Université Gustave Eiffel

Reinforcement learning for car-following on a ring road

Mathematical Modeling for Intelligent Systems

Nadir Farhi

ELABORATED BY:

Arfaoui Fairouz

Ayachi Hosni

Marnissi Skander

M2 SIA

2020-2021

Introduction

Through this project, our main goal is to get a car to learn how to self drive on a ring road and that will be done by reinforcement using the Q-Learning algorithm on the SUMO traffic simulator and with the help of TraCi.
We will be having a certain number of cars following each other on a ring road with no possibility of overtaking.
We will simulate the dynamics of all the cars but one using SUMO while the last will be manipulated using Python and more specifically, Traci to utilize the Q-Learning algorithm.
We will be going through the whole process we've been through working on the project in this report.

Working Environment

SUMO

Vehicle traffic in a Flow experiment can be specified arbitrarily, either by an initial set of vehicles in the network or using SUMO vehicle inflows.

This network density can come from the initial vehicles on the road, for closed networks, or in the form of inflows for open networks in which vehicles enter and leave the network. Flow supports SUMO's built-in longitudinal and lateral controllers and includes a number of configurable car-following models. Arbitrary acceleration-based custom car-following models are supported as well.

The road used in our simulation is composed of 12 edges connected to each other in a loop, and the simulator controls all the cars within it except for one.

Traci

We used Traci as a communication tool with SUMO to be able to manipulate the one car we need to control using the environment variable SUMO_HOME on Python to be able to set the parameters we need and we used traci.simulationStep() to go one step further in the simulation.
⇒ The red car will be the one controlled through TraCi.
⇒ All the other cars are yellow.

Implementation: Agent & Environment

The vehicle.py file contains the Agent and the Environment for our simulation, implementing the Q-learning agent controlling the car and the environment all this action will take place. It defines the Q-table, with its discretization and possible actions, and the Q-learning algorithm, with its hyper parameters(epsilon,gamma,alpha) and set the reward system and fix the way the environment works.

The Q-table

Q-Table is a technique that utilizes a table Q where rows represent the potential states, and columns represent actions.

The one and only **Control Variable** in this simulation is the speed of the controlled car, which can be set by the function traci.vehicle.setSpeed(car_id, speed) .

State variables :

- The speed of the controlled car, discretized in [0, 150 Km/h].
- The relative speed (speed of the leader car of the controlled car – speed of the controlled car), discretized in [-30 km/h, 30 km/h]
- The space-headway (the position of the leader car of the controlled car – the position of the controlled car), to be discretized in [0, the length of the road] for example.

With 3 possible actions that are either deceleration, acceleration or remaining at the same speed.
Obviously, the real state variables are continuous, and only the Q-table is discretized, with linearly spaced bins for the state variables to be grouped by.
A framing function is used to find the right discretized bin in which to enclose the continuous state variable.

The Algorithm: Q-Learning Algorithm

Q-learning is an off policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed. More specifically, q-learning seeks to learn a policy that maximizes the total reward.

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
    $S \leftarrow S'$;
  until S is terminal

The algorithm advancement revolves around 3 basic steps:

- Agent starts in a state (s1) takes an action (a1) and receives a reward (r1)
- Agent selects action by referencing Q-table with highest value (max) **OR** by random (epsilon, ε)
- Update q-values

As we can see on the algorithm stated above, there are three hyper parameters to be set which are the following:

- **The discount factor γ:** It's used to balance immediate and future reward. From our update rule above you can see that we apply the discount to the future reward. Typically this value can range anywhere from 0.8 to 0.99.
- **The learning rate α:** can simply be defined as how much you accept the new value vs the old value. Above we are taking the difference between new and old and then multiplying that value by the learning rate. This value then gets added to our previous q-value which essentially moves it in the direction of our latest update.
- **The epsilon greedy probability ε** :is a simple method to balance exploration and exploitation by choosing between exploration and exploitation randomly.

We will be setting them respectively to 0.99, 0.1 and 0.1.

