Nathan George
Smart Cab Project Report
7-13-2016

Implement a basic driving agent

In your report, mention what you see in the agent's behavior. Does it eventually make it to the target location?
It eventually does make it to the target location, but each movement is random, of course. It doesn't take into account the reward from it's actions, so it will sometimes run red lights, violate right-of-way, and often does not follow the waypoints.

Identify and update state

Justify why you picked these set of states, and how they model the agent and its environment.
I picked these for the set of states:
next_waypoint: [None, 'left', 'right', 'forward']
light: ['red', 'green']
'oncoming': [None, 'left', 'right', 'forward']
'left': [None, 'left', 'right', 'forward']
'right': [None, 'left', 'right', 'forward']

The next_waypoint models where the car should go according to the planner, and helps the agent determine which moves to make to most efficiently reach the target. Following the waypoints gives a reward, so that helps the agent learn. The light, oncoming, left, and right traffic variables model the agent's immediate surroundings and help it make good driving choices. The light helps it learn to not run red lights (since it is penalized for running a red light), the oncoming, left and right traffic variables teach it how to obey traffic laws (since it is penalized for violating right-of-way). For example, the agent should not turn left if there is an oncoming car, and it will be penalized if it does so.

I didn't pick deadline because there would be far too many states, and the Q-learning would not be as effective. Why should a state with a green light, waypoint forward, and no cars at the intersection be any different with deadline 15 compared to 20? It shouldn't, therefore I omitted the deadline from the set of states.

Implement Q-Learning

What changes do you notice in the agent's behavior?
The agent appears to learn how to drive efficiently after the first route is completed. It follows the waypoints almost all of the time. However, it still makes some nonsense moves, such as not following the waypoints, breaking traffic rules, and sometimes not moving when it should. The penalties from running red lights and breaking right-of-way laws help it avoid those actions, while the rewards for following the waypoints help it follow the proper path.

Enhance the driving agent

Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does it perform?

I tuned the alpha and gamma parameters by running trials and varying alpha from 0.1 to 0.9. In a nested loop, I varied gamma from 0.1 to 0.9. I ran 10 trials for each alpha/gamma combination, then saved the data in pickle (.pk) files, for penalties, rewards (including rewards and penalties), total_actions (number of time steps), and non-ideal actions (meaning the action didn't match the expected action based on the waypoint and traffic laws).

The best alpha/gamma differed for each metric:

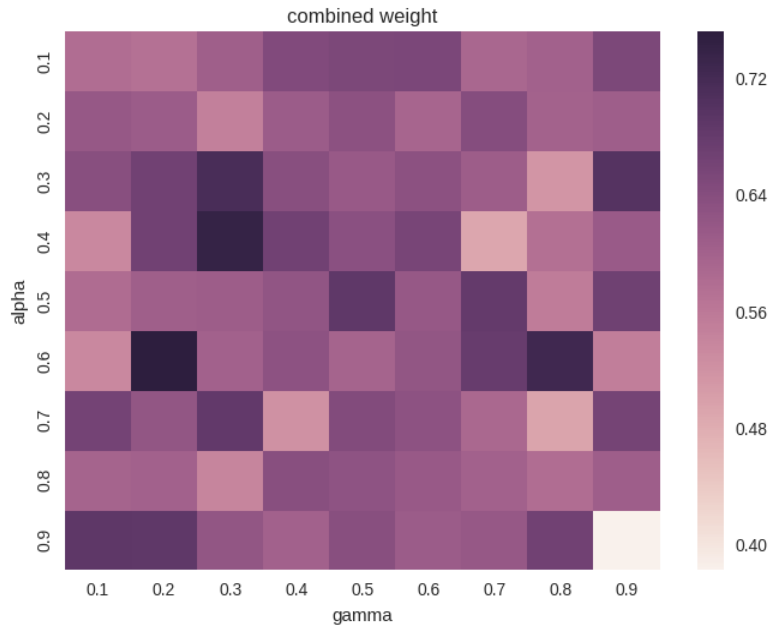
best by penalties: a=0.1, g=0.4 -74.5
best by rewards: a=0.7, g=0.3 6821.5
best by time steps: a=0.9, g=0.1 3712
best by non_ideal_actions: a=0.9, g=0.4 64

so I combined them in a weighted sum (see analyze_a-g_opt.py) and used that as my evaluation metric. I gave rewards a weight of 50%, penalties and timesteps a 20% weight, and non-ideal moves a 10% weight. The results gave:

