Faculty of Engineering

Computer Department

Smart Cities Traffic Management System

By:

Ahmed Saeed Mohamed Sharf

Al-Amir Bahrawy Khalifa Hamada

Hussien Mohamed Salah Mohamed Ismail

Mariam Mohamed Ibrahim Abdel-Halim

Mohammed Emadeldin Mahmoud Radwan

Supervised by:

Assoc. Prof. Amr El Sayed

Acknowledgment

We would like to express our full grateful and own a deep sense of gratitude to our Graduation Project Supervisor **Assoc. Prof. Amr El Sayed** for the advice, knowledge, insightful discussions, and suggestions throughout the entire journey, and patiently guiding us through his processes. You made us believe that we have what it takes and the ability to reach even greater heights in our graduation project.

Abstract

Modern society depends on road transport for the movement of people and goods. Ensure that vehicles can move efficiently on the road infrastructure Useful for everyone where the number of vehicles is constantly increasing faster than the traffic available to support them, congestion is a difficult problem to deal with and is made worse by car accidents. However, the population growth and increase the number of vehicles on the streets often exceeds the current Infrastructure capacity, resulting in more congestion and increased travel times and more emissions.

This problem affects many aspects of modern society, including economic development, traffic accidents, increased greenhouse emissions, time spent, and health damage. In this context, modern societies can rely on the traffic management system to reduce traffic congestion and its negative effects.

Table of Contents

Table of Figures

Chapter 1: Introduction

1.1. Problem statement

Individuals suffer from daily traffic congestion. This results in wasting time and effort. This has a negative effect on users' lifestyle [1]. On a bigger scale it affects economy and surrounding environment. Countries is affected by losing billions of dollars due to traffic. Egypt loses $8 billion dollars due to traffic. In 2018, the United States lost $87 billion dollars and Japan lose 12 trillion yen due to traffic congestion [1,2,3]. These numbers are expected to increase in the future with increase of the number of people that lives in urban areas. Traffic congestion leads to increase the emission of greenhouse gases such as carbon dioxide. For example, transportation produced 28% of the total greenhouse emission in the United States. This helps increasing global warming and affects the health of urban areas residents.

For the previously explained reasons, traffic congestion has become a problem that needs urgent solution. We solved it by minimizing the waiting time, hence minimizing the total travel time on the road network. This can be done by efficiently managing traffic at the intersections using traffic signal. Most traffic signal that is implemented are fixed which is not an efficient way because traffic demand is not static. It is changing with the time. So, we need to make traffic signals adaptive to the traffic demand.

To achieve that, currently the most efficient approach is the use of deep reinforcement learning specifically DQN algorithms to determine the traffic phase based on the current traffic demand and the green light time of the phase. To further increase the efficiency of the system considering the future traffic demand. The related work and the design of our approach in solving the traffic congestion problem at the intersections and how we managed to incorporate the use of both traffic prediction with DQN is shown below. Despite the massive efforts by the Egyptian government to tackle traffic congestion and environmental deterioration, by introducing a metro system and a comprehensive bus network, traffic congestion remains a serious problem in the GCMA with substantial adverse effects on personal travel time, vehicle operating costs, air quality, public health, business environment and business operations. The causes of traffic congestion are complex, as are the range of possible policies and investments that could be arrayed to address the problem.

1.2. Related Work

There are many aspects to traffic management that includes routing of vehicle, use of traffic signal and the reaction to emergencies. Traffic signal control have many approaches, but most cities right now use fixed time traffic signal which inefficient with the increase of traffic flow and does not adapt to the real-time demand of traffic. In Order to deal with this problem, studies are actively Inducted in the field of smart traffic signal control which intel the use of current traffic information to determine the phase and time [4-7]. A Smart Traffic Management System (STMS) using Internet of Things (IoT) was introduced in [4] where they increase the green light time when high density is detected. Real-time traffic density information is collected using internet of things and image processing. But more researchers are drawn to the use of artificial intelligence [5-11]. For example, the use of fuzzy logic-based algorithms has shown shows performance improvements [5-7] over approaches like found in [4].

Other approaches include the use of Reinforced learning has shown great performance improvements over traditional approaches in both single-intersection and multi-intersection environments [8-10] using the current traffic state to determine the next action. To further improve the performance in traffic control Deep reinforced learning algorithms like Deep Q-Network (DQN) are utilized which solves several problems including the "curse of dimensionality" [11-13]. The DQN extensions are also utilized both separately and combined to further stabilize the learning process [14-17].

More recently the combine use of traffic prediction specially using deep learning algorithms in solving traffic signal control yielded better results as they consider the randomness and dynamics of the traffic characteristics. Algorithms like the CNN, RNN and RNN variants were used to help optimizing the DQN's Q-function by predicting characteristics which including weather and time to represent the real-world variables that cannot be simulated. Although these algorithms can capture either the spatial dependency or the temporal features of the network [18,19], they cannot capture the characteristic about the structure of the network unlike the GNN algorithms which yielded better results as the network is easily represented as a graph [20-26].

Various graph representations were used to define the road network including defining the nodes as the vehicles [27], defining the nodes as

the road segments [28-30], defining the installed devices on the road network as the nodes [31-33] and defining the intersections as the node [34]. Another approach to combing traffic prediction with the DQN algorithm in solving the traffic signal control is to use it to calculate the expected green-light time based on the future traffic demand and compare it to the output of the DQN algorithm which is the green time light required by the current traffic demand taking the mini-mum of both as the time for the phase selected by the DQN algorithm as in [34].

1.3. Motivation

Our Motivation is to help increase the quality of life for all people by reduce the bad effects of traffic congestions which include time delay, fuel consumption and the increase of travel times. Another motivation is to help the governments reduce the economic losses that can be better invested in more important aspects like health and education. Also helping to save the environment by reducing the amounts of greenhouse gases emissions that results from traffic thus helping in the prevention of global warming and also that serves our first motivation by reducing the air pollution and having a healthier environment for the people.

1.4. Aim and Objectives

1.4.1. Aim

Our aim is to enhance the performance of traffic signal to reduce the waiting time at the intersections and overall travel time for the road network in constraints of a smart city

1.4.2. Objectives

- Being able to predict Future traffic features to calculate greenlight duration based on future traffic demand.
- Being able to calculate the greenlight time based on current demand of traffic.
- Being able to efficiently choose the phase of traffic based on the current traffic demand.
- Being able to efficiently choose the appropriate greenlight duration of the previously chosen phase and based on previously calculated greenlight durations.

Chapter 2: Necessary background

2.1. Machine Learning

Machine learning is the study of computer algorithms that improve automatically through experience and using data. It is seen as part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as "training data," in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a variety of applications, where it is difficult or unfeasible to develop traditional algorithms to perform the required tasks. A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers; But not all machine learning is statistical learning. The study of Mathematical Optimization provides methods, theory, and application areas in the field of machine learning. Data mining is a related field of study, with an emphasis on exploratory data analysis through unsupervised learning, and machine learning in its application across business problems, is also referred to as predictive analytics.

2.1.1.1. Supervised Learning

Supervised learning algorithms construct a mathematical model of a set of data that include both the inputs and the desired outputs the data is known as sampled data and consists of a set of training examples. Each training example has one or many inputs and a desired output, also known as a supervisory signal. In the mathematical model, each training example is displayed by a vector or array, at sometimes it is called a feature vector, and the training data is displayed by a matrix. Through reduplicate optimization of an objective function, supervised learning algorithms learn a function that can be used to guess the output associated with new inputs. An optimal task will allow the algorithm to correctly determine the output for inputs that were not a piece of the training data. An algorithm that enhances the accuracy of its outputs or predictions overtime is said to have learned to implement that task.

Similarity learning is a wide range of supervised machine learning relative to regression and classification, but the objective is to learn from examples using a common function that measures how similar or related two objects are. It has applications in recommendation systems, visual identity tracking, and face verification and ranking.

2.1.1.2. Unsupervised Learning

Supervised learning refers to using a set of input variables to predict the value of a labeled output variable. It requires labeled data (think of this like an answer key that the model can use to evaluate its performance). Conversely, unsupervised learning refers to inferring underlying patterns from an unlabeled dataset without any reference to labeled outcomes or predictions. (Figure 2.1)

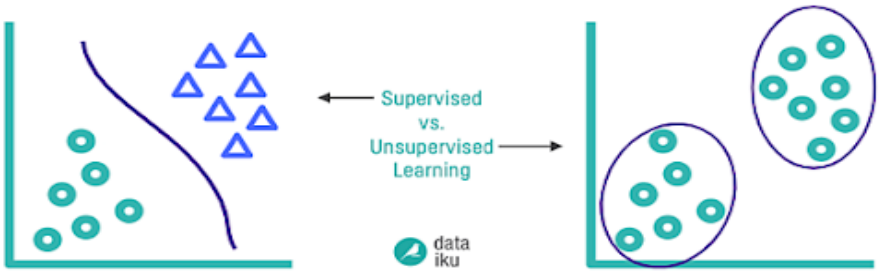

Figure 2.1: Comparison between supervised and unsupervised learning

There are several methods of unsupervised learning, but clustering is far and away the most used unsupervised learning technique. Clustering refers to the process of automatically grouping together data points with similar characteristics and assigning them to "clusters."

2.1.1.3. Reinforcement Learning

Reinforcement learning is the training of machine learning models to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment. In reinforcement learning, an artificial intelligence faces a game-like situation. The computer employs trial and error to come up with a solution to the problem. To get the machine to do what the programmer wants, the artificial intelligence gets either rewards or

penalties for the actions it performs. Its goal is to maximize the total reward.

Although the designer sets the reward policy–that is, the rules of the game–he gives the model no hints or suggestions for how to solve the game. It's up to the model to figure out how to perform the task to maximize the reward, starting from totally random trials and finishing with sophisticated tactics and superhuman skills. By leveraging the power of search and many trials, reinforcement learning is currently the most effective way to hint machine's creativity. In contrast to human beings, artificial intelligence can gather experience from thousands of parallel gameplays if a reinforcement learning algorithm is run on a sufficiently powerful computer infrastructure.

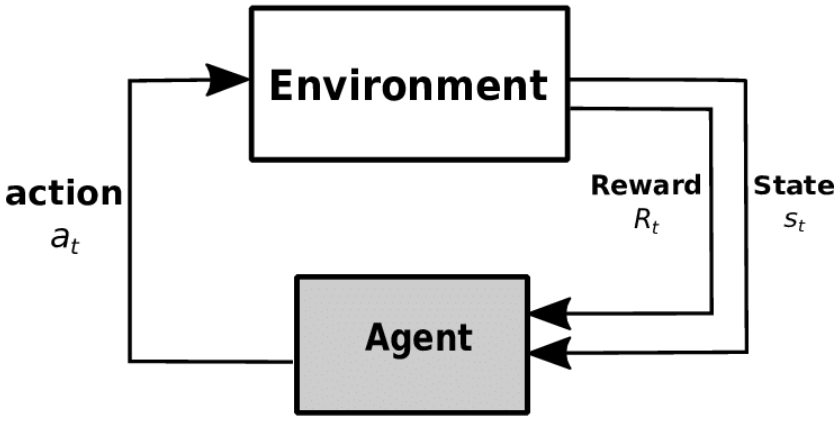

Figure 2.2: Reinforcement Learning

Distinguishing between reinforcement learning, deep learning, and machine learning is hard. In fact, there should be no clear divide between machine learning, deep learning, and reinforcement learning. It is like a parallelogram – rectangle – square relation, where machine learning is the broadest category and the deep reinforcement learning the narrowest one.
In the same way, reinforcement learning is a specialized application of machine and deep learning techniques, designed to solve problems in a particular way.

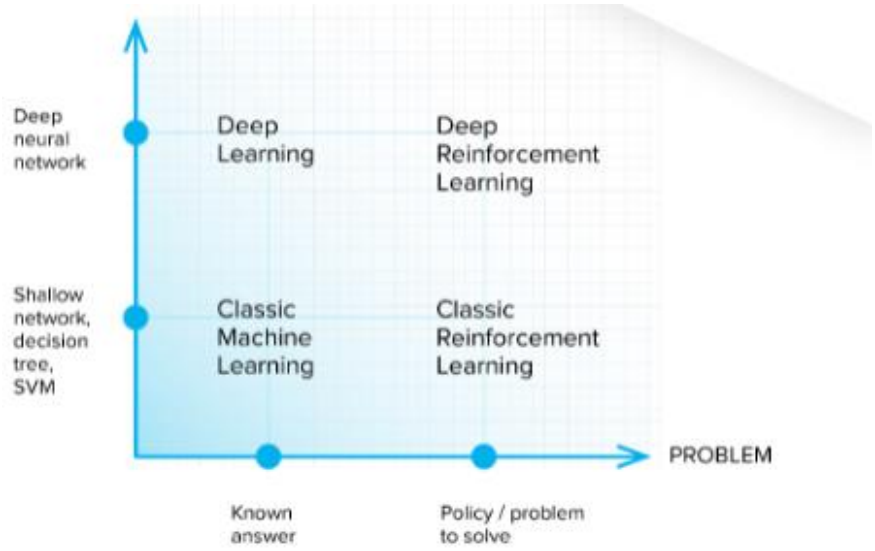

Figure 2.3: Comparison between different learning types

2.1.1. Q-Learning

Q-Learning is a form of model-free reinforcement learning. It consists of assigning a value, called the *Q-value*, to an action taken from a precise state of the environment. Formally, in literature, a Q-value is defined as in equation (2.1).

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

Equation 2.1

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 em- phasize 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 fu- ture'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 (5.1) is used and it is presented in equation (2.2)

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

Equation 2.2

Where the reward r_{t+1} is the reward received after taking action a_t in state s_t. The term $Q^{J}(s_{t+1}, a_{t+1})$ 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 (2.1), the discount factor γ denote a small penalization of the future reward compared to the immediate reward. The above equation (2.2) is a rule that update the Q-value of the cur- rent action a_t taken in state s_t with the immediate reward and the discounted Q-value of future actions. Therefore, the term $Q^{J}(s_{t+1}, a_{t+1})$ that represents the value of future actions implicity holds the maximum discounted reward of the state after s_{t+1}, i.e. $Q^{JJ}(s_{t+2}, a_{t+2})$ and likewise, it holds the maximum discounted reward for the next state which is $Q^{JJJ}(s_{t+3}, a_{t+3})$ and so on. This is how the agent can choose the action a_t based on not just the immedi- ate reward, but also based on the expected future discounted rewards. For simplicity, the rule can be unpacked and the result is the equation (2.3)

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

Equation 2.3

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

2.2. Deep Learning

Deep learning is an artificial intelligence (AI) function that simulates the work of the human brain in processing data and creating patterns for use in decision-making. Deep learning is a subset of machine learning in artificial intelligence that has networks capable of unsupervised learning from unstructured or disaggregated data. It is also known as deep neural learning or deep neural network.

Also, it uses the hierarchical level of artificial neural networks to implement the machine learning process. Artificial neural networks are built like the human brain, with neural nodes connected to each other like the Internet. While traditional software adopts analysis using data in a linear manner, the hierarchical function of deep learning systems enables machines to process data with a non-linear approach.

Deep learning can automatically extract features and achieve higher accuracy than traditional AI technologies. As a result, deep learning can be applied to a wide variety of scenarios. Additionally, open-source development tools like TensorFlow and Caffe are accelerating

advances in deep learning. Research on the suitability of deep learning in resource-limited mobile or embedded platforms will undoubtedly push a big step forward towards pervasive deep learning.

2.3. Neural Network

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial neural networks (ANNs) are comprised of a node layer, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

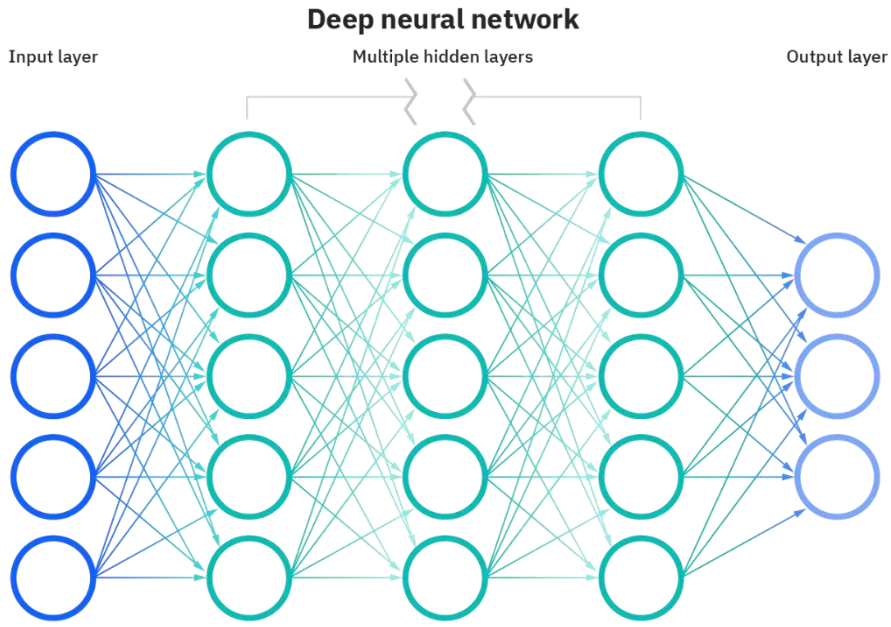

Figure 2.4: Deep Neural Network Representation

Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take

minutes versus hours when compared to the manual identification by human expert.

A deep neural network is an artificial neural network with many layers between the first (input) and last (output) layers. The DNN finds the right mathematical handling to turn the input into the output, so it can be non-linear relationship or a linear relationship. The network moves between the layers calculating the probability of every output. For example, a deep neural network that is drilled to recognize dog breeds will go over the given image and calculate the probability that the dog in the image is a certain breed. The actor can check the results and select which probabilities that the network should display (above a certain threshold, etc.) and return the chosen label. Each mathematical manipulation as such is considered a layer, and complex DNN have multiple layers. DNN can pattern complex non-linear relationships.

DNN architectures produce compositional models where the object is expressed as layered composition of primitives. The extra layers enable composition of features from the further down layers, potentially modeling complex data with fewer units than a similarly performing shallow network. Deep architectures contain many different of a little basic approach. Every architecture has found success in specific fields.

It is not always possible to compare the performance of much architecture, except if they have been evaluated on the same data sets. DNNs are typically feedforward networks in which data flows from the first (input) layer to the last (output) layer without returning. At first, the DNN creates a map of virtual neurons and assigns random weights, or numerical values, to connections between them. The weights and inputs are multiplied and return an output between zero and one. If the network didn't carefully recognize a particular pattern, an algorithm would correct the weights. That way the algorithm can make many parameters more influential, until it determines the right mathematical manipulation to fully process the data.

2.4. Types of Neural Networks

Neural networks can be classified into different types, which are used for different purposes. While this isn't a comprehensive list of types, the below would be representative of the most common types of neural networks that you'll come across for its common use cases:

The perceptron is the oldest neural network, created by Frank Rosenblatt in 1958. It has a single neuron and is the simplest form of a neural network:

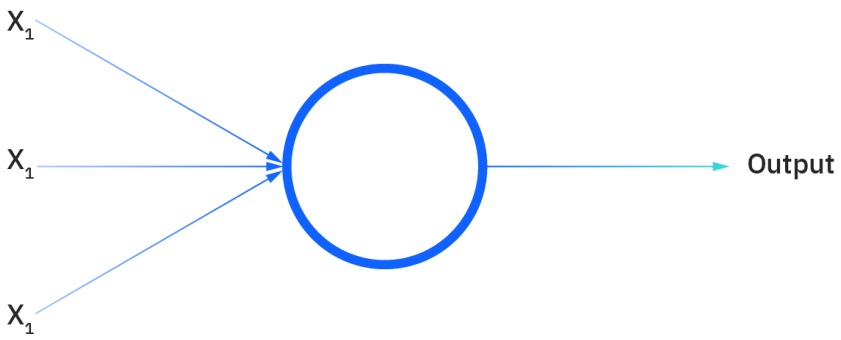

Figure 2.5

Feedforward neural networks, or multi-layer perceptrons (MLPs), are what we've primarily been focusing on within this article. They are comprised of an input layer, a hidden layer or layers, and an output layer. While these neural networks are also commonly referred to as MLPs, it's important to note that they are comprised of sigmoid neurons, not perceptrons, as most real-world problems are nonlinear. Data usually is fed into these models to train them, and they are the foundation for computer vision, natural language processing, and other neural networks.

Convolutional neural networks (CNNs) are similar to feedforward networks, but they're usually utilized for image recognition, pattern recognition, and/or computer vision. These networks harness principles from linear algebra, particularly matrix multiplication, to identify patterns within an image.

Recurrent neural networks (RNNs) are identified by their feedback loops. These learning algorithms are primarily leveraged when using time-series data to make predictions about future outcomes, such as stock market predictions or sales forecasting.

2.4.1. Graph Neural Network

Graphs are a kind of data structure which models a set of objects (nodes) and their relationships (edges). Recently, research on analyzing graphs with machine learning have been receiving more and more attention because of the great expressive power of graphs, i.e. graphs can be used as denotation of a large number of systems across

various areas including social science (social networks, natural science and protein-protein interaction networks), knowledge graphs and many other research areas. As a unique non-Euclidean data structure for machine learning, graph analysis focuses on tasks such as node classification, link prediction, and clustering. Graph neural networks (GNNs) are deep learning-based methods that operate on graph domain. Due to its convincing performance, GNN has become a widely applied graph analysis method recently.

Graph Neural Network is a type of Neural Network which directly operates on the Graph structure. It provides a convenient way for node level, edge level, and graph level prediction task. There are mainly three types of graph neural networks in the literature:

1. Recurrent Graph Neural Network
2. Spatial Convolutional Network
3. Spectral Convolutional Network

The intuition of GNN is that nodes are naturally defined by their neighbors and connections. To understand this we can simply imagine that if we remove the neighbors and connections around a node, then the node will lose all its information. Therefore, the neighbors of a node and connections to neighbors define the concept of the node.

Having this in mind, we then give every node a state (x) to represent its concept. We can use the node state (x) to produce an output (o), i.e., decision about the concept. The final state (x_n) of the node is normally called "node embedding". The task of all GNN is to determine the "node embedding" of each node, by looking at the information on its neighboring nodes.

2.4.2. Recurrent Graph Neural Network

RecGNN is built with an assumption of Banach Fixed-Point Theorem. Banach Fixed-Point Theorem states that: Let (X, d) be a complete metric space and let ($T:X \rightarrow X$) be a contraction mapping. Then T has a unique fixed point ($x*$) and for any $x \in X$ the sequence $T_n(x)$ for $n \rightarrow \infty$ converges to ($x*$). This means if I apply the mapping T on x for k times, x^k should be almost equal to $x^{(k-1)}$, i.e.:

$$x^k = T\left(x^{k-1}\right), \mathrm{k} \in (1, n)$$

Equation 2.4

RecGNN defines a parameterized function f_w:

$$\boldsymbol{x}_n = f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]})$$

Equation 2.3

where l_n, l_{co}, x_{ne}, l_{ne} represents the features of the current node [n], the edges of the node [n], the state of the neighboring nodes, and the features of the neighboring nodes.

