smartcab

October 19, 2016

1 Machine Learning Engineer Nanodegree

1.1 Reinforcement Learning

1.2 Project 4: Train a Smartcab How to Drive

1.3 Project Overview

In this project you will 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.

1.4 Description

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.

1.5 Definitions

1.5.1 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 there 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. To understand how to correctly yield to oncoming traffic when turning left, you may refer to this official drivers' education video, or this passionate exposition.

1.5.2 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.

1.5.3 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.

1.6 Tasks

1.6.1 Project Report

You will be required to submit a project report along with your modified agent code as part of your submission. As you complete the tasks below, include thorough, detailed answers to each question *provided in italics*.

1.6.2 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.
- The current time left from the allotted deadline.

To complete this task, simply have your driving agent choose a random action from the set of possible actions `(None, 'forward', 'left', 'right')` at each intersection, disregarding the input information above. Set the simulation deadline enforcement, `enforce_deadline` to `False` and observe how it performs.

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?*

Answer: When the agent takes random actions, it eventually makes it to the destination. It is possible to see that sometimes the **smartcab** doesn't comply with the *U.S. Right-of-Way rules*, either by running a red light or by not noticing the presence of other **smartcabs** at the intersections. It is interesting to notice that by taking random actions, the **smartcab** doesn't learn anything about the environment, wasting the feedback (*reward* and the *next state*) returned by the interaction with the environment.

1.6.3 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.

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?*

Answer: We could map each state as being the following tuple: `(left, oncoming, right, next_waypoint)` where `left`, `oncoming` or `right` are `True` if the **smartcab** can go into the underlying direction, and `False` otherwise; and `next_waypoint` can be `None`, `left`, `oncoming` or `right` depending on the suggested action by the `planner`.

Following this definition, each state can be divided and summarized as the following:

1. `(left, oncoming, right)` represents the directions that the **smartcab** can go/turn respecting the *U.S. Right-of-Way rules*. This information takes into account the traffic lights and the other **smartcabs** at the intersection. Therefore, it describes the agent *awareness* of the surroundings and the allowed directions to go according to the *U.S. Right-of-Way rules*.
2. `next_waypoint` represents the action suggested by the `planner`. This information represents the *best action* to take not considering the agent's surroundings (traffic lights and other agents).

OPTIONAL: *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?*

Answer: There are 32 possible states in total regarding the above state definition. This number seems reasonable given that the size of the environment (6x8 grid) and this quantity have the

same order of magnitude. Moreover, as the model runs multiple times (100 times by default), the **smartcab** makes hundreds of moves, and as a result, it visits each state multiple times.

1.6.4 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.

The formulas for updating Q-values can be found in this video.

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?*

Answer: When the actions are taken using a policy instead of randomness, it is possible to see that the agent learns over time. In the beginning, the **smartcab** takes random actions as if it didn't know yet the relationships between *states, actions, and rewards*. However, once it has visited the same state twice or more times, the agent can better assess which direction to take.

This behavior happens because the **smartcab** learns the best action to take given its current state (function known as *policy*). Specifically, the agent learns the best action in that state by choosing the action that maximizes the long-term reward achieved by being in that state and taking one of available actions (function known as *quality function*).

1.6.5 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`).
- Observe the driving agent's learning and smartcab's success rate, particularly during the later trials.
- 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.

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?*

Answer: We can tune the learning rate (`alpha`), exploration rate (`epsilon`), and the discount factor (`gamma`) parameters using the following functions:

1. The learning rate (`alpha`) is always the rational function *1/x*.

2. The exploration rate (`epsilon`) is one of the constant functions *(0.1, 0.2, or 0.3)* or the rational function *1/x*.
3. The discount factor (`gamma`) is one of the exponential functions *pow(0.8, t), pow(0.5, t), or pow(0.8, t)*.

In total, we have 12 combinations of parameters, as we can see in the table below:
Low exploration rate : `epsilon=0.1`

Learning rate (`gamma`)	Aborted Trials	Successful Trials	Total Trials	Percentage Aborted
pow(0.25, t) (low)	1	99	100	1%
pow(0.5, t) (median)	4	96	100	4%
pow(0.8, t) (high)	7	93	100	7%

Median exploration rate : `epsilon=0.2`

Learning rate (`gamma`)	Aborted Trials	Successful Trials	Total Trials	Percentage Aborted
pow(0.25, t) (low)	9	91	100	9%
pow(0.5, t) (median)	6	94	100	6%
pow(0.8, t) (high)	15	85	100	15%

High exploration rate : `epsilon=0.3`

Learning rate (`gamma`)	Aborted Trials	Successful Trials	Total Trials	Percentage Aborted
pow(0.25, t) (low)	24	76	100	24%
pow(0.5, t) (median)	19	81	100	19%
pow(0.8, t) (high)	25	75	100	25%

"Rational" exploration rate : `epsilon=1/x`

Learning rate (`gamma`)	Aborted Trials	Successful Trials	Total Trials	Percentage Aborted
pow(0.25, t) (low)	5	95	100	5%
pow(0.5, t) (median)	5	95	100	5%
pow(0.8, t) (high)	6	94	100	6%

The best results for the exploration rate are achieved either by this function being a low constant value `epsilon=0.1` (first table) or by being the rational function *1/x* (last table). Considering the combination of the three parameters, the best result is achieved by the combination `(alpha, epsilon, gamma) = (1/x, 0.1, pow(0.25, t))` which returns 99% of success

rate in this test.

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?*

Answer: Yes, the agent gets close to find an optimal policy, reaching the destination in the minimum possible time. However, in this scenario the optimal policy is just theoretical. If neither the traffic lights nor the other **smartcabs** were present in the environment, the optimal policy could be the function `self.planner.next_waypoint()` returned by the `planner`. However, with the traffic lights and other agents interacting in the environment, the optimal policy doesn't agree with the above many times. In this case, the optimal policy would sometimes be the actions suggested by the `planner`, but other times, it would take different actions along the way (that ultimately maximize the long-term reward of the agent).