Università degli Studi di Milano Bicocca

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di Laurea Magistrale in Informatica

Deep Q-learning
From a single agent to a multi-agent perspective for Traffic Signal Control

Relatore: Prof. Giuseppe Vizzari

Co-relatore: Thomas Cecconello

Tesi di Laurea Magistrale di:

Romain Michelucci

Matricola 867644

Anno Accademico 2020-2021

Abstract

In an increasing number of urban areas worldwide, traffic congestion management is an important and still open challenge. The traditional Traffic Light Control System (TLCS) is no longer an adequate solution to this issue. The combination of Reinforcement Learning (RL) and Deep Neural Network (DNN) resulting in the Deep Reinforcement Learning (DRL) framework has shown promising results in solving complex and high-dimensional problems, including traffic congestion. In the context of the second year master thesis, this report describes, first, a DRL approach to adaptive traffic lights management on a four-way intersection [1]. From this previous work, improvements to the model have been carried out and are discussed. Finally, a multi-agent perspective is introduced where two intersections are placed end-to-end with one agent regulating each TLCS.

Contents

List of Figures

List of Tables

1 Introduction

The total number of passenger kilometers (pkm) is the unit of measurement representing the transport of one passenger by a defined mode of transport (road, rail, air, sea, etc...) over one kilometer. Knowing that this unit continuously increases and that its vast majority is covered by passenger cars, it ensures that the efficient management of traffic flow is necessary. Population growth allied with an explosive increase in the number of vehicles on the streets and the limited road network expansion are factors leading to road transport maintaining its dominant role in the transport activity. Therefore the exceeded capacity of some of the actual infrastructure brings to the surface a real problem that every city is facing nowadays: *traffic congestion*.

To address this issue, the most plausible solution seems to operate on the existing road configuration by improving the systems that impact directly the traffic flow. Working with the *traffic signal controllers* currently equipped is a preferable solution rather than expanding the transportation capacity and building higher quality road systems. It requires low cost in terms of time and resources needed, without penalizing the traffic during the process.

Traffic Signal Control problem.

The Traffic Signal Control (TSC) problem has become even more important as traffic congestion has been more intractable. This problem seeks an efficient schedule for traffic signal settings at intersections. The challenge in TSC is to find an optimal traffic signal configuration that maximizes the traffic flow in a network or graph of intersections. In other words, the explicit objective of solving this problem is to determine optimal phase sequences and durations for each phase. It can be solved by optimizing various performance criteria, such as minimizing the average vehicle delay or maximizing the throughput of the network.

This thesis focuses on the development of an artificial intelligent technique able to control traffic light systems efficiently with the overall objective to improve the traffic flow on a larger scale.

Deep Reinforcement Learning for TSC.

The first promising techniques in the AI domain have been developed using the Reinforcement Learning (RL) technique because of its ability to learn the complex dynamics of the problem without any human knowledge or intervention. But the main issue of these proposed approaches lies in the curse of dimensionality that comes from the exponential growth of the action and state spaces.

Allied with deep neural networks, RL is able to better handle this issue. This kind of method entitled Deep Reinforcement Learning (DRL) is considered as one of the most successful AI models and the closest machine learning paradigm to human learning and enhances the performance of RL for large scale problems.

This master's thesis work takes its roots in a deep reinforcement learning agent controlling a set of traffic lights frameworked by Andrea Vidali [1].

Thesis goal.

The problem addressed in the baseline work is defined as follows:

> *"Given the state of an intersection, what is the traffic light phase that the agent should choose in order to optimize the traffic efficiency (and therefore, reduce traffic congestion) ?"*

The design structure is a Deep Q-learning agent which combines the Deep Q-networks and Q-learning AI techniques that will be comprehensively detailed. Thus, starting from this implementation, the interrogations that are raised are both:

1. How can the model be improved to make the traffic flow even more ?

2. How would it behave by scaling it to a wider intersection graph ?

To answer the first one, improvements to the model and bringing more realism to the traffic simulations is carried out.

Answering the second question is also be an important part of this thesis work since the TSC problem can not only concern a unique traffic light controller. In the case of a network with multiple intersections, a Multi-Agent Reinforcement Learning (MARL) approach needs to be developed. In TSC, the overall goal of the agents is to cooperate in order to help the traffic flows. There are two main issues with high-dimensional systems in multi-agent DRL in terms of state and actions: *stability* and *adaptation of agents to the environment*. When each agent optimizes its action without considering close agents, the optimal learning for the system would eventually become non-stationary.

But since the robustness of the agent framework has to be proven in order to integrate it into a wider network of intersections. That is why, the passage to a multi-agent perspective consisting in a minimalist multi-agent approach (with only two agents) is introduced in a *decentralized* way. These two independent agents are placed to control one intersection each. All agents learn individually and their communication limitations are defined by the environment. The need to better understand the agents behaviors in this two-intersections simulation has motivated multiple experiments with the overall objective to observe the agents reaching a form of collaboration. Therefore, cooperative learning will be our focus and especially, it will be analysed through the Independent Deep Q-Networks (IDQN) technique.

The analysis of the results (i.e. evaluated performances of the agents) is mostly *descriptive* (as opposed as *prescriptive*).

The problem addressed in this thesis can be stated as follows: **How the Deep Q-learning framework behave starting from a single agent to a minimalist multi-agent perspective for Traffic Signal Control ?**

Thesis structure

- Section 2 describes the state-of-the-art where an overview of the deep reinforcement learning framework is followed by the analysis of the different settings in the field of traffic signal control. Then, an analytical approach of some of the recent work on this context is carried out.

- Section 3 is about the introduction of Andrea Vidali's baseline work, starting from the agent definition to the learning algorithm implemented.

- Section 4 details the improvements provided to the baseline model in a one-intersection simulation which are, then, evaluated in Section 5.

- In a similar manner, Section 7 presents and discuss the performances evaluated during the experiments as part of the two-intersections simulation presented in Section 6.

- Finally, Section 8 aims at highlighting possible future investigations before closing this master's thesis with a conclusion in Section 9.

Code source

The code relative to this thesis, as well as the agents' models evaluated in Section 5 and 7 are fully available on GitHub[a] and a virtualized Docker environment has also been made available [b].

[a] https://github.com/GameDisplayer/Deep-QLearning-Multi-Agent-Perspective-for-Traffic-Signal-Control

[b] https://hub.docker.com/repository/docker/gamedisplayer/sumo-experiments

2 State of the art

In this chapter, an overview of related works on traffic control systems is delineated. Primarily, it will be introduced an overview of the theoretical background of traditional Reinforcement Learning (RL) and major deep RL algorithms, including a focus on the one that is at the center of the study: *Deep Q-learning*. This will be the starting point of the next subject, which is about the multiple design possibilities of Deep Reinforcement Learning (DRL) elements in the context of Traffic Signal Control (TSC) tasks. Finally, state-of-the-art DRL techniques for TSC will be described and discussed.

Table 1 presents the generic notations used in this section combined with the specific notations used in the RL models and algorithms presented.

Category	**Notation**	**Meaning**
General	RL	Reinforcement Learning
	DRL	Deep Reinforcement Learning
	AI	Artificial Intelligence
	(A)TLCS	(Adaptive) Traffic Light Control System
	(A-I)TSC	(Adaptive - Intelligent) Traffic Signal Control
	t	A single time instant $t = 1, 2, ...$ (or timestep)
RL	S	The set of all states s
	A	The set of all possible actions a of an agent
	a_t	Single action of an agent at time t
	s_t	State of the environment at time t
	r_t	Reward received by an agent at time t

Table 1: Notations state-of-the-art

2.1 Deep Reinforcement Learning: an overview

Deep RL is one of the most successful AI models that is the closest machine learning paradigm to human learning. It incorporates deep learning into the solution to take in very large inputs while being more efficient and stabilizing function approximations. And this subfield of machine learning takes its roots into the reinforcement learning framework which will be introduced in the very next part.

2.1.1 Reinforcement Learning

One of the goals of AI is the development of machines that resemble the *intelligent behavior* of a human being. In order to achieve this, it needs to interact with the environment and learn how to correctly act inside this environment. An established area of AI that has been proved capable of experience-driven autonomous learning is reinforcement learning. In other words, reinforcement learning is about an agent interacting with the environment, learning an optimal policy, by trial and error, for sequential decision making problems in a wide range of fields in both natural and social sciences, and engineering.

In a typical reinforcement learning problem, an autonomous agent observes the environment and gets a state s_t (which is the state of the environment at time t). The agent chooses an action a_t leading to a transition of the environment to the state s_{t+1}. Then, the agent obtains a reward r_{t+1}, which tells the agent how good a_t was, with respect of a performance measure. The standard cycle of reinforcement learning is shown in Figure 1.

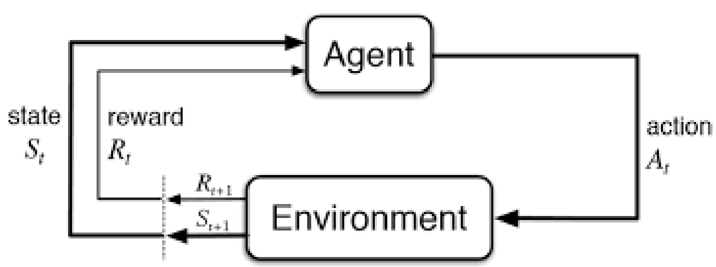


Figure 1: The agent-environment interaction in a Markov Decision Process. [2]

Markov Decision Process.

The mathematical formalism for this setting is called **Markov Decision Processes**.

Definition 2.1 (Markov Decision Process)**.** A Markov Decision Process (MDP) is a 5-tuple $\langle S, A, R, P, \rho_0 \rangle$ where:

- S is the set of all valid states,
- A is the set of all valid actions,
- $R : S \times A \times S \to \mathbb{R}$ is the reward function with $r_t = R(s_t, a_t, s_{t+1})$,
- $P : S \times A \to P(S)$ is the transition probability function, with $P(s' \mid s, a)$ being the probability of transitioning into state s' if s is the starting state and a the next action taken,
- ρ_0 is the starting state distribution.

The name Markov Decision Process refers to the system obeying the *Markov property* which states that transitions only depend on the most recent state and action, and no prior history i.e. given the state s_t, the next state s_{t+1} of the system is independent from the previous states $(s_0, s_1, ..., s_{t-1})$.
Informally, equation 1 expresses the fact that, at a given time step and knowing the current state (*present*), gathering information about the *past* will not give any additional information about the *future*.

$$P(\mathit{future} \mid \mathit{present},\ \mathit{past}) = P(\mathit{future} \mid \mathit{present}) \tag{1}$$

In transportation systems, MDP models are *episodic* in which the system has a starting and terminal point for each episode based on the end time T (or the end state s_t). It is opposed as *continuous* tasks that continue forever until the agent is stopped. And the goal of an MDP agent lies in finding the optimal policy (see definition 2.2), denoted as π^*, which specifies the correct action to perform in each state i.e maximizing the expected cumulative reward $\mathbb{E}[G_t | \ s, \pi]$ for each state s int the state space S and discounted cumulative reward (also called *return* 2.3).

Definition 2.2 (Policy function)**.** A **policy** is a rule used by an agent to decide what actions to take. It can either be deterministic or stochastic and is a mapping from state space to action.

$$\pi(s) : S \to A \tag{2}$$

Definition 2.3 (Return)**.** The **return** is defined as:

$$G_t = \sum_{i=0}^{T-1} \gamma^i \, . \, r_{t+i} \tag{3}$$

with $\gamma \in [0, 1]$ being the discount factor which reflects the importance of future rewards (by penalizing it). The larger γ is, the higher dependency on future reward the agent's actions will be.

Predominantly, an RL agent can act:

1. either by knowing or learning the transition probability P from the state s_t to s_{t+1}
2. or by exploring the environment without learning a transition model.

Agents which use a model are called **model-based** methods and the others are called **model-free**.

Figure 2 presents a non-exhaustive and simplified version of the taxonomy of RL models (which is convenient for this introduction, on the basis that our concern is the Q-learning algorithm). That is why the branch of the taxonomy of RL algorithms that will be explored in details is the model-free one.

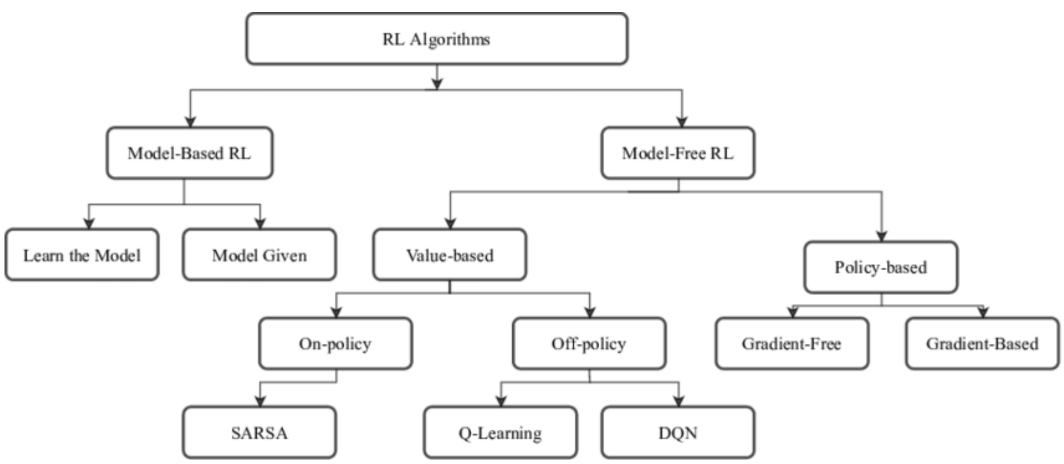


Figure 2: Taxonomy of Reinforcement Learning algorithms [3] (which was reproduced and shortened from [4])

And more specifically, while RL algorithms belonging to this class are divided into two main groups as **value-based** and **policy-based** methods, the one that is in our interest is the value-based model. Nevertheless, a slight description of policy-based RL algorithms will be given.

Value-based RL.

Based on *temporal difference* (TD) learning (Sutton, 1988 [5]), the agent at each iteration updates a value function $V^{\pi}(s)$ that determines how good a state is for the agent by estimating the value of being in a given state s under a policy π (as in equation 4).

$$V^{\pi}(s) = \mathbb{E}[G_t|\, s, \pi] \tag{4}$$

The optimum value function $V^{*}(s)$ which describes the maximized state value function over the policy for all states can be defined as:

$$V^{*}(s) = \max_{\pi} V^{\pi}(s), \forall s \in S \tag{5}$$

Bringing the effect of action, state-action pair value function is defined as the quality function also known as Q-function (definition 2.4).

Definition 2.4 (Q-function)**. Q-function** is used to reflect the expected return in a state-action pair as:

$$Q^{\pi}(s, a) = \mathbb{E}[G_t|s, a, \pi] \tag{6}$$

Figure 3 illustrates that the Q-function can be seen as any function that accepts a state and an action and returns the value (the expected rewards) of taking that specific action given that state.

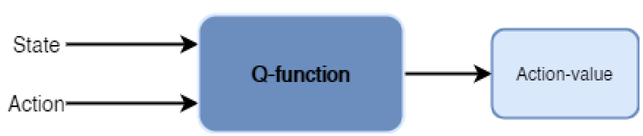


Figure 3: Q-function illustration

Similarly to the optimum state value function (equation 5), the optimum Q-function $Q^{*}(s)$ describing the maximized expected return over all states of S is given by:

$$Q^{*}(s) = \max_{a} Q^{\pi}(s, a), \forall s \in S \tag{7}$$

Therefore, optimum Q-function $Q^*(s, a)$ provides the optimum policy π^* by selecting the action a that maximizes the Q-value for the state s. It can be stated as:

$$\pi^*(s) = \arg\max_a Q^*(s, a), \forall s \in S \tag{8}$$

Based on the equations above, there are two main value-based RL algorithms: Q-learning [6] and SARSA [7]. The first one is classified as an *off-policy* algorithm while the second is an *on-policy* one (as specified in Figure 2).

Off policy vs. On-policy learning

Off-policy learning is when a policy is learned by the agent while collecting data using a different policy. In the case of on-policy algorithms, the policy is learned while also simultaneously using it to collect data for learning.

In both algorithms, Q-values are learned thanks to the Bellman equations combined with the Markov Property (equation 9).

$$Q^\pi(s_t, a_t) = \mathbb{E}[r_t + \gamma.Q^\pi(s_{t+1}), \pi(s_{t+1})] \tag{9}$$

And, in practice, the estimation of Q^π (equation 10) is updated with a learning rate α to improve it incrementally.

$$Q^\pi(s_t, a_t) \rightarrow Q^\pi(s_t, a_t) + \alpha(y_t - Q^\pi(s_t, a_t)) \tag{10}$$

where y_t is the temporal difference (TD) target for $Q^\pi(s_t, a_t)$.

And the difference between these TD-like methods are that, in Q-learning, actions of the agent are updated (with a greedy approach) by **maximizing Q-values over the action** (equation 11).

$$y_t = G_t^{(n)} + \gamma^n max_{a_{t+n}} Q^\pi(s_{t+n}, a_{t+n}) \tag{11}$$

Whereas, in SARSA, actions of the agent are updated **according to the policy π derived from the Q-function** (equation 12).

$$y_t = G_t^{(n)} + \gamma^n Q^\pi(s_{t+n}, a_{t+n}) \tag{12}$$

To follow the scope of this thesis, Q-learning, only, will be described.

Q-learning

Q-learning is a form of model-free reinforcement learning. It consists of assigning a value, called the *Q-value*, to an action taken from a precise state of the environment. Formally, it learns the action-value function thanks to the update rule written in equation 13. (This update rule is obtained by combining equation 10 and 11).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \,.\, max_A Q(s_{t+1}, a_t) - Q(s_t, a_t)] \quad (13)$$

where $Q(s_t, a_t)$ is the value of the action a_t taken from state s_t. The equation consists on updating the current Q-value with a quantity discounted by the learning rate α. The term r_{t+1} is the reward associated to taking action a_t from state s_t. $0 < \gamma < 1$ represents the discount factor and $\gamma \,.\, max_A\ Q(s_{t+1}, a_t)$ represents the highest possible accumulated reward expected to be received.

In other words, the Q-value at time t is updated to be the current predicted Q-value <u>plus</u> the amount of value expected in the future (given that the play is optimal from our current state).

Q-learning also refines the policy greedily with respect to action values by the max operator (max_A means that, among the possible actions a_t in state s_{t+1}, the most valuable one is selected).
Algorithm 1 presents the pseudo code for Q-learning.

Algorithm 1: Pseudocode of Q-learning

```
// Initialization
Initialize Q(s,a) arbitrarily
// Main loop
for each episode do
    Initialize state s
    for each step of episode where state s is not terminal do
        Choose action a for s derived by Q (using ϵ-greedy)
        Take a
        Observe r and s′
        Update Q(s,a) using the update rule (equation 13)
        Take s′ as s
    end
end
```

Policy-based RL.

These types of algorithms treat the policy π_θ as a probability distribution over state-action pairs which are parameterized by θ. These policy parameters are updated in order to maximize an *objective function* $J(\theta)$. And this objective function is often linked to the expected return as written down in equation 14.

$$\mathop{\mathbb{E}}_{\pi_\theta}[G_t \mid \theta] = \mathop{\mathbb{E}}_{\pi_\theta}[Q^{\pi_\theta}(s_t, a_t) \mid \theta] \tag{14}$$

The advantage of policy-based methods over value-based ones is that their performance is typically better on continuous control problems with an infinite-dimensional action space since evaluating the policy does not require to explore all states and storing them in a table.

As seen in Figure 2, two types of policy methods are defined but algorithms which are the most used and effective in RL problems are the *gradient-based* ones. In this regard, REINFORCE (introduced by Williams in 1992 [8]) is the most well-known policy gradient algorithm. In brief, the objective function is the expected return (equation 14) and the algorithm uses the stochastic gradient descent technique to approximate the gradient. This gradient is then employed to update the parameters θ.

2.1.2 Deep Reinforcement Learning

In many practical decision making problems, the states of the MDP are high-dimensional (for instance, the raw sensor stream from a robot or images from a camera) and cannot be solved by traditional RL algorithms. Deep RL algorithms incorporate deep neural networks to solve such MDPs.

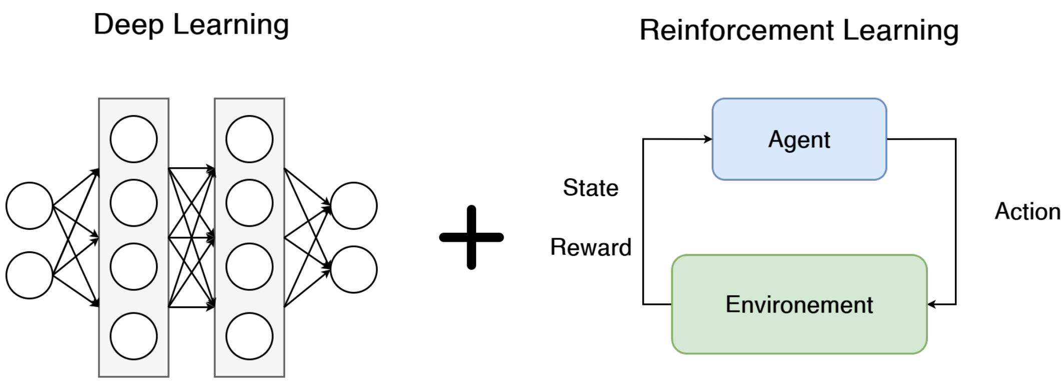


Figure 4: Deep Reinforcement Learning

Deep Q-Network.

Value-based RL algorithms, theoretically, learn the Q-function by populating a Q-table (for all state-action pairs). However, it becomes quickly computationally infeasible in a large state space and continuous action problems. In this respect, the need for **function approximations** arose. The leading approach to this problem is called Deep Q-Network (DQN) [9]. The goal is to approximate the Q-function with deep neural networks. Therefore, the Q-function approximation is labelled as $Q(s, a; \eta)$, denoting the neural network η from which its output is the best action selected according to equation 8.

But deep Q-learning may suffer from instability and divergence when combined with a nonlinear Q-value function approximation. And this major contribution of Mnih et al. [9] aimed at stabilizing learning with two novel techniques:

- **Target Network**: DQN has two separate networks. The *main network* approximates the Q-function and the *target network* gives the Temporal-

Difference (TD) target for updating the main network. During the training phase, the target network parameters are updated after a certain period of steps C (where C is an hyperparameter to choose) while the main network parameters are updated after every action. The reason is that it helps prevent the short-term oscillations that would result from the exaggerated change due to the feedback loop between the main and the target network. This would exacerbate the unstable learning.

- **Experience Replay** (initially introduced in [10]): Recent experiences are stored in a replay memory. During Q-learning updates, samples are drawn randomly from the replay memory for training the neural network. The benefit is that it removes correlations in the observation sequences (since one sample could be used multiple times) and smooths over changes in the data. Moreover, instead of learning over full observations, the agent can learn over mini-batches that increases the efficiency of the training. The fixed-size memory stores only recent samples by removing the oldest experience.

Experience Replay in details.

As this technique will be adopted in this thesis, it needs further attention. Its goal is to improve the performances of the agent while feeding the Neural Network with experiences. The information for learning is submitted in the form of randomized group of samples called *batch*. This batch is taken from the memory which stores every sample collected during the training phase.
A sample m is defined as a quadruple (equation 15).

$$m = \{s_t, a_t, r_{t+1}, s_{t+1}\} \tag{15}$$

This method belongs to the set of *Offline Learning* methods where the agent is not operating with the information taken in real time. Experience Replay is well-know to mitigate the problem of catastrophic forgetting which is common with online training algorithms.

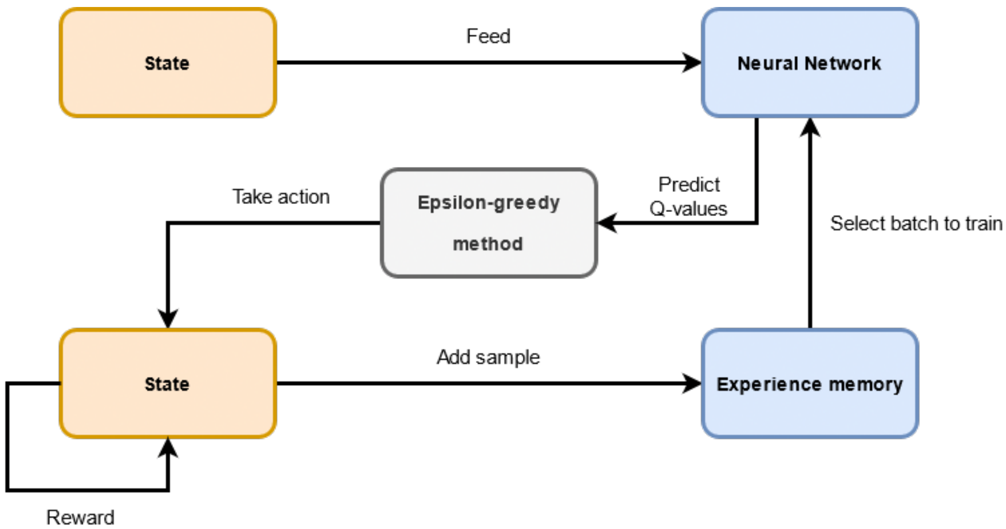


Figure 5: General overview of Experience Replay

The following points descries how experience replay works (figure 5):

1. In state s_t, the agent takes action a_t and observes a new state s_{t+1} while receiving the reward r_{t+1}.
2. The sample $m = (s_t, a_t, s_{t+1}, r_{t+1})$ is stored in memory.
3. The memory buffer is filled until a specific length.
4. A randomly selected batch is extracted.
5. The Q-Network is trained while iterating on the subset of samples chosen.
6. Then, the NN predicts Q-values leading to a new action a_{t+1}. This loops until the last timestep.

> Old experiences are overwritten in the experience replay memory.

There are many extensions of DQN in the literature to improve the original design such as *Prioritized Experience Replay* [11], which consists in giving more importance to some experiences than others by changing the sampling distribution of the algorithm or *DQN with dueling architecture* [12] which estimates state-value function (denoted as $V(s)$) and advantage function denoted as $A(s, a)$ with a shared network. The idea behind this improved version of the standard Q-learning algorithm with a single estimator is that both DQN and Q-learning overestimate some actions due to having single Q-function estimations.

Deep Q-learning

In Deep Q-learning, a neural network is used to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output. Figure 6 aims at illustrating the comparison between Q-learning and deep Q-learning.

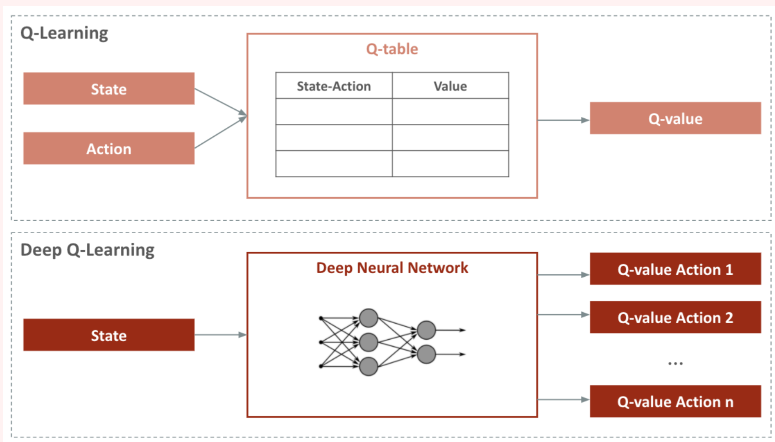


Figure 6: Q-learning vs. Deep Q-learning taken from [13] (which is inspired from [14])

Figure 7 has the objective to summarize the deep Q-learning overall process *without experience replay* as it will be presented in Section 5.

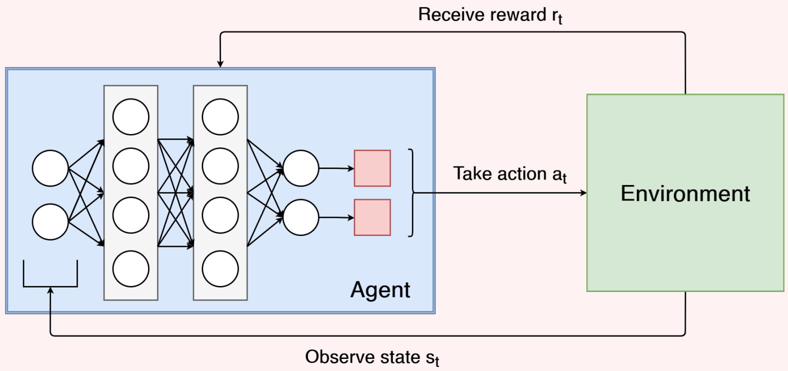


Figure 7: Deep Q-learning

Clarifications are required to better appreciate the understanding of the overall process:

1. All the past experiences are stored in memory.

2. The next action is determined by the maximum output of the Q-network.

3. The problem faced is a **regression** one.
   So, given the update rule (equation 13), the loss of the network is the following loss function (equation 16)

$$loss = [r_{t+1} + \gamma \, . \, \max_{A} Q(s_{t+1}, a) - Q(s_t, a_t)]^2 \tag{16}$$

   where $r_{t+1} + \gamma \, . \, \max_A Q(s_{t+1}, a_t)$ is the *target* and $Q(s_t, a_t)$ is the *prediction*.
   The network is, then, updated using *backpropagation* to eventually converge [15].

Q vs. policy network.

To put in perspective what was presented in the previous part, in the case of deep Q-networks, the output of the Q-network was Q-values corresponding to each action for a given state (Figure 8).

Figure 8: Q-network

And given the predicted Q-values from the Q-network, some strategy is applied to select actions to perform.

In the case of policy function using neural networks, the *selection of a policy on top of the DQN* part is skipped and the trained neural network outputs an action directly. This network is called a *policy network*. It will return a probability distribution over the actions and then, the selection of an action can be sampled directly from the outputs. Figure 9 simplifies the process.

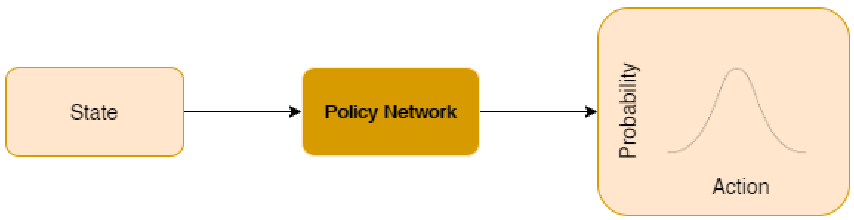


Figure 9: Policy network

2.1.3 Multi-agent Reinforcement Learning

Many real world problems require the interaction of multiple agents to maximize the learning performance. Reaching a globally optimum solution when considering other agent's actions is a challenging task. This is even more the case, when, for scalability, increasing the number of agents also makes the state-action dimensions soar. The first main issue with high-dimensional systems is the **stability**. When each agent optimizes its action without considering close agents, the optimal learning for the overall system would eventually become *non-stationary*. This central problem in Multiagent Systems (MAS) also leads to the (in)adaptation of agents into the environment. There are several approaches to address this, such as distributed learning, cooperative learning and competitive learning [16].

The one that will be investigated in this thesis is related to *independent learners* paradigm. But, first, some formalism needs to be introduced coming from game theory.

Stochastic Game.

The generalization of the MDP to the multi-agent case is called the *stochastic game*. In this thesis, there is a focus on 2-player stochastic games. So, in order to be more specific to the future study case, a 2-agent stochastic game definition is provided.

Definition 2.5 (Stochastic Game)**.** A 2-agent stochastic game (SG) is a 6-tuple $\langle S, A^1, A^2, r^1, r^2, P\rangle$ where:

- S is the discrete state space,
- A^g is the discrete action space of agent g where $g = 1, 2$,
- $r^g : S \times A^1 \times A^2 \times \rightarrow \mathbb{R}$ is the reward or payoff function for agent g,
- $P : S \times A^1 \times A^2 \rightarrow P(S)$ is the transition probability function, where $P(S)$ is the set of probability distribution over state space S

To have a closer look at a stochastic game, consider a process observable at discrete timesteps and controlled by two agents referred to as agent 1 and agent 2. In state s, each agent independently chooses actions $a^1 \in A^1$ and $a^2 \in A^2$ and receives rewards $r^1(s, a^1, a^2)$ and $r^2(s, a^1, a^2)$, respectively. When the addition of both rewards equals 0, the game is called *zero sum*. When the sum is not restricted to 0 or any constant, the game is called a *general-sum* game.

In this game the objective of each player is to maximize a discounted sum of rewards. Let π^1 and π^2 be the strategies of agents 1 and 2. For a given initial state s, both agents receive the following values for the game:

$$v^1(s, \pi^1, \pi^2) \tag{17}$$

$$v^2(s, \pi^1, \pi^2) \tag{18}$$

A risk when setting individual reward in this kind of multi-agent systems is that it might not capture the overall system optimum. Basically, each agent makes the right choice, but due to choices and actions of other agents it can still lead to a negative reward. In stochastic games, this situation is called **Nash equilibrium** (definition 2.6).

Definition 2.6 (Nash equilibrium)**.** A Nash equilibrium point is a pair of strategies (π^1_*, π^2_*) such that for all state $s \in S$

$$v^1(s, \pi^1_*, \pi^2_*) \geq v^1(s, \pi^1, \pi^2_*), \ \ \forall \pi^1 \in \Pi^1 \tag{19}$$

and

$$v^2(s, \pi^1_*, \pi^2_*) \geq v^2(s, \pi^1_*, \pi^2), \ \ \forall \pi^2 \in \Pi^2 \tag{20}$$

Each player is assumed to know the equilibrium strategies of the other players and no player has anything to gain by changing only their own strategy. In addition this definition is set , assuming that the players have complete information about the payoff functions of both players (which is nearly never the case in a real-word multi-agent system). Therefore, this kind of situation is quite relevant to the Traffic Signal Control problem and is going to be addressed with a well known technique.

Independent DQN.

Independent Q-learning (IQL) was proposed in [17] to train agents independently while each agent can learn its own policy by treating other agents as part of the environment. Each agents learns its own Q-function that only conditions the actual state of the environment and its own action. On the one hand, IQL overcomes the scalability problems of centralized learning. On the other hand, it introduces the problem of the environment becoming *non-stationary* from the point of view of individual agents. And because it involves other agents learning at the same time, it may rule out any convergence guarantee [18].

