Università degli Studi di Milano Bicocca

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di Laurea Magistrale in Informatica

Simulation of a traffic light scenario controlled by a Deep Reinforcement Learning agent

Relatore: Stefania Bandini

Co-relatore: Luca Crociani

Tesi di Laurea Magistrale di:

Andrea Vidali

Matricola 780747

Anno Accademico 2017-2018

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern society relies on road transportation for the movement of people and goods. Ensuring vehicles can move efficiently on the road infrastructure is beneficial for everybody. However, the population growth and the increase of the number of vehicles on the streets have gone often beyond the current capacity of the infrastructure, leading to more congestion, higher travel times and more emissions. To address this issue, two solutions are possible: the first is to expand the transportation capacity and build higher quality road systems to handle the higher traffic flow. However, this solution requires a significant amount of time and resources, and it penalizes the traffic efficiency during the process. The second solution is to operate on the existing infrastructure by improving the systems that have a significant impact on the traffic flow, such as traffic signal controllers. This approach is desirable since it has a low cost of implementation and reuses the current equipment. Therefore, in this thesis the recent advancements in the field of Artificial Intelligence are used to research and develop a learning agent that is able to control a traffic light system, with the objective to increase the efficiency of road transportation.

The problem addresses in this thesis 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?

Agent definition

The agent developed in this thesis makes use of a deep neural network to choose which light phase activate and it is trained using the Q-learning rein-

forcement learning algorithm. Reinforcement learning is an area of machine learning where one or more agent learns to solve a task in an environment, using the experience generated by the interactions between the agent and the environment itself.

The environment in which the agent acts is a 4-way intersection where 4 lanes per arm approach the intersection from the compass directions and lead to 4 lanes per arm leaving the intersection. On every arm, each lane defines the possible directions that a vehicle can follow: the right-most lane enable vehicles to turn right or going straight, the two central lanes bound the driver to go straight while on the left-most lane the left turn is the only direction allowed. In the center of the intersection, a traffic light system, controlled by the agent, manages the approaching traffic. In particular, on every arm the left-most lane has a dedicated traffic light, while the other three lanes share a traffic light.

To learn how to develop an optimal policy, the agent interacts with this environment. The interaction is handled by the traffic micro-simulator SUMO. The simulator is used to let the agent experience a multitude of situations and learn by mistake which effects have the light phase activated during the simulations.

In the reinforcement learning framework, the agent is defined by three fundamental properties, which are a state representation, a set of possible actions and the reward function.

- The state is the representation of the environment at a certain moment from the agent's perception. In this thesis, the state of the environment is represented as a discretization of the road surface. The purpose of this representation is to inform the agent about the positions of vehicles inside the environment at a certain time.

- The actions set identifies the possible ways of interaction available to the agent. In this thesis, an action of the agent is defined as a configuration of the traffic light that implies the green phase for some lanes for a fixed amount of time.

- The reward function generates feedback from the environment to the agent, to evaluate its chosen action. The agent uses the reward received to improve his action choice policy in the future. A positive reward is a consequence of a good action, while a negative reward is received after

a bad action. In this thesis, two reward functions based on the vehicles' waiting times are defined and therefore tested.

The learning algorithm

In a typical iteration of the algorithm, the agent starts with the choice of the next light phase to activate. When the chosen action is activated, the simulation can proceed until a new action from the agent is required. At this point, the agent receives from the environment two fundamental information: the current state of the intersection and the reward. The state of the intersection is used to select the next light phase, while the reward is used to understand the effects of the previous action and make better actions in the future.

In order to (i) select the next light phase and (ii) understand the effects of the previous action and choose better actions in the future, the agent makes use of the combination of two methods of Artificial Intelligence: deep neural network and Q-learning respectively. 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 particular state. The purpose of the Q-value is to quantify the goodness of an action using the immediate reward combined with the maximum expected future rewards, so the agent knows which actions will lead to the best overall policy. The mapping between a state and the action values from that state are then approximated by a deep neural network. In summary, during the simulation the agent iteratively gains knowledge about the value of actions sequences and, in the end, hopefully it is able to select the action's sequence that leads to the higher cumulative reward and therefore the best performance.

The agent is trained by submitting episodes of 1 hour and 30 minutes where the agent explores the state space and understand the values of actions. Four different scenarios are submitted to the agent during the training: low traffic, high traffic, and two scenarios where most of the traffic respectively comes from two opposite arms of the intersection.

Thesis structure

Chapter 2 describes the state of the art, where the typical agent structures and the reinforcement learning framework are briefly introduced. Then, an overview of the reinforcement learning elements in the field of traffic signal control is described, followed by an approach analysis of some of the recent

works on this context. Chapter 3 is about the design of the proposed agent divided into state, action, reward and learning mechanism. Chapter 4 describe the training phase of the agent and the techniques used in this process, such as the agent exploration strategy or the traffic generation algorithm. In Chapter 5 are presented and discussed the performance of the agent on the tested scenarios.

The code relative to this thesis is available on GitHub[1].

[1] `https://github.com/AndreaVidali/Deep-QLearning-Agent-for-Traffic-Signal-Control`

Chapter 2

The state of the art

In this chapter, an overview of related works on traffic control system is delineated. Primarily, it will be introduced the concept of learning agents following by a brief overview of a reinforcement learning problem. This is will be the starting point of the next subject, which is about the multiple design possibilities of reinforcement learning elements in the context of Traffic Signal Control (TSC). Finally, state-of-the-art artificial intelligence techniques for TSC will be described and discussed, including a brief summary of real-world appliances.

2.1 Learning agents

> *"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators."*

Russell and Norvig [1] use this sentence to define a general agent that is involved in some activity in the within the boundaries of an environment. The agent can percept the current abstraction of the environment by using sensors and directly influence the next state of the environment using actuators. The words *current* and *next* are not casual since the interaction between the agent and the environment is controlled by the flow of time. The action choice of the agent not only depends on the current perception of the environment but could also depend on the percept sequence until this instant. Therefore, the behavior of the agent is defined by an action function which maps a percept sequence to an action.

An agent that always do the right action is called *rational agent* [1]. The action function alone does not indicate which action is the right action to make in a given instant. In order to give the agent the ability to understand the goodness of actions, a performance measure has to be involved. It gives feedback on the agent's actions by analyzing the next percept sequence and understand the consequences of the previous action. In summary, what is rational at a given instant depends on the performance measure, the prior environment knowledge, the actions available at that instant and the percept sequence to date.

A classification of agents has been introduced to distinguish them by complexity [1] and it is the following.

Simple reflex agent

This agent selects the action taking into consideration only the current percept and ignoring the rest of the percept sequence. This reduces the intelligence of the agent to only react to the current situation of the environment, following the *condition-action rule* "if condition, then action". This agent mechanism works only when the environment is fully observable, otherwise one or more condition-action mapping will be missed, resulting in the inability to select the best action.

Model-based reflex agent

A model-based reflex agent evolves from a simple reflex agent by including an internal structure that describes the part of the environment that cannot be perceived. This part of the agent is called *model* of the world. Therefore, this agent is not only limited to fully observable but can also handle partially observable environments.

Goal-based agent

The goal-based agent further expands the capability of a model-based agent by adding the information about the goal that the agent is trying to achieve. This leads to different decision-making because this agent knows which actions will help him reach the goal state.

Utility-based agent

The goal knowledge is sufficient to the agent to complete the task with success but is not telling him how to complete the task i.e. maximize the efficiency. The utility-based agent includes a utility function which maps a state to the measure of the utility of that state (de facto, the performance measure mentioned before). Therefore, a rational utility-based agent chooses the action that maximizes the expected utility of the chosen actions.

Learning agent

Figure 2.1 shows a schematic representation of a learning agent.

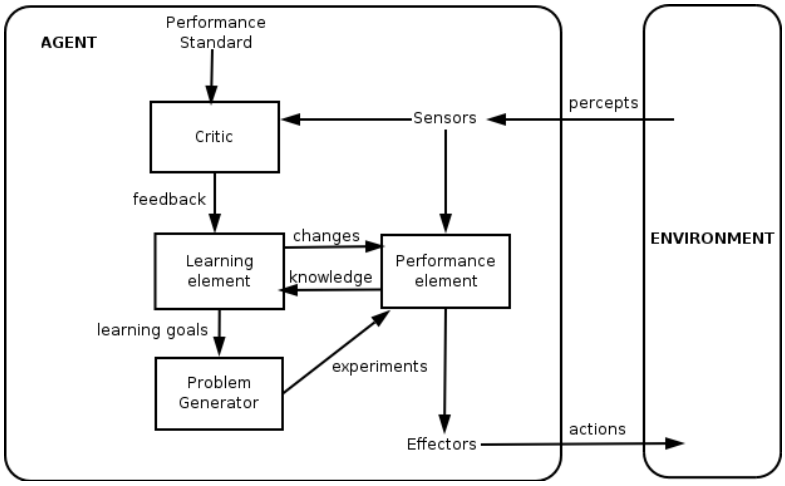


Figure 2.1: A learning agent.

A learning agent is composed of four elements. The *performance element* is responsible for selecting the action to make. The *learning element* is dedicated to making improvement in the action choice of the performance element. In other words, the performance element is what was considered the entire agent in the previous agent's types, because it receives a percept and selects the next action. But now the percept is also gathered by the *critic* which gives feedback to the learning element on how the agent is doing with respect to a performance standard and determines if the performance element should be modified to make better actions in the future. The last component is the *problem generator* which suggests to the performance element to

occasionally take exploratory actions that can lead to new and informative experiences.

In summary, learning agents are the most complex type of rational agent, but also the most promising. In order to design and implement a real learning agent, several techniques had been developed in the field of *reinforcement learning* (RL), and they are described in the next section.

2.2 The reinforcement learning approach

One of the goals of AI is to develop 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 [2]. Several complex tasks were successfully completed using reinforcement learning in multiple fields, such as games [3], robotics [4], and traffic signal control.

In a reinforcement learning problem, an autonomous agent observes the environment and get a state s_t, which is the state of the environment at time t. Then the agent chooses an action a_t which leads to a transition of the environment to the state s_{t+1}. After the environment transition, the agent obtains a reward r_{t+1} which tells the agent how good a_t was with respect of a performance measure. The goal of the agent is to learn the policy π^* that maximizes the cumulative expected reward obtained from rewards of actions taken while following π^*. The standard cycle of reinforcement learning is shown in Figure 2.2

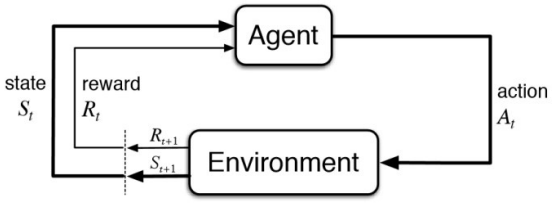


Figure 2.2: The reinforcement learning cycle.

2.3 Reinforcement learning for traffic signal control

Traffic signal control is a task that is well suited for RL techniques. 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. In this section, the most widely used approaches to design RL components (state, action, reward) in the context of traffic signal control are described.

2.3.1 Motivation

The use of RL for traffic signal control is motivated by several reasons [5]. First, if trained properly RL agents can adapt to different situations such as road accidents or bad weather conditions. Second, RL agents can self-learn without supervision prior knowledge of the environment. Third, a model of the environment that describes every variable of the environment is not needed since the agent learns using the system performance metric i.e. the reward.

RL techniques applied to traffic signal control address the following challenges: [5]

- **Inappropriate traffic light sequence**. Traffic lights usually choose the phases in a static, predefined policy. This method could cause the activation of an inappropriate traffic light phase in a situation that could cause an increase in travel times.
- **Inappropriate traffic light durations**. Every traffic light phase has a predefined duration which does not depend on the current traffic conditions. This behavior could cause unnecessary waitings for the green phase.

In this thesis the problem of *inappropriate traffic light sequence* will be addressed.

2.3.2 State representation

The state is the agent's perception of the environment in an arbitrary timestep. In literature, state space representations particularly differ in in-

formation density.

- **Low information density.** The lanes of the intersection are discretized in cells along the length of the lane. Lane cells are then mapped to cells of a vector, which marks 1 if a vehicle is inside the lane cell, 0 otherwise [6]. Another approach is to gather simple data about the intersection performance, such as the queue length of the incoming lanes [7]. Also, the occupancy of a lane and average speed can be used and they have the advantage to be easier to collect since just a simple detector is necessary [6].

- **Medium information density.** In this case, more data is collected. One of the most widely adopted state representation of this class is composed by the vector of presence cells but with the addition of a vector encoding the relative velocity of vehicles [8]. Also, the current traffic light phase could be added as a third vector [9].

- **High information density.** Some works rely on an image of the current situation of the whole intersection i.e. a snapshot of the simulator used. Then multiple snapshots will be usually stacked together to give the agent a sense of the vehicle motion [10]. Moreover, some approaches rely on every information that the intersection could provide, such as the queue length, the number of vehicles, the cumulative wait time, the current and next light phase, and an image of the intersection [11].

2.3.3 Action representation

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 and they are described below.

- **Low flexibility.** Among this group the agent has usually a defined set of light combination where he can choose from. When an action is selected, a fixed amount of time will pass before the agent can select a new action [8].

- **Medium flexibility.** Here again, the agent chooses the action from a defined set of light combinations. But, in this case, timings could be

flexible. For example, if the selected action is the same as the previous action, the current traffic light will last one more second, and then the agent has to choose the new action. If the action is different, the new action lasts a fixed amount of time [7].

