Short term traffic prediction using local information.

Improving graph-based time series prediction using local information flows.

by

R.J. Dijkhuizen

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday January 27, 2025 at 13:45.

Student number: 4866177
Project duration: June 3, 2024 – January 27, 2025
Thesis committee:

dr. C. Kraaikamp,	TU Delft, chair
dr. M. Vittorietti,	TU Delft, supervisor
Ir. L. Krol,	Goudappel B.V, supervisor

TU Delft

Abstract

This thesis presents localized methods for traffic prediction and analysis. The prediction method presents an extension of a state-of-the-art Graph Neural Network inspired by traffic flow characteristics on a local level. This inspiration from traffic flow characteristics consists of two parts. The first intuition is that state at some location and at time T in the future will be not influenced by information which is further than traveling time T away. The second intuition is that traffic information traveling with or against the stream of traffic behaves differently. The developed model leverages these intuitions to increase model prediction performance. Further, a modification is made which allows the model to be applied at an arbitrary location in a network, at the cost of performance. Alongside these model extensions, a novel method of visualizing a local traffic state is presented through constructing a novel traveltime diagram. This diagram can be used as a visual tool for analyzing traffic locally. Further, the traveltime diagram is designed to be summarized using Topological Data Analysis to a quantity called the Travel Lifetime which can represent traffic states ranging from extremely calm to imminent congestion to a congested state in a single number. The newly proposed Travel Lifetime is tested as an input to a Neural Network model for predicting traffic speed showing that its use as an input can improve model performance.

Contents

1 Introduction **1**
1.1 Problem definition and context . . . 1
1.2 Content overview . . . 3
1.3 Methodological choices . . . 3
1.4 Contributions . . . 4
1.5 Project structure . . . 5

2 Existing methods **7**
2.1 Data . . . 7
2.2 Traffic characteristics . . . 7
2.3 ML time series modeling . . . 8
2.3.1 Time series . . . 8
2.3.2 Working with ML . . . 9
2.3.3 Neural networks . . . 10
2.3.4 Recurrent neural network . . . 11
2.3.5 Graph neural networks . . . 12
2.3.6 SpaceTime Neural Network . . . 12
2.3.7 Random forest . . . 13
2.4 Topological Data Analysis . . . 15
2.4.1 Simplicial complexes . . . 16
2.4.2 Homotopy . . . 17
2.4.3 Homology . . . 18
2.4.4 Persistent homology . . . 19
2.4.5 Previous work on TDA for time series analysis . . . 21
2.5 Software . . . 22

3 Developed methods **23**
3.1 Forecasting methods . . . 23
3.1.1 Direction intuition . . . 23
3.1.2 Locality intuition . . . 24
3.1.3 SP-STNN . . . 24
3.1.4 Distance based GNN . . . 24
3.2 Traffic visualization and summarization . . . 26
3.2.1 Traveltime diagram . . . 26
3.2.2 TDA transformation . . . 27
3.2.3 Generalization to merging paths . . . 28

4 Results **33**
4.1 Preliminary study . . . 33
4.1.1 Submodels . . . 33
4.1.2 Results . . . 34
4.2 Extending STNN . . . 35
4.2.1 Comparison with preliminary work . . . 35
4.2.2 Larger datasets . . . 36
4.2.3 SP-STNN . . . 37
4.2.4 Running time . . . 38
4.3 Previous work on TDA for time series analysis . . . 38
4.4 Travel lifetime . . . 39
4.4.1 Summarization value . . . 40

5 Discussion and conclusion **43**
5.1 Discussion 43
5.2 Conclusions 45
5.3 Closing points 46

A Appendix **53**

The following table describes the meaning of various abbreviations and acronyms used throughout the thesis.

Abbreviation	Meaning	First mention	Explanation
ML	Machine Learning	1	9
(A)NN	(Artificial) Neural Network	3	10
CNN	Convolutional Neural Network	3	3
GNN	Graph Neural Network	3	12
TDA	Topological Data Analysis	3	15
MVVS	Multimodaal VerkeersVoorspellend Systeem (Multimodal Traffic Predicting System)	5	5
DEXTER	Data EXploraTion and ExporteR	7	7
FCD	Floating Car Data	7	7
NDW	Nationaal Dataportaal Wegverkeer (National Dataportal Road traffic)	7	7
MAE	Mean Absolute Error	8	9
MSE	Mean Squared Error	8	9
ReLU	Rectified Linear Unit	10	10
GRU	Gated Recurrent Unit	11	11
RNN	Recurrent Neural Network	11	11
STNN	SpaceTime Neural Network	12	12
P-STNN	Propagation SpaceTime Neural Network	23	23
SP-STNN	Selfless Propagation SpaceTime Neural Network	23	24
LOOCV	Leave-One-Out Cross Validation	36	36

1 Introduction

This thesis is aimed at improving methods for short-term traffic prediction and analysis. The prediction will be based on data-driven, Machine Learning (ML) methods, whereas the analysis will utilize an emerging mathematical method. This chapter introduces the problem and its context and sheds some light on the choices required for tackling the problem, as will be done in later chapters.

1.1. Problem definition and context

Imagine yourself in traffic not moving at all, just in front of a tunnel. For safety reasons, in case fire breaks out for example, it is better to stand still before the tunnel than inside it. This is also the reason you are at a complete standstill before the tunnel; the road operator has closed the tunnel because they deemed congestion inside the tunnel a possibility. Let's assume in this hypothetical situation that they deemed this a possibility because of congestion occurring at the tunnel exit. If you had known that congestion after the tunnel was likely just 5 minutes ago, then you could have taken another route, reducing your travel time.

The hypothetical situation above is a real-world [1] example of why predicting traffic could be useful. Traffic prediction is a problem with multiple scopes. In general, two broad scopes of traffic prediction are distinguished. The first is long-term prediction, for which long-term trends are of interest. The second is the previously mentioned short-term prediction, for which the deviations from usual situations are of interest. Long-term prediction has its uses in a strategic setting, for example, in deciding where to build new roads to handle traffic that one could expect to occur in the next 5-30 years.

Short-term predictions are the scope of this thesis and their use case is mostly in dynamic routing advice. Currently, road operators mostly work with a *"wait and see"* method, responding to events once they occur [2]. The example above: Once congestion occurs near a tunnel, the road is closed to prevent traffic from standing still in a tunnel. Dynamic routing advice may thus take shape as road closures to prevent congestion in sensitive areas, or it could be used to improve navigation services which find the fastest route taking traffic into account.

This thesis is a project to obtain a degree in mathematics and is a joint project with a mobility consultancy company, this joint nature is reflected in existing research on the topic of (short-term) traffic prediction. The problem of short-term traffic prediction finds itself being of interest to two groups of researchers, with a different motivation behind the same goal of the best prediction accuracy. The two groups could be identified as traffic researchers (i.e. civil engineers) and data scientists (i.e. mathematicians).

A traffic scientist, in this context, is focused on traffic prediction. As a result of the research background, the methods of study are strongly based on understanding complex factors which explain traffic behavior such as routing choices [3]. There, complex factors are used in algorithmically simpler models. This method guarantees some desirable properties of its output. As an example, this can guarantee the conservation of the number of vehicles in a system; if at time 0 there are 10 cars in a village and the method predicts 4 entering and 3 leaving, then there will be 11 cars in the prediction. Such a property

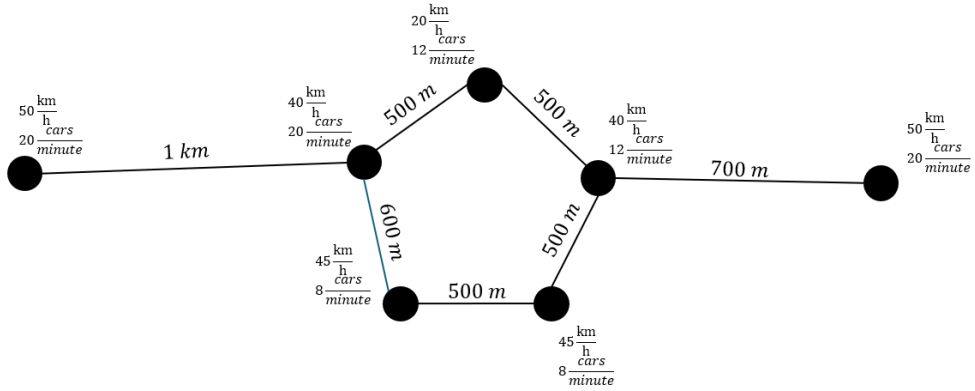


Figure 1.1: Traffic state on a graph, edge properties are shown as the distance between two locations. Node properties are the measured traffic speed and flow.

is desirable because it will follow the reasoning the traffic scientist expects to see, also allowing them to easily identify if and where things are going wrong when applying a model. On the other side of the coin; these models are typically quite slow in calculating a prediction. Also, these models are known to flip-flop between traffic moving at the speed limit and moving slowly, for example near traffic lights.

A data scientist, in this context, has developing a method for general time series prediction as motivation, which can be indicated by good traffic prediction. Common methods for time series prediction were typically ARIMA and GARCH, which were also applied to traffic problems [4, 5]. This thesis will use a more modern method: (graph-based) neural networks. Basic neural networks have existed for over 50 years [6], but their use and development have surged in the last years because of modern data collection and computation power [7]. Traffic prediction can be seen as a multivariate time-series prediction problem where the variables are related according to some graph structure i.e. the road network. To explain this a bit further, on a road network we measure how many cars pass and how fast they drive. This can mathematically be represented with a graph with varying combinations of node and edge attributes. In this thesis we use a graph representation which has node attributes: measured speed and flow (vehicles per hour), and edge attribute: length or distance between two nodes (see figure 1.1). In this sense, traffic is perhaps one of the most intuitive representations of time-varying graph problems we are familiar with. The data scientist is concerned with developing an all-capturing model but may be less interested in every little detail of the problem the model is applied to. This difference in interest as opposed to a traffic scientist becomes apparent from introductions in published research on this topic [8–10]. Alongside this familiarity with its graph nature, there also exists a lot of recorded data on how traffic is flowing. The combination of intuition and abundant data makes traffic prediction an ideal proving ground for new methods for graph time-series prediction, such as graph-based neural networks. These new methods, though developed on traffic data, are then applied to many different problems, such as modeling epidemics or protein folding [11, 12]. Further, machine learning methods typically require long training times, after which fast inference can be performed. The advantage in computational practicality over the more classical models is then clear: we spend a long time training once after which we can produce many predictions almost in real-time, as opposed to doing medium-length calculations for every new prediction.

The motive with which traffic prediction is studied in this thesis is mostly improving ML time series prediction methods. The goal of the thesis is to arrive at the best possible traffic prediction. Though the motivation is more theoretical, the way in which the improvement is achieved is very much inspired by knowledge of traffic characteristics, which comes in the form of two ideas central to this thesis:

Locality The traffic state at a given location results from information flowing towards it in a neighborhood whose size is determined by the prediction horizon. In simpler words; a change at some location T minutes in the future is the result of traffic information which is T minutes of travel time away. As such, the traffic prediction problem is studied locally, instead of globally on an entire network.

Direction The direction in which information propagates is relevant to the way in which the information interacts with traffic. As such we can leverage knowledge of the different interactions to improve prediction accuracy.

These intuitions are named for easier reference throughout the thesis.

1.2. Content overview

Before starting an introduction regarding the specifics of the thesis content, we first give a general overview of what is to come. The reader will find that the content of this thesis follows two different paths to improve traffic prediction. The first describes changes to the inner workings of the ML methods used. The second describes transforming inputs to extract information in a different way than the ML methods used could. This thesis thus shows two approaches to make a step towards the goal of the best possible traffic prediction and operate on different aspects of producing a prediction out of measured data.

