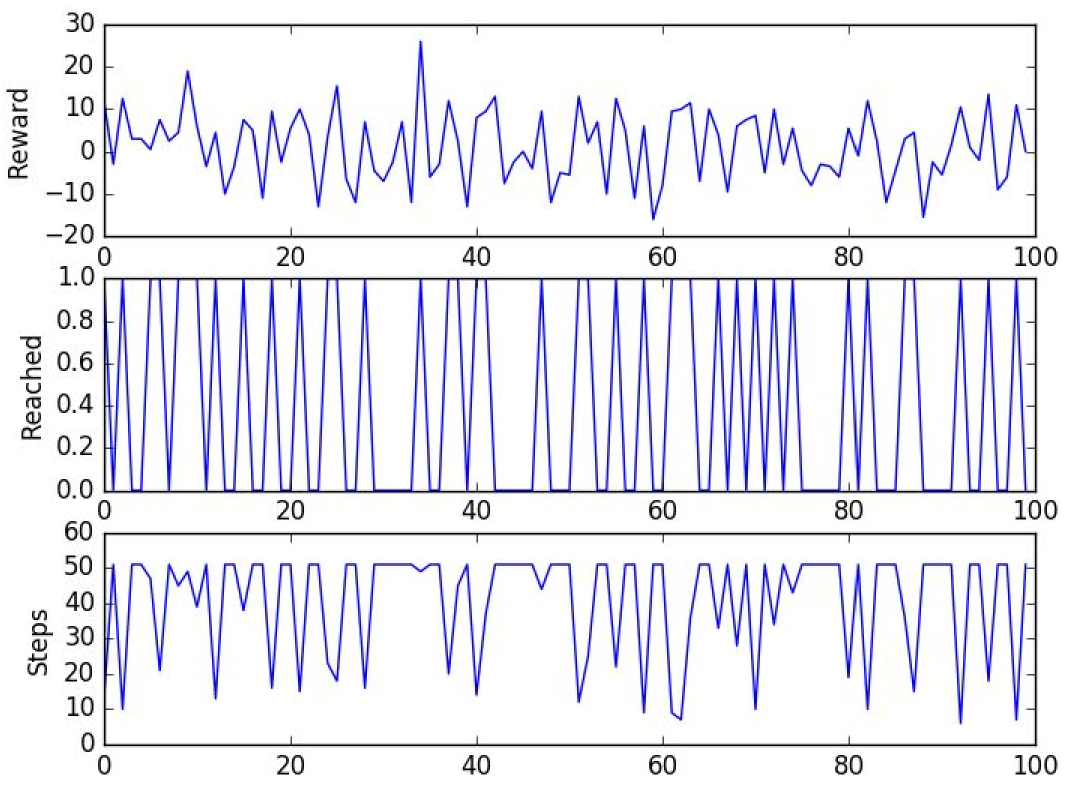

Figure 1: Random Actions

Identify and update state

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? 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?

Solution:
The following states are appropriate for modeling the smartcab:

- Traffic light at the intersection that the agent is in ['red', 'green']
- Oncoming Traffic [None, 'forward', 'left', 'right']
- Traffic from Left [None, 'forward', 'left', 'right']
- Traffic from Right [None, 'forward', 'left', 'right']
- Route Planner’s suggestion for the action toward the ideal next waypoint [None, 'forward', 'left', 'right']

The goal of this project is to help the agent learn the traffic rules and to help guide it to the destination in the fewest possible steps. Therefore the inputs that determine a state must comprise of

- Traffic situation (Light, oncoming, left and right) at any given point of time
- Ideal next action that results in the shortest path towards the destination. Many times, the shortest path suggestion may be in violation of traffic rules. It is important to capture these mistakes as a part of the learning. Therefore, the ideal next action in conjunction with the traffic situation together determine the states to model the smartcab.

Implement Q-Learning

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?

Solution:
As the Q-Learning algorithm progresses, the algorithm updates the Q-table with the q-values corresponding to each of the actions [None, 'forward', 'left', 'right'] in every traffic state the agent encounters. An agent may receive a negative score for taking a certain action in a given (traffic situation, ideal next action) pair. When the agent encounters the same situation

again, it will likely avoid the same mistake and choose an alternate action. Thus, the agent is learning the best action to take in every situation it progressively encounters.

