Smart Cab

by

Jonathan Lee
2016.05.23

Table of Contents

Implement a Basic Driving Agent

In this step, we process inputs:

- Next waypoint location
- Intersection state
- Current deadline value

Created an output:

- Random move of either none, forward, left or right

Code implementation:

```
possibleActions = [None, 'forward', 'left', 'right']
action = random.choice(possibleActions)
reward = self.env.act(self, action)
```

Note that in this case the inputs are irrelevant since we are just randomly moving the agent around.

Observation:

The agent just randomly moves around. If the state just so happens to be the waypoint location, then the agent moves and is rewarded +2. If the move is illegal, then the agent cannot move and is awarded -1. If the move is legal but is not the waypoint location, then the agent is awarded -0.5.   And finally if the agent stays, it is awarded +0. After some dumb luck, the agent will eventually reach the destination. Sometimes, it will be able to randomly do this within the deadline and is awarded +10 extra points. But most of the time, it eventually reaches the goal way beyond its deadline.

Identify and update state

Given the primary inputs:

- Waypoint location
- Light color
- Traffic coming from left
- Traffic coming from right
- Traffic oncoming

We can generate a state space which includes a combination of the following inputs:

- inputs['light']
- inputs['oncoming'']
- inputs['left']
- Self.next_waypoint

I have decided not to include the inputs['right'] because when it is a red light it doesn't matter if the car is coming from the right since that won't prevent you from turning right. If the light is green, again you are free to go straight and turn left or right and the traffic coming from right won't matter because you always have right of way.

I have also decided not to include the deadline as the input because the smartcab is designed to move the same regardless if the deadline is 40, 20, 10 or even 1. Instead, I have used to the deadline to determine whether we pass or fail. If the deadline reaches zero then I log a fail. This is explained in further detail under Enhance the driving agent.

Thus, an example state will contain:

state= [light, oncoming, left, waypoint]

Where we have the following situations:

light = [red, green]
left = [forward, left, right, None]
oncoming=[forward, left, right, None]
waypoint=[forward, left, right, None]

This gives 512 different state space combinations to learn from.

In summary, there are two basic light states. Then within each of these light states, it is further broken down to more states depending on the location of traffic from oncoming, left and waypoints. Obviously, the light is important because if it’s red, your car cannot go anywhere except to have the possibliity of going right. And if it’s green you can pretty much go anywhere unless there’s oncoming traffic. Cars from the left is important because this will affect the status of a car being able to turn right when the light is red. Cars oncoming is important because this will affect the possibiilty of being able to turn left when the light is green. Finally, the waypoint is VERY important because this lets the cab know what is the best move. And these are the only four inputs we need to represent in the state space.

Implement Q learning

Initially, when implementing Q learning I used the following equation to update Q:

Q(s,a) = R(s,a) + gamma*(max(Q(s’,a’)))   # I have assumed alpha = 1

Here s is the previous state, a is the previous action and s’ is the next state and a’ is the next action.

Initially, gamma was chosen to be 0.8. I also initialized all the Q-states to zero because I want the agent to have a clear brain and learn everything from scratch.

And then for the action, I am selecting the action where the Q-state value is the highest with the following policy equation.

pi(s) = argmax(Q(s,a))

Initially with the gamma randomly chosen at 0.8. It is performing much worse than the random state. What I mean by worse is that as the agent is learning incorrectly. This means that the Q-value for the wrong action is the highest value. Therefore, the agent may begin to run around in circles and/or get stuck in a None state. And the score will become highly negative. This is worse than the random state because at the very least the Agent will randomly be able to make it home. But here the agent is unable to make it home initially and changes must be done to make it function properly. This will be explored in further detail in the following section.

Enhance the driving Agent

Here is the full equation and test out how it goes. The learning equation:

Q(s,a) := Q(s,a) + alpha*(R(s,a) + alpha*max(Q(s',a')) - Q(s,a))

Notice that if alpha, the learning rate, is 1 the equation is the same as the equation in the section Implementing Q-Learning. Gamma is the discount factor. So, it is time to experiement with the learning rate and discount factor.

First off, keep in mind that there are two phases in this Q-learning implementation. The first phase is the training phase. Next, is the trial running phase. During the training phase, the agent will travel around randomly and learn the environment. After the training phase, the agent will move greedily and keep score.

