Project Report

Udacity Machine Learning Nanodegree
Thalles Santos Silva

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

Running the smartcab simulator with a policy that just picks one of the four possible positions (None, left, right, forward) at random and returns it as an action shows some interesting behavior from the agent. First off, we can see the agent moving around the grid world in a very erratic fashion. Also, we notice the rewards it gets when it performs some correct moves (2.0 points) at the intersections as well as the penalties it suffers when it violates some of the rules of world, in this case traffic rules (-1.0 points) and when it make a valid move (does not violate any traffic rule) but it turned out be the wrong one (it turned right when it should have gone forward) -0.5 points for this situation. However, even moving in a random way, it does, eventually, get to the destination, and when it happens, we can see a larger reward (10 points). In fact, for the hundred trial project setup, using this random policy, the agent accomplishes its goal with a roughly 20% accuracy.

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

Based on the problem setup, I believe the best way to choose the possible states that the agent can be is by looking at the set of inputs available and make some combinations that model both the agent and the environment with the most description but avoiding too much complexity. According to the problem specification, the agent has an egocentric view of the grid intersection it is at being able to see if there are other agents in front of itself, or in one of the both roadways side. The agent can also sense the state of the semaphore, thus, being able to identify it as green or red along with the next waypoint that it should take in order to reach the desired destination. Based on these data, we can map these input to possible states the agent can be in by combining these different input variables. For instance, suppose the agent is at some intersection where the semaphore is **green**, and there are no agents in any of the incoming roadways. This state is represented as:

light: green, left: None, oncoming: None, nextwaypoint: forward

As another example, suppose the agent is at an intersection and there is another agent in front of it that wants to turn right, and in this case the semaphore is **red**.

light: red, left: None, oncoming: right, nextwaypoint: forward

Note also that in both cases the agent should go forward to comply with the destination goal.

With this structure, we can model the necessary number of states that can differentiate the various situations that the agent can be based on the variety of input possibilities that exist.

Based on the current set of inputs available, we chose the **semaphore light**, the indication that there is another agent coming from the oncoming and left roadways along with their respective desired actions, and finally the **next way point to represent** a vadid state in the smartcab world. These input variables were chosen because they can represent the different states in the game with a very clear distinction and without too much complexity. One of the variables left out was the **deadline** that basically represents the amount of remaining time available to the agent to reach the destination. From my research, including this variable as part of the state would make each of them different from one another in a single trial thus, increasing the total the number of possible states. That change would increase the complexity of the model that in turn, would have more trouble to learn the optimal policy as the trials go, in my point of view, it looks like that the model could suffer from variance – a too complex model for a not so much complex data. On the other hand, the semaphore indication makes a more general state because at each intersection the agent might face either a green or red light, so this kind of state makes the agent generalizes better over the world.

One thing that is worth noting is that there is no indication, in this state representation, that there could be an agent coming from the road to its right. That is because according to the traffic rules stated for this project, different from the other two possibilities (agents coming from the oncoming and left roadways) the fact that there is a car coming from the right or not does not affect the current agent at all. In other words, regardless of the semaphore color, the addition of an agent coming from the right does not create a new situation or traffic violation that is not already defined, and that also helps the model to keep a concise Q-Learning table which in turn, benefits the agent's performance.

Another worth mentioning detail is the fact that to keep the QTable as small and concise as possible, I decided to not hard code all of the possible states at the beginning, but incrementally adding the states to the QTable as it is needed.

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

After implementing a basic version of Q-Learning and programming the agent to use, instead of a random selected action, the best action from the Q-Learning table, some aspects regarding the behavior of the agent have changed. Firstly and most importantly, this scenario brings to the light the trade-off between exploration and exploitation. It is pretty clear that only choosing the best action from the QTable from the beginning, hurts the model in the sense that depending on what action it will choose to break ties, the result can be very different.

Because the QTable starts with 0s all over it at the beginning, for any action the agent asks the QTable for, the result will be the same because of the way the Q-Learning equation works. Since the agent has not seen most of the states at the beginning of the process, it did not learn anything about the environment and therefore, when the Q-Learning tries to pick the best option from the QTable, there is no best option yet, so one of the 4 possible actions (forward, right, left, None) must be chosen to break ties. In my experiments, when the None option is the one chosen in this situation, the agent simply does not move and consequently does not learn anything at all, since the None option has 0 reward.

However, when the forward option is chosen, the agent surprisingly acts quite well reaching the destination with an amazing success rate as described below.