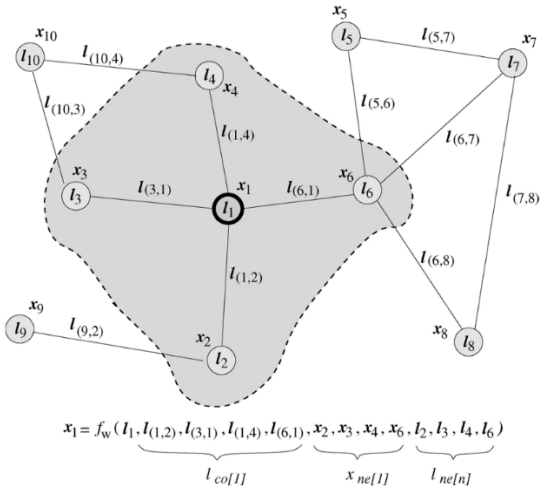

Figure 2.6: An illustration of node state update based on the information in its neighbors.

Finally, after k iterations, the final node state is used to produce an output to decide about each node. The output function is defined as:

$$\boldsymbol{o}_n = g_{\boldsymbol{w}}(\boldsymbol{x}_n, \boldsymbol{l}_n)$$

Equation 2.4

2.4.3. Spatial Convolutional Network

The intuition of Spatial Convolution Network is similar to that of the famous CNN which dominates the literature of image classification and segmentation tasks. In short, the idea of convolution on an image is to sum the neighboring pixels around a center pixel, specified by a filter with parameterized size and learnable weight. Spatial Convolutional Network adopts the same idea by aggregate the features of neighboring nodes into the center node.

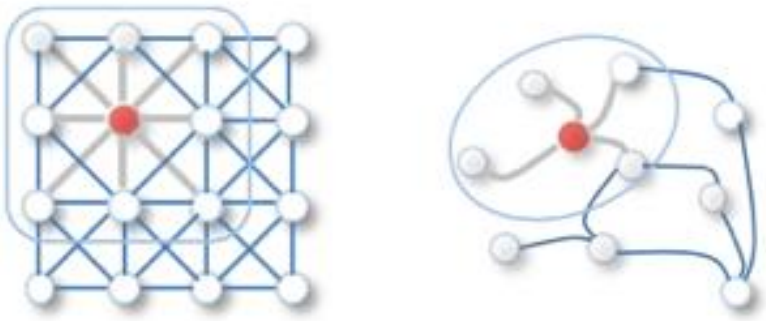

Figure 2.7: Left: Convolution on a regular graph such as an image. Right: Convolution on the arbitrary graph structure.

2.4.4. Spectral Convolutional Network

As compared to other types of GNN, this type of graph convolution network has a very strong mathematics foundation. Spectral Convolutional Network is built on graph signal processing theory. And by simplification and approximation of graph convolution. By Chebyshev polynomial approximation (Hammond et al. 2011), graph convolution can be simplified to below form:

$$g_{\theta'} * x \approx \sum_{k=0}^{K} \theta_k T_k(\Lambda)$$

Equation 2.5

2.5. Deep Reinforcement Learning

is a subfield of machine learning that combines reinforcement learning (RL) and deep learning. RL considers the problem of a computational agent learning to make decisions by trial and error. Deep RL incorporates deep learning into the solution, allowing agents to make decisions from unstructured input data without manual engineering of the state space. Deep RL algorithms can take in very large inputs (e.g., every pixel rendered to the screen in a video game) and decide what actions to perform to optimize an objective (e.g., maximizing the game score). Deep reinforcement learning has been used for a diverse set of applications including but not limited to robotics, video games, natural language processing, computer vision, education, transportation, and healthcare.

Q-Learning is a simple yet very powerful algorithm to create a cheat sheet for our agent. This helps the agent decide exactly what action to take. It is very evident that we cannot infer the Q value for

new cases from the countries that have already been explored. This presents two problems:

- First, the amount of memory required to save and update this table will increase with the increase in the number of cases.
- Second, the amount of time required to explore each case to generate the required Q table would be unrealistic.

In deep Q-learning, we use a neural network to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output. All previous experiences are stored by the user in the memory the next procedure is determined by the maximum Q network output. The loss function here is the mean square error of the expected Q value and the target Q value - Q *. This is basically a regression problem. However, we do not know the objective or actual value here as we are dealing with the problem of reinforcement learning. Back to the equation for updating the Q value derived from Bellman's equation.

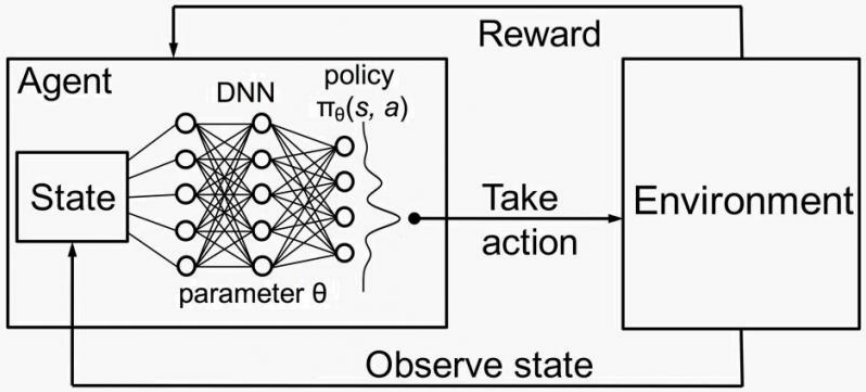

Figure 2.8: Deep Reinforcement Learning

Chapter 3: Design Specification

3.1. Analysis and Requirements

3.1.1. System Requirements

3.1.1.1. Functional Requirements

Table 3.1: Functional Requirement

ID	Label	Requirements
1	**REQ-Traffic Simulation**	Admin only can Show the traffic Graphically using SUMO, but it will take more time to train the model.
2	**REQ-Road Info.**	The algorithm should receive road information to train the model.
3	**REQ-Adjust Duration**	By measuring the traffic lined up on the road, signal timings should be adjusted automatically using RL.
4	**REQ-Green-Light Duration**	The system should calculate the minimum Green-Light duration to prevent congestion using RL.
5	**REQ-Vehicles Count**	The system will calculate the number of vehicles in each lane and avg speed.
6	**REQ-Predict Vehicles**	The system will predict the number of vehicles for the next 5 minutes using GNN based on previous traffic data.

7	**REQ-predict Duration**	The System will predict the green light duration for the next 5 minutes using GNN
8	**REQ-Minimum Time**	The System chooses the minimum possible phase duration that prevents congestion.

3.1.1.3. Non-Functional Requirements

Table 3.2: Non-Functional Requirements

ID	Label	Requirements
9	**REQ-Availability**	The system should be available on devices that have the required libraries installed.
10	**REQ- Reliability**	System should be reliable enough to satisfactorily the performance.
11	**REQ- Supportability**	By measuring the traffic lined up on the road, signal timings should be adjusted automatically using RL. It should provide support to user to easily access all the pages without much effort and It should be capable to update and maintain in future.
12	**REQ-Maintainability**	System should be easily maintainable. It should be flexible enough to stand with change and exceptions. The system should also handle new requirements. It should have capability to maintain in new

		environment.
13	REQ- Usability	The system should be usable and easy to modify and maintain.
14	REQ- Scalability	The system Should be scalable and fit in heavier traffic intersection.
15	**REQ- Security**	Security is the main issue. System should be safe and ensure security. It will ensure secure transfer of data and no one can access the system from an unauthorized server.

3.1.2. Functional Requirement Specification

3.1.2.1. Stakeholders

(1) Government and authorities
(2) Car owners and lenders
(3) Commuters
(4) Car, ride-sharing passengers
(5) Emergencies respond units.
(6) Software Engineers
(7) Construction Companies and workers
(8) Freight transport Companies

3.1.2.3. Actor and Goals

3.1.2.3.1. Actors

1. On board Units (OBUs)
2. Database
3. Predictive Model
4. DQN Model
5. Traffic Signal Controllers

3.1.2.3.2. Goals

6. Reducing congestion at intersections
7. Increasing the efficiencies of traffic signals
8. Increase the throughput of the road network.

3.1.2.4. Use Cases

A. Use Case Description

Table 3.3: Use Case Description

Use Case Name	Description	Actors	Conditions
Send Data	Used by the On-board Units (OBUs) to Send data to our system and save to database	OBU, Database	30 seconds have passing, OBU connected to database
Train Predictive Model	The predictive model gets the training data form the base and train	Predictive Model Service, Database	When initializing the system
Predict Future Features	The predictive model gets the input for prediction from database and then save the output in the database	Predictive Model Service, Database	2.5 minutes passing
Get Green-Light Time	Get the expected green-Light time from the predictive	Traffic Signal Controller, DQN	Current phase close to end

	model service and the required green-light time from the DQN model	Model Service, Database	
Get Green-Light phase	Get the next phase from the DQN Model Service	Traffic Signal Controller, DQN Model Service	Current phase close to end