Our first path is focused on improving existing ML methods for short-term traffic prediction. The goal is to achieve an improvement in prediction accuracy using the intuitions presented in the previous section. Before incorporating these intuitions into a state-of-the-art ML model, a more simple ML model is used to test the waters. Then, based on encouraging results of the simple model, the step to a more sophisticated ML model is made. For implementation, especially the **Locality** intuition will ask for some design choices to be made, which will result in different models. Namely; we say that for predicting 10 minutes ahead, we should take information into account which is 10 minutes of driving away, but at what speed? We may use the speed limit of the roads in the network, or we can take a dynamic approach and use the speed currently being driven. Both approaches are explored in this thesis and their use cases will be discussed. This thesis will further present a way of 'freeing' a model of being constrained to predicting where traffic sensors are placed, at the cost of prediction accuracy.

The second path is focused on visualizing and extracting the information contained in a local neighborhood. The idea for the developed method came after attempts to extract information from raw traffic measurements using an emerging mathematical method called Topological Data Analysis (TDA). These attempts failed, leading to the development of a visualization of the data which would additionally lend itself well to TDA. The chapter thus starts with the visualization which can make traffic information traveling on a path insightful, based on how fast parts of the path are traversed in the current traffic situation and how many vehicles are traversing that path. This visualization is then also summarized using this TDA to a single number, which can be used as an additional input to an ML model alongside the raw measurements.

Both paths are described first in chapter 3, where the newly developed methods are introduced. Results of the new methods are then shown in chapter 4.

1.3. Methodological choices

As mentioned before, this thesis will implement ML methods. Specifically, Neural Network (NN) models will be used. The space of NN models is very large, but some basis to start from has to be chosen.

In this thesis we discuss two major classes of neural networks: Convolutional Neural Networks (CNNs) and Graph Neural Networks (GNNs). Required knowledge of how these NNs work will be presented in chapter 2, for now we just sketch a comparison between their characteristics.

The CNN is an earlier class of neural networks. Due to the wide availability of tutorials and software such as provided by TensorFlow [13], CNNs are easier to implement, though not the most appropriate for a problem with a graph nature. If we were to implement a CNN for traffic prediction, we would give it as input measurements of traffic and expect it to output a prediction for the traffic state at one or more locations at some time in the future. This sounds like exactly what we would want; we measure traffic and make a prediction. However, we can identify two shortcomings when applying a CNN to traffic prediction, both stemming from ignoring the road network. The first is that a CNN considers many more interactions than are possible on the network. As an extreme example; consider two isolated islands. We might hope that a CNN learns that traffic on island A is not related to traffic on island B, but there is no guarantee. As such, performance may be hindered. The second is that transferring a

model to a new road network will be very difficult. This is because the type of ”reasoning” the CNN does is as follows: ”I receive an input vector of 100 numbers, of which number 3 is very high, so I predict congestion”. From such a line of reasoning we see that a permutation of the inputs might already disrupt prediction and applying the same model to a new network is very likely to fail. Despite these shortcomings, CNNs have shown to perform reasonably well on traffic prediction [14, 15]. This thesis also applies a CNN to test the premise of the ideas central to this thesis in the context of predicting traffic speed at a single location on a small road network. A CNN is applied there as a proof of concept as only changes in performance are sufficient and where transferability is not needed.

A more recent class of neural networks is the GNN, which deals with the mentioned shortcomings of the CNN by utilizing the structure underlying the problem, the road network in our case. This does away with relations which we know to be non-existent from the road network. Its strength over CNNs is clear from literature [10, 16], at the cost of requiring an underlying network structure. The advantages of a GNN have already been mentioned in the previous paragraph, but we further distinguish two different kinds of ways in which a GNN can operate:

Network level Some GNNs attempt to combine information of all nodes in a graph and the graph structure itself directly. These methods often rely on algorithmically deciding which node inputs are relevant given the node values and the network structure. A network-level predictor has explicit access to all data and can thus capture long-range dependencies well [8].

Node level Other GNNs rely on nodes exchanging information with their neighbors as dictated by the network. This restriction in inputs can improve short-term prediction performance [9] and scaling, but removes explicit access to long-range dependencies. Another important difference with a network-level GNN is that a node-level GNN can operate locally, on only a part of the network.

Due to the setup of these GNNs, a network-level GNN is naturally better suited to capturing long-range effects. A node-level GNN is more flexible, in the sense that missing data on some part of the graph does not influence other regions, as regions can be separated.

A central idea in this thesis is that the traffic state at a given location results from information flowing towards it in a neighborhood whose size is determined by the prediction horizon. Conversely, longer-range dependencies are deemed not relevant. In short-term predictions, this neighborhood can be relatively small. This is in line with the idea of operating at a node level. As such, the traffic prediction problem is studied locally, instead of globally on an entire network.

Lastly, this thesis will not only use neural networks for predicting traffic, but will combine outputs of multiple NN models. This combination is performed using a different ML method: a random forest decision model.

1.4. Contributions

The traffic prediction problem is studied locally, instead of globally on an entire network. This thesis will present a way of defining a local neighborhood from a measured traffic state combined with knowledge of traffic characteristics. To contrast, current GNNs utilize graph structure taking some information some k edges away into account (i.e. exchanging information, message passing), where k is usually taken as 1 [17]. This thesis shows a way of generalizing from GNNs using information from k edges away to incorporating information from some nodes at distance δ away, where the distance is calculated from edge properties i.e. the measured traffic state combined with knowledge of traffic characteristics. In this way, the model presented in this thesis is constructed to leverage dependencies of a longer range than k-nearest neighbors without becoming a network-level predictor.

Along with leveraging these traffic characteristics, an improvement in the applicability of GNN traffic models will be made. Current GNN traffic models mostly focus on predicting traffic states on known traffic sensor locations and some static road network configuration. Consquently, such a model can only be applied to the locations where traffic sensors are placed. Recently, the requirement of static road configuration has been lifted [9], but the applicability remains limited to sensor locations. This thesis offers an extension allowing GNNs to predict at arbitrary locations within a network.

Using an ML model to give a prediction is a way of gaining insight from information. The information

may contain a large number of variables, even when restricting to a local neighborhood as is done in this thesis. A model combines all these variables into a prediction which is easier to interpret and use. Another way of gaining insight into a traffic state is through visualization and summarization of the large number of variables which define the local traffic state. This thesis presents a way of representing these variables in an intuitive manner and additionally summarizes the representation to a single number using an emerging branch of data analysis called Topological Data Analysis.

TDA is a collection of data analysis methods that are designed to find structure in data and is subjectively one of the nicest applications of topology to the real world (note: this is an opinion, but not just mine [18, 19]). Sometimes, the term TDA is also used to refer to just one of these methods; persistent homology. Persistent homology is also the method which will be used in this thesis. Homology is the mathematical study of holes; persistent refers to how long a hole is present in data with respect to some scale parameter. Thus, persistent homology is a method which can identify holes in data and show how important a hole is. Imagine a noisy set of data points which seem to come from a ring, one can find lots of very small holes, but TDA will identify the hole in the ring as the most persistent feature. This method can especially be useful when applied to high dimensional data as high dimensionality typically means sparse data, known as the curse of dimensionality [20]. Sparse data means data with holes or voids between data points. The method of persistent homology has shown to be powerful in a wide range of applications including classifying time series [21] and reconstructing the structure of a rat's physical environment from measuring its brain activity [22][1].

TDA is especially new in the scope of time-series prediction. Methods for applying TDA to time series directly have been developed, which work well for time series classification [21], but do not seem to work well for time series prediction. This thesis shows an alternative approach; generating a new time series by transforming the data at each timestep using TDA. In short, the traffic state is summarized into a diagram of lines which will be called a "traveltime diagram". Between these lines, there will be gaps or holes, which are exactly what TDA is designed to deal with. Persistent homology will be used to transform these diagrams of lines into topological information, which will be fed to a NN to arrive at the end result of a predicted new traffic state.

1.5. Project structure

This thesis is a project in collaboration with Goudappel B.V. Goudappel is a consultancy company for mobility problems. A current implementation for making these predictions is using their "Multimodaal Voorspellend Verkeersmodel Systeem", MVVS for short (Multimodal Forecasting Traffic model System in English) [23]. The MVVS system falls within the first scope of the goals mentioned earlier; its goal is prediction and it is developed from a strong civil engineering background. This system relies on iterative methods for numerically estimating traffic states, which have as most significant drawback their computation time. Depending on the desired accuracy and resolution, their evaluation might take longer than the prediction horizon. In simple words: predicting 10 minutes ahead can take 15 minutes to calculate, so that the 'prediction' is 5 minutes late. Predictions can be made on time by simplifying the prediction method, at the cost of accuracy. As a consequence, Goudappel is also interested in data-driven traffic prediction. Their interest is currently both in studying the capabilities of data-driven methods and in bridging the gap between academic improvements in GNNs and practical implementation.

This thesis presents answers to the following questions:

How can we use knowledge of traffic characteristics to improve GNNs?
As mentioned earlier, we can take inspiration from traffic characteristics to design a new GNN. The new design is more flexible in the selection of nodes to exchange information with than previous GNN definitions and can operate differently depending on the edge direction in the graph.

How can GNNs be used for short-term traffic prediction in a practical setting?
Current traffic prediction GNNs are limited in where they can be applied in space. For a practical application, we wish to be able to predict at an arbitrary location, so that the model is not limited by the existence of physical traffic sensors.

[1] This publication is also explained more accessibly by the author in the following podcast: `https://open.spotify.com/episode/0DYHgH8G3o0NIBuh2BZ8RR?si=-TbZ1LooRvS9wG2f5TQEJQ`.

How can the traffic state of a local region be summarized?
The traffic state of a local region is described by many different variables (at least 2 measured quantities at every sensor in the region). Making sense of these numbers is difficult. Directly applying a NN to obtain a prediction is a way of gaining insight, but we may also ask how we can condense the information in a local region to some quantity which is more humanly interpretable.

This thesis will present an answer to the first two questions in the form of GNN models. The third research question will result in an algorithm for transforming data.

Before any answer can be presented, the methods used must be established. Newly developed methods are introduced, tested and discussed in separate chapters. This leads to the following thesis structure, numbered as their respective chapters in the thesis:

2. A review of literature on the existing methods used in this thesis.
3. Description of the newly developed methods
4. Result from testing the new methods and applying them to data.
5. Conclusions drawn from the results, with a discussion of the work presented.

Lastly, I would like to refer the reader with interest in reproduction and/or extending my work to my GitHub repository associated with this project.

2

Existing methods

Methods used in this thesis are explained before the innovations of this thesis are introduced. This chapter is divided into a part dedicated to the resources used in this thesis, a part dedicated to time series modeling using machine learning methods and a part that introduces Topological Data Analysis (TDA).

2.1. Data

A project in machine learning and prediction naturally requires data. Data will be used to train models and assess model performance, which is thus essential to the project. Most of the data used in this thesis is gathered in Nationaal Dataportaal Wegverkeer (NDW, English: National Dataportal Road traffic). The data is accessible through their tool DEXTER (Data EXploraTion and ExporteR); which could be accessed through Goudappel B.V. The NDW collects and stores traffic data such as speed and flow (vehicles per hour) on more than 19000 sensor locations throughout the Netherlands every minute. The only data used in this thesis not directly accessible through DEXTER is an aggregation of multiple sources. This aggregated dataset is collected by and available through Goudappel B.V. This dataset extends NDW data with, among others, speed and flow measured at traffic lights. The dataset is used for studying traffic in the city of Groningen on a very fine level and it exists because Goudappel use this dataset for their MVVS study. As such, it is dubbed the MVVS dataset.

In this thesis, two measured traffic quantities are used:

Speed The average speed of all vehicles passing the sensor in some time window

Flow The number of cars passing the sensor in some time window, rescaled to the unit of vehicles per hour. Thus, 3 cars passing in 2 minutes amounts to a flow of 90 vehicles per hour. Can also be referred to as traffic intensity.

The combination of these two quantities gives a good picture of a traffic state; slow or fast and calm or busy, and anything in between. Speed and flow are measured by and at physical sensors, often embedded in the road surface and strategically placed by some road authority. As such, spatial resolution of measured traffic state using sensor data is limited. However, with vehicles becoming more connected to the internet, another data source is becoming more reliable: floating car data (FCD). This data source gathers data from vehicles reporting their position and aggregates the information. The resulting data is speed on arbitrary positions in a road network. It might be possible to infer some flow data from FCD, but the FCD data sources currently do not offer information on flow.

