Udacity Machine Learning Nanogree Project 4 - Train a Smartcab to Drive Report

Jason Osajima

1 Project Description

Apply reinforcement learning techniques for a self-driving agent in a simplified world to aid it in effectively reaching its destinations in the allotted time. You will first investigate the environment the agent operates in by constructing a very basic driving implementation. Once your agent is successful at operating within the environment, you will then identify each possible state the agent can be in when considering such things as traffic lights and oncoming traffic at each intersection. With states identified, you will then implement a Q-Learning algorithm for the self-driving agent to guide the agent towards its destination within the allotted time. Finally, you will improve upon the Q-Learning algorithm to find the best configuration of learning and exploration factors to ensure the self-driving agent is reaching its destinations with consistently positive results.

1.1 Background

In the not-so-distant future, taxicab companies across the United States no longer employ human drivers to operate their fleet of vehicles. Instead, the taxicabs are operated by self-driving agents — known as smartcabs — to transport people from one location to another within the cities those companies operate. In major metropolitan areas, such as Chicago, New York City, and San Francisco, an increasing number of people have come to rely on smartcabs to get to where they need to go as safely and efficiently as possible. Although smartcabs have become the transport of choice, concerns have arose that a self-driving agent might not be as safe or efficient as human drivers, particularly when considering city traffic lights and other vehicles. To alleviate these concerns, your task as an employee for a national taxicab company is to use reinforcement learning techniques to construct a demonstration of a smartcab operating in real-time to prove that both safety and efficiency can be achieved.

2 Definitions

2.1 Environment

The smartcab operates in an ideal, grid-like city (similar to New York City), with roads going in the North-South and East-West directions. Other vehicles will certainly be present on the road, but there will be no pedestrians to be concerned with. At each intersection is a traffic light that either allows traffic in the North-South direction or the East-West direction. U.S. Right-of-Way rules apply. On a green light, a left turn is permitted if there is no oncoming traffic making a right turn or coming straight through the intersection. On a red light, a right turn is permitted if no oncoming traffic is approaching from your left through the intersection.

2.2 Inputs and Outputs

Assume that the smartcab is assigned a route plan based on the passengers' starting location and destination. The route is split at each intersection into waypoints, and you may assume that the smartcab, at any instant, is at some intersection in the world. Therefore, the next waypoint to the destination, assuming the destination has not already been reached, is one intersection away in one direction (North, South, East, or West). The smartcab has only an egocentric view of the intersection it is at: It can determine the state of the traffic light for its direction of movement, and whether there is a vehicle at the intersection for each of the oncoming directions. For each action, the smartcab may either idle at the intersection, or drive to the next intersection to the left, right, or ahead of it. Finally, each trip has a time to reach the destination which decreases for each action taken (the passengers want to get there quickly). If the allotted time becomes zero before reaching the destination, the trip has failed.

2.3 Rewards and Goal

The smartcab receives a reward for each successfully completed trip, and also receives a smaller reward for each action it executes successfully that obeys traffic rules. The smartcab receives a small penalty for any incorrect action, and a larger penalty for any action that violates traffic rules or causes an accident with another vehicle. Based on the rewards and penalties the smartcab receives, the self-driving agent implementation should learn an optimal policy for driving on the city roads while obeying traffic rules, avoiding accidents, and reaching passengers' destinations in the allotted time.

3 Implement a Basic Driving Agent

To begin, your only task is to get the smartcab to move around in the environment. At this point, you will not be concerned with any sort of optimal driving policy. Note that the driving agent is given the following information at each intersection: the next waypoint location relative to its current location and heading, the state of the traffic light at the intersection and the presence of oncoming vehicles from other directions, and the current time left from the allotted deadline.

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?

I started by implementing a basic driving agent that selected a random action (either None, 'forward', 'left', 'right') at each intersection. The agent disregards its next waypoint location relative to its current location and heading, traffic lights, the presence of oncoming vehicles, and the current time left before the deadline.

I set the simulation deadline enforcement to 'False' in order to observe how the agent behaved. As expected, the agent made random as opposed to optimal decisions. It disregarded the 'rewards' associated with actions at each intersection. Some examples of this behavior included taking no action (action = None) when there was no oncoming traffic or a red light, running a red light, and driving in the opposite direction of the destination.

Next, I set the simulation deadline enforcement to 'True' and ran the simulation for 100 trials. The basic driving agent was able to reach its destination 26 times in 100 trials, or 26% of the time. I will use this as my benchmark to compare with the agent using q-learning.

4 Inform the Driving Agent

Now that your driving agent is capable of moving around in the environment, your next task is to identify a set of states that are appropriate for modeling the smartcab and environment. The main source of state variables are the current inputs at the intersection, but not all may require representation. You may choose to explicitly define states, or use some combination

of inputs as an implicit state. At each time step, process the inputs and update the agent's current state using the self.state variable. Continue with the simulation deadline enforcement enforce _ deadline being set to False, and observe how your driving agent now reports the change in state as the simulation progresses.
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?

