Train a Smartcab How to Drive

A smartcab is a self-driving car from the not-so-distant future that ferries people from one arbitrary location to another. In this project, you will use reinforcement learning to train a smartcab how to drive.

Environment

Your smartcab operates in an idealized grid-like city, with roads going North-South and East-West. Other vehicles may be present on the roads, but no pedestrians. There is a traffic light at each intersection that can be in one of two states: North-South open or East-West open.

US right-of-way rules apply: On a green light, you can turn left only if there is no oncoming traffic at the intersection coming straight. On a red light, you can turn right if there is no oncoming traffic turning left or traffic from the left going straight.

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.

Inputs

Assume that a higher-level planner assigns a route to the smartcab, splitting it into waypoints at each intersection. And time in this world is quantized. At any instant, the smartcab is at some intersection. Therefore, the next waypoint is always either one block straight ahead, one block left, one block right, one block back or exactly there (reached the destination).

The smartcab only has an egocentric view of the intersection it is currently at (sorry, no accurate GPS, no global location). It is able to sense whether the traffic light is green for its direction of movement (heading), and whether there is a car at the intersection on each of the incoming roadways (and which direction they are trying to go).

In addition to this, each trip has an associated timer that counts down every time step. If the timer is at 0 and the destination has not been reached, the trip is over, and a new one may start.

Outputs

At any instant, the smartcab can either stay put at the current intersection, move one block forward, one block left, or one block right (no backward movement).

Rewards

The smartcab gets a reward for each successfully completed trip. A trip is considered "successfully completed" if the passenger is dropped off at the desired destination (some intersection) within a pre-specified time bound (computed with a route plan).

It also gets a smaller reward for each correct move executed at an intersection. It gets a small penalty for an incorrect move, and a larger penalty for violating traffic rules and/or causing an accident.

Goal

Design the AI driving agent for the smartcab. It should receive the above-mentioned inputs at each time step t, and generate an output move. Based on the rewards and penalties it gets, the agent should learn an optimal policy for driving on city roads, obeying traffic rules correctly, and trying to reach the destination within a goal time.

Tasks

Download smartcab.zip, unzip and open the template Python file `agent.py` (do not modify any other file). Perform the following tasks to build your agent, referring to instructions mentioned in `README.md` as well as inline comments in `agent.py`.

Also create a project report (e.g. Word or Google doc), and start addressing the questions indicated in *italics* below. When you have finished the project, save/download the report as a PDF and turn it in with your code.

Task 1: Implement a Basic Driving Agent

Implement the basic driving agent, which processes the following inputs at each time step:

- Next waypoint location, relative to its current location and heading,
- Intersection state (traffic light and presence of cars), and,
- Current deadline value (time steps remaining),

And produces some random move/action `(None, 'forward', 'left', 'right')`. Don't try to implement the correct strategy! That's exactly what your agent is supposed to learn.

Run this agent within the simulation environment with `enforce_deadline` set to `False` (see `run` function in `agent.py`), and observe how it performs. In this mode, the agent is given unlimited time to reach the destination. The current

state, action taken by your agent and reward/penalty earned are shown in the simulator.

> The agent acts in a random manner, although it does make it to its target location, it does not do so in an optimal manner. The agent does not always use actions that it has found to produce rewards from past actions that it has taken. `agent.py` was ran three times with 100 trials each and the success rate was 22%, 23%, and 31% respectively.

Identify and Update State

Identify a set of states that you think are appropriate for modeling the driving agent. The main source of state variables are current inputs, but not all of them may be worth representing. Also, you can choose to explicitly define states, or use some combination (vector) of inputs as an implicit state.

At each time step, process the inputs and update the current state. Run it again (and as often as you need) to observe how the reported state changes through the run.

> For this model I chose to use the states traffic lights and waypoint, because they are the only factors that effect the agent on a computational basis. The agent would get a negative reward if it were to run red traffic lights and because the most direct path matters the most is why I chose waypoint. The other states such as oncoming has no effect on the performance of the agent.

Task 2: Implement Q-Learning

Implement the Q-Learning algorithm by initializing and updating a table/mapping of Q-values at each time step. Now, instead of randomly selecting an action, pick the best action available from the current state based on Q-values, and return that.

Each action generates a corresponding numeric reward or penalty (which may be zero). Your agent should take this into account when updating Q-values. Run it again, and observe the behavior.

> I implemented the Q-Learning agent in its own script `qlearn.py`. I notice that the agent is learning the rules of the environment and chooses not to make decisions that impact its score negatively. The agent is obeying the traffic rules that and performs legal moves.

Enhance the driving agent

Apply the reinforcement learning techniques you have learnt, and tweak the parameters (e.g. learning rate, discount factor, action selection method, etc.), to improve the performance of your agent. Your goal is to get it to a point so that within 100 trials, the agent is able to learn a feasible policy - i.e. reach the destination within the allotted time, with net reward remaining positive.

> To try and get the best learning rate for the agent, I simply played around with the values of the learning rate (alpha), exploration rate (epsilon), and the discount rate (gamma). This is a trial and error approach that I took. The below chart shows the success rate of each set of values that I used. The success rate is determined on how often the agent reaches its destination within the deadline. I found that the agent started to learn at a faster pace when I set alpha to 0.6 and gamma to 0.35. Because alpha likes long term rewards and gamma likes immediate rewards, I felt that this was a good trade off.

Alpha	Epsilon	Gamma	Success Rate	Negative Rewards	Total Rewards
1.0	1.0	1.0	26%	719	1484
0.9	0.7	0.75	42%	513	2512
0.7	0.1	0.65	78%	57	3158
0.9	0.0	0.50	99%	0	3434
0.6	0.1	0.35	99%	10	2951
0.7	0.0	0.25	99%	14	2914

> I also ran a 500 trial test and the agent reached its target 100% of the time and only having a total of 18 negative rewards.

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 think the agent is close to finding the optimal policy. It consistently reaches its destination while keeping penalties to a minimum. During red lights, the agent typically does nothing or goes the direction of the route planner. Although this isn't all bad, the agent may want to take a diffrent approach if the deadline is close.

reference

https://github.com/monchote/pacman-projects/blob/master/p3_reinforcement_learning/qlearningAgents.py
https://studywolf.wordpress.com/2012/11/25/reinforcement-learning-q-

learning-and-exploration/ http://www.cs.rutgers.edu/~mlittman/papers/thesis96.pdf
https://www.cs.rhul.ac.uk/home/chrisw/new_thesis.pdf