Tasks

Project Report

You will be required to submit a project report along with your modified agent code as part of your submission. As you complete the tasks below, include thorough, detailed answers to each question*provided in italics.*

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?*
Answer: When agent chooses actions randomly at each iteration using `random.choice(Environment.valid_actions[1:])` it's able to reach destination 100% of the time when regardless of deadline. In majority of situation agent revived −1 or −.5 reward. Average number of steps to destination was 12.6

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?*

I chose the parameters **`oncoming`**, **`left`**, **`light`**, and **`the next waypoint`** for agent states. I chose these parameters because:

- **`oncoming`** and **`left`**: The agent needs to know about any traffic which is oncoming or coming in from the left as it has to yield to cars on these roads according to the traffic rules.
- **`light`**: Agent has to follow traffic rules and can not move unless allowed to do so
- **`next_waypoint`**: Route the agent needs to take to reach destination .

I dined't choose "right" parameter as has little to no use, because if light is green, only data at "left" is useful, if light is "red", having a car on the right or not, I can still turn right, just need to be careful.
After implementing state I am able to see the state information in the simulator.

- It is also required to justify the choice of including/excluding deadline, why wasn't it included? To answer this, consider the size of the state if deadline is included, also consider how this might affect the behavior of the agent on how it obeys traffic rules when the time is running down.

Including the deadline limits all the possible combination agents can visit to the time limit. I chose to exclude the trial deadline from the state recognition. Due to the rules of the current simulation, I cannot think of a situation in which policy would change during the late stages of a trial versus early stages. Inclusion of this feature would drastically increase the number of training examples needed for convergence to the optimal policy. Perhaps if the simulation were to be changed to allow an agent to actually move through a red light (incurring a penalty), then the optimal policy could reward breaking traffic laws if the deadline to destination was approaching. However, a the simulation is set-up knowledge of the deadline is not useful and would simply perturb the curse of dimensionality.

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?*
To calculate Q value I used
s- State a- Action r- reward for entering state; s*'* - new state
Update Q^ (S,a) with & (learning rate) + discounted value of the next state agragmax{ Q^ (S',a')}

After implementing Q learning I have noticed that the agent doesn't make as many mistakes (earning negative rewards) on average number of steps to each the goal reduced by 1/2 to 6.5. After the first run the agent took a very long time (upto 3-4x times to reach destination) after this number of steps to reach the destination was reduced to 1- 1.5 X the average times.

I did not choose to monitor traffic from the right, as theoretically this information is redundant with the traffic light color. Assuming an oncoming agent obeys traffic laws and does not run a red light, there is no situation that could be altered by the presence of a car from the right. For instance, if my agent's light is green, the approaching agent will have a red light and must give right of way if turning right. If my agent's light is red, then the only viable MOVEMENT is a right turn which would be unaffected by a car approaching from the right. Thus inclusion of this variable in the state would not provide any value, but would exponentially increase the number of learning steps needed for Q-learning convergence.

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?*

After running 21 tests I tested effect of changing tuning parameters alpha (Q-learning , gama and epsilon. The parameter that had most dramatic effect on agent performance are alpha and gama while epsilon did not affect results in a big way. Te best combination of parameters I found is Alpha =1, gamma= 0.87 and epsilon 1.

self.alpha	self.gamma	epsilon	% Destination reached
1	1	1	0.983
0.9	1	1	0.988
0.8	1	1	0.798
0.7	1	1	0.558
0.6	1	1	0.658
0.5	1	1	0.391
1	1	0.9	0.336
1	1	0.8	0.306
1	1	0.7	0.343
1	1	0.6	0.297
1	1	0.5	0.281
1	0.9	1	0.987
1	0.8	1	0.988
1	0.7	1	0.992
1	0.6	1	0.991
1	0.5	1	0.99
1	0.4	1	0.994
1	0.3	1	0.993
1	0.2	1	0.992
1	0.1	1	0.996

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 *believe* so, After about 20 trips the agent starts getting a lot less negative rewards and learns optimal policy .

Intuitively, the optimal policy for the agent would be as follows: 1) obey traffic laws, and then 2) move in the direction of the way_point provided. If traffic laws occlude the way_point direction, do not move.

This program version generally took ~20 trials to converge to an optimal policy. If one knew the simulation would incorporate new features at some point, I would choose to use this version of the program, as it is most adaptable to increased state-action space.

The Learning Agent generated from the agent program accomplishes this policy through Q-Learning iteration. I used two strategies to increase the efficiency by which the agent learned the optimal policy. The first version initialized Q^ to zero, and then employed monotonic functions for epsilon and alpha to control the rates at which the agent explored the the state-action space versus exploiting knowledge gained. The discount value for future reward was found to be optimal near the limit of 0. This is due to egocentric nature of the agent, as well as the random aspects of state that cannot be predicted ahead of time. Were the simulation changed to an allocentric view, where the agent could sense where it was in relation to the destination, etc, then gamma term would be a more important parameter for optimal performance and policy generation.

To see that the agent is correctly following the policy and not breaking traffic law, for which agent would incur penalties I examine the state table looking at the action the agent took and the penalties / rewards received for last 20 trials. I see that agent took action leading to penalty in 0.3% of the time comparing to 18% of the time agent took a wrong action. I also see that the agent is taking similar number of steps (27 on average) in the last 20 trips than it did in the first 20 (30) however its doing this with less penalties. In last 20 tries agent reached cumulative reward of 498 vs 455 in first 20. For reference please see results.csv table

However, for the simulation as it stands, the strategy that appeared most efficient was to set the Q^ initialization to an overly optimistic value that exceeded any possible single action reward. This change, concomitant with subsequent changes to make alpha constant (set to 1) and a full discount factor (gamma = 0) generated a highly efficient learning agent that almost always found the destination within small number of tries.