REPORT

TRAIN A SMARTCAB TO DRIVE

USING Q-LEARNING TO TRAIN A SMART CAB TO DRIVE AROUND IDEALISTIC CITY ROADS

ADITYA HARSH

7/20/2016

REPORT

TRAIN A SMARTCAB TO DRIVE

1) Implement a Basic Driving Agent

- **What you see in the agent's behavior. Does it eventually make it to the target location?**
  - In order to implement a basic driving agent, I utilized random.choice over a list of the possible actions. The agent does eventually make it to the target location, but it's just due to random chance.

2) Identify and Update State

- **Identify a set of states that are appropriate for modeling the driving state.**
  - To model my driving agent, I chose to use the color of the light, the presence of an oncoming car, whether or not there is a car present to the left and right of our car, and the next waypoint.

- **Why you picked this set of states and how they model the agent and its environment?**
  - My agent is concerned with making it to the goal state as quickly as possible, while avoiding collisions and other negative consequences that are built into the rewards system. Because the path planning is already implemented for us, this means that we simply want our agent to decide at each step whether it should attempt to move directly toward the next waypoint, or if there are obstacles that prevent this from being a feasible strategy. In order to do that, our state must capture both the waypoint and any potential obstacles, so those are the features that I chose to include in my state space.
  - I initially added **deadline** into the list of states, but it only lowered the performance, therefore I removed it. The lower performance was caused most likely by the fact that the Q matrix got too sparse, that the prediction step often cannot find the required value in Q matrix. In other words the Q values didn't converge (although it will, with much more trials).

3) Implement Q-Learning

- **What changes did you notice in the Agent's behavior?**
  - After implementing Q-Learning to inform the agent's policy behavior. The agent mills around for a while it is in the initial stages of the learning process, but quickly begins to direct its movement towards the goal state.

4) **Improve the Q-Learning Driving Agent**

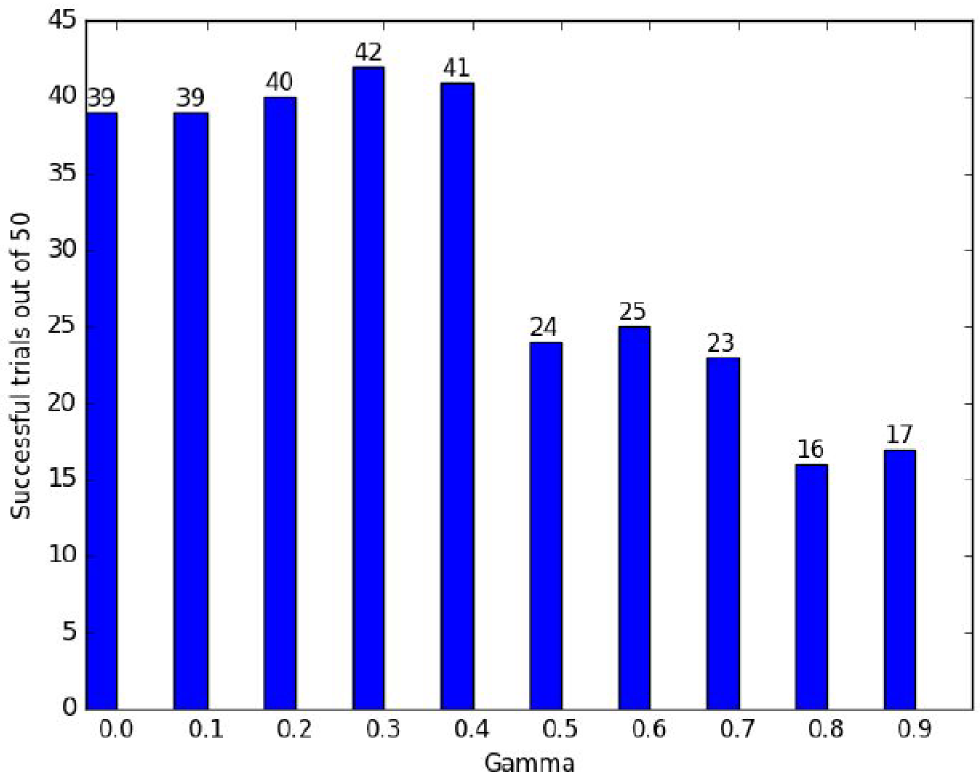
- **Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best? How well does the final driving agent perform?**
    - To model my driving agent, I chose to use the color of the light, the presence of an oncoming car, whether or not there is a car present to the left and right of our car, and the next waypoint.

- **Why you picked this set of states and how they model the agent and its environment?**
    - Tuning parameter includes:

      (1) Learning rate α: in our case, c and b are the tuning learning rate α. When α is close to 0, the agent tends to learn little from the interaction. When α is close to 1, the agent tends to have a "short memory". It reflects more on the recent interactions.

