Report ML Nanodegree P4: Train a Smartcab to drive

Author: Christian Graber

Implement a basic driving agent

Implement the basic driving agent, which processes the following inputs at each time step:

- Next waypoint location, relative to its current location and heading,
- Intersection state (traffic light and presence of cars), and,
- Current deadline value (time steps remaining),

And produces some random move/action `(None, 'forward', 'left', 'right')`. Don't try to implement the correct strategy! That's exactly what your agent is supposed to learn.

Run this agent within the simulation environment with `enforce_deadline` set to `False` (see `run` function in `agent.py`), and observe how it performs. In this mode, the agent is given unlimited time to reach the destination. The current state, action taken by your agent and reward/penalty earned are shown in the simulator.

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

The agents actions are random. There is no correlation between the inputs and the action it takes. Success within the extended deadline is left to chance.

With enforce_deadline set to False, the agent has an additional 100 steps on top of the regular deadline. The additional 100 steps are a hard time limit. The agent sometimes makes it to the target and sometimes it hits the hard time limit before reaching the target. When the hard time limit is hit a new trial is started. By default the simulation has 100 trials.

Example of success, the agent hits the destination by chance:

```
Environment.act(): Primary agent has reached destination!
```

Example of failure, the agent does not hit the destination:

```
Environment.step(): Primary agent hit hard time limit (-100)! Trial aborted.
```

Identify and update state

Identify a set of states that you think are appropriate for modeling the driving agent. The main source of state variables are current inputs, but not all of them may be worth representing. Also, you can choose to explicitly define states, or use some combination (vector) of inputs as an implicit state.

At each time step, process the inputs and update the current state. Run it again (and as often as you need) to observe how the reported state changes through the run.

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

If all the current state is taken into account we have a state space size of:

light+oncoming+left+right+next = 2*4*4*4*3 = 384

Some states can be ignored. Specifically, oncoming right turn and left right turn. They will not interfere with the smartcab and can be switched to None. This reduces state space to:

2*3*3*4*3 = 216.

Still very large state space. To simplify the state space the traffic could be ignored in first approximation. The traffic could be seen as a statistical disturbance, or noise. Q learning can handle some noise and still converge. With the traffic ignored, the state space is now simply:

light+next = 2*3 = 6

Here are those 6 states with their concatenated names:

```
self.states = ['greenforward','greenleft','greenright',
'redforward','redleft','redright']
```

Watching the states progress during simulation with random selection of action:

```
[DEBUG] state: redleft
[DEBUG] state: greenleft
[DEBUG] state: redright
[DEBUG] state: greenright
[DEBUG] state: redforward
[DEBUG] state: greenright
[DEBUG] state: greenright
[DEBUG] state: redleft
[DEBUG] state: redleft
[DEBUG] state: redright
[DEBUG] state: redright
[DEBUG] state: greenright
...
```

Implement Q-Learning

Implement the Q-Learning algorithm by initializing and updating a table/mapping of Q-values at each time step. Now, instead of randomly selecting an action, pick the best action available from the current state based on Q-values, and return that.

Each action generates a corresponding numeric reward or penalty (which may be zero). Your agent should take this into account when updating Q-values. Run it again, and observe the behavior.

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

Use Q learning update rule. Future rewards seem non-deterministic. Instead approximated future rewards with constant. Here is the implemented update rule:

```
nextstate = nowstate * ( 1 - alpha ) + alpha * ( reward + gamma * 2.0)
```

Q matrix with random selection of action after 100 trials:

```
self.alpha = 0.5
self.gamma = 0.5

               forward  freeze  left  right
greenforward       3.0     1.0   0.5    0.5
greenleft          0.5     1.0   3.0    0.5
greenright         0.5     1.0   0.5    3.0
redforward         0.0     1.0   0.0    0.5
redleft            0.0     1.0   0.0    0.5
redright           0.0     1.0   0.0    3.0
```