I believe that is the core of the exploration exploitation dilemma, in order to use what it knows (exploitation) the agent must first learn the environment (exploration), and it is clear in the first case, when the action is None (for breaking ties) there is no exploration and consequently the agent is not able to learn anything about the environment. On the other hand, in the case of the 'forward' action for breaking ties, the agent gets the chance to learn the environment when there is no best option from the QTable yet available, which in turn makes the agent able to fill the table with optimum values that helps in future decisions.

Some very key aspects that are worth pointing here is how the agent behaves with this new policy and how it learns throughout the trials. It is clear that in the very first trials, the agent makes some erratic moves while he is learning from the environment, which is perfectly normal and expected. However, as the trials go, and the agent grows its knowledge about the world, the first thing that can be noticed is that it starts to follow the traffic rules very precisely first and then, it also starts to follow the intended path to the destination. This order in the knowledge gaining is explained by the fact that the agent get higher penalties when it violates a traffic rule (-1.0) than when it turns in the wrong path (-0.5), and because the QTable is not reinitialized after each trial, the knowledge about the traffic rules remain while the new destination path has to be relearned. Still, because the agent already knows the traffic rules, it is now much easier for it to learn the new destination path.

To be more specific about the results, an experiment was done using this basic Q-Learning implementation with no discount factor and a constant learning rate of 0.5. Over the 100 trials, the agent was able to accomplish the goal with an average of 99% accuracy. In this scenario, the forward action is chosen every time there is no best (maximum) value to pick.

4. **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? 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?**

In order to reach the final version of this agent's implementation, some more advanced techniques such as learning rate decay, discount factor, and a new policy was implemented using techniques to better deal with the exploration and exploitation dilemma.

First off, to calculate the Q-Learning function:

$$Q(s,a) = (1.0 - \alpha)Q(s,a) + \alpha[\Upsilon + \gamma \max_{a'} Q(s')]$$

Where (**s** is the state, **a** is the action, $\boldsymbol{\alpha}$ is the learning rate, $\boldsymbol{\Upsilon}$ is the immediate reward, and s' is the new state), as described in the class's videos, we introduced the learning rate decay parameter ($\boldsymbol{\alpha}$), that basically controls how important this value the agent is learning at step t is. However, in order to reach the maximum learning, the values of α, that decreases over time (t), must come from a function f_t that generates values between 0 and 1 in such a way that the following rules apply:

$$\sum_{t=1}^{\infty} f_t = \infty \ and \ \sum_{t=1}^{\infty} (f_t)^2 < \infty$$

And one of the equations that turn out to follow this reasoning is the equation: $f = \frac{1}{t}$ that is used as the learning rate decay equation in this implementation.

Another improvement is the discount factor variable. According to the Wikipedia, the discount factory (ɣ) determines the importance of future rewards. A factor of 0 will make the agent "myopic" (or short-sighted) by only considering current rewards, while a factor approaching 1 will make it strive for a long-term high reward. As you may see in the results below, various combinations of gamma and alpha were tried in order to find the most optimum parameter combination possible.

Finally, in order to access how good this agent is doing in finding the best policy in the time box defined to this project, a series of simulations were performed with a variety of parameter combinations for the learning rate, discount factor and epsilon. Each run followed the rules stated in the project – 100 trials for each parameter combination with the enforce deadline option setup to true, the Q-Learning table is reset only between simulations not between trials.

The following tables present the accuracy of the agent using various combinations of starting learning rate decay (first horizontal row) and discount factor values (first vertical column), for three different policy functions.

Due to the very good results using the 'forward' action to break ties, as described in the question 3, I decided to run this same policy with many combinations of learning rate decay and the addition of a discount factor. The policy algorithm along with the results of its implementation is shown below. Additional information such as the worse combination of parameters and the case where the agent got 100% accuracy are highlighted in red and green respectively.

```
• For each state s take the best action a from the QTable
• If action a is not unique:
    o Take the forward action
• Else:
    o Take the best action a
```

	0.5	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
0.5	100	99	100	100	99	99
0.6	99	99	99	100	100	99
0.7	99	100	99	99	99	99
0.8	99	97	93	99	99	100
0.9	100	98	93	98	74	99
1.0	100	99	100	99	99	100

Accuracy test results when the policy break ties by choosing only the forward action (values is percentages)