- **High flexibility.** The agent chooses an action at every step of the simulation from a fixed set of light combinations. However, the selected action is not activated if the minimum amount of time required to release at least a vehicle has not passed [9] [10]. A different approach would be to have a defined cycle of light combinations activated into the intersection. The agent action is represented by the choice of when it is time to switch to the next light combination, and the decision is made at every step [11].

2.3.4 Reward representation

The reward is used by the agent to understand the consequence of the latest action taken in the latest state. It is usually defined as a function of some performance indicator of the intersection efficiently, such as vehicle delay, queue length, waiting time or throughput.

Most of the works include the calculation of the change between cumulative vehicle delay between actions, where the vehicle delay is defined as the number of seconds the vehicles is steady [9] [10]. Similarly, the cumulative vehicle staying time can be used, which is the number of seconds the vehicle has been steady since his entrance in the environment [8]. Moreover, some works combine multiples indicators in a weighted sum [11]. In some cases can be useful to consider a reward function that balances the queue lengths among the incoming road of the intersection [7].

2.4 Reinforcement learning algorithms for traffic signal control

The most recent reinforcement learning research has proposed multiple possible solutions to address the traffic signal control problem, in which it emerges that different algorithms and neural networks structure can be used, although some common techniques are necessary but not sufficient in order to ensure a good performance.

The most widely used algorithm to address the problem is Q-learning. The optimal behavior of the agent is achieved with the use of neural networks to approximate Q-values given a state. Often, this approach includes a Convolutional Neural Network (CNN) to compute the environment state and learn features from an image [10] or a spatial representation [9] [8].

Genders and Razavi [9] and Gao et al. [8] make use of a Convolutional Neural Network to learn features from their spatial representation of the environment. The output of this network with the current phase is passed to two fully connected layers that connect to the outputs represented by Q-values. This method showed good results in Gao et al. [8] work against different traffic lights policies, such as long-queue-first and fixed-times, while in Genders and Razavi [9] work it is compared to a shallow neural network, in which although it shows a good performance, an evaluation against real-world traffic lights would lead to more significant results.

Mousavi, Schukat, and Howley [10] analyzed a double approach to address the traffic signal control problem. The first approach is value-based, while the second is policy-based. In the first approach, action values are predicted by minimizing the mean-squared error of Q-values with the stochastic gradient-descent method. In the 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, wherein the value-based approach the output is the value of actions, and in the policy-based approach it is a probability distribution over actions. Results show that both the approaches achieve good performance against a defined baseline and do not suffer from instability issues. The authors indicate that future research could include multiple agents and intersections where it is necessary to use multi-agent techniques to handle coordination problems.

The research conducted by Wei et al. [11] make use of a deep neural network to compute the Q-value of switching to the next light phase or keeping active the current light phase. The deep neural network combines several intersection features as well as a visual representation of the environment, pre-computed by a CNN. The interesting functionality added by authors to the network structure is a separate learning process controlled by a switch neuron built inside the network that activates two different part of the network depending upon the current traffic phase. According to the authors, this

method is capable to enhance the fitting capability of the network. Tested on both simulation data and real-world data, this model achieves state-of-the-art performance on the majority of tests. Nevertheless, it should be noted that authors point out that the work has significant limitations on a real-world appliance due to the high volume of data gathered for the agent training.

In Li, Lv, and Wang [7] research, a deep stacked autoencoders (SAE) neural network is used to learn Q-values. An autoencoder is a neural network that sets the target output to be equal to the input. This approach use autoencoders to minimize the error between the encoder neural network Q-value prediction and the target Q-value by using a specific loss function. It is shown that achieves better performance than traditional RL methods.

The problem of Q-values convergence

In Q-learning, the ideal situation is when Q-values converge to stable values. This happens when the agent has efficiently explored the state space and he has updated the neuron's weight correctly. In order to maximize the chance of Q-values convergence, most of the reviewed works [9] [10] [8] [7] make use of one or more techniques that helps the agent to learn more efficiently, such as *experience replay* [12] (see Section 4.1 on page 33), *target network* and ϵ-*greedy exploration policy* (see Section 4.3 on page 38). In particular, regarding experience replay, Wei et al. [11] propose an evolution of the technique that uses multiple memory containers. Every sample will be stored in a different memory container depending on the phase-action pair. When the agent trains, he sample the same amount of samples from every container in order to learn from a balanced set of experiences. This method has shown to improve the fitting capability of the network to predict the Q-values accurately in the author's work.

2.4.1 Directions of future research

As future works, researchers advise that the agent action should also comprehend yellow and red phases control. Moreover, the research should be more focused on analyzing and optimizing the fairness of the agent's policy from a single car's perspective, instead of only increase the intersection performance as a whole by minimizing the delay or maximizing the throughput.

Reinforcement learning techniques for traffic signal control have matured significantly in the last years, especially with the spreading of Deep Learning. Even if results are promising, there is still much more work to do in order to advance beyond simulations. Regarding a real-world appliance, a hypothetical deployment of a reinforcement learning traffic signal controller should be trained on both online and offline data, to be able to adapt to specific scenarios after it has learned a standard traffic-efficient policy [13]. Also, the lack of a standardized dataset of traffic to use to compare experimental results does not help to achieve a real-world appliance.

Chapter 3

Problem statement and design of the agent

In this thesis, the chance of improvement in traffic flow that drives through an intersection controlled by traffic lights will be investigated using artificial intelligence techniques. The analysis will be conducted with a simulation where an agent manages the choice of which traffic light's phase activate with the objective of optimizing the traffic efficiency. In order to choose the best light phase in every situation, some learning mechanism is required by the agent. The learning techniques used in this thesis are related to reinforcement learning and deep learning. The entire system that comprehends the agent, its elements, and the learning techniques in this thesis is called *Traffic Light Control System (TLCS)* and is described in this chapter.

In order to design a system based on the reinforcement learning framework, it is necessary to define the environment, a state representation, an action space, a reward function and the agent learning techniques involved.

In Table 3.1 are listed the terms used in this chapter and in the next chapter to describe the reinforcement learning elements.

Notation	Meaning
TLCS	Traffic Light Control System.
A	Set of actions.
a	A single action.
s	State.
r	Reward.
TL	Set of traffic lights.
IDR	Intersection Discretized Representation.
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.
t	Timestep, the precise moment when the agent interact with the simulation.
h	An episode.
H	Total number of episodes.
B	Batch.
b	a sample contained in the batch.

Table 3.1: Notations.

3.1 Problem definition

In this thesis the environment is represented by a 4-way intersection, where 4 lanes per arm approach the intersection from the compass directions and lead to 4 lanes per arm leaving the intersection. A set of traffic lights TL manages the incoming traffic into the intersection. The TLCS is composed by a single agent that interacts with the environment using a state s, an action a and a reward r. A deep Q-learning neural network is the learning mechanism of the agent. Figure 3.1 shows a summary of the TLCS.

During the simulation, the agent samples the environment and receives a state s_t and a reward r_t, where t is the current timestep. According to the state s_t and the prior knowledge, the agent chooses the next action a_t. At the same time, the agent learns about the consequence of the action a_{t-1} taken in the state s_{t-1} using the reward r_t and the arriving state s_t. This knowledge will be used to train the agent (i.e. the neural network) in order to acquire significant awareness about the consequence of future actions in similar states.

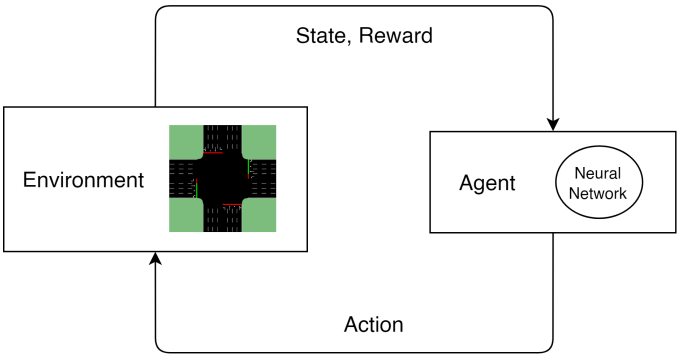


Figure 3.1: Generic working process of the TLCS.

The problem is defined as follows: given the state of the intersection s, what is the traffic light phase a that the agent should choose, selected from a fixed set of predefined actions A, in order to maximize the reward r and optimize the traffic efficiency of the intersection.

It should be noted that in this thesis the possibility of a real-world appliance of the TLCS will be taken into consideration when designing the elements of the agent, so that is not used any elements that could be hard to implement with the currently available technology.

3.2 The environment of the simulation

The simulated environment of this thesis is the intersection represented in Figure 3.2.

It consists of a four-way intersection where there are 4 lanes approaching the intersection from the compass directions connected to 4 lanes leaving the intersection. Each arm of the junction is 750 meters long from the vehicle origin to the intersection's stop line. On every arm, the four lanes used for entering the junction indicate the possible directions that a car can take. When a vehicle approaches the junction, it selects the desired lane based on its destination:

- Turn left: select only the left-most lane.
- Go straight ahead: select the two central lanes or the right-most lane.
- Turn right: select only the right-most lane.

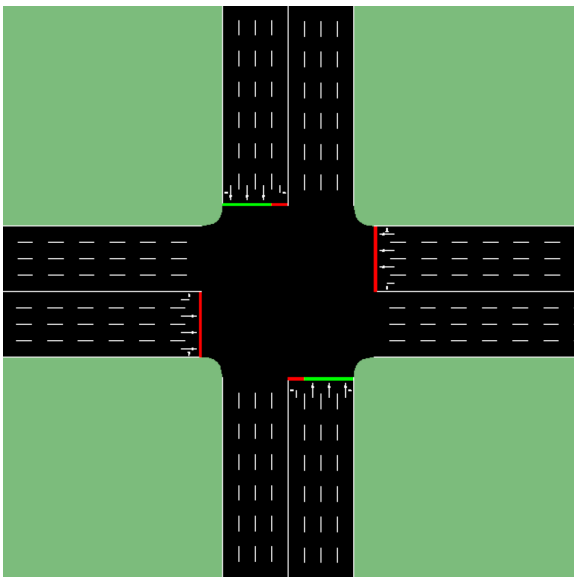


Figure 3.2: A zoomed view of the intersection without vehicles.

The traffic light system

Traffic lights in the environment are indicated by a color on the stop line of every entrance lane, which represents the status of the traffic light for that lane on a precise timestep. For example, Figure 3.2 shows a green light for vehicles coming from the north or south direction that want to go straight or turn right, and red for everyone else. Every possible traffic-light combination will be detailed in Section 3.4 on page 22.

In the environment there are 8 different traffic lights, each of them regulating one or more adjacent lanes. They are shown in the equation (3.1).

$$\mathrm{TL} = \{tl_{\mathrm{N}}, tl_{\mathrm{NL}}, tl_{\mathrm{W}}, tl_{\mathrm{WL}}, tl_{\mathrm{S}}, tl_{\mathrm{SL}}, tl_{\mathrm{E}}, tl_{\mathrm{EL}}\} \tag{3.1}$$

Where the subscript denotes the position of every traffic light. For example, tl_{N} is the traffic light that regulates all the traffic coming from north that wants to turn right or go straight. Alternately, tl_{NL} regulates traffic coming from north but just for vehicles that want to turn left. The same rule applies to every traffic light defined in set (3.1). A representation of every traffic light and their positions in the environment is showed in Figure 3.3

Like in traditional traffic lights, a traffic light tl of the simulation has 3

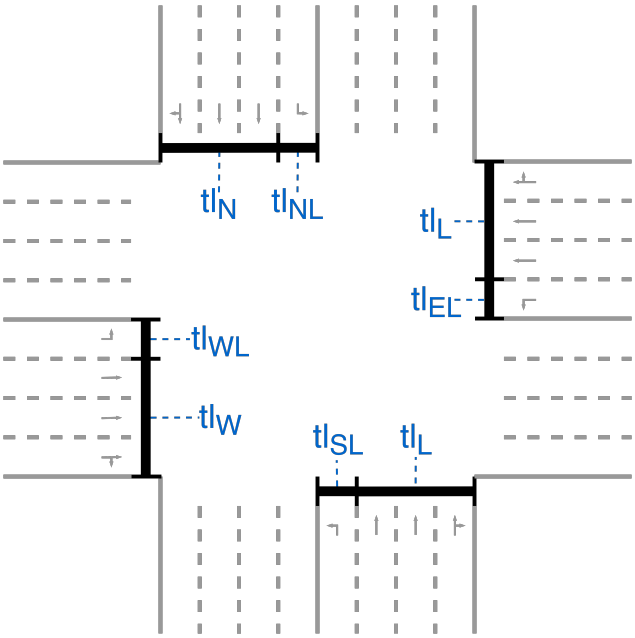


Figure 3.3: Position of every traffic light in the environment.

possible states as described in the set (3.2).

$$\{\text{green}, \text{yellow}, \text{red}\} \tag{3.2}$$

Every traffic light in the environment works accordingly to the following rules:

1. The color phase transition is always the following: red-green-yellow-red.
2. The duration of every traffic light phase is fixed. The green time is always 10 seconds and the yellow time is always 4 seconds. Consequently, the duration of the red phase is defined as the amount of time since the last phase change.
3. For every timestep, at least one traffic light is in yellow phase or green phase.
4. It is not possible to have every traffic light in the red phase simultaneously.

3.3 The state representation

The state of the agent describes a representation of the situation of the environment in a given timestep t and it is denoted with s_t. To allow the agent to effectively learn to optimize the traffic, the state should provide sufficient information about the distribution of cars on each road.