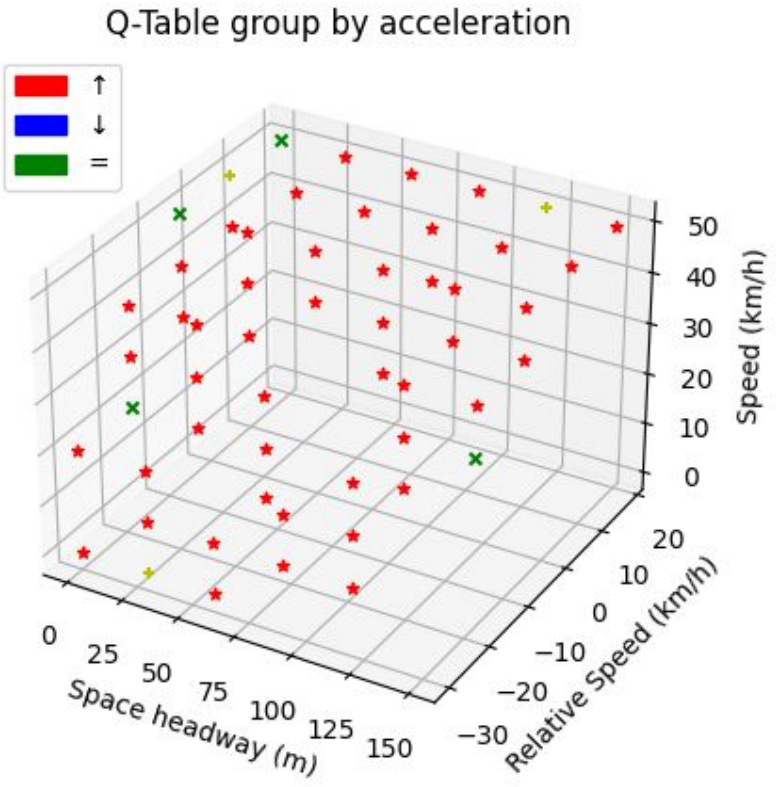
After the completion of the training, plots are created, saved in the project folder and displayed on the screen.

Output

In this section, we will be looking at the output of our simulation based on a reward system taking into account collisions, distance of security and speed limit. We set the speed limit to 55 km/h. The reward system works this way :

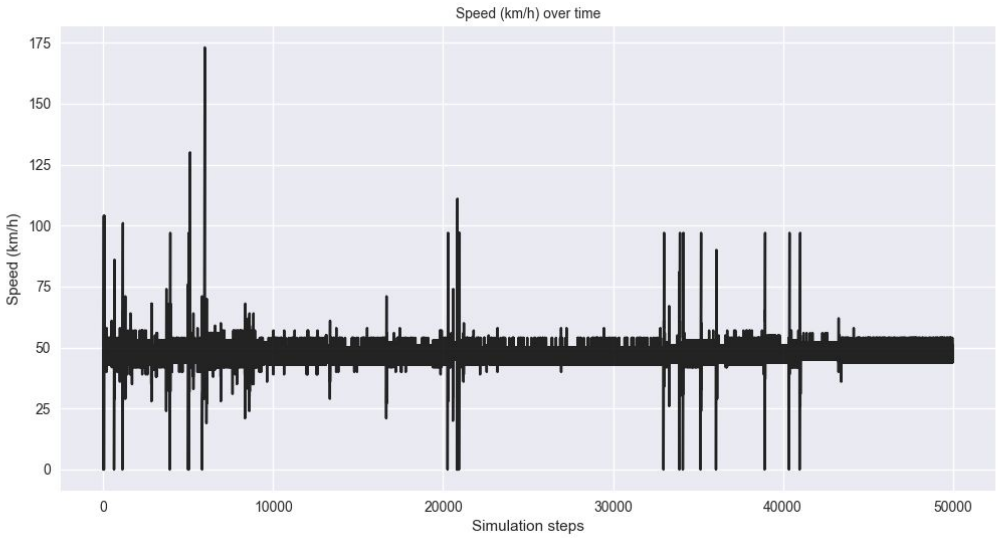
- 0 if there is a collision.
- Decreased by 90% if the speed limit is not respected.

These are graphs we get as an output:

