README.md

Project 4: Reinforcement Learning

Train a Smartcab How to Drive

Install

This project requires **Python 2.7** with the pygame library installed

Code

Template code is provided in the `smartcab/agent.py` python file. Additional supporting python code can be found in `smartcab/enviroment.py`, `smartcab/planner.py`, and `smartcab/simulator.py`. Supporting images for the graphical user interface can be found in the `images` folder. While some code has already been implemented to get you started, you will need to implement additional functionality for the `LearningAgent` class in `agent.py` when requested to successfully complete the project.

Run

In a terminal or command window, navigate to the top-level project directory `smartcab/` (that contains this README) and run one of the following commands:

```
python smartcab/agent.py  python -m smartcab.agent
```

This will run the `agent.py` file and execute your agent code.

Implement a Basic Driving Agent

To begin, your only task is to get the smartcab to move around in the environment. At this point, you will not be concerned with any sort of optimal driving policy. Note that the driving agent is given the following information at each intersection:

The next waypoint location relative to its current location and heading. The state of the traffic light at the intersection and the presence of oncoming vehicles from other directions. The current time left from the allotted deadline. To complete this task, simply have your driving agent choose a random action from the set of possible actions (None, 'forward', 'left', 'right') at each intersection, disregarding the input information above. Set the simulation deadline enforcement, enforce_deadline to False and observe how it performs.

QUESTION: 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?

- Yes. With no deadline and a 'random walk' through the city streets it will eventually make it to the destination but will take a very long time to do so. Because it listens to no inputs the only interesting behavior is it's complete randomness.

Code on commit: 0b302db

Inform the Driving Agent

Now that your driving agent is capable of moving around in the environment, your next task is to identify a set of states that are appropriate for modeling the smartcab and environment. The main source of state variables are the current inputs at the intersection, but not all may require representation. You may choose to explicitly define states, or use some combination of inputs as an implicit state. At each time step, process the inputs and update the agent's current state using the self.state variable. Continue with the simulation deadline enforcement enforce_deadline being set to False, and observe how your driving agent now reports the change in state as the simulation progresses.

QUESTION: 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?

OPTIONAL: 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?

- State is defined in agent.py now. Inputs include 'violation' and next_waypoint. The viloation state combines the traffic and the light to create one aspect of the state space in an effort to reduce the total state space while preserving the various outputs the of traffic. Each of these states include valuable input that is needed to make the next action. I believe the current 'viloation' state is making assumptions around the model and takes away from the exercise in building a reinforcement learner as we are making assumptions around the cost of such an action. (it was suggested by the reviewer, although I'm worried I might have misunderstood.)

These inputs were decided on so as to minimize the q-table which allows for optimal learning assuming all needed states are included. The biggest ommission is 'deadline' which is because that information is embedded in 'next_waypoint'. The viloation state is very powerful as it includes all inputs that can cause a negative reward and powers the 'rules of the road'.

Code on commit: 9758de9

Implement a Q-Learning Driving Agent

With your driving agent being capable of interpreting the input information and having a mapping of environmental states, your next task is to implement the Q-Learning algorithm for your driving agent to choose the best action at each time step, based on the Q-values for the current state and action. Each action taken by the smartcab will produce a reward which depends on the state of the environment. The Q-Learning driving agent will need to consider these rewards when updating the Q-values. Once implemented, set the simulation deadline enforcement enforce_deadline to True. Run the simulation and observe how the smartcab moves about the environment in each trial.

The formulas for updating Q-values can be found in this video.

QUESTION: 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?

Well that is very cool. The first time I wrote it I didn't have randomness in the 'select action' method and the car just went around in a circle, always taking a right. The changes here are significant of course. The random driving car had a very low probability of getting to the destination and while at the start they looked very similair, by the end of the simulation every simulation resulted in a success. After turning on the simulator view it also looked like near the end it was pretty efficient as well. Looking at the q-table was the easiest way for me to see that by the end of the runs the cab had properly learned the rules of the road and to follow the waypoint:

State	Ouput	Optimal?
'violation: , next_waypoint: right': {'right': 27.843821572888867, None: 0.0}	Yes	
'violation: left, next_waypoint: right'	{'forward': -0.45, 'right': 1.8, None: 0.0, 'left': -0.45}	Yes

---

State	Ouput	Optimal?
'violation: red, next_waypoint: forward'	{'forward': -0.9, 'right': -0.45, None: 0.0, 'left': -0.9}	Yes
'violation: left, next_waypoint: forward'	{'forward': 8.642614959180001, 'right': -0.45, 'left': -0.45}	Yes
'violation: left, next_waypoint: left'	{'left': 12.438}	No
'violation: red, next_waypoint: right'	{'right': 23.33036904894389, 'left': -0.9}	No
'violation: , next_waypoint: left'	{'left': 29.075114953276987}	Yes
'violation: , next_waypoint: forward'	{'forward': 38.21669650442579, None: 0.0, 'left': -0.45}	Yes
'violation: red, next_waypoint: left'	{'forward': -0.9, 'right': -0.1009163384892992, None: 0.0, 'left': -0.9}	Yes

