Machine Learning Nanodegree - Project 04

Pedro M Duarte
pmd323@gmail.com
October 23, 2016

I. Implement a Basic Driving Agent

Q: *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?*

A: To answer this question I implemented the `RandomAgent` class. In addition to selecting a random action, this class saves a counter for each transition observed in the system. A transition is defined in this case as the tuple given by

- next waypoint
- intersection state (state of traffic light and oncoming traffic)
- action selected
- reward

With a random action choices the smartcab starts random walking over the grid and eventually makes it to the destination.

It was interesting to keep track of the unique transitions that the agent performed. I ran the simulator 10 times with a setting of `num_dummies=20`. The large number of dummy agents created more complex traffic conditions and increased the possibility of having the smartcab encounter other traffic at intersections.

Below are some notable observations I made after analyzing the simulation results:

- If the action selected violates traffic rules, a reward of -1.0 is given to the agent.
- If the action selected respects traffic rules but does not match the next waypoint, a reward of -0.5 is given to the agent.
- If the action selected respects traffic rules and matches the next waypoint, a reward of +2.0 is given to the agent.
- Whenever the selected action violates traffic rules, the simulator rejects the action and the agent does not move in the grid.
- Even with 20 dummy agents, the most commonly encountered intersection state is that one where the smart cab is by itself at the intersection (no other traffic).
- The situation where there are two dummies at the intersection was encountered a total of 25 times my in simulation, compared with 192 times where the smartcab was with only one dummy at the intersection, and 481 times the smartcab was by itself at the intersection. (Keep in mind that these numbers are for `num_dummies=20`)

II. Inform the Driving Agent

Q: *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?*

A: Based on the study of the simulations performed with the `RandomAgent` I was able to identify the most common states encountered by the agent. The number of states to consider for the Q learning algorithm is important. If we pick too many states we will require a lot of observations to be able to learn. On the other hand, if we reduce the situation to only a handful of states, then the agent might not be able to have enough information to distinguish between some kinds of actions, and this will result in the smartcab taking some actions that lead to negative rewards.

For my implementation of the Q learning algorithm I will use the following variables to define the possible states:

- is light green (boolean)
- waypoint (4 possibilities)
- car oncoming and turning (boolean)
- car oncoming (boolean)
- car left (boolean)
- car right (boolean)

Considering the cardinality of the state variables outlined above, we have a total of $2^5 * 4 = 128$ possible states to consider. Notice that the main piece of information that I decided to leave out was related to the turning direction of other traffic at the intersection. For cars coming from the left and right I completely ignored the turning direction. For oncoming cars I only differentiated between turning and not turning, and did not include the specific turning direction in the state.

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?*

A: If we consider all possible variables, we would have a state representation as follows:

- light state (red or green)
- waypoint (4 possibilities)
- car on left (4 possibilities: None, turning left, turning right, going forward)

- car on right (4 possibilities)
- car oncoming (4 possibilities)

That makes up a total of 512 possible states. Some of the states covered by these possibilities are much less likely to occur than others. Q-learning will take longer to converge if there are states that are not really seen very often in practice. A better approach is to consider a subset of only the most significant states to allow for faster convergence of the Q-learning algorithm.

III. Implement a Q-Learning Driving Agent

The Q-Learning driving agent was implemented with the following properties:

A. Q-matrix initialization

All of the entries in the Q-matrix were initialized to a random value between 0 and 1.

B. Learning rate decay

The learning rate was given the dependency

$$\alpha_t = \alpha_0 \Theta / (\Theta + t) \tag{1}$$

where the variable Θ represents an offset parameter that can be used to control the decay of the learning rate. This dependency satisfies the criteria for convergence of the Q-matrix, but is designed to decay slowly so that we continue to have a sizable learning rate even after several hundred transitions have been observed by the smartcab.

C. Exploration versus exploitation

To simplify the parameter space for Q-learning, the `epsilon` exploration probability is set to be equal to the learning rate, except scaled down by a parameter which we called S:

$$\epsilon_t = \alpha_t / S \tag{2}$$

D. Summary of Q-learning parameters

Below is a summary of the Q-learning parameters along with the values that were picked for the first implementation of the Q-learning algorithm:

- Q matrix random initialization interval: [0, 1]
- Learning rate starting value ($\alpha_0 = 0.1$)
- Learning rate offset parameter ($\Theta = 10^4$)
- Epsilon scale down factor ($S = 10$)

Optional:*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?*

A: The main difference after implementing Q-learning is that the agent now reaches the destination for the vast majority of the trials!! On the first few trials the rate of success is low, as the smartcab is learning and the Q matrix is only starting to converge. But after only a handful of trials the smartcab has seen enough transitions and has learned how to get to the destination while obeying traffic rules.

Top 5:

alpha_0	theta	S	success_rate	penalty_rate	objective
1.0	100.0	100	1.00	0.024283	0.951435
0.5	100.0	100	0.99	0.021792	0.946416
1.0	10000.0	100	1.00	0.027821	0.944359
1.0	1000000.0	100	1.00	0.028488	0.943024
0.5	1000000.0	100	0.99	0.026886	0.936229

Worse 5:

alpha_0	theta	S	success_rate	penalty_rate	objective
1.0	1000000.0	1	0.220000	0.559863	-0.899726
2.0	1000000.0	1	0.210000	0.558591	-0.907182
1.0	10000.0	1	0.190000	0.560738	-0.931475
2.0	10000.0	1	0.222222	0.582402	-0.942581
2.0	100.0	1	0.160000	0.572389	-0.984779

TABLE I
Realization number 1.