Therefore, over time, the agent makes more correct than incorrect moves - making it gain higher rewards and helping it reach the destination in fewer steps.

Enhance the 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?

The parameters I used to tune the algorithm are:

- Discount Factor: gamma [0, 0,3,0.5, 0.7, 1]
- Learning Rate: alpha (Decaying from 0.45 to 0.99 exponentially with increasing trials)
- Exploration Rate: epsilon (0.005, 0.01, 0.1, 0.5)

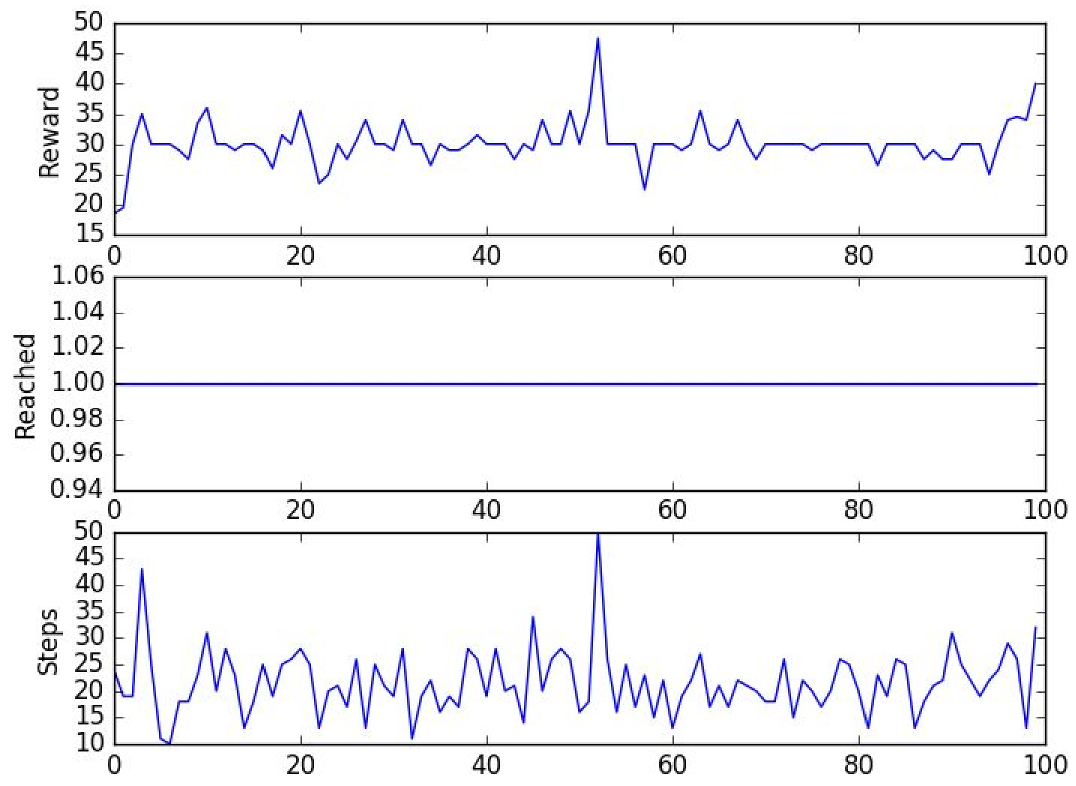

Figure 2: Agent performance thru Q-Learning

- **Epsilon:** With epsilon = 0.5, the agent only reached the destination in ~75% of the trials. An epsilon value of 0, 0.005 and 0.1 did not change the outcome noticeably. Therefore, I chose the value of 0.1 to allow for exploration without compromising on the outcome and reward.
- **Gamma:** In this example, increase in value of gamma very strongly correlated with more negative results. I therefore chose to set the value of gamma = 0. On further thought this outcome makes sense. *gamma* indicates the extent to which the future actions will in the next state will contribute to the learning in the current state. In this project, the future state constantly varies for any given current state. Therefore, learning on the randomness of the future is not as useful as learning on the reward on the current state.
- **Alpha:** I chose alpha to decay over time (thru an exponential function). I chose the values such that the algorithm learns significantly early on and progressively reduces the learning rate. I chose values decaying from 0.45 to 0.99 exponentially with increasing trials.