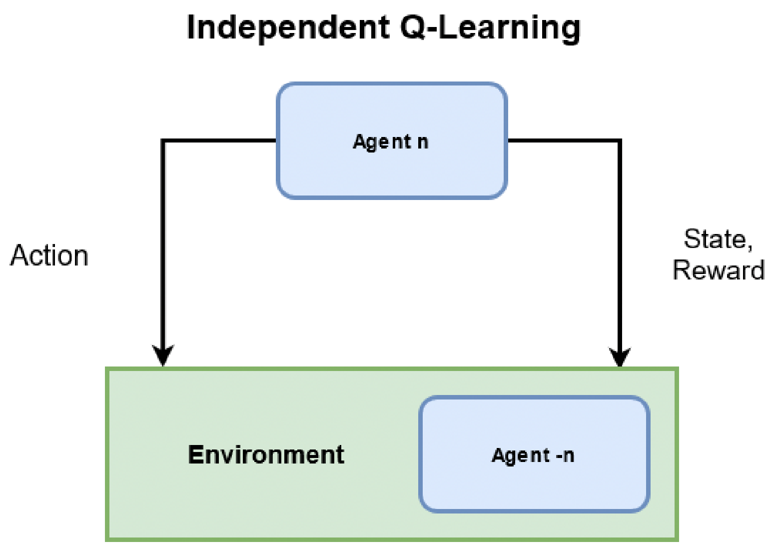


Figure 10: Diagram of Independent Q-Learning

Figure 10 illustrates that in independent Q-learning, an agent (agent n) does not perceive the actions of other agents (which are defined as agent $-n$). All of these are considered as being part of the environment.

Independent DQN (IDQN) is an extension of IQL with DQN for cooperative environment. Each agent g observes a partial state s_t^g, selects an individual action a_t^g and receives a team reward r_t which is shared among all agents. [19] combines DQN with independent Q-learning where each agent independently and simultaneously learns its own Q-function. In DRL, this can be implemented in a *decentralized* way by giving to each agent a DQN on its own observations and actions learning individually. However, it also exists *centralized* learning where there is one-only brain deployed across the agents.

2.2 Deep Reinforcement Learning settings for TSC

Up to this point, the importance of AI and theoretical background of RL, in particular deep RL have been highlighted. The aim of this section is to focus on a particular area of interest of deep RL applied to ITS which is the *control of signalized intersections.*

In this context, one or more autonomous agents have the goal of maximizing the efficiency of traffic flow that drives through one or more intersection controlled by traffic lights. Therefore, the most widely used approaches to design deep RL components (respectively, state, action, reward, and neural network) are reported. Figure 11 brings together various aspects of the traffic signal control problem.

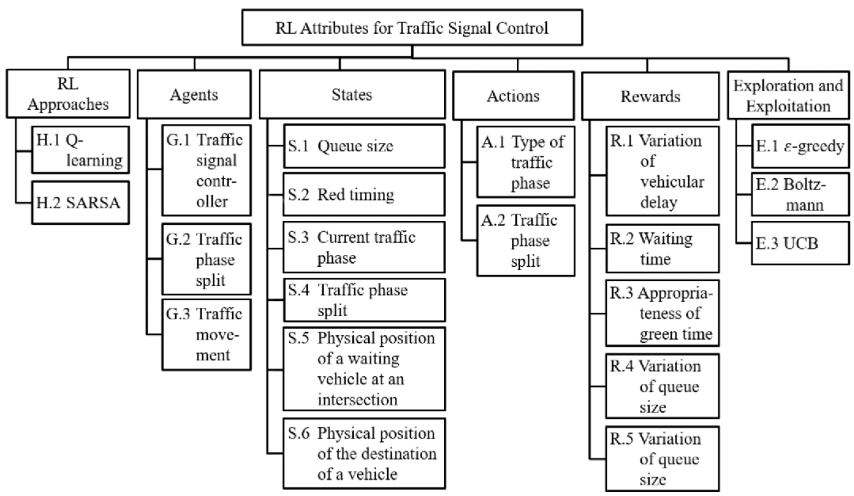


Figure 11: Traffic signal control attributes [20]

2.2.1 State

State definition

The state is the agent's perception of the environment in an arbitrary timestep.

And because the learning performance of an agent is highly dependent of the state representation, the latter varies in RL applications on traffic lights. Yau et al. [20] details in their survey six main representations for a state s:

- *Queue size*: the number of vehicles waiting in a lane.
- *Red timing*: the time elapsed since the signal of a lane turned into red.

- *Current traffic phase*: the information about the current traffic phases in a wider network.
- *Traffic phase split*: the time interval allocated for a traffic phase.
- *Physical position of a waiting vehicle at an intersection*: information about vehicles on discretized lanes (each lane is segmented into small cells; each can accommodate a vehicle).
- *Physical position of the destination of a vehicle*: in a large network, it corresponds to the edge nodes through which the vehicles goes by.

But the differences in the choice of state space representations in the literature can also be viewed by the *level of information density*.

- **Low information density.** The lanes of the intersection are discretized in cells along the length of the lane. These cells are then mapped to cells of a vector which contains binary information: 1 if a vehicle is inside the lane cell, 0 otherwise [21]. The authors add that the occupancy of a lane and average speed can also be used while they have the advantage to be easier to collect since just a simple detector is necessary.
- **Medium information density.** More data is collected in this instance. One of the most widely adopted state representation of this subclass is composed by the vector of presence cells in addition with a vector encoding the relative velocity of vehicles [22]. A third vector could be added taking into account the current traffic light phase [23].
- **High information density.** This case relies on the works that take an image of the current situation of the whole intersection i.e. a snapshot of the simulator used. The process operates by stacking multiple snapshots together to give the agent a sense of the vehicle motion [24]. Furthermore, some approaches relies on every information that the intersection could provide. As an instance, the state component introduced in Hua Wei et al [25] is composed of the queue length, the number of vehicles, the updated waiting time, in addition with, the image representation of the vehicle's position, the current phase and the next phase.

2.2.2 Action

Action definition

An agent's action is defined as the interaction of the agent within the environment.

In the context of traffic signal control, the agent's actions are implemented with multiple degrees of flexibility.

- **Low flexibility.** The agent has usually a defined set of light combination from which he can choose one. Then, when an action is selected, a fixed amount of time will pass before the agent can select a new action [22].

- **Medium flexibility.** In this case, timings could be flexible. If the selected action is the same as the previous one, the current traffic light will last longer [26].

- **High flexibility.** The agent chooses an action at every time step of the simulation. However, if the minimum amount of time required to release at least a vehicle has not passed, the selected action is not activated [23] [24].

Two main representations for an action can be described [20]:

1. *Type of traffic phase*: The selection of a combination of signals can be *in-order* in which the traffic phases are activated in a round-robin manner, or, in an opposite way, *out-of-order*.

2. *Traffic phase split*: The action, this time, acts on the time interval to be allocated for a traffic phase at an intersection. 2 different approaches emerges in this case. First, the action represents the traffic phase split for one of the lanes of the intersection. Second, the action plays on the choice of either keeping the current traffic phase or switching to another traffic phase.

2.2.3 Reward

Reward definition

The reward is used by the agent to understand the consequence of the latest action taken in the latest state.

It is widely defined as a function of some performance indicator of the intersection efficiency. In other words, reward is always a scalar value which is a function of the traffic data. There are four main indicators that can stand out from the crowd [20].

1. *Waiting time*: an agent is penalized when the average waiting time of the vehicles at an intersection increases.

2. *Appropriateness of green time*: an agent can be penalized when the green light timing is inappropriate, resulting in congestion or green idling.

3. *Queue size*: an agent is rewarded when the queue size reduces. It can either be based on the difference of the number of vehicles crossing an intersection at time t and $t+1$, or the difference between the number of vehicles crossing an intersection and the queue size.

4. *Vehicle delay*: When the vehicle delay of lane reduces, the agent is rewarded.

2.2.4 Neural Network Structure

As described in Section Deep Reinforcement Learning, in many practical decision making problems, the states of the MDP are high-dimensional and cannot be solved by traditional RL algorithms. Deep reinforcement learning algorithms incorporate deep learning to solve such MDPs where the policy becomes a learned function as a neural network. That is why the structure of deep neural networks has a high impact on the agent's learning capability. Thus, different neural network structures are proposed in the literature.

- *Standard fully connected neural network models* (FCN): It is a useful tool for classic data classification and has the advantage of being faster to train. It is one of the most used in TSC applications [1].

- *Convolutional neural network* (CNN): It is an extension of multi-layer perceptron (MP) with kernel filters. It provides high performance on mapping image to a desired output. In many TSC applications, CNNs uses the pixel frames as a state definition [22] [23] [25].

- *Recurrent neural networks* (RNN) with Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU): These neural networks are designed to work with sequential data which is the case in TSC controlling [27] [28].

- *Autoencoder*: It learns an encoding for high-dimensional input data into a lower-dimensional subspace. It is commonly used for clearing the noise out of the input data [26].

2.3 Deep RL Applications for TCS

In recent years, traffic signal control applications have been discussed and described in detail in comprehensive surveys. Ammar et al. [29] authors have been focused on the latest deep reinforcement learning based traffic control applications. Wei et al. [30] covers more widely acknowledged transportation approaches and gives a comprehensive list of recent literature on reinforcement learning for traffic signal control. Yau et al. [20] details the different motivations of using RL for traffic signal control. And this work also highlights the variety of attributes of the traffic signal control problem which are the challenges of inappropriate traffic phase sequence or phase split, the traffic network models, the traffic signal control models and the different performance measures.

2.3.1 Standard Reinforcement Learning applications for traffic signal control

This section focuses on RL studies for ATSC where the learning models are classified into two sub-groups:

- *single agent RL* which concentrates on evaluating one agent for the TSC network
- *multi-agent RL* which implements multiple agents to eventually find an optimal policy

Single Agent.

Different RL algorithms are tested in different state, action and reward settings on a single intersection.

In 1996, Thorpe et al. [31] work pioneered the traffic signal control with RL-based machine learning where the model-free SARSA (initially proposed as "Modified Connectionist Q-Learning") algorithm [7]. It is applied to a simulated traffic light control problem and compared with a fixed-duration strategy and a simple, rule-based strategy. The performance of SARSA is analyzed with three different traffic state representation: the *vehicle count* representation, which corresponds to the sum of the number of vehicles in the incoming lanes, the *fixed distance* representation which is an indication of the relative distance of vehicles from the intersection and the *variable distance* which is a partitioned version of the fixed distance representation.

After this initial research, another SARSA-based ATSC method for a single intersection is introduced by Wen et al [32], in 2007, with a stochastic control mechanism including more realistic traffic situations. The novelty lies into the traffic dynamic model taking the random nature of the flow turning fraction and the lane assignment into account.

And more recently, in 2021, differential semi-gradient n-step SARSA algorithm (which unifies Monte Carlo method and SARSA) was applied in Kekuda et al. [33] where an optimal policy that effectively minimizes the probability of congestion in the network is highlighted.

Model-free Q-learning algorithm for a single intersection has also been proposed in multiple works. Abudhlai et al [34], suggested an introduction to Q-learning followed with a case study involving application to traffic signal control. The results of this work are compared with a static traffic light controller in different traffic flow patterns expressed in terms of average delay per vehicle ratio.

In 2003, Camponogara and Kraus Jr. [35] proposed a similar RL model based on a distributed Q-learning approach presented as a «distributed, stochastic game »that was evaluated on two intersections by assigning separate Q-values for each individual agent.

In 2017, authors of Toubhi et al. [36] work focused on assessing three reward definitions: queue length, cumulative delay and throughput applied with Q-learning on a single intersection.

Multi-agent.

Single agent RL can be a good solution up to a certain point, however large intersection networks suffer from this approach.

The first multi-agent papers proposed model-based RL algorithms that form a transition probability distribution as in Wiering [37], Steingrover et al [38] and Isa et al. [39]. In this context, three different algorithms based on the coordination between vehicles and intersections were initially introduced. The state representation was formed with the position and destination of the vehicles at each intersection (which is not realistic for a real world appliance). Then, other works extended the Wiering's approach in various perspectives: Steingrover's by enhancing the state representation i.e. including congestion information on other intersections and Isa's by incorporating congestion and accident information in the state representation. Other works also used Wiering's approach as a benchmark.

Some years later, *multi-objectivity* gained popularity in RL thanks to its capabilities in complex environments. In addition, before deep Q-learning was putted forward, function approximators were popular for Q-functions with large state space. In [40] and later in [41], Prashanth proposed two RL models for TSC using function approximation based Q-learning. The paper tackles this problem of the curse of dimensionality by effectively using feature-based state representations that use a broad characterization of the level of congestion as low, medium, or high. The algorithms developed use full-state representation and incorporates *prioritization of traffic* (it provides a means to control the volume of traffic being sent in a specified period).

An interesting to mention approach is from Araghi et al. [42]. They present a distributed Q-learning controller model designed to adjust the green time for traffic signals by the aim of reducing the vehicles' travel delay time in a multi-intersection network. Each controller adjusts its own intersection's congestion while attempting to reduce the travel delay time in the whole traffic network.

2.3.2 Deep Reinforcement Learning applications for traffic signal control

In 2013, Mnih et al. released a deep version of Q-learning, they termed deep Q-networks (DQN), that showed impressive learning results to play Atari video games [43]. Since then, the development of deep RL techniques for adaptive intersection control arises.

Single Agent.

The earliest work of a full DQN algorithm is realised by Wade Genders and Saiedeh Razavi [23]. In this paper, authors make use of discrete traffic state representation based on detailed information from the traffic environment (DSTE) to produce an image-like state representation. This state is inputted into a Convolutional Neural Network (CNN) to learn features and approximate the Q-values of discrete actions. It is compared with Q-learning using a Shallow Neural Network (which consists of only 1 hidden layer).

A year later in 2017, Gao et al. [22] proposed a new Neural Network architecture in which state is a combination of the position and speed of vehicles based again on DTSE. The performances are evaluated against different traffic lights policies, such as long-queue-first and fixed-times.

Mousavi, Shcukat and Howley [24] analyzed a double approach to address the traffic signal control problem. The first one is *value-based* i.e. action values are predicted by minimizing the mean-squared error of Q-values with the stochastic gradient-descent method. The second one is *policy-based*. In this alternative approach, the policy is learned by updating the policy parameters in such a way that the probability of taking good actions increases. A CNN is used as a function approximator to extract features from the image of the intersection. In the value-based approach the output is the value of the actions while in the policy-based approach it is a probability distribution over actions. Results obtained show that both approaches achieve good performance against a defined baseline without suffering from instability issues.

In 2018, Zhang et al. [44] worked at evaluating a deep Q-learning model with experience replay for traffic control with partial detection of vehicles. Their paper is of particular interest in analysing how DRL manages «Partially Detected Intelligent Transportation Systems »(PDITS) . They raise the theoretical problem of assuming that every vehicle is detected which is not the case of transportation systems such as Dedicated Short Range Communications (DSRC) technology. The numerical results are compared on sparse, medium, and dense arrival rates.

Multi-agent.

The first DRL-based multiple intersection control mechanism is proposed by Van der Pool et al. [45]. The authors introduce a new reward function and suggests the combination of DQL algorithm with a coordination algorithm for controlling a wider number of traffic lights. To be more concise, for coordination of multiple intersections to have a high traffic flow rate, this paper uses a transfer planning technique for a smaller set of intersections and links the learning results to a larger set of intersections with the max-plus coordination algorithm.

Independent Deep Q-learning (IDQL) is one of the most popular multi-agent RL approaches. Liu et al [46] introduced a cooperative DRL model for controlling multiple intersections with multiple agents. They make use of DQN with a ResNet structure to form the state space. It is a variant of IDQL where multiple agents can be applied experience replay to speed up the training process. Besides, the design of a new reward considering the driver patience and cumulative delay is trying to better reflect the reward from the road network. This work shows how cooperation between agents is assured by sharing the policy with other agents. The experiments are done using a 2-by-2 intersection model.

> In fact, multiple traffic intersections can be represented as a network graph and it is widely studied in this way in the literature.

In [47], an agent with an independent double dueling DQN model with prioritized experience replay is assigned to three connected heterogeneous junctions in low and high traffic conditions. The results show that the proposed algorithm outperforms the fixed-time controller while investigating this cooperative multi-agent deep RL model.

Chu et al [48] also recently implemented a comparison between different independent deep learning paradigms on a wider 5-by-5 grid and, even, the Monaco city map [49] !

3 Problem statement and background

The improvement in traffic flow driving through an intersection (or multiple ones) controlled by traffic lights using artificial intelligent techniques is a problem tackled by numerous researches. And this will be the starting point of this thesis, which is about improving, understanding and going beyond the baseline model proposed by Andrea Vidali [1].

This section aims at presenting the problem statement and the design of the agent made in Vidali's thesis [50]. The goal was to develop an entire *Traffic Light Control System* that comprehends the agent, its elements and the learning techniques. In order to present in the next chapter the improvements built upon in this work, it is necessary, beforehand, to present, explain and describe the context and the formal problem that this thesis is facing.

In Table 2 are listed the terms used in this chapter to describe the general, traffic and more specific reinforcement learning elements.

Category	**Notation**	**Meaning**
General	h	Episode number
	H	Total number of episodes
Traffic	TL	Set of traffic lights
	NSA	North-South Advance
	NSWA	North-South Left Advance
	EWA	East-West Advance
	EWLA	East-West Left Advance
	twt	Total waiting time
	wt	Waiting time
	veh	Vehicle
RL	IDR	Intersection Discretized Representation
	NSR	New State Representation
	B	Batch
	m	Sample contained in B

Table 2: Notations Section 2

3.1 Problem statement: Traffic Signal Control

RL techniques are well suited for the traffic signal control task. It can be illustrated by one or more autonomous agents having the goal of maximizing the effectiveness of traffic flow. This traffic flow is represented by a certain amount of vehicles driving through one or more intersections which are controlled by traffic lights at discrete time steps t. An agent observes a state $s_t \in S$, then it chooses and actuates an action $a_t \in A$ which corresponds to one of the configurations the traffic signal can take (color phases). After the action a_t, the agent transitions to the next state $s_{t+1} \in S$ and receives a reward r_t. A deep Q-learning neural network is the learning mechanism of the agent. Figure 12 illustrates how deep Q-learning can be applied to TLCS (which is a reference of Figure 7).

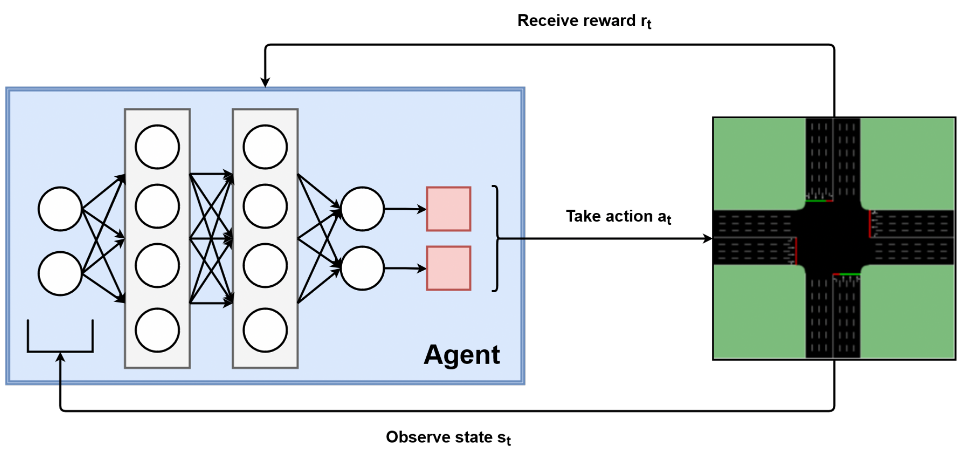


Figure 12: Overall process of the Deep Q-Learning implementation on TLCS

Given the state of an intersection, the agent should choose the traffic light phase that enables a better flow of vehicles. The minimization of the negative reward is linked with the optimization of the traffic efficiency.
In Vidali's thesis the problem faced can be stated as: «Given the state of the intersection, what is the traffic light phase that the agent should choose, selected from a fixed set of predefined actions, in order to maximize the reward and optimize the traffic efficiency of the intersection ? »

Starting from this one-intersection simulation, one of the thesis aim concerns the improvement of the model. Then, taking into account the limitations of the model, the second objective is to propose a new approach within a minimalist multi-agent perspective.

Therefore, the questions raised are how multiple agents would operate on an environment with multiple intersections ? What level of information need a single agent to collaborate with the others to improve the traffic efficiency ?

To provide answers to these interrogations, a minimalist multi-agent approach will be considered where two agents taking the control of one TLCS each, will be the starting point of the experiments (Figure 13).

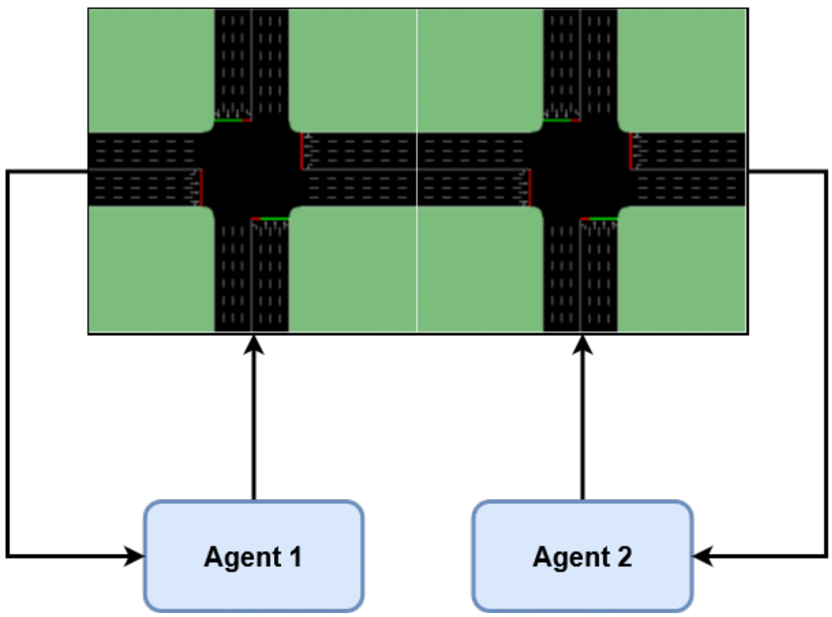


Figure 13: Minimalist multi-agent approach with 2 agents

> It should be noted that the possibility of a real-world appliance of the environment has been taken into consideration, so that the agents do not used any elements that could be hard to implement with the currently available technology.

3.2 Baseline model

A description of the micro-simulator, the environment simulated and the agent internal mechanism will be provided in order to develop, in the next chapter, the tests and the improvements executed.

3.2.1 Traffic micro-simulator

First and foremost, a description of the traffic simulation package utilized must be provided.

SUMO [51] (**S**imulation of **U**rban **MO**bility) is the traffic micro-simulator chosen. It is a free and open source software package that enables modelling complex traffic systems including road vehicles, public transport and pedestrians.
Among the wide range of packages that SUMO offers, the ones used in this work are described here.

- *NetEdit* is the visual editor that helps to design the static elements of the environment, i.e. the edges, the road connections across the intersection, the traffic light systems and the different junctions.

- *TraCI* (**Tr**affic **C**ontrol **I**nterface) allows the generation of the traffic simulation and the retrieve of vehicles values while manipulating their behavior "on-line" i.e. during run-time. TraCI made possible to get the status of the intersection at every time-step to set the action chosen by an agent.

- *SUMO-GUI* extends SUMO by a graphical user interface and lets the user the possibility to change the simulation speed and the graphical representation of a simulation. This tool leaves the possibility to visually check the performances of the models.

> As a simulation time reference, a SUMO step is equal to 1 second. For all the works done (except when it is mentioned), every episode consists of 5400 steps which translates into a traffic flow of 1 hour and 30 minutes.

3.2.2 Environment of the simulation

The simulated environment of the base model is composed of a traffic light system (Figure 15) integrated in a 4-way intersection illustrated in Figure 14.

This intersection consists of 4 edges composed of 8 lanes each (4 lanes approaching the intersection and 4 lanes leaving it). Each arm of the junction is 750 meters long starting from the vehicle spawn to the intersection's stop line. When a vehicle approaches the junction, it selects the lane based on its destination. Thus, below are listed the possible directions a car can take based on its lane choice:

- *Go straight* from the two central lanes and the right-most one.
- *Turn right* only from the right-most lane.
- *Turn left* only from the left-most lane.

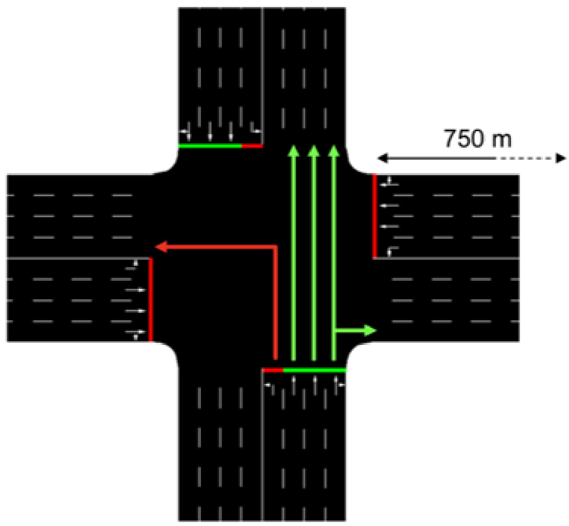


Figure 14: A zoomed view of the 4-way intersection without vehicles with lane directions

Traffic light system.

The traffic light system of the intersection is composed of 8 different traffic lights. Each of them are indicated by a color on the stop line of every incoming lanes. This color represents the status of the traffic light on a given time step. To illustrate, Figure 14 is an appropriate example. It shows a green light for vehicles coming from North and South direction that either want to go straight or turn right. A red light is displayed for every other lanes.

The set of traffic lights regulating the intersection can be viewed as in the following equation (21).

$$TL = \{tl_N, tl_{NL}, tl_W, tl_W, tl_E, tl_{EL}, tl_S, tl_{SL}\} \tag{21}$$

The subscript denotes the position in which the traffic light is located. For instance, tl_S represents the traffic light that controls all the cars coming from **S**outh that go straight or turn right. Following this logic, tl_{SL} manages the incoming traffic from **S**outh but only for the vehicles wanting to turn **L**eft.

The Figure 15 clarifies the number and the position of the different traffic lights in the junction.

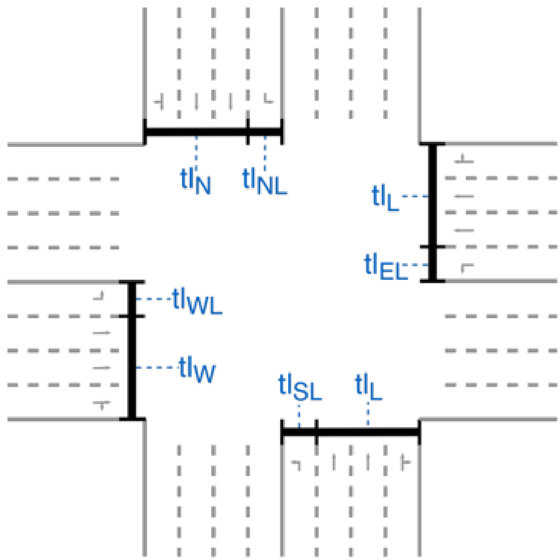


Figure 15: The position of every traffic light involved in the simulated environment.

The possible states of a traffic light are the same as in a real-life case (equation 22).

$$S = \{Green, Yellow, Red\} \tag{22}$$

Therefore, we can define a surjective function f from the set of traffic lights TL to the possible states set S:

$$f\colon TL \longrightarrow S \tag{23}$$

Finally, these traffic lights obey specific rules that are listed below:

- For every time-step, at least one traffic light is either in yellow or green phase.
- The color phase transition is cyclic: Red-Green-Yellow-Red.
- The duration of every phase is fixed i.e. 10 seconds for green time and 4 seconds for yellow time. The red phase lasts, consequently, the amount of time since the previous phase change.

3.2.3 State representation of the agent

In an effort to apply a learning algorithm, it is required to, first, define the state representation of the agent. The state is the agent's perception of the environment given a time-step t and is usually denoted as s_t.

The choice of a decent representation of the situation is crucial in order to allow the agent to effectively learn to optimize the traffic. This choice is also motivated by the amount and the quality of information to feed the agent.

Vidali's approach proposed a discretized representation of each arm of the intersection where each cell is not regular. It is inspired from the DTSE [23] with the difference that less information is encoded. As seen in Figure 16, not every cell has the same size: the further the cell is from the stop line, the longer it is, so more lane length is covered. The length of the shortest cells (the ones that are closest to the stop line) are exactly 2 meters longer than the length of a car (which is 5 meters long in SUMO).

It must be notified that a particular cell does not necessarily describe the situation in a single lane ! In fact, the 3 lanes dedicated to going straight and turning right share the same cells since they share the same traffic light.

There are 10 cells along the length of a lane which, added up to every arm of the intersection, results in a total of 80 cells.

To sum it up, in each arm of the intersection, incoming lanes are discretized in cells that can identify the presence or absence of a vehicle inside them.

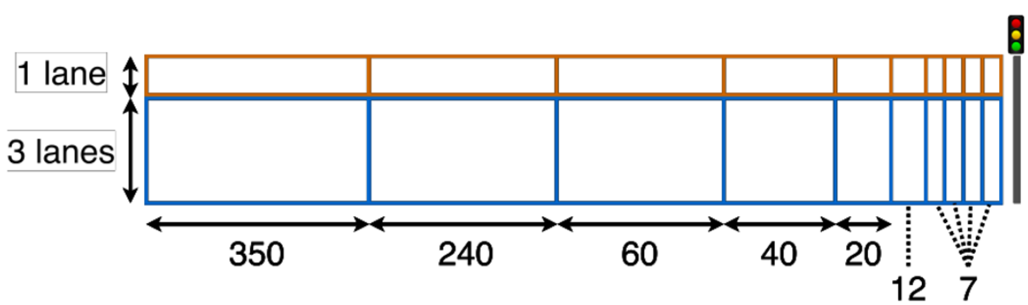


Figure 16: State representation design in the west arm of an intersection with cell lengths

That is why **the Intersection Discretized Representation** (IDR) was the chosen representation formally defined as in the following equation 24.

$$IDR_k \coloneqq c_k \tag{24}$$

where c_k is the k-th cell. The values of the IDR values are the ones in the equation 25.

$$IDR_k = \begin{cases} 1 & \text{if there is one or more vehicle inside} \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

where c_k is the k-th cell.

3.2.4 Action set

The possible actions that the agent can take are identified as the action set A. In our study case, the agent controls the traffic light system, this is why an action translates into applying a green phase to a set of lanes and implicitly keeping it fixed for some time.

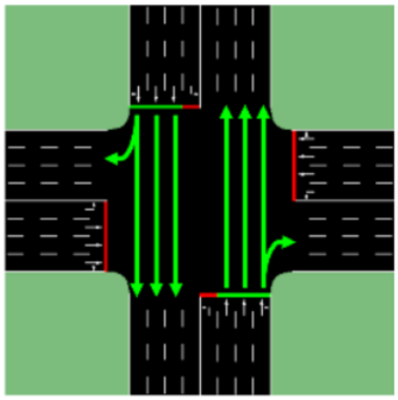
The cardinality of the action set is 4 and the latter is reported in equation 26.

$$A = \{NSA, NSLA, EWA, EWLA\} \tag{26}$$

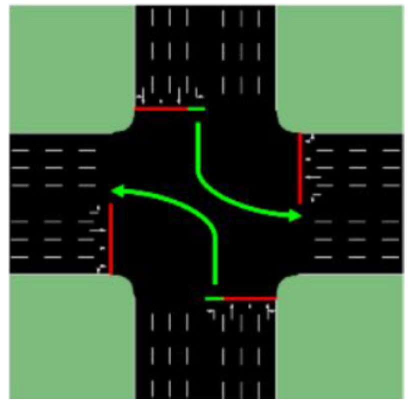
Each of the action a of A corresponds to the activation of the green phase depending of the incoming traffic.
The 4 possible choices are described below:

- **North-South Advance** (*NSA*) is activated for the vehicles that are in the North and South arm that goes straight or turn right.
- **North-South Left Advance** (*NSLA*) is green for the vehicles coming from North and South that want to turn left.
- **East-West Advance** (*EWA*): green phase is activated for vehicles that are in the East and West arm wanting to proceed straight or turn right.
- **East-West Left Advance** (*EWLA*) corresponds to the situation where vehicles are in the East and West arm turning left.

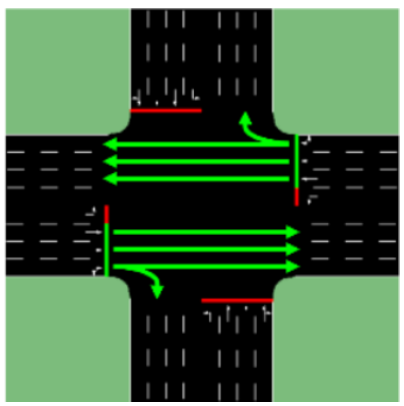
Figure 17 shows a visual representation of the possible actions.



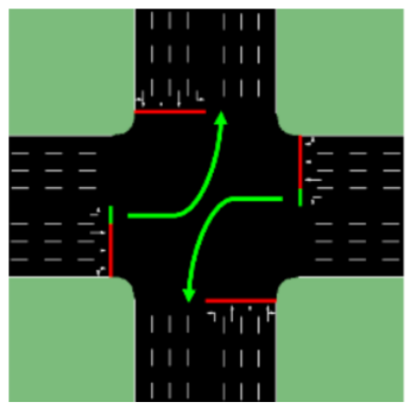
(a) North-South Advance



(b) North-South Left Advance



(c) East-West Advance



(d) East-West Left Advance

Figure 17: The four possible actions that composed the action set A

Consecutive actions case

If the action in time-step t is the same as the action taken in the last time-step $t-1$ there is no yellow phase. On the contrary, if the action is different, a yellow phase (that lasts 4 seconds) is initiated before the new action.

By recalling that 1 simulation step is equivalent to 1 second in SUMO. Therefore, 10 seconds are separating 2 same consecutive actions instead of 14 (combination of green and yellow phase).

3.2.5 Reward function

After choosing an action the agent receives a feedback from the environment called the reward denoted r_t. In RL, it represents a crucial aspect of the learning process. It has two possible values positive or negative. In this application, a negative reward is generated from bad actions. The objective is to minimize the negative reward in order to maximize the traffic flow through the intersection over time.

In order to increase it, there are multiple candidate measures such as throughput, mean delay and travel time. The chosen metric was the total waiting time which is the sum of individual waiting times of each car in the environment in timestep t. The waiting time of a vehicle increments when this vehicle has a speed below than $0.1m/s$. It is defined in equation 27.

$$twt_t = \sum_{i=1}^{n} wt_{i,t} \tag{27}$$

where twt_t represents the total waiting time at timestep t and $wt_{i,t}$ the waiting time of a vehicle i at timestep t.