I identified four relevant state variables:

- Direction of next waypoint: [None, 'forward', 'left', 'right']
- Light status ['red', 'green']
- Oncoming traffic: [None, 'forward', 'left', 'right']
- Traffic to the left: [None, 'forward', 'left', 'right']

The actions are: [None, 'forward', 'left', 'right']

I selected direction of next waypoint in order for the agent to prioritize reaching the destination instead of randomly selecting actions. I decided to use light status so the agent would learn to stop when the light is red. This will prevent the agent from going forward and breaking traffic rules and potentially crashing into another car. I determined that the state variables oncoming traffic and traffic to the left are also relevant to prevent the agent from running into another car and receiving a penalty.
The other state variables are traffic to the right and deadline. I chose to ignore traffic to the right because it won't affect the agent's actions. Traffic to the right has four states: [None, 'forward', 'left', 'right']. If traffic to the right is None, it has no effect on the agent. If traffic to the right is 'forward' or 'left', the light will be red for the agent and the agent won't be affected. If traffic to the right is 'right', the agent has the right of way and can go 'forward', 'right', or 'left.'
I also decided to omit deadline. While the deadline state variable would be useful for the smartcab to use when making decisions, I felt that using deadline would increase the set of possible states exponentially and decrease the effectiveness of Q-Learning for a limited amount of trials.

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?*

In total, I chose 4 state variables, which means there are 4*2*4*4 = 128 different states. This number seems reasonable, since Q-Learning can iterate through each of the 128 states many times in 100 trials to determine which ones are optimal to getting to the destination.

5 Implement a Q-Learning Driving Agent

With your driving agent being capable of interpreting the input information and having a mapping of environmental states, your next task is to implement the Q-Learning algorithm for your driving agent to choose the best action at each time step, based on the Q-values for the current state and action. Each action taken by the smartcab will produce a reward which depends on the state of the environment. The Q-Learning driving agent will need to consider these rewards when updating the Q-values. Once implemented, set the simulation deadline enforcement enforce_ deadline to True. Run the simulation and observe how the smartcab moves about the environment in each trial.

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?

After implementing q-learning and setting initial parameters arbitrarily (copied below) the agent was able to reach its destination 97 out of the 100 trials. This is a 273% improvement when compared with the basic agent's performance (the basic agent reached its destination 26 out of 100 times).

Table 1 - Parameters and Results for Q-Learning Smartcab Agent (Run 1)

	alpha	gamma	epsilon	default Q	trials completed successfully
Run 1	0.9	0.2	0.1	0	97

With each subsequent trial, the agent becomes more informed through the q-learning algorithm and makes decisions that minimize negative rewards and maximize positive ones. It is clear that in the beginning the agent is still learning, because it still executes actions that appear to be random, such as waiting when there is a green light with no oncoming traffic.

6 Improve the Q-Learning Driving Agent

Your final task for this project is to enhance your driving agent so that, after sufficient training, the smartcab is able to reach the destination within the allotted time safely and efficiently. Parameters in the Q-Learning algorithm, such as the learning rate (alpha), the discount factor (gamma) and the exploration rate (epsilon) all contribute to the driving agent's ability to learn the best action for each state. To improve on the success of your smartcab:

- Set the number of trials, n_ trials, in the simulation to 100.
- Run the simulation with the deadline enforcement enforce_ deadline set to True (you will need to reduce the update delay update_ delay and set the display to False).
- Adjust one or several of the above parameters and iterate this process.

This task is complete once you have arrived at what you determine is the best combination of parameters required for your driving agent to learn successfully.

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?

I decided to test the following combinations of parameters, with the following results:

Table 2 - Parameters and Results for Q-Learning Smartcab Agent (Runs 1-4)

	alpha	gamma	epsilon	default Q	trials completed successfully
Run 1	0.9	0.2	0.1	0	97
Run 2	0.8	0.35	0.01	5	96
Run 3	0.85	0.45	0.001	10	98
Run 4	0.95	0.55	0.001	20	95

After four runs with different parameters, the agent performs in Run 3, with an alpha of 0.85, gamma of 0.45, epsilon of 0.001, and default Q of 10. The agent in Run 3 performs 98 out of 100 trials successfully.

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?

I would say yes, my agent is close to finding an optimal policy. Looking at the last ten trials of Run 3, the agent reaches its destination successfully with the following deadlines:

Table 3 - Summary of Efficiency of Smartcab Run 3 Trials 90-99

trial number	steps before deadline	steps taken	ratio
90	20	12	0.60
91	20	15	0.75
92	30	11	0.37
93	20	11	0.55
94	20	13	0.65
95	25	17	0.68
96	20	16	0.80
97	30	13	0.43
98	30	16	0.53
99	25	14	0.56

While there is not a clear increase in efficiency (since the ratio vacillates), the agent is able to reach its destination in a little more than half of its allotted steps before the deadline.

Through q-learning, the agent is able to find an optimal road policy while still being able to experiment with alternative policies through exploration.

The agent will typically stop at red lights and when there is oncoming traffic or traffic coming from the left. The agent will for the most part follow the route to its destination.

I would describe an optimal policy in a similar vein to what was described earlier. An agent will stop at red lights and stop when there is oncoming traffic or traffic coming from the left. An agent will follow next_ waypoints which will lead it to its destination.