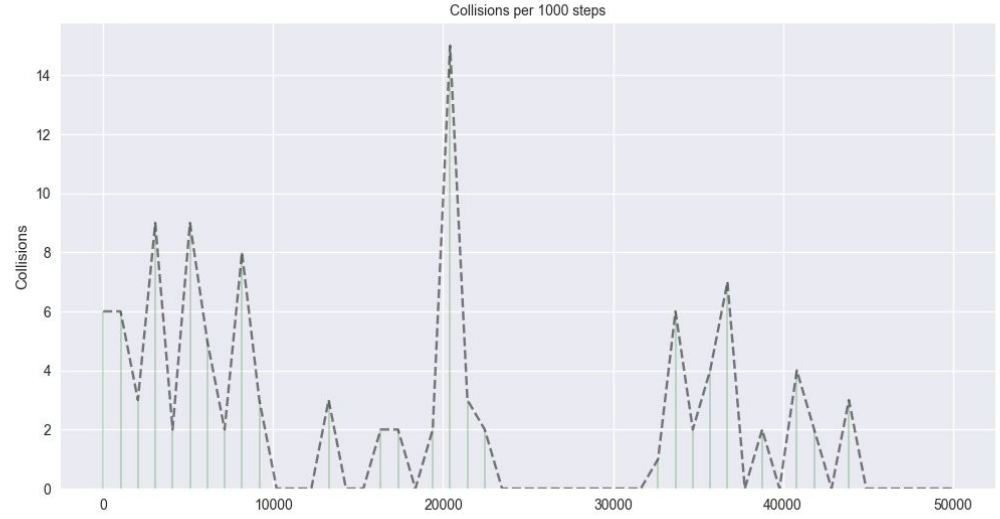


3D Plot of the Q-Table

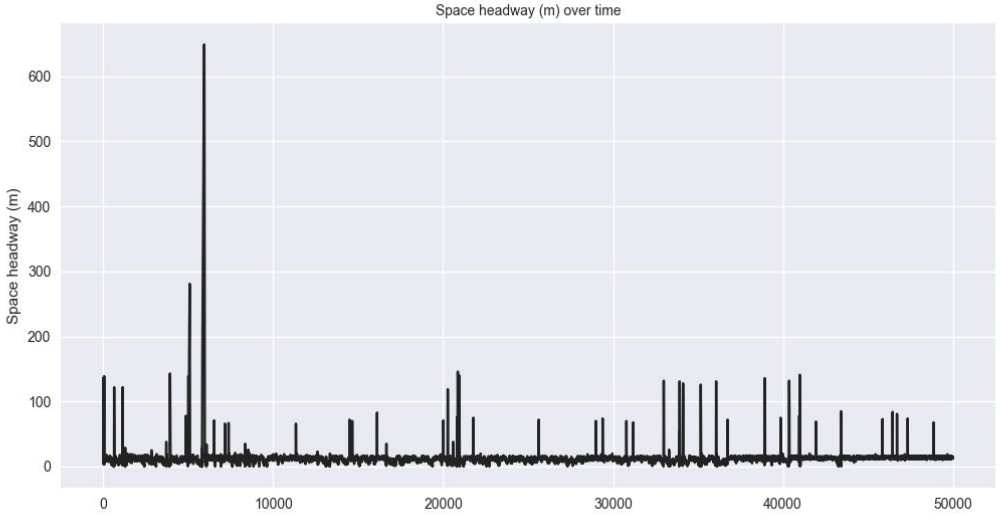
Speed over time



Collision over time



Space headway over time



Collision and state

Agent speed resume(km/h)

Steps	Mean	St deviation
1000	47.22	10.56
2000	47.39	6.82
3000	47.26	3.51
4000	47.31	6.98
5000	47.33	4.33
6000	47.8	18.79
7000	47.02	4.76
8000	47.06	3.22
9000	47.08	5.05
10000	47.42	2.63
15000	47.3	2.51
20000	47.46	2.91
25000	47.24	2.57
30000	47.41	2.63
35000	47.33	5.92
40000	47.35	2.68
45000	47.29	2.61
50000	47.28	2.62

Relative speed resume(km/h)

Steps	Mean	St deviation
1000	0.01	10.61
2000	0.13	6.64
3000	0.04	3.86
4000	0.07	7.32
5000	-0.07	4.68
6000	0.57	18.76
7000	-0.04	4.89
8000	-0.0	3.75
9000	-0.11	5.43
10000	0.1	3.14
15000	-0.02	3.16
20000	0.05	3.16
25000	-0.01	3.11
30000	0.02	3.15
35000	-0.02	6.14
40000	0.01	3.19
45000	-0.01	3.11
50000	-0.0	3.15