B. Use Case Diagram

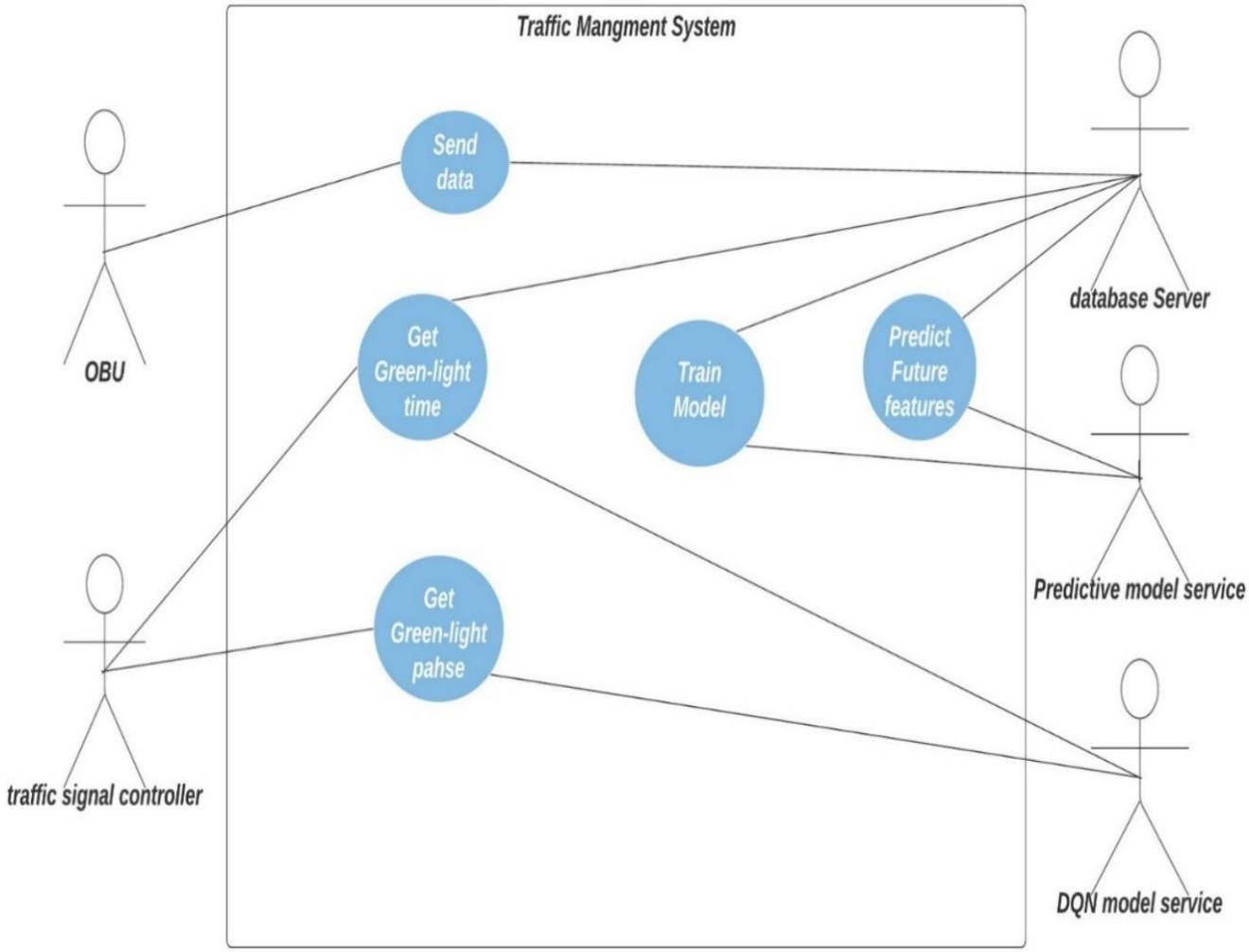

Figure 3.1:Use Case Diagram

3.2. Design

3.2.1. System Sequence Diagram

3.2.1.1. Send Data Sequence Diagram

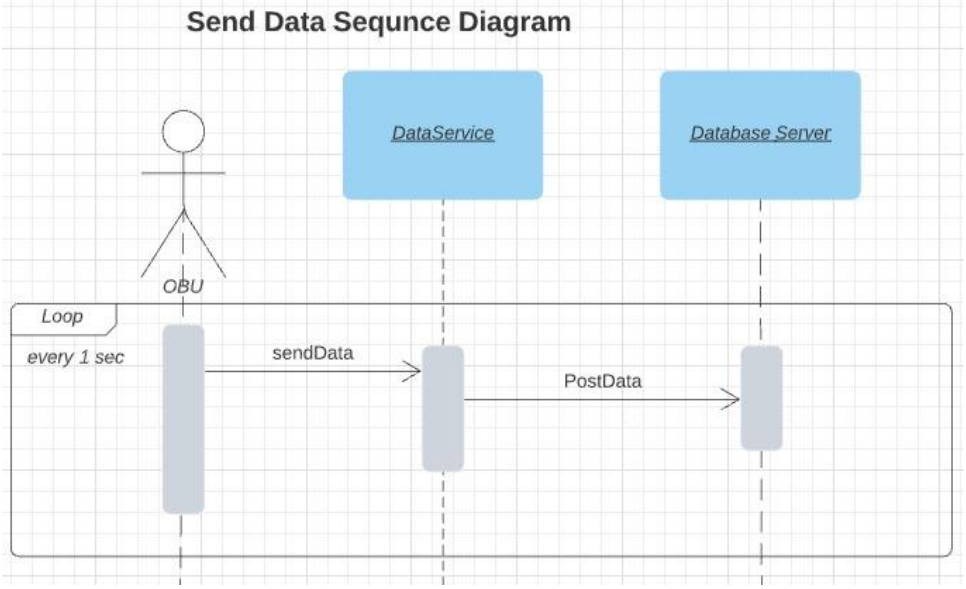

Figure 3.2: Send Data Sequence Diagram

3.2.1.2. Train Model Sequence Diagram

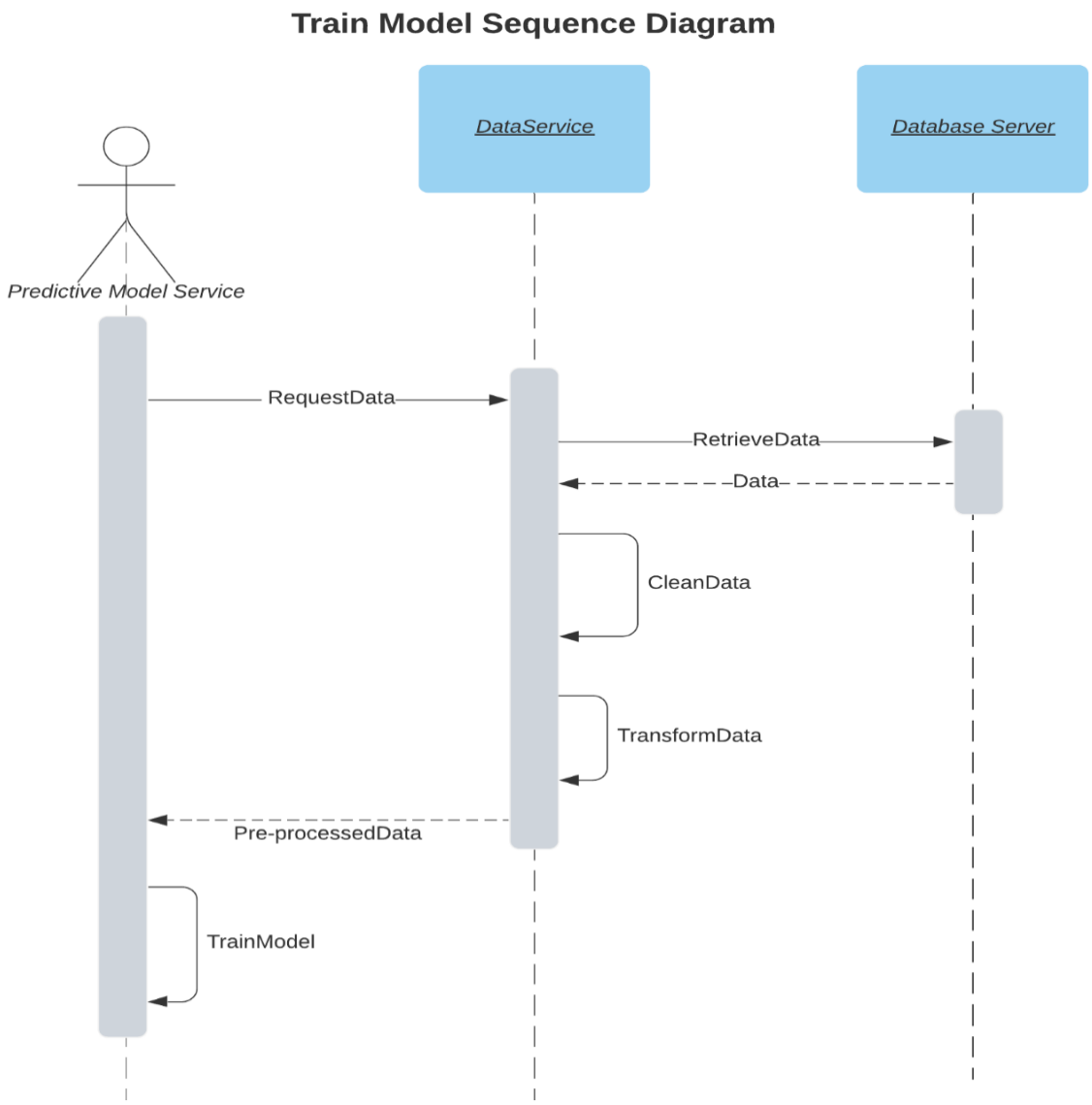

Figure 3.3: Train Model Sequence Diagram

3.2.1.3. Get Green-Light Time Sequence Diagram

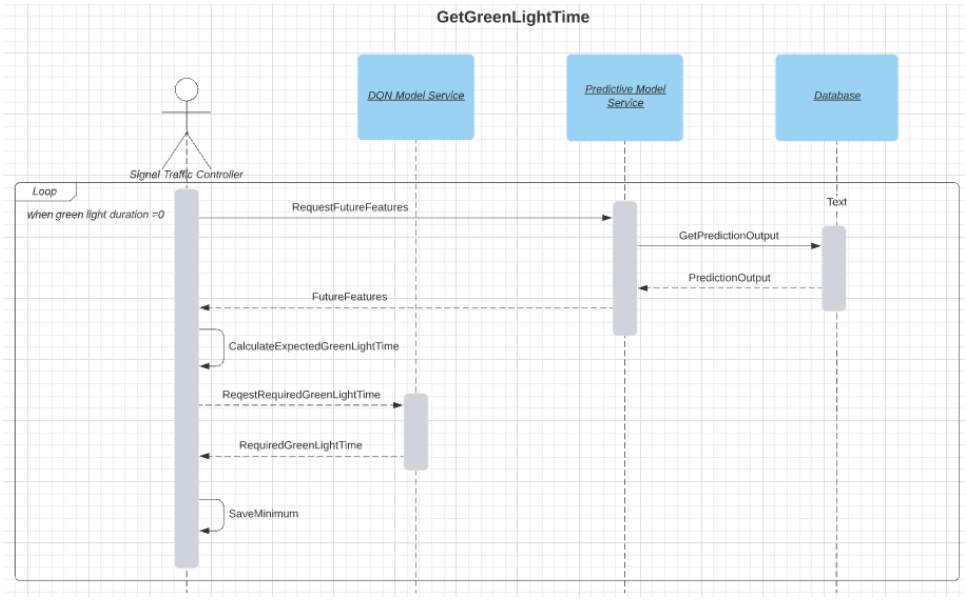

Figure 3.4: Get Green-light Time Sequence Diagram

3.2.1.4. Get Green-Light Phase Sequence Diagram

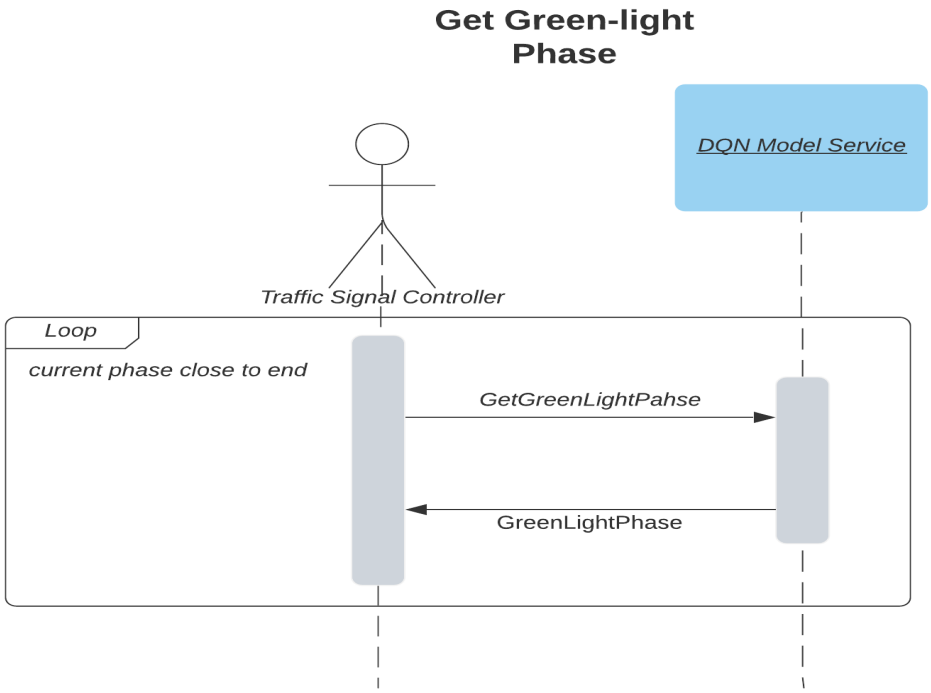

Figure 3.5: Get Green-Light Phase Sequence Diagram

3.2.1.5. Predict Future Features Sequence Diagram

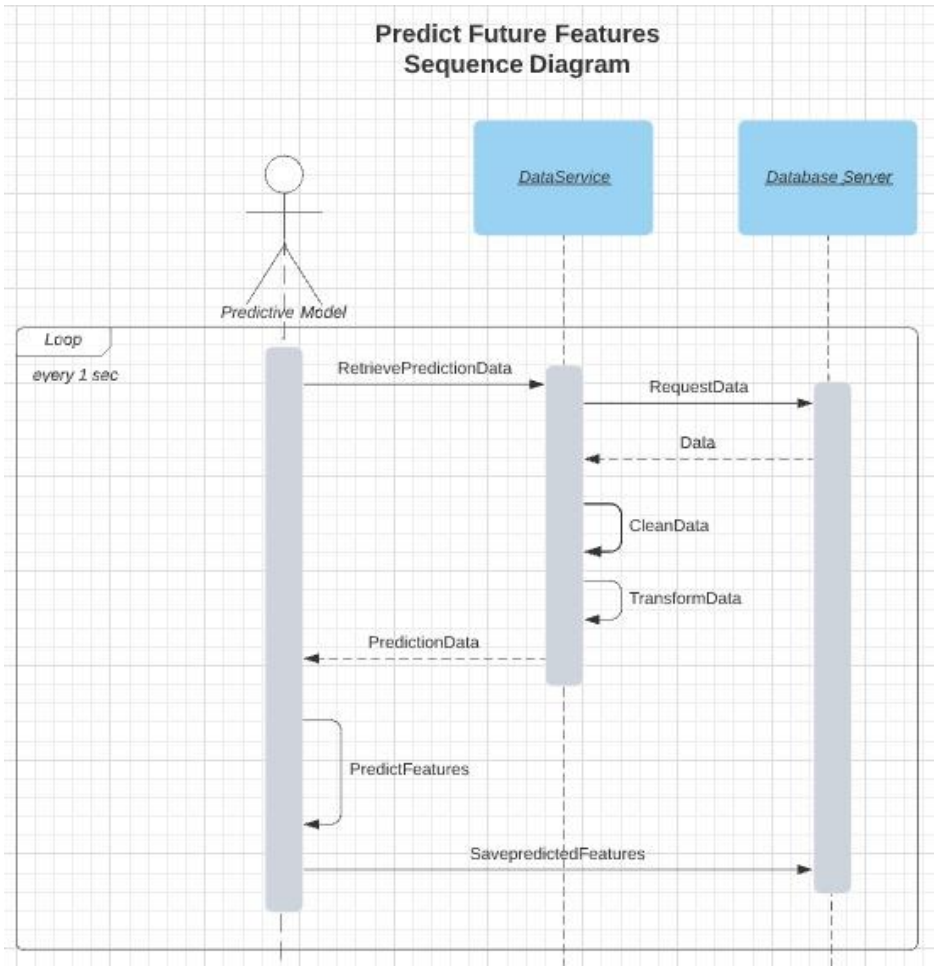

Figure 3.6: Predict Future Feature Sequence Diagram

3.2.2. Class Diagram and Interface

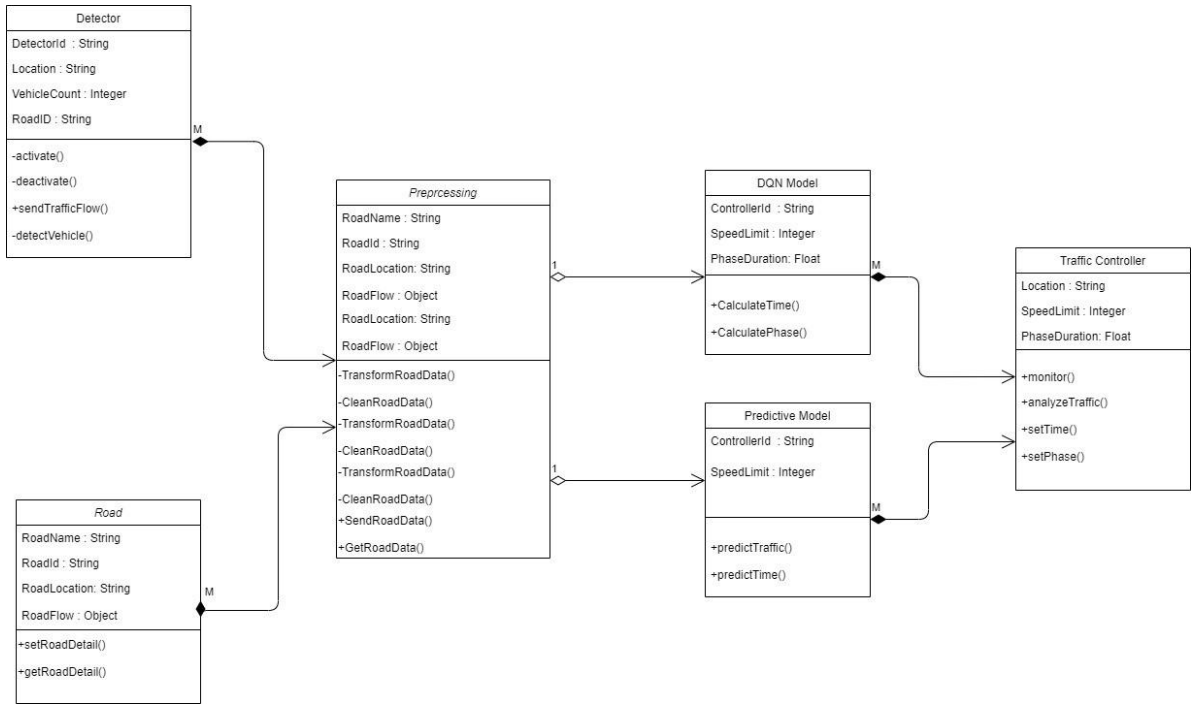

Figure 3.7: Class Diagram

Chapter 4: Development Tools and Environment

4.1. Simulation of Urban Mobility (SUMO)

The traffic micro simulator used for this project is SUMO (Simulation of Urban MObility). SUMO provides 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 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.

4.1.1. Network Building

A SUMO network file describes the traffic-related part of a map, the roads, and intersections the simulated vehicles run along or across. At a coarse scale, a SUMO network is a directed graph. Nodes, usually named "junctions" in SUMO-context, represent intersections, and "edges" roads or streets. Note that edges are unidirectional. Specifically, the SUMO network contains the following information:

- every street (edge) as a collection of lanes, including the position, shape, and speed limit of every lane,
- traffic light logics referenced by junctions,
- junctions, including their right of way regulation,
- connections between lanes at junctions (nodes).

Also, depending on the used input formats and set processing options, one can also find:

- districts,
- roundabout descriptions.

4.1.1.1. Edges and Lanes

"plain" and "internal" edges are encoded almost the same way, but they differ by the mandatory and used attributes.

4.1.1.1.1. Normal Edges

A "normal" edge is a connection between two nodes ("junctions"). <edge id="<ID>" **from**="<FROM_NODE_ID>" to="<TO_NODE_ID>" priority="<PRIORITY>">

Name	Type	Description
ID	Id(string)	The id of the edge
From	Id(string)	The id of the node it starts at
To	Id(string)	The id of the node it ends at
Priority	Id(string)	Indicates how important the road is (optional)
Function	enum ("normal", "internal", "connector", "crossing", ..)	An abstract edge purpose (optional with default "normal"

For the simulation, only the "function" attribute is of interest. It describes how the edge is used, and whether it is an edge that can be found within the real world or only a helper construct used for assignment. The following purposes are defined:

- **normal**: The edge is a plain part of a road network, like a highway or a normal street which connects two roads
- **connector**: The edge is a macroscopic connector - not a part of the real-world road network. Still, within the simulation, no distinction is made between "connector"

roads and "normal" nodes. Only sumo-gui allows to hide connector edges.

- **internal**: The edge is a part of an intersection (is located within the intersection)

4.1.1.1.2. Lanes

Each edge includes the definitions of lanes it consists of. The following example shows a single edge with two lanes

```
<edge id="<ID>" from="<FROM_NODE_ID>" to="<TO_NODE_ID>" priority="<PRIORITY>">
        <lane id="<ID>_0" index="0" speed="<SPEED>" length="<LENGTH>" shape="0.00,495.05 248.50,495.05"/>
</edge>
```

Name	Type	Description
ID	id (string)	The id of the lane
index	running number (unsigned int)	A running number, starting with zero at the right-most lane
speed	float	The maximum speed allowed on this lane [m/s]
Length	float	The length of this lane [m]
Shape	Position vector	The geometry of the lane, given by a polyline that describes the lane's center line; must not be empty or have less than two positions

4.1.1.2. Internal Edges

An internal edge lies within an intersection and connects and incoming *normal* edge with an outgoing *normal* edge. Internal edges are not included if the network was built using the **--no-internal-links** option. An example of an internal edge may look like this:

```
<edge id="<ID>" function="internal">
... lanes ...
 </edge>
```

Name	Type	Description
ID	id (string)	The id of the internal edge
Function	"internal"	Always "internal" for an internal edge

4.1.2. Traffic Light Programs

A traffic light program defines the phases of a traffic light.

```
<tlLogic id="<ID>" type="<ALGORITHM_ID>"
programID=<PROGRAM_ID>" offset="<TIME_OFFSET>">

     <phase duration="<DURATION#1>" state="<STATE#1>"/>
        <phase duration="<DURATION#1>"
state="<STATE#1>"/>

     ... further states ...

     <phase duration="<DURATION#n>" state="<STATE#n>"/>
</tlLogic>
```

4.1.3. Plain Junctions

Junctions represent the area where different streams cross, including the right-of-way rules vehicles must follow when crossing the intersection. An example may be:

```
<junction id="<ID>" type="<JUNCTION_TYPE>"
x="<X_POSITION>" y="<Y_POSITION>"
incLanes="<INCOMING_LANES>"
intLanes="<INTERNAL_LANES>" shape="<SHAPE>">

     ... requests ...

</junction>
```

Name	Type	Description
id	id (string)	The id of the junction; please note, that a traffic light definition must use the same ID when controlling this intersection.
x	x-position (real)	The x-coordinate of the intersection
y	y-position (real)	The y-coordinate of the intersection
Inclanes	id list	The ids of the lanes that end at the intersection; sorted by direction, clockwise, with direction up = 0
Intlanes	id list	The IDs of the lanes within the intersection
Shape	position list	A polygon describing the road boundaries of the intersection

4.1.3.1. Requests

What follows are "requests", looking like:

```
<request index="<INDEX>" response="<RELATIVE_MAJOR_LINKS>" foes="<FOE_LINKS>" cont="<MAY_ENTER>"/>
```

Name	Type	Description
Index	index (unsigned int)	The index of the described connection in the right-of-way matrix

response	Bit set (string)	A bitstring that describes for each connection whether it prohibits un-decelerated passing of the intersection for vehicles at this connection. The rightmost bit corresponds to index 0.
foes	Bit set (string)	A bitstring that describes for each connection whether it conflicts with this connection. The rightmost bit corresponds to index 0.
cont	bool	Whether a vehicle may pass the first stop line to wait within the intersection until there are no vehicles with higher priority. This is typically the case for left-moving streams from the prioritized direction.

4.1.4. Internal Junctions

"internal" junctions are used to define a waiting position within the intersection for some traffic relations. This is typically used for left-turning vehicles from the prioritized direction or for right-turning vehicles that have to wait for straight-going pedestrians.

These junctions do not need a right-of-way matrix but only following information:

- the lanes that are incoming into the junction the internal junction is in and prohibit to pass the internal junction if a vehicle is approaching
- the internal lanes that must not be occupied for crossing the internal junction

An internal junction is encoded like this:

```
<junction id="<ID>" type="internal" x="<X-POSITION>" y="<Y-POSITION>" incLanes="<INCOMING_PROHIBITING_LANES>" intLanes="<INTERNAL_PROHIBITING_LANES>"/>
```

The ID is the same as the lane to use when crossing the internal junction itself. The type is always "internal". Please note that an internal junction is usually encoded in one line. The attributes of an internal junction are:

Name	Type	Description
id	id (string)	The id of the junction; please note, that a traffic light definition must use the same ID when controlling this intersection.
x	x-position (real)	The x-coordinate of the intersection
y	y-position (real)	The y-coordinate of the intersection
Inclanes	id list	The ids of the lanes that end at the intersection; sorted by direction, clockwise, with direction up = 0
Intlanes	id list	The IDs of the lanes within the intersection

4.1.5. Demand Modelling

After having generated a network, one could look at it is using sumo-gui, but no cars would be driving around. One still needs some kind of description about the vehicles. This is called the *traffic demand*. From now on we will use the following nomenclature: A **trip** is a vehicle movement from one place to another defined by the starting edge (street), the destination edge, and the departure time. A **route** is an expanded trip, that means, that a route definition contains not only the first and the last edge, but all edges the vehicle will pass. sumo and sumo-gui need routes as input for vehicle movements. There are several ways to generate routes for SUMO. The choice depends on your available input data:

1. Using trip definitions

2. As described above, each trip consists at least of the starting and the ending edge and the departure time. This is useful for when you want to create *demand* by hand or when writing your own scripts to import custom data. You may either use duarouter to turn your trips into routes.
3. Using flow definitions

   This is mostly the same approach as using trip definitions, but one may join vehicles having the same departure and arrival edge using this method
4. Using Randomization

   This is a quick way to get some traffic if you do not have access to any measurements, but the results are highly unrealistic.
5. Using OD-matrices
6. Using flow definitions and turning ratios
7. Using detector data (observation points)

4.1.6. Simulation

In the following, we would like to describe the input needed by the simulation.

4.1.6.1.1. Road Network

For a simulation, a SUMO Road Network must be given using the option **--net-file** <*NETWORK_FILE*> (or **-n** <*NETWORK_FILE*>). The network is normally built using netconvert or netgenerate.

4.1.6.2. Traffic Demand (Routes)

The vehicles to simulate must be given. Their description normally includes vehicle types, vehicles, and vehicle routes. Routes are normally given to the simulation modules using the option:

--route-files <*ROUTES_FILE*>[,<*ROUTES_FILE*>]* (or **-r** <*ROUTES_FILE*>[,<*ROUTES_FILE*>]*). As you can see, you can use more than one route file within a single simulation run.

The routes MUST be sorted. The reason is that we want to simulate large road networks with up to millions of routes. Using a plain PC this is only possible if you do not keep all routes in memory. All files given as parameter to **--route-files** <*ROUTES_FILE*>[,<*ROUTES_FILE*>]* are read step-wise. Starting at the begin time step, new routes are loaded every n time steps for the next n time steps. n may be controlled using the **--route-steps** <*INT*> where <=0 forces sumo/sumo-gui to load the file completely. Fetching routes for the next steps only implies that the vehicle types - or maybe "global" routes - must be given in prior to the routes that use them.

4.2. Traci

TraCI is the short term for "**Tra**ffic **C**ontrol **I**nterface". Giving access to a running road traffic simulation, it allows to retrieve values of simulated objects and to manipulate their behavior "on-line". TraCI uses a TCP based client/server architecture to provide access to sumo. Thereby, sumo acts as server that is started with additional command-line options: --remote-port <INT> where <INT> is the port sumo will listen on for incoming connections. When started with the --remote-port <INT> option, sumo only prepares the simulation and waits for all external applications to connect and take over the control. Please note, that the --end <TIME> option is ignored when sumo runs as a TraCI server, sumo runs until the client demands a simulation end. When using sumo-gui as a server, the simulation must either be started by using the play button or by setting the option --start before TraCI commands will be processed.

4.3. NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical

operations, random simulation and much more. We used the NumPy library mainly in creating arrays and matrices to apply numerical operations on them. Also, NumPy is used in generating random numbers of the Weibull distribution NumPy could be installed using python package manager by running the following commands.

```
pip install numpy
import numpy as np
```

4.4. Pandas

Pandas is an open-source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named Numpy, which provides support for multi-dimensional arrays. As one of the most popular data wrangling packages, Pandas works well with many other data science modules inside the Python ecosystem, and is typically included in every Python distribution. We used pandas mainly in creating DataFrames to deal with generated structured data from SUMO We used pandas in converting a list of lists to structured DataFrames with specific columns names, then saved the DataFrames in a CSV file. Pandas could be installed using python package manager by running the following commands.

```
pip install pandas
import pandas as pd
```

4.5. Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. We used matplotlib at the end of the simulation to visualize the results for better understanding of the model performance, Then compare each model with another. Matplotlib could be installed using python package manager by running the following commands.

```
pip install matplotplib
import matplotlib as plt
```

4.6. TensorFlow & Keras

TensorFlow can train and operate deep neural networks for handwritten number classification, image recognition, word embedding, repetitive neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE (partial differential equation) based simulations. The best thing is that TensorFlow supports production forecasting on a large scale, with the same models used in training.

Keras is an inherently high-level neural network library written in Python - which makes it extremely simple and easy to use. Works as a wrapper for low-level libraries such as TensorFlow. written in Python and serving as the wrapping for TensorFlow and deep learning tools.

we build both models based on Keras API and compared the performance of each model at the end of the simulation to find the best performance model. And this yields to the basic ANN model performing better than the CNN model.

Tensorflow and Keras APi could be installed using python package manager by running the following commands.

```
pip install tensorflow
import tensorflow as tf
from tensorflow import keras
```

4.7. Spektral

Figure 4.1: Spektral library

Spektral is a Python library for graph deep learning, based on the Keras API and TensorFlow You can use Spektral for classifying the users of a social network, predicting molecular properties, generating new graphs with GANs, clustering nodes, predicting links, and any other task where data is described by graphs. Spektral implements some of the most popular layers for graph deep learning including

- Graph Convolutional Networks (GCN)
- Chebyshev Convolutions
- Graph Sage
- ARMA Convolutions
- Diffusional Convolutions

In our project we used Spectral library in the Predictive GNN model to calculate the Chebyshev convolution Spektral could be installed and used by running the following commands.

```
pip install spektral
from spektral.layers.convolutional import ChebConv
```

Chapter 5: Network & Models Design

5.1. Road Network Representation

We use a complex road network with multiple intersections as a scenario for traffic prediction and control. Multiple intersections of the road network use traffic lights to control the flow of traffic passing by, and every two intersections are connected by directed lanes, as shown in the Figure 5.1.

For each intersection, there are four approaches in four directions: east, west, north, and south, denoted by E, W, S, N. This is one of the most common situations in real-world traffic. The approach in each direction is divided into two directions: incoming and outgoing. Vehicles approach the intersection from incoming lane, pass the intersection and then leave from outgoing lane. The incoming lane is also divided into three lanes: left, right and straight, which means the vehicles travelling on it will go in three different directions. We set up an agent at each intersection, which uses three traffic lights (i.e., red, yellow, and green) to control the traffic flow.

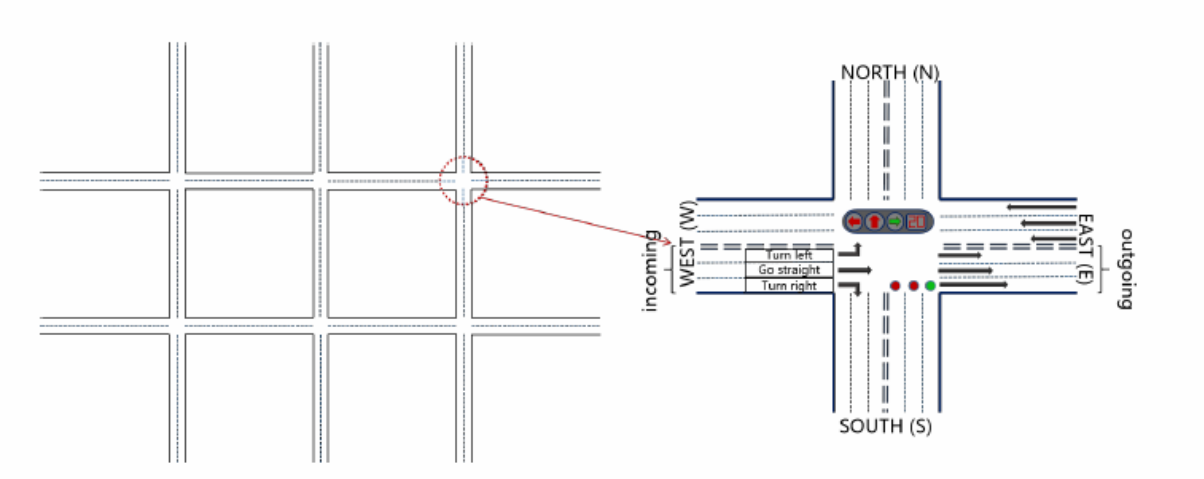

Figure 5.1: Network Representation

Project architecture starts with on board unit (OBU) which is like navigation system in the car which give us some useful information about the car (longitude, latitude, speed, timestamp, angle), then preprocessing operation is applied to this data as with longitude and latitude we can know the road id, then the data is stored in any data server like cloud in the shape of 3D matrix. The x-axis represents the different features, the y-

axis represents different nodes, and the z-axis represents different time stamps, which mean that one row represents features for a certain node at certain time. The data reach this shape as we see which data came at the same time stamp and then see the road id which determine the car between which 2 nodes and finally with the angle, we can determine the car is incoming to which node, then we can update the number of car in this direction and the average speed.

The data then goes to the predictive model and DQN model, the output of predictive model is the expected green light time (the time expected that the traffic will need it to be green on it). The output of DQN model is the required green light time and green light phase. Then the two times are compered, and the smaller time will be the actual time that the traffic will be green on it.

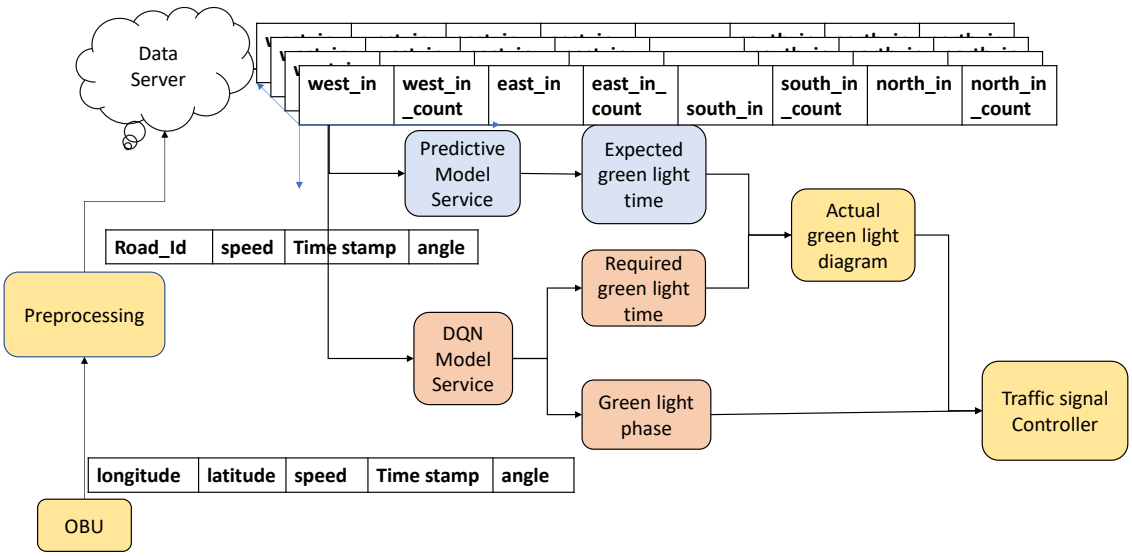

Figure 5.2: Model Architecture

5.2. The Predictive Model

The predictive model is consisting of two spatial-temporal convolution blocks and a fully connected layer, which are cascaded together. Each convolution block contains two gated temporal convolutional layers and a spatial graph convolutional layer in the middle of them. The spatial-temporal correlation information of traffic flow is extracted by convolution blocks, and the features obtained are integrated and processed by the fully-connected output layer to generate prediction.