The objective of this representation is to let the agent knows the position of vehicles inside the environment at timestep t. For this purpose the approach proposed in this thesis is inspired to the DTSE [9] (see Section 2.4 on page 11), with the difference that less information is encoded in this state. In particular, this state design includes only spatial information about the vehicles hosted inside the environment, and the cells used to discretize the continuous environment are not regular. The chosen design for the state representation is focused on realism: recent works on traffic signal controller proposed information-rich states, but in reality they hard to implement since the information used in that kind of representations is difficult to gather. Therefore, in this thesis will be investigated the chance of obtaining good results with a simple and easy-to-apply state representation.

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 3.4 shows the state representation for the west arm of the intersection.

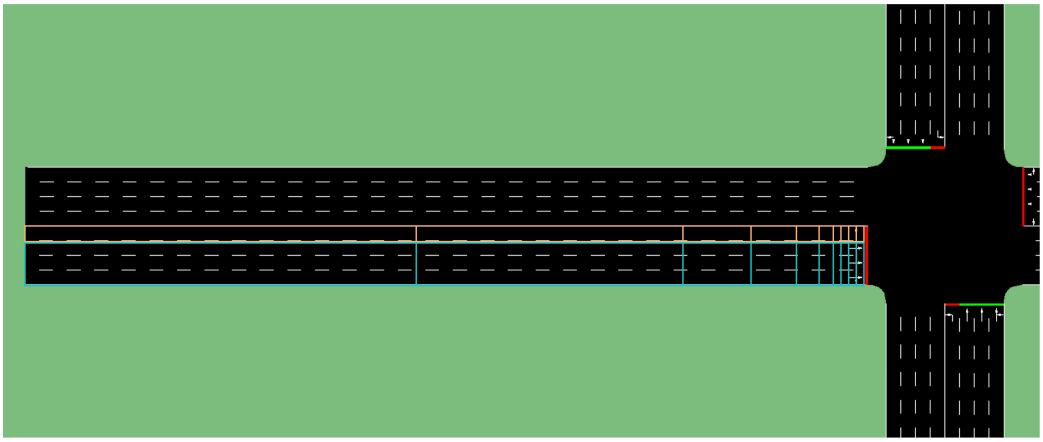


Figure 3.4: The state representation in the west arm of the intersection.

It must be noted that a particular cell does not necessarily describe the situation in a single lane. As seen in Figure 3.4, in fact, the 3 lanes dedicated

to going straight and turning right share the same cells since they share the same traffic light, while the lane dedicated to turning left has a separate set of cells. Along the length of a lane there are 10 cells, which means that in every arm of the intersection there are 20 cells and in the whole intersection 80 cells. As seen in Figure 3.5, 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 choice of the length of every cell is not trivial: if cells were too long, some cars approaching the crossing line may not be detected; if cells were too short, the number of states required to cover the length of the lane increases, bringing to higher computational complexity. In this thesis, the length of the shortest cells, which are also the closest to the stop line, is exactly 2 meters longer than the length of a car.

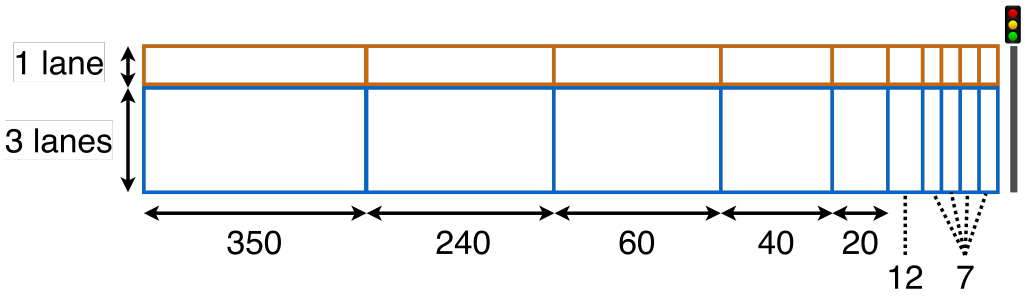


Figure 3.5: Design of the state representation in the west arm of the intersection, with cells length. The lenght of the longer cells has been reduced for readability.

The Intersection Discretized Representation

Formally, is defined a vector IDR (Intersection Discretized Representation) as the mathematical representation of the state space, where every element IDR_k is computed following equation (3.3).

$$\text{IDR}_k = c_k \tag{3.3}$$

Where c_k is the k-th cell. This means that every cell c is mapped to a position of the vector IDR. The vector IDR is updated following the rule (3.4)

$$\text{IDR}_k = 1 \quad \text{if there is more than one vehicle inside } c_k, \quad 0 \text{ otherwise.} \tag{3.4}$$

When the agent sample the environment at a timestep t, it receives a vector IDR_t containing the discretize representation of the environment in that timestep. This is the principal information about the environment that the agent receives, so it is designed in order to be as much precise as possible but without being excessively detailed in order to not increase the computational complexity of the neural network's training. In reinforcement learning, the time that the agent takes to explore the state space is crucial with respect to the performance of the agent itself: if he does not explore a significant part of the state space, it will not be capable to estimate correctly what is the best action in every situation. After the training, the agent should be able to choose the best action even in an unseen state because, within his experience, it resembles a similar state for which he knows the goodness of every action. This means that the design of the state space should be suitable for the expected learning time of the agent.

The proposed state space is composed of 80 boolean cells. This means that the number of possible states is 2^{80}. The choice of boolean cells for the environment representation is also crucial because the agent has to explore just the most significant subset of the state space in order to learn the best behavior. For example, a critical state of the environment is when there is at least one vehicle stopped, waiting for the green phase. This is a state where the best action is needed the most, in order to let the vehicles go and improve the overall intersection efficiency. In other words, the cells that are closer to the stop line are more important than the cells further. This means that the combinations of states where there are active cells closer to the stop line contribute more at the good performance of the agent, reducing significantly the size of the state space to a quantity where the training duration is in line with expectations. The specific training durations are detailed in Chapter 5.

3.4 The action set

The action set identifies the possible actions that the agent can take. The agent is the traffic light system, so doing an action translates to turning green some traffic lights for a set of lanes and keep it green for a fixed amount of time. As described in Section 3.2, the green time is set at 10 seconds and the yellow time is set at 4 seconds. In other words, the task of the agent is to initiate a green phase choosing from the predefined ones. The action space

is defined in the set (3.5).

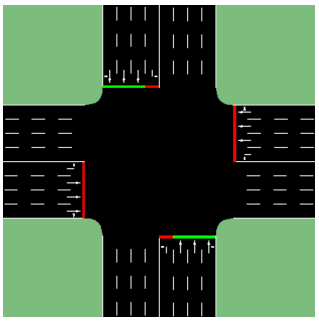
$$A = \{\text{NSA}, \text{NSLA}, \text{EWA}, \text{EWLA}\} \tag{3.5}$$

The set represents every possible action that the agent can take. Every action a of set (3.5) is described below.

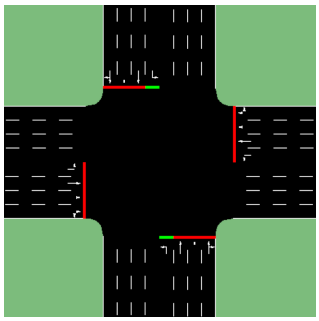
- North-South Advance (NSA): the green phase is active for vehicles that are in the north and south arm and wants to proceed straight or turn right.
- North-South Left Advance (NSLA): the green phase is active for vehicles that are in the north and south arm and wants to turn left.
- East-West Advance (EWA): the green phase is active for vehicles that are in the east and west arm and wants to proceed straight or turn right.
- East-West Left Advance (EWLA): the green phase is active for vehicles that are in the east and west arm and wants to turn left.

Figure 3.6 shows a visual representation of the 4 possible actions.

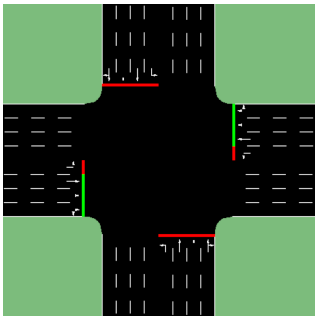
If the action chosen in timestep t is the same as the action taken in the last timestep $t - 1$ (i.e. the traffic light combination is the same), there is no yellow phase and therefore the current green phase persists. On the contrary, if the action chosen in timestep t is not equal to the previous action, a 4 seconds yellow phase is initiated between the two actions. This means that the number of simulation steps between two same actions is 10 since 1 simulation step is equal to 1 second in SUMO Simulator (see Section 4.4 on page 38). When the two consecutive actions are different, the yellow phase counts as 4 simulation steps and then the selected action counts as 10 simulation steps, for a total of 14 simulation steps. Figure 3.7 shows a brief overview of this process.



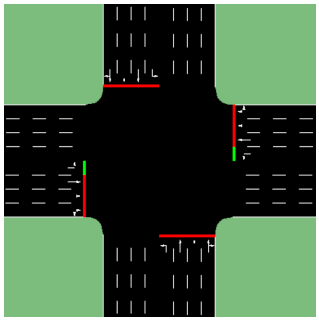
(a) North-South Advance



(b) North-South Left Advance



(c) East-West Advance



(d) East-West Left Advance

Figure 3.6: The four possible actions.

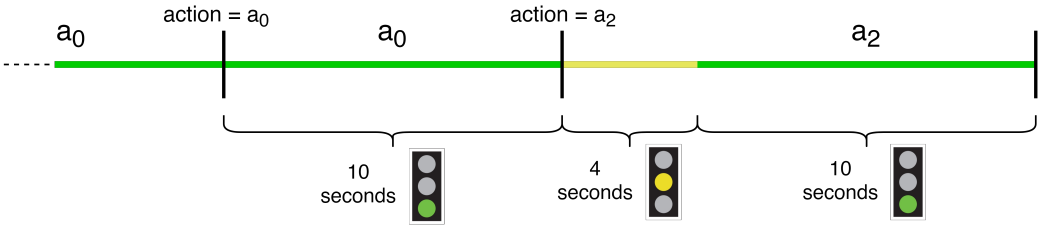


Figure 3.7: Possible differences of simulation steps between actions.

3.5 The reward function

In reinforcement learning, the reward represents the feedback from the environment after the agent has chosen an action. The agent uses the reward to understand the result of the taken action and improve the model for future action choices. Therefore, the reward is a crucial aspect of the learning process. The reward usually has two possible values: positive or negative. A positive reward is generated as a consequence of good actions, a negative reward is generated from bad actions. In this application, the objective is to maximize the traffic flow through the intersection over time. In order to achieve this goal, the reward should be derived from some performance measure of traffic efficiency, so the agent is able to understand if the taken action reduce or increase the intersection efficiency. In traffic analysis, several measures are used [14], such as throughput, mean delay and travel time. In this thesis, the candidate measures to generate the reward were the following.

- Queue length: the number of vehicles with a speed of less than 0.1 m/s.
- Total waiting time: the sum of individual waiting times of each car in the environment in timestep t. Each waiting time is defined as the amount of time a vehicle has a speed of less than 0.1 m/s.
- Throughput: the number of vehicles that crosses the intersection over a defined period of time.

Among the proposed, the chosen measure is the total waiting time. Formally, the total waiting time is defined in equation (3.6).

$$twt_t = \sum_{veh=1}^{n} wt_{(veh,t)} \tag{3.6}$$

Where twt_t is the total waiting time at timestep t and $wt_{(veh,t)}$ is the amount of time in seconds a vehicle veh has a speed of less than 0.1 m/s at timestep t. n represents the total number of vehicles in the environment in timestep t. The most efficient intersection is the one that prevents cars from waiting for the green phase. Therefore, the concept of waiting time is crucial for the choice of the reward measure. The total waiting time is the most accurate measure among the proposed. In fact, the queue length and the throughput

does not take into consideration the time spent by cars in a stationary position, but just the fact that a car is stopped. Consequently, the total waiting time is the chosen measure.

The reward functions defined for the agent in this thesis are two. The first reward function is defined following the approach of Genders and Razavi [9], while the second reward function is a slight modification of the first with the purpose of improving the training efficacy of the agent.

3.5.1 Literature reward function

The first reward function that generates the reward for the agent at timestep t is called *literature reward function* and is defined in equation (3.7).

$$r_t = twt_{t-1} - twt_t \tag{3.7}$$

Where r_t represents the reward at timestep t. twt_t and twt_{t-1} represent the total waiting time of all the cars in the intersection captured respectively at timestep t and $t-1$.

As said before, the reward usually can be positive or negative, and this implementation is no exception. The equation 3.7 is designed in such a way that when the agent chooses a bad action it returns a negative value and when it chooses a good action it returns a positive value. A bad action can be represented as an action that, in the current timestep t, adds more vehicles in queues compared to the situation in the previous timestep $t-1$, resulting in higher waiting times compared to the previous timestep. This behavior increases the twt for the current timestep t and consequently the equation 3.7 assumes a negative value. The more vehicles were added in queues for the timestep t, the more negative r_t will be and therefore the worst the action will be evaluated by the agent. The same concept is applied for good actions.

3.5.2 Proposed reward function

The second reward function that generates a reward for the agent is called *proposed reward function* and is defined in equation (3.8)

$$r_t = 0.9 \cdot twt_{t-1} - twt_t \tag{3.8}$$

with the notations being the same as the first reward function. The only difference with respect to the former is the 0.9 factor. This modification

is made to increase the magnitude of the received reward, both positive or negative. The effect of this variation is that the agent has a stronger representation about the consequences of actions and therefore the effectiveness of the training is improved.