2.2. Traffic characteristics

The introduction of this thesis already eluded to using knowledge of traffic characteristics to arrive at better traffic predictions, this section will introduce the characteristics which will be used.

The most important realization is that traffic information flows differently depending on the direction

in which it flows. We distinguish two directions: upstream, against the direction of the vehicles, and downstream, with the direction of the vehicles. Information flowing downstream is carried by the vehicles themselves and thus travels with the speed of the traffic. Information flowing upstream is different. When a traffic slow-down occurs at one location, vehicles approaching that location will also slow down to avoid collisions. The slow-down will travel upstream with a speed lower than that of the vehicles, which can intuitively be seen as vehicles piling up in front of a traffic jam. This is described as a wave in vehicle density (amount of vehicles per unit length of road), which travels upstream with a speed of at most $20\,\mathrm{km\,h^{-1}}$[24]. We can thus say that information travels upstream with the speed of this wave. This upstream wave speed will be used when selecting relevant inputs for an ML model later in the thesis.

2.3. ML time series modeling

Now we will start introducing the ML methods that will be used in this thesis, but first we note that the traffic prediction problem can be seen as a time series problem on a graph. To this end, some concepts crucial in time series modeling are introduced before delving into ML specifics. The main ML method used in this thesis is called a Graph Neural Network, the introduction of which will start with simpler Neural Networks. The random forest, another ML method, is also used, but this thesis does not propose any changes to the random forest, so its introduction is a bit shorter. The goal of this section is to establish how time series modeling and the machine learning methods used in this thesis work individually and to establish how they are used together.

2.3.1. Time series

We begin with general methods of modeling time series and assessing the performance of a model. Suppose there exists time series data of n-dimensional variable $x_t \in \mathbb{R}^n$, for some arbitrary $n \in \mathbb{N}$. This variable may correspond to traffic flow and speed at some sensor location for example. The goal of time series modeling is to obtain a function $f : \mathbb{R}^n \mapsto \mathbb{R}^n$ which predicts the variable x at some time in the future. One might want to predict the variable at an arbitrary time step in the future, so we introduce the prediction horizon h, such that $f(x_0)$ is aimed at predicting x_h. Further, more information than just the current observation can be used. Let the input width $W \in \mathbb{N}$ be the number of timesteps used as input and let $1 : W$ denote $1, 2, .., W$. The goal of a so-called single-step estimation problem is then to find the function f such that $f(x_{1:W}) = x_{W+h}$. This process is outlined in figure 2.1. This process is extended in a multi-step estimator $f(x_{1:W}) = (x_{W+h_1}, x_{W+h_2}, ..., x_{W+h_m})^T$ for arbitrary $h_i \in \mathbb{N}, m \in \mathbb{N}$. This windowing process means that there are some requirements for the data. Note that a time-series of T consecutive steps will generate $1 + T - (W + h)$ samples. Thus, some time-series of m groups of W long consecutive data points will yield no usable data for the chosen value of W.

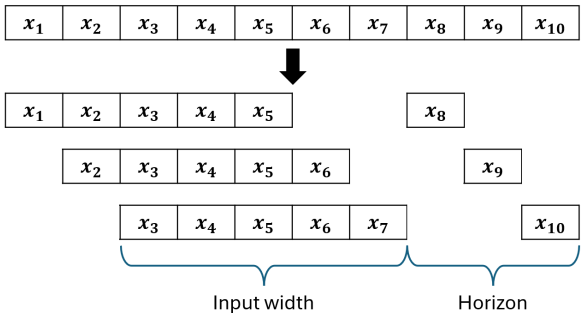


Figure 2.1: The windowing process of a single-step estimator is shown for an input width of $W = 5$ and a horizon of $h = 3$. The time series is thus split up into inputs of size W and a prediction target at time h ahead of the last entry in the corresponding input.

There exist many methods of optimizing f, most share the property that they are intended to optimize the Mean Squared Error (MSE) or Mean Absolute Error (MAE). In this thesis, functions will be optimized w.r.t. the MSE. Given some time series $\mathbf{x} = x_1, x_2, ..., x_T$ for some $T \in \mathbb{N}_{\geq W+h}$, the MSE and MAE are formulated as follows:

$$\mathrm{MSE}(f, \mathbf{x}) = \frac{1}{1 + T - (W + h)} \sum_{i=W}^{T-h} \left(f(x_{i:i+W}) - x_{i+W+h}\right)^2,$$

$$\mathrm{MAE}(f, \mathbf{x}) = \frac{1}{1 + T - (W + h)} \sum_{i=W}^{T-h} \left|f(x_{i:i+W}) - x_{i+W+h}\right|.$$

Common methods for finding an f approximating the true relations, thus modeling time series, are the ARIMA [25] and GARCH [26] models (and their plentiful variants). These models however tend to share the assumption that the process they are modeling is stationary in some sense. This assumption does not hold in the traffic problem. Some simple reasons for this are that people tend to travel more during rush hours causing congestion, whereas traffic at night is very stable since there are few vehicles on the road, or that traffic is influenced by random external factors such as the weather.

In the last 20 years, some of the focus in time series prediction has shifted to using machine learning methods to obtain f. The large data availability and complex spatio-temporal dependence of the traffic problem make ML well suited [27].

2.3.2. Working with ML

In this thesis ML methods will be used, but what is ML? The original description can be paraphrased as 'statistical models which can perform a task without being explicitly programmed to do so' [28]. In other words: a computer uses statistics to 'learn' how to perform some task. In this thesis ML will be used to arrive at the best possible traffic prediction. This thesis will have regression and classification models. A regression model predicts a continuous value such as future traffic flow or speed from its input data. Classification models give a value in some finite set of values; this thesis will use this as automation in combining outputs of multiple regression models into a single prediction.

Most of the models used are based on artificial neural networks, which is the most popular form of machine learning judging by the sheer volume of research published on the subject [13, 16, 29–31]. In this thesis, artificial neural networks for regression and random forests for classification will be used. Both will be explained here.

Machine learning models tend to have a large number of parameters, which is both their strength and weakness. To illustrate how many parameters can be used; over $5 \cdot 10^5$ is very common: the popular Chat-GPT4 model is rumored to have 1.8 trillion parameters[32]. Having so many parameters is a part of what makes neural networks good general predictors. On the other hand, this much freedom will almost definitely lead to overfitting if no countermeasures are taken. Most of these measures are based on restricting data usage. Two methods used in this thesis are explained below.

The first method is splitting the available data into three sets: training, validation and testing data. Their respective sizes used in this thesis are 70%, 15% and 15%, which is a commonly occurring partitioning [33]. The neural network models are trained by passing over the training data multiple times, each pass is called an epoch. After each training pass, the prediction performance of the model on the validation data is calculated. The model with the best validation performance over all epochs is taken as the final model. The performance of this model may now be correlated with the validation data, so the final performance of the model is calculated on the testing data. This method is designed to 'stop' the training when overfitting to the training data occurs. In the case of a random forest, no validation step is needed, so data is simply split into training and testing sets of sizes 85% and 15% respectively.

The second method is aimed at preventing overfitting to the training data and is called drop-out. Within this thesis, it is applied to neural networks only. Given some input $x = (x_1, x_2, \ldots, x_N)$, the drop-out method randomly sets the x_i to 0 with probability p and then rescales the vector by $1/(1-p)$. That is, if $y_i \sim \mathrm{Ber}(p)$ i.i.d., then

$$\mathrm{Dropout}(x) = \frac{1}{1-p} \cdot (x_1 \cdot y_1, x_2 \cdot y_2, \ldots, x_N \cdot y_N).$$

This random dropout is applied after every layer in the neural network. The y_i are independently drawn on every evaluation of the Dropout function, so that the dropped x_i may differ for every sample and every epoch. This method has shown to be very useful in preventing overfitting and improving performance on testing data [34]. During validating or testing, $p = 0$ is taken, so that $\mathrm{Dropout}$ is just an identity mapping.

2.3.3. Neural networks

As previously mentioned, the most popular form of machine learning models is the (Artificial) Neural Network ((A)NN). Their popularity is understandable, given their ease of use and performance on seemingly arbitrary tasks, even flying airplanes [35]. A simple neural network model is first described below, after which more modern adaptations are explained.

The initial idea of the ANN was to mimic neural signaling in the brain so that a computer might be able to learn the same way a human can. Their main structure consists of so-called neurons, grouped in ”hidden layers”. Input data would then propagate to a hidden layer through some weighted linear aggregation, after which a non-linear function is used to calculate the values of the neurons in the hidden layer. The neuron values are then the inputs for the next layer and so on. The structure is also shown in figure 2.2.

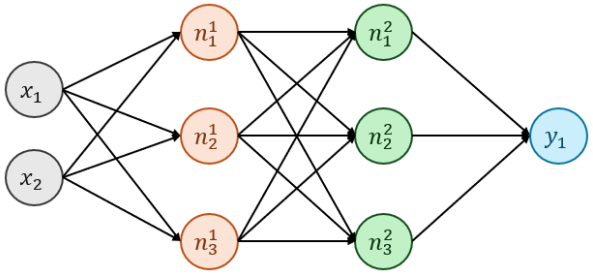


Figure 2.2: Basic neural network with 2 inputs, 2 hidden layers of each 3 neurons and 1 output. The arrows indicate the existence of a weight between two neurons.

In more technical terms, let x^ℓ be the vector of values in layer ℓ of the neural network. For a current layer ℓ with n neurons and next layer $\ell + 1$ with m neurons, we have that the inputs are aggregated linearly, after which a non-linear function g is applied to obtain $x^{\ell+1}$. This non-linear function is often taken to be the so-called Rectified Linear Unit (ReLU), given by $\mathrm{Relu}(x) = \max(0, x)$. Since the aggregation is linear $y = ax + b$, we can write the aggregation of x^ℓ as a weight matrix $W \in \mathbb{R}^{m\times n}$ and some bias $b \in \mathbb{R}^m$. Written out fully:

$$\text{Let } z = Wx^\ell + b, \text{ then } x^{\ell+1} = (g(z_1), g(z_2), ..., g(z_m))^T.$$

The weights and biases are then adjusted layer for layer at every training step of the model, propagating backwards from the model output, based on the gradient of the output of each layer. Thus, the weights and biases are what is optimized during the training of a neural network and are also called the ’learnable parameters’. The transformation of the data according to the weights W and biases b is also called a ’learnable’ transformation. See [36] for more details on how the backward propagation is calculated. The layers in the neural network sketched above are often called ’Dense’ layers.

A recent improvement over the simple NN introduced above is the addition of ”soft” weights through the so-called attention mechanism [37]. This mechanism makes the weights going from layers ℓ to $\ell + 1$ dependent on the output x^ℓ of layer ℓ. The idea behind this mechanism is that some parts of an input might be more relevant in some situations. The attention mechanism used in this thesis will be explained on an intuitive level, with in-depth details found in [9, 31, 37].

Consider a device which can generate two periodic signals; a positive signal with a period of 3 and a negative signal with a period of 5. Given a sample of 5 steps of one of the waves, we task a model with outputting the signal continuously. As the signal is periodic, the model could output the value 1 period earlier to perfectly predict the signal. To obtain what the period of the signal is, the model could ”pay *attention*” to the polarity of the signal; if negative, it will output the part of the input corresponding to 5

steps ago. Thus, if we describe the model as a NN with 5 inputs, no hidden layers and 1 output, we may write the weights as

$$W = \begin{cases} (0,0,1,0,0) & x_1 > 0 \\ (0,0,0,0,1) & x_1 < 0. \end{cases}$$

This automatic calculation is often not as clear cut in practice, so the model is allowed to 'learn' what it needs to pay attention to itself. How much attention it needs to pay to each aspect of the input is normalized using what is called a softmax function, though readers from other field than mathematics or computer science might recognize this function as exponential weighting or a Boltzmann distribution [38]. More details regarding for example how training works can be found in one of [9, 31, 37].

Note that in figure 2.2, the network could also stop after the first layer, then it would be a neural network with 1 layer and 3 outputs. The other way around, we could append to the network in figure 2.2 another neural network predicting some quantity z from y, forming a larger NN in which we see figure 2.2 as a more complicated layer. From here on, the terms neural network and layer can be used interchangeably.