Q matrix with picking action from Q matrix, while it is being trained. This does not work. The Q matrix does not converge to Q*.

```
self.alpha = 0.5
self.gamma = 0.5

               forward  freeze  left  right
greenforward       3.0     0.0   0.0    0.0
greenleft          0.5     0.0   0.0    0.0
greenright         0.5     0.0   0.0    0.0
redforward         0.0     0.0   0.0    0.0
redleft            0.0     0.0   0.0    0.0
redright           0.0     0.0   0.0    0.0
```

Enhance the driving agent

Apply the reinforcement learning techniques you have learnt, and tweak the parameters (e.g. learning rate, discount factor, action selection method, etc.), to improve the performance of your agent. Your goal is to get it to a point so that within 100 trials, the agent is able to learn a feasible policy - i.e. reach the destination within the allotted time, with net reward remaining positive.

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?

The agent has to find a balance between exploration and exploitation. Exploration means picking an action at random. Exploitation means picking the best action from the Q matrix.

The implemented policy will randomly select exploration or exploitation based on parameter epsilon [0,1]. If a random sample in the range [0,1] is smaller or equal to epsilon a random action is picked. Otherwise the best action is picked. Epsilon will decrease linearly until a floor of 0.05 is reached. That means the agent will still explore at 5% of all moves.

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 Q matrix converges to Q*. Epsilon will have reached its floor with the large majority of actions being exploitative.

As can be seen in appendix A, progression of 100 trials, the agent reaches it destination with positive total reward reliably after about 50 trials.

Here is the Q matrix reached at the end of 100 trials with described policy:

	forward	freeze	left	right
greenforward	3.705915	1.000000	0.500000	0.500000
greenleft	0.499969	0.999939	4.875059	0.499939
greenright	0.500000	1.000000	0.499998	8.312500
redforward	0.000000	1.000000	0.000000	0.500000
redleft	0.000000	1.000000	0.000000	0.499996
redright	0.000000	1.000000	0.000000	3.000029