In order to make the best decisions at each time step, the random policy used by the agent was firstly changed to a policy that always picked up the best value from the QTable. However, as mentioned in the question 3, this approach showed some limitations mainly with the exploration/exploitation dilemma. Basically, in order to use what it has learned, the agent must first explore the environment, and that is the key point added to the new current policy. In this new approach, at each time-step, the agent will ask the QTable for the best action to take but it will take this action with probability (1-ε), where ε is a very small value that will control when the agent must simply take an action at random for the sake of exploring (learning) the environment. Also, this value of ε is set to decrease over time exponentially. This guarantees that at the beginning of the game, the agent will be allowed to explore/learn more from the environment and while its knowledge from the world grows, it will rely more and more in the values that it learned instead of taking random decisions. That added some exploration to the policy. Nevertheless, another problem stated in question 3 was the fact that when the agent asks for the best action, the QTable might actually have more than one optimum actions to deliver, and now we have to deal with the problem of breaking these ties in the best way possible. From my experiments, I found the best I could do at this point would be simply to add more exploration (take some random action), still, in some cases, we might do not know which is the best action to take but we may possibly know that some specific action(s) should not be taken at all. Based on the reasoning, when the QTable returns more than one best action, the agent will select randomly, only among these possible actions, the best one to take excluding the None action since this action does not add valuable information to the agent. The policy follows as:

```
• For each state s take the best action a from the QTable
• With probability (1 - ε)
    o If action a is not unique:
        ▪ At random choose one of the action candidates
          (excluding the None action)
    o Else:
        ▪ Take the best action a
• With probability ε
    o Take an action at random
```

The results for that policy implementation are showed in the table bellow.

	0.5	***0.6***	***0.7***	***0.8***	***0.9***	***1.0***
0.5	98	96	96	98	100	97
0.6	93	97	95	97	95	100
0.7	92	91	100	94	93	96
0.8	95	100	88	95	94	92
0.9	97	95	96	98	97	94
1.0	98	63	100	87	90	82

Accuracy test results applying an epsilon threshold for picking a random action *(values is percentages).*

As we can see by looking at the percentages, it is clear that this new policy does not accomplish, overall, the same great performance acquired by the last policy. From my experiments, I found that the reason for this downgrade in accuracy was the fact that when it is time to take a random action, which happens with probability **ε**, there is a 25% chance that this action will be the **None** action which by definition, does not add anything in terms of learning to the agent (0 reward), therefore, the solution would simply be to remove the possibility of taking this **None** action. Then I realized that this would be almost the same strategy done when there is more than one best action to take. As described above, when there is more than one best action to take, the policy will pick one of these available actions at random. The problem with just using this approach (taking out the decreasing **ε** factor), would be that once the agent identifies one route as the best one to take, it would always choose this one over the others, even though there might exist another action with a higher positive reward. However, if we look at this smartcab problem, the agent gets a positive reward only in one occasion i.e. when the agent makes a move that does not violate any traffic rule and it is the right one to choose. Consequently, we can conclude that, within a specific trial, once the agent learns an optimal action ***a*** for the state ***s***, this move (**a**) is guaranteed to be the best one throughout the trial. When a new trial begins, the information from the previous trial is maintained in the QTable, however, the new destination route has changed, as a result, the best action **a** for the state ***s*** in the previous trial might not be the best for this new trial. Still, in the beginning of this new trial, every time the agent gets to state **s**, it will get the best action **a** learned in the previous trial but now the agent might get a -0.5 penalty for taking the wrong path. As a result, it will eventually learn which direction is the best one to choose in this new trial context. Finally, we prove that under these circumstances, the **ε** random probability is not necessary, so the final policy can be summarized as:

```
• For each state s take the best action a from the QTable
• If action a is not unique:
    o At random choose one of the action candidates (excluding
      the None action)
• Else:
    o Take the best action a
```

	0.5	***0.6***	***0.7***	***0.8***	***0.9***	***1.0***
0.5	100	100	98	100	98	99
0.6	100	100	98	99	99	100
0.7	98	100	99	99	99	97
0.8	100	99	100	99	99	99
0.9	100	99	100	99	100	100
1.0	99	100	98	99	100	100

Accuracy test results for picking a random action, excluding None, when there are more than one best actions.

As we can see, the results look very good, from the 36 total simulations, in 16 of them the agent got 100% accuracy – 44%. Further in 30 of these simulation, the agent accomplished between 99 and 100 percent which is 83% total. Another interesting result is the 97% got with learning rate equal to 1.0 and discount factor of 0.7 which was the lowest accuracy percentage.