2.3.4. Recurrent neural network

Although the previously sketched NN can already be quite powerful in certain tasks, neural network use in time series prediction can be made more natural using a Recurrent Neural Network (RNN). The basic idea of an RNN is that its inputs are not only the input at time t, but also a secondary output of the RNN at time $t-1$, as shown in figure 2.3. This passing along of some information between the RNN allows the NN to better capture time-varying signals[1].

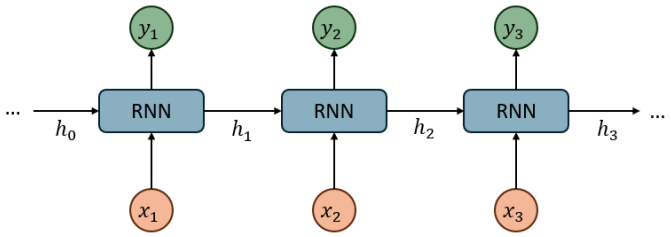


Figure 2.3: Basic RNN structure. Inputs are denoted by x_i, outputs by y_i and the between-RNN in-/output is denoted by h_i.

The space of RNNs has seen improvements in dealing with long-term dependencies resulting in the form of RNN used in this thesis: the Gated Recurrent Unit (GRU) [39]. The exact calculations of the GRU are not the focus of this thesis, so they are omitted here. We do however show figure 2.4, which is a slight adaptation of the figure seen before. The GRU does not pass a secondary output, but rather uses its output as an additional input at the next time step.

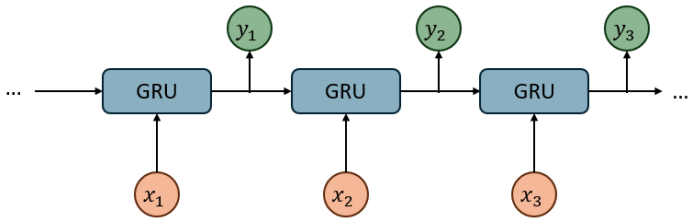


Figure 2.4: GRU structure. Inputs are denoted by x_i, outputs by y_i.

The use of previous predictions allow the RNN to have a form of memory, which aids in time-series prediction.

[1] Language can also be seen as a time-varying signal, RNNs are often used in large language models: chatbots.

2.3.5. Graph neural networks

Another extension of the ANN is Graph Neural Networks (GNNs), which will leverage a graph structure underlying the traffic problem to improve prediction performance. As in the introduction, the best way to motivate GNNs is perhaps by imagining the shortcomings of a neural network which does not utilize the graph structure. To this end, consider feeding a NN with traffic information of an entire country: speed and flow measured at sensor locations. If by chance there exists some correlation in the training data between nodes which are at different extremes of the country, the neural network will pick up on this correlation and use it for prediction, even though a human observer would consider this relation non-existent. Still, one might ask why picking up on this relation is bad. The answer is that probably, most of the time it is not, traffic intensity is often correlated through rush hours. However, if an accident occurs at one location, we know that this should not influence the other extreme of the country.

A graph neural network extends artificial neural networks by incorporating the underlying graph structure of a problem. There exist many ways to incorporate this structure, but since we will be applying the GNN to operate locally, the most relevant types are based on what is called 'message-passing'. Message passing means passing information to neighboring nodes, so that we may write a general GNN layer as follows

$$x_u^\ell = NN\left(\{x_v^{\ell-1} : e_{vu} \in G\}\right).$$

In this notation, a node v is said to be a neighbor of node u if there exists an edge from v to u, denoted by e_{vu}, in the graph G. Information on the edge along which information is passed to a neighbor can also be used, writing

$$x_u^\ell = NN\left(\{x_v^{\ell-1}, e_{vu} : e_{vu} \in G\}\right).$$

Here u is the output node, v is a possible input node, e_{vu} denotes an edge from node v to node u in the graph G and NN denotes an NN applied to the graph-based inputs. Taking into account only the direct neighbors of u is called a 1-hop GNN and is the most common implementation of a GNN. There also exist k-hop GNN implementations, where all neighbors within k-hops are included [17].

The GNN can be seen way of restricting or streamlining the information which can be used in predicting at some specific edge or node. Aside from this restriction, there is often also information added regarding the path the information took, for example how long a section of road between two sensors is. This new way of selective information usage improves the prediction performance of artificial neural networks [10, 16].

2.3.6. SpaceTime Neural Network

Chapter 3 of this thesis extends a state-of-the-art GNN. The GNN used as a base model is called SpaceTime Neural Network (STNN), by Yang *et al.* [9]. The largest difference to other GNNs [10, 16] applied to traffic prediction is that it performs a node-level task. This means switching to local estimation and incorporating information on only a local sub-network. By training a single model on many sub-networks, the goal is to obtain a general traffic predictor applicable in an arbitrary region of a network.

Switching to local prediction weakens the dependence on the larger structure with the following advantages over network-level predictors:

1. Switching from predicting on an entire network to predicting locally means that evaluation scales linearly with the number of nodes in the network instead of quadratically. This leads to faster training and inference.
2. Predicting locally means that the model can be trained and evaluated on all nodes of an arbitrary network. This is called having high transferability.
3. Larger sample size from the same measurements. Given N measurements at K traffic sensors, a network-level predictor has at most N data samples. A local predictor can split sub-networks for each traffic sensor, giving at most $N \times K$ data samples.

The local sub-network is a sub-network in both space and time. Suppose we are trying to predict traffic at some sensor which we will now call the target sensor. Given an input width W and an amount of neighbors to consider M, STNN will take as inputs all past W measurements of the M sensors which

are closest to the target sensor in travel distance. This process can be described in the framework of a 1-hop GNN, though it is a bit cumbersome. A better suiting framework is presented in chapter 3.

Consider transforming the W timesteps of the M nodes by creating a local space-time graph with nodes v_j^i where $i \in \{1, 2, ..., W\}$ denotes a time and $j \in \{1, 2, ..., M\}$ denotes the sensor. Edges are added in this new space-time graph from all v_j^i to the target sensor v_t^W, then the STNN process can be seen as applying a 1-hop GNN layer to all past W-measurements at the M closest sensors. These new edges have as weight the shortest path distance to the target sensor. An example of this transformation is shown in figure 2.5, which is also used as input to the next layers.

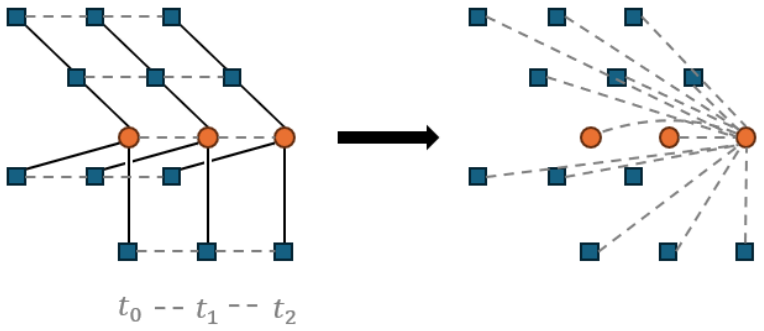


Figure 2.5: The graph transformation process used in STNN. The target node is indicated with an orange disc and other nodes are shown as blue squares. In the left network, roads are indicated with solid black lines and the time dimension is shown using grey dashed lines. On the right, we have a graph with all nodes linking to the target node at the latest timestep.

After using this GNN step to collect input information, STNN automatically decides which of its $W \times M$ inputs to ”pay attention” to using an attention layer as briefly explained in subsection 2.3.3 before applying more conventional Dense (ReLU) layer steps. Details of the parameters and setup of these layers are not relevant to this thesis, but can be found in [9].

It has been shown that for traffic prediction, STNN is capable of outperforming other GNNs when the prediction horizon is less than or equal to 15 minutes [9]. STNN, being a node-level predictor, was also compared against network-level predictors (like [8]). The latter performed better at longer prediction horizons of more than 30 minutes. The fact that STNN as a local predictor performs well with a shorter prediction horizon is expected within the **Locality** intuition used in this thesis, as we deem a local neighborhood which can be traversed in around 15 minutes unlikely to capture information important to the traffic state 30 minutes into the future.

Further, as a local predictor does not depend on the network structure of an entire network, STNN trained on one city and then tested on another city has shown to perform comparably to some previously state-of-the-art GNNs. Applying these previously state-of-the-art GNNs to a different city is either not possible at all or does not perform well [9].

The combination of both improvements seems too good to be true. As mathematicians often say, there is no such thing as free lunch. The catch is that STNN only performs this well in practice on a prediction horizon of less than 15 minutes.

2.3.7. Random forest

As eluded to in the description of traffic characteristics, information in traffic will propagate differently based on its direction. This thesis will generate models for traffic prediction based on these different directions, whose outputs will combined into a final prediction using random forests.

To explain a random forest we first describe the population of the forest: (classification) trees. A classification tree splits data into groups using inequalities. As an example, consider how tall people are. We use two categories: tall and short. The classification tree would then decide whether a person is tall or short, given their length and gender. It would deem men above $181.1\,\mathrm{cm}$ and women above $167.4\,\mathrm{cm}$

tall, the decision tree is shown in figure 2.6. The tree has 4 endpoints, which are called leaves, which does not need to correspond to the number of categories.

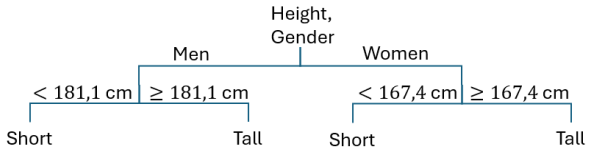


Figure 2.6: A decision tree which decides on whether a person is tall based on their height and gender.

These trees are formed automatically in practice according to some optimization rule, often entropy optimization [40]. Decision trees however suffer from high variance; a small change in training data can cause a large change in its outputs. This is mitigated by taking the mean output of many trees, trained on datasets created by sampling with replacement from the original dataset. Typically, if the original dataset contains N samples, then so does every resampled dataset N. This process is called bagging. To further reduce variance, each tree is only allowed to consider a small subset of the features, typically $\sqrt{M}$ where M is the number of available features. The model resulting from bagging and restricting features is called a random forest. For details on the cases in which resampling does not reduce the variance see [40].

The final output of the random forest can be the class predicted by the majority of the trees, or it can be a normalized vector of how many trees of the forest *voted* on a certain class. In the context of the example above, suppose we have a random forest with 300 trees, 200 of which gave output Tall and 100 of which gave output Short. The output could thus be either the majority vote Tall, or it could be the vector $(2/3, 1/3)^T$ corresponding to classes Tall and Short respectively.

The way in which the outputs of the NNs and random forests will be combined is explained later in the modeling chapter.

2.4. Topological Data Analysis

We now switch to a deeper mathematical topic in Topological Data Analysis, which is another important method used in this thesis. TDA will be used to summarize traffic data with the goal of extracting more useful information to use in prediction.

TDA is based on the idea that the 'shape' of data is important. To be more precise, TDA is concerned with the absence of data in n dimensional regions or holes. As mentioned in the introduction, this thesis will show a way of visualizing local traffic information in a diagram of lines with gaps between them. These gaps will be transformed into holes, which are analyzed using TDA. This section will show how holes in a cloud of data points are identified and tracked using TDA.

As an example, consider the point cloud as shown in figure 2.7a. It is sampled from a circle with some noise added to it. We can visually inspect this figure and might conclude that there is a hole. Seeing that 2.7b is sampled from a torus however is more difficult and higher dimensional data can no longer be shown in a single image. TDA can be used to identify holes in data and provide information on their size and underlying topology. In order to track holes, the notion of a topological space and homotopy equivalence is introduced below.

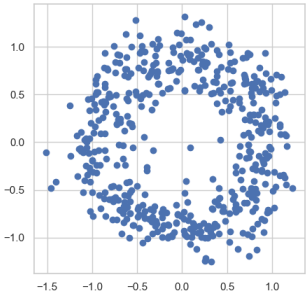
Definition 1 (Topological space)**.** Let D be a set and let τ be a collection of subsets of D, such that

- $\emptyset \in \tau$ and $D \in \tau$,
- Any (finite or infinite) union of members of τ belongs to τ,
- Any finite intersection of members of τ belongs to τ.

