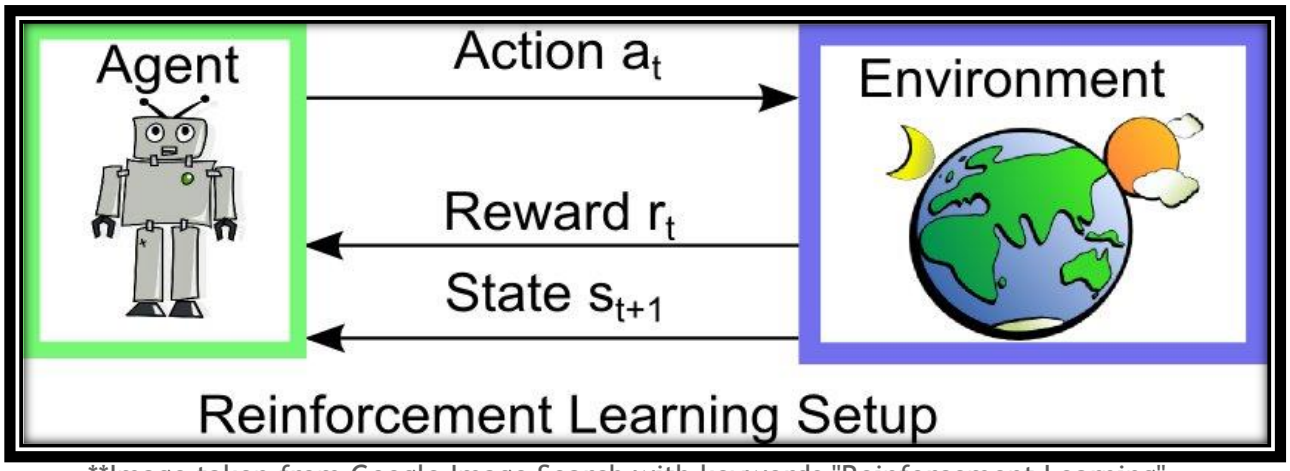

**Image taken from Google Image Search with keywords "Reinforcement Learning"

Project 4: Reinforcement Learning

Train a Smart-cab to Drive

Kiran Mohan

Udacity Machine Learning Nanodegree

07/31/2016

Reinforcement Learning
Train a Smart-cab to Drive

Question 1
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?

The agent has been configured to move randomly irrespective of any inputs from its environment. Hence, this is similar to a "random walk". In the limit, the robot would have covered all the state-action pairs. This means that even the goal state would be covered, in the limit. Hence, the agent does eventually make it to the destination. In fact, in a few runs of the simulation, the smartcab also reached the destination quite quickly. Although this random approach does eventually help the agent reach its destination, we can definitely do better. One important observation that opposes the use of this algorithm is that the smartcab does not obey the traffic rules and also causes a considerable number of accidents.

Question 2
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?

The states used for modeling the smartcab agent and its environment are as follows.

1. **'light'**

   This is the traffic lights observation as obtained from the environment. This is important for this problem as this input helps the agent to learn to follow traffic signals. Whenever the smartcab agent breaks traffic rules, it gets negative rewards (penalties) and hence, over time, the agent would learn that it should follow traffic rules.
2. **'oncoming'**
3. **'left'**
4. **'right'**

   The above three states indicate the status of traffic (other agents) in the proximity of the agent. 'oncoming' represents the next action that an oncoming vehicle would take in an intersection where the agent and the other vehicle meet. Similarly, 'left' and 'right' represent the next actions that any vehicle to the left and right of the agent would take in an intersection where the agents meet.
5. **next_waypoint**

   The next_waypoint state gives the action as decided by the planner that runs in the background. This is significant enough to include in the state as this is what we want the agent to learn. In an ideal case, we would like the agent to follow the next_waypoint exactly while also following traffic rules and causing zero accidents. Since the above 5

states capture exactly this information – nothing more, nothing less – they have been chosen to model the agent and its world.