As seen in Figure 2, with the above optimizations implemented on Q-Learning, the agent performs significantly better - reaching the destination in the allotted time in every trial. We can also observe that the total reward quickly ramps up and hovers around 30. Note that I have modified the start/destination selection to choose points that have an L1 distance of 10.

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?

An optimal policy in this problem is one where the agent incurs no penalties over time and reaches its destination in the allocated number of steps. As seen in Figure 3, while the agent definitely incurs significantly less penalty over time, we also see that even close to 100 steps, the agent still incurs a penalty. This agent needs to be trained further so that all the entire Q-table is fully filled with values - thus allowing the agent to perform in an optimal policy.

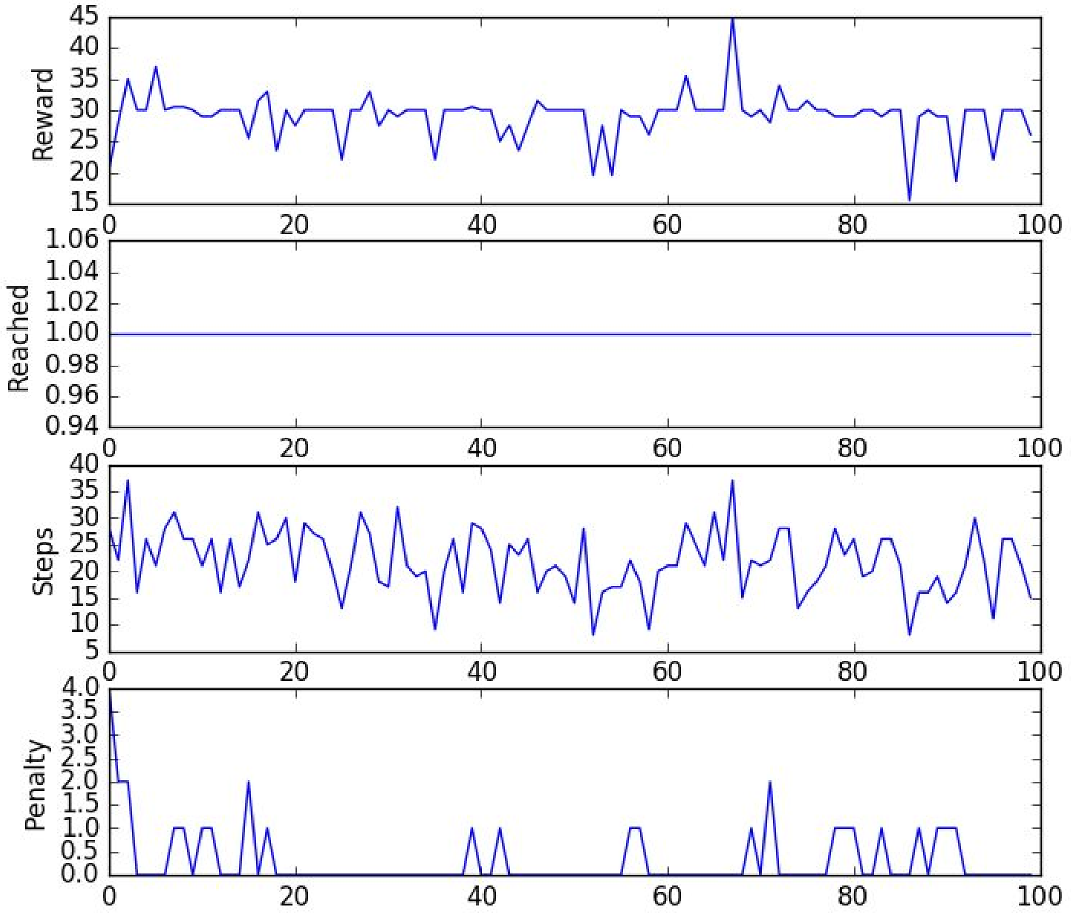

Figure 3: Penalty steps