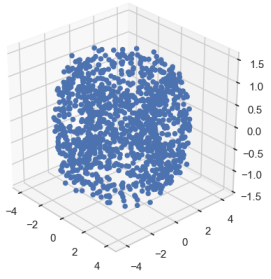
The members of τ are called open sets and τ itself is called a topology. If we write the combination $X = (D, \tau)$, then X is called a topological space.

Definition 2 (Homotopy)**.** Let X and Y be topological spaces and let f and g be continuous functions mapping from X to Y. A homotopy between $f, g : X \mapsto Y$ is a continuous function $H : X \times [0,1] \mapsto Y$ such that $H(x,0) = f(x)$ and $H(x,1) = g(x)$ for all $x \in X$.

As such, a homotopy continuously deforms f to g. In visual terms; a homotopy H continuously deforms the image of $f(X)$ to $g(Y)$, meaning that its topological holes. As such, two functions are said to be homotopy equivalent if there exists a homotopy between them. Homotopy equivalence is what is meant when mathematicians say that a donut and a mug have the same shape.



(a) Data sampled from a circle with some noise.



(b) Data sampled from a torus with some noise.

Figure 2.7: Point clouds sampled from shapes with holes.

This section will explain how to extract these holes from data. Also, we will stop talking about holes and will talk about topological features instead. This is because the topological features are of some dimension. Suppose we were to keep talking about holes, a 1-dimensional is like that shown in figure 2.7a. A 2-dimensional hole could be the circular void inside a torus. However, 0 or higher dimensional holes are hard to imagine. To avoid confusion, we will talk about topological features in this section. Later parts will refer to 1-dimensional features as holes.

2.4.1. Simplicial complexes

At the basis of TDA lie the mathematical objects named simplicial complexes. Simplicial complexes will allow us to define topological features later in the explanation of TDA. We start with a simplex. Consider a set of $n+1$ points $\mathbb{X} = \{x_0, x_1, ..., x_n\}$ in $\mathbb{R}^d$ which are affinely independent. The convex hull of $\mathbb{X}$ is also called the n-dimensional simplex $\sigma = [x_0, x_1, ..., x_n]$, where the x_i are called its vertices. The simplices spanned by the subsets of $\mathbb{X}$ are called the faces of σ. A geometric simplicial complex K in $\mathbb{R}^d$ is defined as follows:

Definition 3 (Geometric simplicial complex)**.** A geometric simplicial complex K is a collection of simplices such that

1. Any face of a simplex $\sigma \in K$ is also in K
2. The intersection of two simplices in K is either empty, or a common face of both.

Note that the intersection taken is with regard to the space spanned by a convex hull. An example of when this fails is shown in figure 2.8. The union of all simplices in K is called the underlying space of K and it inherits topology from $\mathbb{R}^d$, So, K can also be seen as a topological space through its underlying space. Also, note that K can be fully described by a combinatorial description of the elements of its underlying space following some rules. The possibility of describing a geometric simplicial complex through combinatorics gives rise to an abstract simplicial complex as shown in figure 2.9.

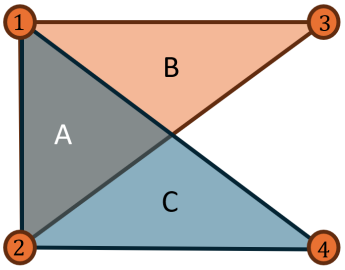


Figure 2.8: Intersection of simplices $[1, 2, 3]$ and $[1, 2, 4]$. Their intersection is region A, which is not a possible simplex given these data points. Simplices $[1, 2, 3]$ and $[2, 3, 4]$ would satisfy condition 2 in the definition of a geometric simplicial complex.

Definition 4 (Abstract simplicial complex)**.** Let V be a set. An abstract simplicial complex with vertex set V is a set $\tilde{K}$ of finite subsets of V such that

1. The elements of V are in $\tilde{K}$
2. For any $\sigma \in \tilde{K}$, all subsets of σ are in $\tilde{K}$

The elements of $\tilde{K}$ are called its simplices or its faces.

Note that the abstract simplicial complex has no notion of spanned space, so any intersection of simplices is simply the intersection of their vertices.

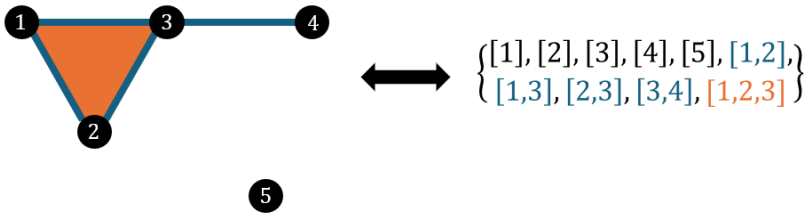


Figure 2.9: An example of switching between geometric and abstract simplicial complexes. The black points correspond to 0-dimensional simplices, blue lines to 1-dimensional simplices and the orange area to a 2-dimensional simplex.

The reverse relation also exists: consider some abstract simplicial complex $\tilde{K}$ with some associated topological space $|\tilde{K}|$ from which it could have been constructed. Then, if geometric simplicial complex K has combinatorial description $\tilde{K}$, the underlying spaces of K and $|\tilde{K}|$ are homeomorphic. Homeomorphy is stronger than homotopy, so the underlying spaces of K and $|\tilde{K}|$ are homotopic. Note how this allows tracking topological features in some geometric simplicial complex by studying its abstract simplicial complex.

There exist many ways to generate simplicial complexes from data, but this thesis makes use of one widely used method: the Vietoris-Rips complex. However, as the theoretical framework can be explained more naturally with another method, we will first introduce the Čech complex and later define the Vietoris-Rips complex and discuss their differences.

Definition 5 (Čech complex)**.** Given points $\mathbb{X} = \{x_1, x_2, ..., x_n\} \subset \mathbb{R}^d$ and $r > 0$, the Čech complex of $\mathbb{X}$ at radius r is the abstract simplicial complex defined by the set of simplices $[x_1, x_2, ..., x_k]$ such that the k closed balls $B^r_{x_i}$ have non-empty intersection. Thus,

$$\check{\text{Cech}}_r(\mathbb{X}) = \left\{[x_1, x_2, ..., x_k] : \cap_{i=1}^{k} B^r_{x_i} \neq \emptyset\right\}.$$

2.4.2. Homotopy

As remarked earlier, studying the abstract simplicial complex associated with some geometric simplicial complex tracks its topological features. In practice, the geometric complex is the data to be studied. The question then arises, what are the topological features in the data? An intuitive way to make these topological features apparent is to consider the union of balls with radius r centered at the datapoints $x \in (\mathbb{X} \subset \mathbb{R}^d)$. It has been proven that, assuming some well-chosen value of r, the union of balls is homotopy equivalent to some offset of the support from which $\mathbb{X}$ is sampled, see reconstruction theorem in [41] for more details. From the union of balls, an abstract simplicial complex can be generated using nerves. Thus, nerves are introduced.

Definition 6 (Nerve)**.** Given a cover $\mathcal{U} = (U_i)_{i\in\mathcal{I}}$ of $\mathbb{M}$, that is a family of sets U_i such that $\cup_{i\in\mathcal{I}} U_i \supseteq \mathbb{M}$, the nerve of $\mathcal{U}$ is the abstract simplicial complex $C(\mathcal{U})$ with vertex set $\{U_i\}_{i\in\mathcal{I}}$ such that

$$\sigma = \left[U_{i_0}, U_{i_1}, ..., U_{i_k}\right] \in C(\mathcal{U}) \iff \cap_{j=0}^{k} U_{i_j} \neq \emptyset.$$

An example of some data points with balls of radius r as cover turning into a nerve is shown in figure 2.10.

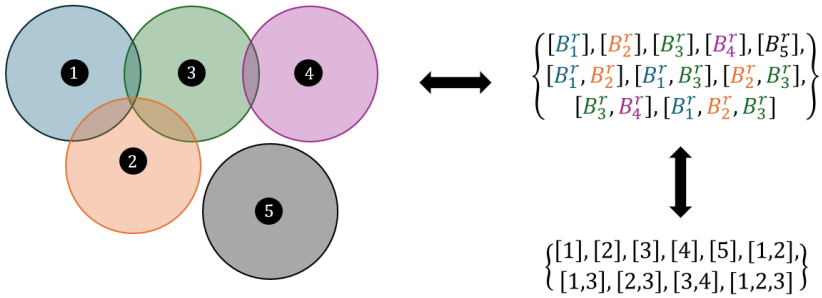


Figure 2.10: The nerve resulting from the union of balls on some example data points. Note that this corresponds to the abstract simplicial complex in figure 2.9 as each ball is identified by its center.

The nerve also admits homotopy through the nerve theorem:

Theorem 1 (Nerve theorem)**.** Let $\mathcal{U} = (U_i)_{i\in\mathcal{I}}$ be a cover of a topological space X by open sets such that the intersection of any subcollection of the U_i's is either contractible or empty. Then, X and the nerve $C(\mathcal{U})$ are homotopy equivalent.

Now stringing the nerve theorem (theorem 1) and the reconstruction theorem (again, see [41]) together: for some well-chosen radius r, the nerve of the union of balls on some data $\mathbb{X}$ is homotopy equivalent

to the support the data was sampled from. We have seen the nerve of the union of balls with radius r before: the Čech complex with radius r. In other words, if r is well-chosen, the Čech complex is homotopy equivalent to the support of the data.

2.4.3. Homology

So far, the possibility of tracking topological features through homotopically equivalent objects has been shown. The definition of topological features in complexes is studied using homology groups of chain complexes. Let K be a finite simplicial complex and let $k \in \mathbb{N}_{\geq 0}$. A simplex $\sigma = [x_0, x_1, ..., x_k]$ of $k+1$ elements is called a k-simplex. The space of k-chains on K, denoted $C_k(K)$ is the set of finite sums of the k-simplices in K. That is, for σ_i k-simplices in K, a k-chain c is defined as

$$c = \sum_i^p \varepsilon_i \sigma_i, \quad \text{with } \varepsilon_i \in \mathbb{Z}_2.$$

This notation is simply stringing together simplices, where requiring that the ε_i are in $\mathbb{Z}_2 = \{0, 1\}$ automatically removes doubly counted simplices. To see this, we let $c = \varepsilon_1 \sigma_1$ and $c' = \varepsilon_1' \sigma_1$, with $\varepsilon_1 = \varepsilon_1' = 1$. Then, $c + c' = (\varepsilon_1 + \varepsilon_1')\sigma_1 = (1+1)\sigma_1 = 0$, as $1 + 1 = 2 \equiv 0$ in $\mathbb{Z}_2$. Imagine two adjacent triangles, with the edges between their vertices being the 1-simplices. If c_1 is the chain enclosing the first triangle and c_2 is the chain enclosing the second triangle, $c_1 + c_2$ encloses the diamond formed by the union of the two triangles. This is shown in figure 2.11.

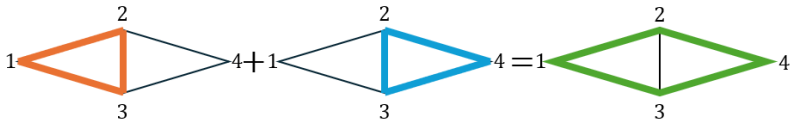


Figure 2.11: Sum of two triangular chains.

