University of Bucharest

Faculty of Mathematics and Informatics

Data Science Specialization

Dissertation

Traffic Flow Optimization with Reinforcement Learning

Graduate

Ioan Dragoș Marian Giumanca

Coordinator

Ciprian Păduraru

Bucharest, 2021

ABSTRACT

The purpose of my Master's Degree Dissertation is to research Reinforcement Learning methods for traffic flow optimization in car junction. There are many articles and scientific papers that discuss this matter with proposed solutions to the traffic congestion. My objective was to implement multiple solutions from the mentioned articles together with some of my own and compare them in the scope of finding the simplest configuration that can give good results in the simulated junction scenario. For the simulation of the junction and the traffic, SUMO was used and the Reinforcement Learning agent was based on an implementation of a Deep Q-Network. Multiple configurations for the observation, reward, neural network architecture were implemented. The model was trained to control the traffic light phases on a two road junction with three lanes per road and the results were compared to a default program of the traffic light. The agent achieved good performance in light and medium traffic scenarios but struggled to get close to the traditional system in the case of heavy traffic.

I. INTRODUCTION

1. CONTEXT

In the recent years, **Neural Networks** are the golden standard for many Artificial Intelligence products. Due to the high research and computational capabilities that modern Graphical Processing Units provides, more problems are being solved using the power of Neural Networks.

A great use of **Deep Neural Networks** comes in the context of **Reinforcement Learning.** Those are used to give more learning power to the model when the complexity of the task is getting harder for classical methods to find an exact solution in a reasonable amount of time.

The problem of solving the **congestion of vehicles** in traffic is in the attention of multiple organizations that are trying to tackle it from multiple sides, from improving the infrastructure, to administrative restrictions and to intelligent software. One such approach is to use **Deep Reinforcement Learning** to optimize the traffic flow of cars by observing patterns and adapting to new situations.

2. PROBLEM

Big cities suffer from the bad operation of actual **traffic control systems** that fail to adjust to dynamic scenarios. A classical system of changing the Red-Yellow-Green lights may work in certain situations but can not **reduce the waiting time** for low traffic or adjust to heavy flow of cars. Even more, there are scenarios where road directions should have higher priority due to special vehicles, like police cars in mission. Some intelligent systems are using sensors to count the number of the cars of each lane and act accordingly, but even though this is better than a classical traffic light programme, it can not interpret all the possibilities of how cars are moving in a city.

Reinforcement Learning can solve the static problem of the traffic light programme and can act in certain ways that even a human could not have thought. In ideal situations, the model should continuously learn from new scenarios and improve itself.

A problem that arises when developing Reinforcement Learning solutions is the **large amount of training** and testing is required. When the model has to learn to do a complex

task and navigate through multiple possibilities, it may take days or even weeks to wait for a result. Another problem is the **high number of configurations** and how the model is programmed to interpret the task and how to act. All those taken into account, the process of developing practical solutions that can overtake the traditional system can be very time consuming.

3. OBJECTIVE

The objective of this project is to explore and compare multiple implementations of Reinforcement Learning solutions for the traffic flow problem. The scenario in focus will be a classic 2 road junction with 4 green-light phase. More of the actual setup will be discuss in the next chapters.

Multiple experiments were conducted of different configurations of the Artificial Intelligence model in the way of how informations is received from the environment. The scope is to find simple solutions that can have good performance. This helps better understand the complexity of the task and how should developers expect results from other experiments. This thesis does not intend to solve the traffic problem on the large and practical scale, but to give valuable insights for this task. First, a restricted scenario needs to be solved with reasonable resources before moving on with next more complicated steps.

At the end of my thesis I will explain what my personal thoughts are on the matter and how I interpret the useful insights from the experiments for future development. As a far future objective, anyone that is doing research in the area of urban optimizations with Reinforcement Learning models, should aim at the solution that can solve the traffic problem with better performance than any other standard solution.

4. MOTIVATION

The world as we know it is getting more complex every day. The problems arising from the fast development of society are getting harder to solve by traditional computer algorithms. More people are moving into cities and the urban area is evolving horizontally and vertically. With the high growth in size, cities are getting more crowded with a population that constantly needs urban transportation.

More cars on the road and with an old road infrastructure, cities get overwhelmed. This is also the case of Bucharest, the capital of Romania. According to the INRIX Global Traffic Scoreboard[1] , Bucharest ranks in the top spot with the biggest number of hours lost in traffic for the average driver.

A big factor for this problem, apart from the old road infrastructure, is the static traffic light systems that do not adjust for the heavy traffic at rush hours. By being able to actively change the durations of green phases and to prioritise road directions according to the current situation, the waiting time in a junction can be optimized.

The research and implementation of such intelligent systems can solve the congestion on small areas and further work can lead to distributed systems that can control the traffic flow in the whole city.

5. CONTRIBUTION

In this project, I show how different configurations of the Reinforcement Learning model affect performance. The results are compared to the default programme of the traffic light to see if and in what scenarios the AI performs better.

Apart from the academic papers analysed for this project, the setup of my work implies as few restrictions as possible to verify if the Reinforcement Learning agent can find solutions to dynamic traffic without much help or guidance.

Some of the parameters and functions that were tested in the experiments are the following:

- **Multiple functions for observation, reward and action**. The agent was trained with different representation of the environment together with multiple ways of interacting with it. The experiments started with simple computations and went to more complex numbers that gave the agent more context of the moving cars.
- **Number of training steps**. Because training is time and energy consuming, test were conducted in finding the minimum number of training steps for the agent to match the default program performance

[1] https://inrix.com/scorecard
Available on June 7, 2021

- **Neural Network architecture**. Similar to the problem of training time, experiments were made to see how complex should the Neural Network Architecture be.
- **Policy and Memory of the agent.** How much exploration is necessary to find new optimal solutions.
- **Multiple configurations of the junction.** How different configuration of the junction affect the experiments.
- **Training and Testing on different scenarios**. The scope of those experiments was to see if the agent can act to new traffic scenarios from a restricted one.

6. THESIS STUCTURE

The next chapter of the project focuses on the **PRELIMINARIES** section which will list and explain all the technologies that were used. This part will present the simulator used, the key concepts of Reinforcement Learning and some notable Python libraries.

Afterwards, the thesis continues with the **CONTRIBUTION** chapter in which it will be explained in detail the methods used for computing all the input data necessary in the training of the AI model, together with how the performance is tested. This will be the most important section of the thesis where all the experiments and the commentaries are presented.

This chapter will present the structure of the code. The classes are designed to be used with different configuration of junctions or even system of junctions and to be easily further developed to incapsulate more functionalities.

The experiments will be presented from the first and simple ones to the more complex kinds. The most notable models will be compared with each other and with the default static programme to draw final conclusions.

The final **CONCLUSIONS** will be detailed and put in the context of the whole project in the final chapter**.**

7. HISTORICAL KEY POINTS

One of the very first papers that propose Reinforcement Learning methods to solve traffic is [1]. Here, the algorithm learns to **estimate the waiting time for each car** at each

junction, then it optimizes the reward gained from moving cars. The experiments were done in Green Light District software. In this simple scenario the AI is capable to learn and better optimize the car flow than the manual set programme.

Another big step in research was the last development of the **Simulation of Urban MObility** (SUMO) simulator in 2012 [2]. The SUMO simulator was first introduced in 2002 [3], but recent development added new features that are useful to simulate more realistic scenarios together with a Python interface that allows full control of the simulation from code.

More recent studies like [4] are using SUMO together with modern Reinforcement Learning approaches, like Q-Learning. In this research, the simulation space is discrete with dense information. This encoding is used as input for the Neural Network.

Actual studies go further away from the car flow problem and also focuses on pedestrians. This paper [5] uses a combination of Reinforcement Learning and Recursive Neural Networks to optimize the waiting time in a junction for both people on street and in cars.

II. PRELIMINARIES

The purpose of this chapter is to serve as an introduction to basic theory and details. This way the reader can easily understand the work and experiments presented in the following section.

1. REINFORCEMENT LEARNING

This section will present a short introduction in the field of Reinforcement Learning. This will help to understand the experiments and overall the objective of this thesis. Part of the knowledge is gathered from Chapter 18 of Aurelien Geron's book [6].

Reinforcement Learning is a field of Machine Learning that focuses of training Artificial Intelligence models, called **agents**, to learn from data so that they can predict accurate results. Apart from traditional supervised learning, in Reinforcement Learning the data is generated from the interactions of the agent with the **environment.** From the information gathered from the state of the environment, called **observation**, the agent takes an **action** and receives a **reward**. The objective is to find a way to **maximize** the reward received through the steps iteration of one **episode.**

The internal objective of the environment is not explicitly defined. The agent has to learn from the feedback of its actions, from trial and error.

Classical Reinforcement Learning algorithms date back 70 years ago, but the domain become more popular in 2013 with the implementation of neural networks that gave the power to the agent to learn complex Atari games just from the pixels of the games. This started the research on how Deep Neural Networks can help the model solve more complex tasks.

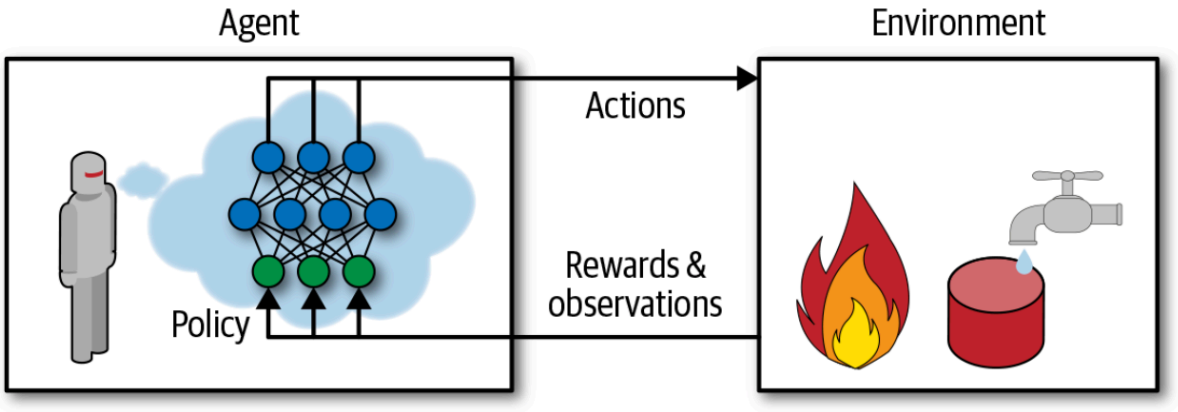

Figure 1. Reinforcement Learning process

The Neural Network who serves as the brain of the agent is the way of how the agent chooses the next action. This is called a **policy**. As is it presented in *Figure 1,* the weights of the network are update after every action applied to the environment by receiving feedback from the reward, together with the observation of the new state.

