Project Description - 1 - Udacity

P4: Train a Smartcab to Drive

Submit your project

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.

1/5

Next

Project Description - 2 - Udacity

P4: Train a Smartcab to Drive
Submit your project

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.

2/5

Next   Previous

Tasks - 1 - Udacity

P4: Train a Smartcab to Drive
Submit your project

Tasks

Setup

You need Python 2.7 and **pygame** for this project: https://www.pygame.org/wiki/GettingStarted
For help with installation, it is best to reach out to the pygame community [help page, Google group, reddit].

Download

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.

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?

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.

3/5

Next | Previous

Tasks - 2 - Udacity

P4: Train a Smartcab to Drive
Submit your project

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.

What changes do you notice in the agent's behavior?

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.

Report what changes you made to your basic implementation of Q-Learning to achieve the final version of the agent. How well does it perform?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?
The formulas for updating Q-values can be found in [this](#) video.

4/5
[Next] [Previous]

Udacity Reviews

Project Specification

Train a Smartcab to Drive

Implement a basic driving agent

Criteria	Meets Specifications
Agent accepts inputs	Student is able to implement the desired interface to the agent that accepts specified inputs.
Produces a valid output	The driving agent produces a valid output (one of None, 'forward', 'left', 'right') in response to the inputs.
Runs in simulator	The driving agent runs in the simulator without errors. Rewards and penalties do not matter - it's okay for the agent to make mistakes.

Identify and update state

Criteria	Meets Specifications
Reasonable states identified	Student has identified states that model the driving agent and environment, along with a sound justification.
Agent updates state	The driving agent updates its state when running, based on current input. The exact state does not matter, and need not be correlated with inputs, but it should change during a run.

Implement Q-Learning

Criteria	Meets Specifications
Agent updates Q-values	The driving agent updates a table/mapping of Q-values correctly, implementing the Q-Learning algorithm.

| Picks the best action | Given the current set of Q-values for a state, it picks the best available action. |

| Changes in behavior explained | Student has reported the changes in behavior observed, and provided a reasonable explanation for them. |

Enhance the driving agent

Criteria	Meets Specifications
Agent learns a feasible policy within 100 trials	The driving agent is able to consistently reach the destination within allotted time, with net reward remaining positive.
Improvements reported	Specific improvements made by the student beyond the basic Q-Learning implementation have been reported, including at least one parameter that was tuned along with the values tested. The corresponding results for each value are also reported.
Final agent performance discussed	A description is provided of what an ideal or optimal policy would be. The performance of the final driving agent is discussed and compared to how close it is to learning the stated optimal policy.

English