Udacity MLP4 – Smartcab – Neel Shah

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?

Eventually the smartcab does make it to the destination marker, though only sometimes (in other cases it hits the hard deadline of -100). This is expected behavior from a random choice algorithm – given infinite time it will explore all states, one of which will be the destination. Random choice agent loves nonsensical actions – sort of expected – but what's nice about the environment is the agent stays in the same position if it tries to run a red light, for example, or if interfering traffic is present. So we will be penalized for an unsafe action, but remain in the same location/heading state.

Another observation is that there is rarely any traffic in the vicinty of the smartcab agent. The vast majority of output sentences indicate there is no traffic oncoming, left, or right. This in turn implies the majority of unsafe actions taken by the agent are illegal red light crossings.

I should also note I'm suffering from the Windows BMP fault from running pygame on Ubuntu 14, which according to the Udacity support forums remains unresolved. So perhaps there is some interesting viz pattern I'm missing out on here.

Udacity MLP4 – Smartcab – Neel Shah

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?

The full set of states available to the smartcab seem to be: 1) traffic light 2) nearby traffic 3) current location 4) destination 5) next_waypoint, 6) deadline. Traffic light and nearby traffic represent important state variables for performing safe actions. Current location, destination, and next waypoint are also important for figuring out an optimal route.

Next_waypoint is particularly nice because it rolls location, heading, and destination into an optimal action based on a simple grid structure. If moving to next_waypoint is unsafe then we can compute an alternate route from location and destination. We could just drop these two and try to proceed to next_waypoint when it is safe. This would reduce the state space at perhaps some cost of efficiency.

Nearby traffic breaks down further into combinations of (oncoming, left, right) and (none, left, right, forward) – 64 possible traffic states. We can reduce this state space by eliminating nonsensical combinations, but it will be easier to just have a sparser Q-table. The traffic state is especially important to learn optimal actions because it has a direct impact on the safety-based reward/penalty to the agent.

We will drop the deadline as part of our state space since it doesn't actually add any more information about what optimal action to take. In addition the state space it creates would be too large for the agent to reliably visit them all and learn optimal actions. We may lose some 'urgency' behavior for the agent but since our primary goal is safety this is okay.

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?

For this configuration there are 384 states: traffic light (2) * nearby traffic (64) * next waypoint (3)

This seems an excessive number of states for Q-learning, especially since a significant quantity are nonsensical entries. Our Q-table will be sparse, and will not be able to make informed decisions about nonsensical states, or states that violate common US traffic patterns.

Our Q-table state keys will look like 'lwORL', where l (light state) = g or r, w (waypoint) = f, l, r and O/R/L (oncoming/left/right traffic) = n, f, l, r.

Udacity MLP4 – Smartcab – Neel Shah

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.
https://classroom.udacity.com/nanodegrees/nd009/parts/0091345409/modules/e64f9a65-fdb5-4e60-81a9-72813beebb7e/lessons/5446820041/concepts/6348990570923

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?

The most notable change after implementing Q-learning is that the agent slowly improves in the ways we would like. It slowly adjusts to preferring actions for which there are rewards. It begins to reach its destination in fewer steps. It incurs fewer penalties for unsafe actions. By the end of a 100 run trial the Q-learning agent approaches its destination with a greater success rate than a random agent.

This behavior is occurring because we implemented Q-learning! We have granted the agent a memory, stored in the form of a Q-table with states as rows and actions as columns. Initially the Q-table defaults to zeroes* making it no better than random chance. But as the agent picks an action and gains or loses reward points, it updates its Q-table to prefer some actions over others in a given state. We have also implemented the lookahead part of Q-learning – the agent takes into account (at a discounted *alpha* and *gamma* of 0.5 – we will optimize this later) the Q of the state it will end in as a result of the action.

* We have chosen to initialize the Q-table with zeroes as opposed to random numbers. The advantage of zeroes is that as soon as the agent obtains a non-zero reward for an action in a given state, it will decisively prefer this action. This is not guaranteed with random initialization. However initializing with zeroes means the max Q value of a given row will default to the first column, which is no action (random initialization would avoid this issue). To circumvent the infinite None issue, we introduce *epsilon* at 0.5 – it will choose to explore a random action 50% of the time, and consult the Q table the other 50%.

