

Train a Smartcab 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

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.

In your report, mention what you see in the agent's behavior. Does it eventually make it to the target location?

The random agent is eventually reaching the destination with variables at play (other vehicles, oncoming traffic, traffic lights, etc) but the approach to reach the goal is clearly random and not optimal. In fact the agent is not optimizing the rewards gain. The agent does not prefer actions that it has tried in the past and found to be effective in producing reward. In fact the agent is constantly in "discovery" mode where he try new action without any knowledge of the past rewarding (or not) actions.

The agent does not exploit what it has already discover in order to obtain reward. For example the agent does not understand, run after run, that running a red-light produce bad reward.

We noticed that several times, the agent, being close to the goal choose a different direction than the preferred to reach the destination.

We can also remark the fact that the agent sometimes remain in position (action = `'None'`) even when there is no oncoming traffic or red light.

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.

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