The concept of waiting time is crucial for the choice of the reward definition and seems the most accurate measure among the proposed ones. Indeed, the total waiting time does take into consideration the time spent by cars in a stationary position, not just the fact that a car is stopped (which is the case of queue length and throughput).

The reward function that generates the reward for the agent is called *literature reward function* and is defined in equation

$$r_t = twt_{t-1} - twt_t \tag{28}$$

where r_t is the reward at timestep t, twt_{t-1} and twt_t act for the total waiting time of all cars in the intersection, respectively, at time $t-1$ and t.

This reward function is designed is such a way that when the agent chooses a poor action it returns a positive value. It can be rephrased in this manner: Because of a bad action at timestep t, it leads to more vehicle waiting in queue compared to the previous situation at timestep $t-1$. It results in higher waiting times values. That is why the goal of the agent is to minimize the bad actions (or maximize the good ones) in order to minimize the negative r_t.

3.2.6 Agent's learning mechanism

The agent's learning mechanism involved is called Deep Q-Learning which is a combination of Deep Q-Networks and Q-Learning (see section 2.1.2).

Q-Learning

In the baseline work, a slightly different version of the classic Q-value update (13) is used and presented in equation 29.

$$Q(s_t, a_t) = r_{t+1} + \gamma.max_A Q'(s_{t+1}, a_{t+1}) \tag{29}$$

where the reward r_{t+1} corresponds to the reward received by the agent after taking action a_t in state s_t. $Q'(s_{t+1}, a_{t+1})$ is the Q-value associated to the action a_{t+1} in state s_{t+1}. The term γ is the discount factor of future reward compared to the immediate reward. It helps integrating the notion of trade-off between short and long-term reward. Therefore, the above equation (29) is a rule that updates the Q-value of the current action a_t in state s_t with the immediate reward and the discounted Q-value of future actions. To be more concise, the term $Q'(s_{t+1}, a_{t+1})$ represents the value of future actions implicitly holds the maximum discounted reward of the state after the state s_{t+1}. To illustrate, equation 30 unpacks the previous equation.

$$Q(s_t, a_t) = r_{t+1} + \gamma.r_{t+2} + \gamma^2.r_{t+3} + \gamma^3.r_{t+4} + ... + \gamma^{n-1}.r_{t+n} \tag{30}$$

where n is an arbitrary value that indicates the last timestep before the end of the episode (up to this point there are no more future actions and therefore the future reward is 0).

This expanded Q-value aims at making clear the type of reward that the agent receives.

Deep Q-Learning.

To map a state s_t to Q-values, the deep neural network takes as input the vector IDR_t (or NSR_t as presented in Section 3) which is the state of the environment at timestep t. Formally, the input of the NN is defined in equation 31.

$$x_{i,t} = IDR_{i,t} \tag{31}$$

where $x_{i,t}$ is the i-th input of the neural network at time t and $IDR_{i,t}$ is the i-th element of the vector IDR at time t. $|x|$ is the input size of the neural network.

$$y_{j,t} = Q(s_t, a_{j,t}) \tag{32}$$

where $y_{j,t}$ is the j-th output at time t and $Q(s_t, a_{j,t})$ corresponds to the Q-value of the j-th action taken from state s_t. The output cardinality of the NN is $|A| = 4$ (as specified in 3.2.4).

The description of the neural network structure will be presented in Section .

3.3 The training process

This subsection aims at showing how all the agent components, previously described, work together.

To make the link with the previous section, Figure 18 introduces the Deep Q-Learning mechanism implied in this thesis. First, the agent retrieves the environment state s_t. Next, using the delay times of the current timestep, it calculates the reward r_{t+1} associated to the previous action a_t. Then, the agent stores information tuples (samples) in the form $(s_t, a_t, s_{t+1}, r_{t+1})$ into a memory used in the agent's training. Finally, the agent chooses the new action to apply in the environment.

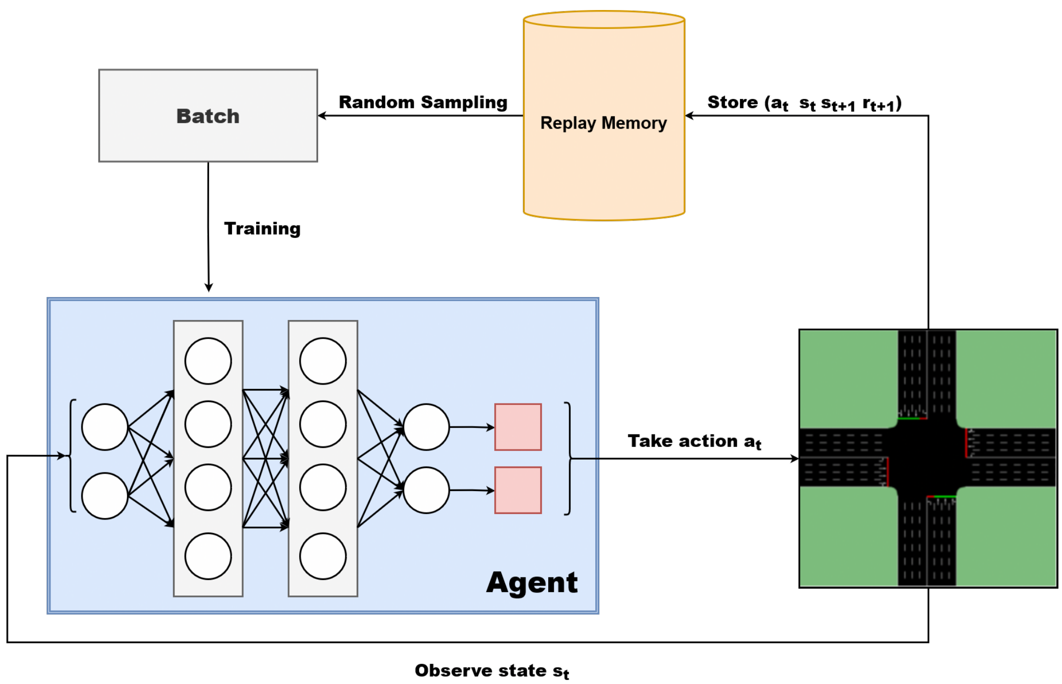


Figure 18: Deep Q-learning implementation process

3.3.1 Experience Replay characteristics

In this thesis, the characteristics of the memory are:

- **Memory size.** It contains the number of samples the memory can hold. It is set at 50000.

- **Batch size.** It is the number of samples retrieved from the memory in one training instance. As mentioned above, the oldest sample is removed to make space for the new one when the memory is filled.

Sampling strategy.

Multiple types of samples gathering from the memory could be used but the one that was chosen from Vidali's experiments is called the *Intensive* one. It consists in initiating the training phase at every simulation step, instead of timesteps, and setting the batch size at 100 samples. This king of intense training makes the agent gather approximately 540000 samples per episode (which about 10 times the memory size).
The purpose of this strategy is to learn a more stable policy by training the agent's neural network with more examples in a shorter period of time.

3.3.2 Exploration vs. Exploitation trade-off

In RL, the agent tries to discover its environment and relies on *trial-and-error*. During the training, the agent has a set of actions to select from. The agent can choose to explore by selecting an action with an unknown outcome to get more information about the environment, or it can choose to exploit i.e. choose an action based on its prior knowledge.

A *greedy strategy* simply consists of always taking the decision that seems to be the best with respect to the current knowledge. And, in Vidali's decision making process, ϵ-greedy algorithm, which is the most naive but yet efficient way to explore, was chosen.

The idea is to take either the action that seems to be optimal with probability $1 - \epsilon$ (which is the **exploitation**) or a complete random action with probability ϵ (the **exploration** phase). Formally, the ϵ-greedy can be stated in an equation as being a probability ϵ for the current episode h to choose an exploratory action or a probability $1 - \epsilon$ to choose an exploitative action (equation 33).

$$\epsilon_h = 1 - \frac{h}{H} \tag{33}$$

3.3.3 The overall process

Every time a training instance of the agent is initiated, the following process is executed.

- A sample m is added to the memory
- Depending on the sampling strategy, a fixed number of samples are randomly picked from the memory constituting the batch B.

Then, for each and every sample the four following operations are performed using equation 29.

1. Computation of the Q-value $Q(s_t, a_t)$
2. Computation of Q-values $Q'(s_{t+1}, a_{t+1})$
3. Update of the new Q-value
4. Training of the neural network

The next time the agent encounters the state s_t (or a similar one), the DNN will be more likely to output the Q-value of action a_t that is considered as the best action to create the best future situation.

Algorithm 2 corresponds to the algorithm implemented and summarizes the components and the main steps of the training process of the agent described in this section.

Algorithm 2: Deep Q-learning with Experience Replay in SUMO

```
// Initialization
Initialize replay memory M with capacity C
// Main loop
for episode = 1, E do
    Initialize s_0 = −1, a_0 = −1
    // Simulation loop
    for step = 1, U do
        Get state representation s_t
        Initialize p as a random number ∈ [0, 1[
        if p < ϵ then
            select random action a_t
        else
            select a_t = argmax_a Q*(s_t, a_{t−1})
        end
        Simulate action in SUMO
        Receive reward r_t
        Store transition (s_{t−1}, a_{t−1}, r_t, s_t) in M
    end
    // Experience Replay
    Sample random batch of transitions from M
    for samples in batch do
        Update the learning equation
    end
    Train the Neural Network
end
```

Notes: In our particular study case, $C = 50000, E = 100, T = 5400$

3.3.4 Scenarios

Traffic generation is one of the central part of the simulation because maintaining a high degree of reality is necessary. That is why Weibull distribution with a shape equals to 2 has been chosen. Figure 19 illustrates the distribution in the form of an histogram where on the x-axis are defined the number of simulation steps and on the y-axis is the number of vehicles generated during a windows step. This approximation simulates a bell curve where, at the early stages the number of cars increases until reaching a peak. Afterwards, the number of incoming vehicles decreases.

For every car, the source and destination are randomly determined using a seed changed for every episode (which they are specified in Section 5).
Four different scenarios has been developed:

- **High-traffic** scenario: 4000 cars are generated. This scenario corresponds to an overwhelming number of vehicles. It represents the traffic load in peek times. This scenario produces long queues even with good control. The idea is to check if it is possible and how much can it be mitigated.
- **Low-traffic** scenario: 600 cars are generated. This scenario represents a normal flow where there is not peek time. However, it is a good amount of cars to produce traffic jams if the traffic control is not good.
- **NS-traffic** scenario: 2000 cars generated with 90% of the traffic coming from the North and South junctions. The 10% remaining ones are starting from the East and West junctions. This scenario relates an unbalanced situation where a traffic deviation could arise and force vehicles to drive on the East-West axis.
- **EW-traffic** scenario: 2000 cars generated with 90% of the traffic coming from the East and West junctions. The motivation is analog as the one described in the above point.

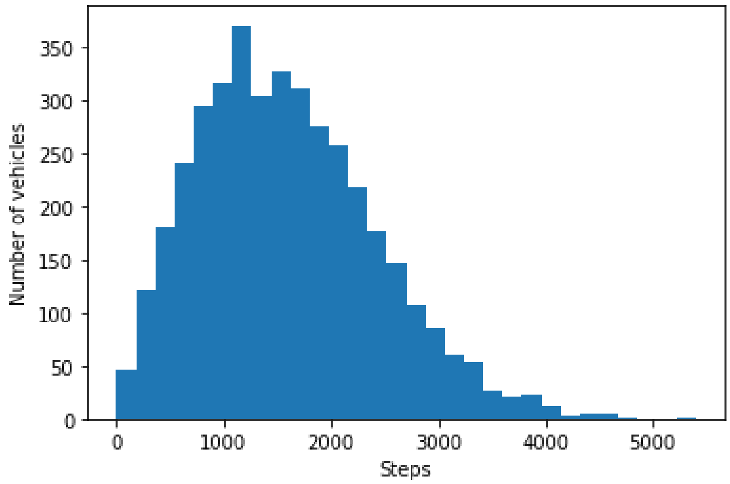


Figure 19: Traffic generation distribution over one episode for the High scenario. The number of bins in this histogram is 30. And the seed chosen corresponds to the first seed of the testing in Section 5 (45715).

Source and destination of vehicles. Every car has the same generation probabilities. In High and Low traffic scenarios, the car has a probability of 75% to go straight and 25 to turn left or right. From this distribution, source and destination are uniformly chosen. In NS and EW scenarios, the difference lies in the source probability. In fact, in NS-traffic, a car has a probability of 90% of coming from North or South and 10% of being sent from East or West. It is the exact contrary for the EW-traffic scenario. Given this source probability, the destination is chosen with a uniform probability.

> The seed used for the random generation of source and destination of every car in one episode corresponds to the episode number, so is not possible to have the same car generation pattern in two different episodes.

Finally, every car generated in one episode have the same characteristics described in Table 3.

Attribute	**Value**
Acceleration	1 $m.s^{-1}$
Deceleration	4.5 $m.s^{-1}$
Vehicle length	5 m
Min gap between two vehicles	2.5 m
Max speed	25 $m.s^{-1}$

Table 3: Vehicle characteristics

These information were extracted and adapted from Vidali's thesis and paper. The aim was to describe the baseline model from which this thesis could be built upon. So, for more information and in-depth explanations do not hesitate to have a look at his work.

4 Proposed approach: one 4-way intersection

In this thesis, the chance of improvement in traffic flow that drives through an intersection controlled by traffic lights will be investigated using Andrea Vidali's work as a baseline work (see the details in Section 3). Multiple experiments will be conducted in order to improve and better analyse the different elements of the framework.

In Table 4 are listed the terms used in this chapter.

Category	**Notation**	**Meaning**
Traffic	TL	Set of traffic lights
	N	North junction of TL
	E	East junction of TL
	W	West junction of TL
	S	South junction of TL
	AQ(s)	Artificial Queue(s)
RL	IDR	Intersection Discretized Representation
	NSR	New State Representation

Table 4: Notations Section 3

4.1 Deep Q-learning agent improvements

The first part of the thesis approach lies in the improvement of the base model while testing its limits. From enhancing the state environment to speeding the NN's training, multiple experiments will be conducted in order to understand deeper what a Deep Q-learning algorithm is capable of on TSC tasks.

The following parts can be subdivided into two main groups : the first four are set up in order to **improve the raw agent**, while the last two can be considered as an *in-depth analysis* which could give some ideas for future improvements and would require further study to enhance the single agent.

4.1.1 Enhancement of the state representation

As of the discretized state representation implemented (*IDR* is detailed in 24), the intent is to give more information to the agent about the current environment situation. It is also taking into account the fact that these features have to be easy to collect (which could be the case with a simple lane detector).

Below are listed the new knowledge points per each cell:

1. the number of cars
2. the average speed of the vehicles
3. the accumulated waiting time
4. the number of queued cars

In a more formal way, let NSR_k be the **New State Representation** of a cell c indexed at position k and denoted c_k (in a similar way as 24).
And let i be the vehicle identifier belonging to the interval $[\![0; n-1]\!]$ with n being the number of vehicles involved in the simulation scenario chosen. Equation 34 defines the variable w_i which states that if a vehicle i is in the cell c_k then $w_i = 1$, otherwise $w_i = 0$.

$$w_i = \begin{cases} 1 & \text{if the vehicle } i \text{ is in the cell } c_k \\ 0 & \text{otherwise} \end{cases} \tag{34}$$

Then, the number of vehicles in the cell c_k can be summed up as:

$$N_{c_k} = \sum_{i=0}^{n-1} w_i \tag{35}$$

Let $speed_i$ be the speed of the vehicle i whose values belong to $[0; 13.89]$ m/s, the average speed of vehicles in the cell c_k can be formulated as following:

$$AvgSpeed_{c_k} = \frac{\sum_{i=0}^{n-1} w_i \times speed_i}{N_{c_k}} \tag{36}$$

Let $WaitingTime_i$ be the waiting time of the vehicle i during the last timestep, subsequently, the accumulated waiting time of vehicles in the cell c_k is:

$$AccWaitTime_{c_k} = \sum_{i=0}^{n-1} w_i \times WaitingTime_i \tag{37}$$

Equation 38 defined the variable q_i which helps to determine whether a vehicle is stopped or not.

$$q_i = \begin{cases} 1 & \text{if the vehicle i has a speed} \leq 0.1\ m.s^{-1} \\ 0 & \text{otherwise} \end{cases} \tag{38}$$

Therefore, the number of queued vehicles in the cell c_k can be formally set as being:

$$NQueued_{c_k} = \sum_{i=0}^{n-1} w_i \times q_i \tag{39}$$

Thus, each 80-uplet corresponds to a vector describing the knowledge points listed before.

$$\begin{aligned} \overrightarrow{N} &= (N_{c_1}, N_{c_2}, ..., N_{c80}) \\ \overrightarrow{AS} &= (AvgSpeed_{c_1}, AvgSpeed_{c_2}, ..., AvgSpeed_{c80}) \\ \overrightarrow{Awt} &= (AccWaitTime_{c_1}, AccWaitTime_{c_2}, ..., AccWaitTime_{c80}) \\ \overrightarrow{NQ} &= (NQueued_{c_1}, NQueued_{c_2}, ..., NQueued_{c80}) \end{aligned}$$

Finally, NSR_k is the combination of the above vectors i.e.

$$\overrightarrow{NSR_k} = (\overrightarrow{N}, \overrightarrow{AS}, \overrightarrow{Awt}, \overrightarrow{NQ}) \tag{40}$$

State space.

The vector length is 320 which is 4 times bigger than the one of the initial representation. In addition with the fact that the previous number of possible states (with the boolean cells) was 2^{80}, it is largely outnumbered in this case. But the agent must explore the most significant subset of the state space in order to learn the best behavior. Indeed, when there is at least one vehicle stopped waiting for the green phase, the best action to take in this particular state is to let the vehicles go. Expressly, the cells closer to the stop line are the most important ones, which means that the combinations of states where there are active cells closer to the stop line have a bigger contribution to the food performance of the agent. Therefore, the size of the state space is significantly reduced to a quantity where the training duration is not overwhelming. The training durations are specified in Section 4.

> In short
>
> When the agent samples the environment at each timestep t, it receives $\overrightarrow{NSR_k}$ carrying the new discretized representation of the environment. As said above, it is important to keep it balanced but quite precise at the same time. A trade-off must be found in order to not over-increase the computational complexity of the neural network's training.

4.1.2 Introduction of a new Deep Neural Network

The DNN which is in charge of the Q-values selection is at the center of the Deep Q-Learning framework implemented.

The chosen structure is a *feed forward* neural network i.e. an artificial neural network (ANN) wherein connections between the neurons are straight forward (do not form a cycle). A graphical representation has been introduced in Figure 20. Some modifications have been added to the one proposed by Andrea Vidali. The former model is composed of a 80 neurons input dimension (which is consistent with the *IDR*), 4 hidden layers of width 400 and a 4 dimensions output layer. This output layer is correlated with the 4 Q-values of the agent (which are its 4 possible actions).

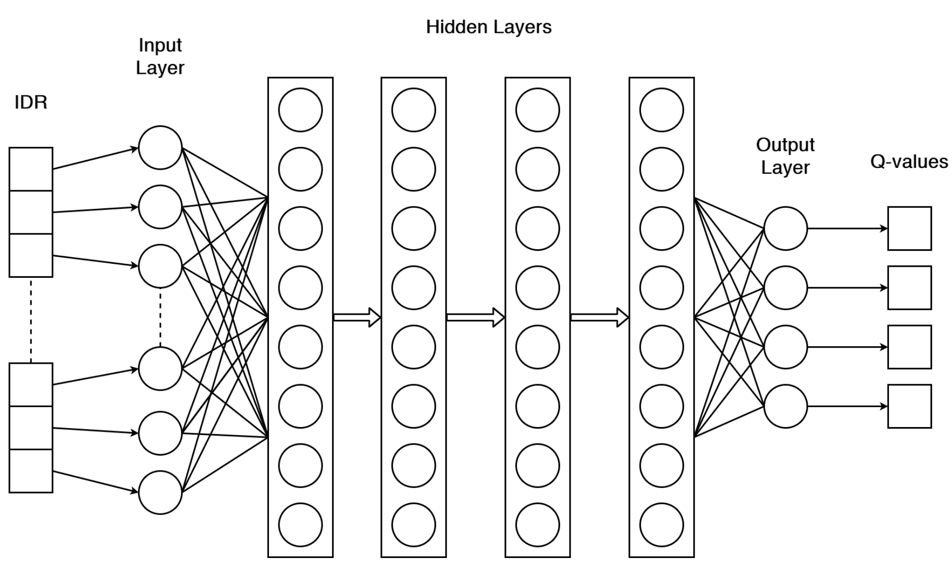


Figure 20: Representation of the feed forward base network

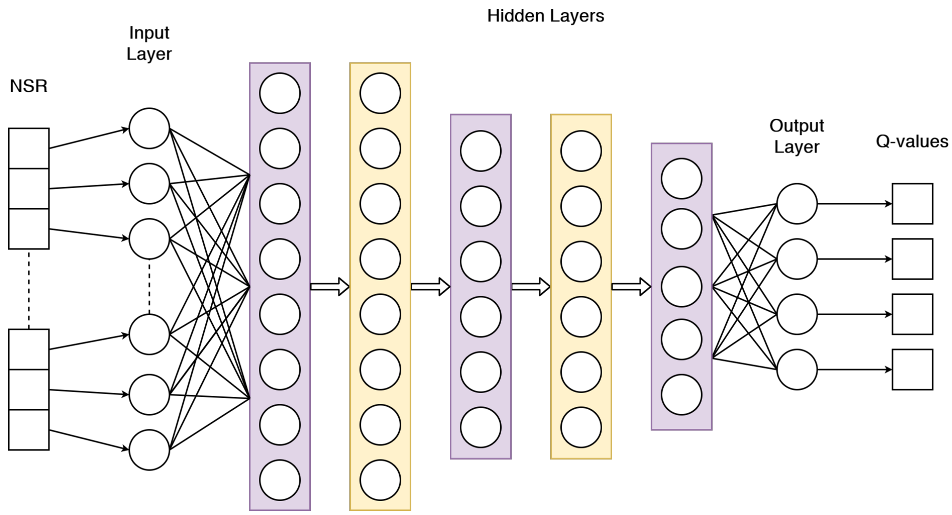


Figure 21: Illustration of the new DNN structure implemented

Different paths have been followed to improve the base structure beginning with the addition of dropout layers to integrate a form of regularization.

Regularization

Regularization is a well-known and well-used technique in Machine Learning to handle the problems of *Model Generalization* and *Over-fitting*. Dropout is one of the regularization method which probabilistically removes inputs during training. The Figure 22 illustrates the process: the ensemble of all the possible sub-networks of the base one (a) is trained resulting in (b). (This ensemble is constrained by hyperparameters controlling the probability of the mask.)

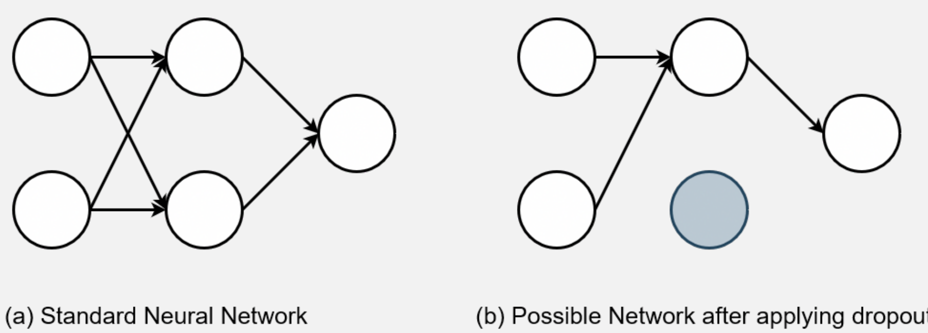


Figure 22: Example of Dropout Regularization

Then, layers width has been increased (up to 320 with respect to *NSR*). But, the reported results (in Section 4) will show that too much of the increase is detrimental to the learning capacity. Finally, the idea to funnel down the width of the layers has been followed.
Figure 21 illustrates the modifications applied. The state representation has now shifted to the *NSR* instead of the *IDR*. The hidden layers structure have been changed: purple ones correspond to dense layers and yellow ones represent the dropout layers. Notice that the width of the hidden layers has been decreased one at a time.

In the end, compared to equation 31, the new input of the NN is, now, defined as:

$$x_{i,t} = NSR_{i,t} \tag{41}$$

4.1.3 Visualization options

Data visualization makes data easier for the human brain to understand and pull insights from. Its main goal is to make it easier to identify trends, patterns and outliers.
In our particular study case, the need to understand better the agent's learning mechanism and the simulations behavior has driven the addition of more options to visualize the training procedure of the agent.
A detailed list of the plots realized and its associated description is provided below:

- **Average (and minimum) loss of the neural network** - Loss is a number indicating how bad the model's prediction was. Its value is starting from zero meaning that the model's prediction is perfectly accurate (the lower the better). Many possible measure of errors are possible: Mean Average Error (MAE), Root Mean Square Error (RMSE), Huber Loss, Log-Cosh Loss, etc. The one chosen by default is the Mean Square Error (MSE) also known as Quadratic Loss or even L2 loss. It is the most commonly used regression loss function. (Equation 42).

  $$MSE = \frac{\sum_{i=1}^{n}(y_i - x_i)^2}{n} \tag{42}$$

  $(y_i - x_i)^2$ is the *squares of the errors* where y_i is the prediction and x_i the theoretical true value.

  But as the choice of a good loss is not evident and often underestimated, an experiment (section) has been realised to outline the impact of the loss on the overall result.

- **Cumulative delay** - The retrieved waiting time of every car in the incoming roads is summed up in each episode. This information is considered as an aggregate measure that helps understand if and when the agent is able to choose better action phases that will impact positively (or negatively) the traffic flow.

- **Cumulative negative reward** - Each negative reward is added up. The directional relationship (convergence, stability or divergence) between the episode number and this sum defines the learning ability of the agent.

- **Average queue length** - It corresponds to the average queue length of vehicles of every incoming roads of the intersection which is analog to the cumulative delay and is another form of aggregate measure.

- **Average waiting time per vehicle** - It represents how much time a vehicle has waited, in average, in an episode. This metric allows a better and finer appreciation of the analysis of the results as being more human-centered.

> The above measures are strongly correlated but a difference on their curve can help understand some particular behaviors or defects happening in the training process.

The **Fundamental diagram of traffic flow** is also an important macroscopic (or aggregate) metric that is often used ([52], [53]) to predict and describe the capacity of a road network. This diagram gives the relationship between the traffic flow [vehicles/hour] and the traffic density [vehicles/km] of every edges.
An alternative to this representation has been proposed. The traffic density can be replaced by the traffic occupancy. It corresponds to the ratio of the sum of the lengths of the vehicles to the length of the road section in which those vehicles are present [%].

4.1.4 Multiprocessing Python approach

Working on improving the developed framework training time has been necessary at a certain point. In fact, the intention was to reduce the training time of the model without provoking any significant loss.

In the baseline model, the application structure was quite simple and thought for a simple case. It consisted in launching one simulation for one specific scenario... So, for the purpose of training an agent directly on the four different scenarios $\{Low, High, NS, EW\}$, it was not convenient.

Therefore, the new implementation process is described in Figure 23. It can be summed up as the following ordered steps. First, from the main class a pool of 4 processes that corresponds to each specific scenario are launched. Each of these simulations communicate with the Flask [54] server. This local server is set up to handle different requests like *initializing the agent* or *storing the samples in the agent* *memory*. After updating the agent memory and model instances it can send back responses to the workers. Finally, when these tasks are done, the application submits a last request to the server in order to train the agent on samples of the simulations it has experienced. This process is done 100 times until the final results are saved.

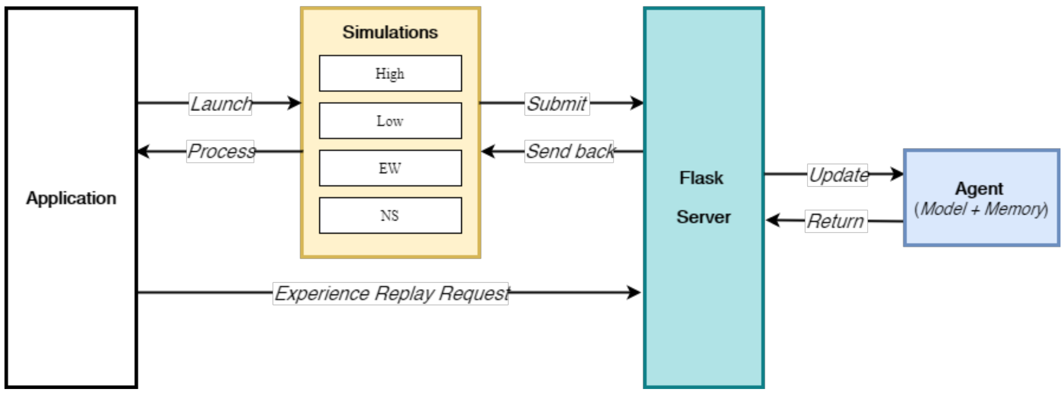


Figure 23: Multiprocessing process described

For the sake of explaining the thought process resulting to this approach, the experiments made chronologically are listed and described:

1. **Without**: It corresponds to the simulation with the baseline model without any other modifications. Its purpose is to set the reference time.

2. **With**: It takes into account the Python multiprocessing implementation developed in Figure 23 but experience replay is done by the agent after each simulation scenario (in other words 4 times as many for one episode).

3. **With PR**: It corresponds exactly to the diagram procedure. (No significant changes have been observed in terms of results.)

4. **With PR big**: This experiment has the objective to show the impact and the role of changing the number of weights in the DNN model with respect to run time.

4.1.5 Adaptive Green Light

One approach that has not yet been addressed is to *change the green light phase duration depending on the traffic flow*. The agent has not the explicit capacity to modify the green time. Actually, the agent already has the implicit possibility to learn to extend the green phase duration by continuously choosing the same phase multiple times in a row. But, sometimes, it is not enough. In High-traffic scenario, the agent decision oscillates between multiple edges when the best strategy would have been to go all in for one specific axis and, after some timesteps, change for another axis.

Therefore, two possibilities are proposed which could lead to the definition of an adaptive framework.

Increase or decrease of the green phase duration.

First, the agent is trained and tested with a higher green phase duration. 10 seconds as green phase duration may not be enough compared to the 30 seconds of the static traffic light phases (EWA and NSA). In the High, NS and EW scenarios, letting vehicles pass through the intersection a longer time may help to increase the stability in the learning curve (instead of oscillating between all directions if all edges are accumulated).

Modification between training and evaluation.

Second it could also be interesting to experiment the behavior of an agent that was trained on a higher duration but evaluated on a lower one (and respectively). It could help to better understand and appreciate a possible future application in a real-world case. It would test the capacity of the agent to better adapt to the situation.

> The main idea of these experiments are that understanding the agent performances better helps to choose the best compromise and, therefore, to integrate the capacity of changing the duration depending on the scenario.

4.1.6 The importance of a good loss function

As introduced in the section 4.1.3 the loss choice is not straightforward. One loss function over an other might be more appropriate on a particular problem. That is why few alternatives have been investigated to analyse the impact it could have on the learning curve and, eventually, the end results.

- **Mean Absolute Error**, L1 loss (MAE) - it has been motivated by the fact that MAE is a generic and even measure of how well the model is performing. No detailed information about the outliers is needed.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \tag{43}$$

  where $|y_i - x_i|$ is the arithmetic average of the absolute errors where y_i is the prediction and x_i the theoretical true value.

- **Huber Loss** also called Smooth Mean Absolute Error - it is often a good trade-off between MAE and MSE since it is basically absolute error becoming quadratic when the error is small.

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{if } |y - f(x)| \leq \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} \tag{44}$$

where δ is an hyperparameter that can be tuned to determine the sensitivity to outliers. It could be summed up by saying that when $\delta \sim 0$ the loss approaches MSE and when $\delta \sim \infty$ it approximates MAE.

There are other loss functions that could have been investigated such as the *Log-Cosh* function, which is the logarithm of the hyperbolic cosine of the prediction error. The *quantile function* is worth mentioning because, intuitively, it carries in its definition the uncertainty of the predictions. It is definitely where our case study falls into: a real-work prediction problem where knowing the range of predictions as opposed to only point estimates could improve the decision making process.

4.2 Introduction of artificial queues

At this point, the work has been centered on the deep reinforcement learning framework improvements. But, it would have been interesting to test the robustness of the model by increasing the *realism of the traffic scenarios*. Indeed, in a real life scenario, traffic behavior is not even close to be smooth and easily predictable.

That is why, as it is done is some works in ATSC [47] [55], the will to add more realism to the scenarios combined with the idea of testing the robustness of the model led to the creation of artificial queues (AQs).

At every outcoming junction, instead of having a **dead-end** where the vehicles are in «free-flow »and disappearing, the vehicles must wait some predefined time before coming out of the edge. The implication of this strategy is to create some congestion at the end of road which could, then, lead to some observable changes in case of an agent acting poorly. It will eventually create a queue that will block the intersection and (dramatically) affect the overall traffic flow.

Implementation of AQs.

Figure 24 shows an AQ in N (the North junction of TL). Each vehicle has to wait before leaving the edge. Thus, the vehicle which is the furthest from the stop line has to wait 3 times the regular amount of time.

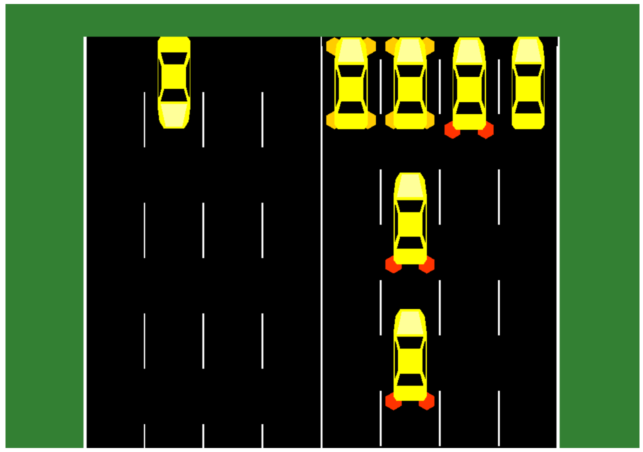


Figure 24: Artificial queues simulated at the North junction

In this regard, the downtime has been set up stochastically thanks to a Gaussian distribution. Equation 45 recalls the general form of the probability density function.

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{45}$$

where μ is the mean and σ the standard deviation. σ^2 is the variance. Knowing that the function has its peak at the mean and its spread increases with the standard deviation, the following constants have been assigned:

$$\mu = 2000 \qquad \sigma = 1500 \tag{46}$$

resulting in figure 25.