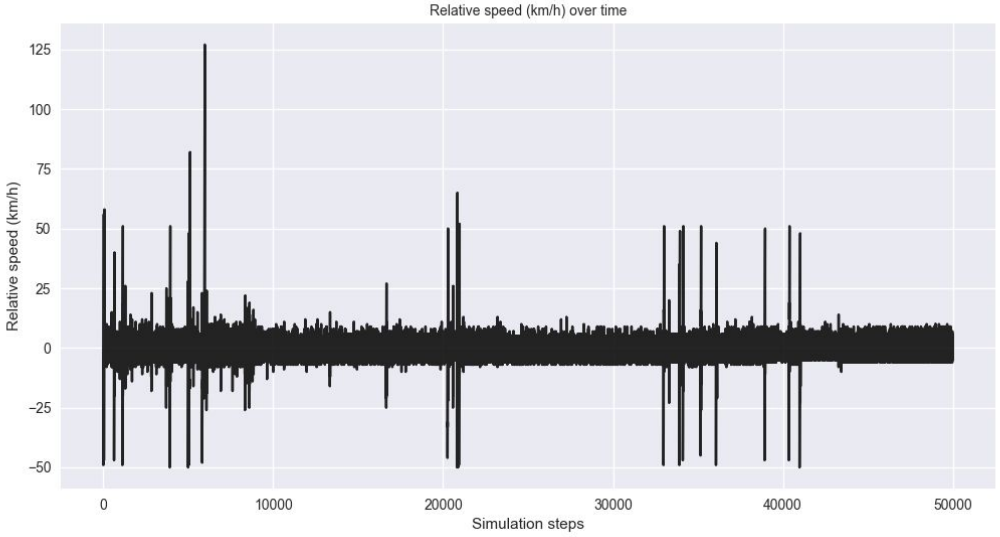
spaceHeadway resume(m)

Steps	Mean	St deviation
1000	17.26	25.36
2000	13.89	14.6
3000	10.63	3.59
4000	12.54	18.07
5000	10.94	5.51
6000	85.53	156.75
7000	13.06	4.16
8000	12.03	4.19
9000	12.74	3.78
10000	10.83	3.51
15000	10.89	4.13
20000	10.74	3.38
25000	11.17	2.77
30000	12.75	2.67
35000	12.56	16.41
40000	12.27	3.17
45000	13.2	1.73
50000	13.8	1.2

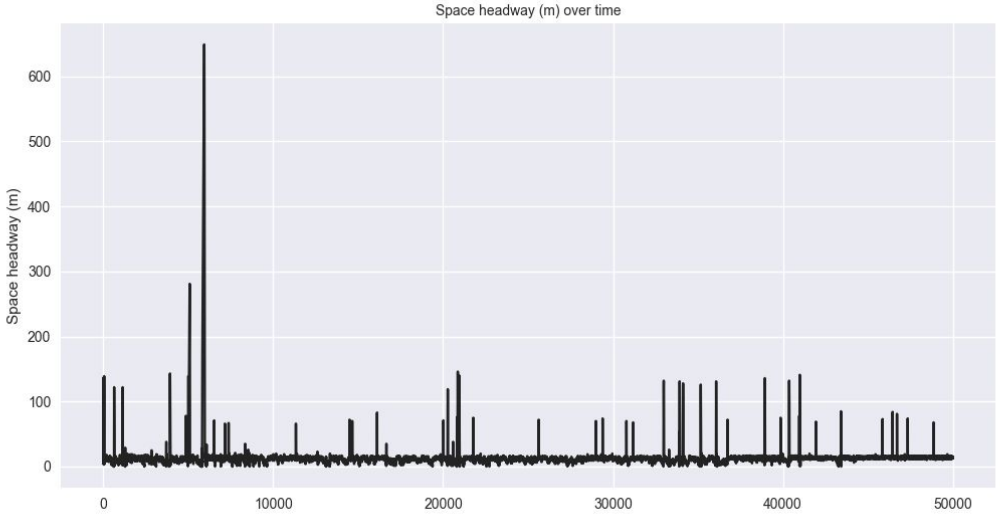
Collisions

Steps	Collisions	Total
1000	6	6
2000	6	12
3000	3	15
4000	9	24
5000	2	26
6000	9	35
7000	5	40
8000	2	42
9000	8	50
10000	3	53
15000	0	56
20000	2	62
25000	0	82
30000	0	82
35000	2	91
40000	0	104
45000	0	113
50000	0	113

Relative speed over time



Space headway over time



Conclusion

In this series of experiments, we have seen how important it is to define a good rewarding system to the training of the car. The more optimal it is, the easier it will be for the agent to learn to have the best behavior in its environment. Although it was hard at the beginning to begin to learn about both SUMO and TraCi from scratch to be able to set our environment, the agent and the parameters needed for our simulation, we finally managed with some help to be able to successfully run the simulations. We didn't know at first how to make collisions possible, since using SUMO was not feasible, therefore, we had to understand how to take full control of an agent, hence here a car through TraCi. Tuning the parameters for the simulations to go well wasn't that easy either. The fact that we were working on vanilla Q-learning didn't make it any easier but it was a difficult but good learning process we have gone through.