Appendix A: Progression of 100 trials

```
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.act(): Primary agent has reached destination! Total reward: -1.0
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.act(): Primary agent has reached destination! Total reward: 8.5
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.act(): Primary agent has reached destination! Total reward: 8.5
Environment.act(): Primary agent has reached destination! Total reward: 3.5
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.act(): Primary agent has reached destination! Total reward: 3.0
Environment.act(): Primary agent has reached destination! Total reward: 8.5
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.act(): Primary agent has reached destination! Total reward: 7.0
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.act(): Primary agent has reached destination! Total reward: 4.5
Environment.act(): Primary agent has reached destination! Total reward: 12.5
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.act(): Primary agent has reached destination! Total reward: 7.0
Environment.act(): Primary agent has reached destination! Total reward: 14.0
Environment.act(): Primary agent has reached destination! Total reward: 11.0
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.act(): Primary agent has reached destination! Total reward: 3.0
Environment.act(): Primary agent has reached destination! Total reward: 10.0
Environment.act(): Primary agent has reached destination! Total reward: 15.5
Environment.act(): Primary agent has reached destination! Total reward: 15.0
Environment.act(): Primary agent has reached destination! Total reward: 9.0
Environment.act(): Primary agent has reached destination! Total reward: 10.0
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.act(): Primary agent has reached destination! Total reward: 5.0
Environment.act(): Primary agent has reached destination! Total reward: 8.0
Environment.act(): Primary agent has reached destination! Total reward: 9.0
Environment.act(): Primary agent has reached destination! Total reward: 7.0
Environment.act(): Primary agent has reached destination! Total reward: 9.5
Environment.act(): Primary agent has reached destination! Total reward: 11.0
Environment.act(): Primary agent has reached destination! Total reward: 12.0
Environment.act(): Primary agent has reached destination! Total reward: 16.0
Environment.act(): Primary agent has reached destination! Total reward: 9.5
Environment.act(): Primary agent has reached destination! Total reward: 5.5
Environment.act(): Primary agent has reached destination! Total reward: 13.0
Environment.act(): Primary agent has reached destination! Total reward: 10.0
Environment.act(): Primary agent has reached destination! Total reward: 18.0
Environment.act(): Primary agent has reached destination! Total reward: 10.0
Environment.act(): Primary agent has reached destination! Total reward: 19.5
Environment.act(): Primary agent has reached destination! Total reward: 8.0
Environment.act(): Primary agent has reached destination! Total reward: 11.5
Environment.act(): Primary agent has reached destination! Total reward: 8.0
Environment.act(): Primary agent has reached destination! Total reward: 8.0
```

```
Environment.act(): Primary agent has reached destination! Total reward: 7.0
Environment.act(): Primary agent has reached destination! Total reward: 17.0
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.act(): Primary agent has reached destination! Total reward: 9.5
Environment.step(): Primary agent ran out of time! Trial aborted.
Environment.act(): Primary agent has reached destination! Total reward: 16.0
Environment.act(): Primary agent has reached destination! Total reward: 6.0
Environment.act(): Primary agent has reached destination! Total reward: 13.5
Environment.act(): Primary agent has reached destination! Total reward: 4.0
Environment.act(): Primary agent has reached destination! Total reward: 12.0
Environment.act(): Primary agent has reached destination! Total reward: 18.0
Environment.act(): Primary agent has reached destination! Total reward: 8.0
Environment.act(): Primary agent has reached destination! Total reward: 6.0
Environment.act(): Primary agent has reached destination! Total reward: 11.5
Environment.act(): Primary agent has reached destination! Total reward: 6.0
Environment.act(): Primary agent has reached destination! Total reward: 12.0
Environment.act(): Primary agent has reached destination! Total reward: 10.0
Environment.act(): Primary agent has reached destination! Total reward: 10.0
Environment.act(): Primary agent has reached destination! Total reward: 7.0
Environment.act(): Primary agent has reached destination! Total reward: 18.0
Environment.act(): Primary agent has reached destination! Total reward: 10.0
Environment.act(): Primary agent has reached destination! Total reward: 10.0
Environment.act(): Primary agent has reached destination! Total reward: 10.0
Environment.act(): Primary agent has reached destination! Total reward: 6.0
Environment.act(): Primary agent has reached destination! Total reward: 18.0
Environment.act(): Primary agent has reached destination! Total reward: 6.0
Environment.act(): Primary agent has reached destination! Total reward: 9.5
Environment.act(): Primary agent has reached destination! Total reward: 18.0
Environment.act(): Primary agent has reached destination! Total reward: 2.5
Environment.act(): Primary agent has reached destination! Total reward: 12.0
Environment.act(): Primary agent has reached destination! Total reward: 4.0
Environment.act(): Primary agent has reached destination! Total reward: 8.0
Environment.act(): Primary agent has reached destination! Total reward: 19.5
Environment.act(): Primary agent has reached destination! Total reward: 14.0
Environment.act(): Primary agent has reached destination! Total reward: 6.0
Environment.act(): Primary agent has reached destination! Total reward: 8.0
Environment.act(): Primary agent has reached destination! Total reward: 8.0
Environment.act(): Primary agent has reached destination! Total reward: 9.5
Environment.act(): Primary agent has reached destination! Total reward: 24.0
Environment.act(): Primary agent has reached destination! Total reward: 6.5
Environment.act(): Primary agent has reached destination! Total reward: 10.0
Environment.act(): Primary agent has reached destination! Total reward: 6.0
Environment.act(): Primary agent has reached destination! Total reward: 6.0
Environment.act(): Primary agent has reached destination! Total reward: 7.0
Environment.act(): Primary agent has reached destination! Total reward: 9.0
Environment.act(): Primary agent has reached destination! Total reward: 13.5
Environment.act(): Primary agent has reached destination! Total reward: 6.0
Environment.act(): Primary agent has reached destination! Total reward: 10.0
```