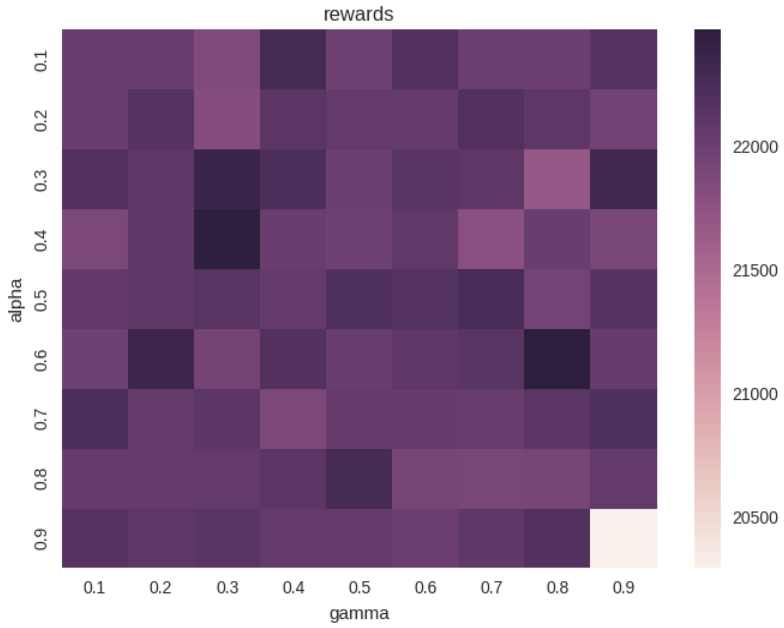
best: a=0.6, g=0.2, combined weight score of: 0.754
scaled rewards of best: 0.945
scaled penalties of best: 0.899
scaled total actions of best: 0.393
scaled non-ideal actions of best: 0.228

So I used the best alpha and gamma of 0.6 and 0.2, respectively.

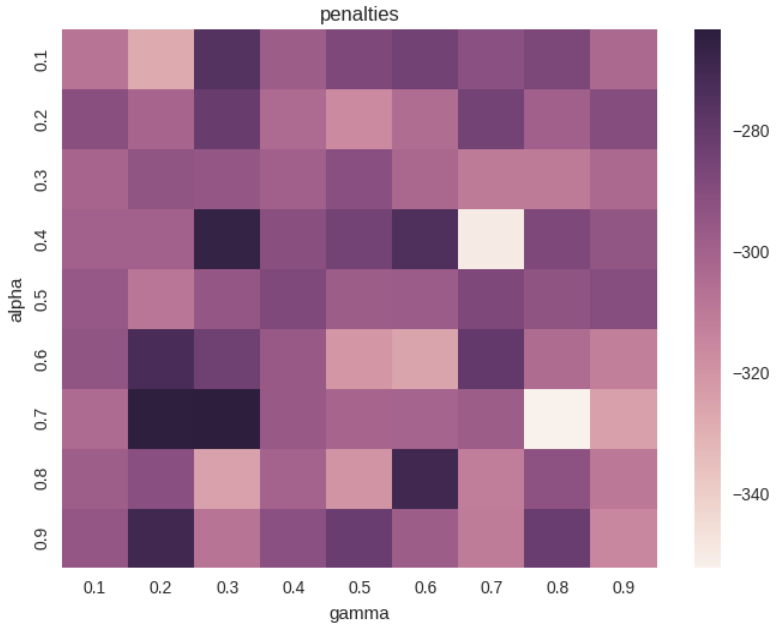
A heatmap of the combined weights show the highest values are scattered:



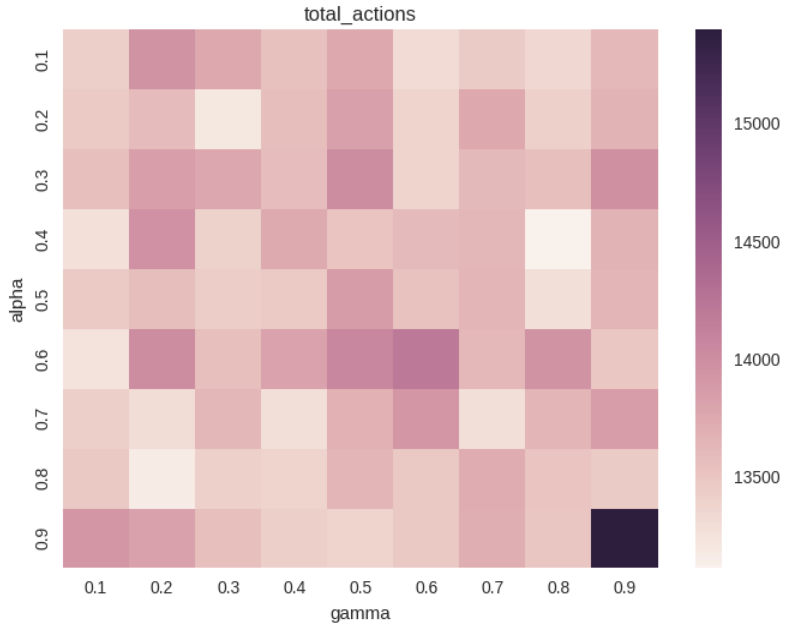
The rewards heatmap is also scattered, with 0.9, 0.9 being terrible:



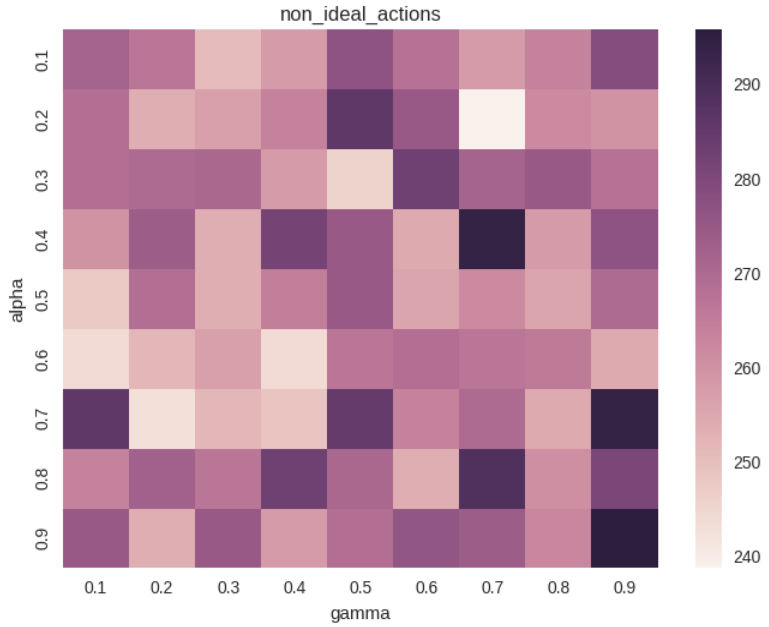
The penalties is more clustered around 0.7, 0.2



Total actions has scattered minima:



Non-ideal actions tend to be grouped around intermediate alphas and lower gammas.



Overall, the results are pretty scattered, with 0.9, 0.9 being terrible in all cases except for penalties.

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?

I would say the optimal policy is one that maximizes reward, minimizes penalties, minimizes total actions, and minimizes non-ideal actions (actions that do not match the waypoint). In that case, the agent nearly finds the optimal policy with a few exceptions. A mistake is made in about 10% of the last 10 trials. I compared the waypoint direction to the action direction, and found they are almost always identical, with a few exceptions near the end of the 100 trials. This means the agent is following the planner path, which should be the shortest distance. In about 10% of the last 10 trials, about one of the actions is non-ideal (action does not match waypoint, i.e the agent turned left when it should've gone straight). See the test1.txt, test2.txt, test3.txt files for logs.

During 3 trials, the agent broke some rules in the last 10 iterations:

non-ideal move:
action, waypoint left right
State(waypoint='right', light='green', oncoming=None, left='forward', right=None)

penalty!
action, waypoint forward forward
State(waypoint='forward', light='red', oncoming='right', left=None, right=None)

penalty!
action, waypoint left forward
State(waypoint='forward', light='red', oncoming=None, left=None, right='right')

It ran a few red lights when the waypoint was forward and there was traffic nearby, and went the completely wrong direction once when a car was going forward on its left at a green light.