The boundary $\partial_k(\sigma)$ of a k-simplex $\sigma = [x_0, x_1, ..., x_k]$ is the $k-1$-chain of all $k-1$-sub-simplices of σ:

$$\partial_k(\sigma) = \sum_{i=0}^{k} [x_0, x_1, ..., \not{x}_i, ..., x_k],$$

where $[x_0, x_1, ..., \not{x}_i, ..., x_k]$ denotes the simplex with vertex x_i removed. Note that the k-simplices form a basis of C_k and that ∂_k can be extended to applying on chains as a linear map, called the boundary operator. In the scope of figure 2.11,

$$\begin{aligned}\partial_k\left([1,2,3] + [2,3,4]\right) &= \overbrace{[1,2] + [2,3] + [1,3]}^{\text{simplices from [1,2,3]}} + \overbrace{[2,3] + [2,4] + [3,4]}^{\text{simplices from [2,3,4]}} \\ &= 1 \cdot [1,2] + 2 \cdot [2,3] + 1 \cdot [1,3] + 1 \cdot [2,4] + 1 \cdot [3,4] \\ &\overset{\varepsilon_i \in \mathbb{Z}_2}{=} [1,2] + [1,3] + [2,4] + [3,4] \in C_{k-1}.\end{aligned}$$

A k-chain which has zero boundary is a k-cycle, with the kernel $Z_k(K) = \{c \in C_k(K) : \partial_k(c) = 0\}$. The space of K-cycles which are the boundary of some $k+1$ cycle in K is called the space of k-boundaries of K; $B_k(K) = \{c \in C_k(K) \mid \exists c' \in C_{k+1}(K) : \partial_{k+1}(c') = c\}$. By realising that $\partial_k \circ \partial_{k+1} = 0$, one can see that $B_k(K) \subseteq Z_k(K)$. Then, we arrive at the following definition:

Definition 7 (Simplicial homology group and Betti numbers)**.** The k-th simplicial homology group of K is the quotient vector space

$$H_k(K) = Z_k(K)/B_k(K).$$

The k-th Betti number of K is the dimension of the vector space $H_k(K)$; $\beta_k(K) = \dim H_k(K)$.

From this definition it follows that the k-th Betti number of K corresponds to the number of k-dimensional topological features in K. Also, note that two k-cycles are homologous if they differ[2] by a k-boundary,

[2]A difference is equivalent to a sum since we are dealing with the field $\mathbb{Z}_2$.

so the boundary of a $k+1$-chain which exists in K. Conversely, if two k-cycles differ by a cycle which can not be a boundary of a $k+1$ chain in K, they are not homologous. If a cycle can not be a boundary of a $k+1$ chain in K, then it is homologous to the boundary of a k-dimensional topological feature in K. This is also illustrated in figure 2.12.

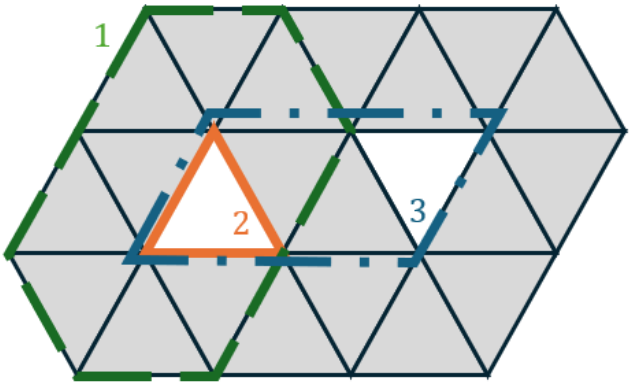


Figure 2.12: An example of cycles. All shaded areas are 2-simplices, the unshaded areas are absent in the simplicial complex. Note that the cycle indicated by dashed line 1 and the cycle indicated by the solid line 2 differ by a boundary of shaded regions, so they are homologous. The cycle indicated by dash-dotted line 3 does not differ by a boundary of a shaded region; it encloses the right hole (1-dimensional topological feature). As such, cycle 3 is not homologous to cycle 1 and cycle 2. This example has $\beta_1 = 2$, as there are two holes i.e. 1-dimensional topological features.

With this knowledge of homology combined with what is explained in subsection Homotopy, we can see how topological features can be identified and we know how they relate to the homology of the support of the sampled data. However, recall that earlier we discussed placing balls of radius r around data points. The choice of r still remains uninformed, which is addressed in the next subsection.

2.4.4. Persistent homology

At this point, it is clear that the Čech complex built from data can be used for identifying topological features in the support of the data. However, the choice of r remains uninformed. This subsection will introduce the important notion of *persistent* homology. Persistent homology will deal with the choice of scale parameter r and will fill form the bridge between the abstract mathematics in the previous subsections and real-world data.

We also take a step away from the Čech complex, as it is expensive to compute. This is often tackled by switching to the aforementioned Vietoris-Rips complex. Furthermore, the Vietoris-Rips complex depends on distances between datapoints, whereas the Čech complex on the positions of the datapoints in $\mathbb{R}^d$. The Vietoris-Rips complex can thus also be used on data which does not embed into $\mathbb{R}^n$ for any n. The Vietoris-Rips complex can hence also be used on for example a correlation matrix. However, the Vietoris-Rips complex at some radius r is not guaranteed to be homologous to the union of balls with radius r on the data. Both the new Vietoris-Rips problem and choice of r are tackled by making use of the to be introduced notion of persistence.

First, define the Vietoris-Rips complex, sometimes also referred to as simply Rips complex.

Definition 8 (Vietoris-Rips complex)**.** Given points $\mathbb{X} = x_1, x_2, ..., x_n \subset \mathbb{R}^d$ with some distance between points $d(x_i, x_j)$, let $r > 0$. The Vietoris-Rips complex of $\mathbb{X}$ at radius r is the abstract simplicial complex defined by the set of simplices $\sigma = [x_1, x_2, ..., x_k]$ such that the distance between all vertices of any simplex is less than r. Thus,

$$\text{Rips}_r(\mathbb{X}) = \left\{\sigma : \forall_{x_i, x_j \in \sigma} d(x_i, x_j) \leq r\right\}.$$

It can be shown that

$$\check{\mathrm{C}}\mathrm{ech}_r(\mathbb{X}) \subseteq \mathrm{Rips}_{2r}(\mathbb{X}) \subseteq \check{\mathrm{C}}\mathrm{ech}_{2\delta r}(\mathbb{X}) \subseteq \mathrm{Rips}_{4\delta r}(\mathbb{X}), \text{ where } \delta = \sqrt{\frac{d}{2(d+1)}}. \tag{2.1}$$

As a consequence, there exist maps between their homology groups with the following commutative diagram [42]:

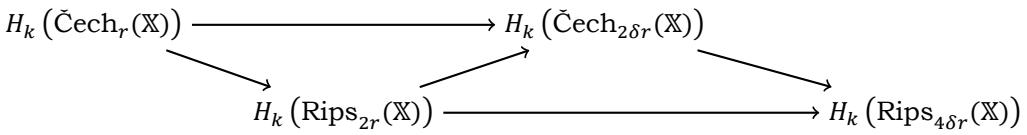


Figure 2.13: Mapping between homology groups of Čech and Vietoris-Rips complexes.

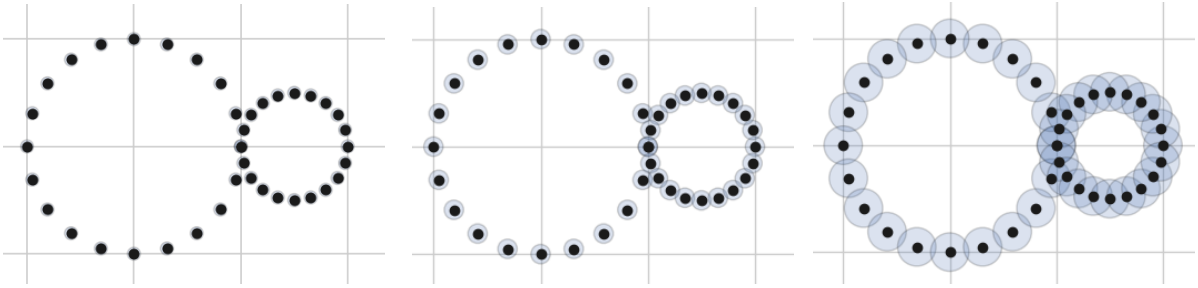
This diagram implies that any topological feature of the Čech complex which appears at r and $2\delta r$ also shows up in the Vietoris-Rips complex at radius $2r$; the Rips complex can capture all features a Čech complex can, for r chosen well. Conversely, if a topological feature appears in the Vietoris-Rips complex at r and $2\delta r$, it must also exist in the Čech complex at radius δr, so any feature captured in the Rips complex is a true topological feature of the data, again for some well-chosen r.

It is clear that the choice of r is important. There might exist a single value which properly extracts all features, but finding this value is very impractical. As a solution, consider *all* possible values of $r \geq 0$. This is done via filtrations. A filtration of a simplicial complex K is a nested family of subcomplexes $(K_r)_{r\in\mathcal{R}}$ where $\mathcal{R} \subseteq \mathbb{R}$ with $\cup_{r\in\mathcal{R}} K_r = K$. The nesting means that $K_r \subseteq K_{r'}$ for $r \leq r'$. The index set $\mathcal{R}$ may be infinite, but in practice, as K is built on finite data, the filtration only changes at a finite number of times, so $\mathcal{R}$ can be taken finite.

The idea of persistence is then as follows: at every $r \in \mathcal{R}$, track which topological features appear and disappear. We say that a feature is born if it appears and dies if it disappears. Prominent topological features will have long lifetimes. The process of growing a Čech complex with respect to its radius is shown in figure 2.14, with the births and deaths of every topological feature tracked in the persistence barcode (fig 2.14f). The barcode shows with lines when features are born and when they die, long lines correspond to prominent features. The Čech complex is shown as it visualizes more clearly than the Rips complex, but the process is the same for both. It is now also time to address the meaning of a 0-dimensional topological feature. A 0-dimensional topological feature is by definition a connected component. Thus, at $r = 0$, every datapoint corresponds to a 0-dimensional topological feature or 0-dimensional topological feature. When r grows, simplices containing more than one point appear, connecting two data points. This reduces two data points to one connected component until there is only one 0-dimensional topological feature left once everything has become connected. Also shown in the figures are 1-dimensional topological features, which we call holes and are present in figure 2.14c.

The persistence barcode is an insightful way of visualizing the persistence information of this example, but can become cluttered when a lot of data points are considered; there is at least 1 line for every individual data point. Alternatively, persistence can be visualized in a persistence diagram. The persistence diagram transforms the bars into points located at $(\mathrm{birth}, \mathrm{death})$. Features which are similar in the persistence barcode will then be very close together, or a point may even lie at the same point. If this happens we say that that point has a multiplicity equal to the amount of features represented by that point. The barcode in figure 2.14f translates to the persistence diagram in figure 2.15. Points far from the diagonal correspond to more persistent topological features. We note that there exist other representations of persistence like persistence landscapes, but these are not used in this thesis.

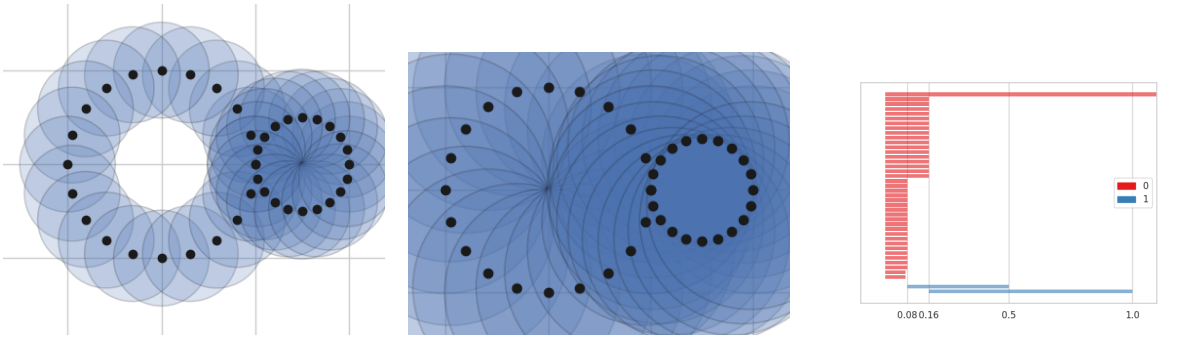
Now revisiting the data shown in figure 2.7 generates persistence diagrams shown in figure 2.16. Observing the data in figure 2.7b did not clearly show its torus nature. We now inspect the persistence diagram in figure 2.16b. We see a lot of topological features, most of which are near the diagonal. Two points however are much farther from the diagonal than other points (from their respective dimensionality); a 1-dimensional feature and a 2-dimensional feature: the signature of a torus.