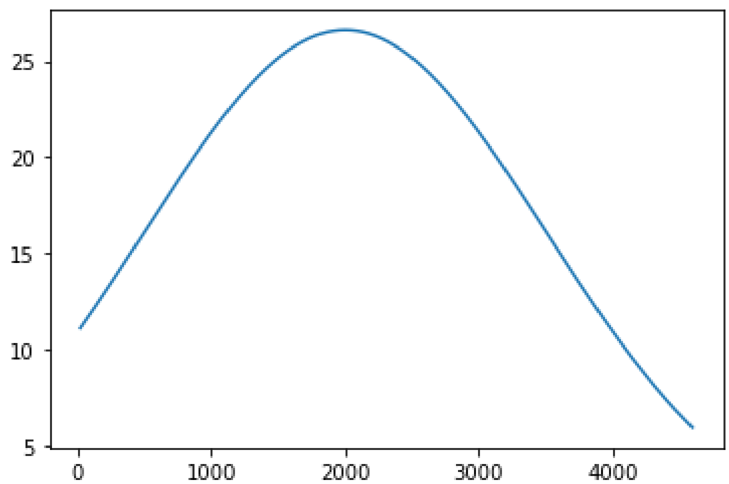


Figure 25: Normal distribution of the stop time in out coming roads

The x-axis shows the episode number from 0 to 5400 while the y-axis shows the stop waiting time in seconds. The latter values have been multiplied by $1e5$ to obtain a reasonable time in seconds.

The waiting time at the stop line is setup at around 10 seconds at the beginning of the simulation. Then, it attains a peak at 25 seconds at a time where there is also a peak of vehicle generation (19). The idea is to create an accumulation of stopped vehicles to amplify the number of cars in the overall simulation which exacerbates the need for the agents to make good decisions in order to avoid catastrophic results. Finally, the waiting time is slowly decreasing until the end of the simulation, where 5 seconds only is needed for a car to quit the edge.

Calibration of AQs.

One additional change has been added. Depending on the current scenario (EW or NS), the stop waiting time is discounted by a factor $l = \frac{1}{2}$ (equation 47).

$$l = \begin{cases} \frac{1}{2} & \text{for E and W outings in EW-traffic or N and S outings in NS-traffic} \\ 1 & \text{otherwise} \end{cases} \tag{47}$$

For instance, if the scenario is the NS-traffic one, then a longer time will be associated to East and West stop roads. It is also very helpful in the future study case of the two-intersections where a specific need to calibrate the results will arise (see section 6.2).

It is, finally, important to mention that the introduction of artificial queues led to a modification in setting up the destination of the cars. In each and every scenario, a random lane number had to be specified to overwrite the one chosen by SUMO. In fact, by default, when adding a stop junction, the lane set for the vehicles is the right most one (which corresponds to a real-case scenario). But, the idea is to create a uniform queue for approximating a traffic jam. Consequently, on top of this rule, the destination is chosen with a uniform probability.

5 Results and discussion: one 4-way intersection

In this section, experiments that have led to the development of the thinking process of a single deep Q-learning agent will be presented and discussed. First, the baseline results chosen to evaluate the performances of the tested agents will be explained. Then, the improvements delineated in the previous section will be expressed in terms of comparable results.

5.1 Performance Metrics

In order to evaluate an agent performance, a set of fixed experiments has been conducted on every trained agent. For each of the four scenarios, a particular agent is evaluated on 5 episodes on which a chosen random seed for vehicle generation is determined. The car generation's distribution used for the evaluation will be the Weibull distribution (described in 19). The results obtained after the 5 simulations are averaged and considered as the performance of the agent on that specific scenario. Below are listed the metrics used for the evaluation:

- **Average negative reward**

$$nrw_{avg} = \frac{\sum_{ep=1}^{5} \sum_{t=0}^{nt} nrw_{t,ep}}{5} = avg_{ep}\left(\sum_{t=0}^{nt} nrw_{t,ep}\right) \tag{48}$$

  It corresponds to the sum of the negative rewards received at every timestep t in an episode ep, averaged among 5 episodes.

- **Total waiting time**

$$twt = avg_{ep}\left(\sum_{v=0}^{n-1} wt_{v,ep}\right) \tag{49}$$

  It represents the sum of the waiting times wt for every vehicle identified by v which belongs to the interval $[\![0; n-1]\!]$ with n being the number of vehicles involved in the simulation. This result is averaged among 5 episodes and is expressed in seconds.

- **Average waiting time per vehicle**

$$awt/v = median_{ep}\left(\frac{\sum_{wv} wt_{wv,ep}}{|wv|}\right) \tag{50}$$

  This is the median value of the sum of the waiting time of the vehicles that have waited wv over the number of vehicles that have waited in this episode. Its value is measured in seconds.

Please note that the values are rounded to the nearest tenth while the calculated difference (in percents) is rounded to the nearest one.

Seeds

In the training case, each seed used for the random generation of source and destination of every car in one episode corresponds to the episode number (51).

$$seeds_{training} = \{0, 1, 2, 3, 4\} \tag{51}$$

For testing, the seeds for vehicle generation are randomly chosen with the constraint that they were not initially used for the training. In other words, they are disjoint sets (52).

$$seeds_{training} \cap seeds_{testing} = \emptyset \tag{52}$$

For the purpose of **reproducibility**, the seeds are reported in 53.

$$seeds_{testing} = \{45715, 92490, 80265, 3957, 40983\} \tag{53}$$

It is important to take into account that other seeds could have led to noticeable changes in the results.

5.2 Performance plots

To provide a deeper analysis of the results, two type of plots will be analysed (which are the two main ones to provide interesting analysis).

- For the training phase of the agents the chosen figure will be the **cumulative negative reward per episode**.

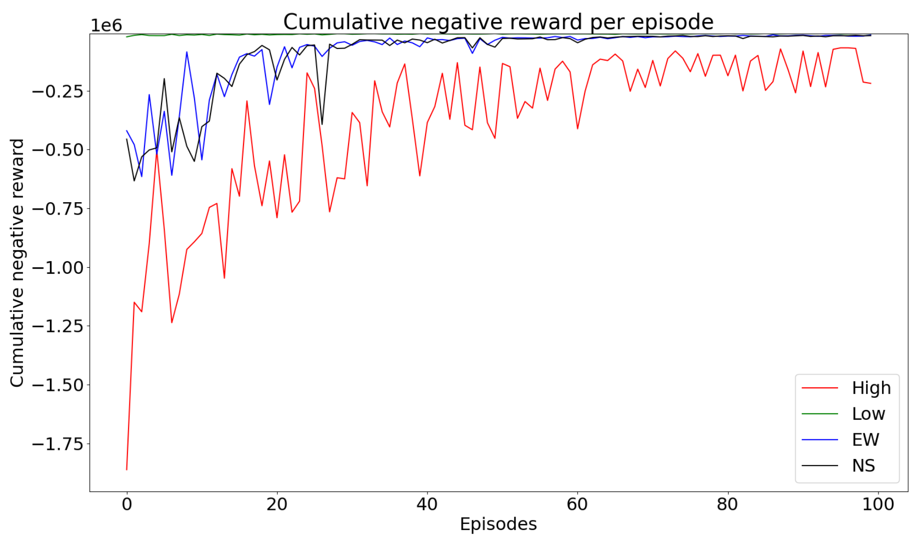


Figure 26: Example of the plotted cumulative negative reward curves for each scenario and per episode of a trained agent

Figure 26 is an example where the y-axis corresponds to the cumulative negative reward and the x-axis to the episode number. There are 4 curves: the green one corresponds to the Low-traffic, the blue one to the EW-traffic, the black one to the NS-traffic and the red one to the High scenario.

- The analysis for the testing phase will be produced thanks to the **average queue length of vehicles per steps over the 5 episodes** relative to a specified scenario.
  An example will be presented in the very next section (27).

> However, other interesting plots that could have been discussed are also presented and analysed in Appendices.

5.3 Static Traffic Light System

The evaluation of the performances is made with respect to a Static Traffic Light (STL) system where the cycle of the phases is predefined in equation 54 and described in Section 2.6.

$$A_{STL} = \{NSA, NSLA, EWA, EWLA\} \tag{54}$$

The STL will cycle through every traffic light phase always in the same order, and the phases have a fixed predefined duration. In particular, the order of the traffic light phase activation is **[NSA - NSLA - EWA - EWLA]**. According to Koonce and Rodegerdts [56] the duration of NSA and EWA phases are 30 seconds, while NSLA and EWLA 15 seconds. Between each green phase a yellow one is necessary and lasts 4 seconds (Table 5).

Phase	**Duration (s)**
NSA	30
NSLA	15
EWA	30
EWLA	15
Yellow	4

Table 5: STL phases duration

The evaluation of the STL results are shown in Table 6. This reported data will be the standard baseline for the agent's performance to compare with.

	Low	**High**	**NS**	**EW**
nrw_{av}	-5985.4	-129757.8	-81487.6	-79461.6
twt	15709.2	606655.6	146007.0	143470.2
awt/v	37.3	111.6	61.6	60.3

Table 6: Results of Static Traffic Light (STL)

In Low-traffic, a fixed cycle is not a good strategy since many vehicles can wait a significant amount of time while the green light phase will be activated for empty

lanes. In High-traffic scenario, a lot of vehicles is coming from all directions, therefore the policy of the STL is arguably an efficient one.
In NS and EW-traffic scenarios, the lanes that need a longer green light phase duration are not prioritized.This will create longer queues resulting in mediocre performances.

As an example for future comparison, graphs similar to Figure 27 will be used. The blue line represents the average queue length (in vehicles) experienced over the length of the evaluation steps. The data of the graph is relative to the High-traffic scenario because, as it will be shown, this scenario is the one that brings out the most differences between agents.

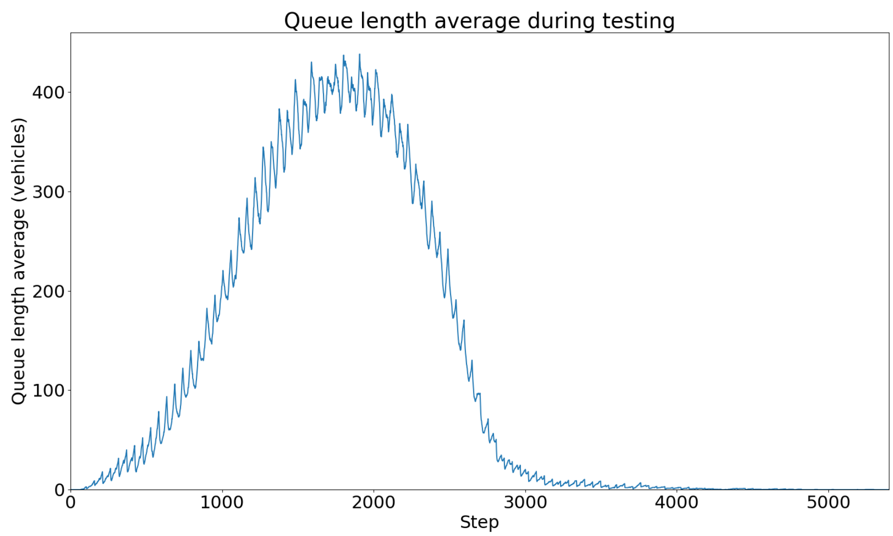


Figure 27: Average queue length of vehicles per steps over 5 episodes with the STL in High-traffic scenario.

5.4 Enhanced agent performances

This first part of the thesis work concerned the implementation of the *NDR* as the new state representation with the new NN structure (presented in 4.1.2) and accelerating the training phase of an agent.
Multiple experiments have been simulated to converge eventually to the best (or, at least, a better) solution.

Figure 7 shows (and recall) the model characteristics for training the agent that will not be changed over the following results. (Vidali's work already consisted in tuning these hyperparameters).

Characteristics	Values
Number of episodes	100
Number of steps in an episode	5400
γ	0.75
Learning rate	$1e-3$
Training epochs	800
Batch size	100

Table 7: Models characteristics

5.4.1 New state representation and Neural Network

Considering the new structure of the NN, multiple attempts have been realised to find the right number of neurons per layer. The goal is to improve the results but without increasing too much the training time. There is a clear correlation between the two.

In Table 8, three different models experienced have been described.

Model Name	Layers width
Small	320 - 200 - 200 - 100 - 100 - 50 - 4
Big	320 - 600 - 600 - 300 - 300 - 150 - 4
Best	320 - 480 - 480 - 240 - 240 - 120 - 4

Table 8: Experiments characteristics of the neural network structure

Agent $Agent^{Small}$ **with the** ***Small*** **model.**

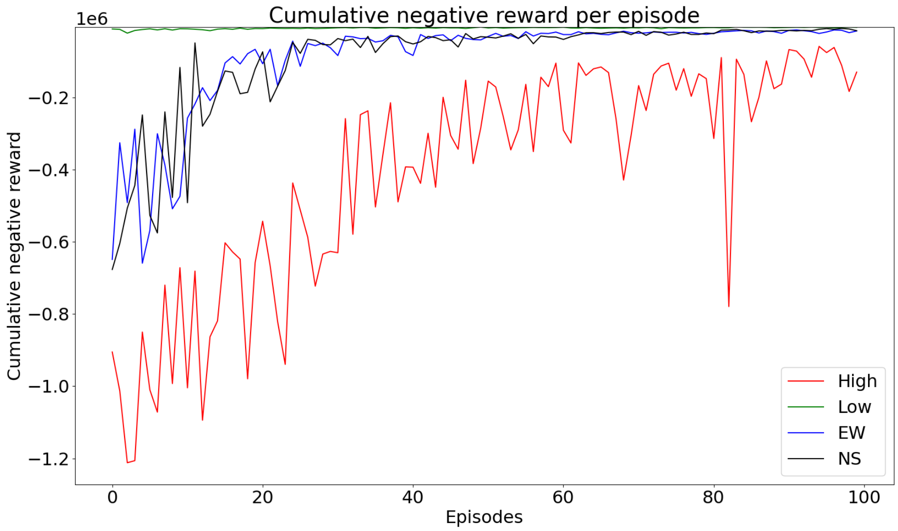


Figure 28: Cumulative negative reward per episode of $Agent^{Small}$ training

	Low	STL (%)	**High**	STL (%)	**NS**	STL (%)	**EW**	STL(%)
nrw_{av}	-6069.6	+1	-101352.4	-22	-10362.4	-87	-12754.2	-84
twt	10762.4	-32	506959.0	-16	36886.4	-75	40555.0	-72
awt/v	29.4	-21	107.4	-4	26.8	-57	23.4	-61

Table 9: Results obtained with the $Agent^{Small}$ agent

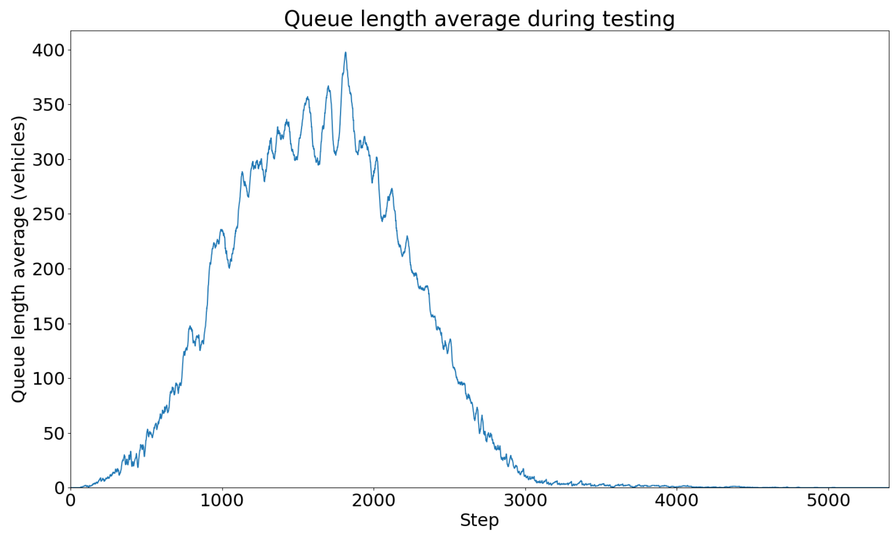


Figure 29: Average queue length of vehicles per steps over 5 episodes with the $Agent^{Small}$ agent in High-traffic scenario

Agent $Agent^{Big}$ for the *Big* model.

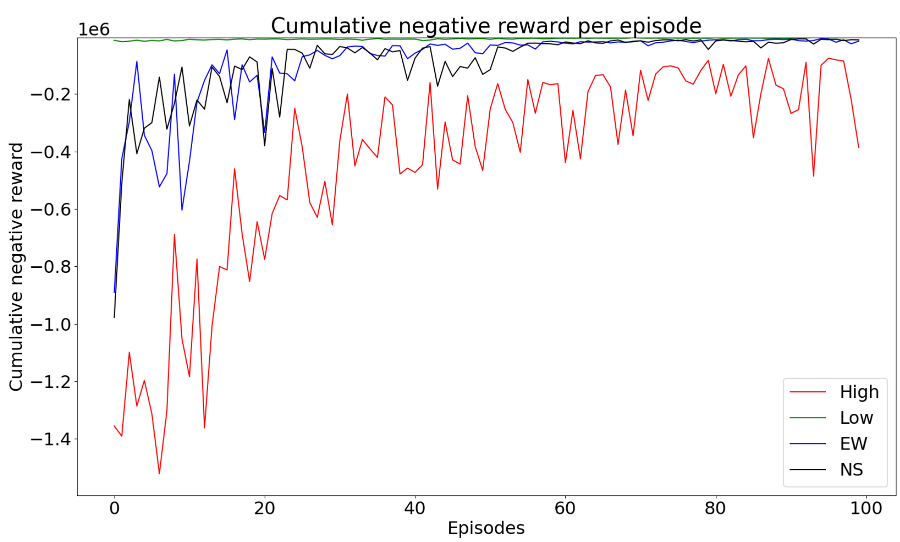


Figure 30: Cumulative negative reward per episode of $Agent^{Big}$ training

	Low	STL (%)	**High**	STL (%)	**NS**	STL (%)	**EW**	STL(%)
nrw_{av}	<u>-5142.6</u>	-14	-468424.0	+261	<u>-8760.2</u>	-111	-14624.4	-82
twt	<u>9741.6</u>	-38	869506.4	+43	<u>30306.0</u>	-79	45755.2	-68
awt/v	26.2	-30	105.5	+4	25.1	-59	24.8	-59

Table 10: Results of the $Agent^{Big}$ agent

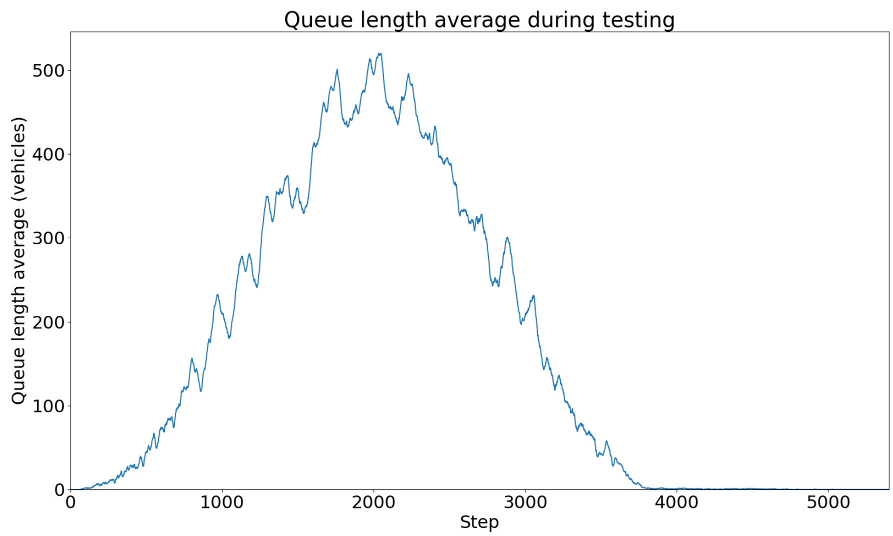


Figure 31: Average queue length of vehicles per steps over 5 episodes with the $Agent^{Big}$ in High-traffic scenario

Agent $Agent^{Best}$ for the *Best* model.

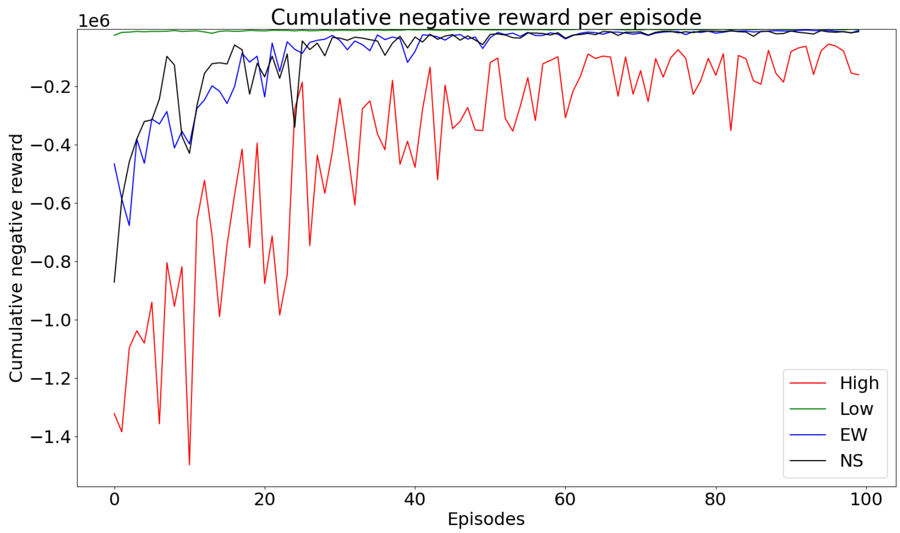


Figure 32: Cumulative negative reward per episode of $Agent^{Best}$ training

	Low	STL (%)	**High**	STL (%)	**NS**	STL (%)	**EW**	STL(%)
nrw_{av}	-6240.8	+4	-83899.4	-35	-11679.4	-86	-12713.8	-84
twt	12070.8	-23	469253.2	-23	37891.2	-74	43813.4	-70
awt/v	25.7	-31	104.4	-7	21.1	-66	25.9	-57

Table 11: Results of the $Agent^{Best}$ agent

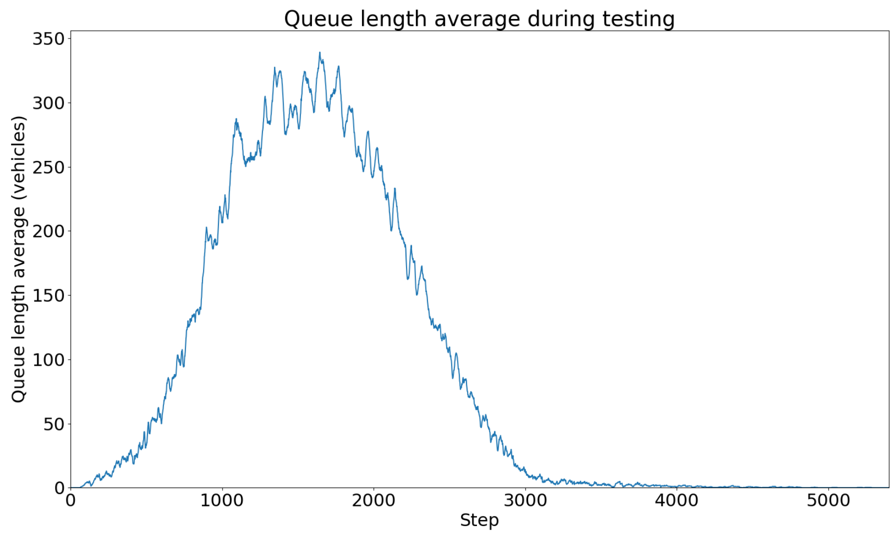


Figure 33: Average queue length of vehicles per steps over 5 episodes with the $Agent^{Best}$ in High-traffic scenario.

Cumulative negative reward analysis.

Figures 28, 30 and 32 represent the cumulative negative reward per episode of the 3 trained agents (respectively $Agent^{Small}$, $Agent^{Big}$ and $Agent^{Best}$) on the 4 different scenarios $\{Low, NS, EW, High\}$. From what can be observed a few things are worth describing:

- Each learning curves are converging which is a mark that the agents are able to learn from past experiences.
- The learning curves of $Agent^{Small}$ are more chaotic compared to the others models which is especially illustrated by the increasing peak around the 80^{th} episode.
- According to the profile curves of $Agent^{Best}$ compared to the other ones, it seems to learn to act better. First, the learning curves of the different scenarios are converging faster and, second, there are less oscillations (or at least their amplitude is smaller) from the 40^{th} episode until the end.

Tables analysis.

Table 9 brings to light that $Agent^{Small}$ performs better in each and every situation compared to the STL results (reported in Table 6). In NS and EW-traffic scenarios the agent achieves an increase of more than 50 %. However, in the High scenario, the results are just slightly better (around 5 to 20 %).

$Agent^{Big}$ results produced (Table 10) show outstanding improvements in NS and EW scenarios which is an excellent signal of a good action choice policy. In Low-traffic, performances are also better than STL (and even better than the previous agent $Agent^{Small}$). Nonetheless, the agent is doing poorly in High-traffic with an increase of 43 % in the total waiting time (the lower the better).

Table 11 saves the best performances of the agents. Moreover, it is a good trade-off in terms of time and efficiency. And in High-traffic, the policy gives him the best overall results with respect to every metric, which manifests a better abstraction in difficult scenarios.

Average queue length analysis.

Two types of observations on the testing curves can illustrate what has been deduced above.

1. **the peak**: the higher the peak is, the more bad decisions have been made which led to accumulating vehicles on lanes.

2. **the spread**: Again, the greater the spread is, the more bad decisions have been made by the agent. But it may not necessary be correlated with the peak since it could have a smaller peak compared to other curves but a wider spread (and inversely). This would illustrate multiple bad decisions made which help not traffic to flow.

3. **smoothness of the curve**: the greater the oscillations and roughness, the greater the increase in vehicle queues (at these particular steps).

In our study case, Figure 29 exhibits that the peak is smaller than the STL one (27). The spread is comparable. However, some big oscillations around the 2000^{th} step can be remarked which is probably led by the peak of vehicles generation combined with the agent not choosing the best actions at that time. With respect to the three points enumerated above, Figure 31 expresses that the peak is higher, the spread is wider and the presence of noticeable oscillations explains the poor results obtained in this High scenario by $Agent^{Big}$. Finally, compared to the other ones, Figure 33 indicates the best curve profile for this particular scenario.

The underlined results in tables emphasize the best results obtained with respect to the scenario and the metric in the previous set of experiments.

Instead of testing «manually »and finding a good overall end result, it could have been interesting to use a Machine Learning approach through an hyperparameter optimization framework to automate hyperparameter search as Optuna [57] proposes. It has not been performed because it is not the purpose of this work and it would have been time-consuming while the amelioration of the model would probably not have been worth it.

5.4.2 Multiprocessing training time performances

The run time of the different investigations are reported in Table 12. The implementation *With PR* has permitted to reduce the training time by almost 3 times !

	Without	With	With PR	With PR big
Run time	18h06m35s	15h11m08s	**06h44m36s**	07h04m21s

Table 12: Running time of different approaches

The results reported have been realised using a computer equipped with a GeForce RTX 2070 with Max-Q Design and 12 CPUs.

	Low	STL (%)	**High**	STL (%)	**NS**	STL (%)	**EW**	STL(%)
nrw_{av}	-2805.2	-53	-93991.4	-28	-8567.4	-90	-7925.0	-90
twt	6907.6	-56	480404.6	-21	36505.2	-75	33492.6	-77
awt/v	19.3	-48	99.6	- 11	21.9	-65	22.9	-62

Table 13: Results calculated for agent $Agent^{Fast}$ with the faster approach

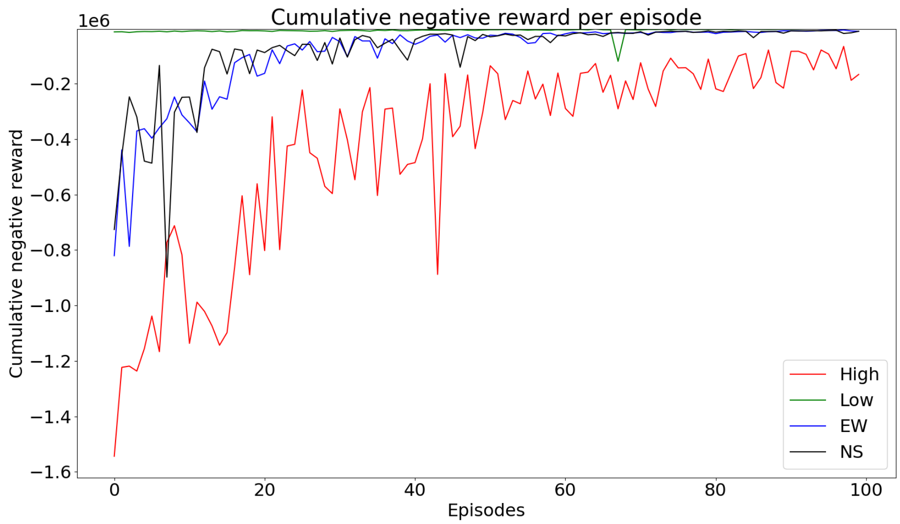


Figure 34: Cumulative negative reward per episode of $Agent^{Fast}$ training

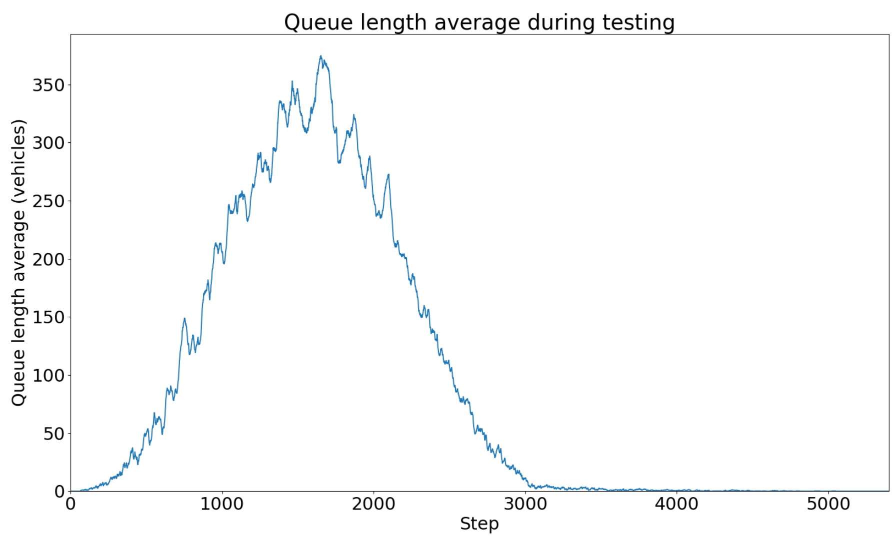


Figure 35: Average queue length of vehicles per steps over 5 episodes with the $Agent^{Fast}$ agent in High-traffic scenario.

Cumulative negative reward analysis.

Except for the peak in High-traffic around the 50^{th} episode and the one around the

10^{th} episode for the NS scenario, differences in learning curves between Figure 34 and 32 are negligible since convergence and smoothness criteria are kept.

Table analysis.

Results obtained by $Agent^{Fast}$ (Table 13) are quite similar with the ones reported in Table 11 ($Agent^{Best}$). In fact, similar results for Low, NS and EW scenarios can be noticed while in High-traffic a small decrease is noted (i.e. increase in terms of total waiting time) . However, the loss in performance (in High-traffic) is very low with respect to TLS results.

> Implicitly, it tells that training the agent's model on the experiences memorized by the agent after a simulation is the same as (over-)training the agent after every scenarios. This could lead to less adaptation in other cases/scenarios that the agent would have never encountered.

Average queue length analysis.

According to the three criteria previously enumerated, the analysis between Figure 33 and Figure 35 is, again, comparable. The maximum peak accumulates more than 350 vehicles but the spread is similar to the previous ones and the oscillations are minors (except for the one around the 1750^{th} step).

5.5 Variation of green phase duration

These set of experiments investigate how the increasing of green phase duration in training an agent modifies or affects the results.

The two agent cases that will be set forth are, first, an increase from 10 to 15 seconds (the agent will be denoted as $Agent^{15s}$) and, second, from 10 to 50 seconds ($Agent^{50s}$). There are the ones that seem the most relevant to comment among the different experiments made (from 20 to 60 seconds, ten by ten).

Agent $Agent^{15s}$: 15 seconds.

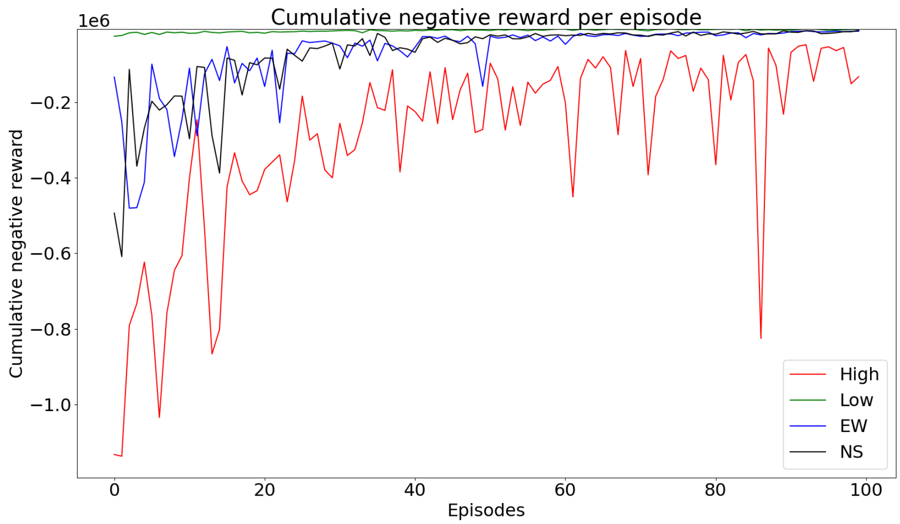


Figure 36: Cumulative negative reward per episode of $Agent^{15s}$

	Low	STL (%)	**High**	STL (%)	**NS**	STL (%)	**EW**	STL(%)
nrw_{av}	-8210.2	+37	-81513.0	-37	-15842.0	-81	-13323.2	-83
twt	14799.2	-6	453436.4	-25	47884.4	-67	39145.6	-73
awt/v	39.7	+6	95.1	-15	27.9	-55	30.7	-49

Table 14: Results obtained with a modification of the green phase duration up to *15 seconds*

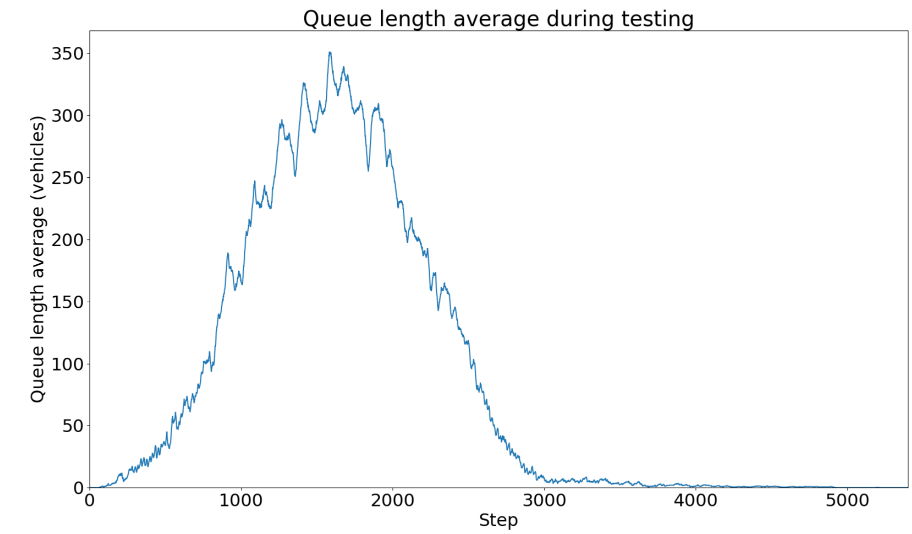


Figure 37: Average queue length of vehicles per steps over 5 episodes with the $Agent^{15s}$ agent in High-traffic scenario.