IV. 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?*

In order to improve the Q-Learning Driving Agent I will implement a grid search over the Q-learning parameter space. To make the search a little simpler, I will keep the Q matrix random initialization interval constant. After all, the range of the initial values only sets an overall scale factor that can also be affected via the learning rate starting value.

The remaining parameters will be varied as follows:

- α_0: [0.05, 0.1, 0.5, 1, 2]
- Θ: [10^2, 10^4, 10^6]
- S: [1, 10, 100]

The above parameter space gives us a total of $5*3*3 = 45$ parameter combinations that can be tried out to improve the smartcab performance.

The objective function that will be used to evaluate the smartcab performance will be a combination of the success rate and the total number of negative rewards (penalties) received over the 100 simulation trials. Giving a factor of 2 more importance to the penalty rate, I defined the following objective function:

$$\mathrm{objective} = \mathrm{success\,rate} - 2 \times \mathrm{penalty\,rate}) \tag{3}$$

In tables I to IV, we show the top 5 and worse 5 parameter combinations for four runs of the simulation. The following conclusions can be made:

- Having a large desire for exploration (low value of S) hurts the success rate for all values of the other parameters. In fact, all of the 5 worse results have $S = 1$, and all of the top 5 results have $S = 10$.
- The value of Θ is not very important as long as the value of α_0 is 0.5 or 1.0.
- It seems like the sweet spot for the value of α_0 is between 0.5 and 1.0. This makes sense if we keep in mind our random initialization of the Q-matrix, with random values

Top 5:

alpha_0	theta	S	success_rate	penalty_rate	objective
0.5	100.0	100	0.99	0.025868	0.938263
1.0	1000000.0	100	1.00	0.031155	0.937690
0.5	10000.0	100	0.98	0.025758	0.928485
1.0	10000.0	100	0.98	0.028256	0.923487
0.5	1000000.0	100	0.98	0.035186	0.909627

Worse 5:

alpha_0	theta	S	success_rate	penalty_rate	objective
1.0	1000000.0	1	0.270000	0.578393	-0.886786
1.0	10000.0	1	0.250000	0.569881	-0.889762
2.0	100.0	1	0.230000	0.582621	-0.935241
2.0	10000.0	1	0.171717	0.569337	-0.966958
2.0	1000000.0	1	0.160000	0.579194	-0.998387

TABLE II
REALIZATION NUMBER 2.

Top 5:

alpha_0	theta	S	success_rate	penalty_rate	objective
1.0	10000.0	100	1.00	0.028932	0.942136
0.5	10000.0	100	0.99	0.026377	0.937246
0.5	1000000.0	100	0.99	0.032951	0.924097
0.5	100.0	100	0.97	0.024353	0.921294
1.0	100.0	100	0.99	0.034647	0.920707

Worse 5:

alpha_0	theta	S	success_rate	penalty_rate	objective
1.0	1000000.0	1	0.242424	0.563698	-0.884971
2.0	100.0	1	0.230000	0.564867	-0.899734
2.0	1000000.0	1	0.210000	0.573132	-0.936264
2.0	10000.0	1	0.190000	0.572574	-0.955149
1.0	10000.0	1	0.210000	0.586822	-0.963644

TABLE III
REALIZATION NUMBER 3.

in the range [0, 1]. The random initialization range will be tightly coupled with the scale of the rewards and the value of α_0.

- For our initial Q-learning parameters, $\alpha_0 = 0.1$, $\Theta = 10^4$ and $S = 10$, the success rate is 98% on average, and the penalty rate is approximately 5% on the average. After the grid search we see that by optimal choice of the Q-learning parameters this can be boosted up to nearly 100% success rate and approximately 3% penalty rate.

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?*

A: From the optimization of the Q-learning parameters shown above I do believe that the smartcab is doing pretty well. In some cases we even see 100% success rate, which means that even on the first simulation trial the smartcab is already learning enough to make it to the destination within the allotted time.

An optimal policy for this problem involves both the success rate and the penalty rate. As was mentioned above, I gave twice the importance to the penalty rate than I did the success rate, when considering the objective function for optimization of the Q-learning parameters. In a world where it is very costly to commit a traffic violation, the weight given to the penalty rate could be set to an even higher value.

I do believe that the smartcab could do better in terms of penalty rate, at the expense of missing some trip deadlines. The best result shown so far is 3% penalty rate. I am a driver and I would not want to commit a traffic violation at that rate. I think a more reasonable penalty rate that would make the smartcab comparable to a human driver would be around one traffic violation for every 1000 intersections visited, or a 0.1% penalty rate.

Top 5:

alpha_0	theta	S	success_rate	penalty_rate	objective
0.5	10000.0	100	1.00	0.030158	0.939685
1.0	1000000.0	100	1.00	0.031939	0.936122
1.0	100.0	100	0.99	0.027437	0.935126
0.5	1000000.0	100	0.99	0.028593	0.932814
1.0	10000.0	100	0.99	0.031019	0.927962

Worse 5:

alpha_0	theta	S	success_rate	penalty_rate	objective
1.0	1000000.0	1	0.190000	0.570755	-0.951509
2.0	1000000.0	1	0.210000	0.586322	-0.962643
2.0	10000.0	1	0.170000	0.567735	-0.965470
1.0	10000.0	1	0.170000	0.578001	-0.986001
2.0	100.0	1	0.141414	0.585741	-1.030068

TABLE IV
REALIZATION NUMBER 4.