Deep Q-Learning [7] is a type of *state-of-the-art* Reinforcement Learning algorithms that uses the principle of tuning the Q-Values of a *state-action* space. The Q-Values stand from Quality Values and are a type of policy that calculate the estimated reward over time of tacking certain actions.

The specific version of the Deep Q-Network used in this research is the Dueling Network [8] that uses two Neural Networks as two different estimators for the value function and for the action that need to be taken. The weights are transferred to the decision network at a specific number of training steps if the performance is increased.

As for the exploration-exploitation policy, a linear annealed method is used to decrease the *epsilon* index of how much random action should the agent perform in the hope of finding better moves.

More details of the values of the hyper-parameters will be discussed in the experiments section.

2. SUMO

SUMO[2] is an open source urban mobility simulator that can model complex junction and traffic route systems. The simulator provides a GUI software where the user can create road networks and car routes. Also, the TraCI interface gives the possibility to retrieve values and manipulate the simulation from code. The interface comes in the form of a Python library.

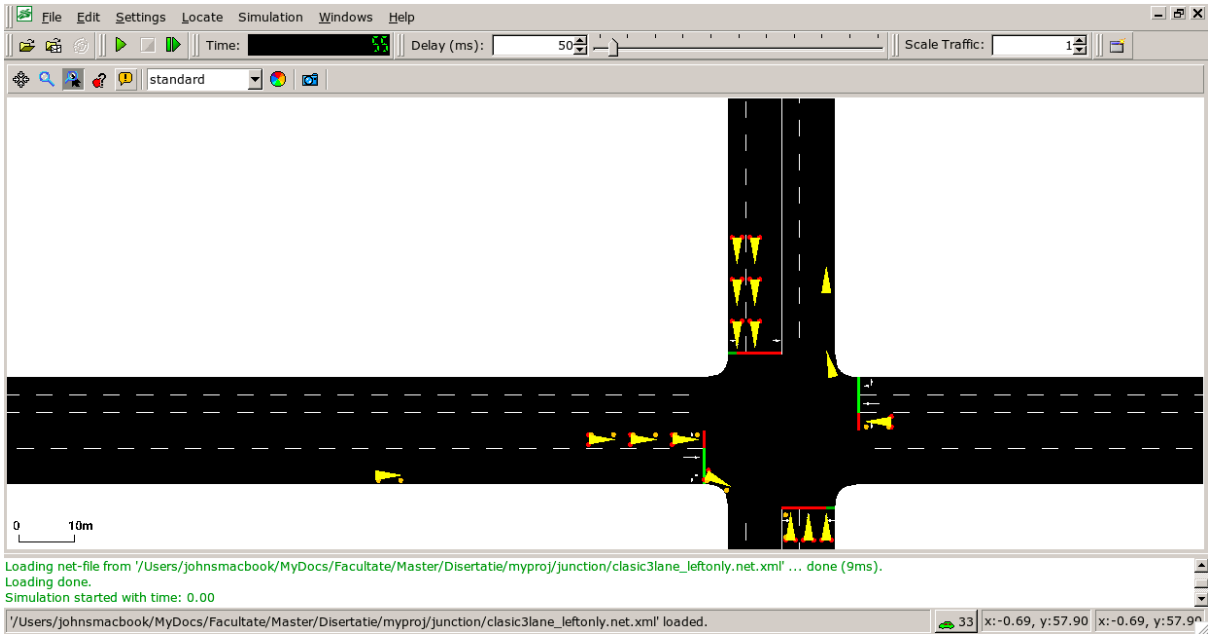

Figure 2. SUMO GUI

In *Figure 2* is presented the Graphical User Interface of SUMO and an example of road network. Here the user can build roads, configure junctions or road restrictions or even import real life networks.

3. KERAS-RL

As a final mention, the project was developed with Python and Keras-RL[3] as one of the main libraries. Keras is one of the most popular Deep Learning libraries that makes the implementation and running of DNNs much faster. Keras-RL is the version that focuses on Reinforcement Learning algorithms.

---

[2] https://sumo.dlr.de/docs/index.html
Available on June 7, 2021

[3] https://github.com/keras-rl/keras-rl
Available on June 7, 2021

III. CONTRIBUTION

The main part of this work comes in the development of **Custom Environment** class that makes possible the interaction of the DQN agent with SUMO. The Custom SUMO class acts like an OpenAI Gym[9] environment that uses the TraCI api to retrieve and manipulate the simulation. This is where the computations are made and where all the different configurations can be done. For the experiments, a classic traffic flow scenario was developed in SUMO.

1. TRAFFIC JUNCTION

This section will describe the network setup and the traffic situation that SUMO will simulate. There are two main parts of the simulation: the junction (network) and the traffic (routes).

The network of the experiment is composed of a singe **two road cross intersection**. Each road has 3 lanes, one for straight-right directions, one for straight direction and one for straight-left direction.

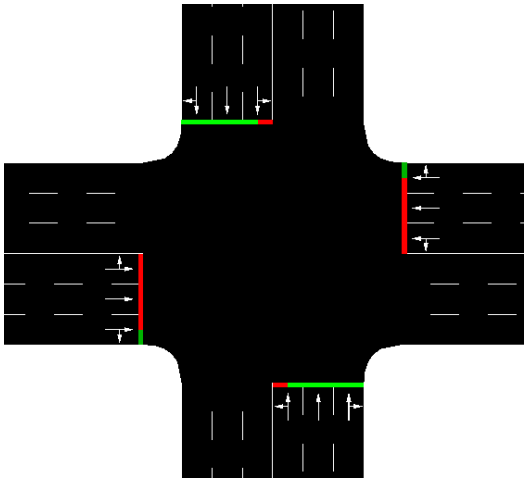

Figure 3a. Network

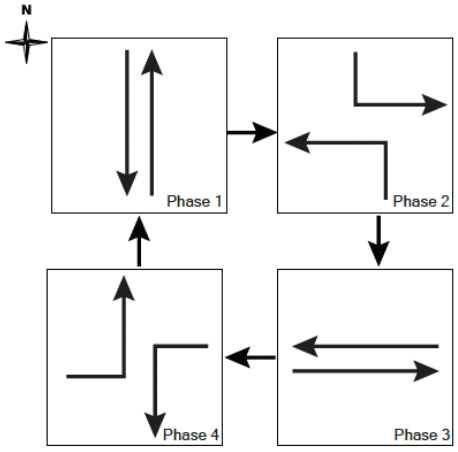

Figure 3b. TLC Green-Phases

A simple version of the junction with the Traffic Light Control (TLC) Green-Phases was inspired from [10]. The visual representation of the network can be seen in *Figure 3a* and the default directions of green phases in *Figure 3b.*

As a baseline for performance, the junction has a default programme for changing the lights of the TLC. The cycle consists of :

- 42 seconds for N-S green
- 10 seconds for N-S left turn green
- 42 seconds for W-E green
- 10 seconds for W-E left turn green

Between every change of green phase, a yellow phase is on with the duration of 3 seconds.

The routes of the vehicles consists of every possible start lane to every possible end lane. For example, there are no restriction of left turns. The traffic consists of car generated from a Non-Uniform **Probability Distribution**. A specific weight or probability can be set to increase or decrease the traffic on each possible route.

In later experiments, I modified the network to not accept straight moving vehicles on the third lane for each direction. More details will be presented in the coming sections.

2. CUSTOM ENVIRONMENT

The custom environment class respects the Gym standard in which some key methods need to be implemented. Those are the following:

- The *step* function that simulate the next state of the environment after a certain action is performed. This returns the current state as an observation, together with the reward performing the action and boolean variable that marks the end of the episode
- The *reset* function prepares the environment for a new simulation, bringing it at its initial state
- The *render* method gives visualisation of the simulation
- The *close* method exists the started simulation software.

There are multiple implementation for the observation, reward and action. Multi agent solutions can be easily implemented because the class is design to work with complex networks with multiple junctions.

The source code of the project can be seen on **Github**[4].

[4] https://github.com/JohnGiumanca/traffic-rl
Available on June 14, 2021

For the **action space**, I chose a *Discrete* gym space of 4 possibilities representing the four different standard phases of the TLC. Even though the objective of this thesis is to analyse how the agent can perform in a non restrictive scenario, giving it the possibility to turn any individual green light is not a good idea. Reason being, the agent needs a certain amount of exploration even after many training steps. It is not acceptable to turn green lights to intersecting routes. Even with very good preforming agents, in real life scenario this could be a dangerous aspect.

The **observation space** is implemented as a *Box* (min-max interval) gym space and it depends on every multiple computational method. The methods used to represent the observation are the following:

- **Observation 1**. Because the most important aspect of the traffic junction is the moving and stopped cars, the first implementation of the observation is the count of stopped cars for each lane. Another important aspect of the environment state is the current traffic light phase. Putting them all together, the observation is represented by an array of size 13, the first digit represents an integer value in a *[0,3] Box* representing the **current green phase**, followed by 12 digits in the *[0, inf] Box* that represents the **number of stopped cars** on each lane. The stopped cars are considered to be waiting at the traffic light.
- **Observation 2.** The first method for observation is a more naive method of Observation 2. Because the observation is fed as input for a Neural Network, a better representation is necessary in the scope of faster convergence and good performance in less training time. So the second method is similar to the first but it uses **One-Hot Encoding** for the current green light phase and **normalized values** for the number of stopped cars. The values are transformed in the *[0,1]* interval and they represent the percentage of space occupied by stopped cars. The array now becomes of size 16, accommodating the 4 values of the encoding for the traffic light state.
- **Observation 3.** In the next method, the agent receives more context from the traffic by also seeing the **total number of cars**. Another 12 normalized values are appended to the observation array representing the percentages of moving and stopped cars on each lane. This should give a better understanding of the

dynamics of the environment and should make the agent perform actions depending also on the moving cars. In this case, the input values represents a 28 digit array with values in the *[0,1]* interval.

