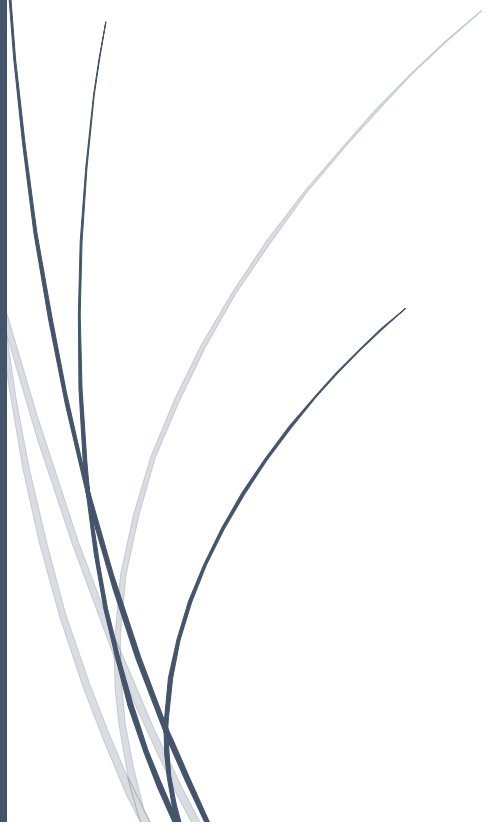
01.07.2016

P4 - Report

Train a Smart cab to drive



Pratheerth Padman

1. IMPLEMENT A BASIC DRIVING AGENT

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

Put simply, the agent takes a random turn at every intersection. The agent simply chooses a random action from all the possible actions (None, Forward, Right, Left).

The agent does eventually, purely co-incidentally reach its destination. It ignores all the traffic rules and since no deadline is being enforced, it has no compulsion to reset the trip if it does not reach on time.

2. IDENTIFY & UPDATE STATES

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

I have picked the following set of states:

1. **Next_Waypoint (waypoints to destination)** - This controls the direction and heading of the agent and its position in relation to the target.
2. **Traffic Lights (Red or Green)** - This state helps the agent learn the rules of the road. It prevents illegal travelling thereby avoiding collisions and/or delays.
3. **Oncoming, Left, Right (Direction and heading of the cars in the oncoming, left and right lanes)** - This helps the agent to sense the direction and heading of the cars in the oncoming, left and right lanes, helping avoid collisions. Each of these inputs can have four states - None, forward, left or right indicating the direction in which the car in those lanes are heading.

Deadline or time_step is also one of criteria used to assess the success of a trip. However, I find the modelling the agent based on this criterion is unlikely to result in better results as there isn't any way that, for example having less time, could increase the quickness with which the agent reaches its destination. To explain further, Deadline acts a reference point to reset the trip, but to consider it as criterion increases the complexity of the state space so drastically that

the agent cannot be expected to feasibly explore the entire state space in just 100 trials. Another factor to consider is, it would be dangerous to model the agent in such a way that it ignores traffic rules and/or collides with other agents in order to satisfy the argument of reaching the destination before the allotted time elapses.

3. IMPLEMENT Q – LEARNING

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

The agent senses the state, assesses the reward, calculates the maximum q value for a particular state – action pair and then takes the most suitable action.

Previously, the agent chose it action randomly. It wandered aimlessly around the environment. It flouted traffic rules, frequently crashing into other agents. No part of the agent's system placed any importance on reaching the destination.

On implementing Q learning, the agent moves more deliberately and purposefully towards its final destination. The change is due to the fact that agent learns the rules of the environment, learning which steps to take to earn maximum possible rewards and which to ones to not take to avoid getting negative rewards.

The working is explained below:

1. The agent senses the environment, stores the current state in self.state

2. The agent then performs an action. Previously, it used to be a random choice from the valid_actions in the Environment modue. Now, the action is based on the best_action function.

3. Based on the action taken, the agent collects a reward.

4. The agent arrives in a new environment. It senses the new environment and new inputs. All new inputs are stored in the updated_state.

5. The Q – Table is updated

How does the **best_action** function work??

a. First, we write a function to calculate the q – value. The value that will be returned from this function is the “q_new” value.

b. This q_new is used for many purposes. To start, it is used to update the Q – table. The calculation done, based on the updated state (taken from action 3), q_old (which is initially taken from the value (index value of a particular action of valid_actions) that returns as part of the best_action function), and the reward which is assigned in the Environment module.

c. We then write a function get_q to get the particular value of key ‘q’, which is a function of state and action. Initially (or in the absence of state – action pairs) it is set to 3 (higher than any of the rewards) in order to “motivate” the agent to explore all the state - action pairs as it was found that on equating it to 1, the agent would, eventually, get stuck in a loop where the reward and maximum q value constantly equates to 1 which means that the agent has no motivation to move from its current position. To test this, I have tested with four values as the intial q – value: 0, 1, 2 and 3.

The number of unsuccessful trials (U.T) for each value is as follows:

Value -0. U.T – 82

Value -1. U.T – 57

Value – 2. U.T – 3

Value – 3. U.T – 0

This phenomenon presumably occurs as a result of q –value and reward being equal for certain state – action pair. Due to the reward and q –value being equal or the often times, the reward being worse, leads the agent to be fixated on a single state till some dramatic state change occurs. Having the initial value as 3 completely eliminates this issue as it exceeds all the rewards, thereby encouraging the agent to explore all the state spaces.

d. Then, we write a function to find the maximum value of q for that state, action pair.

Firstly, we initialize a list to the variable q. This list will contain the q values which are called by using the get_q function as a function of the updated_state and action for the actions in Environment.valid_actions. This list will help in case there is multiple values of q_max for that state, action pair. This function will return q_max and q.

4. ENHANCE THE DRIVING AGENT

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?

To achieve the final version of the agent, I have tuned the alpha and gamma values.

The alpha value signifies the learning rate of the agent. The learning rate determines to what extent the newly acquired information will override the old information. A factor of 0 will make the agent not learn anything, while a factor of 1 would make the agent consider only the most recent information

The gamma values signify the discount factor. It determines the importance of the future rewards. A factor of 0 will make the agent consider only the immediate rewards while a factor of 1 will push the agent to strive for long term rewards.

The trials I've taken into account has been tabulated below:

Sl No.	Alpha	Gamma	Penalties	No. of Trials with Penalties	Unusuccessf
1	0.5	0.5	12	7	3
2	0.5	0.6	8	6	1
3	0.5	0.7	10	7	98
4	0.5	0.9	5	5	90
5	0.5	0.3	8	4	0
6	0.5	0.2	16	9	2
7	0.7	0.5	11	7	1
8	0.9	0.5	11	7	0
9	0.9	0.3	11	6	3
10	0.9	0.2	10	7	3

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?

Yes, the agent finds an optimal policy with a learning rate of 0.5 and a discount factor of 0.3. Here, the agent reaches its destination 100% of the time. Among the trials that can be reasonably considered (> 90 % success), this particular combination provides us with the least number of penalties with 8 total penalties. It is also the combination that has the minimum "Number of trials with penalties" with 4. It means the effective success rate (the number of successful trials with 0 penalties) is at 96 % which is considerably high.

The penalties primarily occur during the early trials where the agent is still exploring the state space. Higher the number of trials the agent is exposed to, the higher will be its effective success rate due to being thoroughly exposed to the state space.