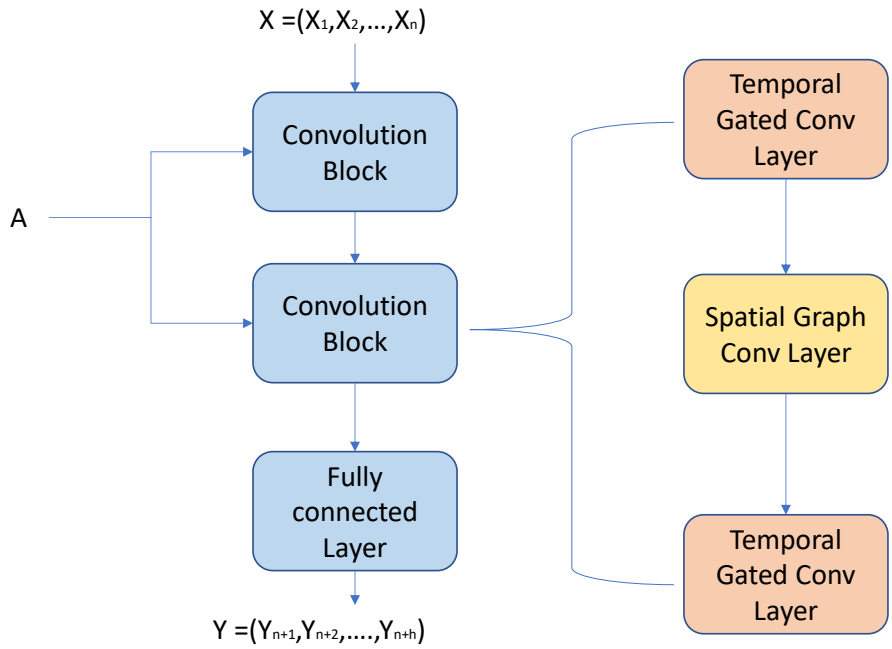

Figure 5.3: Predictive Model

5.2.1. gated temporal convolutional layers

The temporal convolutional layer is used to capture the temporal characteristics of the traffic flow, which consists of a one-dimensional causal convolution and a gated unit. Let Y2 Rmcin represents the input to the time convolutional layer, where m represents the size of temporal, and cin represents channel dimensions. The temporal gat×ed convolution can be defined as:

$$b\square\_Y = Y_1 \_ \_ (Y_2)$$

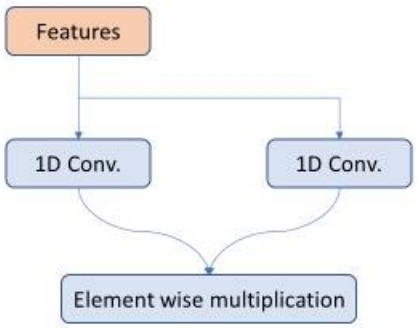

Figure 5.4: Gated Temporal Convolution Layer

5.2.2. spatial graph convolutional layers

GNN is used to extract spatial information from previous traffic flow observations. The convolution is carried out directly on the data with graph structure, which preserves the spatial structure features of the traffic network. To apply the standard convolution to the graph structure, the Fourier transform is used to apply the convolution to the spectral domain, commonly known as the spectral graph convolution. In the concept of spectral convolution, the graph convolution operator is defined as follows, such as the convolution between the convolution kernel _ and an input signal x:

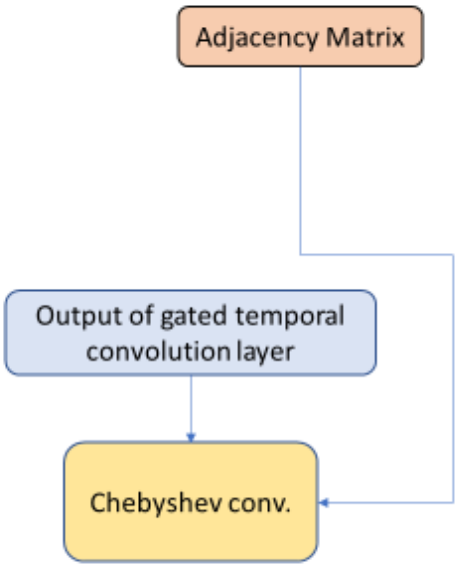

Figure 5.5: Inputs of Chebychev Convolution

5.3. Deep Q-learning Model

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 (5.1).

$$n_{i,t}^{out} = Q(s_t, a_{j,t}) \tag{5.1}$$

Where 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). 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 5.1.

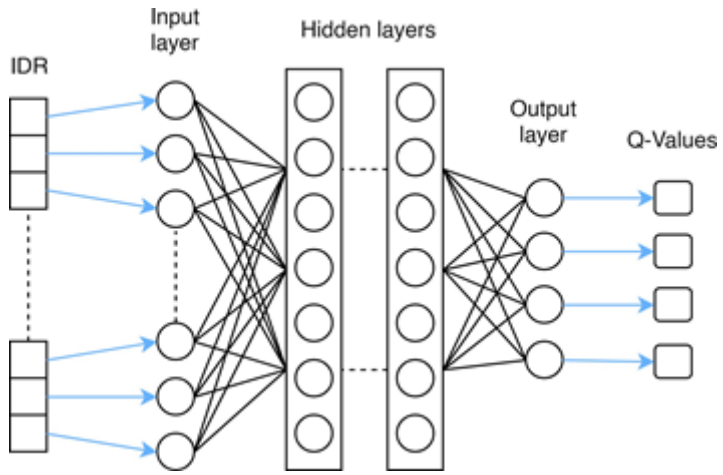

Figure 5.6: Design of the NN of the DQN

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.

5.3.1. 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. . 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 5.7 shows the state representation for the west arm of the intersection.

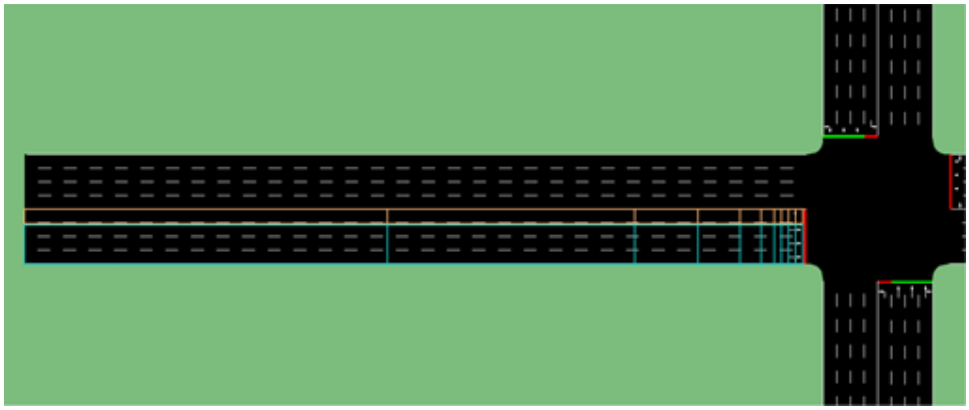

Figure 5.7:The state representation in the west arm of the intersection.

It must be noted that a particular cell does not necessarily describe the situation in a single lane. As seen in Figure 5.2 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 5.2, 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.

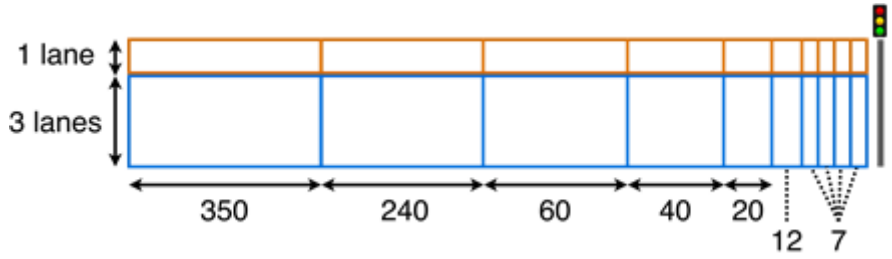

Figure 5.8: Design of the state representation in the west arm of the intersection, with cells length. The length of the longer cells has been reduced for readability

5.3.2. 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 some amount of time. As described in Section 5 the green time is set at m seconds(minimum time between predictive and DQN models) 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 (5.5).

$$A = \{\text{NSA, NSLA, EWA, EWLA}\} \qquad (5.5)$$

The set represents every possible action that the agent can take. Every action *a* of set (5.5) is described below and is represented in figure 5.4.

- 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.

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 m(t)+m(t+1) since 1 simulation step is equal to 1 second in SUMO Simulator . When the two consecutive actions are different, the yellow phase counts as 4 simulation steps and then the selected action counts as model m(t) simulation steps, (which m(t) is the minimum time between Predictive model and DQN at timestep t) for a total of 4+m(t) simulation steps.

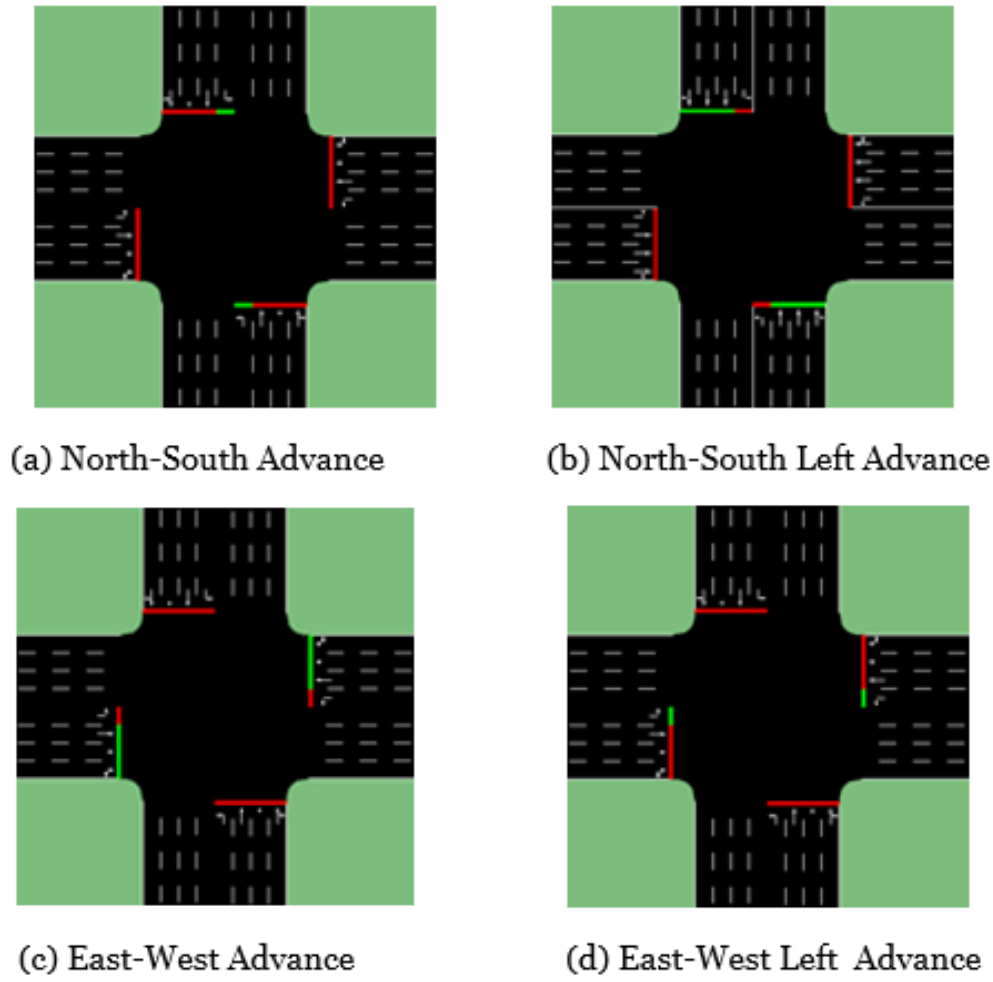

Figure 5.9: The Four possible actions

5.3.3. The reward functions

In reinforcement learning, the reward represents the feedback from the en- vironment 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 fu- ture 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 mea- sure of traffic efficiency, so the agent is able to understand if the taken action reduce or increase the intersection efficiency. In traffic analysis, several mea- sures are used [14], such as throughput, mean delay and travel time. In

this thesis, the candidate measures to generate the reward were the following.

1. Queue length: the number of vehicles with a speed of less than 0.1 m/s.
2. 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.
3. 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. For- mally, the total waiting time is defined in equation (5.6).

$$twt_t = \sum_{veh=1} wt_{(veh,t)} \quad (5.6)$$

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. The reward function that generates the reward for the agent at timestep t and it is defined in equation: $r_t = twt_{t-1} - twt_t$

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. the reward usually can be positive or negative, and this implementation is no exception. The equation 5.7 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

Chapter 6: Experimental setup and training

The environment setup was divided into two experiments first one was Single intersection, and the second experiment was done on Multi-intersection Network (4 Traffic Lights). For the Single and Multiple we generated the training data using SUMO on three levels (low - moderate - high) density the data was generated according to Weibull distribution. Then the training is done on two models the predictive model and the Deep Q -network model.

6.1. Single Intersection

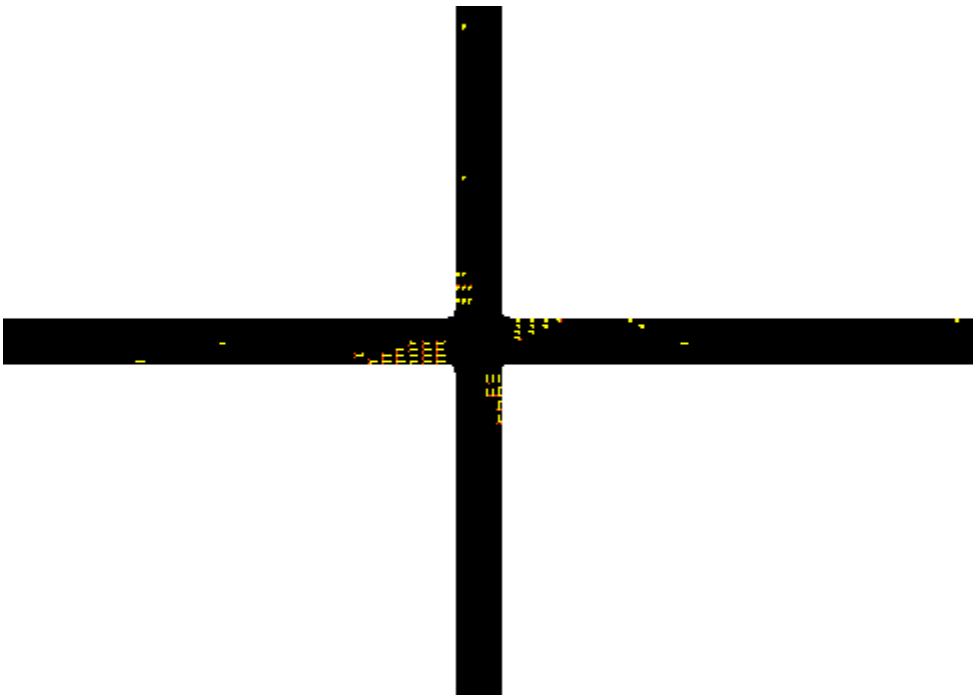

Figure 6.1: SUMO Single Intersection

Figure 6.1: SUMO Single Intersection

In the intersection, there are four edges in four directions: east, west, north, and south, denoted by E, W, S, N. The edge in each direction is divided into two directions: incoming and outgoing. Vehicles approach the intersection from incoming edges, pass the intersection and then leave from outgoing edges. The incoming edge is also divided into four lanes. And the outgoing has is divided the same and has three outgoing directions: left, right and straight, which means the vehicles travelling on it will go in three different directions. Each lane length is 750 meters.

We set up an agent at the intersection, which uses three traffic lights (i.e., red, yellow, and green) to control the traffic flow. Each

intersection arm is divided into cells representing the presence of a car as previously mentioned in chapter 5.4

6.1.1. Traffic data generation

- low density → 1000 cars generated, 5400 timestep.
- moderate density → 2000 cars generated, 7500 timestep.
- high density → 2500 cars generated, 5400 timestep.

6.1.2. Static Traffic light timing:

Green phase = 10 seconds, yellow phase = 4 seconds

6.1.3. Adaptive Traffic light timing:

The timing depends on the DQN model and the Predictive model timings, predictive model output is the expected green light time for the next step (The time that the traffic light should be green to solve the traffic congestion for the net time step). the DQN model output the required green light time for the current time step. Then the traffic control selects the least time and execute it.

6.1.4. Single Intersection Training data

The generated data from simulation was by getting each car info by vehicle_id and save this information in .xml file.

```
<timestep time="56">
    <vehicle id="N_S_0" x="-11" y="557" angle="180" type="standard_car" speed="14" pos="196" edge="N2TL" lane="N2TL_0" />
</timestep>
```

Figure 6.2: Single intersection Car data

then we read the data.xml and calculated cars_count and Avg_speed for each lane. This process was taking extra time for the calculations, then preprocessing is done and save the structured data in a .csv file contain the training features.

Time_ step	east_ speed	east_ count	west_ speed	west_ count	north_ speed	north_ count	south_ speed	south_ count

The features are extracted for each time step we calculated 2 parameters (average Speed, Count) for each lane (east, west, north, south), resulting 8 features for each time step.

6.2. Multiple Intersection

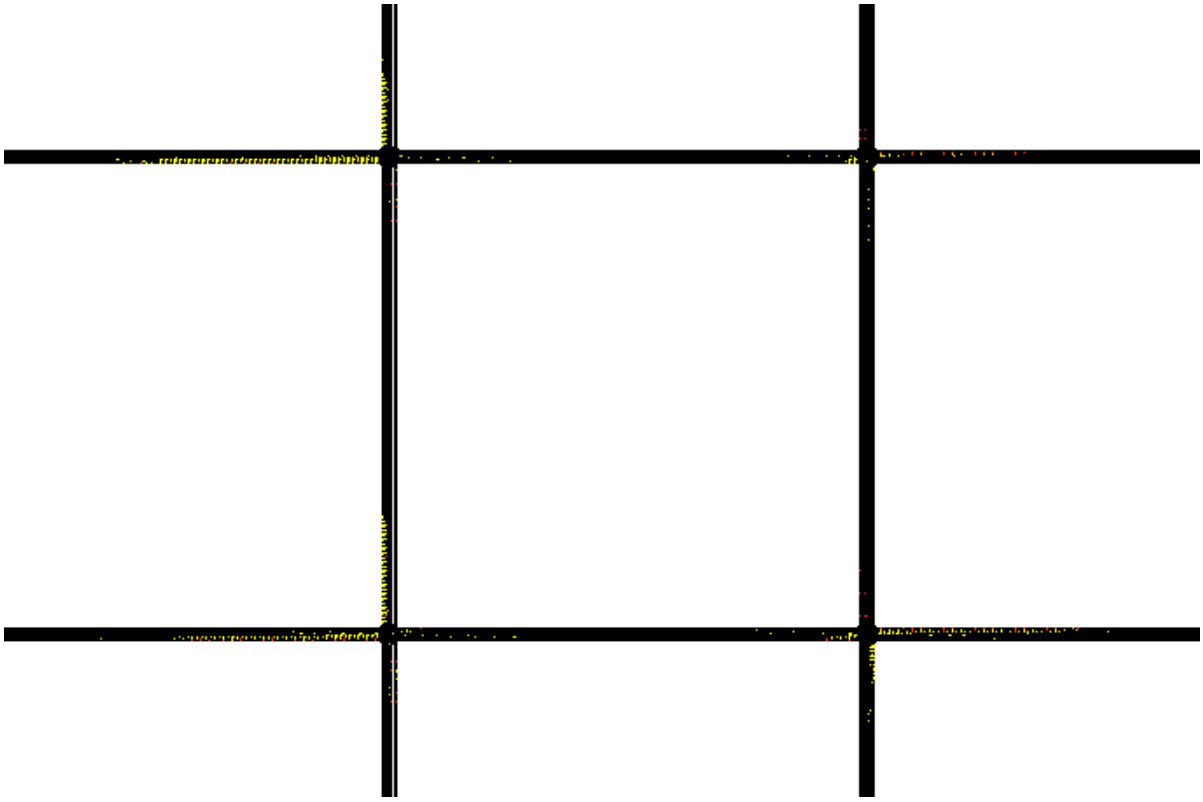

Figure 6.3: SUMO Multiple Intersection

In each of the four intersections, each intersection connected with another two intersections each with a unique edge, each edge length is 800 meters. Hence each intersection is affected by the congestion and the performance of the other intersections also. The lanes are divided as previously mentioned in the single intersection lanes.

6.2.1. Traffic Data generation

- low density → 2000 cars generated, 5400 timestep.
- moderate density → 4000 cars generated, 7500 timestep.
- high density → 5000 cars generated, 5400 timestep.

6.2.2. Static Traffic light timing:

Green phase = 10 seconds, yellow phase = 4 seconds

6.2.3. Adaptive Traffic light timing:

The timing depends on the DQN model and the Predictive model timings, predictive model output is the expected green light time for the next step (The time that the traffic light should be green to solve the traffic congestion for the next time step). the DQN model output the required green light time for the current time step. Then the traffic control selects the least time and execute it.

6.2.4. Multiple Intersection Training data

For Multiple intersection we generated the data in a more efficient method. The 2 features (Vehicles_Count, Lane_avgSpeed) of each edge is directly fetched from the simulation using *Traci* library from *SUMO.*

```
TimeStamp,Node,east_speed,east_count,west_speed,west_count,north_speed,north_c
ount,south_speed,south_count
```

Time stamp	Node id	8 * features of the node

Now we have the extracted data from simulation in a .csv file, this shape for each time step we have 8 features for each traffic intersection.

TimeStam	Node	east_spee	east_coun	west_spe	west_cou	north_spe	north_cou	south_spe	south_count
0	tl1	0	0	0	0	0	0	0	0
0	tl2	0	0	0	0	0	0	0	0
0	tl3	0	0	0	0	0	0	0	0
0	tl4	0	0	0	0	0	0	0	0

Figure 6.4: Multiple intersection lanes data

The data is preprocessed and cleaned to be ready for model training, the csv file is grouped by [‘Node’] column, then apply to.numpy() method to transform the dataset into a 3D matrix (TimeStamp * Node * 8 Features).

