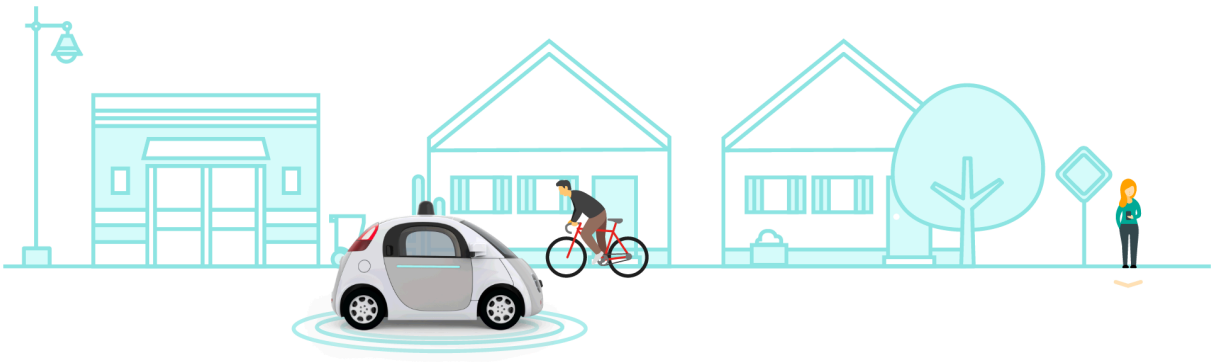
Train a Smartcab to Drive

Revision 1



Ramil Sharifsoy
August 6, 2016
Udacity
Machine Learning Engineer Nanodegree

Questions and Answers

QUESTION: 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?

Change and add code as follows:
action = random.choice([None, 'forward', 'left', 'right']) and
e.set_primary_agent(a, enforce_deadline=False)

Initialize additional variables
self.reward = None, at the starting point agent has 0 rewards
self.next_waypoint = None, next intersection initially unknown
self.state = None, initial state is unknown

Before every new trip reset following variables:
self.state = None, clear state and set to unknown
self.next_waypoint = None, next intersection is unknown before new trip
self.reward = 0, reset reward to 0 after reaching the destination

Interesting part is allocation of rewards. For correct action reward is 2, for passing on red light penalty is -1, for acceptable but wrong action penalty is -0.5, and for non-action on green reward is 0. It takes a lot of penalties and steps to reach destination, and in few cases agent doesn't reach destination at all. Violation of traffic rules, like passing on red light is not safe and is not acceptable, final algorithm must eliminate penalty -1.0 completely and minimize -0.5s to negligible.