The **reward** function has multiple implementations but all of them are related to the idea of a penalty given to the agent if cars are waiting too much at the red traffic light. The implemented methods compute the reward/penalty as follows:

- **Reward 1**. One way to punish the agent if it leaves too many cars waiting on the red light is to receive a **-1** value for **each car stopped**. At one step of the simulation, the agent will get the negative value of the total waiting cars. In this case, the maximum of reward that the agent receives if it gets a perfect run in an episode is 0 for no cars waiting.
- **Reward 2**. The second method has the same principles as the first one but it is adapted to the normalized values of **Observation 2** .
- **Reward 3.** The next method uses the **accumulated wait time** for each vehicle. In this way, the agent receives a more sever penalty for leaving cars on red light on longer periods of time. A car that just stopped at the red light will return its waiting time as a negative value for the reward function, in this case *-1,* in the second simulated step of waiting it will return -2, and so on. In other words, if in the past method the penalty for leaving the cars waiting increases linearly, in this method it increases exponentially. If a queue of cars receives a green light, but not all the cars will cross the junction, the cars that stop again will reset their waiting time. To prevent this total value from exploding, the reward function uses the average of the accumulated waiting times of every lane.
- **Reward 4**. The last method is based on the academic paper [11] and it consists of using the **speed of the cars** from the simulation. This way, the agent has to maximise the speed of the cars that are traversing the junction. This single value is calculated as the average of all speed values of every car.

A simulation episode **ends** in two main ways: the **maximum number of steps** is reached or the **maximum queue length** limit is passed. Both of the thresholds are configured before the start of the simulation.

The maximum queue length is calculated as a percentage of space occupied from the waiting cars on a specific lane. For example, if the limit is set to 0.50, the episode ends before the maximum number of steps if at least one lane has the stopped cars queue occupying more than 50% of the space. The footprint of the vehicle and the distance between cars is taken into consideration.

In conditions of heavy traffic, there is a high change that the agent, in the learning process, will not be able to keep the queue lengths low. In this case, the agent should receive a penalty. This penalty is reflected in the reward functions. If the occupancy threshold is reached, the reward functions return a very small negative value representing a big penalty instead of the normal reward value.

With this penalty system, before optimizing the traffic flow, the agent must **first find a way to keep the episode running** until the end.

Another safety restriction set in the Custom Environment is for the way the agents sets a new traffic light phase. When a new action is set no change the traffic phase, between 2 green phases, a yellow light is required. This is handled internally by simulating the required number of steps in SUMO before the seconds of the yellow phase end. By doing this, the agents sees the observation of the environment after the yellow light ends. This assures the traffic light will perform close to a real life situation.

A suggestion from [11] implies setting a minimum time for a green light. For this thesis, I consider that the agent should be able to learn to adjust the timing of light. In real life, having green light only for one second may be problematic taking into consideration the human reaction. Because a simulator is used, the reaction of the cars is not a problem and experiments can be done without a **minimum green time**.

3. INITIAL SETUP

The experiments start with the initial setup where modification were made to test other parameters values. In the first experiment, the maximum duration of an episode is **500 steps**. In this length of time the default programme can run for approximately 4 cycles. The traffic is set to **200 cars**. This means that 200 vehicles are spawn randomly in network uniformly over 500 steps. The 200 cars on 500 simulation steps represents **light traffic**.

The weight array for the car spawn probabilities is **uniform**, a car has equal chance to have any start and end route points.

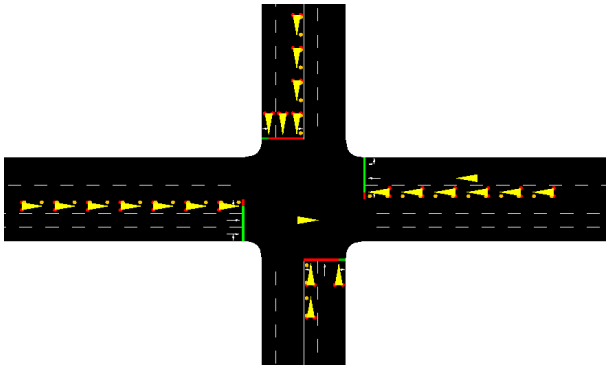

Figure 4. Light Traffic

In *Figure 4,* there is a visualization of light traffic in the simulator. Apart from the left-turn lanes, this situations is managed well by the default system. The downside is that the green phases are too long and after some time, no cars are left to pass.

The Deep Neural Network architecture for the DQN agent is composed of 3 Hidden Layers of 16, 32 and 64 neurons respectively. The more important parameters set to the DQN agent are the size of the memory buffer and the number of steps at which the exploration-exploitation epsilon reaches a minimum value. In other word, from how many examples should the agent learn and how does it handle the exploration-exploitation tradeoff.

Usually those values are set to be half of the total number of training steps. In the first half of the process, the agent explores the environment and memorize its moves and in the second part it uses the knowledge to improve itself.

4. RESULTS

The experiments with **Observation 1** and **Reward 1** all concluded with bad result. The focus will change to the next methods. The second set of observation and reward is similar, but due to the normalized state they have a better chance to show good results in a fewer number of steps.

Experiment 1

In the first experiment, the training was run 100.000 steps using the **Observation 2** and **Reward 2** computational methods. The learning rate was set to 0.001.

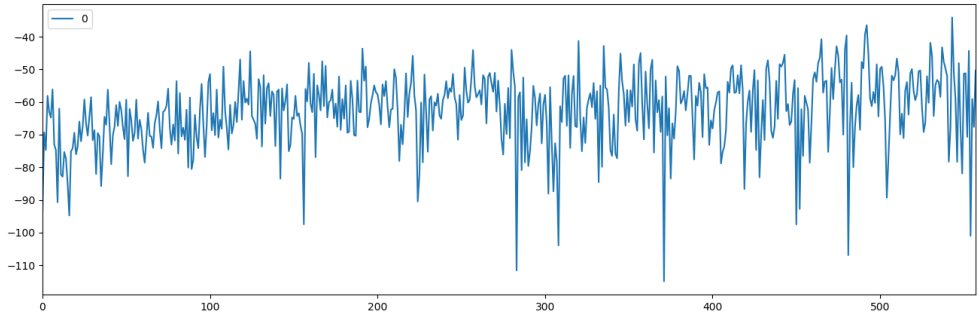

Figure 5. Training history Experiment 1

The training ran for about 600 episodes. In *Figure 5* is presented the training history and how the reward return changed over time. There are no evidence of the reward per episode to go up. With this kind of plot, there are no expectation that the agent will perform any better than the default programme. The training was done on light traffic simulation episodes.

For testing, several new traffic files are generated. This is design to better reflect the performance of the model. A specific number of traffic files are generated and both the agent and the default TLC are run on them. The final history of a reward then is computed from the mean of the values at every step of the simulations. Another thing to mention is that, because the traffic is different at every run, there is the possibility that in certain simulations, one episode will terminate early. That is why in the final evaluation, the arrays are truncated to the size of the smallest ones. Also, the testing is done in different traffic scenarios by configuring the car spawn probabilities. For the next experiments, the tests are done on **5** different **simulations.**

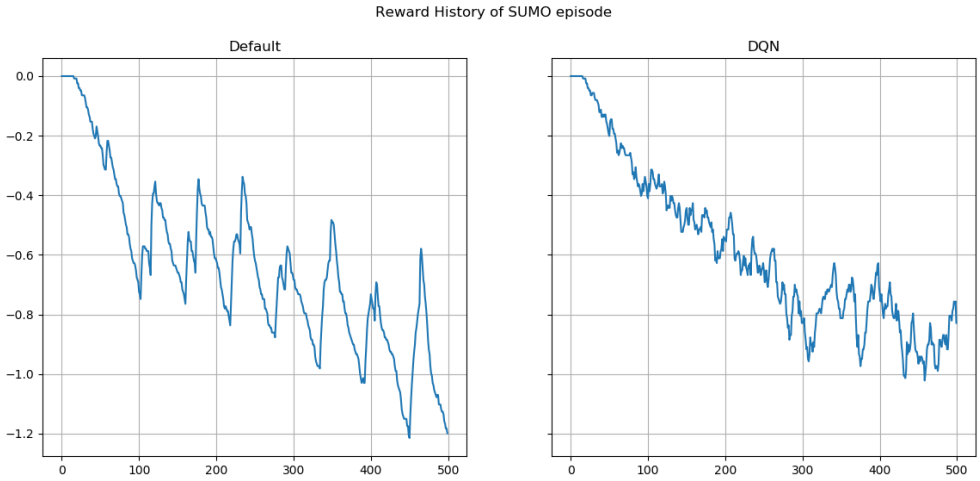

Figure 6. Mean Reward History on Light Traffic

The results can be seen in *Figure 6*, on the left side, the mean reward for a simulation episode controlled with the classic system oscillates in the **-0.0** and **-1.2** values and in the right side the values are between **-0.0** and **-1.0.** The DQN managed to receive bigger rewards compared to the classic TLC. The history line or rewards does not fluctuate as much as the one of the default programme.

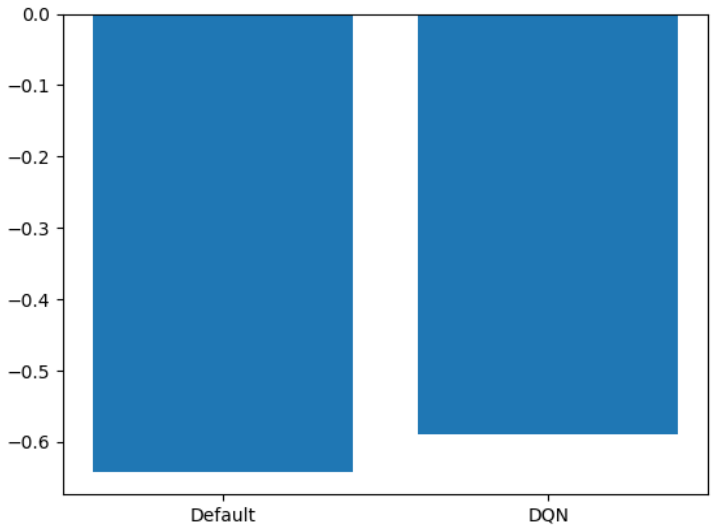

Figure 7. Average Reward per Episode on Light Traffic

Another way of comparing the performance is by looking at the overall average episode reward. In this case (*Figure 7*), the DQN has a better performance than the default TLC with a lower average waiting time .