Agent $Agent^{50s}$: 50 seconds.

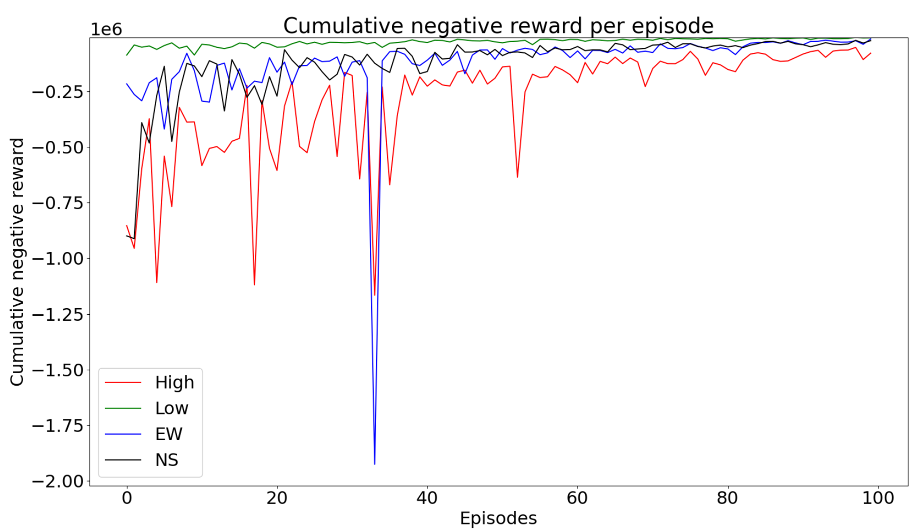


Figure 38: Cumulative negative reward per episode of $Agent^{50s}$

	Low	STL (%)	**High**	STL (%)	**NS**	STL (%)	**EW**	STL(%)
nrw_{av}	-10573.0	+77	-66415.8	-49	-21757.2	-73	-32287.6	-59
twt	30953.2	+97	373177.2	-39	76097.0	-47	91303.8	-36
awt/v	74.8	+101	78.4	-30	55.9	-9	63.6	+6

Table 15: Results obtained with a modification of the green duration up to *50 seconds*

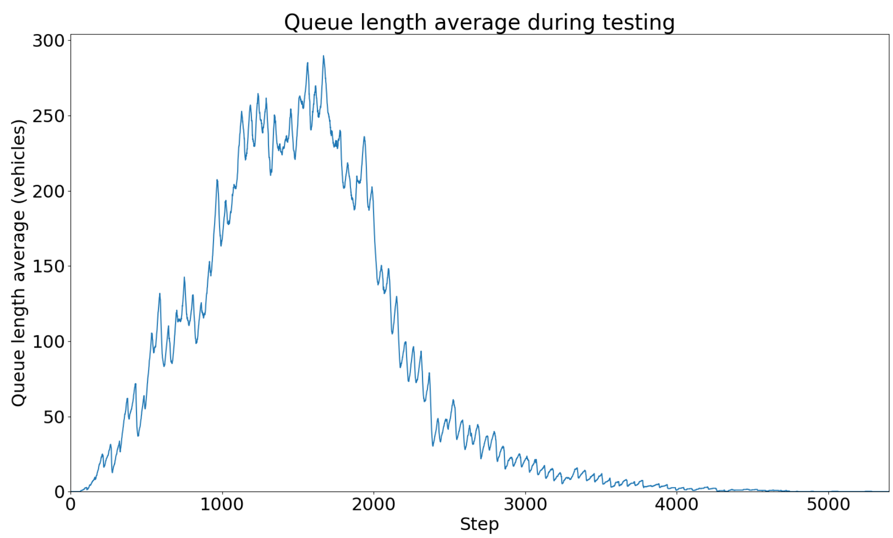


Figure 39: Average queue length of vehicles per steps over 5 episodes with $Agent^{50s}$ in High-traffic scenario.

Cumulative negative reward analysis.

Figure 36 is differentiable from the previous experiments because the first point of each scenario curves is higher (the higher the better) which brings to the fore that the policy that the agent generates at the very beginning appears better. But, in High-traffic, some peaks are degrading the smoothness of the learning curve. Figure 38 is also interesting to inspect because until the late 50^{th} episode, the agent performance is dropping multiple times (and dramatically around the 35^{th} episode) but after the 60^{th}, less roughness can be distinguished on the different curves.

Tables analysis.

Table 14 show how the small change of +5 seconds in the duration of green light

affect the different scenarios. In Low-traffic it led to the similar problem described in the fixed cycle case. Indeed, the less vehicles are passing through the intersection and the more time you give to green light phase, the more vehicles will wait even if good decisions are made by the agent. In the NS and EW scenarios, compared to the faster approach (Table 13) there is a decrease in terms of results. But the most noticeable change is an upgrade in the High-traffic.
And this dynamic of changes are exacerbated in the case of 50 seconds time (Table 15). In fact, in Low-traffic it is grossly underperforming compared to the STL.

Average queue length analysis.

Since the first experiments realised, Figure 39 show how $Agent^{50s}$ develops the best policy for the High-traffic. The peak is under 300 vehicles and the queue length drops before the 2500^{th} step where, usually it is around the 3000^{th} for best cases (as in 37).

> In Appendix C, the results of the 60 seconds time variation show how too much time degrades all of the different scenarios results.

> The important information that points out this set of experiments is that for a real case scenario the need to make the agent capable of adapting this variable (the duration) is necessary. Indeed, the agent is already implicitly able to change it by choosing multiple times the same action but it is not sufficient. Especially, in High-traffic giving the agent a longer green light time would increase the overall performances on this particular scenario **without even changing the model** !

5.6 Introduction of artificial queues

Even if the introduction of artificial queues should have, in theory, affected only the High-traffic scenario and possibly the NS and EW ones. However, in practice, it introduces additional changes in the SUMO internal working mechanisms, so scenarios that are comparable (especially the Low-traffic) could become not really comparable.

To be clear, Low-traffic scenarios *without and with artificial* queues have different results even if the latter should not be sufficiently significant to cause observable changes in the intersection. Sumo documentation explains that «routing »can be affected. And some of the observed quantities rely on the exit and the average speed (which is reduced due to the stop signs of the outcoming junctions) from the overall scenario. With the artificial queues, even though the intersection does not work worse, cars can take more (or less, but it is rarer) to complete the routing and, therefore, vacate the overall scenario.

Moreover, let's notice that artificial queues are introduced to observe how the agent will behave with this additional constraint which is, at least, perceived as noise introduction. And it could be interesting to dive a little further into its conception, but this would relate to traffic engineering task which is not our central problem since our goal is RL traffic control.

> This step would refer as "Face validation" in the process for validating agent-based simulation models [58].

In the same way as with STL, the evaluation of the STL with artificial queues (denoted as STL_{AQ}) performances are shown in Table 16. This reported data will be the standard baseline for the following agents performance to compare with.

	Low	**High**	**NS**	**EW**
nrw_{av}	-5985.8	-144427.4	-82265.2	-80653.8
twt	15703.6	650104.6	148309.2	146202.4
awt/v	35.9	112.7	56.5	53.1

Table 16: Results of Static Traffic Light with artificial queues (STL_{AQ})

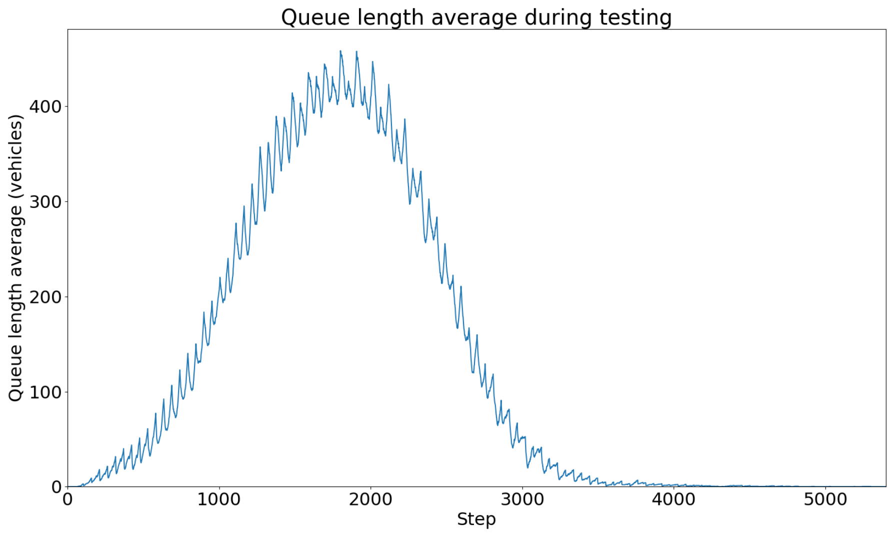


Figure 40: Average queue length of vehicles per steps over 5 episodes with the STL with AQ in High-traffic scenario.

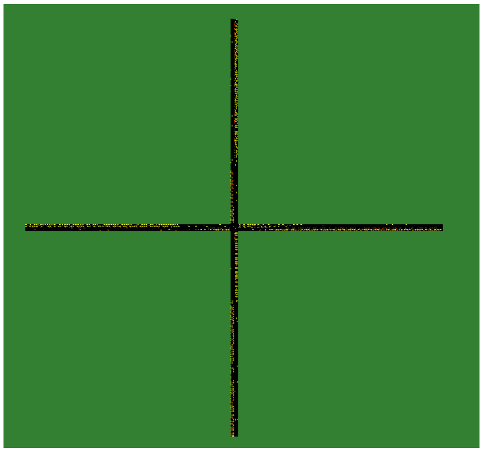
Figure 40 has to be set side by side with the one without artificial queues (Figure 27) for differences to emerge. However, there exists slight ones between both curves (that will bring out changes in the next set of experiments).

1. The left part of the bell curve is increasing slower and the right part is decreasing slower which creates a wider spread.

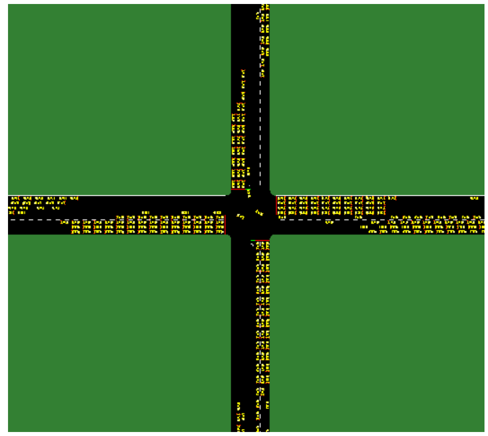
2. Oscillations of the right side are much present in this case.

Qualitative analysis on the impact of introducing AQs.

In order to help the reader to better understand how AQs can affect the overall simulation some screenshots are provided.



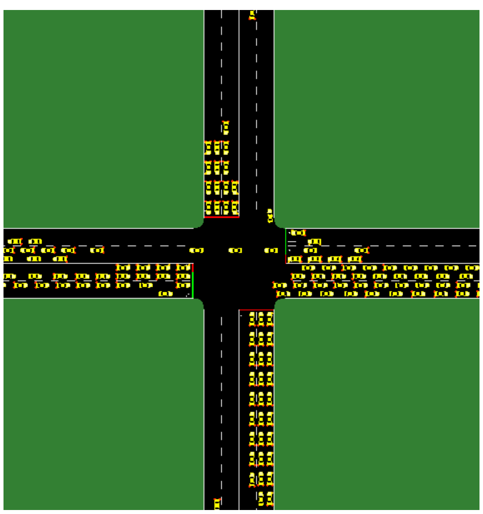
(a) Complete view



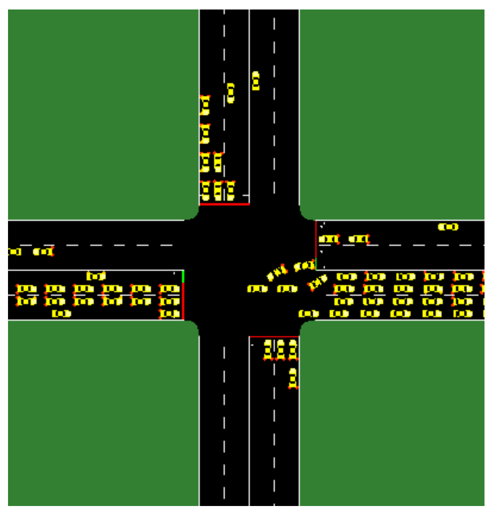
(b) Zoomed view

Figure 41: SUMO screenshots of two different views of the intersection in a simulation with artificial queues in High-traffic

Figure 41 provides two different views of a High-traffic situation where AQs provoke the accumulation of vehicles leading to long queues.



(a) Stopped vehicles at green light



(b) Blocked vehicles at the center of the intersection

Figure 42: Situation encountered when AQs are blocking the intersection, in High-traffic

Figure 42 aims at showing that, despite that SUMO internal mechanism handles the vehicles movements, collisions and intersection management, some problems can be encountered leading to an increase in the overall performances obtained.

In Figure 42a, the vehicles coming from East that want to go straight (the two central lanes) are stopped even if the green light is activated. SUMO handles the fact that the other side if full and do not want to block the intersection (which is an ideal solution since in a real-world case, some drivers will force this law).

But it is also interesting to observe that despite this internal rule, there are situations where it can be broken. Figure 42b show a situation where vehicles coming from East want to turn left to go to South are blocked by vehicles engaged in the center of the intersection. This situation increases, *de facto*, the waiting time of the blocked vehicles .

5.7 Loss performances comparison

The loss of the neural network has shown to influence the results in many Deep Learning applications. In DRL, it is sometimes harder to understand in which ways and how to predict a particular outcome. Because, the input data is not prepared and as known as in deep learning cases. These simulations aim at comparing and trying to discuss the implications of a good/bad loss choice in our particular study case between MSE, MAE and Huber (respectively the agent will be named $Agent^{MSE}$, $Agent^{MAE}$, $Agent^{Huber}$).

> Please note that $Agent^{MSE}$ corresponds to $Agent^{Fast}$ since MSE is the default loss function.

The experiments have been carried out with the artificial queues already implemented.

5.7.1 Mean Square Error loss

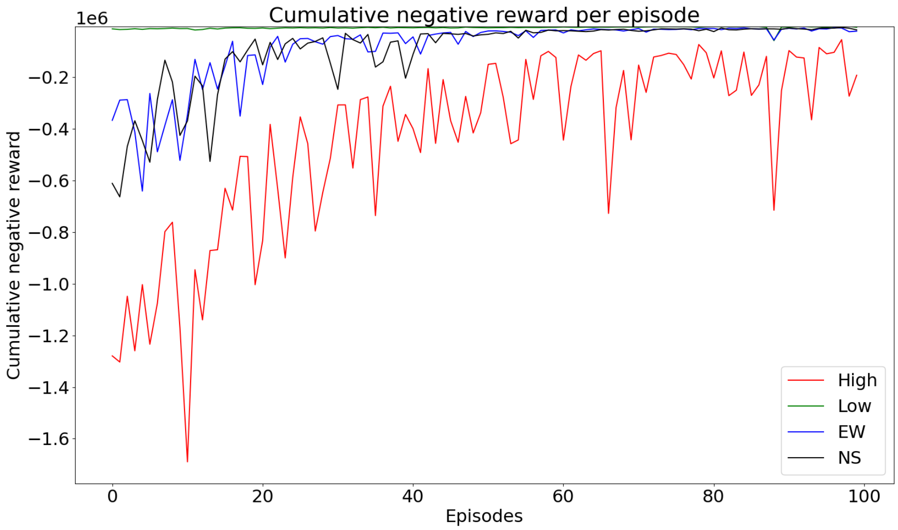


Figure 43: Cumulative negative reward per episode of $Agent^{MSE}$

	Low	STL_{AQ} (%)	**High**	STL_{AQ} (%)	**NS**	STL_{AQ} (%)	**EW**	STL_{AQ} (%)
nrw_{av}	-3479.8	-42	-287271.2	+99	-9136.4	-89	-9532.2	-89
twt	8117.0	-48	684511.0	+5	32233.4	-78	35565.0	-76
awt/v	17.8	-50	185.5	+65	16.2	-71	16.6	-69

Table 17: Results of $Agent^{MSE}$ with the Mean Square Error Loss

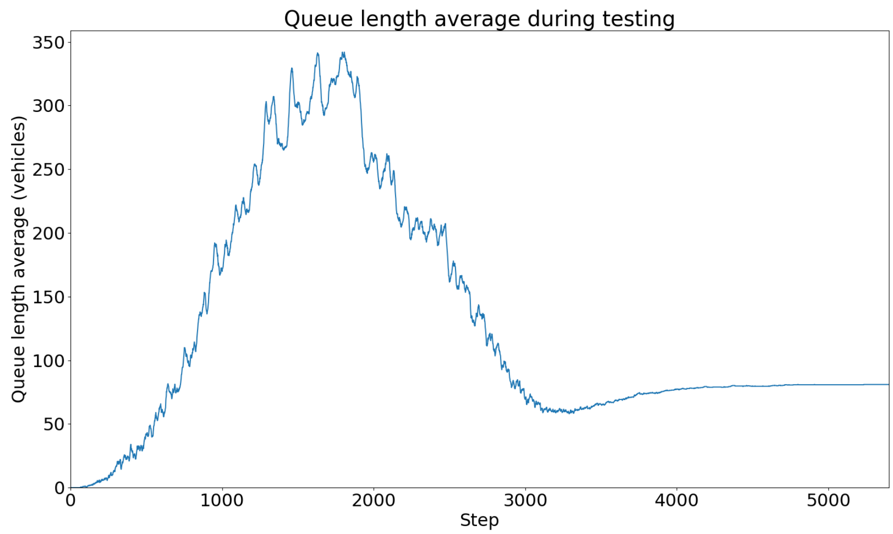


Figure 44: Average queue length of vehicles per steps over 5 episodes with the $Agent^{MSE}$ agent in High-traffic scenario.

5.7.2 Mean Average Error loss

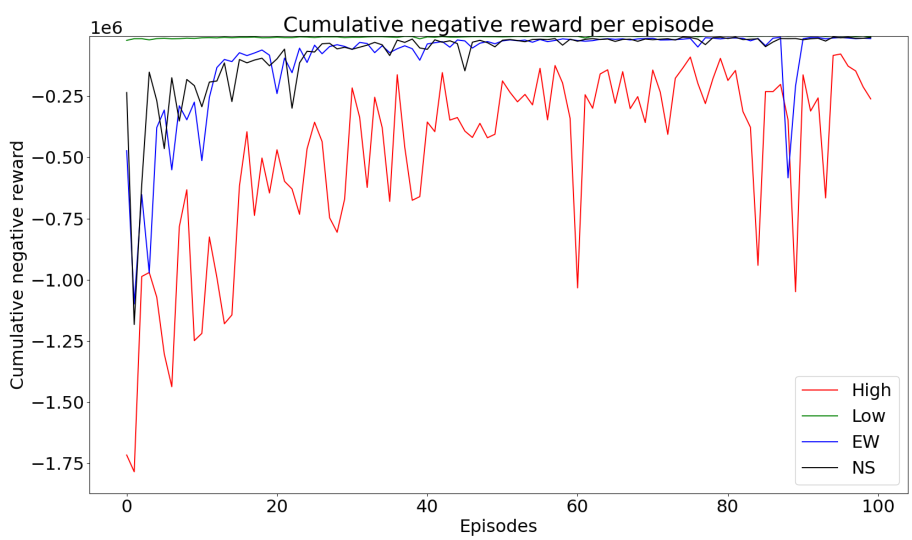


Figure 45: Cumulative negative reward per episode $Agent^{MAE}$

	Low	STL_{AQ} (%)	**High**	STL_{AQ} (%)	**NS**	STL_{AQ} (%)	**EW**	STL_{AQ} (%)
nrw_{av}	-3804.0	-36	-125444.6	-13	-10432.8	-87	-13145.8	-84
twt	8162.8	-48	574584.4	-12	39071.0	-74	43675.4	-70
awt/v	19.3	-46	115.7	+3	18.6	-67	22.1	-58

Table 18: Results of $Agent^{MAE}$ with the Mean Average Error Loss

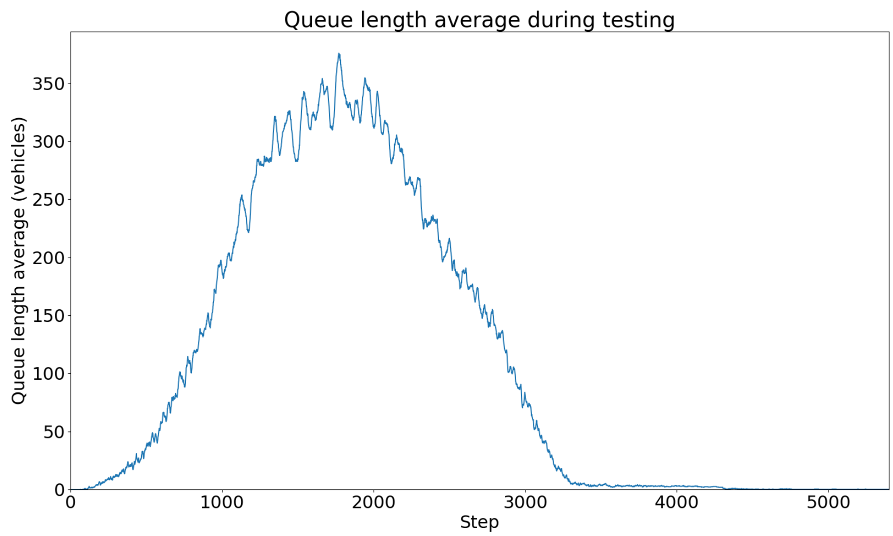


Figure 46: Average queue length of vehicles per steps over 5 episodes with the $Agent^{MAE}$ agent in High-traffic scenario.

5.7.3 Huber loss

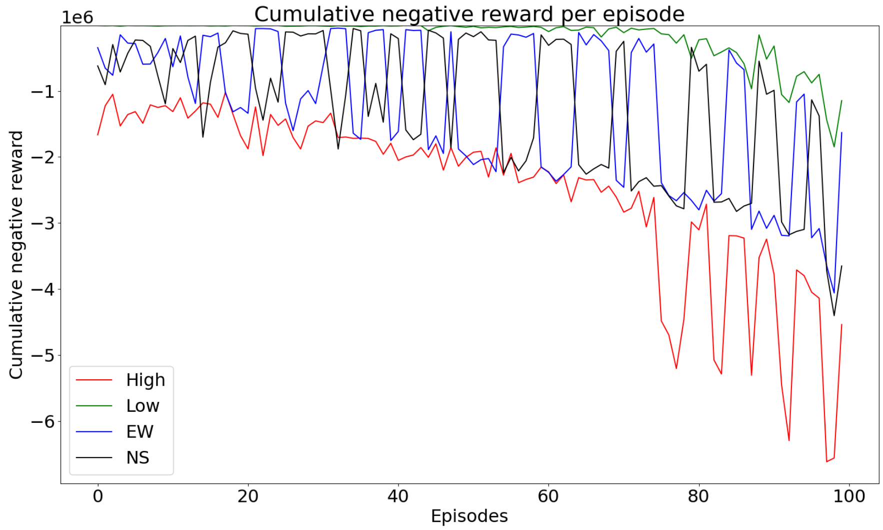


Figure 47: Cumulative negative reward per episode $Agent^{Huber}$

	Low	STL_{AQ} (%)	High	STL_{AQ} (%)	NS	STL_{AQ} (%)	EW	STL_{AQ} (%)
nrw_{av}	-1228917.2	+20431	-4525047.2	+3033	-1595315.0	+1839	-3625568.6	+4395
twt	1230886.4	+7738	4531240.2	+597	1597869.4	+977	3630383.0	+2383
awt/v	3047.2	+8388	1799.9	+1497	918.2	+1525	4191.5	+7794

Table 19: Results of $Agent^{Huber}$ with the Huber Loss

The value of δ in this experiment is 1.0.

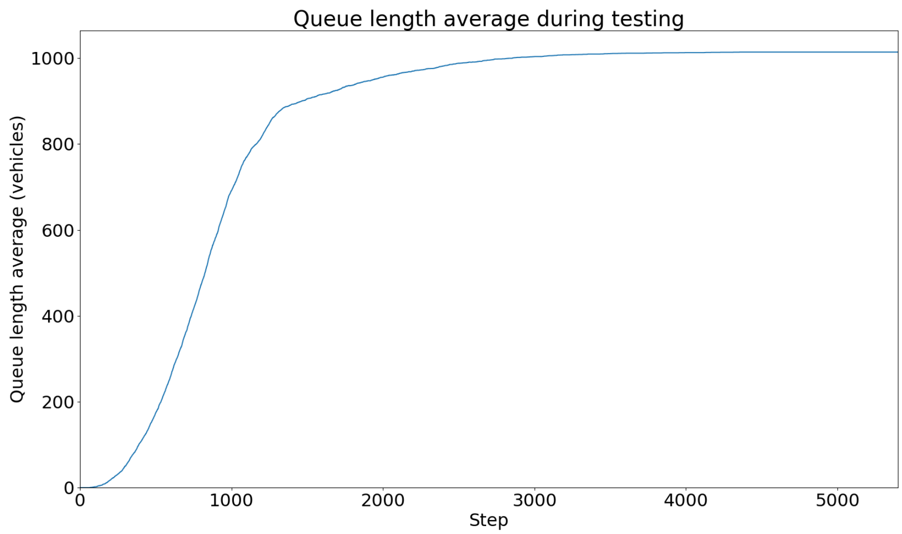


Figure 48: Average queue length of vehicles per steps over 5 episodes with $Agent^{Huber}$ in High-traffic scenario.

Cumulative negative reward analysis.

Figure 47 describes how the agent $Agent^{Huber}$ results are an exception among all the previous ones. Every learning curve is diverging and both the frequency and the amplitude of the oscillations are extraordinary (in the sens that they are not common).

When Figure 43 and Figure 45 are juxtaposed, it may not be obvious to conclude. In the case of $Agent^{MSE}$ despite some drops, the evolution of the reward given to the agent stays quite smooths. Whereas in the $Agent^{MAE}$ case, some instability is visible in the final 20 episodes.

Tables analysis.

Table 19 shows catastrophic results. After viewing the agent's behaviour on the SUMO GUI, it can be easily elucidated. The agent is not changing its action. It stays stuck on the NSA without initiating a different light phase than the one it chooses at first.

$Agent^{MSE}$ results compared to STL_{AQ} ones show that in Low, EW and NS cases, the agent still performs a good policy but not in High-traffic. The intersection can be blocked and it slows down the traffic flow.

However, $Agent^{MAE}$ manages to act better in every scenario (except for some extreme cases of very high vehicles waiting times which can explain why the average waiting time per vehicle has climbed above the STL_{AQ}).

Average queue length analysis.

Figure 46 details a curve profile which is similar to the ones observed in the first simulations. With the exception that the right part of the bell curve is decreasing slowly (it ends after the 3000^{th} episode) because of the AQs (but without disturbing $Agent^{MAE}$).

Figure 48 is an extreme case where the vehicles are blocked at the intersection without the possibility to quit the simulation. Then, the cars are accumulating indefinitely.

And Figure 44 shows an interesting in-between case, where the average queue length curve is common until the agent policy seems to be broken which translates a difficulty of the agent to make vehicles flow through the intersection at some point (around the 3000^{th}). It deteriorates the end results.

Qualitative analysis.

Between $Agent^{MSE}$ and $Agent^{MAE}$, the differences can be elucidated by the fact that around the 3000^{th} episode, AQs are creating situations where there are blocked and stopped vehicles... in only the first case !

It seems that $Agent^{MAE}$ let pass through the intersection less vehicles that $Agent^{MSE}$. When comparing the two simulations side by side, there are always more vehicles waiting before passing the intersection with $Agent^{MSE}$ than with $Agent^{MAE}$. And, intentionally or not, the fact that more vehicles are waiting **before** crossing the intersection (at the traffic lights stop lane) helps to prevent the fact that the accumulation of vehicles on the other side (after the intersection) will disturb the traffic flow.

However it is hard to tell if it is on purpose or if it is just the consequences of lesser good decisions leading to an overall good situation.

> The reason why Huber loss has been chosen is because, in theory, it combines good properties from both MSE and MAE. Withal, the problem with Huber loss is that it needs to find the right δ by training it as an hyperparameter. This iterative process has not been realised and it is surely a first reason why the results are so bad compared to the precedent ones.

The main point of this set of experiments is to show that, generally, the choice of a good loos needs to be more considered in deep reinforcement learning. Some reflections or, at least, deeper analyses of the possible outcomes might be interesting to consider.

With vs. Without AQs

Do agents, trained on simulations *with artificial queues*, perform better in simulations *without artificial queues* ?

The answer seems obvious but it might not be the case. The purpose of these experiments is to have the certitude (or, at least data that could confirm the intuition) that an agent is able to act better on simpler scenarios (and, therefore, able to generalize better).

	Low	STL (%)	**High**	STL (%)	**NS**	STL (%)	**EW**	STL (%)
nrw_{av}	-3469.6	-42	-88277.6	-32	-9587.4	-88	-10927.0	-86
twt	8073.6	-49	453496.0	-25	33759.8	-77	36843.8	-74
awt/v	21.5	-42	93.3	-16	22.9	-63	23.5	-61

Table 20: Results of $Agent^{MSE}$ on simulation without artificial queues

	Low	STL (%)	**High**	STL (%)	**NS**	STL (%)	**EW**	STL (%)
nrw_{av}	-3883.4	-35	-115363.2	-11	-10012.8	-88	-12018.8	-85
twt	8276.8	-47	514309.6	-15	37748.6	-95	43481.2	-70
awt/v	22.4	-40	112.9	1	21.9	-64	26.4	-56

Table 21: Results of $Agent^{MAE}$ on simulation without artificial queues

From what Tables 20 and 21 exhibit, an agent trained on a scenario with AQs performs better, in overall, on a scenario without AQs. This conclusion will make the next experiments stick with artificial queues.

5.8 Agents performance overview

In order to summarize the work realised so far in the previous experiments, here is a recap of the decisions taken. Starting from a baseline agent, improvements have been added up resulting in $Agent^{Fast}$. It is better than STL and faster to train than the previous implementations. With the modifications applied to $Agent^{Fast}$, tests have been realised in order to prove the need of modifying the green duration time depending on the scenario. However, this parameter has not been changed since it probably requires more quantitative results.

Then, starting from the new STL measurements which took into account the introduction of artificial queues (STL_{AQ}), three type of agents have been trained resulting in $Agent^{MAE}$ having the best overall performances.

But for the following part MSE loss will be kept as it has been shown that, depending on the use case it seems to have a more balanced policy. Therefore for the passage to a multi-agent perspective, please keep in mind that it will stock with $Agent^{MSE}$.

Both the green light duration and the choice of the loss function might be seen as possible future works.

> From what have been analysed so far, more variations and improvement about the modifications and design of the agent can definitely be tested in the future.

6 Stepping up: two-intersections simulation

On the basis of the add-ons presented in Section 4 and 5, a minimalist multi-agent approach will be proposed. The analysis will be conducted with a simulation where two agents manage the choice of a set of traffic lights each with the objective of optimizing the overall traffic efficiency.

The reason why the thesis approach has been centered, at first, on the one-intersection-only environment is because it was interesting to implement some modifications to test the limits of a DQL model on a single TSC. But traffic signal control tasks cannot be simplified at one set of traffic lights. The need to observe how the model would react in a two-intersections simulation (Figure 49) is needed to test the robustness the model. The willingness to be close to a real scenario will also help determine the feasibility of a large-scale multi-agent perspective.

Grasping the knowledge (state representation) and the modifications of the agent's learning mechanism (for instance comparing local and shared reward) will be beneficial in bringing more content to an eventual and future possibility to a real-world scenario implementation on a bigger scale.

Table 22 presents the terms used in this section.

Category	Notation	Meaning
Traffic	TL	Set of traffic lights, the only one in the one-intersection simulation, the left one in the two-intersections simulation
	2_TL	Second set of traffic lights, the right one in the two-intersections simulation
	N	North junction of TL
	E	East junction of TL, in the one-intersection and East junction of 2_TL, in the two-intersections
	2_N	North junction of 2_TL
	2_S	South junction of 2_TL
	W	West junction of TL
	S	South junction of TL
	AQ(s)	Artificial Queue(s)
	side-by-side	Environment where two intersections have been placed side by side
	joined	Environment where two intersections have been joined by the central lane
RL	a_1	Agent that controls the actions of TL
	a_2	Agent that controls the actions of 2_TL

Table 22: Notations Section 5

6.1 Environment design

Two new environments developed with NetEdit have been elaborated for future use cases. Before delineating the differences between them, some notations are worth defining for later use cases:

- TL is the left intersection
- 2_TL is the right intersection
- N, W, S are the edge directions of TL
- 2_N, E, 2_S are the edge directions of 2_TL

> TL is considered as the West direction of 2_TL and 2_TL is the East direction of TL

6.1.1 The *side-by-side* environment

Figure 49 presents the first environment (annotated) where two intersections have been added side-by-side.