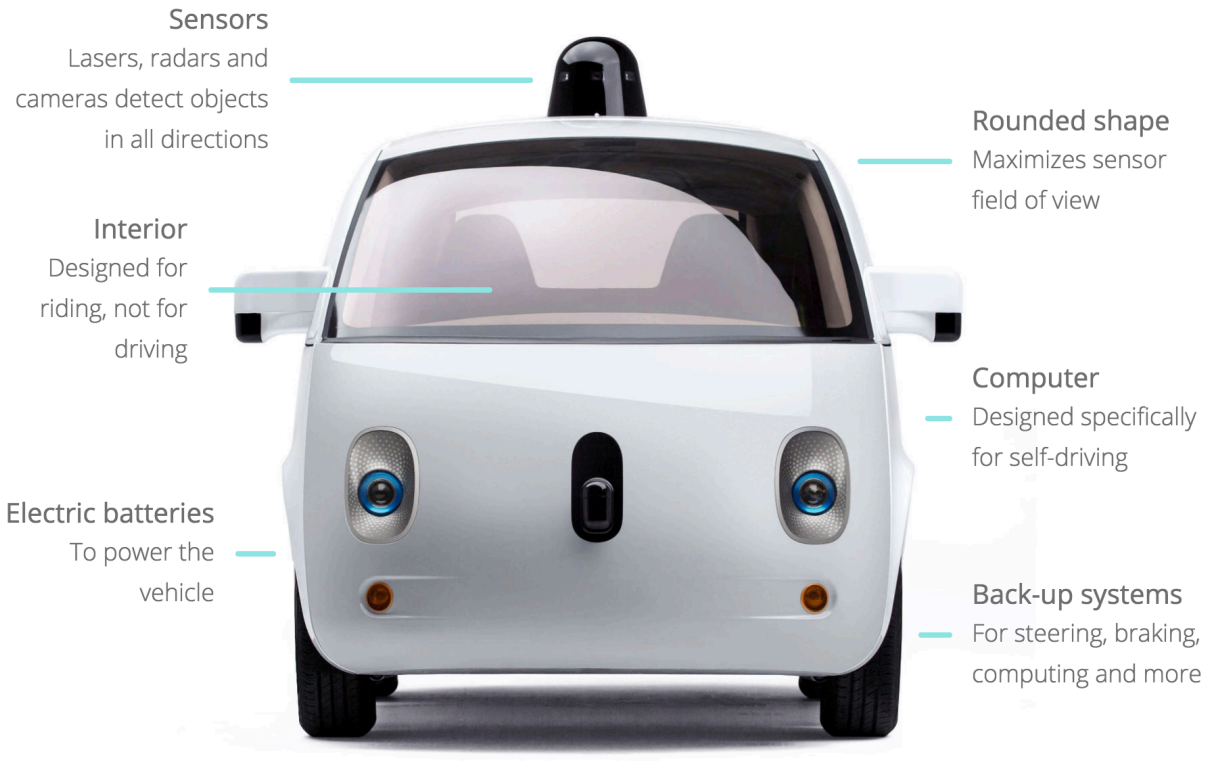
Examples:
inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = right, reward = 2.0
inputs = {'light': 'green', 'oncoming': None, 'right': None, 'left': None}, action = None, reward = 0.0
inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = forward, reward = -1.0
inputs = {'light': 'red', 'oncoming': None, 'right': None, 'left': None}, action = right, reward = -0.5

QUESTION: 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?

In this section we'll explicitly state which "sensory inputs" (self.env.sense(self)) were considered in the state space in addition to next_waypoint and justify why these inputs would be necessary so that the agent has enough critical information about the world in order to make decisions and eventually learn the optimal rules of the road.

First let's explore complex environment in real world, and how smartcab will operate in that environment. What are the sensors, and what are these sensors are designed to sense?

Autonomous cars can detect surroundings using a variety of techniques such as radar, lidar, GPS, odometry, and computer vision. Advanced control systems interpret sensory information to identify appropriate navigation paths, as well as obstacles and relevant signage. Autonomous cars have control systems that are capable of analyzing sensory data to distinguish between different cars on the road.
The smartcabs use their sensors and software to sense objects like pedestrians, cyclists, vehicles and more, and are designed to safely drive around them. See picture one for details.



Picture1: Google self-driving car prototype

Self-driving cars process both map and sensor information to determine where they are in the world. It knows what street it's on and which lane it's in. Sensors help detect objects all around it, see picture 2 for screenshot from self-driving car navigating through construction zone. The software classifies objects based on their size, shape and movement pattern. The software predicts what all the objects around us might do next. It predicts that the cyclist will ride by and the pedestrian will cross the street. The software then chooses a safe speed and trajectory for the car. This car nudges away from the cyclist, then slows down to yield to the pedestrian. As you can see, idea of decision making system with acting system is very similar to human decision and action system.

Picture 2: Navigation screen of Google Self Driving Car, Lexus RX, detecting construction zone.

Last, let's elaborate on our simplified case. In this case, sense function senses location, heading, traffic, and traffic lights. These are the parameters necessary for smart cab to operate, we cannot eliminate any of these factors. Its acting based on whether light is green or red, and whether there is an oncoming, left-turning, or right-turning traffic, and combination of these observations.