In the next tests, the performance on **heavier traffic** is visualised. For that the value of the number of cars generated in one episode is increased from **200 to 400.**

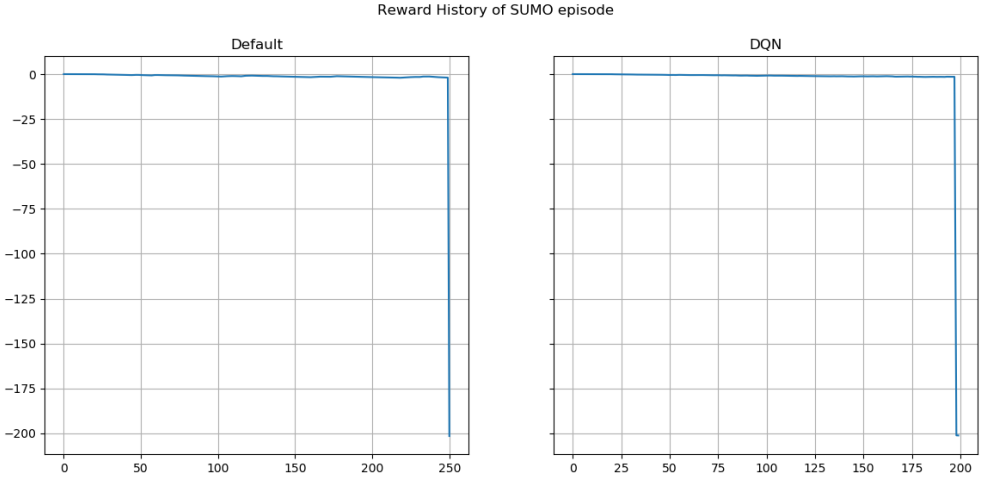

Figure 8. Mean Reward on Episode on Heavy Traffic

Figure 8 shows the test run on heavy traffic. There is an early termination around the 200 step of the episode. In cases of a harder task, the important aspect to observe is how long a programme can hold the simulated episode. In this case the default TLC managed to get a little further to 250, but did not do as well to finish the episode. Observing the average reward in this case is not relevant. Due to the very low reward received from early termination, the DQN will surely perform worse on this test.

Experiment 2

Next experiments focuses on the **Reward 3** and **Observation 3.** The hope is that a more severe punishment computed in Reward 3 with the extra traffic context from Observation 3, will result in better performance.

The setup is similar from the previous experiment but shorter episodes are used. The maximum steps of an episode is **200** with **100** cars simulated. This ratio also implies a light traffic.

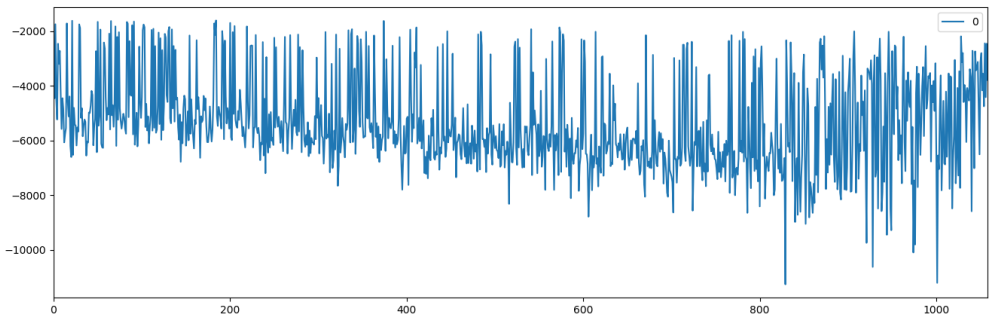

Figure 9. Training history Experiment 2

Due to the shifts from smaller to bigger values, *Figure 9* shows how, from the most part of the training, the model receives a high penalty for early termination. Some episodes are normally finished due to the higher reward value. At the end of the training there might be slightly up trend.

The tests were done at 500 steps episode, so there is a better comparison with the rest of the experiments.

Figure 10. Mean Reward o History on Light Traffic (Exp. 2)

In *Figure 10,* the DQN performs worse in handling light traffic due to the lower values for the reward. The DQN almost reaches the end of the 500 steps episode but it could not keep the queue under the limit.

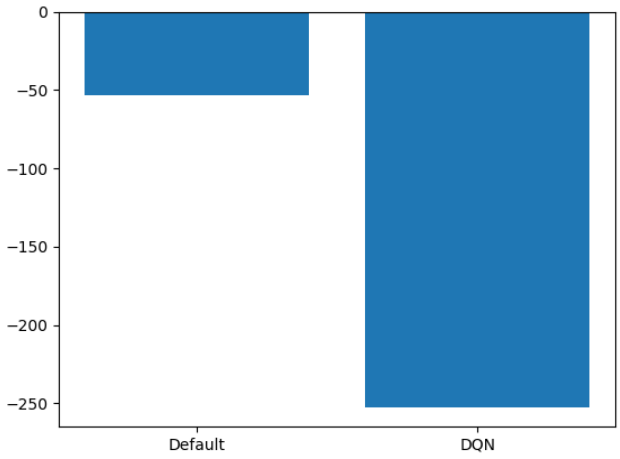

Figure 11. Average Reward per Episode on Light Traffic (Exp. 2)

This is also confirmed by *Figure 11,* where the DQN has a worse reward value corresponding to a longer overall waiting time for cars.

The next test is done on **medium** traffic with 300 cars in the simulation. Here, in *Figure 12,* we see that both the classic TLC and DQN failed to manage the quantity of cars. In this case, the comparison is done by looking at the number of simulation steps. Both algorithms did about 350 steps with a slight advantage to the Default system.

Figure 12. Mean Reward o History on Medium Traffic (Exp. 2)

The average reward is lower for the DQN, *Figure 13* confirms it. But the difference in average reward is rather big, more than double.

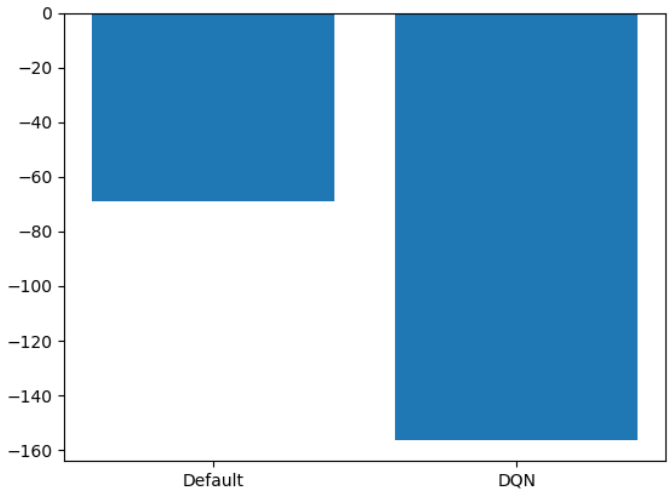

Figure 13. Average Reward per Episode on Medium Traffic (Exp. 2)

As a conclusion, it seems that with the short episode time of only 200 steps, the agent was not possible to learn to control the traffic in efficient ways.

Experiment 3

Next, the agent is trained on the specific traffic scenario of **500** steps per episode with a medium number of **300** cars. The training was done on 100.000 steps.

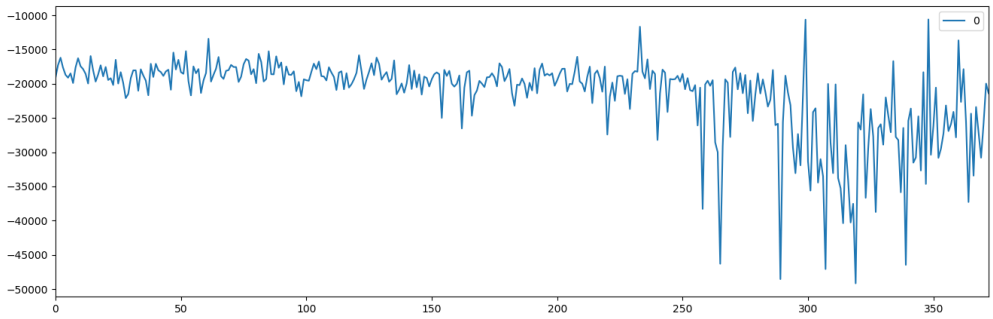

Figure 14. Training history Experiment 3

From the training history there is no conclusions to be made of good performance, the reward does not seem to go up. (*Figure 14*).

Figure 15. Mean Reward o History on Light Traffic (Exp. 3)

The plots from the testing (*Figure 15*) also confirms that the new model is not performing well even in light traffic situation. The agent does not act randomly, it seems that a certain point it managed to release a long queue of cars to decrease the waiting time (or increase the reward).

Experiment 4

In the forth experiment, the training period was increased to **150.000 steps**. The length of one episode was changed to **1000 steps** with **400 cars**.

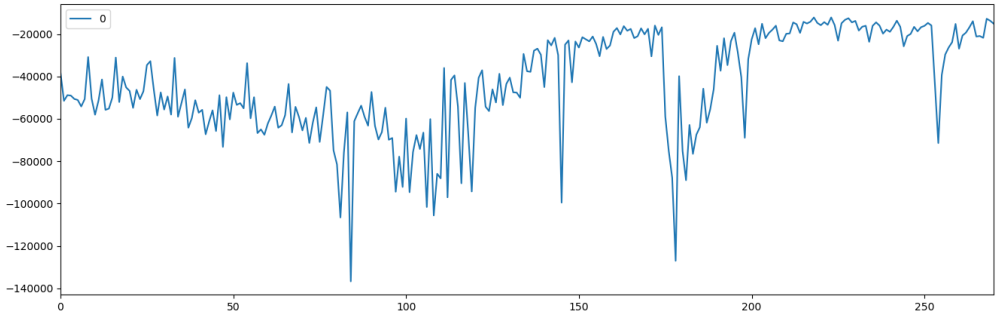

Figure 16. Training history Experiment 4

Good signs are shown in the plot of the training history (*Figure 16*). The total rewards for an episode increases in the end of the training process.

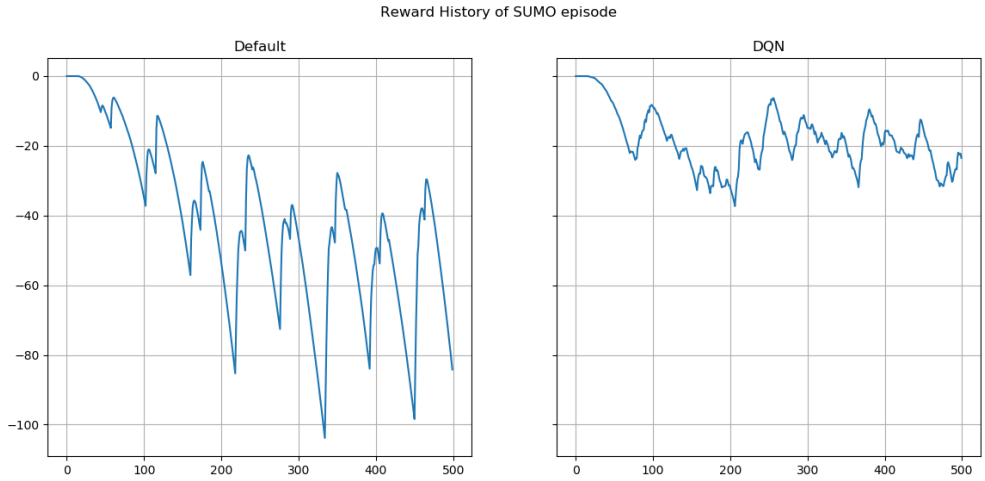

Figure 17. Mean Reward o History on Light Traffic (Exp. 4)

Figure 17 shows how DQN does very well in the scenario of light traffic.This can be by accident, the agent may perform **random moves** that are enough to handle the light traffic. For this we will test the situation of light traffic from a single direction, more specifically the only routes of the cars in the simulation will be West-East.