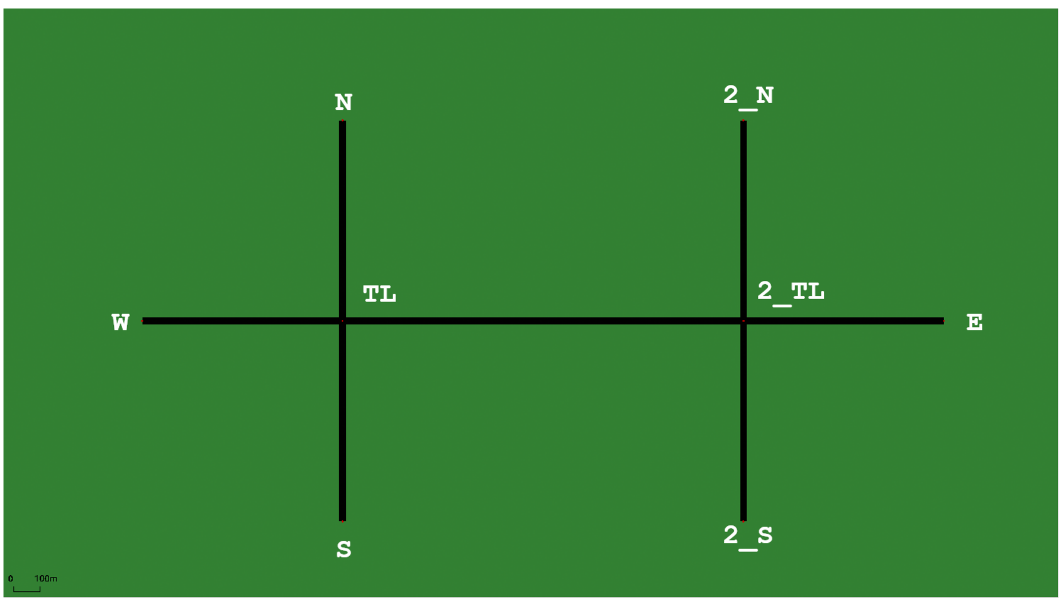


Figure 49: Captured image of the *side-by-side* environment realised with NetEdit including the annotations of the intersections and their dead-ends

The particularity of this environment lies in the central lane (from TL to 2_TL) which is 1500 meters long. The two 4-way intersections of the simulated environment have been placed in such a way that there are no overlap between the state representation of individual agents.

6.1.2 The *joined* environment

For a next use case (explained in 6.3.2), an alternative has also been developed and is displayed in Figure 50. Here, the central lane is 750 meters long which exposes it to the state representation of both agents.

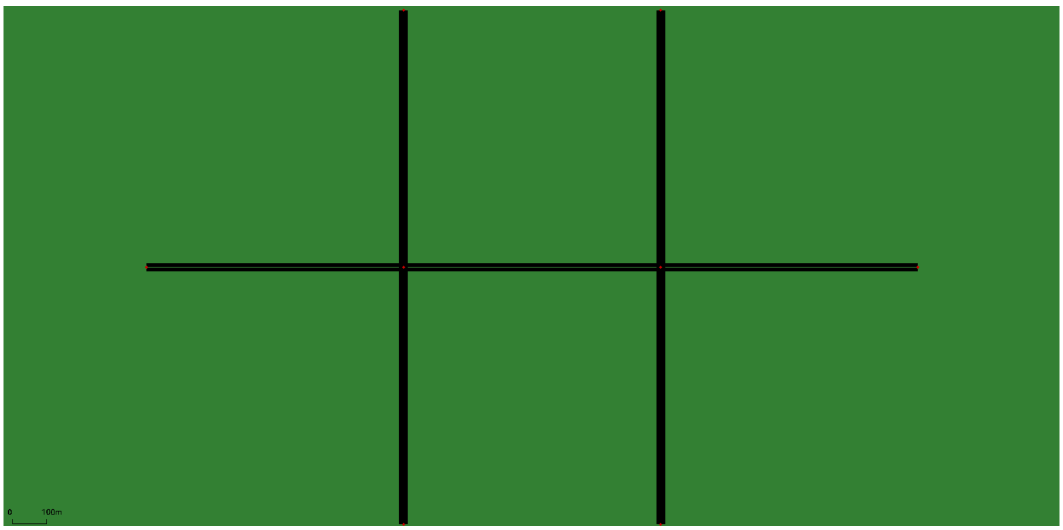


Figure 50: Captured image of the two-intersections environment with a shared central lane (*joined*). *Realised with NetEdit*

6.2 Artificial queues and scenarios

In this new simulation, some specifications have to be written down to explain how the AQs mechanism and vehicle scenarios have been adapted to this two-intersections simulation.

6.2.1 Artificial Queues

AQs are also implemented in every two-intersections experiments since it has been shown that an agent which obtains good results on a simulation *with* AQs performs better on the same simulation *without*. And, as it is mentioned in 4.2, the calibration of the waiting time at the stop lanes is needed. The aim is to diminish the unbalanced results between the NS-traffic and the EW-traffic. Indeed, going from E to W takes much longer for a vehicle than going from N to S (especially with the

side-by-side environment). Therefore, in the case of an NS-traffic, W and E outing waiting times are diminished and, respectively, in the EW-traffic, the ones affected are N, 2_N, S and 2_S (in the same way as 47).

It can be added that, as the free-flow phenomenon is also removed in the central road, the inability to cooperate between the two agents could conduct to traffic jams that will alter or, at least, disturb the traffic flow in both intersections .

6.2.2 Source and destination of vehicles

Compared to the source and destination of vehicles in the one-intersection simulation (3.3.4), the logic is the same with the only difference that the probability of the destination on the second intersection is uniform.

As an example, in EW-traffic, a car has a probability of 90% of coming from East or West and 10% from North or South (N, S or 2_N, 2_S) but if the car passes through the central lane it has a uniform probability to go to any possible destination. To be more concise, imagine a vehicle arriving from N (with a probability of 90% in NS-traffic), if, with a probability of 25% it turns left at TL and rides through the central lane, then it has an *equal probability* to end its path to 2_N, 2_S or E.

6.3 Experiments

The experiments that will be evaluated in the next section are described below. It takes into account the two new environments and changes in the state and reward definitions.

6.3.1 Side-by-side independent DQN

For a starting point, the IDQN framework described in 2.1.3 is convenient because it will help us stay in reasonable training time frames without having the problems linked to the scalability.

Therefore, in a first attempt to observe how behave 2 agents interacting within the same environment, 2 DQNs have been instantiated. The *side-by-side* environment is used. And each intersection is associated to an agent (a_1 for TL and a_2 for 2_TL). Each one is unaware of what the other agent is doing. The reward is locally assigned and the state representation of each agent is not aware of any mutual information.

In other words, *each agent is unaware of what the other is doing and how the other agent's actions might affect itself.*

This approximation may miss the fact that interactions between agents affect the decision-making of each other.

6.3.2 Joined IDQN

In this second experiment, the same agent framework and characteristics are set up. But, the *joined* environment is implemented. Consequently, without changing anything to the state representation of the agents, they will be able to share mutual information. The central lane will be the East edge of the left agent and, at the same time, the West edge of the right agent.

6.3.3 Enhanced state representation

Inspired by [59], the idea is to introduce a shared knowledge: **the other agent's action**. The state becomes *composed*, when the agent uses information from its neighbors to compose the state information. The possible defect with this approach is that the other agent activity in the environment is perceived as noise because the other agent behavior is, at best, only partially predictable and, at worst, unpredictable. But it may also create some kind of collaboration between which could lead to a better understanding of the overall situation.

A bijective function h has been introduced where every element of $I = 0, 1, 2, 3$ has exactly one antecedent in A (action space defined in Section 2.2.4) by h in equation 55.

$$h \colon A \longrightarrow I \tag{55}$$

Consequently, the *NSR* of the agent a_1 updated with the action u of the agent a_2 becomes the vector:

$$\overrightarrow{NSR_k} = (\overrightarrow{N}, \overrightarrow{AS}, \overrightarrow{Awt}, \overrightarrow{NQ}, h(u)) \tag{56}$$

6.3.4 New local reward definition

The new multi-agent perspective brings the necessity to develop a new reward that takes into account various objectives by refining its definition.

The idea is to have an hybrid metric that could deal with both the measured queue length along each incoming lane and the cumulative delay of the vehicles (equation 57).

$$r_t = (c.twt_{t-1} + queue_{t-1}) - (c.twt_t + queue_t) \tag{57}$$

where $c = 0.2$ [veh/s] is a fixed factor that works as a trade-off coefficient, twt [s] is the cumulative delay of the vehicles, $queue$ [veh] is the measured queue length along each incoming lane.

This new definition has the advantage to take into account both traffic congestion and trip delay.

6.3.5 Shared reward

In this final experiment, the objective is to analyse the behavior of the agents learning ability in case they share the same reward. It could potentially lead to some collaborative behavior since both of them would be linked by that measure.

However it could also lead to some disturbances since a good action of a_1 would be negatively perceived if a_2's decision is so bad that the addition of both stays negative. Equation 58 refines the reward function presented in Equation 57 .

$$\begin{cases} r_t^{a_1} = r_t^{a_1} + \zeta . r_t^{a_2} \\ r_t^{a_2} = r_t^{a_2} + \zeta . r_t^{a_1} \end{cases} \tag{58}$$

where a_n is the agent n with $n = 1, 2$, $r_t^{a_1}$ the reward obtained by the agent a_1 at time t and ζ a discount factor which is set, arbitrarily, at 0.5.

The advantage of this partially shared reward is that it incorporates the previous reward taking into account multiple variables (traffic congestion and trip delay) while being partially correlated to both state and action of different agents.

7 Results and discussion: two-intersections simulation

In this section, the new definition of the two 4-way intersections baseline simulation will be debated. First, compared to Section 5, the results chosen to evaluate the performances of the tested *independent* and *decentralized* agents in this new environment, will be explained. Then, the study cases delineated in the previous section will be expressed in terms of comparable results.

7.1 Performance Metrics and Plots

Since an additional agent has been introduced some clarifications need to be expressed.

- **Performance metrics.** A simple addition is made between the results obtained by each agent with respect to the metrics described in Section 5.
- **Performance plots.** Again, the same type of plots will be analysed and described in the future sets of experiments with the difference that both the *individual* figures (relative to each agent performances) and the *overall* one (which is a not-so-simple combination between the two) will be debated. It will help to better apprehend the global and local trends of the agents policies.

> Appendix C add some specifications to the descriptions above.

7.2 Static Traffic Light Systems

Two different baseline STL systems cases have to be differentiated. Both of them have in common the two STL systems (one for each intersection). But they are also **dependent on the environment** on which they are working.

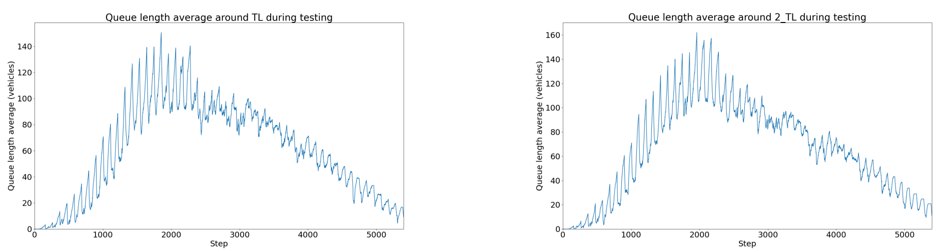
7.2.1 STLs performances on the *side-by-side* environment

The four usual scenarios have been reproduced, resulting in the evaluation performances reported in Table 23.

	Low	**High**	**NS**	**EW**
nrw_{av}	-10678.8	-1459829.8	-28892.6	-438308.0
twt	22603.4	2430910.4	73343.8	676792.8
awt/v	43.3	594.1	37.5	303.1

Table 23: Results of two Static Traffic Light systems in the two-intersections simulation on the *side-by-side* environment ($2STL^{side}$)

It is interesting to note that compared to the reasonable results obtained in Low, High and NS scenarios, the EW ones look very high. And it can be elucidated by the fact that vehicles coming from East or West junctions have a longer road to make until the outcoming edges, if the path they follow passes through the central lane (which is 1500 meters long).



(a) STL on the left intersection

(b) STL on the right intersection

Figure 51: Average queue length of vehicles per steps over 5 episodes of individual STLs in EW-traffic scenario.

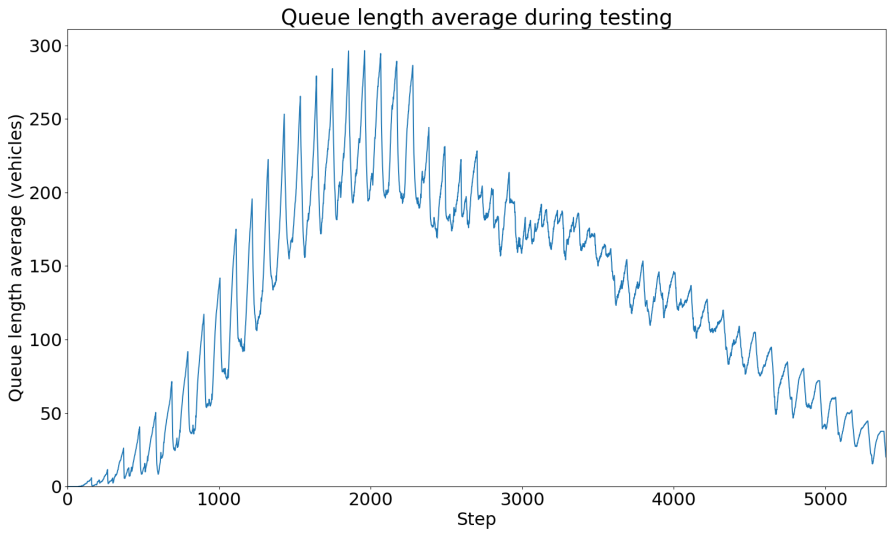


Figure 52: Average queue length of vehicles per steps over 5 episodes of both STLs in EW-traffic scenario.

Figure 52 represents the average queue length of vehicles per steps and over 5 episodes. The choice of of changing the scenario to analyse from High-traffic to EW-traffic has been taken. It will be shown that it is the most interesting one with respect to the agents performances obtained in the next experiments.
But, first, a few points needs to be introduced:

- The big oscillations are provoked by the fact that, while vehicle generation increase, traffic do not flow enough to let pass through enough vehicles. Then, after the peak of vehicle generations, the flow of vehicle is slowly driving away.
- But it remains few vehicles in the simulation after its end. It may be necessary in a future work to calibrate the number of steps or increase the traffic light duration.

Figures 51a and 51b correspond respectively to the average queue length around TL and 2_TL. They have the same profile than the combination of the two. But it will not be always the case as it will be shown later on.

7.2.2 STLs performances on the *joined* environment

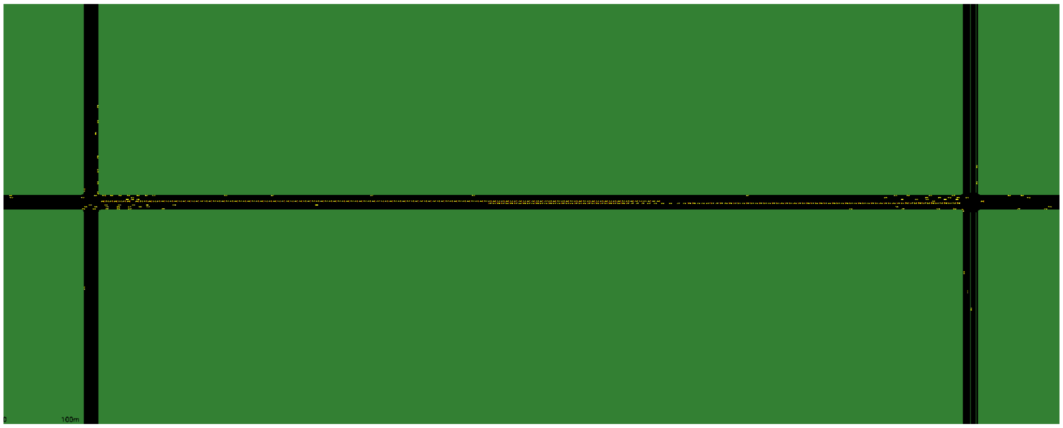
In a similar way, the evaluation performances of the two STL systems in the *joined* environment are reported in Table 24.

	Low	**High**	**NS**	**EW**
nrw_{av}	-14999.6	-2058239.8	-26753.0	-473225.8
twt	31316.6	3063997.0	79410.4	724456.8
awt/v	57.3	755.5	40.1	318.1

Table 24: Results of two Static Traffic Light systems in the two-intersections simulation on the *joined* environment ($2STL^{joined}$)

Compared to Table 23, the results obtained for each calculated metric are higher, in each and every scenario. It can be clarified by the fact that a shorter central lane (750 m) accumulates more vehicles. Indeed, from what it has been observed in the SUMO GUI (Figure 53), vehicles coming from 2_TL and wanting to turn left at TL (EWLA) and the ones coming from TL and wanting also to turn left at 2_TL (EWLA, also) are blocking a part of each intersection. Even if the EWA phase is activated some vehicles stay stopped because there is no more space to exit the intersection.

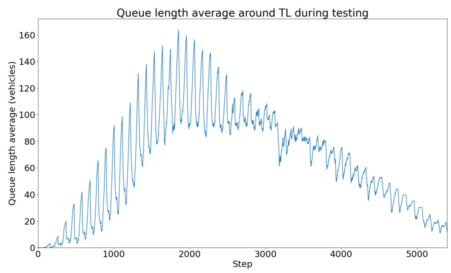
Figure 53a show how the accumulation of vehicles in the central lane is not problematic compared to Figure 53a.



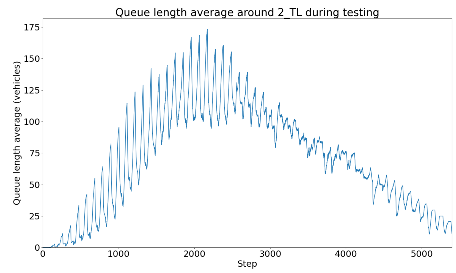
(a) Side-by-side

(b) Joined

Figure 53: SUMO screenshots showing the vehicles distribution on the central lane depending on the environment



(a) STL on left intersection



(b) STL on right intersection

Figure 54: Average queue length of vehicles per steps over 5 episodes of individual STLs in EW-traffic scenario.

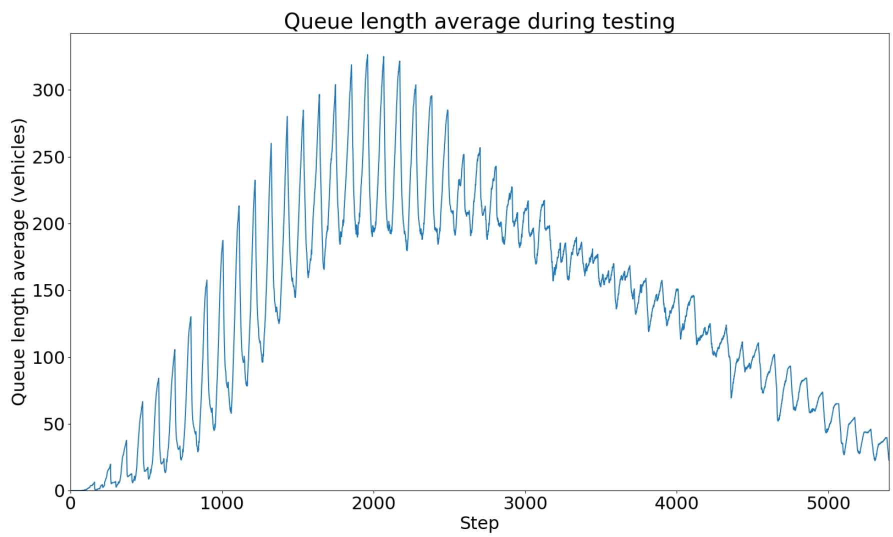


Figure 55: Average queue length of vehicles per steps over 5 episodes of both STLs in EW-traffic scenario.

Figure 55's curve is very close to the one observed in Figure 52 with the exception that there is a higher peak. This is due to the fact that the vehicles blocked by the central lane further increase the waiting lines coming into the intersections (as mentioned above).

In an ideal policy, each agent should keep activating the proper traffic light phases in order to make flow the vehicles stacked in the central lane.

7.3 Agents performances

In the following experiments, the search for an independent but collaborative framework between two agents will be pursued. The notations given by each agent depending on the scenario will respect the following rules :

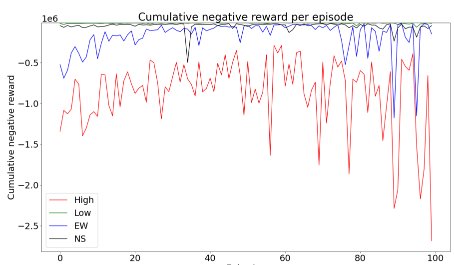
$$Agent_{id}^{name} \tag{59}$$

with *name* being a short nickname chosen arbitrarily for the experiment and $id = 1$ if the agent controls the left intersection and $id = 2$ otherwise.

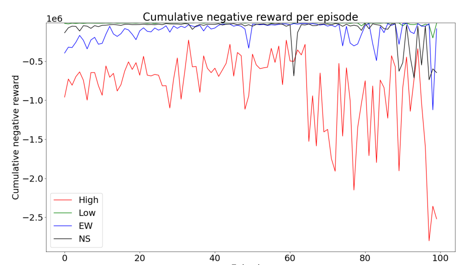
7.3.1 Independent DQN evaluation

The two first experiments which consist in training 2 IDQN agents, respectively, on the *side-by-side* environment and the *joined* environment have been merged into the same sub-section in order to help comparisons to be made.

1. Side-by-side.



(a) Agent $Agent_1^{side}$



(b) Agent $Agent_2^{side}$

Figure 56: Cumulative negative reward per episode of agents

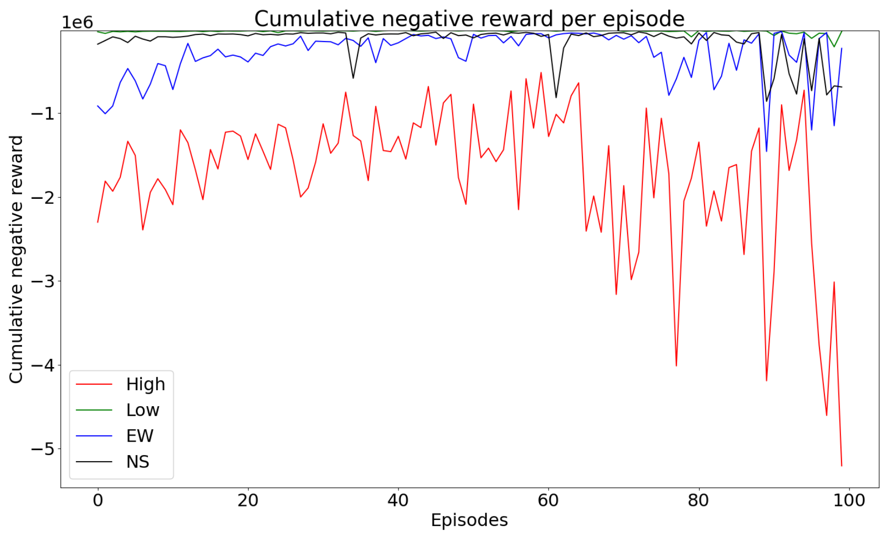
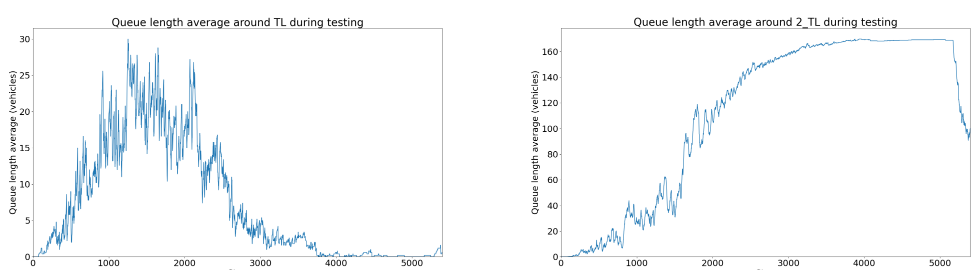


Figure 57: Cumulative negative reward per episode of both agents $Agent_1^{side}$ and $Agent_2^{side}$

	Low	$2STL^{side}$ (%)	**High**	$2STL^{side}$ (%)	**NS**	$2STL^{side}$ (%)	**EW**	$2STL^{side}$ (%)
nrw_{av}	-24289.0	+127	-4080268.0	+180	-65475.4	+127	-341953.6	-22
twt	29041.2	+28	5462122.6	+125	102870.8	+40	643156.0	-5
awt/v	53.2	+23	1375.1	+131	45.7	+22	42.7	-86

Table 25: Results obtained with two IDQN agents on the *side-by-side* environment



(a) Agent $Agent_1^{side}$ (b) Agent $Agent_2^{side}$

Figure 58: Average queue length of vehicles per steps over 5 episodes of individual agents in EW-traffic scenario.

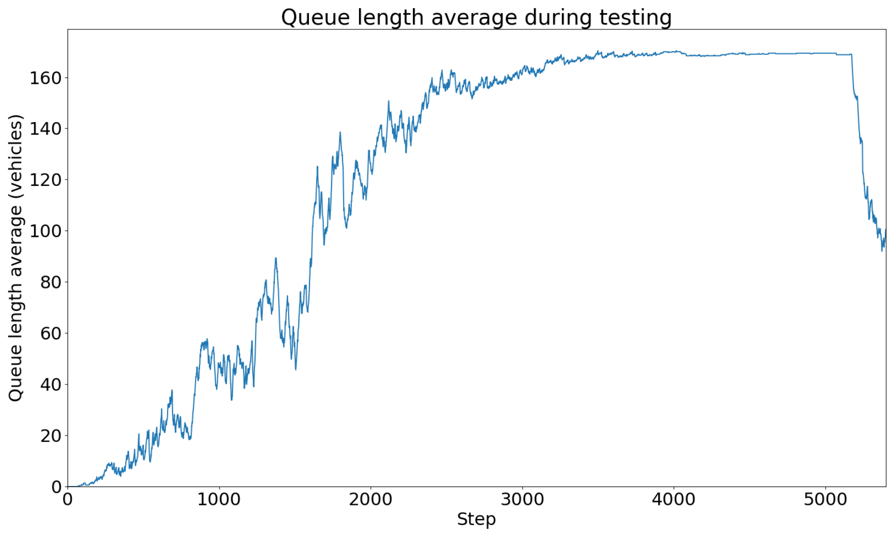


Figure 59: Average queue length of vehicles per steps over 5 episodes of both agents in EW-traffic scenario.

2. Joined.

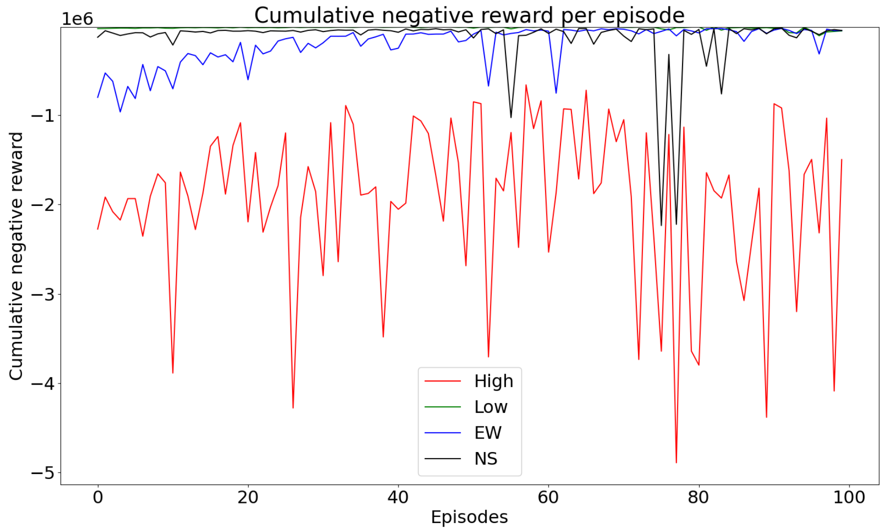
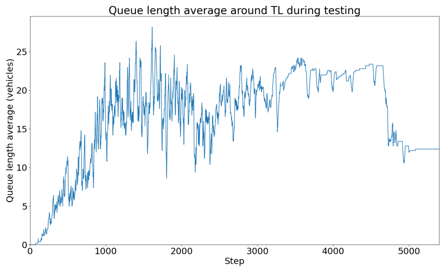


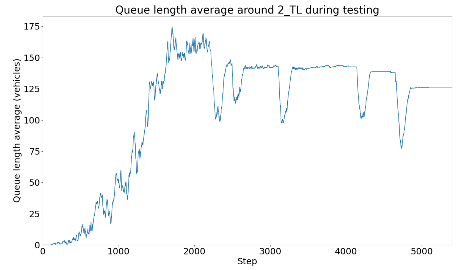
Figure 60: Cumulative negative reward per episode among both agents $Agent_1^{joined}$ and $Agent_2^{joined}$

	Low	$2STL^{joined}$ (%)	**High**	$2STL^{joined}$ (%)	**NS**	$2STL^{joined}$ (%)	**EW**	STL_2^{joined} (%)
nrw_{av}	-156234.0	+942	-4382996.6	+113	-91075.4	+240	-401182.0	-15
twt	156446.8	+400	6202645.2	+102	129959.6	+64	671110.2	-7
awt/v	331.6	+479	1895.8	+151	65.9	+64	72.3	-77

Table 26: Results obtained with the IDQN model on the *joined* environment



(a) Agent $Agent_1^{joined}$



(b) Agent $Agent_2^{joined}$

Figure 61: Average queue length of vehicles per steps over 5 episodes of individual agents in EW-traffic scenario.

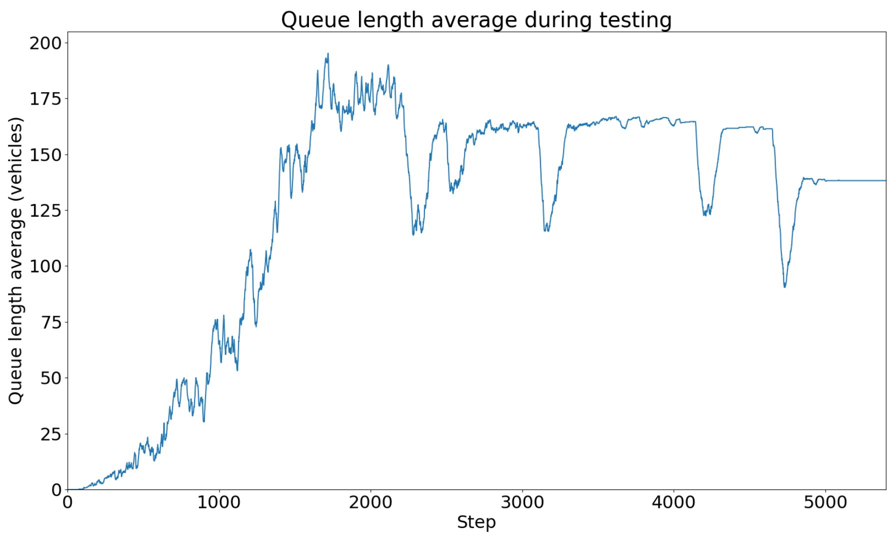


Figure 62: Average queue length of vehicles per steps over 5 episodes with the agents in EW-traffic scenario.

Cumulative negative reward analysis. Figures 56a and 56b have been displayed in order to put in evidence that Figure 57 is the addition of each curve. Thus, the trajectories of the curves are interesting in the sense that from the beginning of the agents training to the late $40^{th}s$ episodes, the convergence of the different scenario lines is observable. But after this point, instability and divergence appear which are synonyms of a poor ability of the agents to continue to collaborate.
In the case of agents training on the *joined* environment, Figure 60 make visible the

same characteristics for EW, NS and Low-traffic but in the High scenario, there is no clear evidence of a sign of convergence. On the contrary, the cumulative agents learning curve is highly unsteady from the beginning of the training.

Table analysis. The results in Table 25 show that both agents are not capable of collaborating and that it under-performs compared to the *2STL* ones (Figure 23). The same observations can be stated in the second experiment. Except that this time, the introduction of a common observation is detrimental. Indeed, the results are worse than the previous experiments (Figure 25). A particular attention is given to the Low-traffic that is completely unreasonable.

Average queue length analysis. An interesting fact that is common in both environments are that the agent controlling the right intersection ($Agent_2^{side}$ and $Agent_2^{joined}$) is the one that have hard times to manage the traffic flow. Indeed, Figure 62 which is the sum of both queue length average of $Agent_1^{joined}$ and $Agent_2^{joined}$ has the same profile that 61. And the same observation can be stated with Figures 59 and 58.

One last thing particularly intersecting when comparing the two global figures is that both from the 2000^{th} step the situations become more complicated. However in the *joined* environment the average queue is slowly decreasing burst after burst whereas in the *side* environment the average queue keeps increasing and then around the 5000^{th} step it decreases all of a sudden.

But looking at the maximum peaks, the end results is favorable in Figure 59 (with a max at 160 against 200 vehicles).

> Since the agents trained on the *joined* environment suffers from the metrics comparisons, the so-called *side-by-side* environment will be held up for the future investigations. Therefore *2STL* is now referring to $2STL^{side}$. However, in order to have more certainty in this choice it could have been interesting to evaluate agents, which have been trained on a particular environment, on the other environment and confront the results to make emerge the best choice.