      (2) Discount factor γ: the value ranges from 0 to 1. The value more close to 0 the influence of future states are reduced and vice versa.

I first incorporated a static Gamma parameter of .5 in my model, but decided to iterate on this design and test out a range of Gamma parameters from 0 to .9 in increments of .1, and came to the conclusion that gamma values around .3 .4 performed best.



As can be seen from the figure above, I tested model performance as a function of my choice of gamma by running 50 trials (with an enforced deadline), and used the number of successfully completed trips out of 50 as my metric. As gamma is increased past .4, we start to see the model's performance degrade, but with a choice of .3 for Gamma, my agent successfully finds the destination within its deadline an average of 42 out of 50 trials, (starting the first trial with a completely untrained agent) while only encountering two penalties.

Results

The results of Q-table after 100 trials:

```
('red', None, 'left')   ['-0.89', '-0.94', '0.20', '0.03']
('red', None, 'right')   ['-0.23', '-0.22', '2.68', '0.80']
('green', None, 'forward')   ['2.30', '0.24', '0.10', '0.73']
('green', 'right', 'left') [  '1.00', '1.00', '1.00', '1.00']
('green', None, 'left')   ['-0.07', '2.40', '0.10', '0.69']
('green', 'left', 'left')   ['0.80', '1.00', '1.00', '1.00']
('red', 'left', 'left')   ['1.00', '1.00', '1.00', '1.00']
('red', 'forward', 'right')   ['0.45', '0.45', '1.00', '1.00']
('green', 'forward', 'right')   ['0.40', '1.00', '1.00', '1.00']
('red', 'right', 'left')   ['-0.55', '0.45', '1.00', '1.00']
('red', 'right', 'forward')   ['-0.55', '0.45', '1.00', '0.77']
('green', 'forward', 'left')   ['0.62', '1.00', '1.00', '1.00']
('green', None, 'right')   ['0.28', '0.10', '2.67', '0.77']
('green', 'left', 'forward')   ['2.17', '1.00', '1.00', '1.00']
('green', 'left', 'right')   ['0.26', '0.78', '1.00', '1.00']
('red', None, 'forward')   ['-1.00', '-1.00', '0.01', '0.00']
('red', 'forward', 'left')   ['-0.55', '1.00', '1.00', '1.00']
('green', 'right', 'right')   ['0.78', '0.59', '1.00', '1.00']
('red', 'right', 'right')   ['-0.55', '0.19', '1.88', '1.00']
('red', 'left', 'right')   ['-0.55', '-0.55', '1.00', '1.00']
('red', 'left', 'forward')   ['-0.55', '-0.55', '-0.11', '1.00']
('green', 'right', 'forward')   ['1.00', '1.00', '1.00', '1.00']
('green', 'forward', 'forward')   ['2.00', '1.00', '1.00', '0.81']
('red', 'forward', 'forward')   ['-0.55', '-0.55', '0.20', '0.25']
```

The great majority of the policy make sense with several expectations, it all involve an upcoming car, possible because the situation hasn't occurred too often for the agent to learn the correlate policy.

Test Results for the Q-Learning Agent

After tuning the alpha, gamma and epsilon hyperparameters, I ran the Q-Learning agent and it learnt to drive efficiently after 10-15 runs. The test results for the last 100 runs are:

```
Iteration:91
Primary agent has reached destination! : 3/20 with a cumulative reward of 28
Iteration:92
Primary agent has reached destination! : 11/35 with a cumulative reward of 40
Iteration:93
Primary agent has reached destination! : 37/45 with a cumulative reward of 12
Iteration:94
Primary agent has reached destination! : 5/30 with a cumulative reward of 32
Iteration:95
Primary agent has reached destination! : 34/45 with a cumulative reward of 17
Iteration:96
Primary agent could not reach destination within deadline!
Iteration:97
Primary agent has reached destination! : 8/20 with a cumulative reward of 21
Iteration:98
Primary agent has reached destination! : 16/25 with a cumulative reward of 13
Iteration:99
Primary agent has reached destination! : 40/45 with a cumulative reward of 9
Iteration:100
Primary agent has reached destination! : 30/35 with a cumulative reward of 9
```

Optimality and the Optimal Policy

Let us now compare the optimality of our Q-Learning agent. Our goal is to train the agent in terms of two aspects:

- Getting to the destination in minimum time steps.
- Avoiding any negative reward steps.

Therefore from the last 10 test results of our agent, we can see that the cab reaches its destination in fairly less number of steps.

Also the cab reaches its destination with a large enough cumulative reward, also the overall rewards are shorter towards the end due to the fact that the agent makes it to the destination early.

Thus, we can say that our agent has learnt the optimal policy by not only learning to take the right moves but by also reaching in the minimum span of time without performing any illegal moves.