Figure 18 shows that in this specific care of light traffic with a single right decision of keeping green light in the **WE direction**, the agent manages to do right that. This proves that the agent does **not randomly act** in the environment**.**

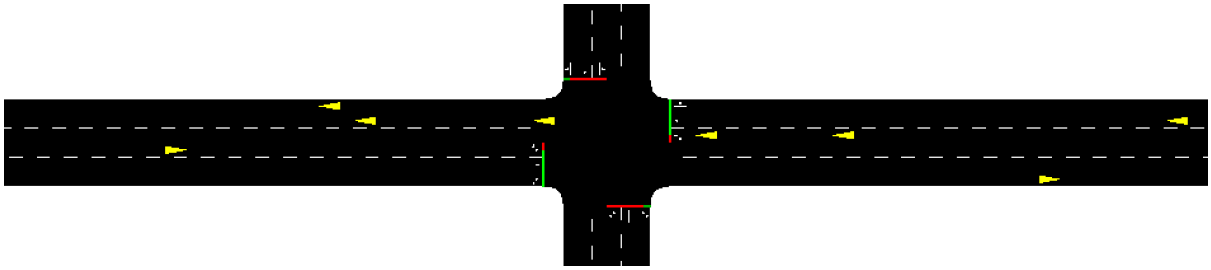

Figure 18. Visualization of the DQN policy

In the next case, the heavy traffic scenario was tested. In *Figure 19,* both algorithm fail the episode around the same time. So, from this perspective, the DQN matches the performance of the classic TLC. But, the overall waiting time is lower for the DQN, the reward values are, for most of the time, over the **-50** value, compared to the default TLC that has many values under **-100.**

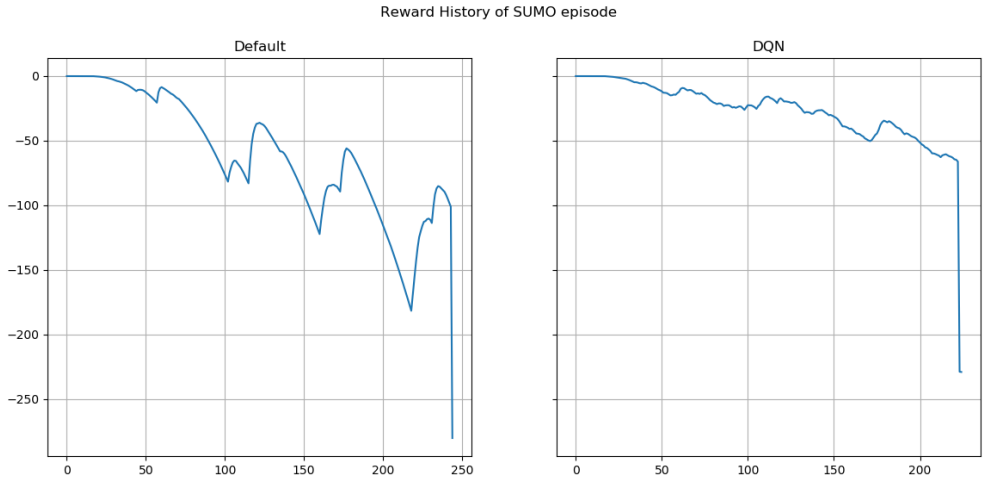

Figure 19. Mean Reward o History on Heavy Traffic (Exp. 4)

This is also confirmed by *Figure 20,* where the plot shows that the average waiting time for the DQN is less than half compared to the classic DQN.

Next, the objective is to see if the model can handle **new cases** that it never seen on training. The next test has the traffic to be **double from one direction**. More specifically there are double the amount of cars that are coming from **West** and going to every direction. In *Figure 21* we see how the simulation went. The 2 algorithms could not reach the final steps of the episode but the DQN manage to keep the simulation running for longer. The agent did reach 160 steps compared to the 100 reached by the classic TLC.

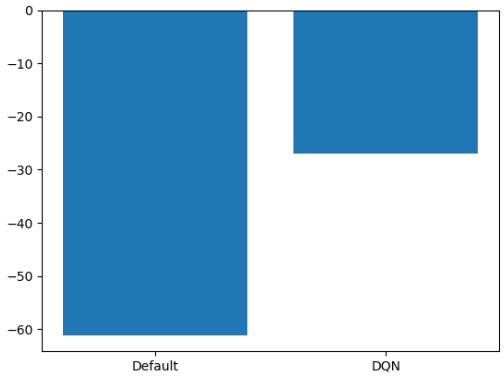

Figure 20. Average Reward per Episode on Heavy Traffic (Exp. 4)

Figure 21. Mean Reward o History on One-Way Traffic (Exp. 4)

Experiment 5

From this experiment on, there will be some modification on the junction configuration. The lane for left turn will no longer accept straight moving cars. The reason is that some of the generated cars with routes going in straight directions will also occupy the last lane designed for left turn. This leads to unnecessary longer wait times for left turning cars. So the last lane of every road is changed to **left-only**.

Another modification comes in the traffic generated for the training of the agent. Now the traffic that goes for a **left turn** will be generated with **half the probability** compared to

the cars going straight or right. The reason is that there is only one lane for left turn compared to 2 lanes for straight moving.

Those modification come in after seeing that the default programme can not beat a heavy traffic situation and the theoretical limit of the junction is reached. This can put the agent in a rather impossible and confusing situation. A visualization of the modified junction can be seen in *Figure 22.*

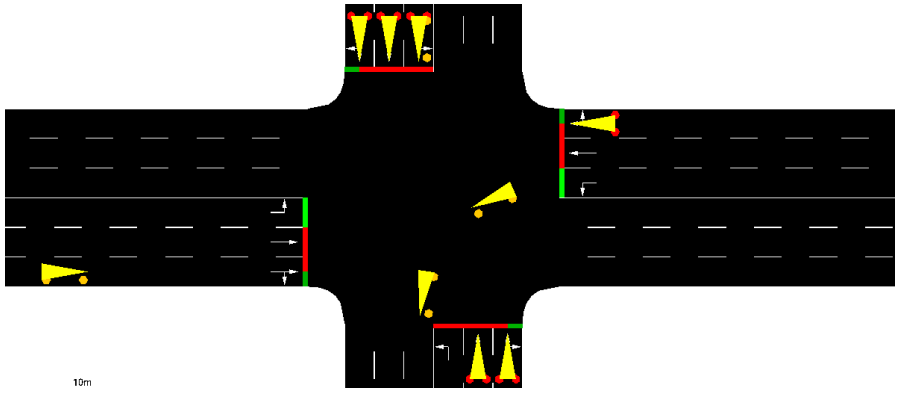

Figure 22. Modified junction with left-only lanes

With the modified junction, a **100.000** steps training is done of light traffic simulations of **500** steps and **200** cars. The training history is presented in *Figure 23. T*he plot looks good with a clear upward trend.

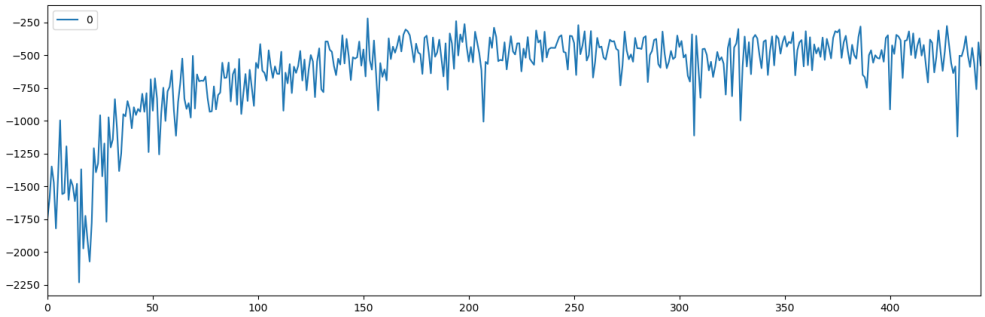

Figure 23. Training history Experiment 5

Because it was proven that the agent can beat the classic programme in light traffic tests, the focus will be only on the tests with **medium** and **heavy traffic**.

Figure 24. Mean Reward o History on Medium Traffic (Exp. 5)

In the case of medium traffic with approximately the same number of cars for each direction, both algorithms finished the episode , but the DQN had lower waiting times in most of the simulations steps. The result can be see in *Figure 24*.

In heavy traffic, the agent could not beat the other method, finishing the episode with 100 steps early compared to the classic system (*Figure 25*).

Figure 25. Mean Reward o History on Heavy Traffic (Exp. 5)

From the tests with double traffic from one direction, the agent did a much better job in light traffic and similar performance in heavy traffic. Both tests can be seen in *Figure 26.* For the light traffic, the **DQN** had **better** times in almost every single moment of the

simulation. In the harder scenario, both methods failed the episode but at rather the same time of **200** steps.

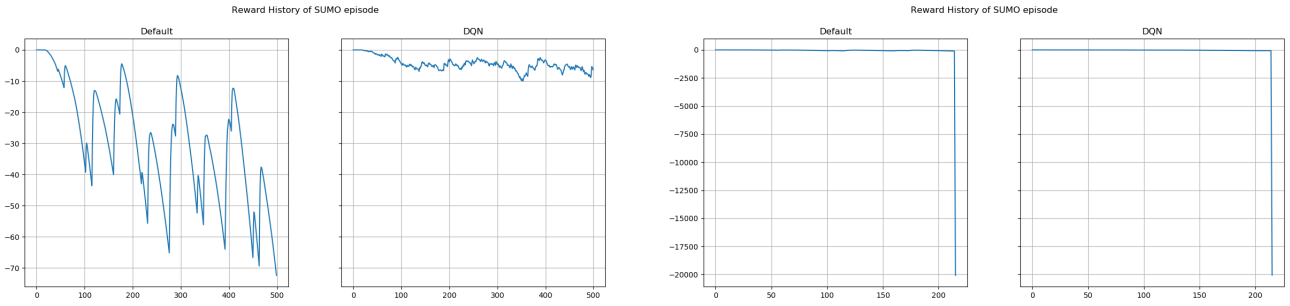

Figure 26. Mean Reward o History on Light and Heavy Traffic (Exp. 5)

In conclusion, training with light traffic episodes for 100.000 steps, is enough for the agent to learn to manage the traffic better than the classic system in similar scenarios. Moreover, the DQN did better in dynamic traffic situations that it never seen before.