Now the 3D matrix is the input training data for our models.

6.3. Training DQN

In chapter 5, 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.

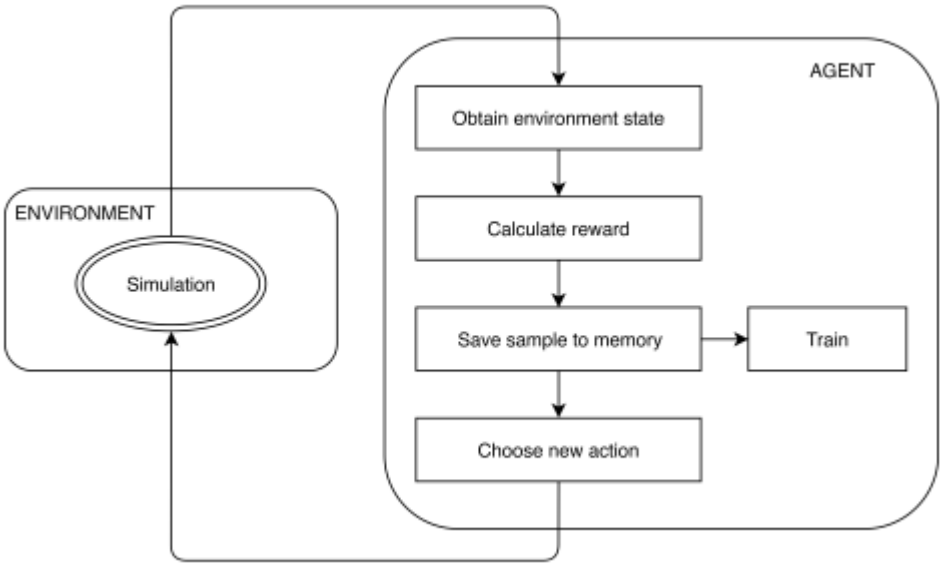

Figure 6.5: The workflow of the agent in a timestep

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 packs the information gathered and save it to a memory which is used for training purposes, as described in next Section 6.3.1 .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 5, the agent is trained and tested with the use of a traffic microsimulator. In this chapter, every decision regarding the simulation phase is described.

6.3.1. Experience Replay

Experience replay is a technique adopted during the training phase 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 gathers 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 (6.1).

$$m = \{s_t, a_t , r_{t+1}, s_{t+1}\} \qquad (6.1)$$

Where rt+1 is the reward received after taking the action at 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 6.6 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 is filled,

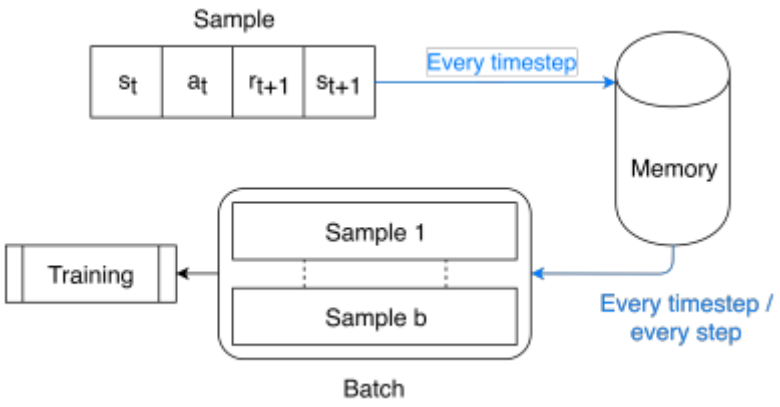

Figure 6.6: The memory handling before training.

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.

6.3.1.1. Sampling strategies

In Figure 6.6 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 7.

6.3.1.2. Advantages

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

- It removes correlations in the observation sequence.

- Refresh the experience of the agent.

In this environment, two consecutive states are naturally correlated since the state of the environment st+1 is a direct evolution of the state st . Most of the information contained in state st+1 does not derive as a consequence of an agent's action, but rather as the spontaneous transformation of the current situation st . 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.

6.3.2. 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 (6.1), 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 6.6 on the preceding page).

A single sample bk ∈ B contains the starting state st , the action made at , the consequent reward received rt+1 and the following state st+1. For every sample bk the following operations are performed.

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

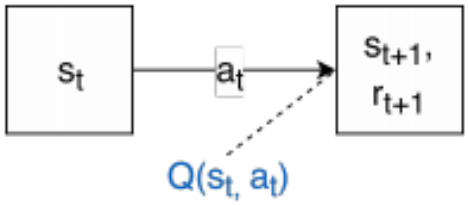

Figure 6.7: Computation of the Q-value Q for one sample.

2. Computation of Q-values Q0 (st+1, at+1) by submitting the vector IDR representing the next state st+1 to the neural network and obtaining the predicted Q-values relative to actions at+1. These represents how the environment will evolve and what values will probably have the next actions.

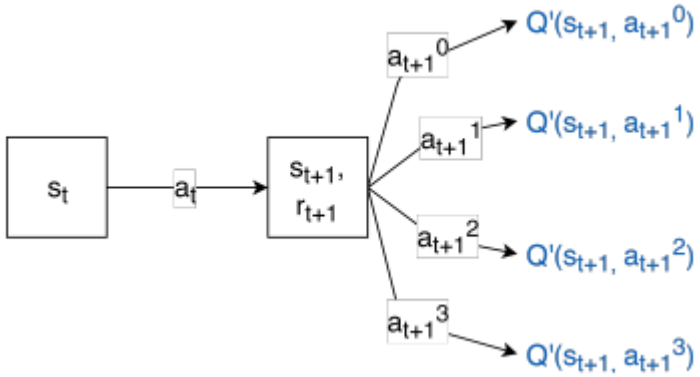

Figure 6.7: Computation of the Q-values Q' for one sample.

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, maxAQ0 indicates that the best possible Q-value is selected, representing the maximum expected future reward. It will be discounted by a factor γ that gives more importance to the immediate reward.

$$Q(st\ ,\ at) = rt+1 + \gamma \cdot \text{maxAQ}\ 0\ (st+1,\ at+1) \quad (6.2)$$

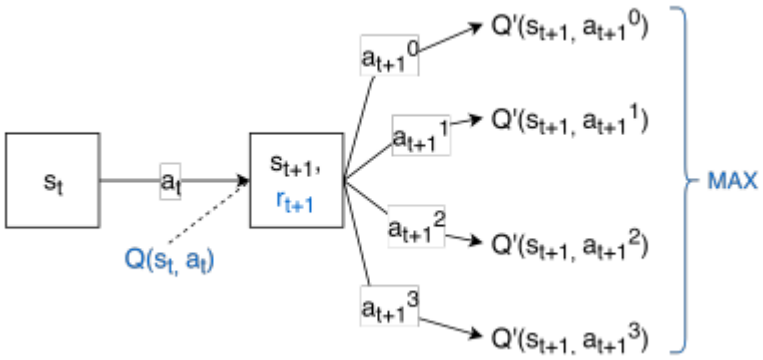

Figure 6.8: Information used (in blue) for the Q-value update using equation (4.2) for one sample.

4. Training of the neural network. The input is the vector IDR representing the state st , while the desired output is the updated Qvalues Q(st , at) that now includes the maximum expected future reward thanks to the Q-value update equation (6.2). By doing this, the next time the agent encounters the state st or a similar one, the neural network will be likely to output the Q-value of action at that is comprehensive of the best future situation.

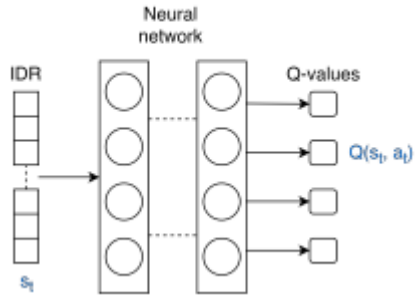

Figure 6.9: Training of the neural network for one sample.

6.3.3. 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 €-greedy. The equation, shown in (6.3) define a probability €-for the current episode h to choose an explorative action, and consequently a probability 1 − €-to choose an exploitative action.

$$€_h = 1 - (h \div H) \qquad (6.3)$$

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

Chapter 7: Results

7.1. Single Intersection

Here is the result of training the spatial-temporal model and after we ran the same 100 episodes of simulation for both the static traffic control and dynamic traffic signal using our system to compare the results of the negative reward, cumulative waiting time (delay) and average queue length of each episode here the plots of each metric for both the static and dynamic traffic signal control.

7.1.1. Predictive Model

After several modification to the Parameter of the models we found that the best result came when the batch size is equal to 9 and that the no. of prediction steps is equal to 1. This is on 100 episodes of simulation each contains 7500 timestep and a total of 2000 vehicles. The model had root mean square error (RMSE) of .4164 and mean absolute error (MAE) of .2604 this was the model we used for traffic prediction when training and running the DQN model.

7.1.2. Low-Density

7.1.2.1. Negative Reward

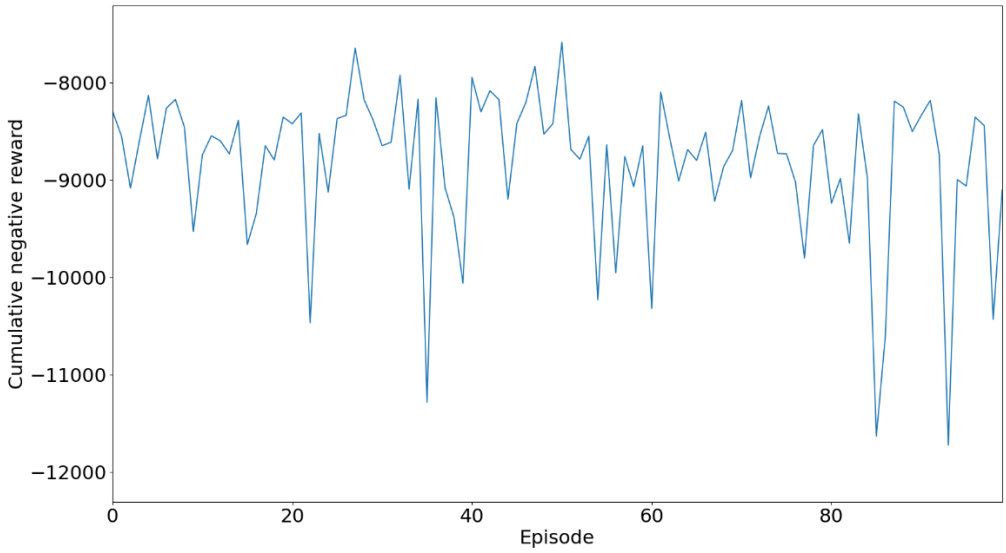

Figure 7.1: Negative reward of low-density static signal control for low-density

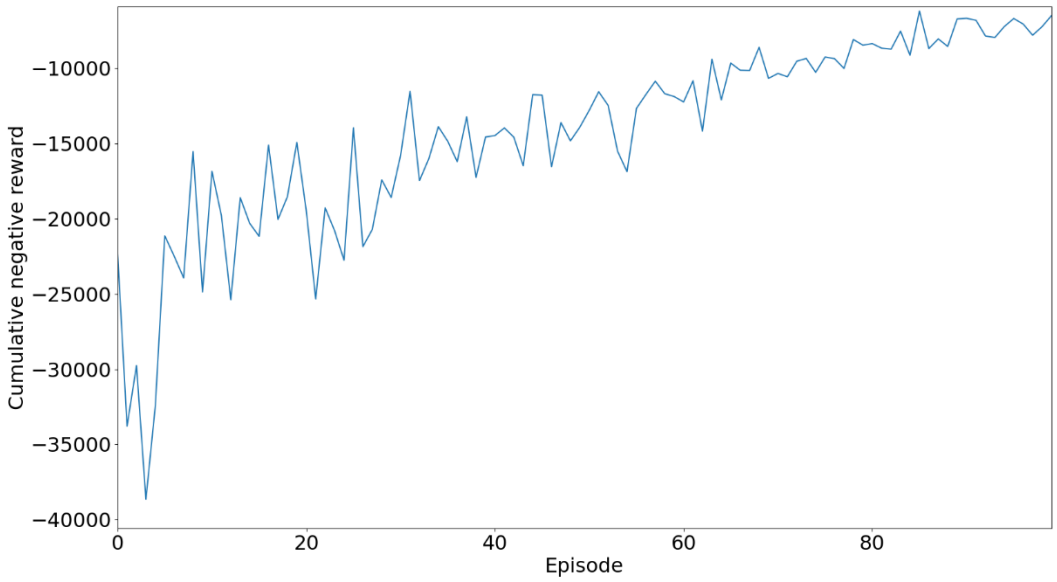

Figure 7.2: Negative reward of the dynamic traffic signal control for low-density

7.1.2.2. Cumulative Waiting Time

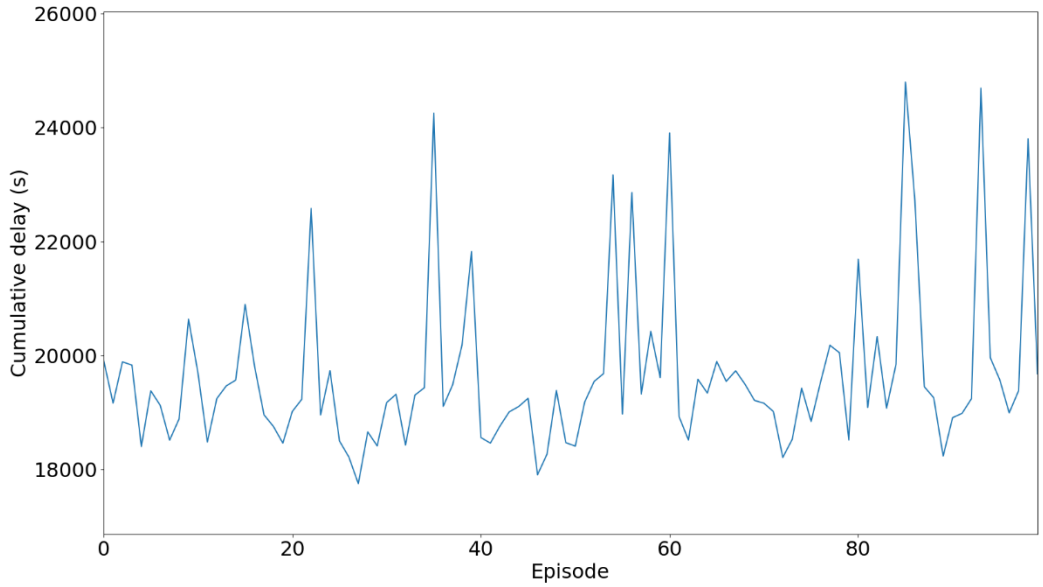

Figure 7.3: Cumulative Delay of static traffic signal control for low-density

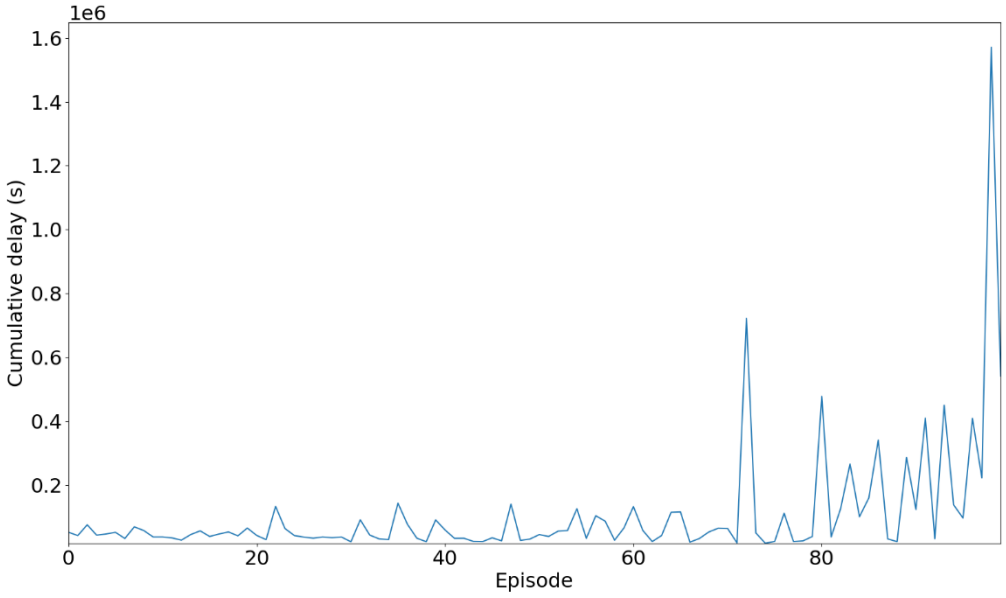

Figure 7.4:Cumulative Delay of dynamic traffic signal control for low-density

7.1.2.3. Average queue length

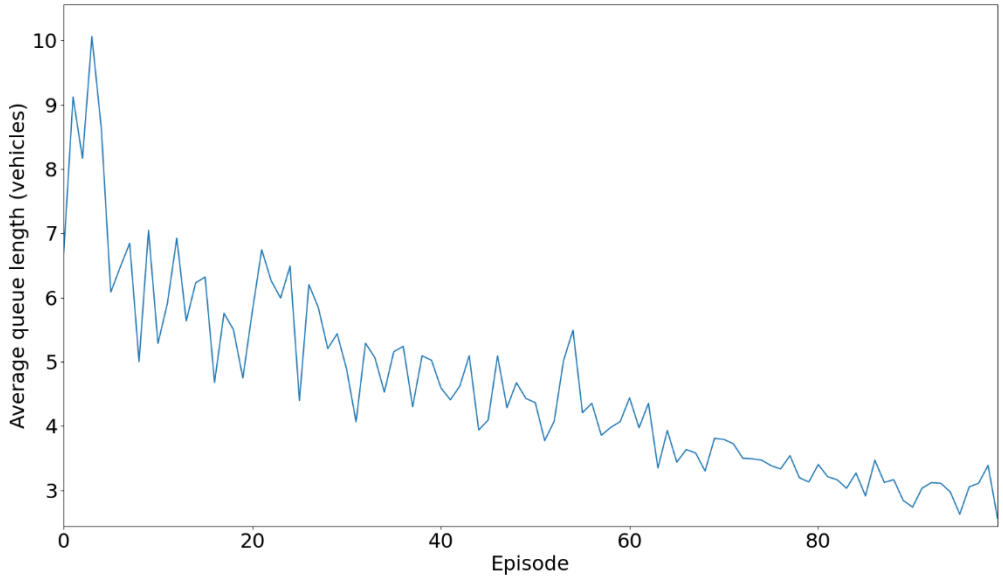

Figure 7.5: Average queue length of static traffic signal control for low-density

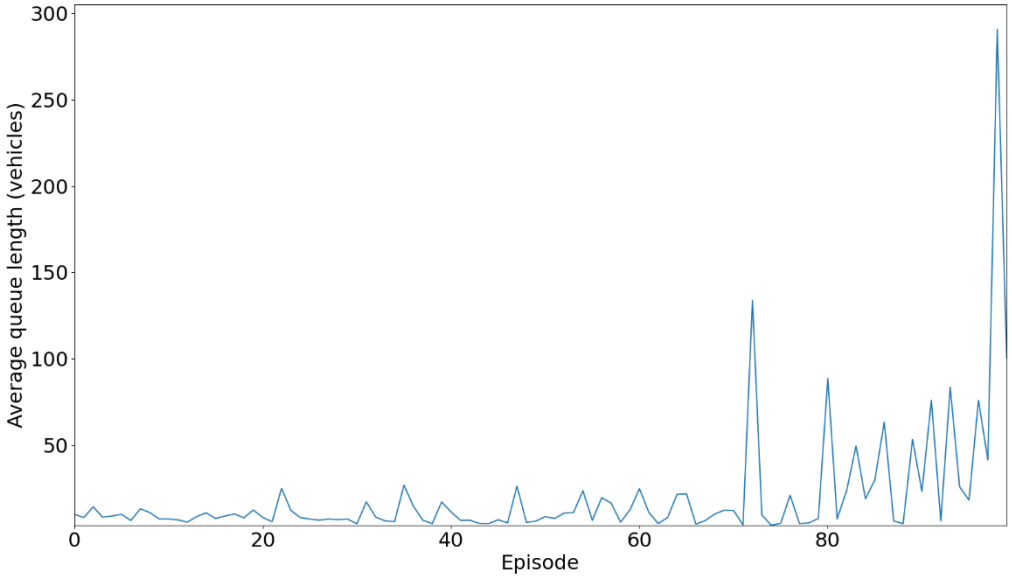

Figure 7.6: average queue of dynamic traffic signal control for low-density

7.1.3. Moderate-Density

7.1.3.1. Negative Reward

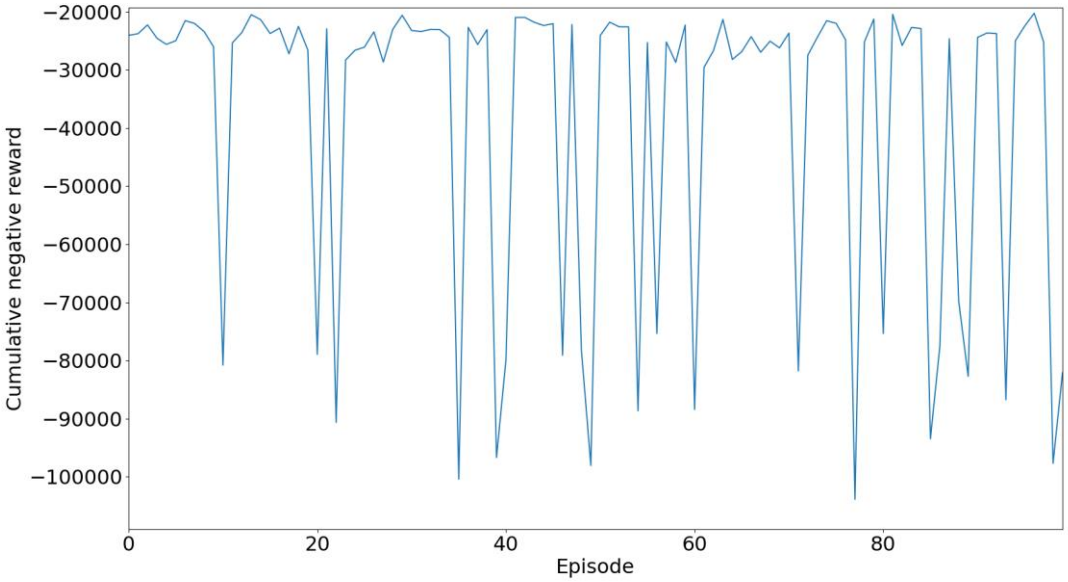

Figure 7.7: Negative reward of the static traffic signal control for moderate-density