Udacity MLP4 – Smartcab – Neel Shah

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?

The table below contains results for various settings of alpha, gamma, epsilon. The initial implementation of Q-learning was 0.5 for epsilon, while exploring a few different values for alpha and gamma. The evaluation metrics we use for now are the success rate (% of runs the agent reaches its destination) and the penalty rate (% of actions with < 0 reward).

	Success rate	Penalty rate
(α, γ, ε) = (0.25, 0.25, 0.5)	0.60	740/2099=0.3525
(α, γ, ε) = (0.25, 0.5, 0.5)	0.58	994/2358=0.4215
(α, γ, ε) = (0.25, 0.75, 0.5)	0.53	948/2210=0.4290
(α, γ, ε) = (0.5, 0.25, 0.5)	**0.79**	**640/1973=0.3244**
(α, γ, ε) = (0.5, 0.5, 0.5)	0.56	832/2056=0.4047
(α, γ, ε) = (0.5, 0.75, 0.5)	0.46	982/2300=0.4270
(α, γ, ε) = (0.75, 0.25, 0.5)	0.57	748/2197=0.3404
(α, γ, ε) = (0.75, 0.5, 0.5)	0.49	1002/2413=0.4152
(α, γ, ε) = (0.75, 0.75, 0.5)	0.50	1033/2403=0.4299

We see a marked improvement in performance when gamma is low (0.25) and alpha is balanced (0.5). We now adjust epsilon to observe its effects on success and penalty rate.

Udacity MLP4 – Smartcab – Neel Shah

	Success rate	Penalty rate
(α, γ, ε) = (0.5, 0.25, 0.5)	0.79	640/1973=0.3244
(α, γ, ε) = (0.5, 0.25, 0.25)	**0.79**	**360/1650=0.2181**
(α, γ, ε) = (0.5, 0.25, 0.1)	0.71	185/2039=0.0907

A lower epsilon reduces the amount of exploratory random actions taken by the agent. This of course improves the penalty rate drastically but has a relatively low effect on success rate. In fact as epsilon gets too low we lose the exploration behavior of the agent, which reduces the effectiveness of the Q-table, which ends up lowering the overall success rate.

In order to further boost performance we explored a strategy for epsilon to adjust it dynamically with respect to the deadline, **ε = deadline / ep_decay_rate, where ep_decay_rate is a constant set to 100**. As the deadline approaches, the agent is less likely to pick a random action (explore) and more likely to consult its Q-table. We evaluate this approach against (α, γ, ε) = (0.5, 0.25, 0.25) using success rate and penalty rate averaged over five runs.

Average over five runs	Success rate	Penalty rate
(α, γ, ε) = (0.5, 0.25, 0.25)	.79/.86/.81/.85/.8 = 0.830	.218/.199/.202/.198/.192 = 0.202
(α, γ, ε) = (0.5, 0.25, decay)	.88/.89/.87/.93/.91 = 0.896	.199/.171/.184/.197/.183 = 0.187

We see that the decay strategy outperforms a static epsilon value by allowing the agent to be more exploratory in situations where the deadline is loose, and restricting the agent to its Q-table when the deadline is more tight. Effectively we hope to instruct the agent to learn while it has time, and follow the optimal course when time is restricted. We observe an increase in the success rate and a decrease in the penalty rate.

One thing that hinders this agent learning faster is the large state space of traffic configurations. If the state space were to be reduced somehow to safe and unsafe configurations, then we could explicitly program the agent to follow the waypoint unless it is 'unsafe'. i.e. in state space {light = red, waypoint = left, oncoming = forward, left = right, right = None} would map to {waypoint = left, safety = false}. But if waypoint = right, then we would want safety state to be True, so there are some hindrances to a simple state space reduction. We could map all five inputs to a single safe/unsafe mapping. However that would defeat the purpose of the Q-learning exercise, and would not be scalable to accepting more inputs. Finally due to the sparsity of dummy agents, the agent learns well what to do in no traffic situations, but it would take many more than 100 runs for it to even cover all the traffic configurations, let alone determine the optimal action in those situations.

