Machine Learning Engineer Nanodegree

Project 4 – Smartcab

Implement a Basic Driving Agent

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?*

Without modifying the provided code, a simulation shows that the agent (red car) does not move at all on the map. Even when the light changes to green, the agent simply stays in place. The other agents move around the map and respond to the traffic lights.

Informing the agent to take a random action from a list of choices (None, ‘forward’, ‘left’, ‘right’) makes it wander around aimlessly on the map. The rules built into the simulator keep the agent from performing an illegal action, but the negative reward is still recorded. If the destination is reached, a reward of 12 is recorded, the environment is reset, and the next trial begins.

The agent rarely makes it to the destination, but this is only after taking a very inefficient path. We can see from the results below that our agent reaches the destination an average of 18 times out of 100 trials.

Trial No.	**Successes**	**Failures**
1	21	79
2	21	79
3	15	85
4	22	78
5	21	79
6	11	89
7	14	86
8	21	79
9	21	79
10	17	83
Average	**18**	**82**

Table 1 - Results of Taking Random Actions

Inform the Driving Agent

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?*

Within the environment, there are several state variables that the agent can sense. The agent can follow a heading towards the destination of either [‘forward’, ‘left’, or ‘right’]. The traffic light the agent approaches at each step can be either [‘red’ or ‘green’]. Any other agent that is coming to the intersection from the other {‘oncoming’, ‘left’, or ‘right’} directions show their actions as being [‘none’, ‘forward’, ‘left’ or ‘right’] through the intersection. The last element of the environment is the deadline count for reaching the destination.

This means that there are [3] * [2] * {[4] * [4] * [4]} * [1] = 384 total states.

This number of states does not seem reasonable. The goal for our agent is to use Q-Learning to make informed decisions about each state. That means that our agent would need to encounter each state and update each state-action pair enough times until they all converge. Our limited trials with a limited number of moves in each trial are not enough to let a policy converge while using so many states.

For our learning agent, four main variables are monitored: [‘red’ or ‘green’] status of the light, [‘none’, ‘forward’, ‘left’ or ‘right’] actions of traffic from the {‘oncoming’ or ‘left’}, and the next waypoint heading of [‘forward’, ‘left’, or ‘right’].

Using these reduced variables, our model considers [2] * {[4] * [4]} * [3] = 96 total states.

The traffic approaching our agent from the ‘right’ is not monitored. It is assumed that all other agents on the map will always obey the traffic rules, and so any traffic approaching from the ‘right’ will not run a ‘red’ light. A ‘green’ light for our agent will be a ‘red’ light for an agent approaching from the ‘right’. A ‘red’ light for our agent will only allow our agent to turn ‘right’, again having unimpeded motion by an agent approaching from the ‘right’. Therefore, traffic approaching from the ‘right’ plays no part in the decision our agent will make.

The deadline count for reaching the destination is also not included in our list of variables. The restrictions of the simulator keep our agent from running ‘red’ lights, and so there isn’t anything that our agent could do differently than follow its optimal policy. If the simulator were modified to let our agent run a ‘red’ light (at the cost of a steep penalty), then there could possibly be a scenario where knowing the deadline countdown would be a necessary piece of information.

By eliminating some variables from our model, Q-Learning will converge much faster to an optimal policy, as fewer training samples will be needed.

Implement a Q-Learning Driving Agent

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?*

Q-Learning is an algorithm for Temporal Difference learning. It learns the optimal policy, even if actions are sometimes exploratory or random. This algorithm is an iterative one.

First, the Q-values for all state-action pairs are assigned arbitrarily. Then, given a state, an action is chosen either at random, or based upon some knowledge stored in the Q-values. For this project, we have assigned the Q_table to be part of an empty dictionary, so all values begin at None- neither positive nor negative. Because of this, Q-Learning will have no choice but to chose a random action for the early parts of test episodes.

Once an episode moves through a few iterations, the Q_table will begin to have more and more values filled in for the various state-action pairs. The algorithm will consider a factor **ε**, which will incorporate a chance that the agent will pick a random action rather than look at the Q_table. This factor helps the algorithm balance out exploration vs. exploitation, so that it doesn't converge to a local maximum. Adding the element of exploration helps to ensure that a global maximum will be found for the policy.

The **Q**-value for being in a state **s** and taking action **a** (**Q(s, a)**) is then calculated using the formula

$$Q(s, a) \leftarrow Q(s, a) + \alpha * [r + \gamma * \mathrm{Max}_{\alpha} Q(s', a') - Q(s, a)]$$

- The learning rate **α** is set between 0 and 1. Setting it to 0 keeps the algorithm from updating the Q-values. Setting **α** to a higher value near to 1 forces the algorithm to learn very quickly.
- The immediate reward for taking an action and arriving in a state is represented by "**r**". This value is given to the algorithm by the environment.
- The discount factor **γ** takes a value from 0 to 1. This factor represents the trade-off between immediate and future rewards. Choosing a discount factor closer to 0 will make the algorithm take immediate rewards into greater consideration. Setting it closer to 1 will make future rewards worth more in deciding what actions to take.
- **"Max_{α} Q(s', a')"** is the maximum reward that is attainable in the next state.