(Note: table without Greedy Epsilon implemented)

As we can see from the above algorithm we can see that while it is near optimal the final q-table still makes mistakes on waypoint 'right' and violation 'red' and violation left and waypoint left. This could be because it a large reward (finishing) by violating the rules of the road, therfore making it positive and never leaving the locally optimal strategy. To make this more realistic we could also change the rewards so that in no case would it make sense to disobey the rules of the road. Another adjustment that could be made would be to add the greedy-epsilon algorithm and adjust the states as there is a clear interplay between violation and next_waypoint but, again, I feel like that is defeating the purpose of a general q-learner.

Code on commit: e22ee58

Improve the Q-Learning Driving Agent

Your final task for this project is to enhance your driving agent so that, after sufficient training, the smartcab is able to reach the destination within the allotted time safely and efficiently. Parameters in the Q-Learning algorithm, such as the learning rate (alpha), the discount factor (gamma) and the exploration rate (epsilon) all contribute to the driving agent's ability to learn the best action for each state. To improve on the success of your smartcab:

Set the number of trials, n_trials, in the simulation to 100. Run the simulation with the deadline enforcement enforce_deadline set to True (you will need to reduce the update delay update_delay and set the display to False). Observe the driving agent's learning and smartcab's success rate, particularly during the later trials. Adjust one or several of the above parameters and iterate this process. This task is complete once you have arrived at what you determine is the best combination of parameters required for your driving agent to learn successfully.

QUESTION: 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?

- alpha = starting alpha (or const if no alpha-d)
- gamma = starting gamma (or const if no gamma-d)
- alpha-d = alpha change calculation
- gamma-d = gammma change calculation
- success% = success rate at iteration 100
- max-% (t) = trial that best success rate was achieved
- effort = indicates how slowly the car got to destination (low is good) This is a measure (over 15) of the number of iterations it took to get to the destination so the faster (lower) it got there the better.

alpha	gamma	alpha-d	gamma-d	success%	max-% (t)	effort

.9	.9	self.alpha / ln(t + 2)	None	99	63	.77
.9	.9	self.alpha / ln(t + 2)	self.gamma / ln(t)	98	97	.80
.9	.9	None	None	99	98	.81
.1	.1	None	None	97	98	.88
.9	.1	None	None	97	97	.77
.9	.9	self.alpha / t	None	98	99	.82
.5	.5	None	None	98	99	.8

I believe the first row (.9, .9, w/alpha decay) is the best. It achieves the highest success% and has one of the lowest effort scores. This indicates the weighing the finnal outcome heavily is in the best interest of this learner by keeping the gamma value high through out. I am a little worried the alpha decay is not really needed but these specific findings are consistent with the lecture learnings. Even though these are run over 100 trials, only one 'learning' run was done for each row and could have led to some inaccurate numbers.

The final driving agent works very well. With very high success rate, it seems to have learned the rules quickly (max-t 63), while also reaching the destination with minimal effort. The table below shows that while this is a stronger learner it is not optimal. This currently preforms well when there is an empty violation parameter which confirms what I said in the previous question, that combining some states, would lead to greater accuracy. The optimal policy is outlined in the table with changes in the 'Optimal Change' column. The incorrect policy lines were discussed in the previous question.

(copied from above)

State	Ouput	Optimal?	Optimal Change (if not)
'violation: , next_waypoint: right'	{'right': 27.843821572888867, None: 0.0}	Yes	
'violation: left, next_waypoint: right'	{'forward': -0.45, 'right': 1.8, None: 0.0, 'left': -0.45}	Yes	
'violation: red, next_waypoint: forward'	{'forward': -0.9, 'right': -0.45, None: 0.0, 'left': -0.9}	Yes	
'violation: left, next_waypoint: forward'	{'forward': 8.642614959180001, 'right': -0.45, 'left': -0.45}	Yes	
'violation: left, next_waypoint: left'	{'left': 12.438}	No	{ None: 0.0 }
'violation: red, next_waypoint: right'	{'right': 23.33036904894389, 'left': -0.9}	No	{ None: 0.0 }
'violation: , next_waypoint: left'	{'left': 29.075114953276987}	Yes	
'violation: , next_waypoint: forward'	{'forward': 38.21669650442579, None: 0.0, 'left': -0.45}	Yes	
'violation: red, next_waypoint: left'	{'forward': -0.9, 'right': -0.1009163384892992, None: 0.0, 'left': -0.9}	Yes	

(Note: table without Greedy Epsilon implemented)

QUESTION: 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?

I think it does get close, but not completely optimal. The penalties are built into the learner and given the q-learning algo is much like a gradient decent algo (the same?) this is learning to incur the lowest amount of penalties.

commit: e1617ff