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?

The optimal policy for this problem could be summarized as “If safe, follow the waypoint.” The agent has learned what is safe through penalties for traffic violations, and the waypoint through rewards for the optimal move. Using the evaluation metric described above we feel comfortable stating the agent

approximates the optimal policy. Given an extra long training period (runs = 1000) we observe the evaluation metrics improve.

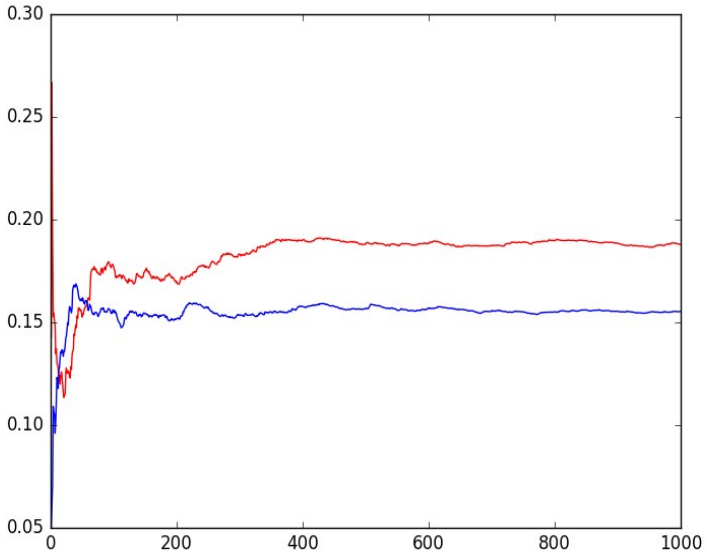

Illustration 1: cumulative penalty rate for static (red) and dynamic (blue) epsilon over time

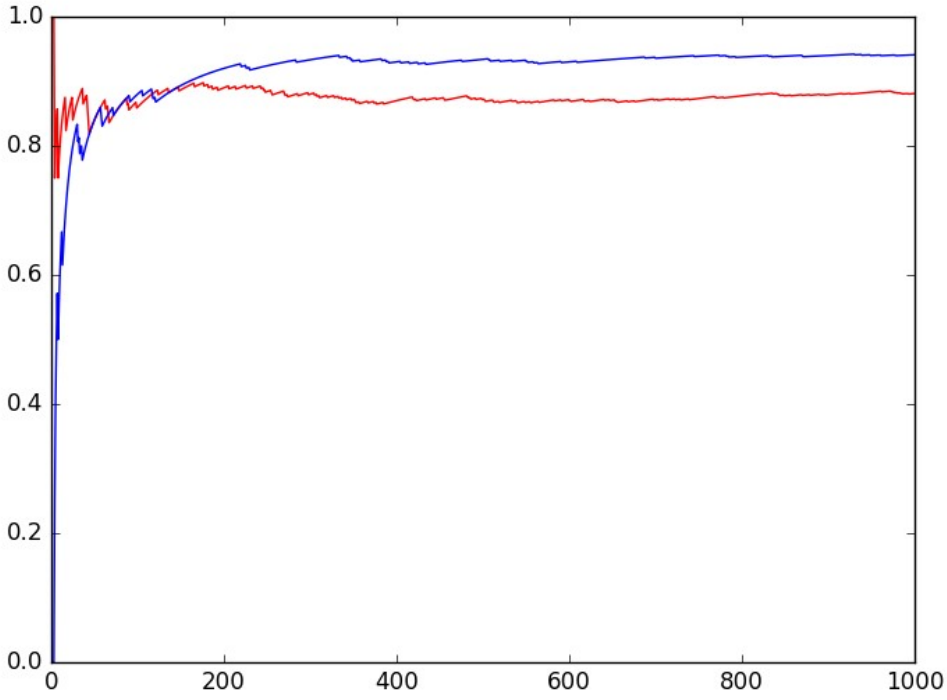

Illustration 2: Cumulative Success Rate for static (red) and dynamic (blue) epsilon

We can confirm our best form of the agent improves to learn the optimal policy over even a 100 trial simulation. A sample of 10 early trials has a 60% success rate – and the times taken to achieve the destination are long:

Simulator.run(): Trial 0
Environment.reset(): Trial set up with start = (7, 1), destination = (5, 3), deadline = 20
Environment.step(): Primary agent ran out of time! Trial aborted.
Simulator.run(): Trial 1
Environment.reset(): Trial set up with start = (8, 5), destination = (3, 5), deadline = 25
Environment.act(): Primary agent has reached destination! t = 16
Simulator.run(): Trial 2
Environment.reset(): Trial set up with start = (4, 4), destination = (7, 1), deadline = 30
Environment.step(): Primary agent ran out of time! Trial aborted.
Simulator.run(): Trial 3
Environment.reset(): Trial set up with start = (5, 3), destination = (8, 5), deadline = 25
Environment.act(): Primary agent has reached destination! t = 12
Simulator.run(): Trial 4
Environment.reset(): Trial set up with start = (2, 3), destination = (5, 1), deadline = 25
Environment.act(): Primary agent has reached destination! t = 11
Simulator.run(): Trial 5
Environment.reset(): Trial set up with start = (6, 2), destination = (2, 1), deadline = 25
Environment.act(): Primary agent has reached destination! t = 10
Simulator.run(): Trial 6
Environment.reset(): Trial set up with start = (4, 5), destination = (1, 1), deadline = 35
Environment.act(): Primary agent has reached destination! t = 5
Simulator.run(): Trial 7
Environment.reset(): Trial set up with start = (8, 5), destination = (4, 6), deadline = 25
Environment.step(): Primary agent ran out of time! Trial aborted.
Simulator.run(): Trial 8
Environment.reset(): Trial set up with start = (8, 4), destination = (1, 4), deadline = 35
Environment.act(): Primary agent has reached destination! t = 6
Simulator.run(): Trial 9
Environment.reset(): Trial set up with start = (6, 3), destination = (1, 6), deadline = 40
Environment.step(): Primary agent ran out of time! Trial aborted.

Meanwhile the last 10 trials are all successful and the time to finish are faster:

Simulator.run(): Trial 90
Environment.reset(): Trial set up with start = (3, 6), destination = (2, 2), deadline = 25
Environment.act(): Primary agent has reached destination! t = 16
Simulator.run(): Trial 91
Environment.reset(): Trial set up with start = (2, 5), destination = (7, 5), deadline = 25
Environment.act(): Primary agent has reached destination! t = 11
Simulator.run(): Trial 92
Environment.reset(): Trial set up with start = (1, 3), destination = (4, 4), deadline = 20
Environment.act(): Primary agent has reached destination! t = 17

Simulator.run(): Trial 93
Environment.reset(): Trial set up with start = (4, 6), destination = (8, 4), deadline = 30
Environment.act(): Primary agent has reached destination! t = 18
Simulator.run(): Trial 94
Environment.reset(): Trial set up with start = (2, 3), destination = (6, 4), deadline = 25
Environment.act(): Primary agent has reached destination! t = 11
Simulator.run(): Trial 95
Environment.reset(): Trial set up with start = (1, 5), destination = (7, 4), deadline = 35
Environment.act(): Primary agent has reached destination! t = 11
Simulator.run(): Trial 96
Environment.reset(): Trial set up with start = (3, 6), destination = (5, 3), deadline = 25
Environment.act(): Primary agent has reached destination! t = 9
Simulator.run(): Trial 97
Environment.reset(): Trial set up with start = (2, 6), destination = (3, 2), deadline = 25
Environment.act(): Primary agent has reached destination! t = 12
Simulator.run(): Trial 98
Environment.reset(): Trial set up with start = (6, 6), destination = (1, 5), deadline = 30
Environment.act(): Primary agent has reached destination! t = 6
Simulator.run(): Trial 99
Environment.reset(): Trial set up with start = (2, 6), destination = (6, 1), deadline = 45
Environment.act(): Primary agent has reached destination! T = 24

There is some further work that could be done – the alpha and gamma tuning was very coarse, and some finer adjustments could yield better performance. Similarly with epsilon – the decay is currently linear; both the shape and rate of decay could be tested as well. If this work proves insufficient these avenues will be explored first.