In subsection 3.6.1 the effects of the two reward functions are visualized and described in terms of cumulative rewards.

3.6 The agent's learning mechanism

The learning mechanism involved in this thesis is called Deep Q-Learning, which is a combination of two aspects widely adopted in the field of reinforcement learning: deep neural networks and Q-Learning.

3.6.1 Q-Learning

Q-Learning [15] is a form of model-free reinforcement learning [16]. It consists of assigning a value, called the *Q-value*, to an action taken from a precise state of the environment. Formally, in literature, a Q-value is defined as in equation (3.9).

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \cdot max_A Q(s_{t+1}, a_t) - Q(s_t, a_t)) \tag{3.9}$$

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. Inside the parenthesis, the term r_{t+1} represents the reward associated to taking action a_t from state s_t. The subscript $t+1$ is used to emphasize the temporal relationship between taking the action a_t and receiving the consequent reward. The term $Q(s_{t+1}, a_t)$ represents the immediate future's Q-value, where s_{t+1} is next state in which the environment has evolved after taking action a_t in state s_t. The expression max_A means that, among the possible actions a_t in state s_{t+1}, the most valuable is selected. The term γ is the discount factor that assumes a value between 0 and 1, lowering the importance of future reward compared to the immediate reward.

In this thesis, a slightly different version of the equation (3.9) is used and it is presented in equation (3.10)

$$Q(s_t, a_t) = r_{t+1} + \gamma \cdot max_A Q'(s_{t+1}, a_{t+1}) \tag{3.10}$$

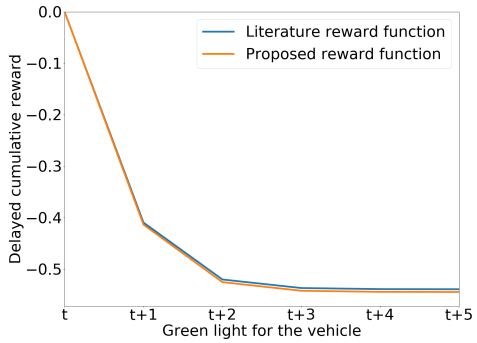
Where the reward r_{t+1} is the reward received after taking action a_t in state s_t. The term $Q'(s_{t+1}, a_{t+1})$ is the Q-value associated with taking action a_{t+1} in state s_{t+1}, i.e. the next state after taking action a_t in state s_t. As seen in equation (3.9), the discount factor γ denote a small penalization of the future reward compared to the immediate reward.

The above equation (3.10) is a rule that update the Q-value of the current action a_t taken in state s_t with the immediate reward and the discounted Q-value of future actions. Therefore, the term $Q'(s_{t+1}, a_{t+1})$ that represents the value of future actions implicity holds the maximum discounted reward of the state after s_{t+1}, i.e. $Q''(s_{t+2}, a_{t+2})$ and likewise, it holds the maximum discounted reward for the next state which is $Q'''(s_{t+3}, a_{t+3})$ and so on. This is how the agent can choose the action a_t based on not just the immediate reward, but also based on the expected future discounted rewards. For simplicity, the rule can be unpacked and the result is the equation (3.11)

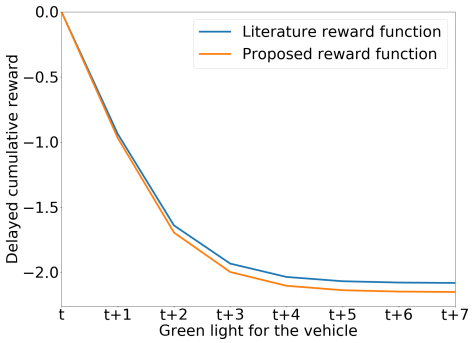
$$Q(s_t, a_t) = r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \gamma^3 \cdot r_{t+4} + ... + \gamma^{y-1} \cdot r_{t+y} \qquad (3.11)$$

where y is an arbitrary value that just indicates the last timestep before the end of the episode, where there are no more future actions and therefore the future reward is at 0. In figures 3.8a, 3.8b and 3.8c is shown a visual representation of the expanded Q-learning equation (3.11) for a γ value of 0.09, 0.25 and 0.75 respectively. The figures represent a situation where a single vehicle approaches the intersection line 5 seconds before the next agent's action, that is a_t. On the x-axis the action when the agent will set the green light for the vehicle is defined. In particular, the first value t means that the traffic light is already green so the vehicle can proceed without stopping. For instance, if the green phase is activated at $t+1$ the vehicle waits 5 seconds, so in equation (3.11) the $Q(s_t, a_t)$ is equal to$-5 + 0.75 * 5 = -1.25$ (using the literature reward function and $\gamma = 0.75$). The purpose of these graphs is to understand of the delayed reward works: when γ is set at 0.09, the agent has a short lookahead and he accounts only a few future actions to compute the value of the current action. When γ is set at 0.25, the agent accounts more actions and with γ equal to 0.75, the result is that almost every future action contributes to the current action value. Moreover, the difference between the two reward functions is showed especially with $\gamma = 0.75$ and it will be proved that help significantly the agent to achieve better performance with a long lookahead.

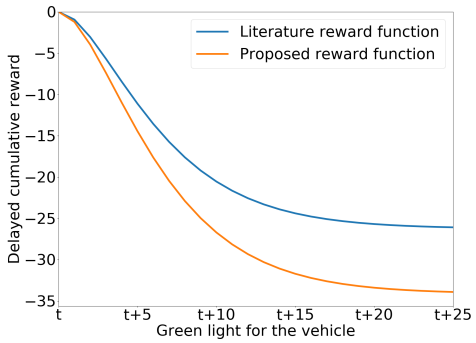
The expanded Q-value equation described here is to make clear the type of reward that the agent receives. As said before, the future rewards made explicit in this paragraph are contained in $Q'(s_{t+1}, a_{t+1})$, the last term of equation (3.11). The process of future rewards update will be described in subsection 4.2 on page 35.



(a) $\gamma = 0.09$



(b) $\gamma = 0.25$



(c) $\gamma = 0.75$

Figure 3.8: Cumulative rewards for one waiting vehicle.

3.6.2 Deep Q-Learning

In order to map a state of the environment s_t to Q-values representing the values associated with actions a_t, a deep neural network is built. The input of the network is the vector IDR_t the state of the environment at timestep

t. The outputs of the network are the Q-values of the possible action from state s_t.

Formally, the input of the neural network n^{in} is defined in (3.12).

$$n^{in}_{k,t} = \text{IDR}_{k,t} \tag{3.12}$$

Where $n^{in}_{k,t}$ is the k-th input of the neural network at timestep t and $\text{IDR}_{k,t}$ is the k-th element of the vector IDR at timestep t. This means that $|n^{in}| = |\text{IDR}| = 80$, that is the input size of the neural network.

The output of the neural network n^{out} is defined in (3.13).

$$n^{out}_{j,t} = Q(s_t, a_{j,t}) \tag{3.13}$$

Where $n^{out}_{j,t}$ is the j-th output of the neural network at timestep t and $Q(s_t, a_{j,t})$ is the Q-value of the j-th action taken from state s_t at timestep t. This means that the output cardinality of the neural network is $|A| = 4$, where A is the action space.

The neural network is a fully connected deep neural network with rectified linear unit activation function (ReLU) [17]. The exact number of layer and the number of neurons per layer are specified in Chapter 5, where multiple agent configurations are tested. A scheme of the neural network is shown in Figure 3.9.

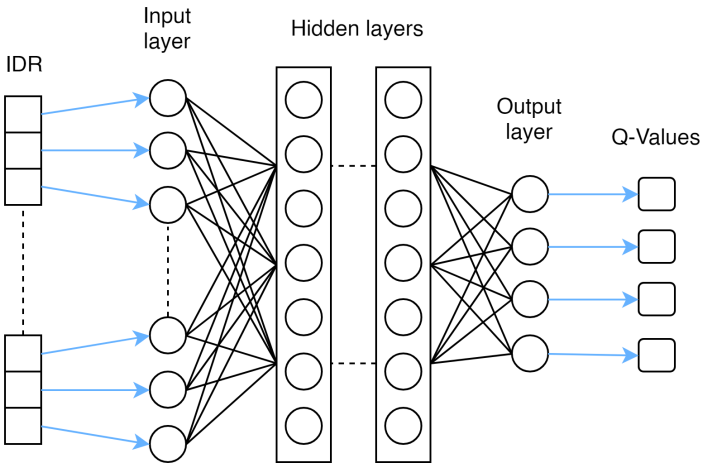


Figure 3.9: Scheme of the deep neural network.

As explained in this section, Figure 3.9 shows the vector IDR as the input of the network, then the network itself with the hidden layers and finally the

output layer with 4 neurons representing the 4 Q-values associated to the 4 possible actions.

Chapter 4

Experimental setup and training

In the previous chapter, the specification of the agent was described, such as the state, the possible actions, and the reward. Figure 4.1 shows how all these components work together to establish the workflow of the agent during one single timestep t.

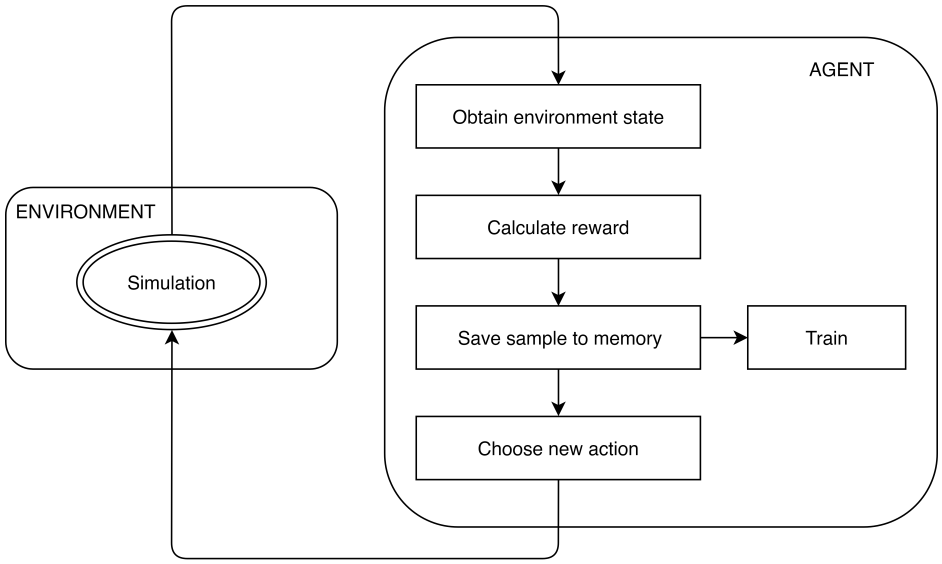


Figure 4.1: The workflow of the agent in a timestep (Light sample strategy)

After a fixed amount of simulation steps, the timestep t of the agent begins. First, the agent retrieves the environment state and the delay times.

Next, using delay times of this timestep t and from the last timestep $t-1$, it calculates the reward associated with action taken at $t-1$. Then the agent pack the information gathered and save it to a memory which is used for training purposes, as described in next Section 4.1. Finally, the agent chooses and set the new action to the environment and a new sequence of simulation step begins. In this work, the agent is trained by using a traffic micro-simulator. The agent will be trained by submitting multiple episodes which consist of traffic scenarios, where he will gather experience from. One episode consists of 5400 steps, which translates to 1 hour and 30 minutes of traffic simulation.

In order to measure the effectiveness of the design choices defined in Chapter 3, the agent is trained and tested with the use of a traffic micro-simulator. In this chapter, every decision regarding the simulation phase is described.

4.1 Experience Replay

Experience replay [12] is a technique adopted during the training phase in order to improve the performance of the agent and the learning efficiency. It consists of submitting to the agent the information needed for learning in the form of a randomized group of samples called *batch*, instead of immediately submitting the information that the agent gather during the simulation (commonly called *Online Learning*). The batch is taken from a data structure intuitively called memory, which stores every sample collected during the training phase. A sample m is formally defined as the quadruple (4.1).

$$m = \{s_t, a_t, r_{t+1}, s_{t+1}\} \tag{4.1}$$

Where r_{t+1} is the reward received after taking the action a_t from state s_t, which evolves the environment into the next state s_{t+1}. A training instance involves the gathering of a group of samples from the memory and the neural network training using the aforesaid samples. In Figure 4.2 is shown a representation of the interactions with the memory.

As said earlier, the experience replay technique needs a memory, which is characterized by a memory size and a batch size. The memory size represents how many samples the memory can store and is set at 50000 samples. The batch size is defined as the number of samples that are retrieved from the memory in one training instance. If at a certain timestep the memory

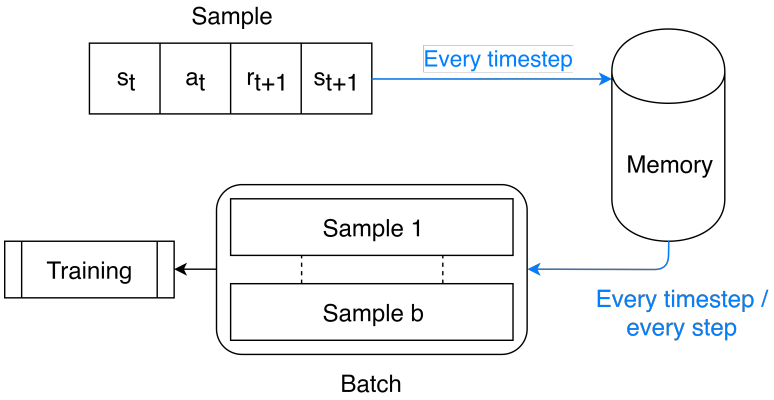


Figure 4.2: The memory handling before training.

is filled, the oldest sample is removed to make space for the new sample. Therefore, if a sample is inserted into the memory at every timestep, the number of episodes that a single sample remains stored in the memory before its elimination is approximately 120 (this calculation i). The number of times a sample is gathered for a training instance depends on the batch size and the frequency of the training instances, as explained below.