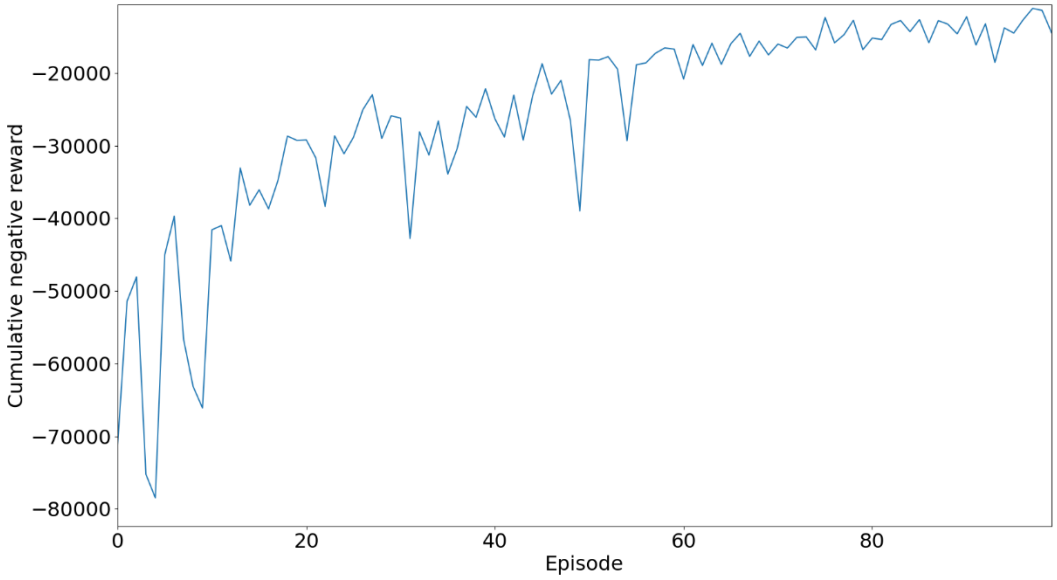

Figure 7.8: Negative reward of the dynamic traffic signal control for moderate-density

7.1.3.2. Cumulative Waiting Time

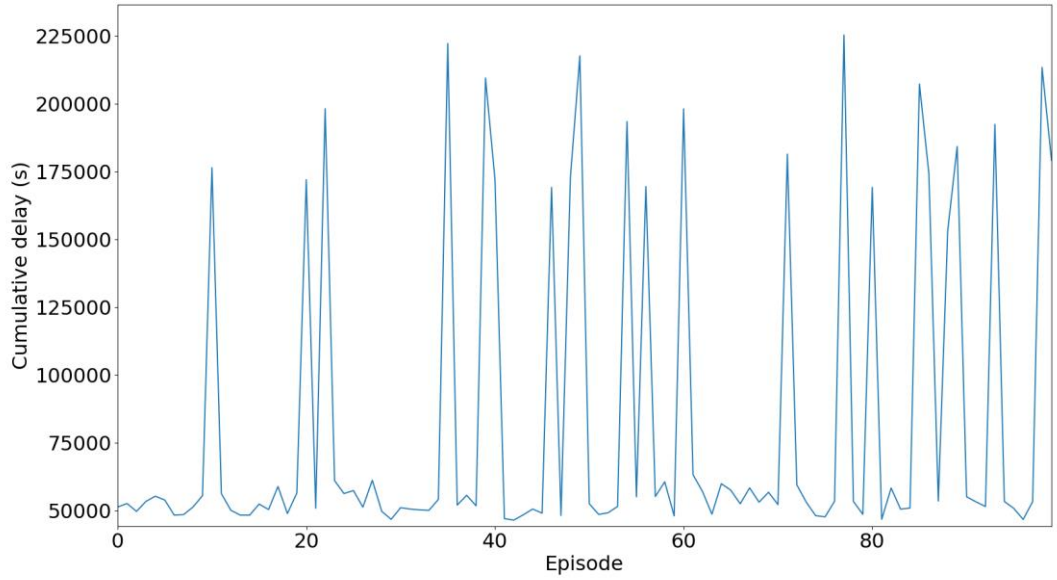

Figure 7.9: Cumulative Delay of static traffic signal control for moderate-density

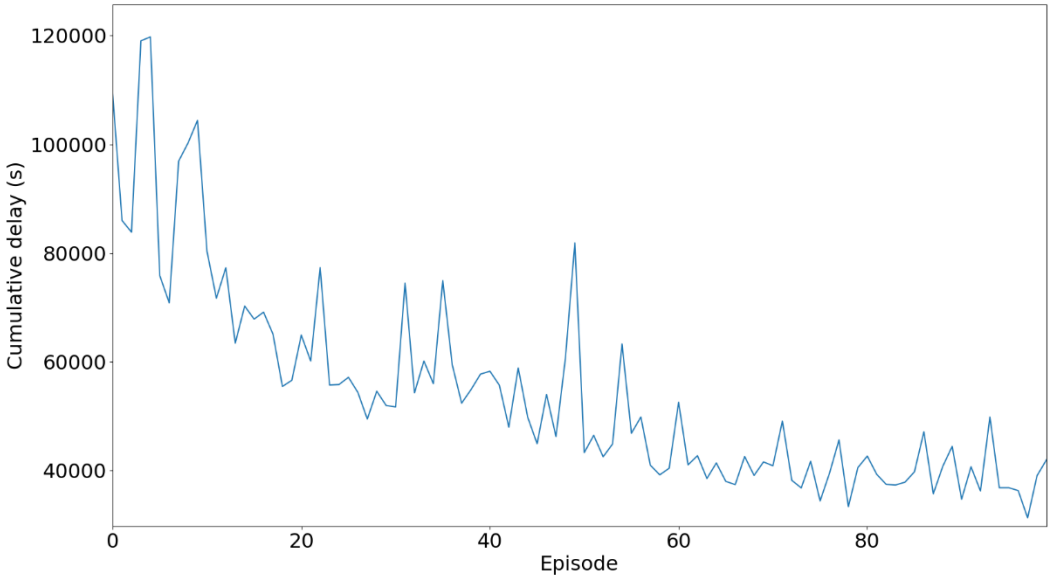

Figure 7.10: Cumulative Delay of dynamic traffic signal control for moderate-density

7.1.3.3. Average queue length

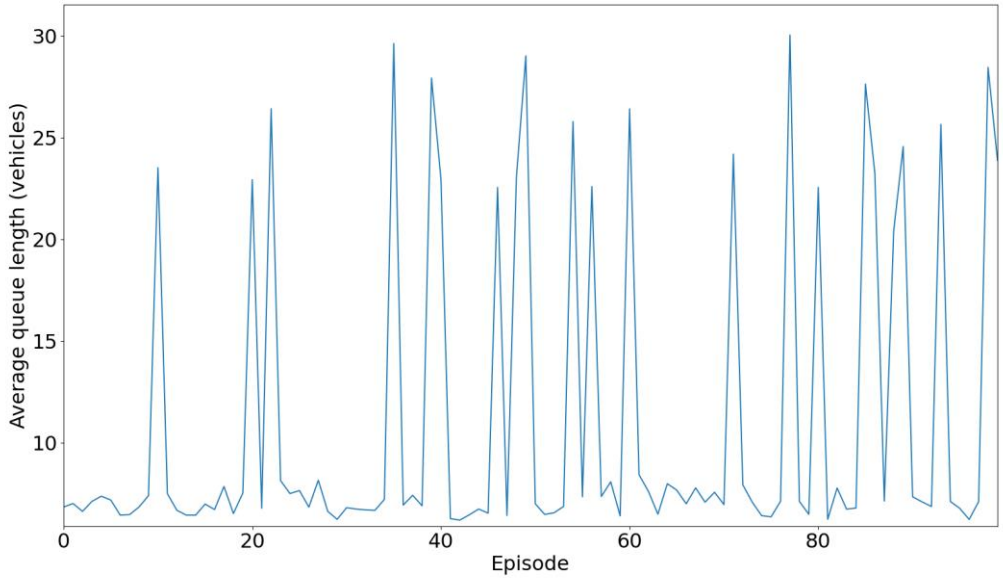

Figure 7.11: average queue length of static traffic signal control for moderate-density

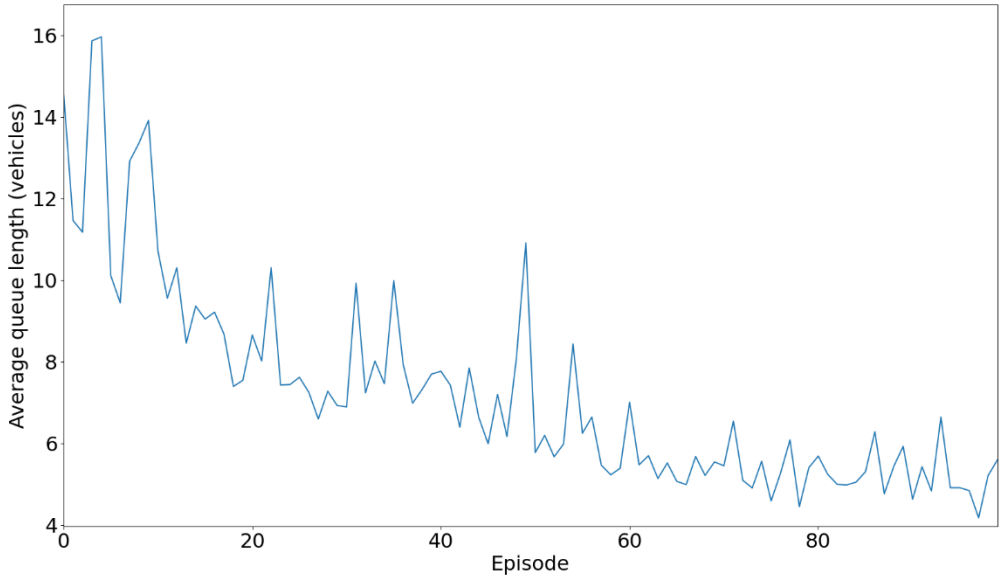

Figure 7.12: average queue length of dynamic traffic signal control for moderate-density

7.1.4. High-Density

7.1.4.1. Negative Reward

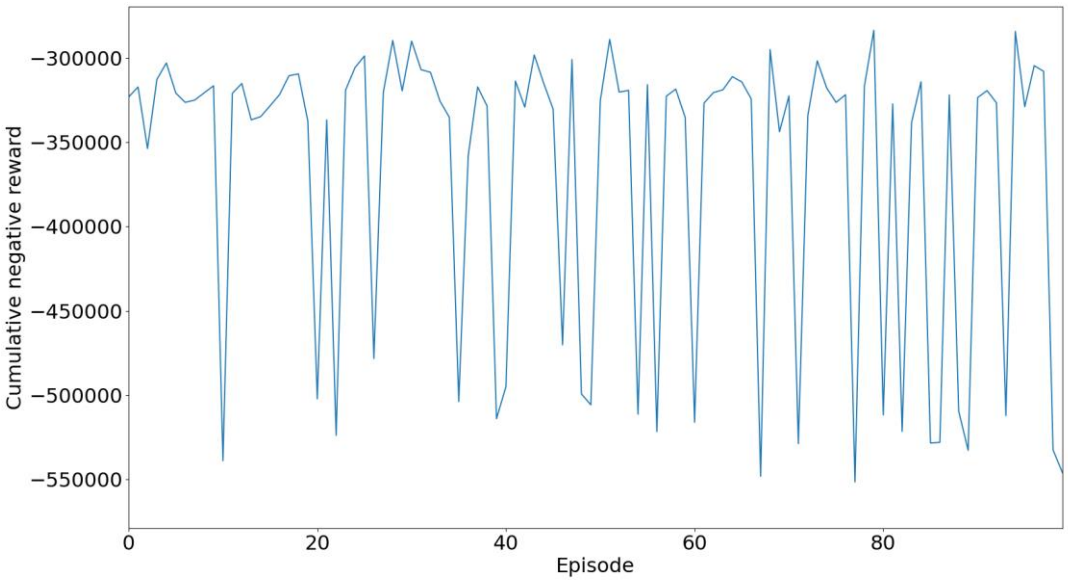

Figure 7.13: Negative reward of the static traffic signal control for high-density

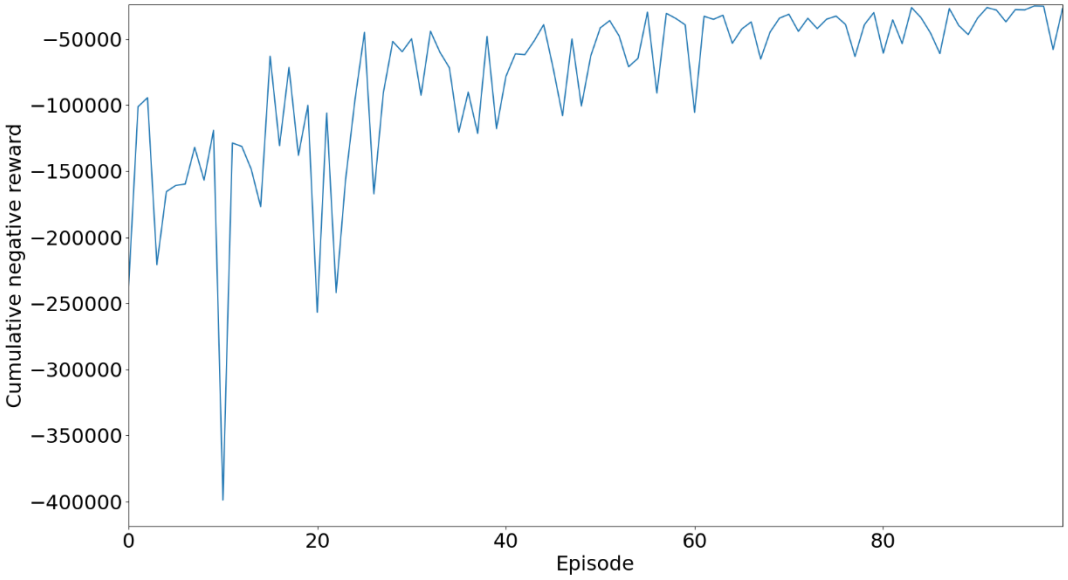

Figure 7.14: Negative reward of the dynamic traffic signal control for high-density

7.1.4.2. Cumulative Waiting Time

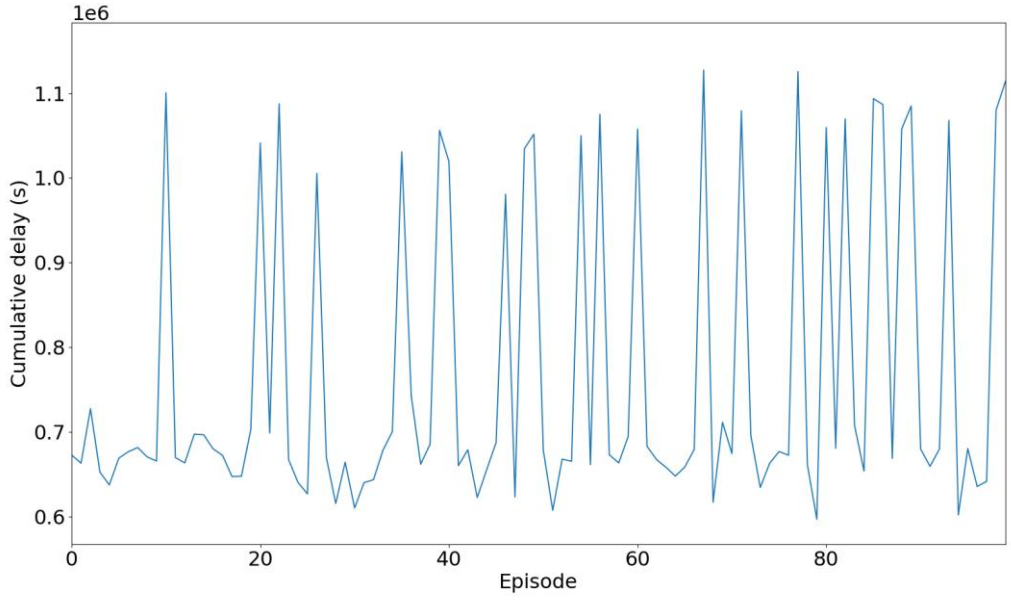

Figure 7.15: Cumulative Delay of static traffic signal control for high-density

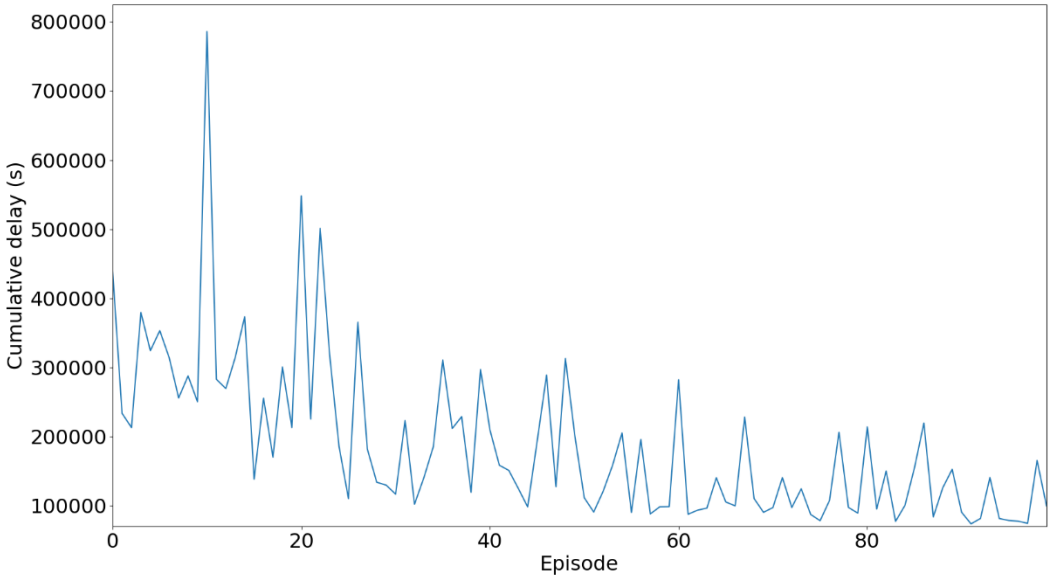

Figure 7.16: Cumulative Delay of dynamic traffic signal control for high-density

7.1.4.3. Average queue length

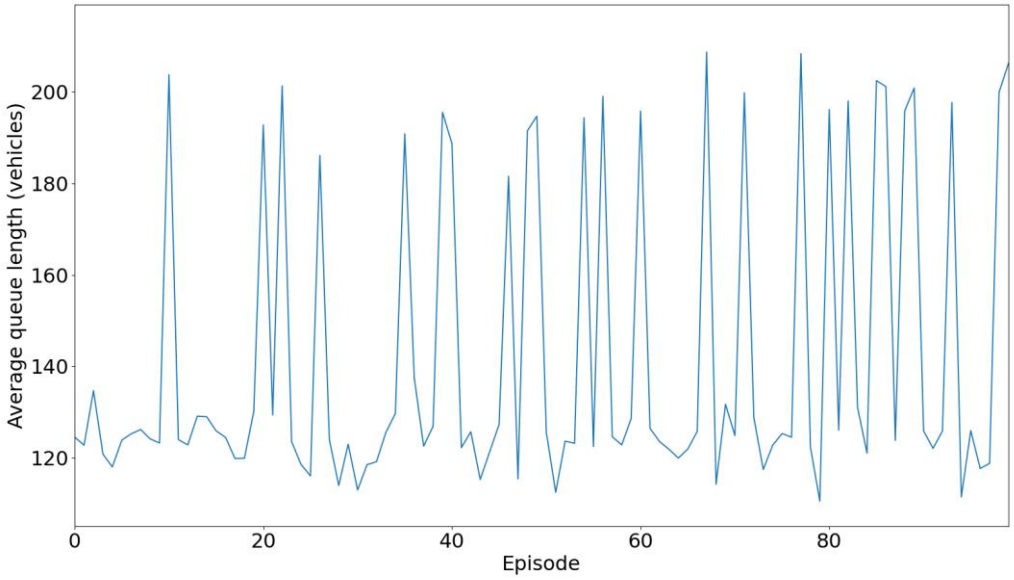

Figure 7.17: average queue length of static traffic signal control for high-density

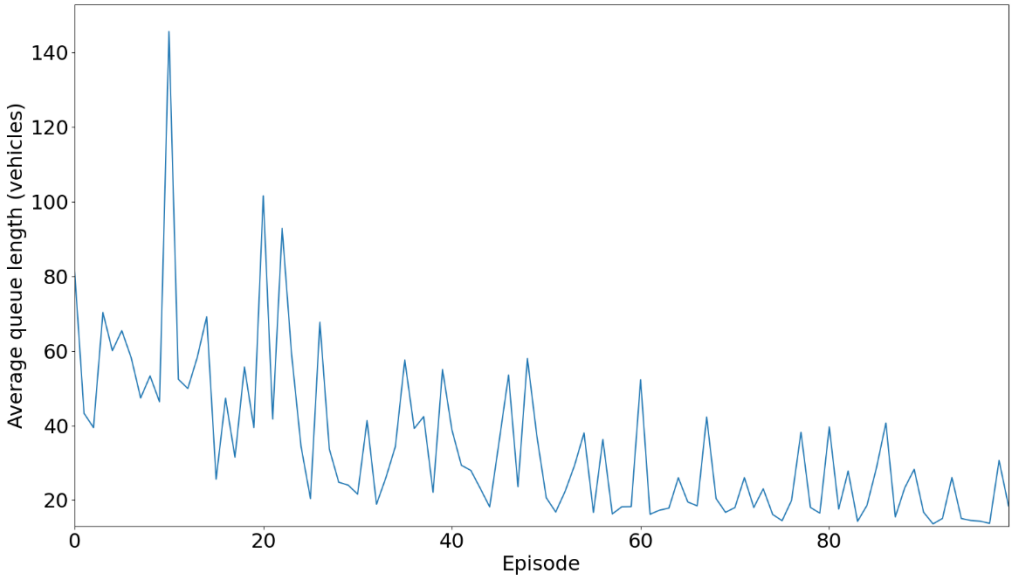

Figure 7.18: average queue length of dynamic traffic signal control for high-density

7.2. Multiple Intersection

Here is the result of training the spatial-temporal model and running the same 100 episodes of simulation for the static traffic control and the training of dynamic traffic signal control DQN models at each intersection to compare the results of the negative reward, cumulative waiting time (delay) and average queue length of each episode.

7.2.1. Predictive Model

After several modification to the Parameter of the models we found that the best result came when the batch size is equal to 5 and that the no. of prediction steps is equal to 1. This is on 100 episodes of simulation each contains 7500 timestep and a total of 5000 vehicles. The model had root mean square error (RMSE) of 2.0922. and mean absolute error (MAE) of 1.0553. this was the model we used for traffic prediction when training and running the DQN model.

