# Project Report – Training a Smartcab to drive

When I initially made the agent choose a random choice from the possible valid actions at that state, the cab would rarely make it to the destination, even when there is no deadline. Arrival at the destination was completely dependent on luck. The cab does not learn from its mistakes, nor does it try to go towards the destination. It merely picks a random choice, irrespective of its reward, and goes that way. Luckily, if it chooses a few correct actions, it gets to the destination.

```
action = random.choice(self.env.valid_actions) # Executes a random action and gets a reward
```

For modeling the smartcab, the traffic lights, oncoming traffic, the traffic to the left of the driver, and the action it takes are things the smartcab should be aware of.

a. Traffic lights: Traffic lights play a major role in controlling the cab's movement, and the cab will be crossing the intersection (unless it's taking a right) only if the light is green.
b. Direction of oncoming traffic: While the cab is trying to go straight at an intersection, there should be no oncoming traffic from the opposite direction trying to take a left. Only then can the cab safely cross. Also, if the cab wants to take a left turn, there should be no oncoming traffic trying to go straight through the intersection.
c. Directions of traffic flow to the left of the cab: The traffic to the left of the cab should not be going straight through the intersection, for the cab to be able to take a right during a red light.
d. Next waypoint: This option lets the cab know what direction it should go in next, so that it can get closer to its destination. This is one of the most important states as it helps the cab reach the destination as soon as possible.
e. Direction of traffic flow to the right of the cab: The traffic to the right of the cab is irrelevant when the smartcab is making a decision, as this traffic does not intervene in anyway with the smartcab, i.e., when the traffic signal is green for the smartcab, the smartcab gets priority and vehicles to the right will not come in the way. When the traffic signal is red, The only action a smartcab can perform is going right, which does not depend on how the right hand traffic is flowing.
f. Deadline: I have chosen to not consider deadline for the qlearning agent, as the smartcab is meant to travel from the source to destination, safely. As the deadline is approaching, it should not take decisions which would put the smartcab or the passengers in harm's way.

```
for i in ['green', 'red']:  #lights
  for j in [None, 'forward', 'left', 'right']:#oncoming
      for k in [None, 'forward', 'left', 'right']:#valid options
        for l in ['forward', 'left', 'right']:  ## possible next_waypoints
            self.Qtable[(i,j,k,l)] = [0.0] *4  # Q table, initialised every element to 0.
```

As seen in the above for-loop, the total states that exist for the smartcab is:

2 (traffic lights) * 4 (directions of oncoming traffic) * 4 (directions of traffic flow to the left of the cab) * 3 (waypoints) = 96 states.

Once the Q learning algorithm is used to train the smartcab as it drives, the cab starts becoming pretty smart, and starts to make right decisions as time progresses. It finds a safe, yet pretty short path to the destination it is intended to go to, almost always within the given deadline.

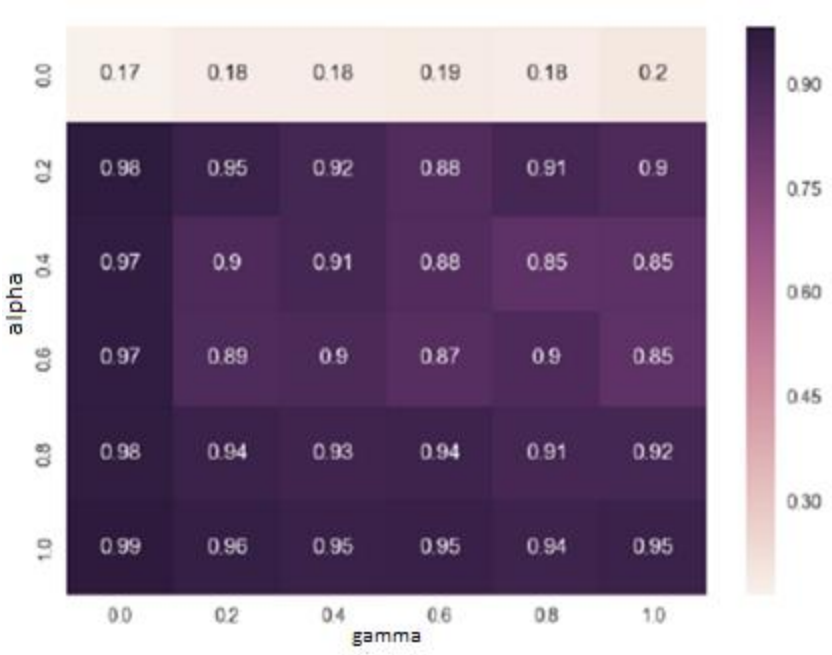
Based on past experience, the smartcab chooses the action based on how that particular action rewarded it previously. If there are more than 1 actions that have proven to be the best, the smartcab chooses an action randomly among the best actions.

These 'smart' decisions are made by the cab, as it learns from every move it makes, which therefore helps it make better decisions henceforth. This learning by the smartcab takes place with the help of Q-learning. A Q table stores the rewards it achieves for performing a certain decision, based on which it learns what the best decision would be, the next time it arrives at the same state.

```
epsilon=1/(self.timer*10);
c=list()
#as time tends to infinity, the cab probability of the cab taking a random action would tend to zero,
#If epsilon(t) > a randomly generated number, the below algorithm chooses a random action, so that it
# can explore options he have never tried before,
#which would hopefully help it discover better options that it has missed while learning.
if(random.random()>=epsilon):
    for i in self.Qtable[self.state]:
        if i==max(self.Qtable[self.state]):
            c.append(i)
else:
    for i in self.Qtable[self.state]:
        c.append(i)
action = random.choice(c)
```

In the above code snippet, you can see that there is always a small chance, reducing inversely with respect to time of the smartcab making a random decision, which is really helpful, especially when there are paths never explored by the smartcab because there were other options which were good enough, like not moving during a red signal, when compared to taking a right during a red signal, which could take it closer to its destination.

The tunable values – alpha and gamma does not seem to have a varying effect on the result, as its values are varied. Irrespective of its value, they always seem to help the cab reach the destination within the given deadline atleast 96-99% of the times, and even 100% at times. These results are with respect to alpha considered as 1/trial*x, where x is tuned to different values. As long as alpha is not 0, the smartcab will almost always reach its destination, as seen by the heatmap below:




This happens because the cab either gets a positive value for a good action, or a negative value for a bad action. Irrespective of how much (alpha) it learns from this action, the positive value will always be greater than the negative value and hence from next time onwards it will always choose this action. The same thing applies for gamma. Irrespective of how valuable the next state action pair is, it will always know that a positive value is better than a negative one, and will be able to differentiate between them.

While implementing a Q Learning algorithm, the cab would always learn only after making mistakes and hence there is always a chance for a mistake to take place when a smartcab is learning with the Q learning algorithm. The optimal policy of this cab would be a policy where the smartcab **always** makes the right decision, wherein these decisions would help it get to the destination in the shortest, fastest path possible. This is possible when the smartcab has explored all of the 96 states, and understands which of these actions it should and should not perform. After a 100 trials, we see that the learning agent is still taking a few wrong decisions, and hence 100 trials don't seem to be enough for the learning agent to develop the optimal policy, and if we let the smartcab travel from different sources to destinations for a very long time, it would have then figured out the best possible move for every state and hence will be following the optimal policy after infinitely large number of trials.

Although the cab doesn't completely obey the optimal policy, we can clearly see from the output of the program that it starts to take the correct action more often as time passes by, and eventually it will be performing only the right actions, which is indicative of the fact that the Learning agent has found the optimal policy for the smartcab.