4.1.1 Sampling strategies

In Figure 4.2 two types of samples gathering from the memory are shown. This is because in this thesis two sampling strategy, and so training strategy, are implemented:

- **Light:** The first strategy is to (i) initiate the training phase at every timestep and (ii) set the batch size at 50 samples. This is identified as a light training phase. In an episode of 5400 steps there are approximately 415 timesteps; with a batch size of 50, the number of samples gathered for the training in an episode is approximately 20750, which is almost half the memory size.

- **Intensive:** The second strategy is to (i) initiate the training phase at every simulation step, instead of timesteps, and (ii) set the batch size at 100 samples. With this more intense training, the agent gathers approximately 540000 samples per episode, which is more than 10 times the memory size. The purpose of this strategy is to learn a more stable

policy by train the agent's neural network with more examples in a shorter time span.

The results of the application of the two strategies to the agent are described in Chapter 5.

4.1.2 Advantages

With the use of experience replay, the training phase has two major advantages:

- It removes correlations in the observation sequence [18].
- Refresh the experience of the agent [12].

In this environment, two consecutive states are naturally correlated since the state of the environment s_{t+1} is a direct evolution of the state s_t. Most of the information contained in state s_{t+1} does not derive as a consequence of an agent's action, but rather as the spontaneous transformation of the current situation s_t. Therefore experience replay has been implemented to not inducing misleading correlation in the agent's neural network. Secondly, during the training process, there is the possibility that the neural network forgets the knowledge gained about a situation visited in the early stages of the training. By using experience replay, the agent occasionally receives a "refresh" of what it has learned before in an older state.

4.2 The training process

Given the description of experience replay, a detailed explanation of the training process is presented. This process is executed every time a training instance of the agent is initiated.

1. A sample m containing the most recent information, described in the previous equation (4.1), is added to the memory.
2. A fixed number of samples, depending on the sampling strategy used, are picked randomly from the memory constituting the batch B (see Figure 4.2 on the preceding page).

A single sample $b_k \in B$ contains the starting state s_t, the action made a_t, the consequent reward received r_{t+1} and the following state s_{t+1}. For every sample b_k the following operations are performed.

1. Computation of the Q-value $Q(s_t, a_t)$ by submitting the vector IDR representing s_t to the neural network and obtaining the predicted Q-value relative to action a_t.

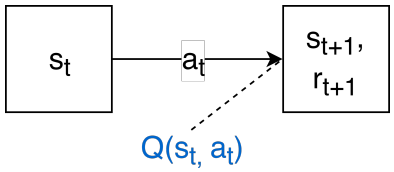


Figure 4.3: Computation of the Q-value Q for one sample.

2. Computation of Q-values $Q'(s_{t+1}, a_{t+1})$ by submitting the vector IDR representing the next state s_{t+1} to the neural network and obtaining the predicted Q-values relative to actions a_{t+1}. These represents how the environment will evolve and what values will probably have the next actions.

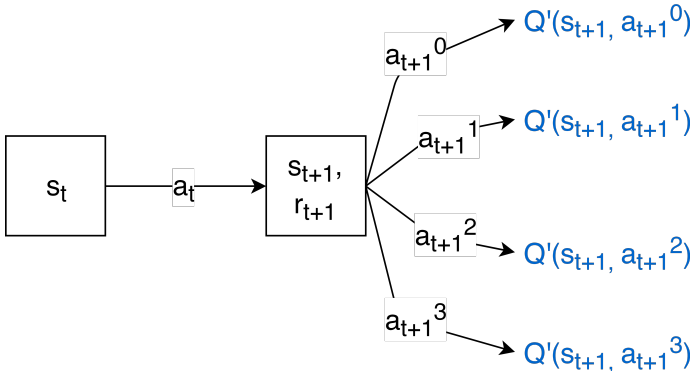


Figure 4.4: Computation of Q-values Q' for one sample.

3. Update of the Q-value using equation (3.10) reported below in equation (4.2). Among the possible future Q-values computed in stage two, $max_A Q'$ indicates that the best possible Q-value is selected, represent-

ing the *maximum* expected future reward. It will be discounted by a factor γ that gives more importance to the immediate reward.

$$Q(s_t, a_t) = r_{t+1} + \gamma \cdot max_A Q'(s_{t+1}, a_{t+1}) \tag{4.2}$$

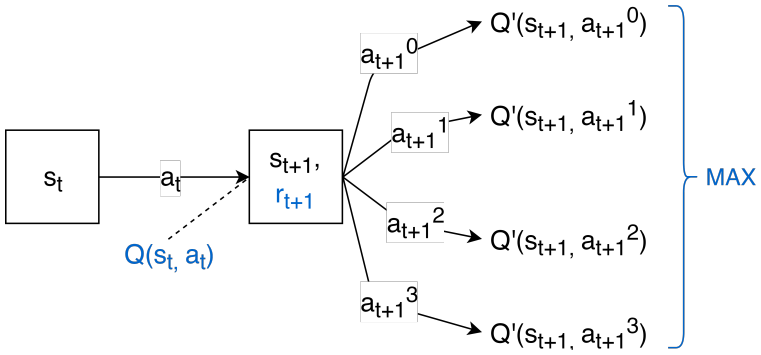


Figure 4.5: Informations used (in blue) for the Q-value update using equation (4.2) for one sample.

4. Training of the neural network. The input is the vector IDR representing the state s_t, while the desired output is the updated Q-values $Q(s_t, a_t)$ that now includes the maximum expected future reward thanks to the Q-value update equation (4.2). By doing this, the next time the agent encounters the state s_t or a similar one, the neural network will be likely to output the Q-value of action a_t that is comprehensive of the best future situation.

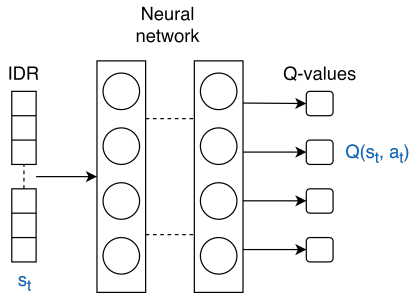


Figure 4.6: Training of the neural network for one sample.

4.3 The exploration/exploitation tradeoff

The training phase of the agent consists of finding the most valuable actions given a state of the environment. With that said, in the early stages of the training the agent does not know which actions are the most valuable ones. In order to overcome this problem, at the beginning of the training the agent should discover the consequences of the actions and do not worry about the performance. Once the agent has a solid knowledge about the actions outcomes in a significant variety of states, it should increase the frequency of exploitative actions in order to find the most valuable ones and consequently increase the performance achieved in the task.

In this thesis, a simple yet effective equation is used to set the probability to choose an explorative action or an exploitative action, at a certain episode h. It is called ϵ-*greedy*. The equation, shown in (4.3) define a probability ϵ for the current episode h to choose an explorative action, and consequently a probability $1 - \epsilon$ to choose an exploitative action.

$$\epsilon_h = 1 - \frac{h}{H} \tag{4.3}$$

where h is the current episode of training and E is the total number of episodes.

4.4 Simulation of Urban MObility (SUMO)

The traffic microsimulator used for this thesis is SUMO [19] [20] (Simulation of Urban MObility). SUMO provide a software package that enables the user to design every element of the desired road infrastructure. Among the packages that SUMO offers, in this thesis the following were used.

First, the visual editor *NetEdit* was used to design the static elements of the intersection, such as the roads characteristics, the distribution of traffic lights and the lane connections across the intersection (see Section 3.2 on page 17).

Then, thanks to the SUMO package *TraCI* (Traffic Control Interface) it was possible to define the type, characteristics, and generation of vehicles that are going to be into the simulation. Moreover, Traci made possible to interact with the simulation during run-time in order to retrieve the status

of the intersection at every timestep and then set the action chosen by the agent.

Finally, the tool *SUMO-GUI* let the user experience a graphical representation of a simulation with the possibility to slow down or accelerate the simulation speed. This tool was used to check the performance of the agent.

In a SUMO simulation, 1 step is equal to 1 second. For this work, one episode consists of 5400 steps, which translates to 1 hour and 30 minutes of simulation.

4.5 Traffic generation

In a simulated environment, the traffic generation is a crucial part that can have a big impact on the agent's performance. In order to maintain a high degree of reality, during the training phase the traffic will be generated according to a Weibull distribution with a shape equal to 2. An example is shown in Figure 4.7. The distribution is presented in the form of a histogram, where on the x-axis are defined the steps of one simulation episode and on the y-axis is defined the number of vehicles generated in that step window. The Weibull distribution is chosen because it can approximate specific traffic situations, where during the early stage the number of cars is rising, representing a peak hour. Then, the number of incoming cars slowly decreases describing the gradual mitigation of traffic congestion.

The distribution in Figure 4.7 provides the exact steps of the episode in which a car will be generated. For every car scheduled, the source and destination are set using a random number generator that will have the seed changed for every new episode. In order to maximize the agent's performance, the simulation should include a significant variety of traffic flows and patterns [21]. Therefore, four different scenarios are defined and they are the following. Each scenario corresponds to one single episode.

- High-traffic scenario, 4000 cars generated.
- Low-traffic scenario, 600 cars generated.
- NS-traffic scenario, 2000 cars generated.
- EW-traffic scenario, 2000 cars generated.

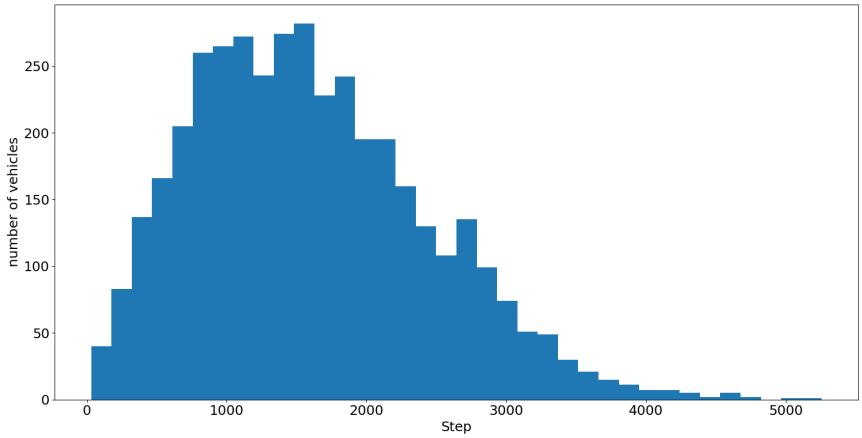


Figure 4.7: Traffic generation distribution over one episode.

Now it will be described how source and destination are set for every car in the four scenarios. It will be considered only the point of view of a single car for simplicity. Every car has the same generation probabilities.

In High-traffic and Low-traffic scenario, the probabilities are the same. First, the car has a probability of 75% to go straight and 25% to turn left or right. Then, source and destination are chosen with a uniform probability. Table 4.1 shows the exact probability distribution for every source and destination combination in the High-traffic and Low-traffic scenario.

NS-traffic and EW-traffic scenarios are meant to increase the knowledge of the agent about a situation where most of the traffic comes from the two opposite compass directions. Like in previous scenarios, a car has a probability of 75% of going straight and 25% of turning left or right. The difference is that in NS-traffic a car has a probability of 90% of coming from north or south and 10% of coming from east or west. On the contrary, in EW-traffic scenario a car has a 90% chance of coming from east or west and 10% chance of coming from north or south. Given this source probability, the destination is chosen with a uniform probability. The exact probability distribution for NS-traffic is given in Table 4.2 and for EW-traffic in Table 4.3

During the training, the four scenarios will be simulated alternately and every four episodes they will be repeated. The seed used for the random generation of source and destination of every car in one episode corresponds

Direction	Source	Destination	Probability
Straight	North	South	0.1875
	South	North	0.1875
	East	West	0.1875
	West	East	0.1875
Left or right	North	East	0.03125
	North	West	0.03125
	West	North	0.03125
	West	South	0.03125
	East	South	0.03125
	East	North	0.03125
	South	East	0.03125
	South	West	0.03125

Table 4.1: High-traffic and Low-traffic probability distribution for a single car.

Direction	Source	Destination	Probability
Straight	North	South	0.3375
	South	North	0.3375
	East	West	0.03375
	West	East	0.03375
Left or right	North	East	0.05625
	North	West	0.05625
	South	East	0.05625
	South	West	0.05625
	West	North	0.00625
	West	South	0.00625
	East	South	0.00625
	East	North	0.00625

Table 4.2: NS-traffic probability distribution for a single car.

Direction	Source	Destination	Probability
Straight	East	West	0.3375
	West	East	0.3375
	North	South	0.03375
	South	North	0.03375
Left or right	West	North	0.05625
	West	South	0.05625
	East	South	0.05625
	East	North	0.05625
	North	East	0.00625
	North	West	0.00625
	South	East	0.00625
	South	West	0.00625

Table 4.3: EW-traffic probability distribution for a single car.

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 4.4.

Attribute	Value
Acceleration	1 m/s^2
Deceleration	4.5 m/s^2
Vehicle length	5 meters
Min gap between two vehicles	2.5 meters
Max speed	25 m/s

Table 4.4: Vehicle carachteristics.