QUESTION: 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?

In random behavior, agent reaches the destination randomly without following certain policy. Agent was reaching the destination slowly by making a lot of violations, most of the time did not reach the destination with in given deadline.

QUESTION: 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?

Learning rate (alpha)

The learning rate determines to what extent the newly acquired information will override the old information. A factor of 0 will make the agent not learn anything, while a factor of 1 would make the agent consider only the most recent information. In practice, often a constant learning rate is used, such as 0.1 for all. I will start iteration with this value.

Discount factor(gamma)

The discount factor determines the importance of future rewards. A factor of 0 will make the agent short-sighted by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward. It is known that starting with a lower discount factor and increasing it towards its final value yields accelerated learning. Starting value for gamma will be 0.02.

Epsilon (Q_0)

Epsilon – greedy method is when the agent chooses the action that it believes has the best long-term effect with probability 1- epsilon, and it chooses an action uniformly at random. Using this policy either we can select random action with epsilon probability and we can select an action with 1-epsilon probability that gives maximum reward in given state. We will start selection with epsilon = 1, which comes with the minimum reward.

Agent performance rates based on different combinations of factors:

Learning rate (alpha)	**Discount factor (gamma)**	**Epsilon value**	Rate of reaching the destination.	Any Traffic violation (-1.0)
0.1	0.02	1	26	Yes, Significant
0.2	0.04	0.95	23	Yes, Significant
0.4	0.1	0.6	57	Yes, Significant
0.6	0.3	0.2	84	Yes, Few
0.8	0.4	0.1	95	Yes, Very Few
0.9	0.5	0.1	89	Yes, Few
0.95	0.4	0.01	96	Yes, Very Few
0.99	**0.3**	**0.01**	**97**	**Yes, Very Few**
0.99	0.2	0.01	95	Yes, Very Few
0.99	0.25	0.01	95	Yes, Very Few
1	0	0	98	None

Nearest ideal condition parameters are highlighted in the table above with success rate of 97%.

QUESTION: 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?

Ideal condition is when agent focused on current state. Very similar to human success factor, focused on now only, without being worrying about past or future. Where alpha is 1, considers only most recent information; short-sighted with gamma = 0; and for epsilon we don't want exploration with random actions, we need to follow policy, therefore epsilon for idea condition should be 0.

Smart cab reached optimal policy, it reaches destination in the minimum possible time, and not incur any penalties almost every time. One example shown below:

Simulator.run(): Trial 96
{'light': 'red', 'oncoming': None, 'right': None, 'next_waypoint': 'right', 'left': None}, action = right, reward = 2.0

{'light': 'green', 'oncoming': None, 'right': None, 'next_waypoint': 'right', 'left': None}, action = right, reward = 2.0
{'light': 'green', 'oncoming': None, 'right': None, 'next_waypoint': 'forward', 'left': None}, action = forward, reward = 2.0
{'light': 'green', 'oncoming': None, 'right': None, 'next_waypoint': 'forward', 'left': None}, action = forward, reward = 2.0
{'light': 'red', 'oncoming': None, 'right': None, 'next_waypoint': 'right', 'left': None}, action = right, reward = 2.0
Environment.act(): Primary agent has reached destination!
{'light': 'green', 'oncoming': None, 'right': None, 'next_waypoint': 'forward', 'left': None}, action = forward, reward = 12.0

References:

- Q-learning, https://en.wikipedia.org/wiki/Q-learning
- Autonomous car, https://en.wikipedia.org/wiki/Autonomous_car
- Google Self-driving Car Project: https://www.google.com/selfdrivingcar/how/
- A Bible On Self Driving Cars: A New Revolution http://techstory.in/self-driving-cars/
- How to implement epsilon greedy strategy / policy, https://junedmunshi.wordpress.com/2012/03/30/how-to-implement-epsilon-greedy-strategy-policy/
- Reinforcement learning, https://en.wikipedia.org/wiki/Reinforcement_learning