Experiment 6

In the next experiments, the attention was on training the agent on heavy traffic situation. For that, the training period was raised to **200.000** episodes with **500** steps simulation and heavy traffic of **400** cars spawned.

A **modification** was made in the **reward** method. Previously, the agent receives a high penalty for the early end of the episode. This penalty is a static value and it adds up to the previous rewards received before that point. Therefor, if in one iteration the agent will end the episode close to the maximum steps allowed, it will receive a greater penalty compared to the situation of ending the episode close to the start because of the extra penalties along the way. This is counter intuitive for the agent and it might confuse it in the learning process because keeping the episode running for longer (but not until the end) will result in a worse rather doing a bad job and ending it earlier. To counter this problem, I set the penalty system for the early termination t**o be computed dynamically** taking in consideration the remaining number of steps until the maximum is reached. For a larger number of remaining steps, the agent will be penalised more compared to a few until the end.

The comparison of the static and dynamic penalty for early ending of an episode can be seen in *Figure 27.* The **upper** plot show the training with the static penalty. All of the

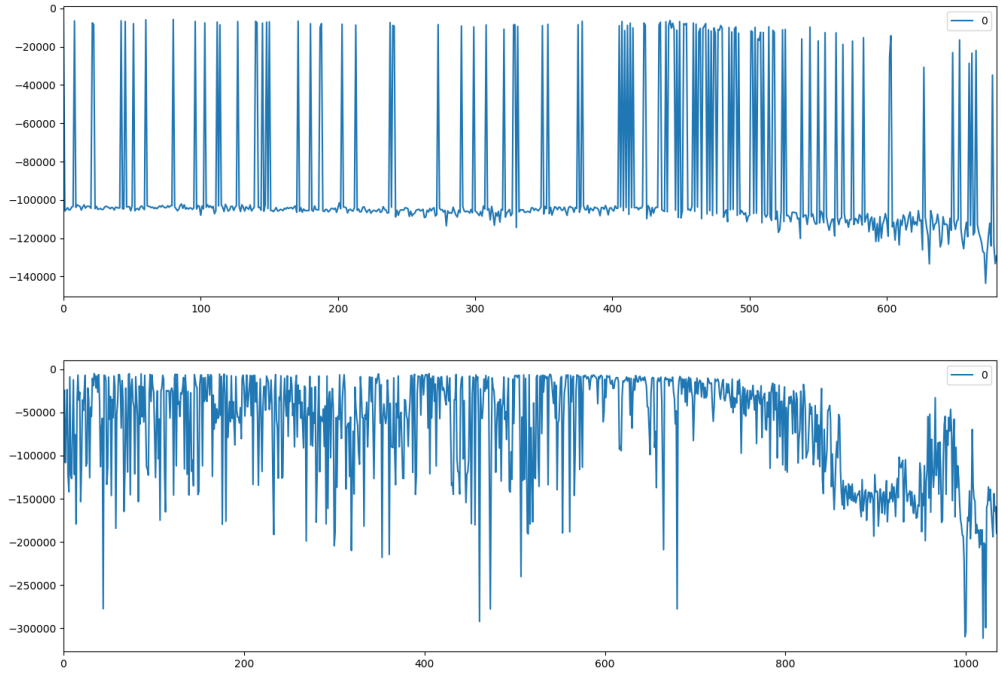

Figure 27. Training history Experiment 6 (Static and Dynamic penalty)

values are split into very high or very low depending if the episode ended earlier than expected. Even though the agent managed to have some episodes running until the end from exploration of other possibilities, it was not able to learn the patterns to keep the good results on a constant rate. The **bottom** part of the figure show how the dynamic penalty helps with convergence. The values stayed up for most part of the training but they fell at the end even though the second agent was train longer for 200.000 steps.

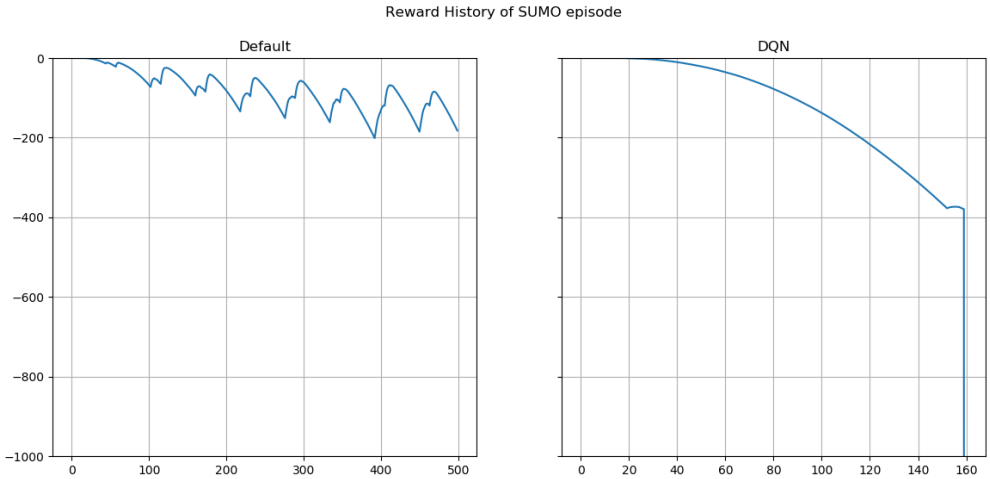

Figure 28. Mean Reward o History on Heavy Traffic (Exp. 6)

Figure 28 show poor results from the testing, the agent got only to the 160 step before ending the episode with an over the limit queue.

Experiment 7

Because finding patterns in complex scenarios implies a longer rate of exploration, the last experiment was run again but with a **larger exploration rate.** More specifically, the parameters of the *epsilon* decay were modified to a larger value. In this case, the final rate of exploration will be greater.

Increasing the exploration rate did not work. The agent did not have time to learn from the data because the rate of random moves was too high. Lots of episodes finished very fast, details in *Figure 29.*

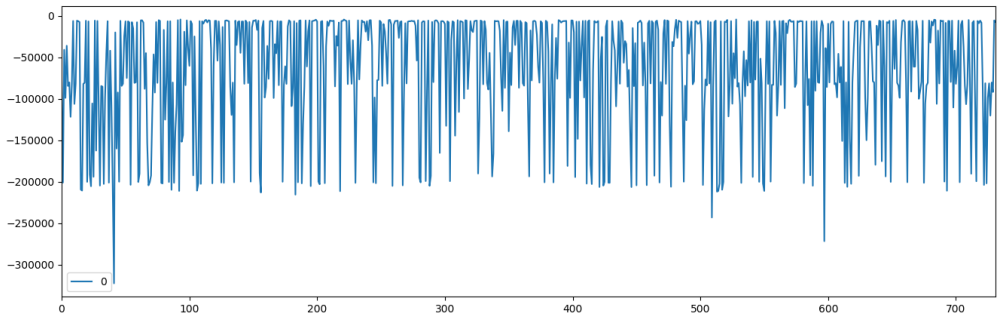

Figure 29. Training history Experiment 7

Experiment 8

The next approach is to simplify the network. The number of neurons my be too big for the network to converge in only 100.000 steps. For the next experiment I removed 2 out of the 3 hidden layers, keeping only the one with **128 neurons.**

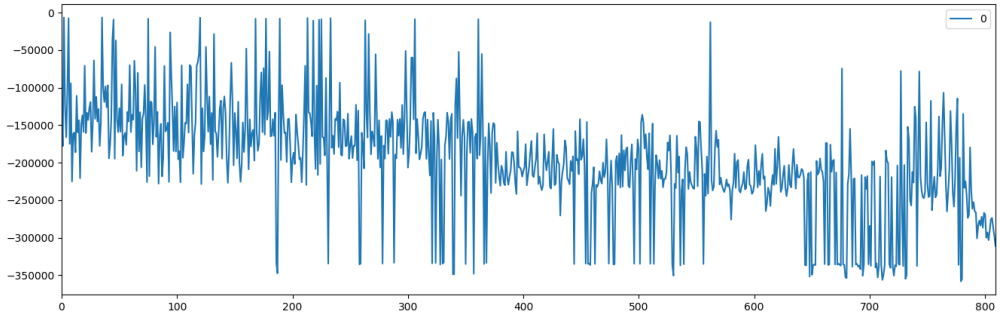

Figure 30. Training history Experiment 8

The training can be seen in *Figure 30* with the trend of the reward line going down. For the past experiments, tests of high traffic performance were not shown because from the training history the agent did not learn to handle big traffic with the last modifications.

Experiment 9

Next, the environment was configured with the **Observation 3 and Reward 4** computational methods. The training was done again on **100.000** simulation steps. The learning rate is still set to 0.001.

The reward history (*Figure 35*) does not suggest good testing results. The reward line does not have an up trend so there is no evidence of convergence for the loss function.

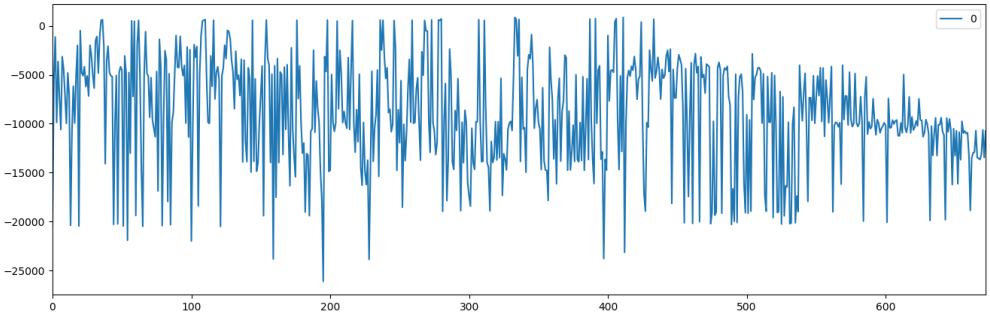

Figure 35. Training history Experiment 9 - Learning Rate 0.001

Included in this section, several experiments were done with the learning rate value to see if bigger or smaller steps in updating the Neural Network weights change the performance of training over 100.000 steps.