Chapter 5

Results

In this chapter, experiments and results will be presented and discussed. In particular, it will be firstly introduced a baseline in order to evaluate the performance of the agent. Then, a comparison of multiple models of the agent will be examined. At the end of the chapter, a comparison among the results obtained will be discussed.

Table 5.1 shows the terminology used in this chapter with the meaning of every term.

Notation	**Meaning**
STL	Static Traffic Light.
ep	Episode.
t	Timestep, the precise moment when the agent interact with the simulation.
nt	Number of timestep in an episode.
nrw	Negative reward relative to one timestep.
wt	Wait time per vehicle. The number of seconds a vehicle has spent in a stationary position in an episode.
veh	A vehicle.
wveh	A vehicle that has been in a stationary position for at least 1 second.

Table 5.1: Notations for the results chapter.

5.1 Evaluation setup and performance metrics used

In order to evaluate the agent performance, a set of fixed experiments has been conducted on every trained agent. For each of the four scenarios, the agent is evaluated on 5 episodes on which the chosen random seed for vehicle generation is not used for the training. The car generation's distribution used for the evaluation will be the Weibull distribution, described in Section 4.5 on page 39. It will be specified later if different distributions are used.

Then, the results of the 5 experiments are averaged and considered as the performance of the agent on that specific scenario. Any graph regarding the performance of the agent in a scenario is generated using data from the episode with the median value of the cumulative negative reward $\sum_{t=0}^{nt} \text{nrw}_t$ across all 5 episodes.

The performance metrics used for the evaluation are the following.

- **Average negative reward.**

$$\text{nrw}_{\text{avg}} = \text{avg}_{ep}\left(\sum_{t=0}^{nt} \text{nrw}_{t,ep}\right) \tag{5.1}$$

  The sum of every negative reward nrw received at every timestep t in an episode ep, averaged among the 5 episodes.

- **Total wait time.**

$$\text{Twt} = \text{avg}_{ep}\left(\sum_{veh} \text{wt}_{veh,ep}\right) \tag{5.2}$$

  The sum of the waiting times wt for every vehicle veh in an episode ep, averaged among the 5 episodes. Measured in seconds.

- **Average wait time / vehicle.**

$$\text{Awt/v} = \text{median}_{ep}\left(\frac{\sum_{\text{wveh}} wt_{veh,ep}}{|\text{wveh}|}\right) \tag{5.3}$$

  The average wait time wt of those vehicles that have waited $wveh$, gathered from the episode with the median value of negative reward. Measured in seconds.

5.2 Static traffic light system

The baseline needed to compare the agent performance is a Static Traffic Light (STL) system.

The STL have the same set of traffic light phases of the agent described in Section 3.4 on page 22 which are reported for readability below, in set 5.4.

$$A_{\text{STL}} = \{\text{NSA}, \text{NSLA}, \text{EWA}, \text{EWLA}\} \tag{5.4}$$

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] and the duration of every traffic light phase is denoted in Table 5.2 according to Koonce and Rodegerdts [22]. Between each phase, a yellow phase is necessary.

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

Table 5.2: Duration of STL phases.

The evaluation of the STL produced the results shown in Table 5.3. This data will be the standard baseline for the agent's performance to compare with.

	Low	**High**	**NS**	**EW**
nrw_{avg}	-7434	-88205	-70271	-68965
Twt	15878	409849	108221	106784
Awt	37.4	111.4	62.2	61.9

Table 5.3: Results of the STL.

The STL have very different results depending on the scenario. In Low-traffic, a fixed cycle is not a good strategy because many vehicles will wait

a significant amount of time and often the light phase will be activated for empty lanes. In High-traffic there are a lot of vehicles coming from all directions, therefore the policy of the STL is arguably the most efficient. In NS-traffic and EW-traffic scenarios, the lanes that need more green time are not prioritized, since there is a fixed cycle. This will make vehicles in longer queues wait for more than necessary, resulting in a mediocre performance.

As an example for future comparison, graphs similar to Figure 5.1 will be used. The blue line represents the cumulative delay in seconds experienced by vehicles over the length of the evaluation episode with the median reward. The figure also shows the car generation's distribution over the episode in background with the red trace, which in this case is the Weibull distribution (for readability purposes, the information that the graph source is the episode with the median reward will be taken for granted from now on). The data in the graph is relative to the Low-traffic scenario.

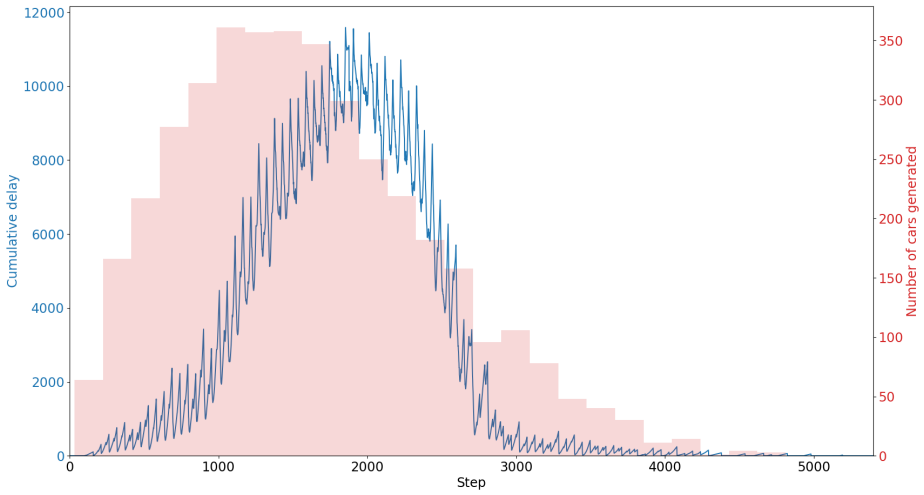


Figure 5.1: Cumulative delay of vehicles with the STL in Low-traffic scenario.

5.3 Performance of the agent

In this section the experiments with hyperparameters variations are described. The principal agent models tested are showed in the following Table 5.4. They are called *Low Gamma Agent* (LGA), *Medium Gamma Agent* (MGA), *High Gamma Agent* (HGA), and *Improved High Gamma Agent* (IHGA).

	LGA	**MGA**	**HGA**	**IHGA**
Neural network structure	5x400	5x400	5x400	5x400
Number of episodes	1600	1600	1600	300
γ	0.09	0.25	0.75	0.75
Reward function	Literature	Literature	Literature	Proposed
Sampling strategy	Light	Light	Light	Intensive

Table 5.4: Agent models tested.

Every agent model has the same neural network structure. There were attempts to discover if a different structure could have better performance and they are listed in Section 5.3.5 on page 59, but none of the configuration tested achieves better results compared to the 5 layers/400 neurons structure [23]. The low, medium and long agents utilize the literature reward function and the light sampling strategy, but experiments show that this configuration is not beneficial to the agent with a high gamma values. Therefore a new model called IHGA is implemented, which includes the *intensive sampling strategy* and *proposed reward function* described in subsection 4.1.1 on page 34 and Section 3.5 on page 25 respectively. For each model of the agent, results compared to the STL are presented. In the last section of this chapter, a performance comparison amongst agent models is discussed.

5.3.1 Low Gamma Agent

The first model of the agent is called Low Gamma Agent (LGA) and has been trained with the following hyperparameters:

- Neural network: 5 layers, 400 neurons each.
- Episodes: 1600
- Gamma value: 0.09
- Reward function: literature
- Sampling strategy: light

The size of the neural network is set at a starting value: variations of the depth and width will be discussed later on this section.

Gamma is set at a low value: this means that the lookahead of the agent is short and it will be more likely to prefer immediate good rewards, rather than waiting for a positive reward that could be acquired after more consequent actions. This agent behavior can be considered greedy.

The training result in terms of reward is showed in Figure 5.2. It represents the cumulative negative reward acquired from the agent from episode 0 to episode 1600 in the Low-traffic scenario, as an example. In order to understand the graph in figure, it is crucial to keep in mind that during the training, the agent follows the ϵ-greedy exploration policy: as the training advances, the agent choose to take explorative actions, i.e. random actions, with a probability that linearly decreases over the number of episodes (see Section 4.3 on page 38). Therefore, it is normal that towards the beginning of the training the agent is having a bad performance since he is taking mostly explorative actions. An indicator of good training is the stability: the reward is decreasing without spikes, which means that the agent has learned a good policy in the Low-traffic scenario. For readability purposes, the training performance of the other scenarios is displayed through the evaluation in table Table 5.5.

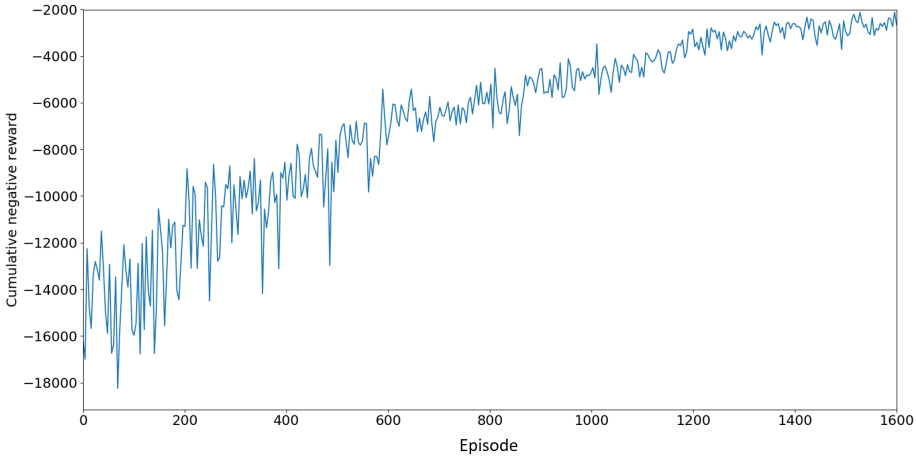


Figure 5.2: Reward of the LGA while training in Low-traffic.

In Table 5.5 are shown the evaluation results on the four scenarios. On the white columns, the results of the LGA are displayed. On the colored columns, the difference in percentage with the STL values of Table 5.2 is highlighted. For example, in the Low-traffic scenario the LGA reduces the total wait time of the STL by 61%. In summary, the LGA achieves good results in 3 of 4 scenarios, with the High-traffic being the only one that the agent does not perform well. This will be a recurrent event in every model of the agent tested, since the policy of the STL is arguably the best policy to follow in the High-traffic scenario and the ability of the agent is not sufficient to achieve good results in this scenario. In particular, the poor performance of the agent is mainly caused by the following situation. The High-traffic scenario causes long queues of more than 200 cars on every lane because the number of generated cars is greater than the intersection capacity. This cause problems to the reward function and consequently to the agent. When the agent sets a green phase where there is a long queue, the starting of cars activates a wave of movement along the length of the queue. This means that every waiting time of every car in the queue resets, but it does not happen instantaneously. For very long queues, the agent receives the reward relative to the last cars in queue very late compared to the timestep when the correspondent phase is activated. This phenomenon leads to confusion from the agent's perspective that receives a reward that does not correspond

just to the latest action. Consequently, the policy that the agent generates is to change phase at every timestep and, in doing so, always activate the yellow phase which is not ideal when the rate of arrivals of vehicles is so high. Hence, the agent shows poor performance in this particular scenario.

	Low	**STL %**	**High**	**STL %**	**NS**	**STL %**	**EW**	**STL %**
nrw_{avg}	-2344	-68	-171409	+94	-8646	-88	-7394	-89
Twt	6119	-61	1084459	+165	28793	-73	29892	-72
Awt/v	15.1	-60	281.8	+153	20.5	-67	20.2	-67

Table 5.5: Results of the LGA (lower is better).

5.3.2 Medium Gamma Agent

Given the results from the previous model of the agent, in this model the gamma is increased. This model of the agent is called Medium Gamma Agent (MGA) and has been trained with the following hyperparameters:

- Neural network: 5 layers, 400 neurons each.
- Episodes: 1600
- Gamma value: 0.25
- Reward function: literature
- Sampling strategy: light

The hyperparameters are the same as the LGA except for gamma, which is set at 0.25. In this way, the agent will try to consider more its prediction of the environment evolution, and therefore choose the current action that maximizes the expected rewards of the next few actions.

Figure 5.3 shows the reward obtained during the training in Low-traffic scenarios. Compared to the training graph of the LGA, the MGA has the same stability if not more, which is a good signal for the evaluation phase.

The scenario of NS-traffic or EW-traffic will have good performance and good training trends across almost every agent model. As an example, Figure 5.4 shows the reward received by the agent during the training for the

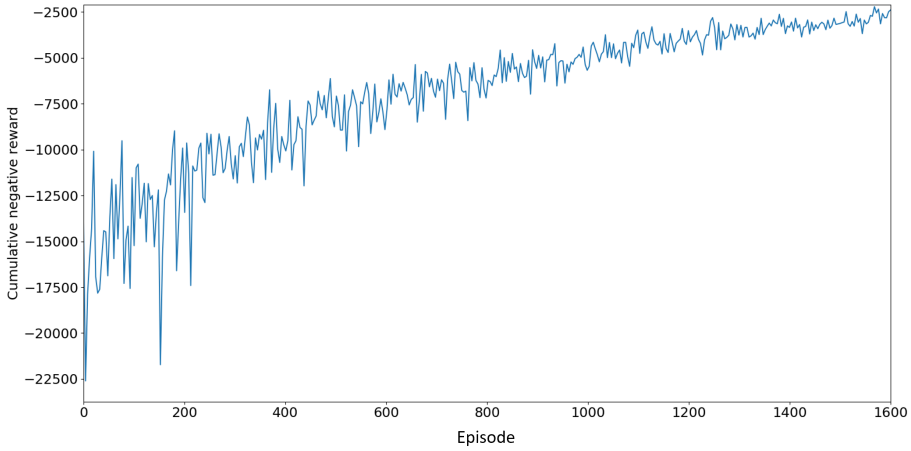


Figure 5.3: Reward of the MGA while training in Low-traffic.