There are totally 8 possible states that could have been considered – The 5 mentioned above along with deadline, location and heading. We did not choose these 3 to model the agent and its environment because the deadline would neither provide much information to the agent regarding its path to the goal nor inform the agent of any rules of the game, the location and heading are inherently integrated via the "next_waypoint" state as the planner takes the location and heading into consideration while preparing the plan. It is arguable that the deadline does in some sense relate to the rules of the game as the game ends when the deadline reaches 0. However, introducing this information increases the dimensionality and the state space of the problem as the deadline variable takes values from the set of natural numbers (of course, limited by some maximum). Even if this maximum was 100, our existing state space would have to be multiplied 100 times to accommodate for deadline. This would increase the training time required significantly.

Question 2 (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?

For the implemented state space, there is a total of 2*4*4*4*3*4 = 1536 states. This is the combination of all states in the Q dictionary. Given that there are 100 iterations that run and each iteration has an average of approximately 20 trials, there would be approximately 2000 updates in the Q table. This would ensure that the agent has learned sufficiently. However, the agent would still not have covered the entire state space within these 2000 updates. The coverage of all states in the state space would happen only when time tends to infinity.

Question 3

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?

When the agent starts out, it makes mistakes and seems to have a random behavior. However, the agent learns the rules of the game quite quickly from the information included in its states. The Q table is gradually built up and once the agent has confidence in the Q table, the agent performs quite well. The first 10 to 15 iterations seems to be the training period. Once this stage is crossed, the agent seems to consistently perform in accordance to the Q table, thus maximizing the speed to get to the destination while simultaneously minimizing the probability of causing accidents and disobeying traffic lights.

Question 4

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?

Trial No.	alpha	gamma	epsilon	Success Rate
1	1/t	0.3	0.3	71%
2	0.2	0.3	0.3	82%
3	0.2	0.2	0.3	80%
4	0.1	0.3	0.3	84%
5	**0.1**	**0.4**	**0.3**	**85%**
6	0.1	0.5	0.3	78%
7	0.1	0.4	0.2	83%
8	0.1	0.4	0.4	74%
9	0.05	0.4	0.3	78%

