Thomas Hepner
3/7/2016
P4 – Train a Smartcab to Drive

Implement a basic driving agent

1. Does it eventually make it to the target location?

The basic driving agent eventually makes it to the target location in all 10 trials of the simulation. However, it was incredibly inefficient, as it would choose an action at random, not taking any inputs into consideration; it only met the deadline in 3 of the 10 simulations. In many simulations, the result wouldn't be reached for over 200 iterations, far past the deadline.

Identify and update state

2. Justify why you picked these set of states, and how they model the agent and its environment.

I chose 5 states to model the agent. The first state is the light of the intersection that the agent is currently at, the second is the direction oncoming traffic is going, the third is the direction traffic is coming from the left, the fourth is the direction of the traffic from the right, and the fifth state is the agent's next waypoint.

I picked the first four states because they are a representation of the agent's environment and they are absolutely necessary for the agent to know in order to follow traffic rules when deciding upon an action: go forward, turn right, turn left, or take no action. I chose the fifth state, the agent's next waypoint, as this is the optimal next location the agent should choose to move to from its current waypoint; knowing this information will enable the agent to reach the destination in the shortest period of time while following the rules of the road.

Implement Q-Learning

3. What changes do you notice in the agent's behavior?

Now that the Q-learning algorithm has been implemented, I notice that the agent now tends to choose actions that yield positive rewards and follow the rules of the road. It completes the tasks much faster than it did before, but still does not usually meet the deadline. It doesn't always choose actions that make the agent closer to its destination; it sometimes gets trapped in loops of the same action actions like *right*, *right*, *right* as it favors the most recent action as long as it has a positive reward.

In my implementation, the agent goes through 100 trials in order to build Q-values for as many states as possible. In the last 10 trials, using 4 different randomly chosen seeds, the agent only reached its destination within the deadline 3.5 out of 10 times.

Enhance the driving agent

4. 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?

To achieve the final version of the agent, I made several changes to the Q-learning implementation. I improved the performance of my agent by tweaking several parameters in the Q-learning algorithm: the learning rate (α), the discount factor (ϒ), and an additional variable, epsilon (ϵ).

Q-Learning Algorithm:

$$Q(s,a) := Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \right]$$

In my initial implementation of the Q-learning algorithm, the learning rate and discount factor were both set to 0, and the variable epsilon did not exist. The learning rate was the first variable I modified. It decides the extent to which new information overrides old information; a value of 0 would mean that the agent would not learn anything, and the agent would only consider the most recent information if the value was set to 1. I attempted several different constants between 0 and 1 and found that 0.5 yielded the best results.

The discount factor was the next second variable I modified. The discount factor is a constant value between 0 and 1 that determines the worth of future rewards. I discovered that steadily decreasing the value of the discount factor in each successive trial improved performance. My implementation set the discount factor as **(100 – trial number) / (100)**. Therefore, for the first trial the discount factor was 100 / 100, for the second it was 99 / 100, the third was 98 / 100, and so on.

The last variable I modified, epsilon, doesn't factor directly into the algorithm. Instead, it is a threshold value between 0 and 1 used to determine if the agent should take a random action or not. It is used to balance the needs of the algorithm for *exploration* and *exploitation*. In my modification to the Q-learning algorithm, for each trial, a random value between 0 and 1 is chosen, and if that value is less than epsilon, the agent takes a random action. I thought that it made sense to prioritize exploration in the first trials, and exploitation in the last trials to acquire and use as much information about the agent and its states as possible. In my implementation, the function determining the value of epsilon is **1 / (trial number + 1)**. Therefore, in the first trial the value of epsilon is 1, in the second it is 1/2, and in the last it is 1/100.

5. <u>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?</u>
My agent gets very close to finding an optimal policy. I tested the performance of my agent on the last 10 trials out of 100 used build the Q table of Q values. The performance of my implementation was tested on 6 different random seeds. On average, the agent reached the destination in 8.67 out of 10 trials, never performing worse than reaching the destination in 8 out of 10 trials. In addition, the net reward of the agent was positive; in 3 out of the 6 iterations, the agent took no actions that resulted in negative rewards, and in the other 3 iterations, the agent took only 11 actions out of 537 that resulted in negative rewards.