Another 3 models were trained with the values 0.01, 0.0001 and 0.00001 for the learning rate. The results can be see in the following figures. In *Figure 36*, the plot represents the training history with a higher learning rate. The rewards oscillate between very high to low values, a sign that the weights update is too strong.

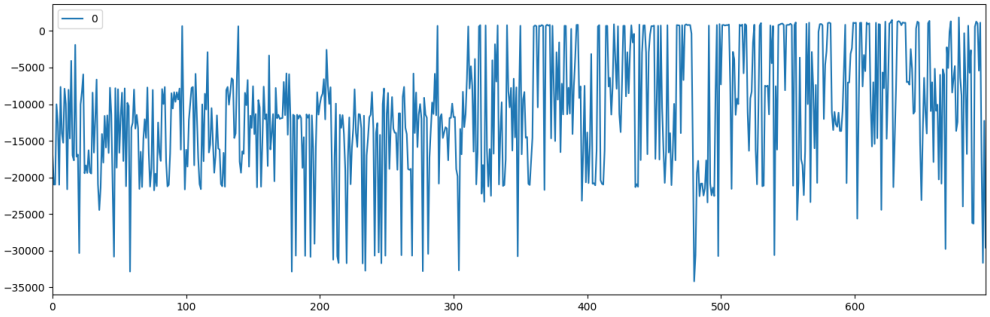

Figure 36. Training history Experiment 9 - Learning Rate 0.01

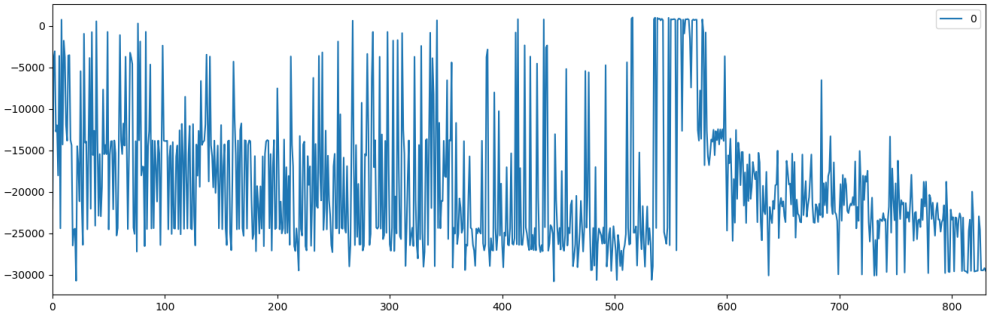

Figure 37. Training history Experiment 9 - Learning Rate 0.0001

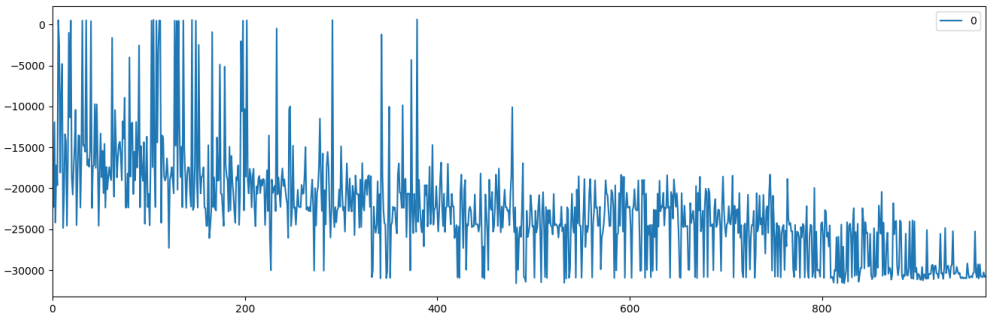

Figure 38. Training history Experiment 9 - Learning Rate 0.00001

The following results presented in *Figure 37* and *Figure 38* suggests that lower values for the learning rate give worse results. Because the plots do not suggest any increase in performance over the past experiments. there is no need in further testing the agent against the default TLC.

Experiment 10

It is clear that in dynamic situations of light and even medium traffic, the agent can outperform the classic system taking in consideration the waiting time of the cars in the junction. In harder situations, the agent with the best performance only matched the score of the classic TLC. The past experiments show that even with lots of modifications to the parameters of the agent and even with changes in the junction setting it is very hard to obtain good results in heavy traffic with small amounts of training.

In the next sections, experiments were done with large training periods of **1 million** and **2 million** steps. The other settings are using **Observation 3** and **Reward 3** again,

because Reward 3 gave better results in testing. The simulation still has 500 steps and 400 cars corresponding to heavy traffic.

A major modification for the next experiments is the use of **multiple observation** as input for the Neural Network. The input data will be **sequences of 4 observations**. This way, the agent can better see the dynamic of the cars incoming.

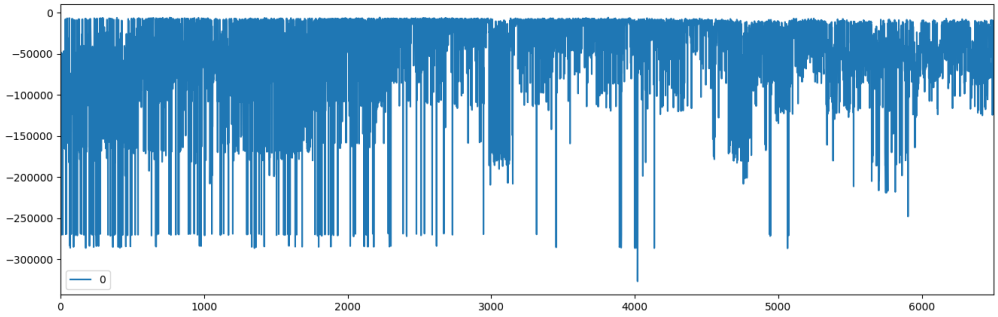

Figure 39. Training history Experiment 10 - 1 mil. steps

The first run of 1 million training steps resulted in a good reward history plot presented in *Figure* 39. The reward per episode converges to a smaller interval at the end of training. It can be observed the high volatility of exploration at the beginning.

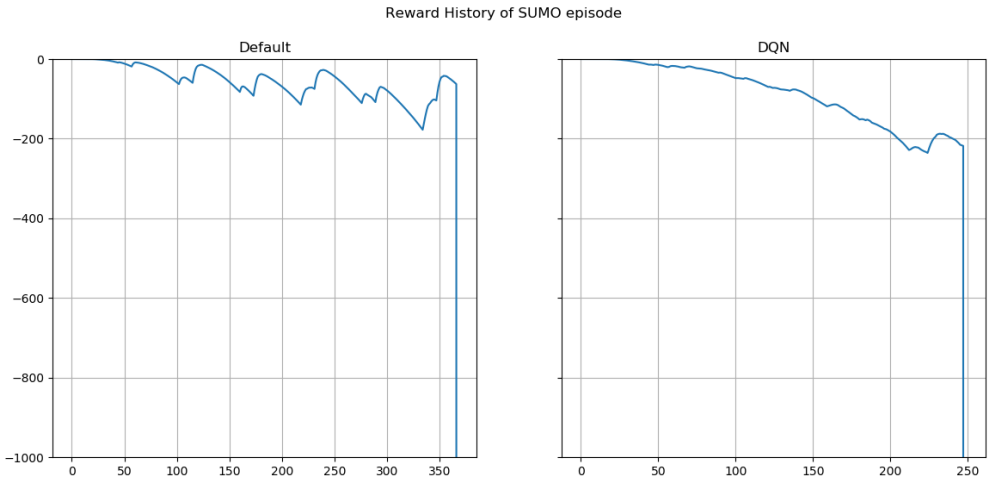

Figure 40. Mean Reward o History on Heavy Traffic (Exp. 10) - 1 mil. steps

Surprisingly, the agent did not perform well in practice. *Figure 40* shows that the agent performed poorly, only getting to the 250 step mark in the heavy traffic simulation.

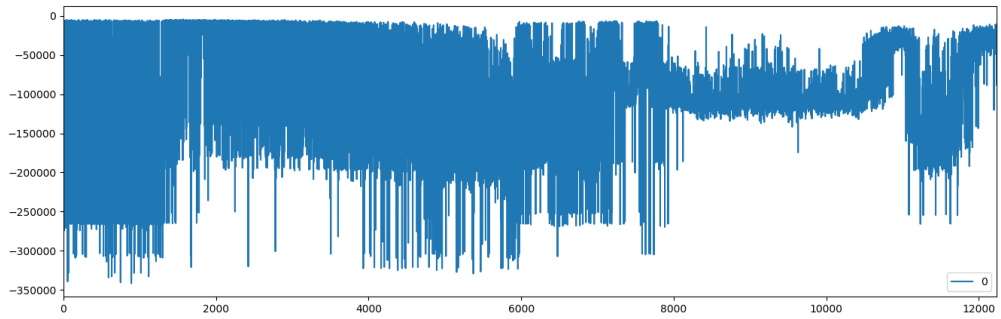

Figure 41. Training history Experiment 10 - 2 mil. steps

In the case of 2 million steps of training (*Figure 41*), the results are similar to first run of this section. There was a large volatility that comes with the high exploration policy followed by the convergence to an upper value.

The results from testing are also the same, the model failed to solve the heavy traffic problem. (*Figure 42*)

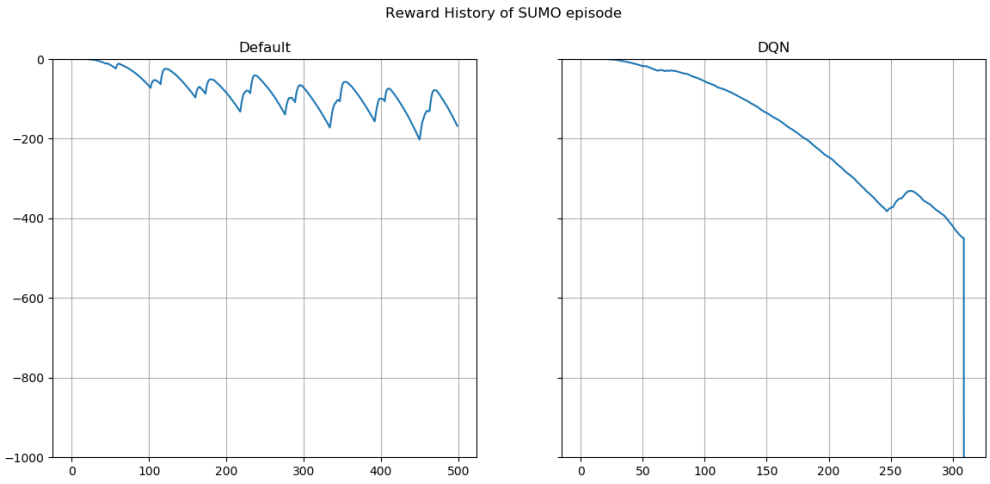

Figure 42. Mean Reward o History on Heavy Traffic (Exp. 10) - 2 mil. steps

One way to interpret the situation is that with such a long period of training and with the cars spawning the same way every simulation, the agent might have overfit the date and only learned that specific episode without being able to generalise to a similar but different situation.