Next we would normally set the next state as the current state and repeat. However, the other agents are constantly moving around the map and going to seemingly random intersections all the time. So we cannot possibly know for sure what the next state will be until we get there. To make Q-Learning work, in this model we set the current state to be the previous state. Iterating through in this way will allow the algorithm to do its job just the same.

To get our agent running with Q-Learning implemented, I initially set the learning rate **α**, discount factor **γ**, and **ε** to all be = 0.5. There is no reasoning behind this other than it is a number right in the middle of 0 and 1, which are the limits for the factors. The next section of this project optimizes these parameters.

Using Q-Learning, our agent makes it to the destination much more frequently than when it simply took random actions. We can see from the results below that our agent reaches the destination an average of 59 times out of 100 trials. The increase in performance is due to the fact that our agent is now taking better actions by using information that it has learned.

Trial No.	**Successes**	**Failures**
1	60	40
2	60	40
3	62	38
4	61	39
5	64	36
6	50	50
7	58	42
8	55	45
9	58	42
10	61	39
Average	**59**	**41**

Table 2 – Results Incorporating Q-Learning

Resources:

Eden, Tim, Anthony Knittel, and Raphael van Uffelen. "Reinforcement Learning - Algorithms". *Cse.unsw.edu.au*. N.p., 2016. Web. 10 Oct. 2016. (http://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html)

McCullock, John. "A Painless Q-Learning Tutorial". *Mnemstudio.org*. N.p., 2016. Web. 28 Oct. 2016. (http://mnemstudio.org/path-finding-q-learning-tutorial.htm)

Improve the Q-Learning Driving Agent

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?*

Below is a table showing the results of running the Q-Learning agent using the shown parameters. The results are the average of running the test five times with each set of parameters.

γ	**ε**	**α**	**Success %**	**Moves**	**Penalty**
0	0	0	25	36	-14
0.25	0.25	0.25	43	25	-8
0.5	0.5	0.5	59	22	-6
0.75	0.75	0.75	76	21	-4
1	1	1	97	12	0
0.75	1	1	97	11	0
0.5	1	1	93	15	-1
0.25	1	1	98	18	0
0	**1**	**1**	**99**	**11**	**0**
0	0.75	1	88	16	-2
0	0.5	1	72	23	-5
0	0.25	1	39	24	-9
0	0	1	21	23	-10
1	0.75	1	37	22	-5
1	0.5	1	28	24	-8
1	0.25	1	22	18	-7
1	0	1	17	33	-14
1	1	0.75	77	19	-2
1	1	0.5	77	14	-1
1	1	0.25	84	21	-1
1	1	0	20	26	-11
1	0	0	20	30	-11
0	1	0	19	25	-11
0	0.98	0.98	99	12	0
0	0.95	0.95	97	13	0
0	0.92	0.92	97	10	0
0	0.9	0.9	96	14	-1

Table 3 - Tuning Parameters - Avg. of 5 Tests

- "Success %" is simply the number of times out of 100 trials that the learning agent successfully made it to the destination before time ran out. "Moves"
- "Moves" is the number of moves the agent took on the 100th trial to reach the destination.
- "Penalty" is the tally of the penalties incurred by the agent on the 100th trial.

The best combination of parameters is highlighted in green, where **γ = 0, ε = 1,** and **α = 1**. This set of values means that our agent learns very quickly, as α is set to its max value. Since ε is set to its maximum value, our agent will not take a random action if it is in a state that it has previously encountered and has a stored value Q-value. It will act purely off of experience rather than decide to explore. The minimum value for **γ** forces our agent to consider only immediate rewards rather than future ones.

Using these values, the agent made it to the destination successfully before time ran out 99% of the time. The agent took an average of 11 moves on the last trial to make it to the destination. It also incurred zero penalties during the last trial. The combinations highlighted in yellow are also worth noting.

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?*

Our agent does get close to finding an optimal policy by the time it is done running 100 trials. The combination of parameters explained above leads the agent to reach the destination in an average of 11 moves on the 100th trial. This is very close to the overall minimum value for moves shown on Table 3. Our agent also consistently reaches the destination without incurring any penalties.

Our optimal combination has a 99% success rate rather than the 97% shown by the combination γ = 0, ε = 0.92, and α = 0.92. This combination reaches the destination in an average of 10 moves, but the lower success rate disqualifies this combination from being the optimal.

The optimal policy would be one that above all else has the highest success rate. We want our passengers to get to their destination, even if the cab has to take an extra step or two to get there. We also want to minimize the number of penalties incurred by our agent, because we want our passengers to get to their destinations safely and without violating any traffic laws. Finally, to make our passengers happy, we want to reach the destination in as few safe moves as possible. Our optimal combination satisfies these conditions, as is supported by the data in Table 3.