7.2.2. Low-Density

7.2.2.1. Negative Reward

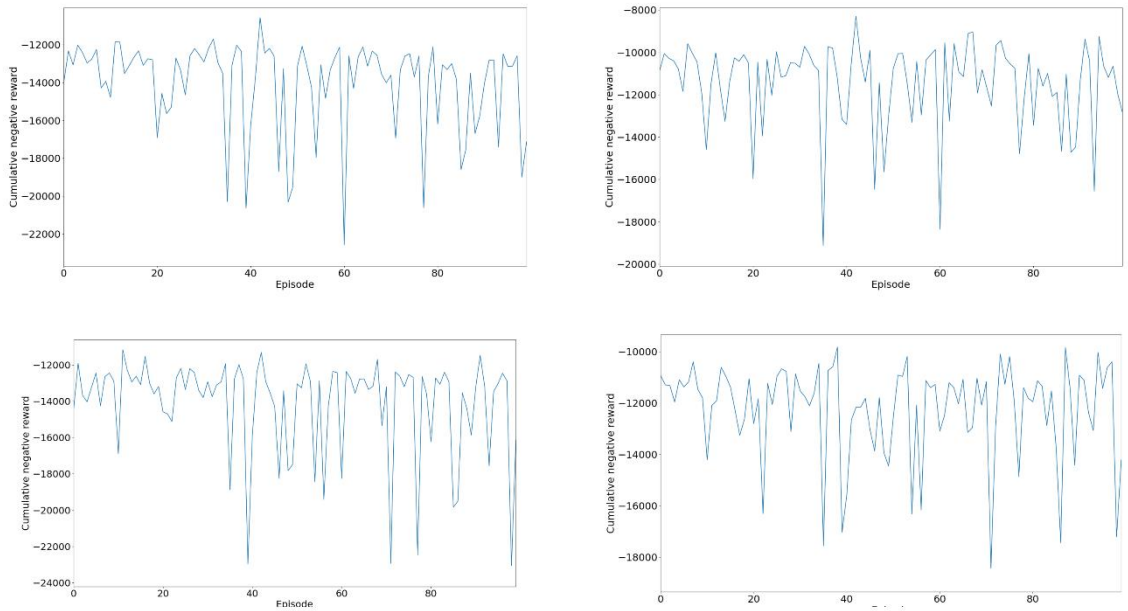

Figure 7.19: Negative reward of low-density static signal control for low-density (upper-left: TL1, upper-right:TL2, lower-left:TL3, lower-right: TL4)

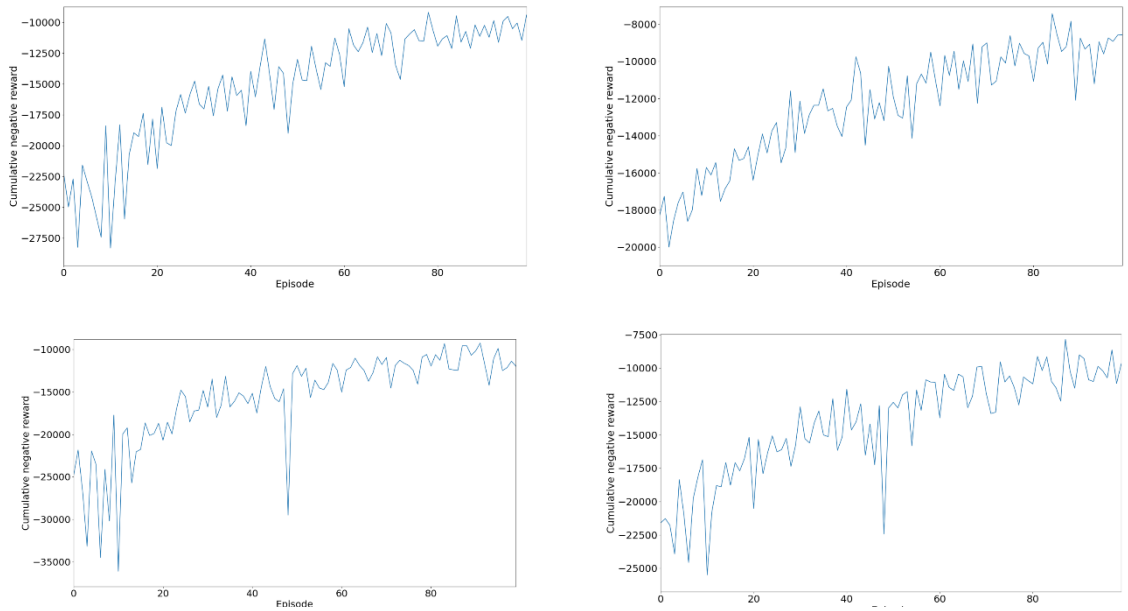

Figure 7.20: Negative reward of the dynamic traffic signal control for low-density (upper-left: TL1, upper-right:TL2, lower-left:TL3, lower-right: TL4)

7.2.2.2. Cumulative Waiting Time (Delay)

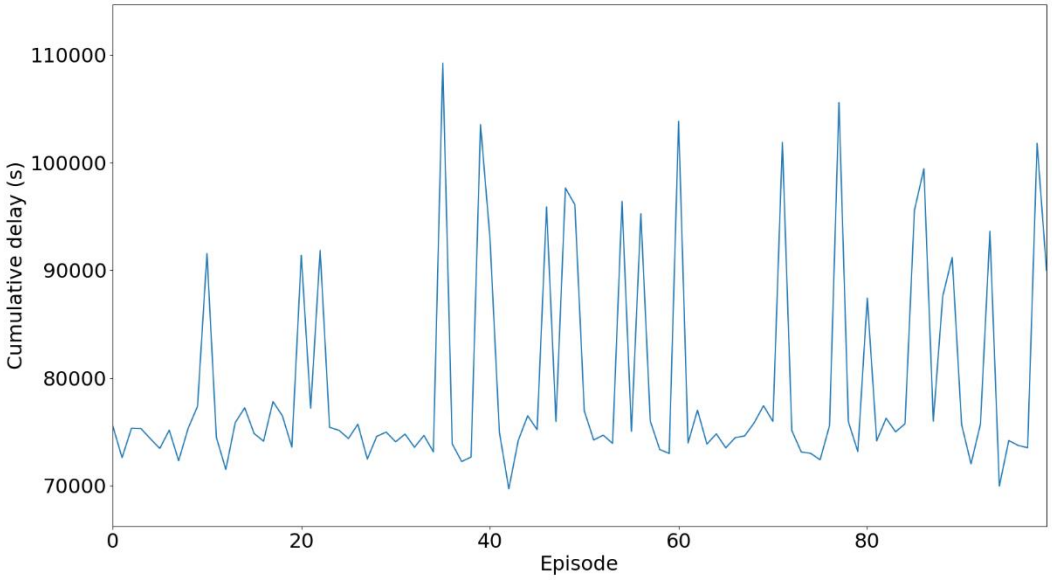

Figure 7.21: Cumulative Delay of static traffic signal control for low-density

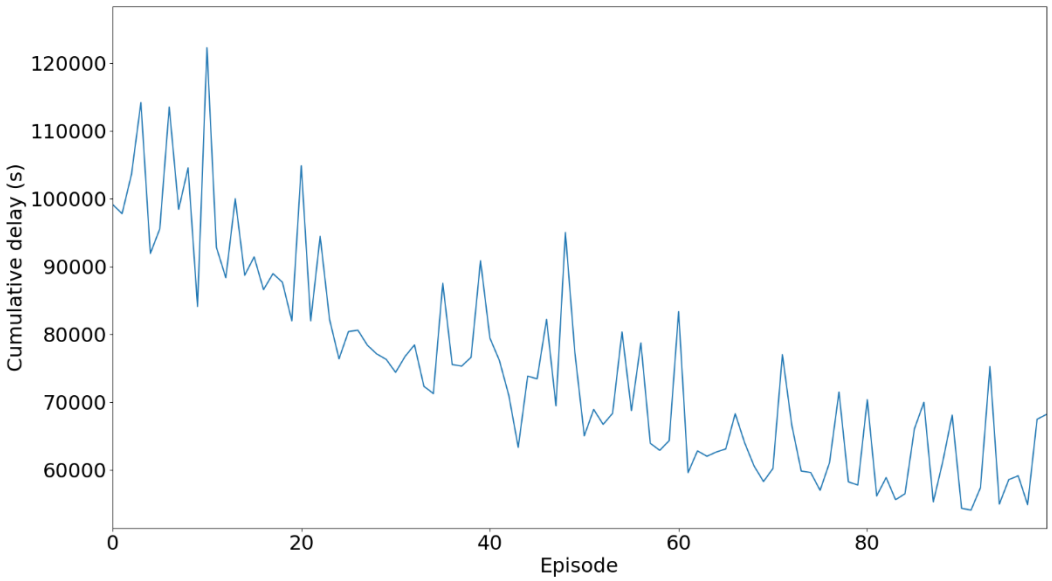

Figure 7.22:Cumulative Delay of dynamic traffic signal control for low-density

7.2.2.3. Average queue length

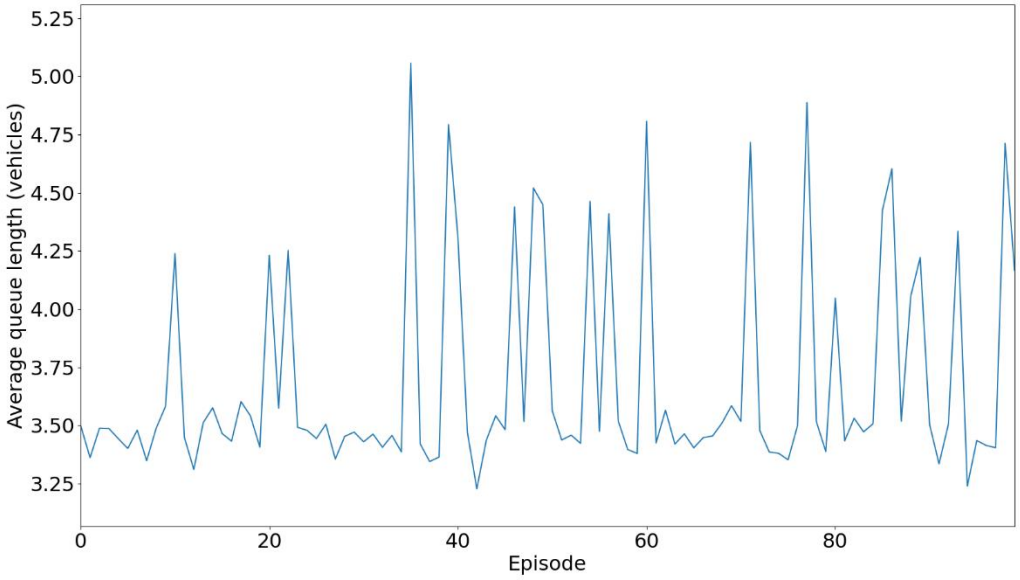

Figure 7.23: Average queue length of static traffic signal control for low-density

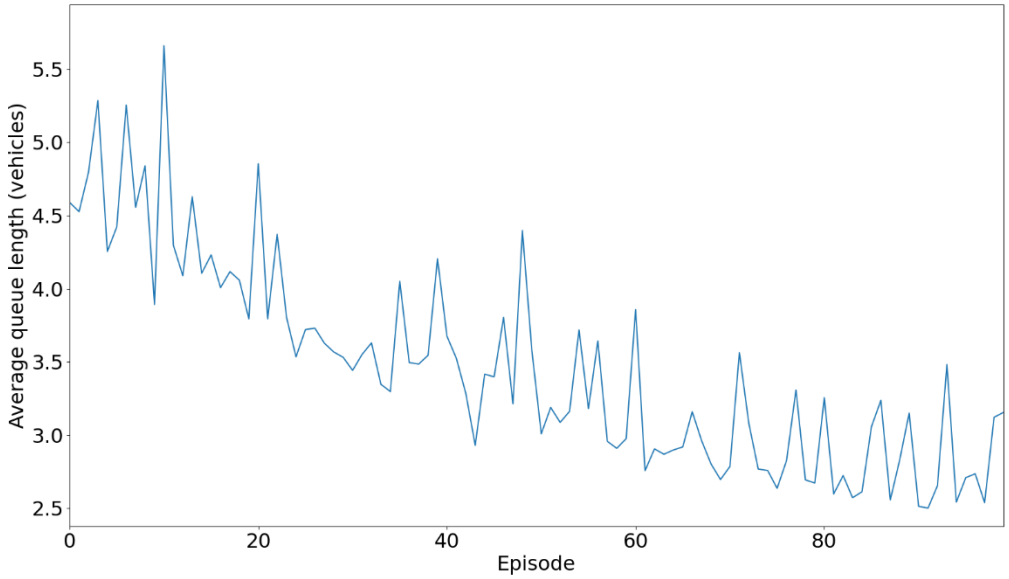

Figure 7.24: average queue length of dynamic traffic signal control for low-density

7.2.3. Moderate-Density

7.2.3.1. Negative Reward

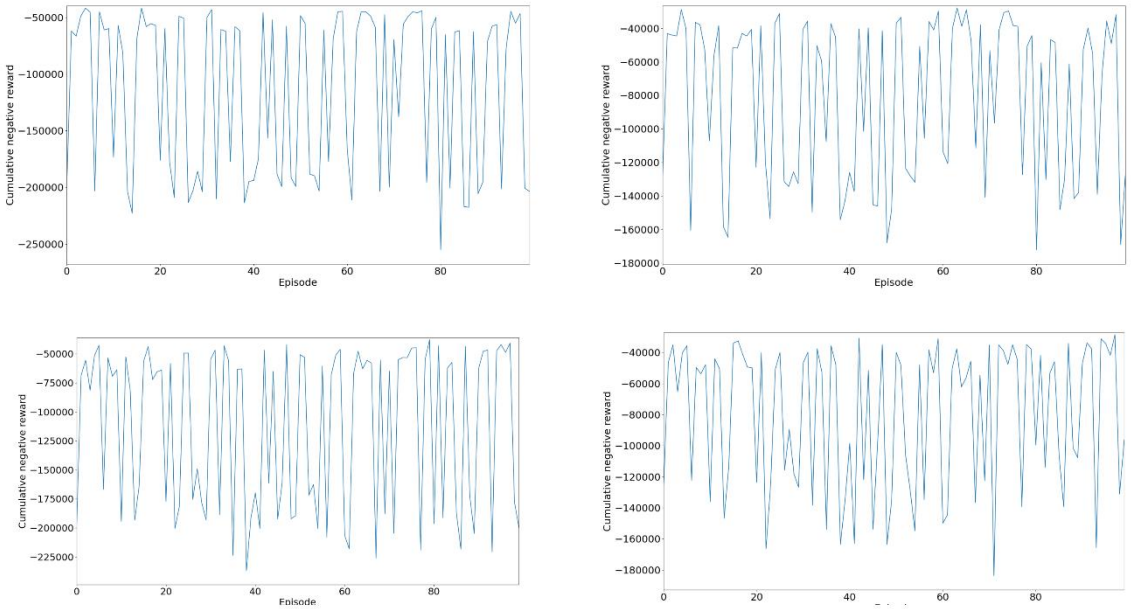

Figure 7.25: Negative reward of the static traffic signal control for moderate-density (upper-left: TL1, upper-right:TL2, lower-left:TL3, lower-right: TL4)

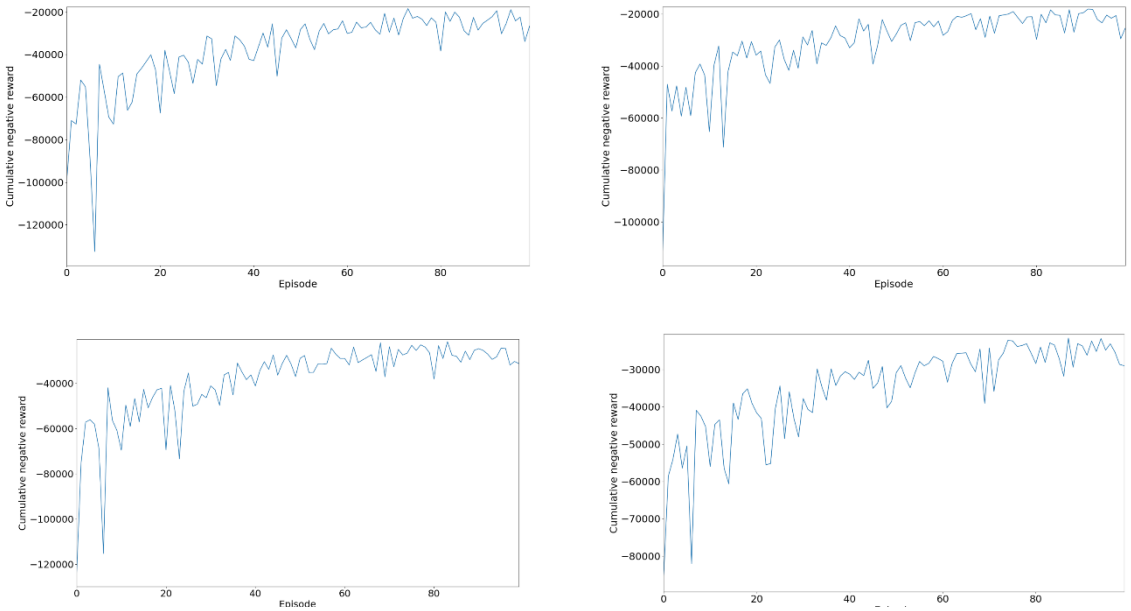

Figure 7.26: Negative reward of the dynamic traffic signal control for moderate-density (upper-left: TL1, upper-right:TL2, lower-left:TL3, lower-right: TL4)

7.2.3.2. Cumulative Waiting Time (Delay)

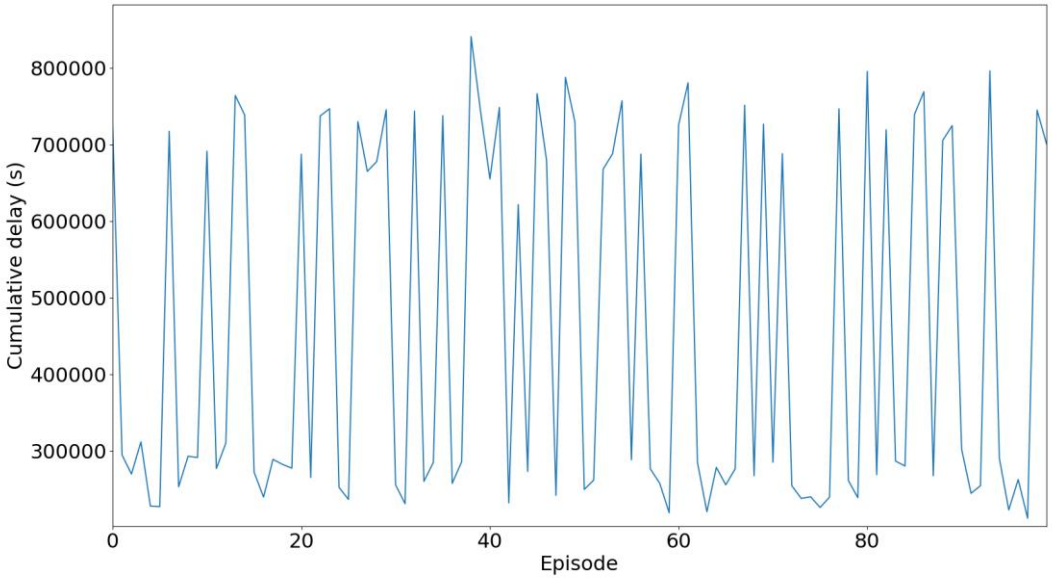

Figure 7.27: Cumulative Delay of static traffic signal control for moderate-density

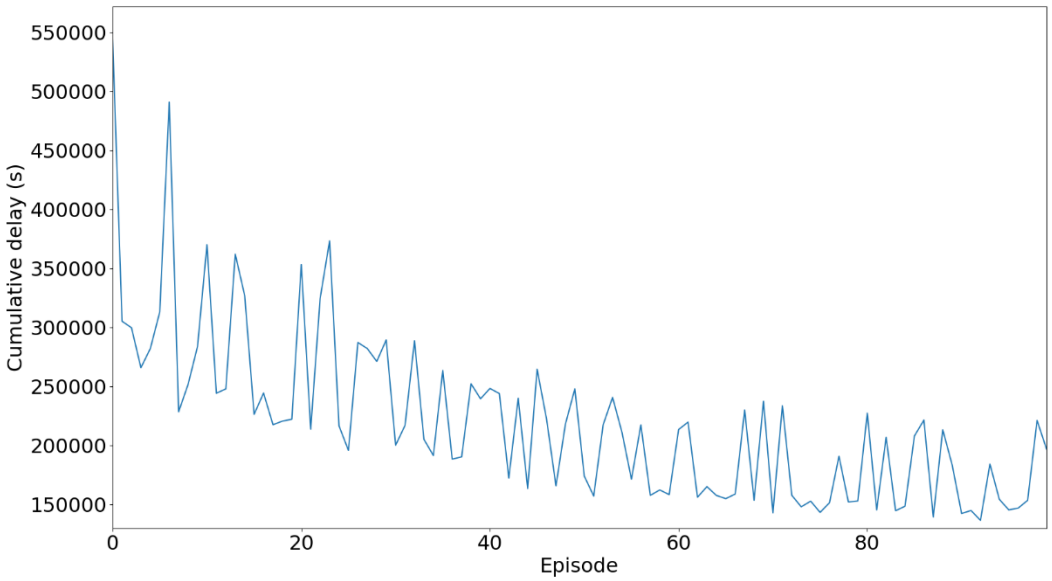

Figure 7.28: Cumulative Delay of dynamic traffic signal control for moderate-density

7.2.3.3. Average queue length

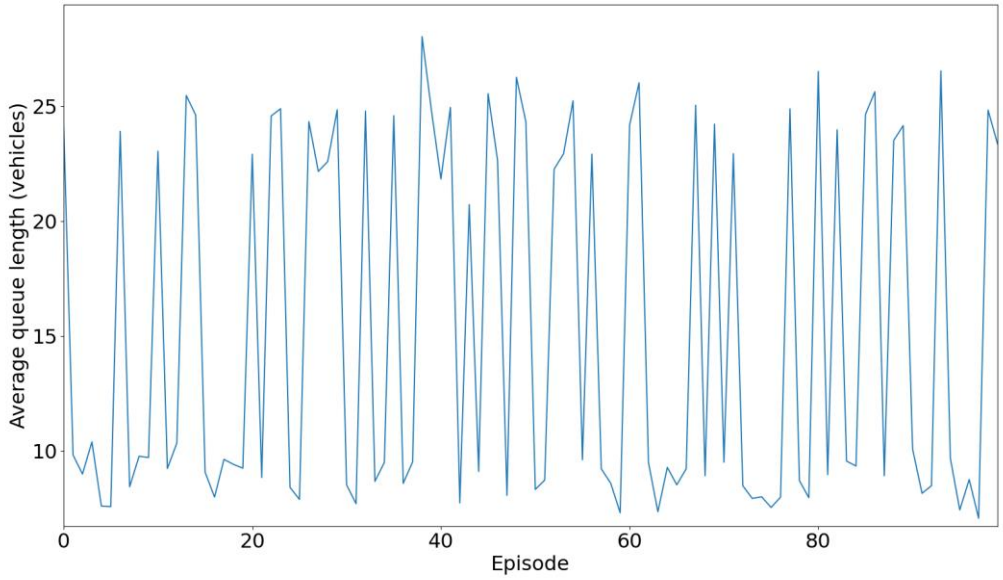

Figure 7.29: average queue length of static traffic signal control for moderate-density