(a) $r = 0.06$, no balls overlap.

(b) $r = 0.09$, a hole in the right circle appears due to overlapping balls around the data.

(c) $r = 0.16$, a hole in the left circle also appears.



(d) $r = 0.5$, a hole in the right circle disappears.

(e) $r = 1$, all holes have now disappeared.

(f) Persistence barcode showing the lifetimes of the homological features

Figure 2.14: Process of growing the Čech complex on data, with associated persistence barcode.

2.4.5. Previous work on TDA for time series analysis

Now that the mathematics of TDA have been established, the focus shifts to how TDA can be used to analyze time series data specifically.

Most work on time series analysis using TDA first transforms the time series to a high-dimensional embedding. For some time series $x_t \in \mathbb{R}$, this embedding consists of the past μ measurements, spaced τ timesteps apart. This embedding is called the Takens embedding and is based on the Takens theorem which states that for well-chosen μ and τ, the dynamics of the studied system can be fully reconstructed [43]. The Takens embedding can be analysed using TDA to distinguish different dynamics of the time series; to classify the time series. This method can for example differentiate human activity when analysing data of some motion sensor attached to limbs [44].

Another method is to use a sliding window. This method is often given its own name (”sliding window”), though it is equivalent to a Takens’ embedding with parameter $\tau = 1$. As an example take four stock prices at closing and a window of 50 days, to obtain a sample of 50 datapoints in 4 dimensions. One would then apply the TDA process of taking the Rips complex of these points and calculating the corresponding persistence diagram for each window. The final metric would be the Wasserstein distance between consecutive diagrams. This method has been shown to be capable of predicting financial market crashes [45]. The Wasserstein distance given some $p \geq 1$ between two diagrams dgm_1 and dgm_2 is defined as the sum of L^∞ distances between optimally matched points in the diagrams. Thus, with $m \subset \mathrm{dgm}_1 \times \mathrm{dgm}_2$ describing an optimal matching of points,

$$W_p(\mathrm{dgm}_1, \mathrm{dgm}_2)^p = \inf_{\text{matching } m} \sum_{(p,q)\in m} \|p - q\|_\infty^p.$$

Both methods will be explored in this thesis, though the brevity of this subsection and the tone of this sentence foreshadow the fruitless outcome of this exploration.

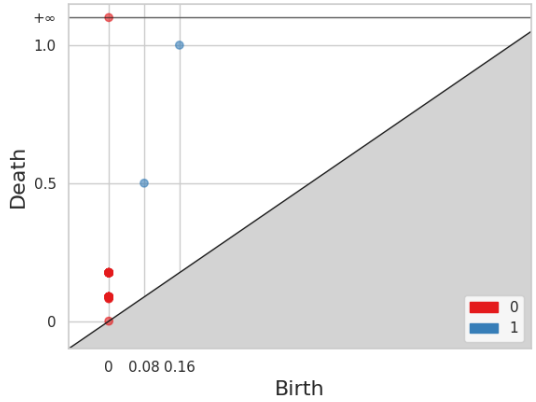
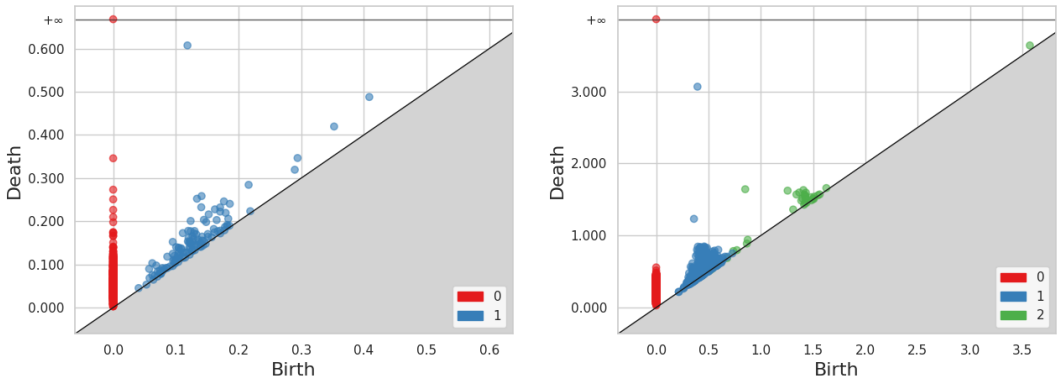


Figure 2.15: Persistence diagram corresponding to the births and deaths of the process shown in figure 2.14



(a) Persistence diagram of circle data in 2.7a.

(b) Persistence diagram of torus data in 2.7b.

Figure 2.16: Persistence diagrams generated from data in figure 2.7 using a Vietoris-Rips filtration.

2.5. Software

The machine learning part of this thesis is implemented in Python through the use of the two most popular packages: TensorFlow [13] and (Py)Torch [46]. The random forests are implemented using Yggdrasil [47]. TDA is performed using the Python package GUDHI [48].

3

Developed methods

This chapter will build new methods for short-term traffic prediction using the existing methods described in the previous chapter. In practice, there is of course a bit of back-and-forth between this chapter and the results in the next chapter. When reading this section there will be references to the next chapter, which refer to some results encouraging that the development process was on a good track.

This chapter will be divided into developed methods in forecasting - arriving at a prediction given some information - and alternative representations of the data - to extract more information from the same data.

3.1. Forecasting methods

This section will implement the **Locality** and **Direction** intuitions into existing (G)NN models to improve their performance. These intuitions will be used to extend the state-of-the-art STNN model to a Propagation-STNN (P-STNN). Additionally, we also modify the new P-STNN model to improve applicability in spatial context, giving a Selfless-Propagation-STNN (SP-STNN). The **Locality** intuition also gives rise to a reformulated GNN framework, as the developed model no longer fits in the framework often found in literature [17].

3.1.1. Direction intuition

Section 2.2 has highlighted how traffic information can travel with the flow of traffic, but also against the flow of traffic if congestion occurs. The **Direction** intuition is that the different directions give rise to a different interaction. As such, different models could be better at predicting traffic from these different interactions.

The implementation of this intuition is achieved through splitting the data into data from traffic sensors upstream of the target sensor and data from sensors downstream of the target. Two regression models are then trained on these split datasets. Having two models means two predictions, which we then combine into a single output using a random forest decision model. Thus we obtain a combined model consisting of regression models and a decision model.

The combined model is generated in two ways; fusing or switching between outputs of the regression. The fusing model is aimed at optimally combining model inputs, whereas switching is an attempt at having the combined result be ”explainable”. Explainable in this sense meaning, for example, ”the model expects congestion due to congestion propagating upstream”. Let X be the space of all possible feature realizations and x_t denote the target quantity to be predicted. The method is based on some grouping of features which we denote $G_i(x)$, where $x_{1:W} \in X$ denotes realizations of input features at timesteps $1, 2, \ldots, W$, where W is the temporal input width of the model. A model for predicting the target feature from the features in the group is made for each group. We will denote these models M_i, refer to them as the regression models and write $M = (M_1, M_2, \ldots, M_n)^T$. Lastly, we also incorporate a random forest classification model $D : X \to \mathbb{R}^n$, where n is the number of groups, which outputs estimated

probabilities that each regression model performs the best, based on all features at $t = W$. We finally obtain "Switch" and "Fuse" models:

$$\text{Switch: } \hat{x}_{W+h} = M_{\arg\max(D(x_W))}(x_{1:W}), \qquad \text{Fuse: } \hat{x}_{W+h} = M(x_{1:W})) \bullet D(x_W). \tag{3.1}$$

There is no theoretical limit on the regression models which the decision model could combine, so in the next chapter we will also include other baselines against which we compare models as regression models for the decision model.

3.1.2. Locality intuition

Another method developed in this thesis relies on the **Locality** intuition that information propagates at some finite speed, so nodes which are located travel time h away, with some margin τ, must be used. Additionally, information on how 'popular' a node is, routing information, must also be used. The latter is necessary to distinguish between nodes, as solely relying on nodes at travel time h will cause the same highway inputs to be used for predicting at every node in a city center. Nodes close to the target node are used for this purpose. Given: n_1 the number of near nodes, n_2 the number of far nodes, A^t a temporal distance matrix at time t and P the physical distances between all nodes, the method for selecting inputs is described in the following algorithm:

Algorithm 1: The algorithm which selects n_1 near nodes and n_2 far nodes to be used for the prediction task, for each target node. $\arg\min_{n_1}$ / $\arg\max_{n_1}$ indicates taking the n_1 smallest/largest arg values.

Data: Target nodes $\mathbf{r}$, possible input nodes v_i, amount of near and far input nodes: n_1, n_2, distance matrices A^t, P at current time t, prediction horizon h and absolute tolerance τ.

Result: Target indexed sets of input nodes $\mathcal{N}_t^r$ for all $r \in \mathbf{r}$.

for $r \in \mathbf{r}$ **do**

$\mathcal{N}_t^r \leftarrow \arg\min_{n_1} A^t_{:,r}$;

$V_r^t \leftarrow \{v_i : (|A^t_{i,r} - h| < \tau)\}$;

$\mathcal{N}_t^r \leftarrow \mathcal{N}_t^r \cup \arg\max_{n_2} \{P_{i,r} | v_i \in V_r^t\}$;

end

Note that this formulation asks for a distance matrix A^t which depends on time. This matrix will store the time it will take information to travel from sensor i to sensor j, based on the shortest path between i and j as calculated from the road network and current speed measurements. We can also take A^t to be a static matrix using, for example, the speed limit instead of current speed measurements. This will be used when applying the method to traffic information traveling against the flow of traffic due to problems in estimating the speed with which this information travels. In that case, the speed used is $20\,\mathrm{km\,h^{-1}}$, based on traffic characteristics described in section 2.2.

3.1.3. SP-STNN

This subsection shows the method improving applicability in space, in contrast to the previous two subsections which improved model performance. Recall from subsection 2.1 that speed data is available at arbitrary locations in a network thanks to Floating Car Data (FCD). Data on flow however is only available at sensor locations. Models described so far have used past measurements of the predicted variable as input, also limiting the locations at which predictions can be made. Thus, when predicting at location A, models also use past measurements at location A, meaning that the model can only be applied to locations where a sensor is placed. A model which does not take past measurements of the predicted variable is free to predict anywhere within a road network. The change; when selecting near sensors to use, omit the target sensor itself as input. We will see in chapter 4 that this will come at the cost of performance.

3.1.4. Distance based GNN

We now zoom out a bit and look at the model developed. Placing the (S)P-STNN model within the context of existing GNNs, we see that it does not fit in the descriptions most commonly found in literature. This change is due to the input selection in algorithm 1. In particular, most classifications of

GNN models are based on the arithmetic of the layers within the model [49], whereas the novelty of (S)P-STNN lies in the inputs of the model. The current general description of GNNs regarding their inputs falls within the class of k-hop message passing frameworks, whereas (S)P-STNN selects its nodes based on edge attributes instead of edge count.

Consider a graph $G = (V, E)$, where V are the nodes and E are the edges, with the edges denoted e_{vu} for the edge going from node v to node u having some weight. Let $I \subset \mathbb{R}$ be an interval. We show the commonly occurring definition of a k-hop shortest path kernel and introduce the new definition of an I-distance shortest path kernel.

Definition 9 (k-hop shortest path kernel)**.** For a node v in a graph G, the k-hop neighbors $\mathcal{N}_{v,G}^{K}$ of v based on shortest path hop count kernel is the set of nodes that have shortest path hop count from node v less than or equal to k. The set $Q_{v,G}^{k}$ denotes the set of nodes which have the shortest path hop count from node v of exactly k.

Definition 10 (I-distance shortest path kernel)**.** For a node v in a graph G, the I-distance neighbors $\mathcal{N}_{v,G}^{I}$ of v based on shortest path distance kernel is the set of nodes u that have shortest path length ℓ_{uv} such that $\ell_{uv} \in I$.