7.3.2 Knowing each other action

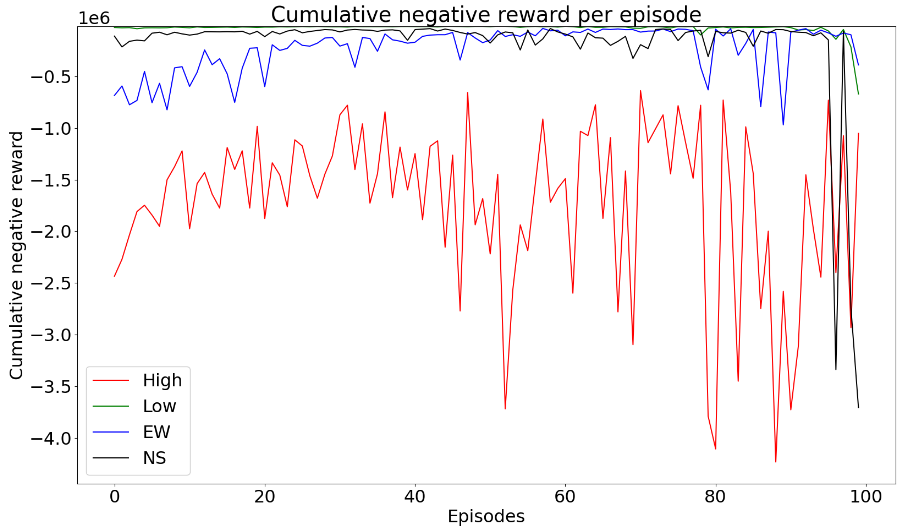
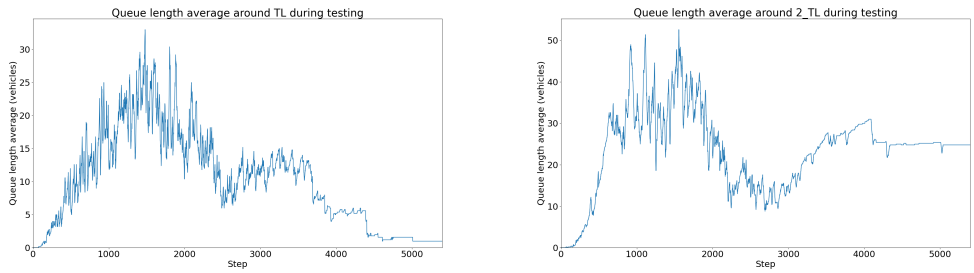


Figure 63: Cumulative negative reward per episode among both agents $Agent_1^{action}$ and $Agent_2^{action}$

	Low	*2STL* (%)	**High**	*2STL* (%)	**NS**	*2STL* (%)	**EW**	*2STL* (%)
nrw_{av}	-809754.8	+7483	-3170679.2	+117	-3484084.6	+11959	-117775.8	-73
twt	820958.4	+3532	3993735.0	+64	3546616.4	+4736	181772.4	-73
awt/v	1646.7	+3703	706.4	+19	1969.8	+5153	84.9	-72

Table 27: Results obtained with the IDQN model with the action knowledge



(a) Agent $Agent_1^{action}$ (b) Agent $Agent_2^{action}$

Figure 64: Average queue length of vehicles per steps over 5 episodes of individual agents in EW-traffic scenario.

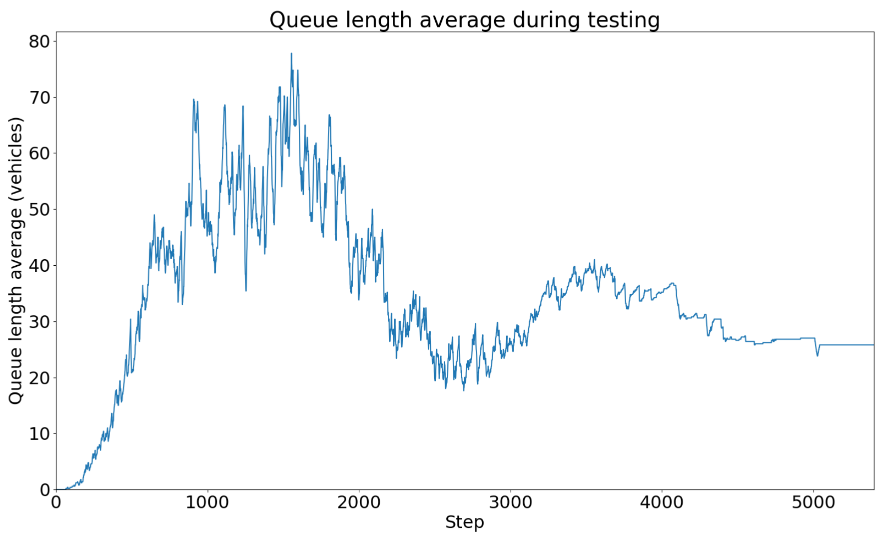


Figure 65: Average queue length of vehicles per steps over 5 episodes with the agents in EW-traffic scenario.

Cumulative negative reward analysis.

It is the first time that the instability in the NS scenario is such that it ends in agents' performances being worse than in High-traffic !

Table analysis.

Table 27 performances are surprising in three different ways.

1. Agents are performing 3 times better than STLs in EW-traffic.
2. High-traffic scenario show better results than IDQN without the action knowledge.
3. Agents metrics in Low and NS-traffic have soared in an exponential way.

However, the latest enumerated point has to be nuanced in the case of NS-traffic because, looking at Figure 63, the results obtained before the late 95^{th}'s episodes would have been much interesting to analyse. (The next tip box proposes a possible future work to prevent this case in point).

Average queue length analysis.

Compared to the previous experiments, Figures 64a and 64b show a more homogeneous trend resulting in Figure 65 being more balanced between both agents results. $Agent_1^{action}$ better manages the end simulation, whereas the policy developed by $Agent_2^{action}$ hits a threshold and do not let drive through the intersection the last vehicles accumulated in the central lane. Some form of collaboration in this particular scenario could be hypothesized between the 2000^{th} and 3000^{th} steps but this policy choice is apparently detrimental for $Agent_2^{action}$ at the end.

> This experiment highlights the need to introduce early-stopping techniques or, callbacks function (such as ModelCheckpoint) in order to avoid overtraining. Overtraining leads to injury and reduced performance. However, in deep reinforcement learning models, the minimization of the loss function (which is the most chosen criteria to stop or save the model weights) is not always directly correlated with the improvement of the model. So it needs a deeper understanding and analysis of the use case but it would definitely help to combat overfitting and unstabilized learning.

7.3.3 New local reward

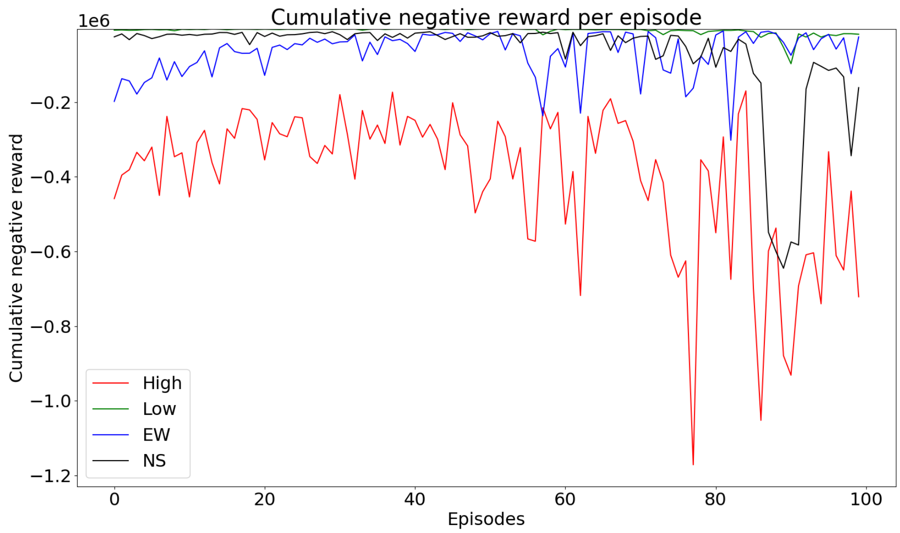
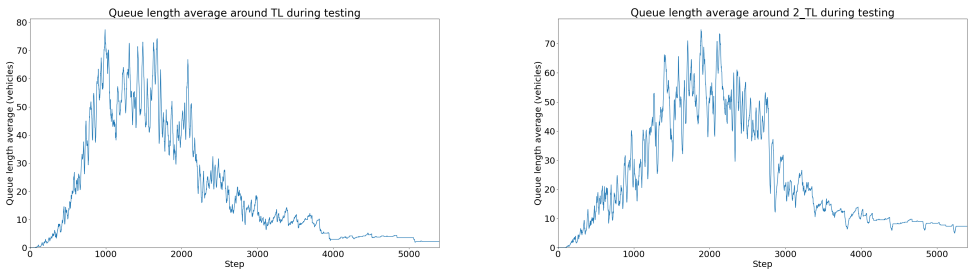


Figure 66: Cumulative negative reward per episode among both agents $Agent_1^{locr}$ and $Agent_2^{locr}$

	Low	*2STL* (%)	**High**	*2STL* (%)	**NS**	*2STL* (%)	**EW**	*2STL* (%)
nrw_{av}	-35202.2	+230	-1233550.9	-16	-300984.1	+942	-35314.4	-92
twt	159644.2	+606	6795419.2	+180	1502698.4	+1949	239661.0	-65
awt/v	264.6	+511	1981.3	+233	806.6	+2051	121.1	-60

Table 28: Results obtained with the 2 IDQN agents with the action knowledge and the new local reward



(a) Agent $Agent_1^{locr}$ (b) Agent $Agent_2^{locr}$

Figure 67: Average queue length of vehicles per steps over 5 episodes of individual agents in EW-traffic scenario.

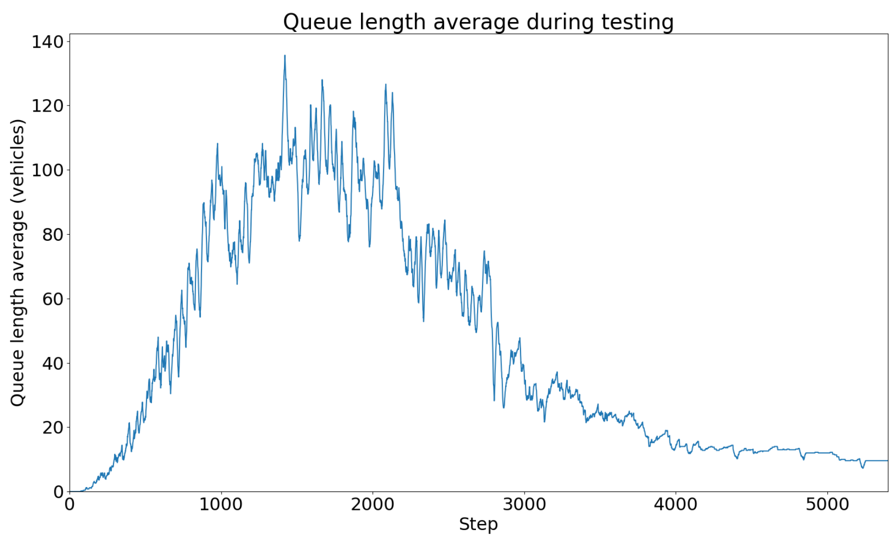


Figure 68: Average queue length of vehicles per steps over 5 episodes with the agents model in EW-traffic scenario.

Cumulative negative reward analysis.

The same observations as before can be made. The learning curve of the agents are deteriorating as it is trying to learn.

Table analysis.

Even if results reported in Table 28 are not much better compared to the previous model it is worth keeping it for future investigations. Indeed, results in Low and NS-traffic are better (8 times for the Low and more than 2 times for the NS). In High-traffic, it is 2 times worse. The introduction of the new local reward helped to stabilize the results in more acceptable ranges!

Attention should be paid to believe that if the reward is higher then the simulation is better since the calculation is just different from the other set of experiment. And looking at the total waiting time, the results of $Agent_1^{locr}$ and $Agent_2^{locr}$ are not improved.

Finally, in the EW-traffic scenario, the *twt* of the vehicles is less than 2 times higher (the lower the better) than in Table 27 but still outperforming the *2STL* baseline.

Average queue length analysis.

It is the first time that the curve profile resemble to the *2STL* one. In this particular (and only scenario-case) it can be deduced that agents are able to cooperate in order to do better than *2STL* (and the other previous models) with the advantage of not freezing an agent policy. The curve does not hit a threshold in the queue length which means that the agent can still evacuate the vehicles.

7.3.4 Partially shared reward

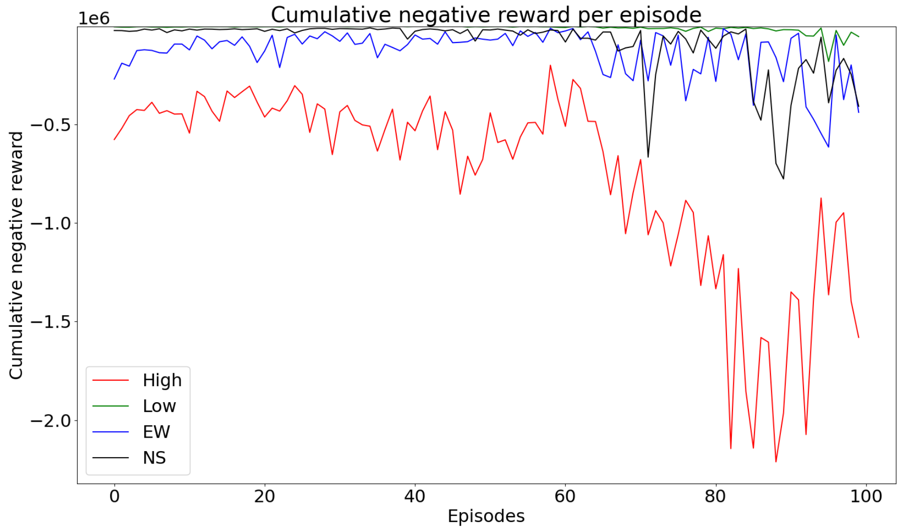
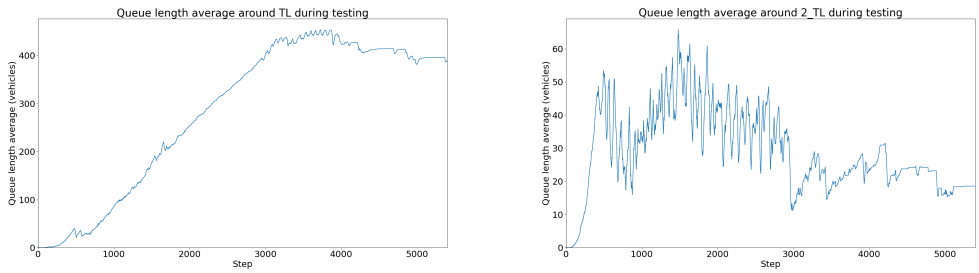


Figure 69: Cumulative negative reward per episode among both agents $Agent_1^{sharedr}$ and $Agent_2^{sharedr}$

	Low	*2STL* (%)	**High**	*2STL* (%)	**NS**	*2STL* (%)	**EW**	*2STL* (%)
nrw_{av}	-53296.7	+399	-1430770.4	-2	-197456.8	+583	-462414.3	+5
twt	230701.6	+921	5065912.6	+108	788216.4	+975	1687041.2	+149
awt/v	430.9	+895	1309.9	+120	396.1	+956	888.9	+193

Table 29: Results obtained with two IDQN agents knowing the action of each other and combining the new and partially shared reward



(a) Agent $Agent_1^{sharedr}$ (b) Agent $Agent_2^{sharedr}$

Figure 70: Average queue length of vehicles per steps over 5 episodes of individual agents in EW-traffic scenario.

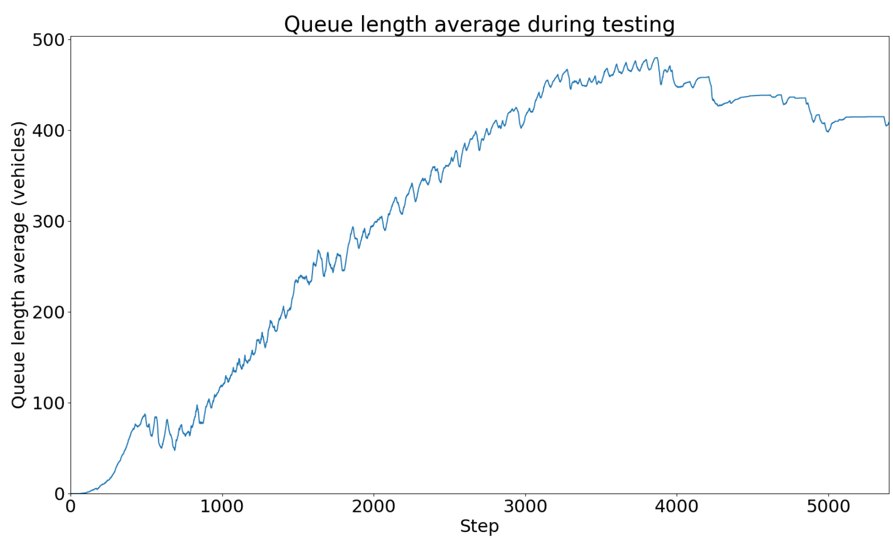


Figure 71: Average queue length of vehicles per steps over 5 episodes of both agents in EW-traffic scenario.

Cumulative negative reward analysis.

Each and every curve from Figure 69 are diverging which is a sign that the introduction of a partial reward is detrimental to the learning process.

Table analysis.

It is confirmed by the overall catastrophic results obtained by the agents.

Average queue length analysis.

This time, $Agent_1^{sharedr}$ is not able to learn any viable policy which makes $Agent_2^{sharedr}$ have hard times to operate based on the other agent decisions.

7.4 Agents performance overview

Overview of the previous experiments.

Table 30 sums up the combination of the different experiments that have been realised with the different components previously described.

	New environment	New state	Local reward	Shared reward
1	✓	✗	✗	✗
2	✓	✗	✗	✗
3	✓	✓	✗	✗
4	✓	✓	✓	✗
5	✓	✓	✓	✓

Table 30: Summary of the experiments realised in the case of the two-intersections simulation

The new environment that has been chosen to investigate the minimalist multi-agent approach has been the *side-by-side*. Then, the addition of the action knowledge has led to minor improvements (in EW-traffic especially). The new reward definition helped $Agents^{new}$ to adopt a better policy in order to make appear some form of collaboration (again, especially in EW-traffic). Finally, the partially shared reward has not been a success and made the agents unable to find a collaborative policy. It may be due to the fact that the discounted factor of $\frac{1}{2}$ is too high and should be better tuned.

All of the experiences are reproducible. Therefore, each trained model which has been evaluated is also available on the thesis GitHub repository.

8 Investigations for future work

Both the one-intersection and two-intersections cases are well suited to additional future works and investigations: starting from the study of alternative reward functions, that could take advantage of a different efficiency measure of each intersection links, to bringing a new state representation that could include partial observations to add more realism to the process. The agent actions should also control the duration of the traffic phases (green **and** yellow times) in order to gain more flexibility and possibly improve the efficiency on a wider range of situations. More complex scenarios could be designed to better match realistic situations (such as a deviation).

Moreover, real-world experiments could be conducted to study the implications of having real vehicles data. For instance, a typical human behavior is to adapt to the current situation to take any possible advantage: in a real-world appliance of the TLCS, a driver could exploit the policy patterns of the system and have a different driving behavior at the intersection in order to take advantage over the agent's behavior, that could lead to a decrease of the intersection efficiency.

Some ideas taken from the literature could also bring some major improvements:

- *Fingerprinting* could be applied regarding the multi-agent learning part. In fact, experience replay memory is a key component of DQN. The combination of experience replay with IQL is problematic because the non-stationarity introduced by IQL results in producing data in memory that no longer indicates the current dynamics in which the agent is learning. To address this issue, introducing fingerprint technique may be beneficial since it associates the data in replay memory with the age of the data («where does it originate from ? »). It gradually changes over training time in order to allow the model to generalise across experiences in which the other agents execute policies as they learn.

- The introduction of *Prioritized Experience Replay* that detaches agents from considering transitions with the same frequency that they are experienced could also lead to further improvements and more stability.

In addition, during this thesis work , the literature analysis of newer frameworks, that could bring some novelty and be more advantageous to the ATSC problem, has been pursued. And this reflection has started from the observation that Deep Q-Networks, despite the fact that it has demonstrated human-level or even superhuman performance in complex, high-dimensional spaces (Atari 2600, AlphaGo, and more lately AlphaGo Zero), is still quite an old framework which has shown his limitations.

An **actor-critic approach** has been investigated. It is based on the paper released in 2019 by Chu et al. [48] which was proposed with a working code. This implementation arises from the well-known asynchronous policy optimization method the A2C [60].

Before diving into the description of this approach and describing how it could be interesting as a future work, some theoretical background has to be enounced since actor-critic RL models are in-between policy-based algorithms and value-based ones.
This is due to the fact that these models have two estimators: an **actor** using Q-value estimation and a **critic** using state value function estimation. The actor role is to control the agent's behaviors based on policy while critic evaluates the taken action based on value function (Figure 72).

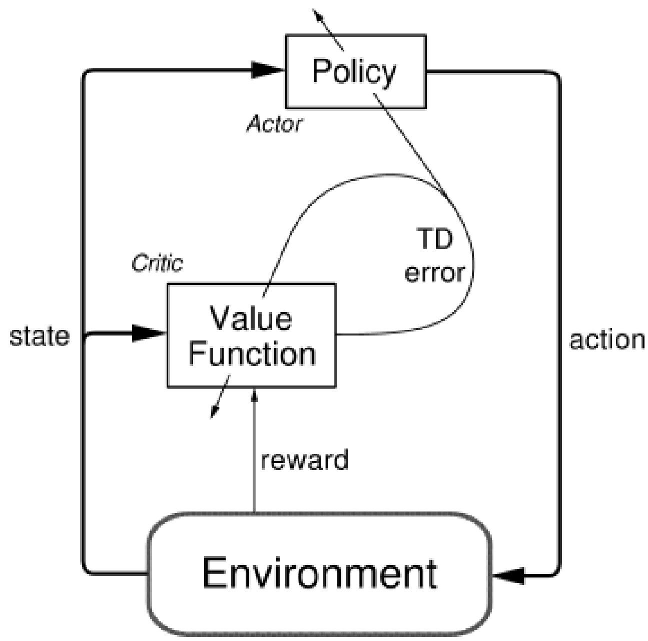


Figure 72: Actor-critic control loop

Figure 73 shows the process of the actor-critic algorithm implemented. The model produces action probabilities (which the role of the **actor**) to sample an action a_t that will result in a reward r_{t+1}. This reward, in addition with the state value outputted by the model (the role of the **critic**) is used to compute the *advantage*. Ultimately, this advantage reinforces the action and is used to train the model.

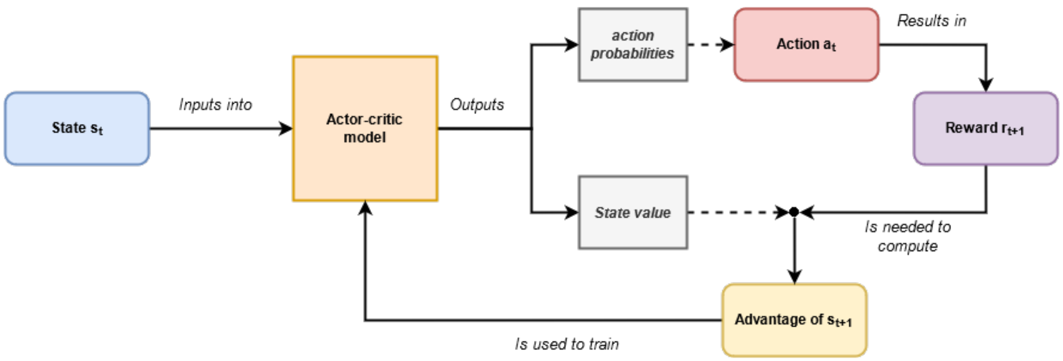


Figure 73: Broad overview of the actor-critic algorithm A2C implemented.

In many complex games and particularly for Natural Language Processing (NLP) tasks, it is common to implement a Recurrent Neural Network (RNN). Indeed, RNNs can store past traces as an internal state (in hidden states) to memorize short history. It is well suited to avoid having to input all historical states and therefore, without increasing the state dimension significantly which may reduce the attention on recent traffic condition. This could also be a point to introduce Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) in a new DNN structure. And it has been done in this A2C approach as it is figured in 74.

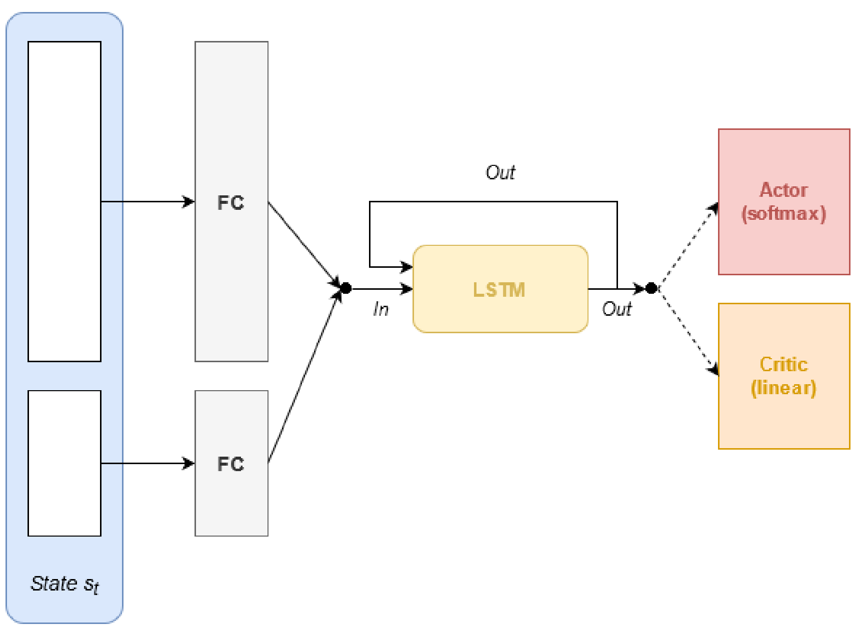


Figure 74: Proposed DNN structure of A2C

The state representation is at, first, processed by separate fully connected (FC) layers. Then, the outputs of the hidden layers are combined and inputted into a LSTM layer. Finally, 2 output layers are used as the last activation functions to process the information:

1. the softmax or normalized exponential function serves to produce the action probabilities.

2. the linear function which produces the state value.

This is a different algorithm that demonstrated good performances and that could be interesting to analyse more in-depth.

> In the paper, the A2C has been implemented as an Independent A2C (IA2C) and as a Multi-agent A2C (MA2C). On the figure 74, the state representation is composed of 2 components. Indeed, in the MA2C approach (the one of our interest), the first component represents the waiting time of the vehicles per each lane while the second includes the *neighborhood policies* which could be also interesting to study in order to improve the observability of each local agent.

Finally, in this search of finding a new algorithm that could be thought in a cooperative setup, the QMIX architecture [61] appeared to be a very promising value-based multi-agent reinforcement learning method. And, in the same context, an even more novel MARL approach has been recently presented. It is called QPLEX which stands for duPLEX dueling multi-agent Q-learning [62]. However, it should be noted that every benchmark of the experiments realised in these novel architectures have only been realised on micro-tasks, trained on a high number of agents and on a very specific environment (the Starcraft II game [63] [64]). It needs some work to dissociate the algorithms architecture from the specifications needed to work on this particular environment in order to evaluate them on a new environment with intersections. However, it has been mentioned because, from my personal point of view, it could certainly be very interesting to be tested in a very large set of Traffic Light Controllers.

> The researches made on the actor-critic framework led to the creation of another public repository (that can be Dockerized) [1]. None of the results obtained could have been put down in this thesis because the comparison would have been impossible since the conceptual architecture is too different from the one explored. But it could be worth pursuing the effort.

9 Conclusion

In a nutshell, the question that this thesis tried to answer was: how Deep Q-learning would behave in different sets of experiments starting **from one single agent to a multi-agent perspective** ? And its humble goal was to provide some analysis and benchmarks that could help to step up and reveal new hints for future investigations.

After squeezing the juice out of the one-only-agent model, the limits of the Deep Q-learning framework arose in an independent minimalist multi-agent approach. It brought to light that a different outlook needs to be expressed. An algorithm that works well on a minimalist multi-agent perspective while being still robust and efficient after scaling up is still worth investigating.

In the future of vehicle transportation, Artificial Intelligence will gradually become a fundamental factor in the process of developing new infrastructures and systems for the improvement of efficiency. Also, a new era of smart vehicles is arising and this could lead to the definition of communication protocols between vehicles and the infrastructure. This scenario will certainly open new ways of developing technologies for the transportation systems that could enrich the current techniques and discover new ones.

Also, in the context of serious increase in urban pollution, adaptive traffic light control systems could play an implicit but yet important role in the reduction of vehicle emissions. While waiting for electric vehicles to be the majority on the road, the air pollution problem can be addressed by reducing the cars waiting time and more widely by optimizing the traffic flow. The development of artificial intelligent systems can help crafting sustainable cities through air quality enhancement: **AI for Good**.

Appendices

A Comprehensive taxonomy of RL algorithms

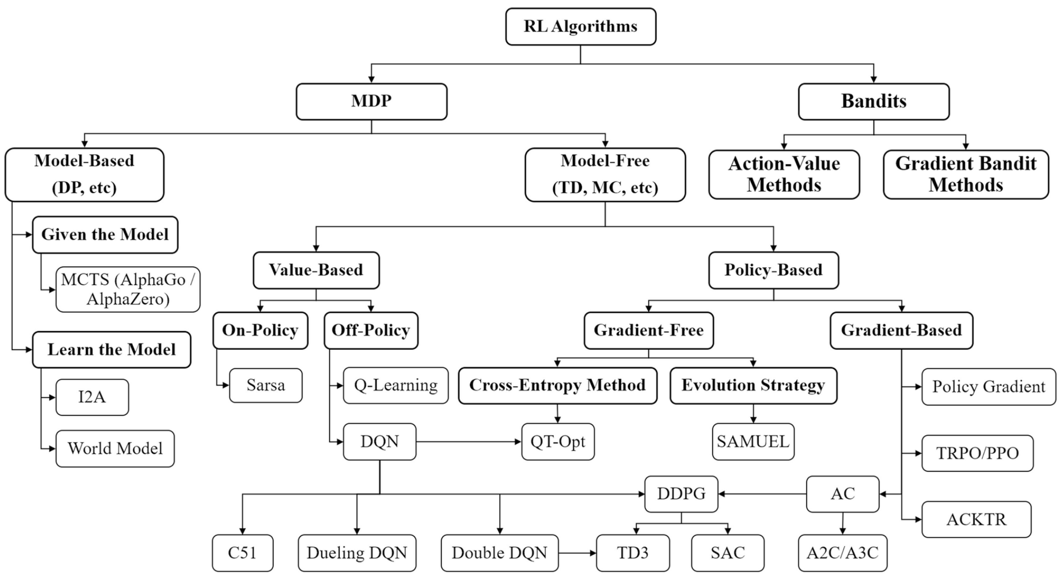


Figure 75: Map of reinforcement learning algorithms [4]. (Boxes with thick lines denote different categories, others denote specific algorithms)

B Visualizations options for one 4-way intersection

B.1 Training phase

During the training phase of an agent, each and every curves are calculated (see 4.1.3) in order to better apprehend and appreciate the results and be able to compare them.

B.1.1 Cumulative delay

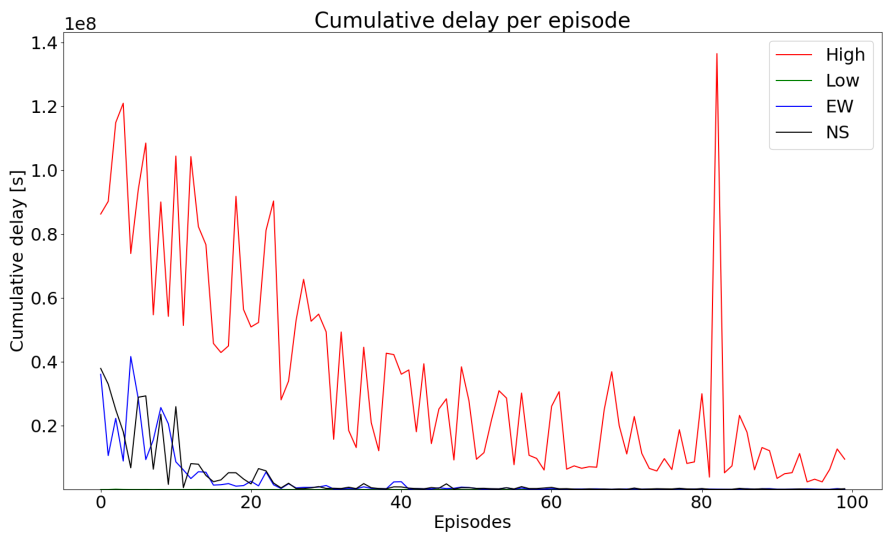


Figure 76: Cumulative delay scenario curves of the agent $Agent^{Small}$ training

This metric is genuinely core-related with the cumulative negative reward as it is the one used for calculating the reward. The scenario curve profiles are very similar. It is well exemplified with the peak in High-traffic around the 80^{th} episode in Figure 76 that can also be found in 28.

B.1.2 Queue length

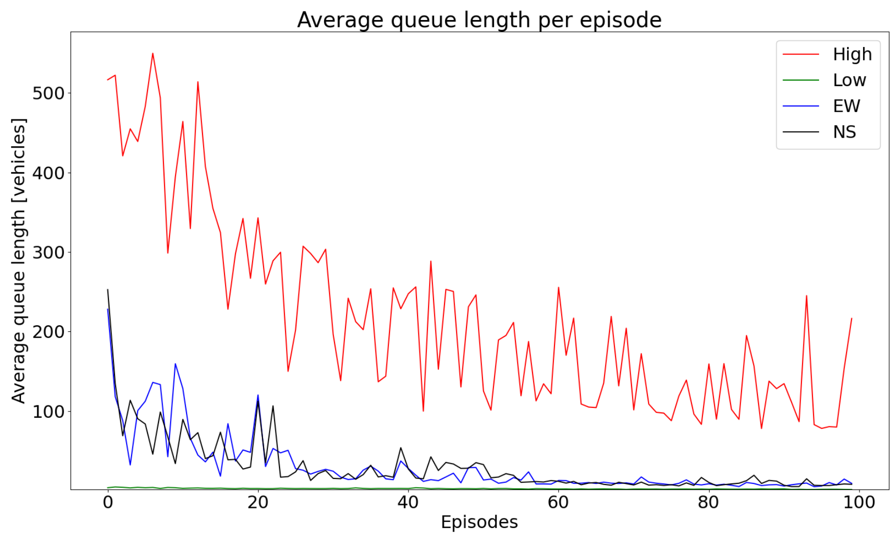


Figure 77: Average queue length (in vehicles) scenario curves of the agent $Agent^{Big}$ training

This metric is not tightly related with the cumulative negative reward but with the average queue length observed at the testing phase of the agent. It is harder to analyse but convergence and stability are the main criteria to keep.