Experiment 11

To handle the problem from the last section, the traffic file was set to be re-generated every single episode. This way, the agent will not get stuck learning only a very specific scenario. Again the agent was trained for 1 million and 2 million steps. The rest of the setting were the same.

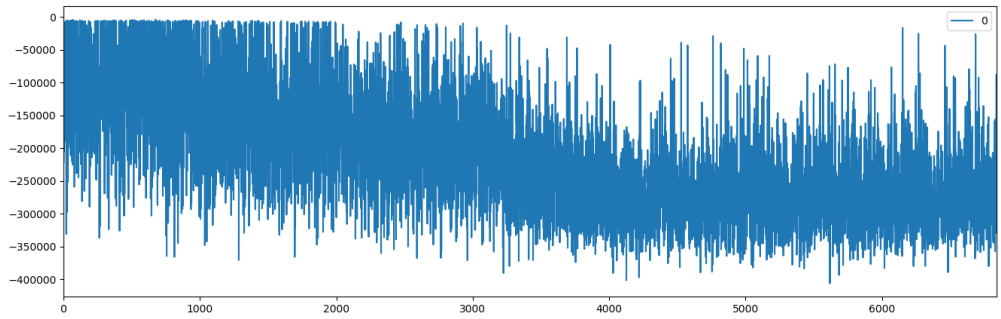

Figure 43. Training history Experiment 11 - 1 mil. steps

The plot in *Figure 43*, does not show sign of learning. The reward per episode stay low in the last part of the training. The testing from *Figure 44* show even worse performance than the last experiments.

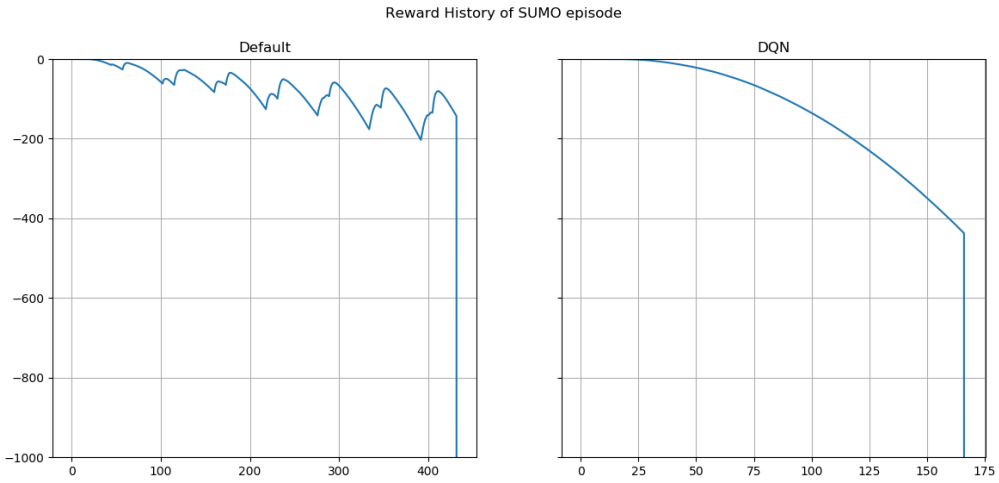

Figure 44. Mean Reward o History on Heavy Traffic (Exp. 11) - 1 mil. steps

The longer period of 2 million steps does not change the results both in training and testing. The results can be see in the following figure: *Figure 45* and *Figure 46.*

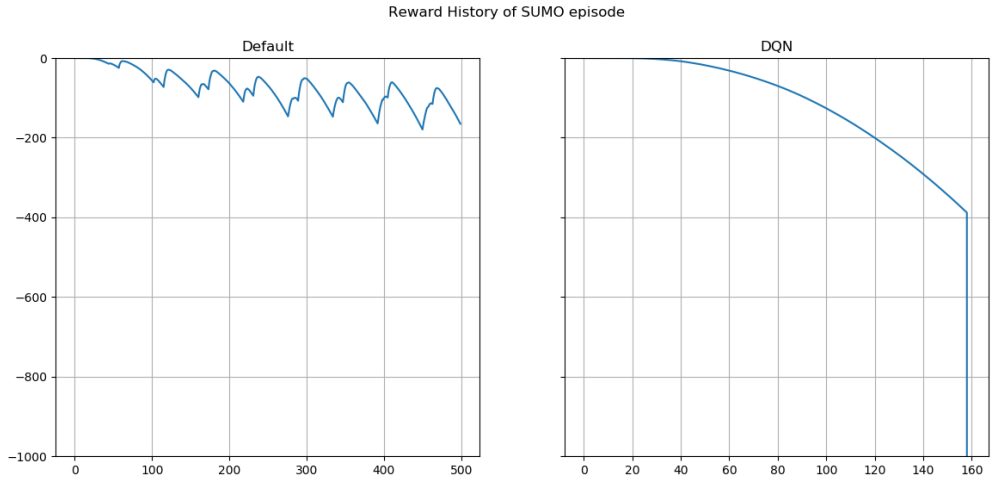

Figure 46. Mean Reward o History on Heavy Traffic (Exp. 11) - 2 mil. steps

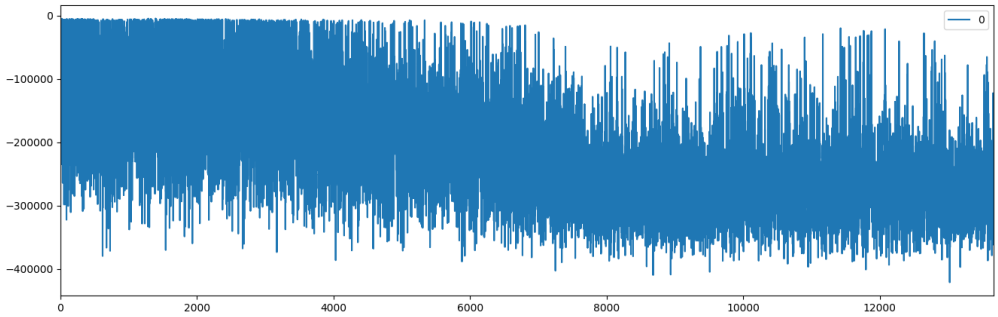

Figure 45. Training history Experiment 11 - 2 mil. steps

In conclusion, even from long periods of training, the **agent can not handle heavy traffic** better with the current configuration of the junction, observation and reward systems.

IV. CONCLUSIONS

In the last section of the thesis I review the insights discovered in the experiments and present my final conclusion for this research study.

From the past experiments the following conclusions ca be made:

- **Experiment 1.** Within the first tries of experimenting with simple observation and reward function, good results can be achieved for the scenario with light traffic.
- **Experiment 2.** The training with short episodes was not efficient, even with better reward and observation functions.
- **Experiment 3.** Training on longer episodes but with medium traffic was not good enough to beat the traditional system in light traffic testing.
- **Experiment 4.** Longer episodes of light traffic together with a longer training process, delivered the best results in testing for light and medium traffic. The performance in harder situations was similar with the classic TLC. The tests show that the results are not accidental, in trivial scenarios, the agent delivers correct actions. This is the best result found.
- **Experiment 5.** Changing the junction and traffic settings and training on small number of cars, did not result in better results. The agent did well in easy traffic but failed in harder levels.
- **Experiment 6.** Changes in the reward system and an even longer training time did not deliver better results in testing.
- **Experiment 7.** The agent did not show signs of training improvement from using a more exploration focused policy
- **Experiment 8.** Bad results also from using a simpler Neural Network architecture.
- **Experiment 9.** Changing the learning rate did not improve the learning process.
- **Experiment 10.** Training on very large number of steps may result in overfitting the simulation and the agent may not be able to generalize even in simple cases.

- **Experiment 11.** Training on very large number of steps with new generated traffic every episode, may be too hard to learn from and may need even longer training periods or more complex observation and reward systems.

The **final conclusions** are that training a DQN agent to control a traffic light better than a traditional algorithm is a complex problem that requires further research to fully solve. The experiment presented in my thesis show that the agent is capable of dealing with dynamic situations better than the classic system. With few resources and a relatively short training process, the DQN managed to outperform the classic TLC in light traffic scenarios and to match the waiting time in harder situations. For the heavy traffic scenario, experiments with more complex environment settings need to be done.

BIBLIOGRAPHY

1. Marco Wiering, J.V., Jelle van Veenen, and Arne Koopman, *Simulation and Optimization of Traffic in a City.* 2004.

2. Daniel Krajzewicz, J.E., Michael Behrisch , Laura Bieker-Walz, Recent Development and Applications of SUMO - Simulation of Urban MObility. 2012.

3. Daniel Krajzewicz, G.H., Christian Feld , Peter Wagner, SUMO (Simulation of Urban MObility): An open-source traffic simulation. 2002.

4. Wade Gendersa, S.R., Using a Deep Reinforcement Learning Agent for Traffic Signal Control. 2016.

5. Tong Wu, S.M., IEEE, Pan Zhou, Member, IEEE, Kai Liu, Member, IEEE, Yali Yuan, Member, IEEE, Xiumin and M. Wang, IEEE, Huawei Huang, Member, IEEE, Dapeng Oliver Wu, Fellow, IEEE, *Multi-Agent Deep Reinforcement Learning for Urban Traffic Light Control in Vehicular Networks.* 2020.

6. Géron, A., Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition. 2019.

7. Volodymyr Mnih, K.K., David Silve,r Alex Graves ,Ioannis Antonoglou ,Daan Wierstra ,Martin Riedmiller, *Playing Atari with Deep Reinforcement Learning.* 2013.

8. Ziyu Wang, T.S., Matteo Hessel, Hado van Hasselt, Marc Lanctot, Nando de Freitas, *Dueling Network Architectures for Deep Reinforcement Learning.* 2015.

9. Greg Brockman, V.C., Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba, *OpenAI Gym.* 2016.

10. Yilun Lin, X.D., Li Li, and Fei-Yue Wang, An Efficient Deep Reinforcement Learning Model for Urban Traffic Control. 2018.

11. Matt Stevens, C.Y., Reinforcement Learning for Traffic Optimization. 2018.