NS-traffic scenario. It is evident that the agent has an excellent training phase and, in the end, it reaches very stable values.

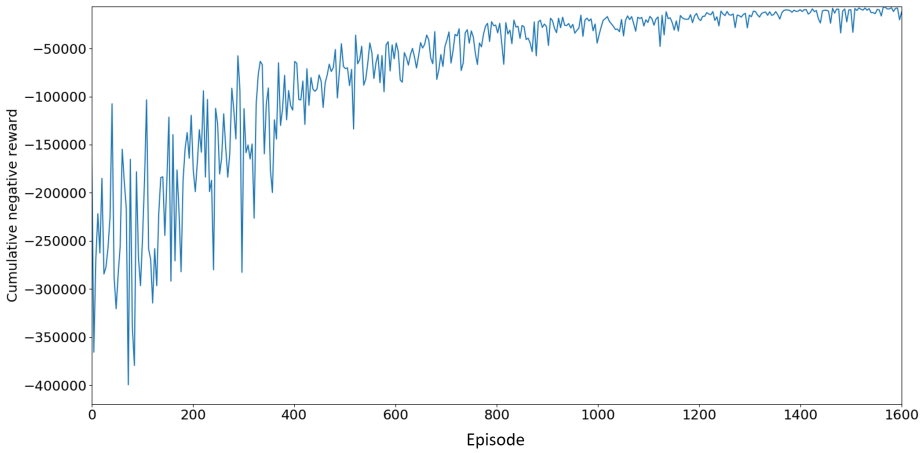


Figure 5.4: Reward of the MGA while training in NS-traffic.

In Table 5.6 are finally shown the evaluation results of the MGA compared to the STL. It can be noted that results are similar to the LGA results. This is

a good signal because it means that increasing the gamma value is beneficial to the agent and it can predict well the consequences of a few actions in the future.

	Low	**STL %**	**High**	**STL %**	**NS**	**STL %**	**EW**	**STL %**
nrw_{avg}	-2259	-70	-157506	+79	-7958	-89	-8408	-88
Twt	6047	-62	1146034	+180	28937	-73	29427	-72
Awt/v	15.8	-58	296.8	+166	20.8	-66	20.3	-67

Table 5.6: Results of the MGA (lower is better).

5.3.3 High Gamma Agent

The model High Gamma Agent (HGA) is trained with the following hyper-parameters:

- Neural network: 5 layers, 400 neurons each.
- Episodes: 1600
- Gamma value: 0.75
- Reward function: literature
- Sampling strategy: light

This time the value of gamma is set at a high value, while the other parameters are still fixed. The high gamma means that the agent is aiming at maximizing the expected cumulative reward of multiple consecutive actions. This can be considered as the true RL agent that has a long lookahead and try to find the policy that overall gives him the best performance with respect to the reward obtained at every step.

The graph in figure Figure 5.5 represents the reward gained during the training in Low-traffic. It is evident that the agent is losing stability, hence performance, towards the end of the training. The end of the training coincide with a very low probability of taking random actions, therefore when

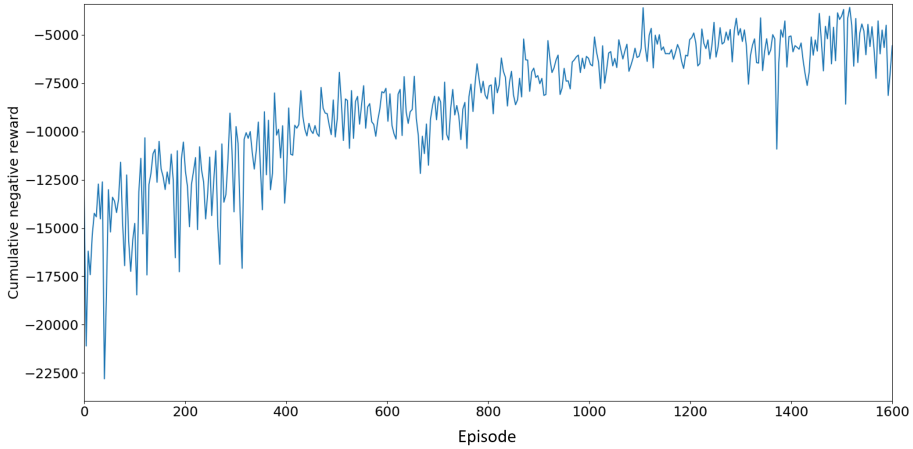


Figure 5.5: Reward of the HGA while training in Low-traffic.

the agent is mostly taking exploitative actions it fails to activate some traffic phase that causes a high waiting time for some vehicles.

Evaluation results are showed in Table 5.7. In Low-traffic, the HGA agent shows poor performance compared to the previous agents tested. In the High-traffic scenario, the bad behavior of the agent is not evident as in the Low-traffic, because the occupancy of lanes is high enough that every phase activated by the agent still release vehicles. In NS-traffic and EW-traffic it can be noted a significant difference of performance between the two scenarios which demonstrate an instability of the agent's training. Overall this agent is not performing as good as the LGA and MGA and the motivation is described below.

	Low	**STL %**	**High**	**STL %**	**NS**	**STL %**	**EW**	**STL %**
nrw_{avg}	-9801	+32	-188046	+113	-12709	-82	-30947	-55
Twt	14081	-11	1112697	+171	31956	-70	68184	-36
Awt/v	35.9	-4	290.1	+160	23.6	-62	39.1	-37

Table 5.7: Results of the HGA (lower is better).

The environment stochasticity

When the γ is set at a high value such as 0.75, the expected reward of several actions in the future influences significantly the value of the action selected at a certain step (see Figure 3.8c on page 29). This approach is in contrast with the nature of the environment: in a deterministic environment, every time the agent takes the action a_t from state s_t, the environment always evolve in the specific state s_{t+1}. However, the simulated dynamics of the vehicular traffic involves a stochastic behavior, since the arrival of the cars to the intersection is not completely predictable. This means that the agent uses the expected value of future actions to compute the value of the current action, although the future states of the system are affected by high stochasticity. Hence, the expected future values of action can be misleading because the agent exploration path can lead to bias and observed trends which in reality do not exist and are only due to the environment stochasticity. This phenomenon can bring to a Q-value instability that prevents the exploitation of a good action policy. The evaluation results for this scenario reflects the instability issue, where it is observed that the agent does not activate the correct light phase, and vehicles wait a long time before the proper traffic light becomes green. A solution to this problem can be to lower the green time duration from 10 seconds to some smaller value, so more actions can be contained in the same temporal window and therefore delay the environment stochasticity. Although this is an efficacy solution, this thesis aims to maintain a certain degree of reality, hence reducing the green time duration in a real-world scenario is impractical because it translates to a higher risk of car accidents. The approach chosen to address the problem of the stochastic environment is described in the next section.

5.3.4 Improved High Gamma Agent

The Improved High Gamma Agent (IHGA) is the agent model that includes the modifications of the *intensive sampling strategy* and the *proposed reward function*, described in subsection 4.1.1 on page 34 and Section 3.5 on page 25 respectively. Also, the following hyperparameters are used:

- Neural network: 5 layers, 400 neurons each.
- Episodes: 300

- Gamma value: 0.75
- Reward function: proposed
- Sampling strategy: intensive

This model is implemented and tested in order to improve the performance of the agent with a high value of gamma during the training phase. The combined choice of the new reward function and sampling strategy are beneficial in order to solve the Q-values instability issues. The *proposed reward function* boost the punishment of bad action choices, so it decreases the chance of situations where the best future action was misleading. The increase of training instances in an episode that is the *intensive sampling strategy* essentially increment the dataset where the agent trains, leading to more stable and gradual learning of the action values. As said before, this approach starts a training instance at every step of the simulation with 100 samples gathered per instance, therefore 1600 episodes are not necessary anymore and they are reduced to 300. For instance, in the previous models, approximately 2.5 million samples were gathered for the training across 1600 episodes. With the intensive sampling strategy, the total number of samples gathered across 300 episodes is approximately 64 million.

The cumulative reward obtained during the training of the IHGA is shown in Figure 5.6. As the graph shows there is clear stability especially towards the end of the training, which is an excellent signal of a good action choice policy.

The results of this agent model are shown in Table 5.8. Recalling that the γ value is set at 0.75, this is a very good result. While in the High-traffic scenario is not performing well, in Low-traffic the IHGA achieves very low values compared to the STL, increasing the overall efficiency of more than 50%. In EW-traffic and NS-traffic very positive results are registered and the equivalence of performance between the two opposite scenarios is a signal that the agent hasn't any bias towards a particular situation.

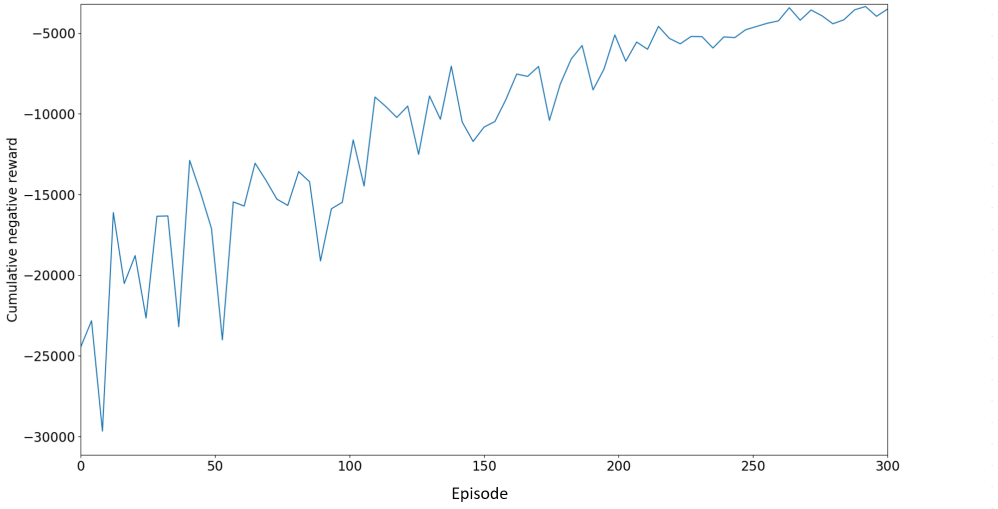


Figure 5.6: Reward of the IHGA while training in Low-traffic.

	Low	**STL %**	**High**	**STL %**	**NS**	**STL %**	**EW**	**STL %**
nrw_{avg}	-3228	-57	-185496	+110	-16905	-76	-18929	-73
Twt	7150	-55	1074011	+162	42413	-61	43718	-59
Awt/v	18.2	-51	284.7	+155	28.4	-54	28.2	-54

Table 5.8: Results of the IHGA (lower is better).

In Figure 5.7 is shown the cumulative delay of vehicles in the Low-traffic scenario with the blue line representing the IHGA and the green line representing the STL. The graph reflects the performance increase described in the evaluation table, with the IHGA able to achieve better performance where the STL registers delay peaks. On the contrary, in Figure 5.8 is shown the same measure but in High-traffic scenario, where every agent does not perform well. In particular, is evident that the timings of the STL that implies a low rate of yellow phases are the right choice in this scenario, and the agents do not succeed on replicating this behavior (see Section 5.3.1 on page 48). Even in a situation with a different vehicles generation pattern like in Figure 5.9, which is a triangular distribution, the IHGA outperform the STL in the Low-traffic scenario. The delay data in every graph is processed using a mobile mean with a window of 50 values in order to enhance the readability.

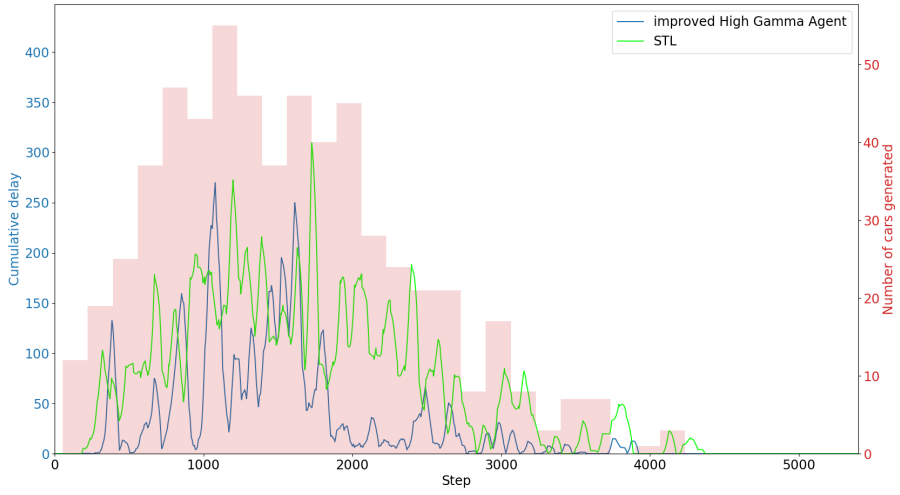


Figure 5.7: Cumulative delay of vehicles with IHGA in Low-traffic scenario.