B.1.3 Average waiting time per vehicle

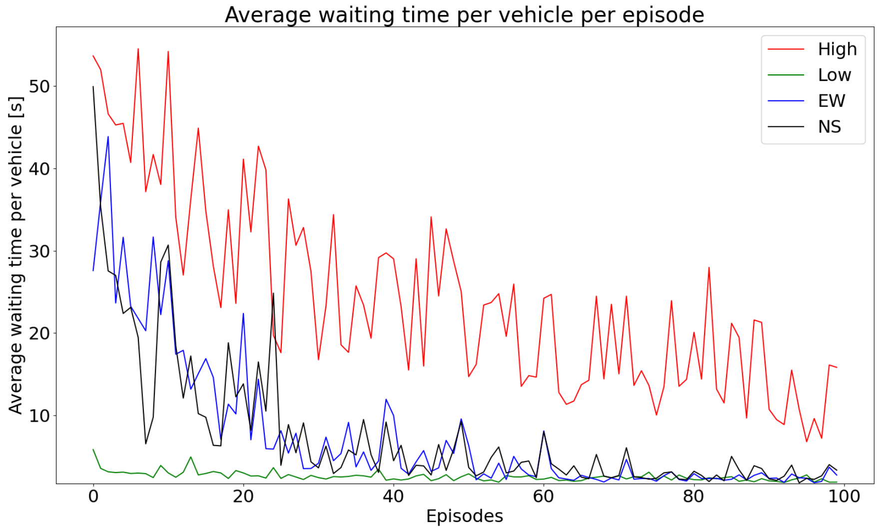


Figure 78: Average waiting time per vehicle during $Agent^{Best}$

This metric is very helpful as it is more human-centered. It simply reveals the average duration that a vehicle waits from source to destination. It is so ingrained in our daily life that it is one of the reason Google Maps is so utilized nowadays.

B.1.4 Average loss

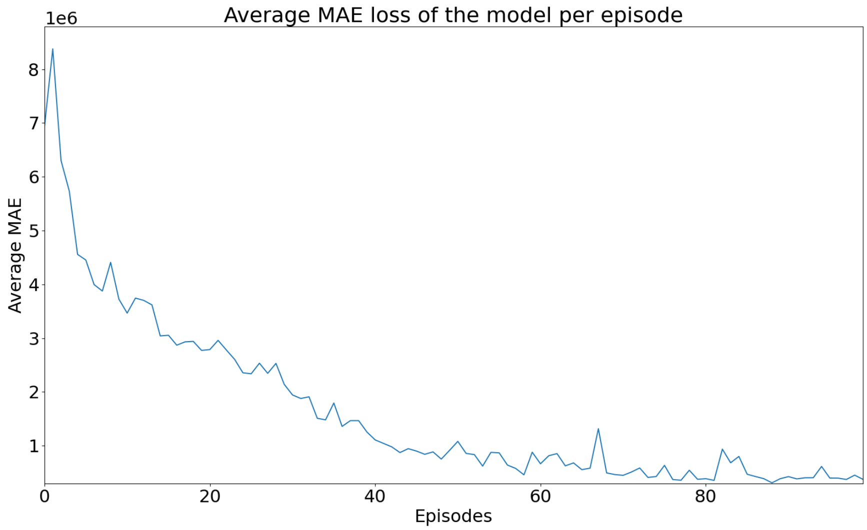


Figure 79: Average MAE loss of $Agent^{MAE}$

The loss in Deep Learning is a very important choice to make depending on the type of problem encountered (regression, classification, ...) and its specifications. Here, the average MAE loss have been used resulting in Figure 79.

As it is discussed, this choice in DRL is even more complex than the data inputted into the model is unknown and not pre-processed. However it helps grasping more knowledge by comparing the different curves and, again, convergence is needed.

Minimum loss

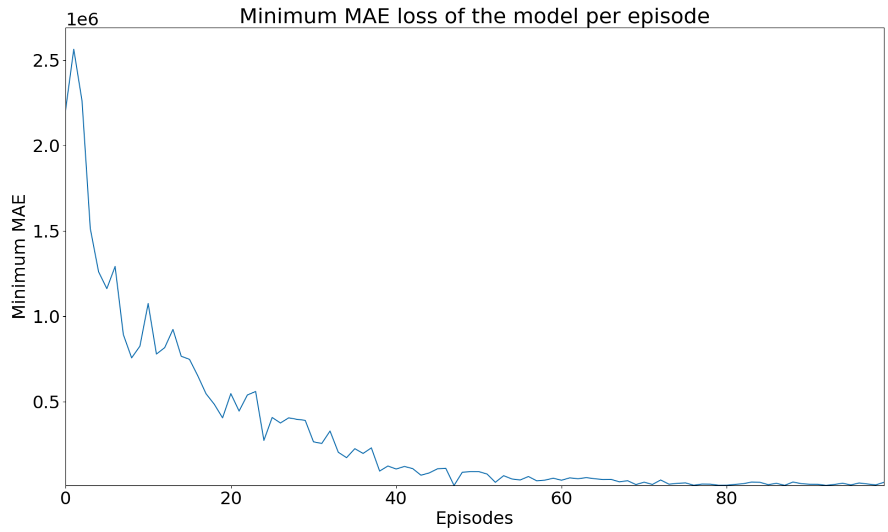


Figure 80: Minimum MAE loss curve of $Agent^{MAE}$

The minimum loss control has no other interest than to try to observe the differences with the average plot. And multiple comparisons show that they follow the same dynamic (Figure 79 and Figure 80) with less roughness.

B.1.5 Fundamental diagram of traffic flow

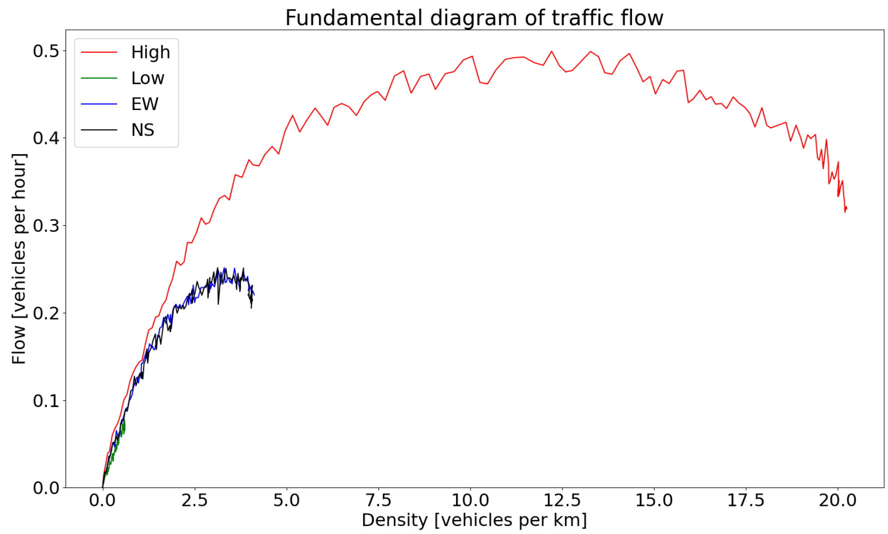
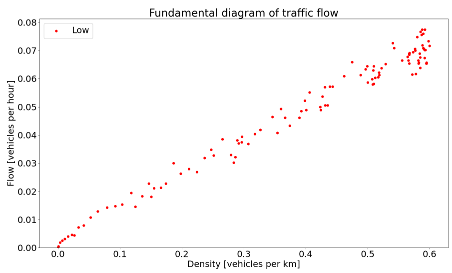


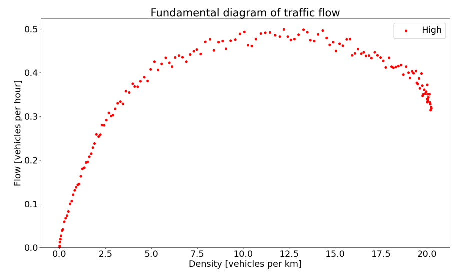
Figure 81: Fundamental diagram of traffic flow depending on the scenario, observed with $Agent^{Best}$

The primary tool for graphically displaying information in the study traffic flow is the fundamental diagram. It is an important aspect which could have been studied in more depth. However, as an example, Figure 81 show normal trends/curves which can be translated by a traffic flowing normally.

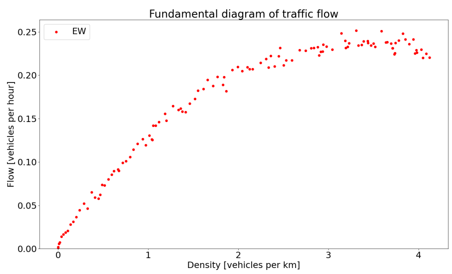
Zoom on the different scenarios



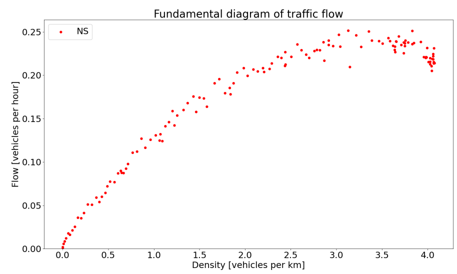
(a) Low-traffic



(b) High-traffic



(c) EW-traffic



(d) NS-traffic

Figure 82: Fundamental diagram of traffic flow for the different scenarios extracted from the *Best* agent

The distinct views depending on the traffic scenario could be interesting for a particular case of interest.

B.1.6 Occupancy diagram of traffic flow

In a similar way, the occupancy diagram of traffic flow can be a tool to analyse the modifications in traffic management.

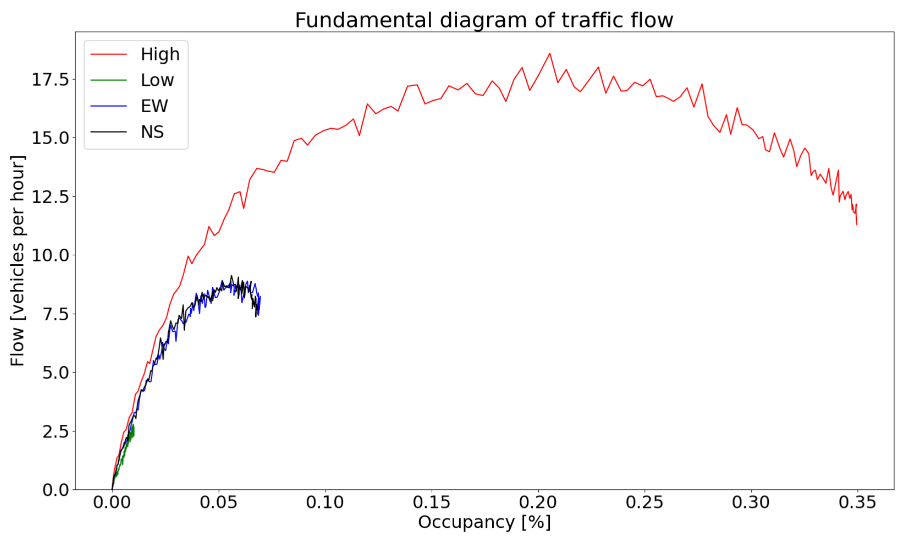
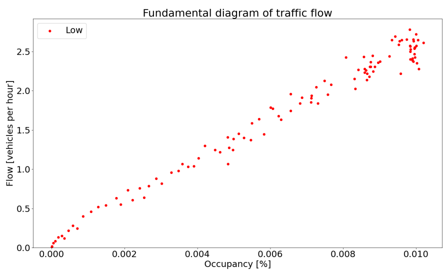
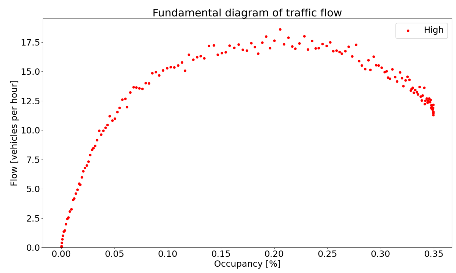


Figure 83: Occupancy diagram of traffic flow depending on the scenario, observed with $Agent^{MAE}$

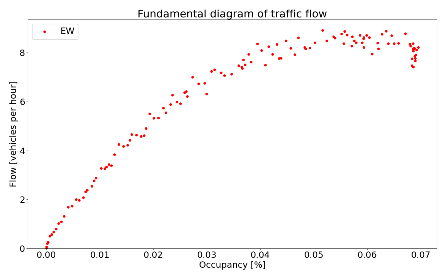
Zoom on the different scenarios



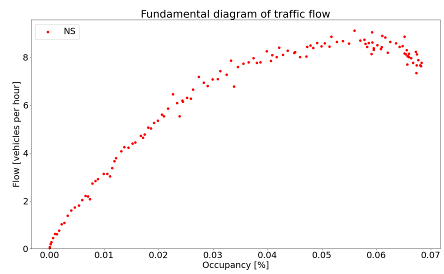
(a) Low-traffic



(b) High-traffic



(c) EW-traffic



(d) NS-traffic

Figure 84: Occupancy diagram of traffic flow for the different scenarios extracted from the $Agent^{MAE}$ agent

B.2 Testing phase

During testing phase, few plots are also saved in order to make the analysis go further.

B.2.1 Average reward

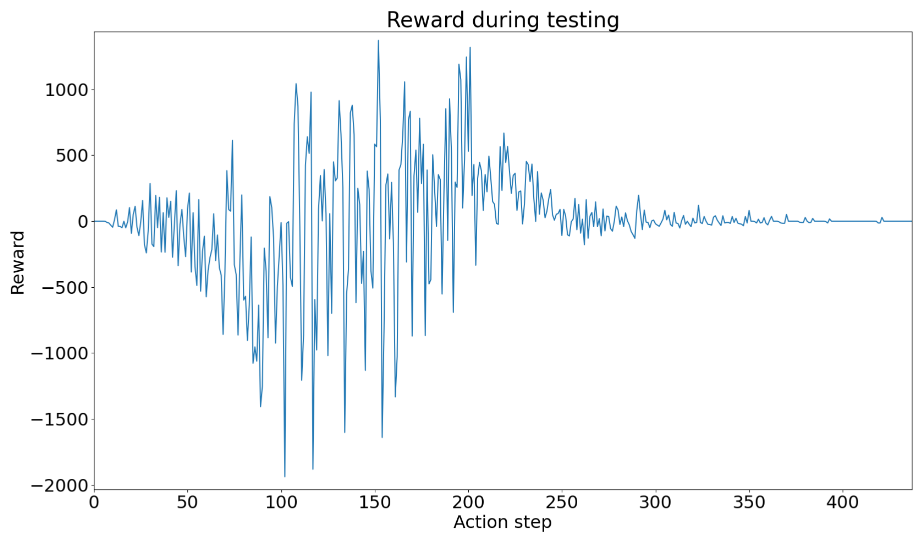


Figure 85: Average reward of $Agent^{Best}$ agent

Similar to the average queue length over the 5 episodes figures analysed in the thesis. It would have been also interesting to compare the average reward. Some links can be made with the increase of amplitude and the difficulty of the agent to to find an optimal policy.

B.2.2 No more average please

Despite this weird title lies a real problem that has been faced in this thesis... **Average is not enough.** Average is, though, a very powerful tool but it misses the outliers, the extrema are giving important hints to better understand the results and put them in perspective.

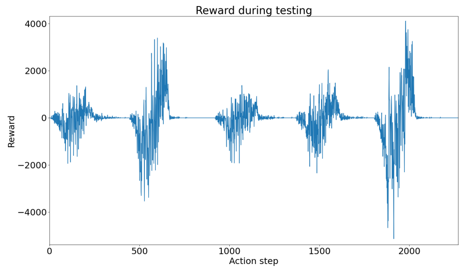


Figure 86: Reward during testing $Agent^{Best}$ for the 5 episodes

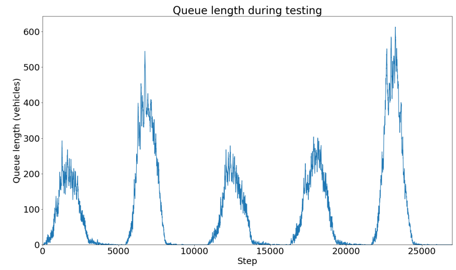


Figure 87: Queue length during testing $Agent^{Best}$ for the 5 episodes

This view is also interesting because it helps showing how difficult is a specific traffic generation scenario (depending on the seed) for the agent. It could lead to further investigations on why and how to remedy these fluctuations.

C Variation of green phase duration up to 60 seconds

	Low	STL (%)	**High**	STL (%)	**NS**	STL (%)	**EW**	STL(%)
nrw_{av}	-14633.8	+145	-138308.4	+7	-50104.6	-39	-35684.0	-55
twt	38637.6	+146	488523.6	-19	118447.8	-19	99844.2	-30
awt/v	94.2	+153	116.9	+5	76.3	+24	72.8	+21

Table 31: Results obtained with a modification of the green time duration up to *60 seconds*

Table 31, from 5.5, shows the threshold attained when increasing too much the green phase duration. In fact, in Low-traffic the results are catastrophic compared to STL and other models. In High-traffic scenario, there are worse which means that 60 seconds is too long for a green cycle phase for every scenario.

D Visualizations options for two 4-way intersections

The only difference with Appendix A lies in having the individual figures in addition with the global one.

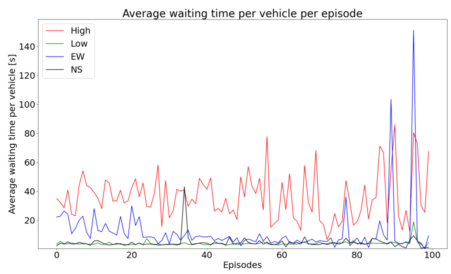


Figure 88: Average waiting time for vehicles in TL

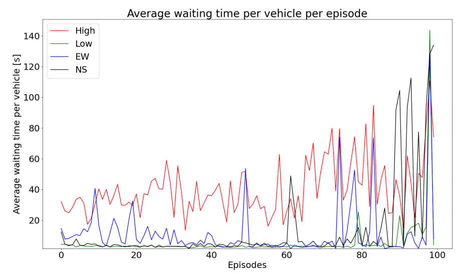


Figure 89: Average waiting time for vehicles in 2_TL

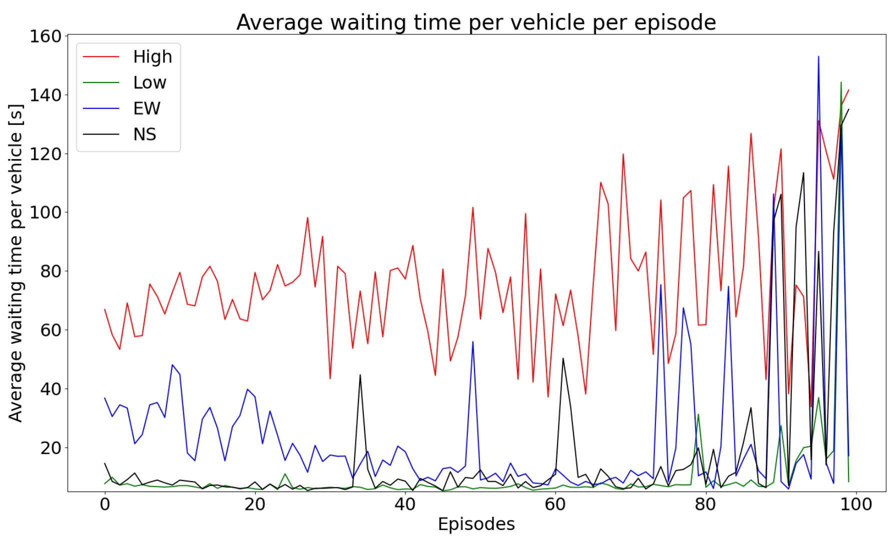


Figure 90: Average waiting time per vehicle in overall

Bibliography

[1] Andrea Vidali et al. "A Deep Reinforcement Learning Approach to Adaptive Traffic Lights Management". In: *Workshop From Objects to Agents WOA* (2019), pp. 42–50. URL: http://ceur-ws.org/Vol-2404/paper07.pdf (cit. on pp. i, 2, 26, 32).

[2] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html (cit. on p. 6).

[3] Neziha Akalin and Amy Loutfi. *Reinforcement Learning Approaches in Social Robotics*. Sept. 2020 (cit. on p. 8).

[4] Hongming Zhang and Tianyang Yu. "Taxonomy of Reinforcement Learning Algorithms". In: *Deep Reinforcement Learning: Fundamentals, Research and Applications*. Ed. by Hao Dong, Zihan Ding, and Shanghang Zhang. Singapore: Springer Singapore, 2020, pp. 125–133. DOI: 10.1007/978-981-15-4095-0_3. URL: https://doi.org/10.1007/978-981-15-4095-0_3 (cit. on pp. 8, 127).

[5] Richard S. Sutton. "Learning to Predict By the Methods of Temporal Differences". In: *Machine Learning* 3.1 (1988), pp. 9–44. URL: http://www.cs.ualberta.ca/~sutton/papers/sutton-88.pdf (cit. on p. 9).

[6] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8 (1992), pp. 279–292. DOI: https://doi.org/10.1007/BF00992698 (cit. on p. 10).

[7] G. A. Rummery and M. Niranjan. *On-Line Q-Learning Using Connectionist Systems*. Tech. rep. 1994 (cit. on pp. 10, 27).

[8] Ronald J. Williams. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Mach. Learn.* 8.3–4 (May 1992), pp. 229–256. ISSN: 0885-6125. DOI: 10.1007/BF00992696. URL: https://doi.org/10.1007/BF00992696 (cit. on p. 13).

[9] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836. URL: http://dx.doi.org/10.1038/nature14236 (cit. on p. 14).

[10] Long-Ji Lin. "Reinforcement Learning for Robots Using Neural Networks". PhD thesis. Carnegie Mellon University, 1993 (cit. on p. 15).

[11] Tom Schaul et al. *Prioritized Experience Replay.* cite arxiv:1511.05952Comment: Published at ICLR 2016. 2015. URL: http://arxiv.org/abs/1511.05952 (cit. on p. 16).

[12] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-learning". In: *CoRR* abs/1509.06461 (2015). arXiv: 1509.06461. URL: http://arxiv.org/abs/1509.06461 (cit. on p. 16).

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning". In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539 (cit. on p. 18).

[16] Lucian Busoniu, Robert Babuska, and Bart De Schutter. "Multi-Agent Reinforcement Learning: A Survey". In: *2006 9th International Conference on Control, Automation, Robotics and Vision.* 2006, pp. 1–6. DOI: 10.1109/ICARCV.2006.345353 (cit. on p. 20).

[17] Ming Tan. "Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents". In: *In Proceedings of the Tenth International Conference on Machine Learning.* Morgan Kaufmann, 1993, pp. 330–337 (cit. on p. 21).

[18] Foerster Jakob et al. "Stabilising experience replay for deep multi-agent reinforcement learning". In: *34th International Conference on Machine Learning, ICML 2017* (2017), pp. 1879–1888 (cit. on p. 22).

[19] Ardi Tampuu et al. "Multiagent Cooperation and Competition with Deep Reinforcement Learning". In: *CoRR* abs/1511.08779 (2015). arXiv: 1511.08779. URL: http://arxiv.org/abs/1511.08779 (cit. on p. 22).

[20] Kok-Lim Alvin Yau et al. "A Survey on Reinforcement Learning Models and Algorithms for Traffic Signal Control". In: *ACM Comput. Surv.* 50.3 (June 2017). ISSN: 0360-0300. DOI: 10.1145/3068287. URL: https://doi.org/10.1145/3068287 (cit. on pp. 23, 25, 27).

[21] Wade Genders and Saiedeh Razavi. "Evaluating reinforcement learning state representations for adaptive traffic signal control". In: *Procedia Computer Science* 130 (Jan. 2018), pp. 26–33. DOI: 10.1016/j.procs.2018.04.008 (cit. on p. 24).

[22] Juntao Gao et al. "Adaptive Traffic Signal Control: Deep Reinforcement Learning Algorithm with Experience Replay and Target Network". In: *CoRR* abs/1705.02755 (2017). arXiv: 1705.02755. URL: http://arxiv.org/abs/1705.02755 (cit. on pp. 24–26, 29).

[23] Wade Genders and Saiedeh Razavi. *Using a Deep Reinforcement Learning Agent for Traffic Signal Control*. 2016. arXiv: `1611.01142 [cs.LG]` (cit. on pp. 24–26, 29, 38).

[24] Seyed Sajad Mousavi et al. "Traffic Light Control Using Deep Policy-Gradient and Value-Function Based Reinforcement Learning". In: *CoRR* abs/1704.08883 (2017). arXiv: `1704.08883`. URL: `http://arxiv.org/abs/1704.08883` (cit. on pp. 24, 25, 29).

[25] Hua Wei et al. "IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 2496–2505. ISBN: 9781450355520. DOI: `10.1145/3219819.3220096`. URL: `https://doi.org/10.1145/3219819.3220096` (cit. on pp. 24, 26).

[26] Li Li, Yisheng Lv, and Fei-Yue Wang. "Traffic signal timing via deep reinforcement learning". In: *IEEE/CAA Journal of Automatica Sinica* 3.3 (2016), pp. 247–254. DOI: `10.1109/JAS.2016.7508798` (cit. on pp. 25, 26).

[27] Saijiang Shi and Feng Chen. "Deep Recurrent Q-learning Method for Area Traffic Coordination Control". In: *JAMCS* 27 (2018), pp. 1–11. DOI: `10.9734/JAMCS/2018/41281` (cit. on p. 26).

[28] Chung-Jae Choe et al. "Deep Q Learning with LSTM for Traffic Light Control". In: *2018 24th Asia-Pacific Conference on Communications (APCC)*. 2018, pp. 331–336. DOI: `10.1109/APCC.2018.8633520` (cit. on p. 26).

[29] Ammar Haydari and Yasin Yilmaz. "Deep Reinforcement Learning for Intelligent Transportation Systems: A Survey". In: *CoRR* abs/2005.00935 (2020). arXiv: `2005.00935`. URL: `https://arxiv.org/abs/2005.00935` (cit. on p. 27).

[30] Hua Wei et al. "A Survey on Traffic Signal Control Methods". In: *CoRR* abs/1904.08117 (2019). arXiv: `1904.08117`. URL: `http://arxiv.org/abs/1904.08117` (cit. on p. 27).

[31] Thomas L. Thorpe and Charles W. Anderson. *Traffic Light Control Using SARSA with Three State Representations*. Tech. rep. IBM Corporation, 1996 (cit. on p. 27).

[32] Kaige Wen, Shiru Qu, and Yumei Zhang. "A Stochastic Adaptive Control Model for Isolated Intersections". In: *2007 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2007, pp. 2256–2260. DOI: 10.1109/ROBIO.2007.4522521 (cit. on p. 28).

[33] Akshay Kekuda, R. Anirudh, and Mithun Krishnan. "Reinforcement Learning based Intelligent Traffic Signal Control using n-step SARSA". In: *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*. 2021, pp. 379–384. DOI: 10.1109/ICAIS50930.2021.9395942 (cit. on p. 28).

[34] Baher Abdulhai, Rob Pringle, and Grigoris J. Karakoulas. "Reinforcement Learning for True Adaptive Traffic Signal Control". In: *Journal of Transportation Engineering* 129.3 (2003), pp. 278–285. DOI: 10.1061/(ASCE)0733-947X(2003)129:3(278). eprint: https://ascelibrary.org/doi/pdf/10.1061/%28ASCE%290733-947X%282003%29129%3A3%28278%29. URL: https://ascelibrary.org/doi/abs/10.1061/%28ASCE%290733-947X%282003%29129%3A3%28278%29 (cit. on p. 28).

[35] Eduardo Camponogara and Werner Kraus. "Distributed Learning Agents in Urban Traffic Control". In: vol. 2902. Nov. 2003, pp. 324–335. ISBN: 978-3-540-20589-0. DOI: 10.1007/978-3-540-24580-3_38 (cit. on p. 28).

[36] Saad Touhbi et al. "Adaptive Traffic Signal Control : Exploring Reward Definition For Reinforcement Learning". In: *Procedia Computer Science* 109 (June 2017), pp. 513–520. DOI: 10.1016/j.procs.2017.05.327 (cit. on p. 28).

[37] Marco Wiering. *Multi-agent reinforcement learning for traffic light control*. 2000 (cit. on p. 28).

[38] Merlijn Steingrover et al. "Reinforcement Learning of Traffic Light Controllers Adapting to Traffic Congestion." In: Jan. 2005, pp. 216–223 (cit. on p. 28).

[39] Jiří Iša et al. "Reinforcement Learning of Traffic Light Controllers Adapting to Accidents". In: (Feb. 2006) (cit. on p. 28).

[40] L A Prashanth and Shalabh Bhatnagar. "Reinforcement learning with average cost for adaptive control of traffic lights at intersections". In: *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. 2011, pp. 1640–1645. DOI: 10.1109/ITSC.2011.6082823 (cit. on p. 29).

[41] Prashanth LA and Shalabh Bhatnagar. "Reinforcement Learning With Function Approximation for Traffic Signal Control". In: *IEEE Transactions on Intelligent Transportation Systems* 12.2 (2011), pp. 412–421. DOI: 10.1109/TITS.2010.2091408 (cit. on p. 29).

[42] Sahar Araghi, Abbas Khosravi, and Douglas Creighton. "Distributed Q-learning Controller for a Multi-Intersection Traffic Network". In: *Neural Information Processing*. Ed. by Sabri Arik et al. Cham: Springer International Publishing, 2015, pp. 337–344. ISBN: 978-3-319-26532-2 (cit. on p. 29).

[43] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG] (cit. on p. 29).

[44] Rusheng Zhang et al. "Partially Observable Reinforcement Learning for Intelligent Transportation Systems". In: *CoRR* abs/1807.01628 (2018). arXiv: 1807.01628. URL: http://arxiv.org/abs/1807.01628 (cit. on p. 30).

[45] Elise Van der Pol and Frans A. Oliehoek. "Coordinated Deep Reinforcement Learners for Traffic Light Control". In: *NIPS'16 Workshop on Learning, Inference and Control of Multi-Agent Systems*. Dec. 2016. URL: https://sites.google.com/site/malicnips2016/papers (cit. on p. 30).

[46] Mengqi LIU et al. "Cooperative Deep Reinforcement Learning for Traffic Signal Control". In: (2017) (cit. on p. 30).

[47] Jeancarlo Calvo and Ivana Dusparic. "Heterogeneous Multi-Agent Deep Reinforcement Learning for Traffic Lights Control". In: Nov. 2018 (cit. on pp. 31, 59).

[48] Tianshu Chu et al. "Multi-Agent Deep Reinforcement Learning for Large-Scale Traffic Signal Control". In: *IEEE Transactions on Intelligent Transportation Systems* (2019) (cit. on pp. 31, 122).

[49] Lara Codeca and Jérôme Härri. "Towards multimodal mobility simulation of C-ITS: The Monaco SUMO traffic scenario". In: *VNC 2017, IEEE Vehicular Networking Conference, November 27-29, 2017, Torino, Italy*. T, Nov. 2017. DOI: http://dx.doi.org/10.1109/VNC.2017.8275627 (cit. on p. 31).

[50] Andrea Vidali. "Simulation of a traffic light scenario controlled by a Deep Reinforcement Learning agent". MA thesis. Università degli Studi di Milano Bicocca, 2018 (cit. on p. 32).

[51] Pablo Alvarez Lopez et al. "Microscopic Traffic Simulation using SUMO". In: *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. URL: https://elib.dlr.de/124092/ (cit. on p. 35).

[52] Carlos F. Daganzo and Nikolas Geroliminis. "An analytical approximation for the macroscopic fundamental diagram of urban traffic". In: *Transportation Research Part B: Methodological* 42.9 (2008), pp. 771–781. ISSN: 0191-2615. DOI: https://doi.org/10.1016/j.trb.2008.06.008. URL: https://www.sciencedirect.com/science/article/pii/S0191261508000799 (cit. on p. 56).

[53] Xinkai Wu, Henry X. Liu, and Nikolas Geroliminis. "An empirical analysis on the arterial fundamental diagram". In: *Transportation Research Part B: Methodological* 45.1 (2011), pp. 255–266. ISSN: 0191-2615. DOI: https://doi.org/10.1016/j.trb.2010.06.003. URL: https://www.sciencedirect.com/science/article/pii/S0191261510000834 (cit. on p. 56).

[55] Blazej Osinski et al. "CARLA Real Traffic Scenarios - novel training ground and benchmark for autonomous driving". In: *CoRR* abs/2012.11329 (2020). arXiv: 2012.11329. URL: https://arxiv.org/abs/2012.11329 (cit. on p. 59).

[56] Peter Koonce and Lee Rodegerdts. *Traffic Signal Timing Manual*. 1200 New Jersey Avenue, SE Washington, DC 20590: US. Department of Transportation. Federal Highway Administration, June 2008 (cit. on p. 66).

[58] Franziska Klügl. "A Validation Methodology for Agent-Based Simulations". In: *Proceedings of the 2008 ACM Symposium on Applied Computing*. SAC '08. Fortaleza, Ceara, Brazil: Association for Computing Machinery, 2008, pp. 39–43. ISBN: 9781595937537. DOI: 10.1145/1363686.1363696. URL: https://doi.org/10.1145/1363686.1363696 (cit. on p. 82).

[59] Denise de Oliveira and Ana L.C. Bazzan. *Multiagent Learning on Traffic Lights Control: Effects of Using Shared Information*. 2009. DOI: http://doi:10.4018/978-1-60566-226-8.ch015 (cit. on p. 99).

[60] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *CoRR* abs/1602.01783 (2016). arXiv: 1602.01783. URL: http://arxiv.org/abs/1602.01783 (cit. on p. 122).

[61] Tabish Rashid et al. "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning". In: *CoRR* abs/1803.11485 (2018). arXiv: 1803.11485. URL: http://arxiv.org/abs/1803.11485 (cit. on p. 125).

[62] Jianhao Wang et al. "QPLEX: Duplex Dueling Multi-Agent Q-Learning". In: *CoRR* abs/2008.01062 (2020). arXiv: 2008.01062. URL: https://arxiv.org/abs/2008.01062 (cit. on p. 125).

[63] Oriol Vinyals et al. "StarCraft II: A New Challenge for Reinforcement Learning". In: *CoRR* abs/1708.04782 (2017). arXiv: 1708.04782. URL: http://arxiv.org/abs/1708.04782 (cit. on p. 125).

[64] Mikayel Samvelyan et al. "The StarCraft Multi-Agent Challenge". In: *CoRR* abs/1902.04043 (2019). arXiv: 1902.04043. URL: http://arxiv.org/abs/1902.04043 (cit. on p. 125).

Sitography

[13] URL: https://towardsdatascience.com/deep-q-network-combining-deep-reinforcement-learning-a5616bcfc207 (cit. on p. 17).

[14] URL: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/ (cit. on p. 17).

[54] URL: https://flask.palletsprojects.com/en/2.0.x/ (cit. on p. 56).

[57] URL: https://optuna.org/ (cit. on p. 75).