Train a Smartcab to Drive: Project Report

Shilpak Banerjee

July 3, 2016

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'). Dont try to implement the correct strategy! Thats 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.

TO DO: In your report, mention what you see in the agents behavior. Does it eventually make it to the target location?

Answer: In the existing code this situation is achieved by setting alpha = anything, gamma = anything, epsilon = 1, enforce_deadline = False. Yes the agent does find its way to the destination most of the times. Only on few occassions trial is aborted because of reaching the hard time limit.

2 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.

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

Answer: I choose next waypoint, traffic light atatus, heading/presence of oncoming traffic and heading/presence of traffic to the left of the smartcab as my state. I skipped right traffic and also the deadline since I do not want my smartcab to crash or diobey traffic rules to reach destination on time. Also traffic to the right do no affect the anything, so I skipped it. (Also after completing the project, it seems to me that there is no severe penalty for crashing or it des not happen very often because of low traffic/low number of trails. So next waypint and traffic light will probably make a better choice of states since this lower number of states will also increase the number of times each state is visited during the trails).

3 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.

TO DO: What changes do you notice in the agents behavior?

Answer: With $\alpha = 0.05$, $\gamma = 0.95$ and $\epsilon = 0.2$, the agent reaches destination most of the time. If we set enforse_deadline = True, then the agen reaches deadline with increasing frequency with time.

4 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 DO: 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?

Answer: I implemented Q-Learning using the following update formula:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\big(R + \max_{a' \in A(s')} Q(s',a')\big)$$

Here s is the present state, a is the action taken and s' is the next state. $A(s')$ is the list of all available actions in the next state. In our case, $A(s') = \{\text{None, forward, left, right}\}$. This way the aagent is able to learn a policy within the allocated 100 trials. See Table 1.

TO DO: 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?

Answer: Yes it does comes close to finding an optimal policy. The reason is that the most important states seems to be the (next_waypoint, traffic light) combination which the agent learns

$(\alpha, \gamma, \epsilon)$	No of success in last 10 trials
$(\ \cdot\ ,\ \cdot\ , 1.00)$	3
$(0.85, 0.50, 0.80)$	3
$(0.85, 0.00, 0.20)$	9
$(0.85, 0.50, 0.20)$	9
$(0.85, 0.95, 0.80)$	2
$(0.85, 0.95, 0.50)$	6
$(0.05, 0.50, 0.80)$	3
$(0.05, 0.00, 0.20)$	8
$(0.05, 0.50, 0.20)$	9
$(0.05, 0.95, 0.80)$	3
$(0.05, 0.95, 0.50)$	7
$(0.05, 0.95, 0.20)$	9

Table 1: Parameter tweaking table: I played around with a bunch of parameters to figure out an optimal set. It seems high γ means high weight on deciding to trying to move to the next waypoint. High ϵ means good policy making but bad immidiate (random) decision taking, so learns but does not use.

very well (8 possibilities and 4 actions, 100 trials with average deadline of 20 or so seems enough for visiting 32 possibilities several times). Since the number of traffic is very low and combined with the fact that the number of trials is also vey low, our smartcab does not always learn not to violate traffic rules and crash in another car. Also I do think that penalty of violating a traffic rule in presence of another traffic (higher accident chance) should be set much higher. Another funny thing I noted: Since our city is a torus, right turn on red is on average the correct decision.