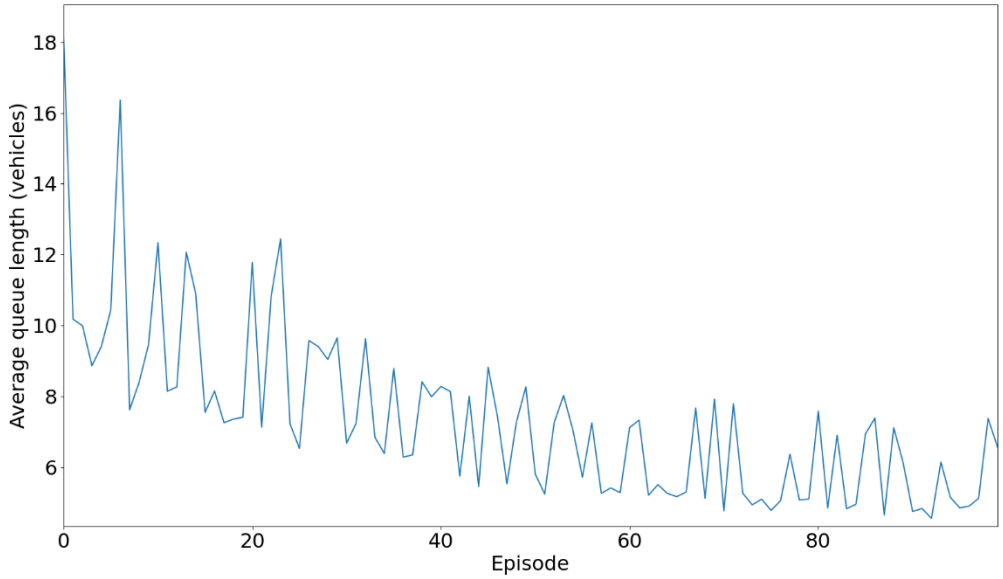

Figure 7.30: average queue length of dynamic traffic signal control for moderate-density

7.2.4. High-Density

7.2.4.1. Negative Reward

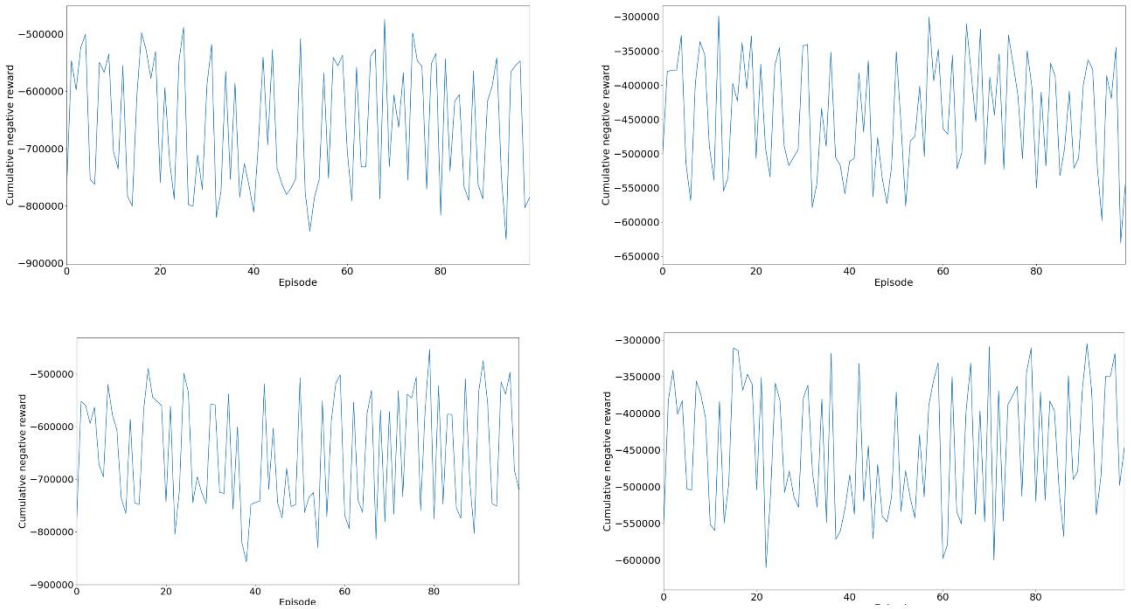

Figure 7.31: Negative reward of the static traffic signal control for high-density (upper-left: TL1, upper-right:TL2, lower-left:TL3, lower-right: TL4)

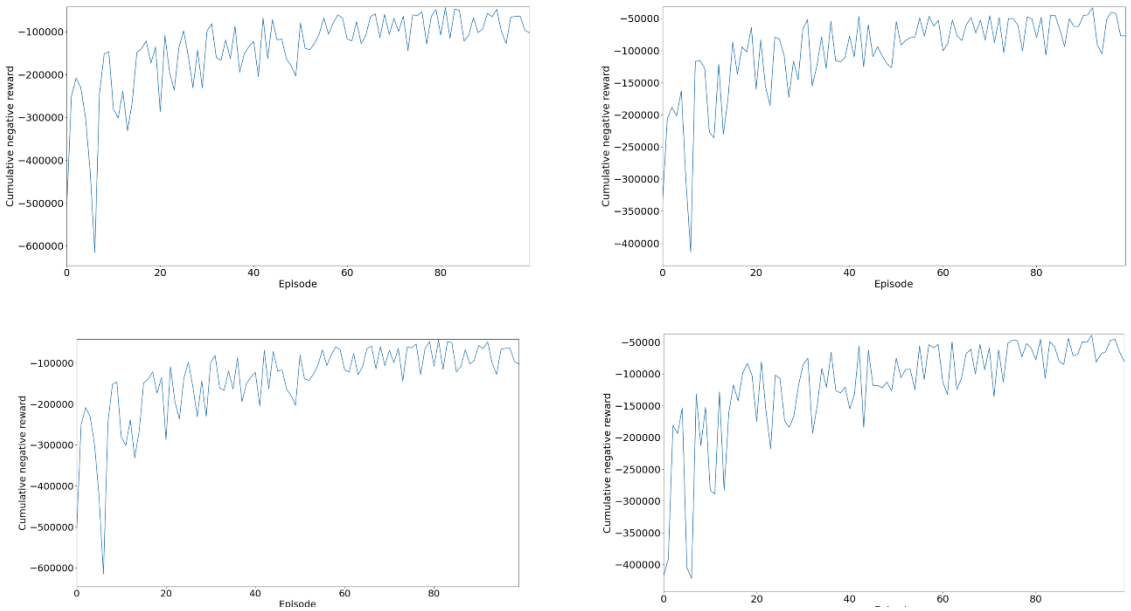

Figure 7.32: Negative reward of the dynamic traffic signal control for high-density (upper-left: TL1, upper-right:TL2, lower-left:TL3, lower-right: TL4)

7.2.4.2. Cumulative Waiting Time (Delay)

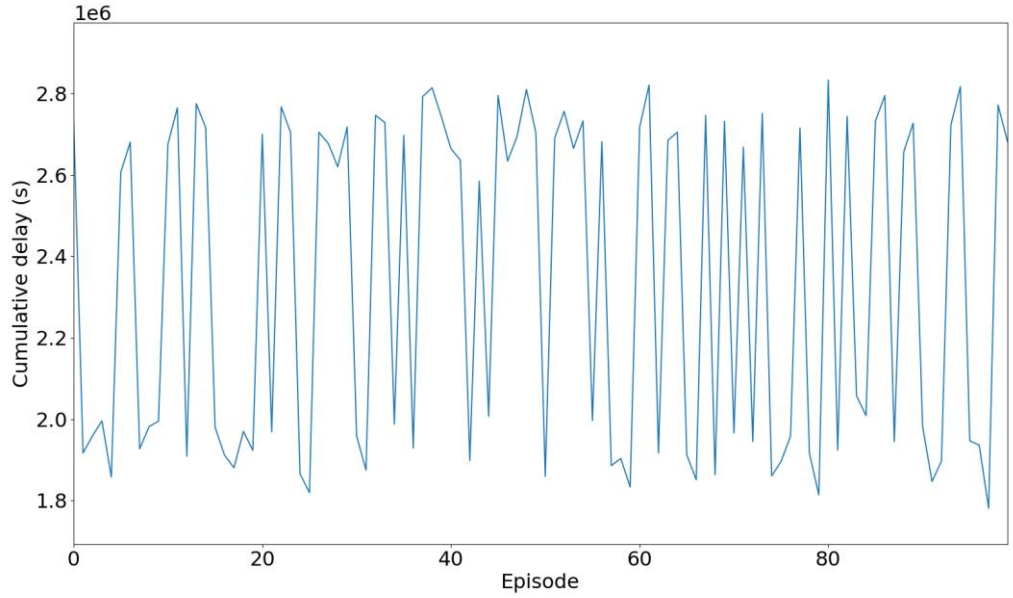

Figure 7.33: Cumulative Delay of static traffic signal control for high-density

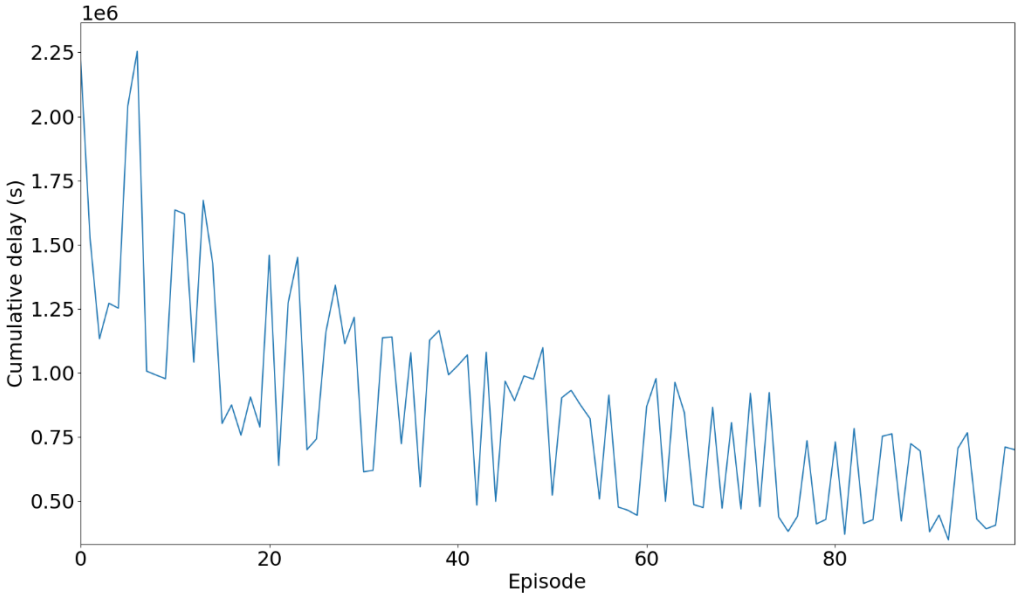

Figure 7.34: Cumulative Delay of dynamic traffic signal control for high-density

7.2.4.3. Average queue length

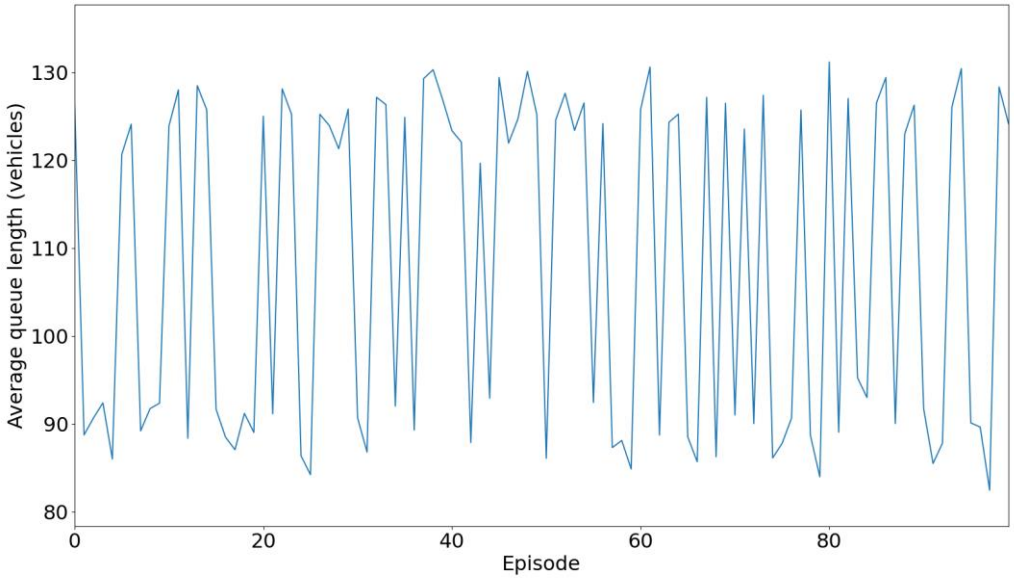

Figure 7.35: average queue length of static traffic signal control for high-density

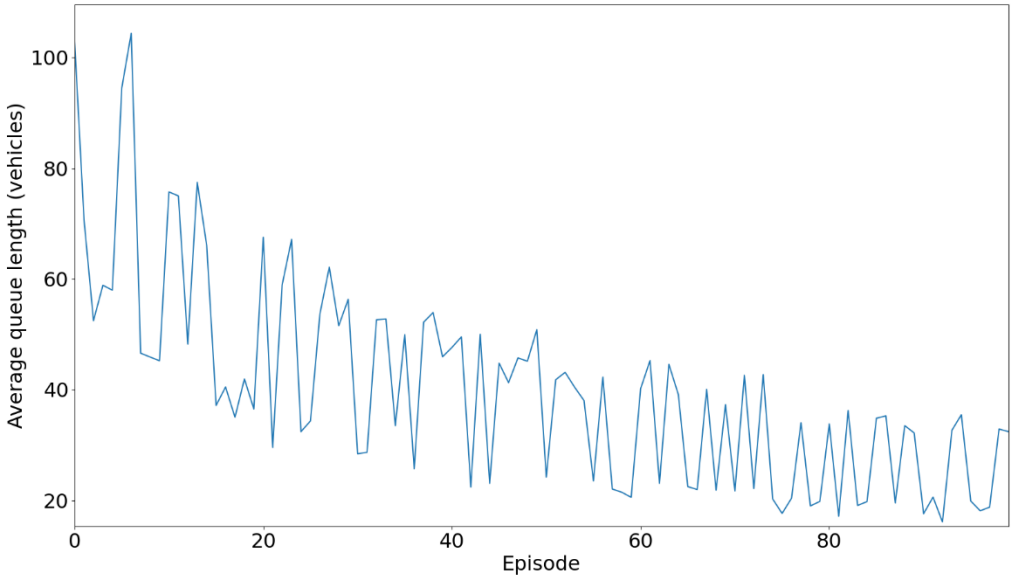

Figure 7.36: average queue length of dynamic traffic signal control for high-density

Chapter 8: Conclusion

From the simulation we ran on both the single and multiple intersection networks and based on the results we have shown before we have determined the following:

8.1. For Low Density

While The performance of dynamic signal control its better than the static is better but the difference in performance metrics isn't big indicating that for low density areas and small cities it might be less costly to use static control.

8.2. For moderate and high density

We can see that the performance of the dynamic traffic control using the combination of traffic prediction and the Deep Q-Learning far exceeds that of the normal static traffic control as indicated by the plots of the cumulative waiting time (delay) and the average queue length across the whole networks in both the single and multiple intersection networks which makes it a far efficient approach of traffic control.

Chapter 9: Future Works

In the future we need to run further simulation on multiple networks from the real world then adding more features that confronts other aspects of the traffic management including dynamic routing of vehicles and reaction to emergencies to further improves traffic flow and reduce traffic congestion in smart cities.

Chapter 10: References

[1] “Traffic congestion cost the US economy nearly $87 billion in 2018,” Weforum.org. [Online]. Available: https://www.weforum.org/agenda/2019/03/traffic-congestion-cost-the-us-economy-nearly-87-billion-in-2018/. [Accessed: 08-Apr-2021].

[2] “Time is Money, especially on Cairo’s Streets,” Worldbank.org. [Online]. Available: https://blogs.worldbank.org/arabvoices/time-money-especially-cairo-streets. [Accessed: 08-Apr-2021].

[3] T. Ito and R. Kaneyasu, “Predicting traffic congestion using driver behavior,” Procedia Comput. Sci., vol. 112, pp. 1288–1297, 2017.

[4] S. Javaid, A. Sufian, S. Tanveer, "Smart traffic management system using Internet of Things," 2018 20th International Conference on Advanced Communication Technology (ICACT), Chuncheon, Korea (South), 2018, pp. 393-398, doi: 10.23919/ICACT.2018.8323770.

[5] L. Cheng, Y. Li, Y. Wang, Y. Bi, L. Feng, and M. Xue, “A triple-filter NLOS localization algorithm based on Fuzzy C-means for wireless sensor networks,” Sensors (Basel), vol. 19, no. 5, p. 1215, 2019.

[6] Y. Bi, X. Lu, Z. Sun, D. Srinivasan and Z. Sun, "Optimal Type-2 Fuzzy System For Arterial Traffic Signal Control," in IEEE Transactions on Intelligent Transportation Systems, vol. 19, no. 9, pp. 3009-3027, Sept. 2018, doi: 10.1109/TITS.2017.2762085.

[7] M. J. Shirvani and H. R. Maleki, "Maximum green time settings for traffic actuated signal control at isolated intersections using fuzzy logic," 2015 4th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS), Zahedan, Iran, 2015, pp. 1-5, doi: 10.1109/CFIS.2015.7391699.

[8] G. Zheng et al., “Learning phase competition for traffic signal control,” arXiv [cs.LG], 2019.

[9] H. Wei et al., “PressLight: Learning max pressure control to coordinate traffic signals in arterial network,” in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD ’19, 2019.

[10] H. Wei et al., “CoLight: Learning network-level cooperation for traffic signal control,” arXiv [cs.MA], 2019.

[11] V. Mnih et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[12] W. Genders and S. Razavi, “Using a deep reinforcement learning agent for traffic signal control,” arXiv [cs.LG], 2016.

[13] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with Double Q-learning,” arXiv [cs.LG], pp. 2094–2100, 2015.

[14] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, “Dueling network architectures for deep reinforcement learning,” arXiv [cs.LG], 2015.

[15] M. Hessel et al., “Rainbow: Combining improvements in deep reinforcement learning,” arXiv [cs.AI], 2017.

[16] H. Ge, Y. Song, C. Wu, J. Ren and G. Tan, "Cooperative Deep Q-Learning With Q-Value Transfer for Multi-Intersection Signal Control," in IEEE Access, vol. 7, pp. 40797-40809, 2019, doi: 10.1109/ACCESS.2019.2907618

[17] C. Ozan, O. Baskan, S. Haldenbilen, and H. Ceylan, “A modified reinforcement learning algorithm for solving coordinated signalized networks,” Transp. Res. Part C Emerg. Technol., vol. 54, pp. 40–55, 2015.

[18] S. Sun, J. Chen, and J. Sun, “Traffic congestion prediction based on GPS trajectory data,” Int. J. Distrib. Sens. Netw., vol. 15, no. 5, p. 155014771984744, 2019.

[19] D. Kim and O. Jeong, “Cooperative traffic signal control with traffic flow prediction in multi-intersection,” Sensors (Basel), vol. 20, no. 1, p. 137, 2019.

[20] Z. Cui, K. Henrickson, R. Ke, and Y. Wang, “Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting,” IEEE Trans. Intell. Transp. Syst., vol. 21, no. 11, pp. 4883–4894, 2020.

[21] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, “Attention based spatial-temporal graph convolutional networks for traffic flow forecasting,” Proc. Conf. AAAI Artif. Intell., vol. 33, no. 01, pp. 922–929, 2019.

[22] K. Guo et al., “Optimized graph convolution recurrent neural network for traffic prediction,” IEEE Trans. Intell. Transp. Syst., vol. 22, no. 2, pp. 1138–1149, 2021.

[23] Y. Zhang, S. Wang, B. Chen, and J. Cao, “GCGAN: Generative adversarial nets with graph CNN for network-scale traffic prediction,” in 2019 International Joint Conference on Neural Networks (IJCNN), 2019, pp. 1–8.

[24] Z. Diao, X. Wang, D. Zhang, Y. Liu, K. Xie, and S. He, “Dynamic spatial-temporal graph convolutional neural networks for traffic forecasting,” Proc. Conf. AAAI Artif. Intell., vol. 33, no. 01, pp. 890–897, 2019.

[25] C. Chen et al., “Gated residual recurrent graph neural networks for traffic prediction,” Proc. Conf. AAAI Artif. Intell., vol. 33, no. 01, pp. 485–492, 2019.

[26] C. Jia, B. Wu, and X.-P. Zhang, “Dynamic spatiotemporal graph neural network with tensor network,” arXiv [cs.CV], 2020.

[27] F. Diehl, T. Brunner, M. T. Le, and A. Knoll, “Graph neural networks for modelling traffic participant interaction,” arXiv [cs.LG], 2019.

[28] D. Xu, H. Dai, Y. Wang, P. Peng, Q. Xuan, and H. Guo, “Road traffic state prediction based on a graph embedding recurrent neural network under the SCATS,” Chaos, vol. 29, no. 10, p. 103125, 2019.

[29] X. Wang, C. Chen, Y. Min, J. He, B. Yang, and Y. Zhang, “Efficient metropolitan traffic prediction based on Graph Recurrent Neural Network,” arXiv [cs.AI], 2018.

[30] L. Zhao et al., “T-GCN: A temporal graph ConvolutionalNetwork for traffic prediction,” arXiv [cs.LG], 2018.

[31] L. Ge, H. Li, J. Liu, and A. Zhou, “Temporal graph convolutional networks for traffic speed prediction considering external factors,” in 2019 20th IEEE International Conference on Mobile Data Management (MDM), 2019, pp. 234–242.

[32] F. Zhou, Q. Yang, K. Zhang, G. Trajcevski, T. Zhong, and A. Khokhar, “Reinforced spatiotemporal attentive graph neural networks for traffic forecasting,” IEEE Internet Things J., vol. 7, no. 7, pp. 6414–6428, 2020.

[33] B. Yu, H. Yin, and Z. Zhu, “Spatio-Temporal Graph Convolutional Networks: A deep learning framework for traffic forecasting,” arXiv [cs.LG], 2017.

[34] X. Hu, C. Zhao, and G. Wang, “A traffic light dynamic control algorithm with deep reinforcement learning based on GNN prediction,” arXiv [cs.LG], 2020