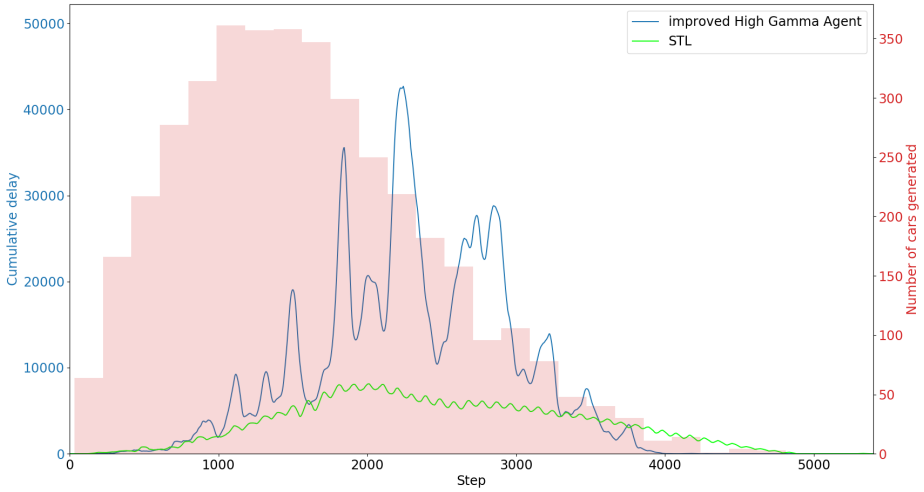


Figure 5.8: Cumulative delay of vehicles with IHGA in Low-traffic scenario.

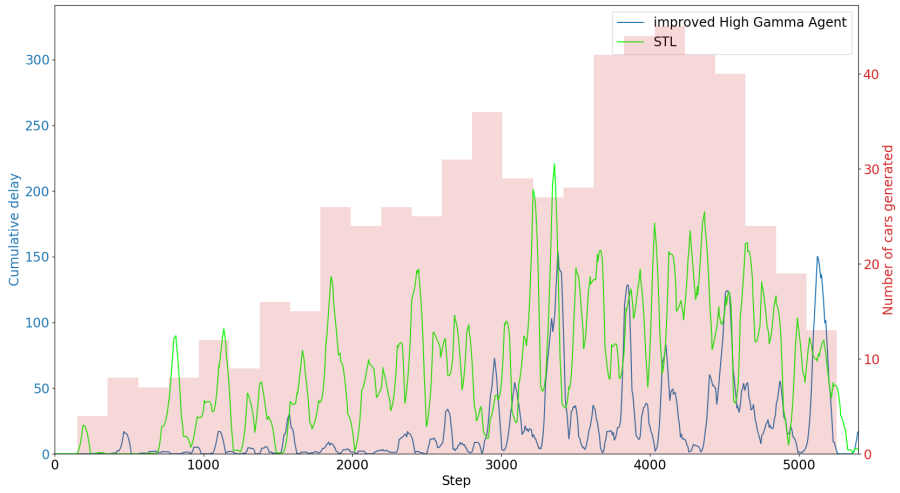


Figure 5.9: Cumulative delay of vehicles with IHGA in Low-traffic scenario, with the vehicles generation according to a triangular distribution.

5.3.5 Neural network variations

In order to exclude the neural network size as a factor that denies the agent a good performance, two models with different network dimensions have been tested.

Smaller network

The first model involved the following parameters:

- Neural network: 3 layers, 400 neurons each.
- Episodes: 1600
- Gamma value: 0.09
- Reward function: literature
- Sampling strategy: light

This version is similar to the LGA but with 3 layers instead of 5. After the training completed, the agent's policy was evaluated and the results are shown in Table 5.9. The results are compared to the most similar model which is in fact the LGA, since the purpose of this model is to understand if different sizes of the network achieve better performance. From the evaluation results, it is clear that a smaller network is not capable to map states to action values like the models presented before. For example, in the Low-traffic scenario this agent increases the total waiting time of 58% and the average wait per vehicle of 41% compared to the LGA. therefore, a reduction of the neural network will be excluded from future experiments with this agent design.

	Low	**LGA %**	**High**	**LGA %**	**NS**	**LGA %**	**EW**	**LGA %**
nrw_{avg}	-4921	+110	-176025	+3	-11470	+33	-9872	+34
Twt	9676	+58	1147111	+6	33603	+17	32262	+8
Awt/v	21.4	+41	296.0	+5	22.5	+10	21.2	+5

Table 5.9: Results of the smaller-network agent (lower is better).

Bigger network

The model with a bigger network has the following characteristics:

- Neural network: 9 layers, 1000 neurons each.
- Episodes: 1600
- Gamma value: 0.25
- Reward function: literature
- Sampling strategy: light

This agent model is similar to the MGA, which also have the gamma value set at 0.25. Here the network is increased significantly. The goal is to understand if the network of the MGA is enough to represent a good mapping between states and action values, or a bigger network achieves better performance. Evaluation results of this agent are shown in table Table 5.10. As can be seen in the comparison with the MGA, the increase of the network size is not beneficial to the performance. The agent has difficulty to well approximate the mapping state-action and perhaps more training episodes are required with this configuration. However, the time that the agent needs to train on 1600 episodes with this configuration is yet significantly expensive, therefore this configuration has not been further examined.

	Low	**MGA %**	**High**	**MGA %**	**NS**	**MGA %**	**EW**	**MGA %**
nrw_{avg}	-10766	+376	-271858	+73	-20782	+161	-35935	+327
Twt	16050	+165	1401887	+22	50308	+74	83169	+183
Awt/v	38.4	+143	360.0	+21	29.7	+43	48.5	+139

Table 5.10: Results of the bigger-network agent (lower is better).

5.4 Agents performance overview

In this chapter, multiple agents with different parameters and behavior were described and compared to a baseline scenario represented by a static traffic light (STL) with fixed light cycle. In this section, a performance summary is

presented where the results of every agent are compared. The performance overview is showed in Table 5.11 and the results are grouped by scenario. On every group, the performances showed are the percentage variations of the results of every agent with respect to the STL. The analysis is based on the same results showed in the evaluation tables showed in the previous section.

	LGA	**MGA**	**HGA**	**IHGA**
	Low-traffic scenario			
nrw_{avg}	-68	-70	+32	-57
Twt	-61	-62	-11	-55
Awt/v	-60	-58	-4	-51
	High-traffic scenario			
nrw_{avg}	+94	+79	+113	+110
Twt	+165	+180	+171	+162
Awt/v	+153	+166	+160	+155
	NS-traffic scenario			
nrw_{avg}	-88	-89	-82	-76
Twt	-73	-73	-70	-61
Awt/v	-67	-66	-62	-54
	EW-traffic scenario			
nrw_{avg}	-89	-88	-55	-73
Twt	-72	-72	-36	-59
Awt/v	-67	-67	-37	-54

Table 5.11: Agents performance overview, percentage variations compared to STL (lower is better).

As already discussed, the increase of the value of γ from 0.25 to 0.75 highlights a significant difference in performance between the Medium Gamma Agent and the High Gamma Agent, especially in the Low-traffic scenario where the action choice policy can be clearly evaluated. Regarding the HGA, a performance difference between the NS-traffic scenario and the EW-traffic scenario can be exploited and it is caused by an instability of the train-

ing phase. Regarding the Low Gamma Agent, it is interesting to note that it does not show any meaningful performance difference compared to the MGA, therefore the MGA is preferable as a traffic light control system since it can predict a few more actions in the future without compromising the intersection efficiency. A brief comparison among the HGA and the IHGA demonstrates that the proposed reward function and the intensive sampling strategy significantly improved the performance. In particular, in the Low-traffic scenario, the efficiency increases almost to the level of LGA and MGA, while in EW-traffic and NS-traffic the IHGA have found a balance of performance that underlines the efficacy of the modifications proposed. There is still a small performance gap to fill between the IHGA and the lower gamma agents, therefore more variations and improvement about the design of the agent can be definitely tested in the future.

In summary, the application of reinforcement learning techniques in the field of traffic signal control is a challenging task, but with very high potential. Multiple versions of the agent are required to be tested in order to find the appropriate state representation, action set, reward function and learning techniques that enable the agent to achieve the best performance in every traffic scenario. In this thesis, the elements of the agent are defined in such a way that the design is consistent with a real-world implementation. On the contrary, in related works, highly sophisticated and often impractical reinforcement learning elements are used [10] [11], leading to notable results - but unattainable in real-world appliances due to the difficulty of reproducing the chosen model. Therefore, improvements and modifications will be tested in the future in order to maximize the results while maintaining realism to a significant extent.

Chapter 6

Conclusions and future works

In this thesis, a deep Q-learning agent was implemented in the context of signal traffic control in order to investigate the efficiency improvement while maintaining a significant degree of reality. The reinforcement learning agent was designed with a state representation that identifies the position of vehicles in the environment, an action set defined by traffic light configurations with a fixed duration, and two reward functions that capture with different magnitudes the difference of vehicles waiting times between actions. In particular, the elements of the agent are designed to be plausible to implement with respect to a possible real-world appliance. The learning approach applied for the agent's training is the Q-learning equation combined with a deep neural network. The Q-learning is used for the update of the action values as the experience of the agent increases and the neural network is employed for the Q-values prediction and, therefore, the approximation of the state-action function. A traffic micro-simulator was used to replicate a 4-way intersection with multiple lanes, and to reproduce various traffic scenarios with different traffic distributions. The reward was calculated based on the simulated waiting time of vehicles, making the agent aware of the consequence of his actions in different situations. Results indicate that the proposed agent can adapt to several traffic situations and is able to outperform the static traffic light system in situations of low and medium densities. However, in the high density scenarios the tested agents still provide higher delays compared to the static traffic light. This is due to the phenomenon that arises with long queues where the agent is misleading by the high delay of the received rewards. This issue could be solved by a new reward function that takes advantage of a different efficiency measure of the intersection.

In addition to the study of alternative reward functions, this framework is suited to additional future works and investigations: the state representation could include more information such as the relative velocity and the current traffic phase to enhance the agent's perception, but with the risk of over-increasing the state space and worsening the agent training effectiveness; the agent actions could also control the duration of the traffic phases, such as the green times and yellow times, to gain more flexibility and possibly improve the efficiency on a wider range of situations.

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.

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.

Bibliography

[1] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016 (cit. on pp. 5, 6).

[2] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge, 1998 (cit. on p. 8).

[3] Oriol Vinyals et al. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`. 2019 (cit. on p. 8).

[4] Dmitry Kalashnikov et al. "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation". In: *arXiv preprint arXiv: 1806.10293* (2018) (cit. on p. 8).

[5] Kok-Lim Alvin Yau et al. "A survey on reinforcement learning models and algorithms for traffic signal control". In: *ACM Computing Surveys (CSUR)* 50.3 (2017), p. 34 (cit. on p. 9).

[6] Wade Genders and Saiedeh Razavi. "Evaluating reinforcement learning state representations for adaptive traffic signal control". In: *Procedia computer science* 130 (2018), pp. 26–33 (cit. on p. 10).

[7] 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 (cit. on pp. 10, 11, 13).

[8] Juntao Gao et al. "Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network". In: *arXiv preprint arXiv:1705.02755* (2017) (cit. on pp. 10–13).

[9] Wade Genders and Saiedeh Razavi. "Using a deep reinforcement learning agent for traffic signal control". In: *arXiv preprint arXiv:1611.01142* (2016) (cit. on pp. 10–13, 20, 26).

[10] Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. "Traffic light control using deep policy-gradient and value-function-based reinforcement learning". In: *IET Intelligent Transport Systems* 11.7 (2017), pp. 417–423 (cit. on pp. 10–13, 62).

[11] 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*. ACM. 2018, pp. 2496–2505 (cit. on pp. 10–13, 62).

[12] L-J LIN. "Reinforcement Learning for Robots Using Neural Networks". In: *Ph.D. thesis, Carnegie Mellon University* (1993) (cit. on pp. 13, 33, 35).

[13] Patrick Mannion, Jim Duggan, and Enda Howley. "An experimental review of reinforcement learning algorithms for adaptive traffic signal control". In: *Autonomic Road Transport Support Systems*. Springer, 2016, pp. 47–66 (cit. on p. 14).

[14] Richard Dowling. *Traffic analysis toolbox volume vi: Definition, interpretation, and calculation of traffic analysis tools measures of effectiveness*. Tech. rep. 2007 (cit. on p. 25).

[15] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292 (cit. on p. 27).

[16] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". PhD thesis. King's College, Cambridge, 1989 (cit. on p. 27).

[17] John N Tsitsiklis and Benjamin Van Roy. "Analysis of temporal- difference learning with function approximation". In: *Advances in neural information processing systems*. 1997, pp. 1075–1081 (cit. on p. 30).

[18] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), p. 529 (cit. on p. 35).

[19] Daniel Krajzewicz et al. "SUMO (Simulation of Urban MObility)-an open-source traffic simulation". In: *Proceedings of the 4th middle East Symposium on Simulation and Modelling (MESM20002)*. 2002, pp. 183–187 (cit. on p. 38).

[21] Lee A Rodegerdts et al. *Signalized intersections: informational guide*. Tech. rep. 2004 (cit. on p. 39).

[22] Peter Koonce and Lee Rodegerdts. *Traffic signal timing manual*. Tech. rep. United States. Federal Highway Administration, 2008 (cit. on p. 45).

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016 (cit. on p. 47).

Sitography

[20] *SUMO - Simulation of Urban MObility*. 2001. URL: `https://www.dlr.de/ts/en/desktopdefault.aspx/tabid-9883/16931_read-41000/` (visited on 03/03/2019) (cit. on p. 38).