Assume that one knows nothing about the states and initialize all the Q to zero and let the agent randomly travel around for a long period of time (2200 training steps) before it is released into greedy mode. To create a training step first a variable was initialized called self.training_steps and each time the Agent.update has been called, this self.training_steps variable is incremented. The agent is programmed to run around randomly for 2200 of these steps. This number of training steps was chosen because with about 512 states, this will ideally allow the agent travel to each state around 4 times. These 2200 training steps will generally take about 80 trials.

A tally of pass and fail was also created. For any trial that Passes, this is detected by searching for a condition where the reward is greater than or equal to 10. This is because when the agent reaches the goal it will receive 10 points. For any trail which fail, this situation is detected where the deadline reaches zero. If the agent reaches goal, then the zero should never be reached!

These tallies were created to see how many has passed/failed during training phase and the actual trial phase. Recall earlier that the training phase is during the initial 2200 training steps where I allow the agent to randomly choose anywhere. Then after that, training ends and the agent is allowed to greedily choose the move based on the policy.

pi(s) = argmax(Q(s,a))

A dictionary which logs which trial number has passed or failed AFTER the training phase has also been created. This way, it can can easily be seen how the agent performs after it has been trained. Also note that there are a few more dictionaries which log the number of good,ok, or bad moves for the remaining trials. This will also help gauge if the policy learned is optimal.

alpha	gamma	training_steps	case	pass	fail
1	1	2200	worse	0	24
1	0.8	2200	worse	0	24
1	0.5	2200	worse	0	21
1	0.2	2200	worse	0	25
1	**0.01**	**2200**	**worse**	**21**	**2**
0.8	0.01	2200	worse	0	21
0.5	0.01	2200	worse	0	23
0.01	0.99	2200	worse	0	20
0.99	0.01	2200	worse	21	3

Table 1 - Quick summary of the describing worse case pass or fail AFTER the training the agent

Above, I have generated a table (See Table 1) to experiment with different alpha and gamma values. And it looks the best strategy is clearly for alpha to approach 1 while gamma approaches 0. This makes perfect sense.

The reason why a low gamma (discount factor) value is chosen is because one wants to give more priority to the immediate reward. In each trial, one does not know where the goal is located since it is randomly generated. So, it makes sense to favor the immediate reward because the information given is that the highest reward state is the waypoint and we want to encourage the agent to take that route. The low gamma will allow the Q to converge toward the immediate reward instead of the future rewards.

Also, note that the chosen alpha (learning rate) was chosen to be as high as possible. As, alpha equivalent to zero would have the agent learn nothing. As the highest alpha will allow the agent to learn the most recent information which again makes sense. We want the alpha to learn to choose direction in the current state which it is in.

In summary, in this current game plan, first, initialize all the Q to zero. And let the agent learn through 2200 training steps. Then, after correct policy is gained through the 2200 training steps, the agent can greedily choose the correct policy.

The optimal alpha=1 and gamma=0.01 setting has been simulated a countless number of times and from observation, the agent is reaching the goal pretty much every time after this. Worse case, after training the agent properly, is 21 pass and 2 fail in the final 23 trials. This clearly meets the specification of learning a correct policy within 100 trials. It performs better than the initial Q-learning implementation since initially the gamma was chosen at a very high value of 0.8. And it was not discounting the future rewards and therefore it was initially either getting stuck or going in circles.

But in this enhanced learning agent, it is very definitive that a near perfect optimal policy and minimizes penalties. With a very RARE occasion, I can see that only in about 1 out of final 20 trials, after training for approximately 80 trials, where an optimal policy isn't learned. Here there may some bad moves. An example of a bad move was when the light is red and the waypoint is left and the agent tries to go left. So, it was disobeying traffic rules. Another example, that seem to be caught was a situation where the agent tries to turn left with oncoming traffic going right. Here, it caused an accident. In some cases, despite these bad moves the agent still made it to the goal within the deadline. Keep in mind that, these situations is very rare, one would have to simulate perhaps 30 times before seeing these errors occur.

Finally, in the majority case, there are only good moves. The agent going the way of the waypoint and not breaking any laws! Therefore the agent has chosen the most direct and optimal path. Also note that all the score of all these cases end in positive. The agent clearly reached the goal within the deadline. This agent has learned to function, worse case, approximately 85% of the time. This is better than the required 70%. In the best case scenario, which I have seen, this trained agent was perfect, passing 100% of the last 26 trials!