Table 1: Parameter Selection

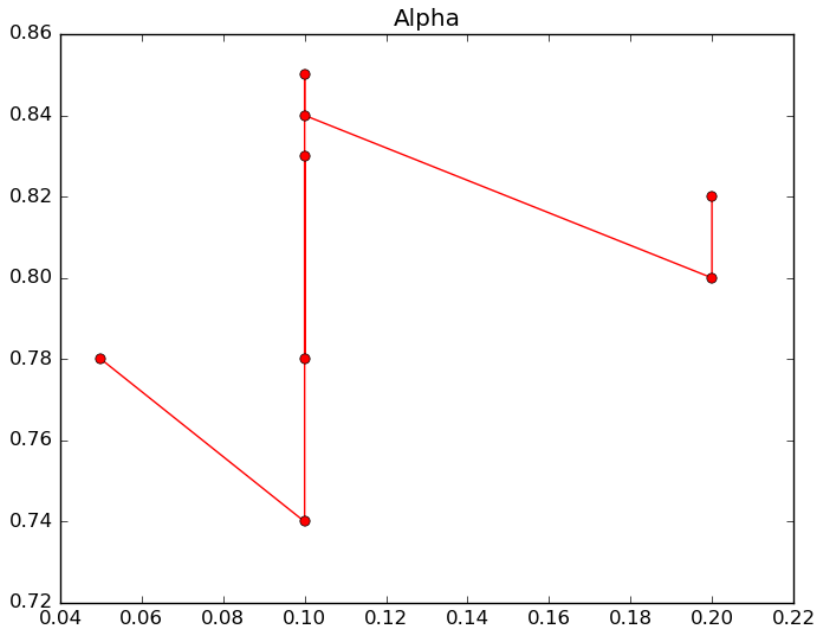

Fig 1: Plot of variation of efficiency with variation in alpha

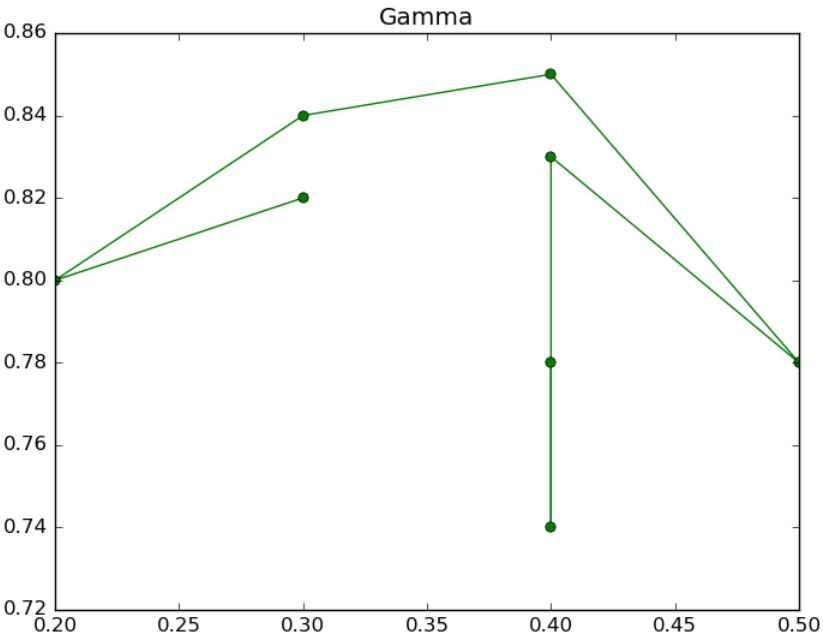

Fig 2: Plot of variation of efficiency with variation in gamma

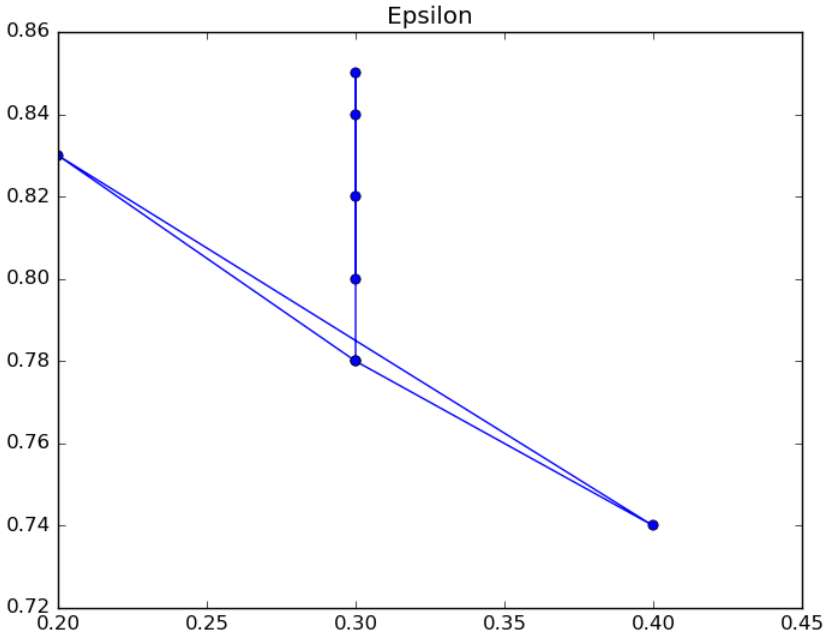

Fig 3: Plot of variation of efficiency with variation in epsilon

From the above table and plots, it is evident that the values,
Alpha = 0.1
Gamma = 0.4
Epsilon = 0.3
gives the best performance – 85% Success rate. It can be seen that increasing or decreasing any of the parameters causes at least a small drop in performance. However, the parameters in trial 4 and trial 7 are quite close in performance to the chosen parameter set.

Question 5

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?

Yes, the agent does get "close" to finding an optimal policy. It does reach the destination reasonably quickly and does mostly avoid penalties. However, since the algorithm used chooses random actions with a certain low probability to prevent getting stuck in local minima, these random actions do sometimes incur penalties. Hence, it is not perfect, but is very much usable. An optimal policy, as discussed earlier would be one which follows the next_waypoint perfectly while simultaneously obeying traffic rules and never causing accidents.