“SmartCab”

P4: report

Prepared by Andrejs Zujevs

Implement a Basic Driving Agent

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?***

Sometimes the agent was obtained a destination point, but this happened during a long run time, also the agent behavior randomized.

The agent` world is cyclic and the agent living in multi-agent environment which also will described as:

- **discrete** – states of environment are determined (agent locations in the grid space which is cyclic, traffic lights with green and red color in NS and EW direction). In the world acting more than one agent with their own location and head direction. All states are discrete or not continuous;
- **partially observable** – the agent can get information of their current location traffic light state and if present other agent’s the next action, get deadline and destination point and also can get current location point. The agent can`t get information about traffic lights and agents in other locations of the world;
- **static** – the environment doesn`t change while the agent is acting;
- **deterministic** - the next environment state completely determined by current state and the agent action, no probability of the agent’s actions;
- **episodic** – the agent location completely depends on the agent previous actions and each episode’s reward depends on the agent`s action.

The active agent marked with red color and destination point also with red.

Inform the Driving Agent

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?***

An agent percept from sensors: oncoming agent with forward, left or right action, traffic light state (red or green) in NS or EW direction.

There were identified such appropriate agent's states:

Turn right allowed: light is green and the next waypoint is right or light is red and agent from left not going forward.

Forward allowed: light is green and the next waypoint is forward.

Turn left allowed: light is green and the next waypoint is left and not oncoming agent with forward or right.

Need to wait: light is red and next waypoint is not turn right or light is red and next waypoint is right and is oncoming agent from left forwards.

Those states are appropriate for the problem, because they include information about intersection light color, agent's next waypoint, and other agents in the intersection and theirs next waypoints.

To create more compact agent states' space, I suggest to use only 3 variables:

Light color + Next waypoint + GoStatus

where GoStatus is simplified state depended on light color, next waypoint, presence of other agents in the intersection and their next waypoints.

<u>GoStatus definition is:</u>

Set GoStatus to 'go' for such combinations of environment states:

- if color == 'red' and next_waypoint == 'right' and other_agent_waypoints['right'] != 'forward', then GoStatus = 'go'
- if color == 'green' and next_waypoint == 'left' and (other_agent_waypoints['oncoming'] != 'forward' or other_agent_waypoints['oncoming'] != 'right') , then GoStatus = 'go'
- if color == 'green' and (next_waypoint == 'forward' or next_waypoint == right'), then GoStatus = 'go'

Set GoStatus to 'stop' for such combinations of environment states:

- if color == 'red' and next_waypoint == 'right' and other_agent_waypoints['right'] == 'forward', then GoStatus = 'stop'

- if color == 'green' and next_waypoint == 'left' and (other_agent_waypoints[‘oncoming’] == 'forward' or other_agent_waypoints[‘oncoming’] == 'right') , then GoStatus = ‘stop’
- if color == 'red' and (next_waypoint == 'forward' or next_waypoint == 'left'), then GoStatus = ‘stop’

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?

The total number of states is: 2 x 3 x 2 = 12

2x – traffic lights (red and green) or traffic directions (SN, EW);

3x – next waypoint (forward, right, left);

2x – GoSattus (go, stop);

Agent actions: none, right, left, forward.

Q-Learning algorithm can find optimal policy for any finite Markov Decision Process and as our states and actions are finite, then Q-Learning algorithm is reasonable for the *smartcab* problem. If the states and actions number would be infinite, then Q-Learning algorithm wouldn`t be used or Q-Learning should be extended and constrained in time, interpolating states map’s entries, as described in the paper.

Taking into account the total states number and agent actions number, should be used 12 x 4 = 48 matrix entries or Q matrix with 12 rows and 4 columns.

As Q-Learning algorithm need many iterations to learn Agent behavior and suggested state space size is relatively small, then I think this number is effective for this problem, because Q(s,a) value updating would arise more frequently than if state space would be large.

Implement a Q-Learning Driving Agent

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?

After Q-Learning algorithm implementation agent behavior become more effective and after some trials the Agent obtained destination point before deadline and in effective way without negative rewards. Such ‘intelligent’ behavior occurred due to using best

possible action (with maximal Q(s,a) value) for the agent's current state instead of using random action selection.

Improve the Q-Learning Driving Agent

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?

Parameters set 1: alpha = 0.15, gamma = 0.15

Results: final agent obtained destination point at 8/20 (in 12 steps) deadline without negative rewards.

Parameters set 2: alpha = 0.30, gamma = 0.15

Results: final agent obtained destination point at 30/50 (in 20 steps) deadline without negative rewards.

Parameters set 3: alpha = 0.60, gamma = 0.15

Results: final agent obtained destination point at 13/25 (in 12 steps) deadline without negative rewards.

Parameters set 4: alpha = 0.95, gamma = 0.15

Results: final agent obtained destination point at 24/30 (in 6 steps) deadline without negative rewards.

Parameters set 5: alpha = 0.95, gamma = 0.30

Results: final agent obtained destination point at 15/35 (in 20 steps) deadline without negative rewards.

Parameters set 6: alpha = 0.95, gamma = 0.60

Results: final agent obtained destination point at 15/30 (in 15 steps) deadline without negative rewards.

Parameters set 7: alpha = 0.95, gamma = 0.95

Results: final agent obtained destination point at 19/30 (in 11 steps) deadline without negative rewards.

Parameters set 8: alpha = 0.65, gamma = 0.95

Results: final agent obtained destination point at 11/20 (in 9 steps) deadline without negative rewards.

Parameters set 9: alpha = 0.45, gamma = 0.95

Results: final agent obtained destination point at 23/30 (in 7 steps) deadline without negative rewards.

Parameters set 10: alpha = 0.25, gamma = 0.95

Results: final agent obtained destination point at 11/20 (in 9 steps) deadline without negative rewards.

Conclusion:

The best parameters set is **Parameters set 4.** Agent learns more quickly and small gamma parameter' value notice that future rewards are worth less than immediate 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? How would you describe an optimal policy for this problem?

The implemented agent acts in optimal way without penalties and in minimum possible time. Optimal policy for particular problem can be described as:

Turn right: light is green and the next waypoint is right or light is red and agent from left not going forward.

Forward: light is green and the next waypoint is forward.

Turn left: light is green and the next waypoint is left and not oncoming agent with forward or right.

Wait: light is red and next waypoint is not turn right or light is red and next waypoint is right and is oncoming agent from left forwards.