Next, let $\delta > 0$ and let $I = (I_1, I_2, \ldots, I_n)$ be a finite partition of $[0, \delta]$. Using the defined notation, given $x_v \in \mathbb{R}^d$ as the input, the δ distance based message passing framework is defined as follows:

$$
\begin{aligned}
m_v^{I_i} &= \mathrm{MES}_{I_i}\left(\left\{(x_u, \ell_{uv}) : u \in \mathcal{N}_{v,G}^{I_i}\right\}\right) \\
h_v^{I_i} &= \mathrm{UPD}_{I_i}\left(m_v^{I_i}\right) \\
y_v &= \mathrm{COMBINE}\left(\left\{h_v^{I_i} : I_i \in I\right\}\right)
\end{aligned}
$$

Here, MES_{I_i} is a message function which gathers the information of the nodes in the I_i-kernel and on the path from these nodes to the target node v. Note that the message function may be different for each interval I_i. In a practical setting, a neural network needs to know how many input features it will receive at any point in its calculations. Hence, a simple form of the message function is selecting only the information from the first n nodes from its input set. UPD_{I_i} are the update functions, which each produce a prediction $h_v^{I_i}$ for y_v. These predictions are then combined to a final output y_v with the $\mathrm{COMBINE}$ function.

We see that (S)P-STNN falls more naturally in this reformulated framework with messages passing from the closest n_1 nodes within some short interval and from n_2 nodes in the interval of $[h - \tau/2, h + \tau/2]$ sorted on the difference in physical distance which is a node property. The combine step combines the near and far predictions. The random forest decision model is a compound model (trained separately) and is thus not represented in this framework.

3.2. Traffic visualization and summarization

The **Locality** intuition takes local data from all data measured on the network. The local information is made up of a lot of numbers from which we want to draw a conclusion. The previous section describes a new method for arriving at a conclusion in the form of a prediction, this section will describe a new method for visualizing and summarizing the current state of a local part of a graph. Thus, this section is aimed at designing a measure on this data which could be used for analyzing a current state of a part of a network. Ultimately, this quantity might be used to aid in traffic prediction as well. The designed measure can be seen as a time series of a TDA-based transformation of the data. We will compare it with other TDA-based time series tools as well as traditional graph metrics.

3.2.1. Traveltime diagram

We start with a new method for visualizing a local traffic state, by plotting out the trajectories information might follow. Consider a road with a length of $6\,\mathrm{km}$ long and with sensors $s_1, s_2, ..., s_5, s_6$ located at $1\,\mathrm{km}$ intervals. Let $v(s_i)$ be the speed measured at sensor s_i and let $d(s_i, s_j)$ be the travel distance between sensors s_i and s_j. We let sensor s_6 correspond to the target location. The current travel time from sensor s_i to target s_6 on the road can be visualized by calculating $t_1 = d(s_0, s_1)/v(s_1)$ and $t_{i+1} = t_i + d(s_{i+1}, s_i)/v(s_{i+1})$ for $i \geq 1$. We can take these times and may say: "Given the current traffic state, vehicles will be at distance $d(s_i, s_5)$ away from the target at time t_i". This intuitively gives a trajectory, which can be shown by interpolating between the $(d(s_i, s_5), t_i)$ points in 2D space. This is shown in figure 3.2a. Now suppose that a slowdown occurs at sensor s_3, then the travel times of paths which start before sensor s_3 are longer, leading to figure 3.2b

Each curve represents how traffic flows between the target and some starting distance away. Note that the most free-flowing traffic will show as straight lines with slope corresponding to (approximately) the speed limit. When traffic slows down, some lines will stretch. The slowdown is thus not just represented by the arrival time at the target becoming later, but also by an increased gap between lines. A slowdown which has not reached the target, but is incoming, will show as a line which seems to bow upward (a concave curve in the diagram).

The traveltime diagram proposed is somewhat similar to (vertically mirrored) graphs of trajectories traversed by individual vehicles such as shown in figure 3.1; a tool familiar to traffic scientists. This means that it follows similar characteristics as these trajectories, most notably bowing out when slowdowns occur. The graph of trajectories carries more information than the traveltime diagram proposed. For example, it is much easier to estimate the speed of a congestion wave traveling upstream. However, the individual car trajectories trace out many lines and require tracking of every vehicle on a road. Tracking every vehicle requires fine monitoring of every road section, which is a very intensive data-gathering process. Further, there are issues with summarizing trajectories, as their length is variable. This could be due to vehicles leaving/entering a road, but could also be due to some tracking system losing track of a vehicle. The traveltime diagram is thus less informative, but can be generated from readily available sensor data. Also, it is designed to be summarizable, which will be discussed in the next subsection.

So far we have used an interpolated line, without motivation as to why interpolation was used and how the distance between the interpolation points was chosen. This is because there is more data to be used than just speed; flow. With the current setup any road section is equally important. This may sound intuitive, but if there is just a single car on a road section driving slower because they have a preference for driving slower, there would be no hindrance to other traffic.

To incorporate how 'important' a section of road is, we interpolate that section using some parameter F such that the Euclidian distance between each interpolation point is $F/f(s_i)$, where $f(s_i)$ is the flow at sensor s_i. In a more practical setting, we limit the interpolation spacing with some lower and upper bounds $r_{\min}$ and $r_{\max}$. The lower bound is to limit the amount of points in the diagram, to compute the persistence diagram quicker. The upper bound is there to stabilize the algorithm in cases of low flow. We are almost ready to apply the procedure to data, but sensors are not spaced as nicely in the real world. As such, we shift the lines such that they intersect the distance axis at some regular interval, taken to be $500\,\mathrm{m}$ in the data study which will follow. We summarize this procedure in algorithm 2. This regularization allows diagrams generated from different road sections to be compared to each other.

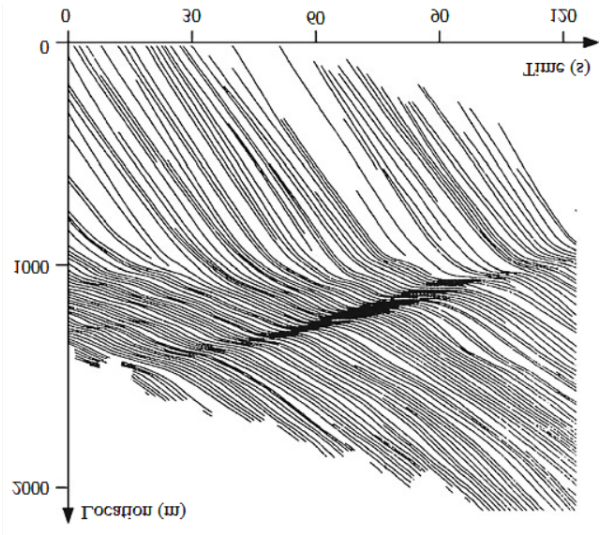
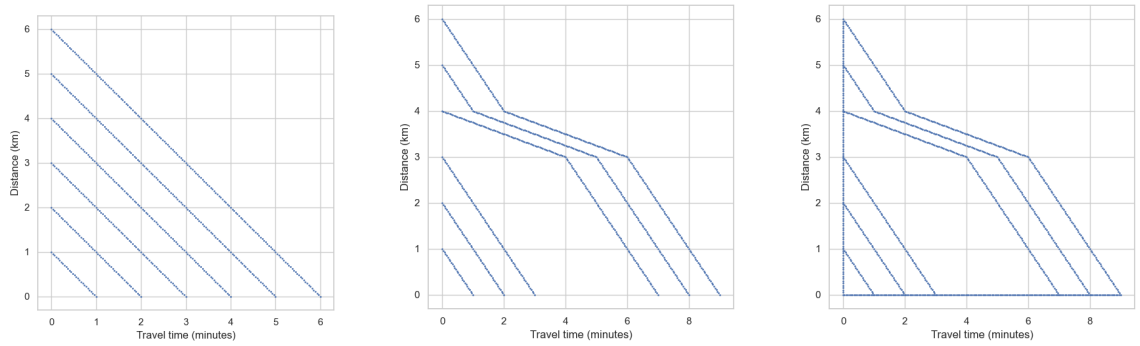


Figure 3.1: Vertically mirrored trajectories traversed by individual vehicles. From [50].



(a) All speeds are $60\,\mathrm{km\,h^{-1}}$.

(b) Speed at sensor s_3 is $15\,\mathrm{km\,h^{-1}}$.

(c) Travel time diagram with interpolated axes.

Figure 3.2: Travel time diagrams.

This would not be possible if we used sensor spacing as physically placed in the real world, as some holes would already be larger than others even in freely flowing traffic states.

Note that the lines in the traveltime diagram spread apart when speeds drop; the gap between them increases. Now connect the $d = 0$ and $t = 0$ axes in the same interpolated way to 'close' the lines, as shown in figure 3.2c, to obtain holes instead of gaps. This hardly changes the visual appearance of the diagram, but is an important step for the summarization in the next subsection. Lastly, the diagram is not yet finished, but the motivation for the final piece is best motivated by describing the summarization. As such, the next subsection will slightly alter the traveltime diagram from what is presented in this subsection.

3.2.2. TDA transformation

We mentioned before when comparing the traveltime diagram with individual vehicle trajectories that the traveltime diagram is designed to be summarizable. From having read the title of this subsection and the second part of chapter 2, the reader can already guess what mathematical tool is used for this; TDA.

The previous subsection gives us a (near-final) traveltime diagram, whose lines spell out holes. At a slowdown, these holes can change in size, or split in two through the pinching seen in figure 3.2b. This allows the use of TDA for analyzing the diagram. Before we do this however, note that due to

Algorithm 2: The algorithm which creates a flow-interpolated travel time diagram on a simple path (straight road). Line is an array of vectors, Line_j^i denotes the i-th element of the vector at position j in the Line array. The operator $\oplus$ denotes appending/concatenating arrays.

Data: On a simple path with N sensors: target node with sensor s_N, other sensors $s_1, s_2, \ldots, s_{N-1}$, with measured speed $v(s_i)$, flow $f(s_i)$ and travel distances between sensors $d(s_i, s_j)$. Let $v_{\max}$ be the speed limit on the road and let $d(s_N, s_{N+1}) = d$ be the distance between the target and the dummy sensor. Let the sensors be ordered by increasing distance from s_N. Let the interval at which we want the lines to be spaced at the distance axis be R and let F be a scaling factor for the interpolation distance. Let $0 < r_{\min} < r_{\max}$ be limits on the interpolation spacing.

Result: Pointcloud in $\mathbb{R}^2$; the flow-interpolated travel time diagram. When used in algorithm 3: sequence of lines yet without interpolation between its ends.

s_{N+1} is the dummy sensor;
$v(s_{N+1}) \leftarrow v_{\max}$;
$f(s_{N+1}) \leftarrow \max_i f(s_i)$;
$\mathrm{Line} \leftarrow$ empty array;
$\mathrm{Lines} \leftarrow$ empty sequence;
$\mathrm{Pointcloud} \leftarrow$ empty array;
for $i \in \{1, \ldots, N\}$ **do**
  $\Delta t \leftarrow d(s_i, s_{i+1}) / v(s_{i+1})$;
  $\mathrm{Linepiece} \leftarrow \mathtt{interp}(\mathtt{start} = (0, d(s_i, s_{N+1}))^T, \mathtt{stop} = (\Delta t, d(s_{i+1}, s_{N+1}))^T, \mathtt{spacing} = \min(\max(F/f(s_{i+1}, r_{\min})), r_{\max}))$;
  $\mathrm{Line} = \mathrm{Line} \oplus \mathrm{Linepiece}$;
end
for $y \in \mathit{interp}(\mathit{start} = d(s_1, s_{N+1}), \mathit{stop} = d(s_N, s_{N+1}), \mathit{spacing} = R)$ **do**
  $j \leftarrow$ index of point closest to y in travel distance;
  $\Delta t \leftarrow \mathrm{Line}_j^1$;
  $\mathrm{Pointcloud} \leftarrow \mathtt{shift\_left}(\mathrm{Line}_{\geq j}, \mathtt{by} = \Delta t)$;
  $\mathrm{Lines}.\mathtt{append}(\mathtt{shift\_left}(\mathrm{Line}_{\geq j}, \mathtt{by} = \Delta t))$;
end
Add points on the distance and time axes with spacing $\min(\max(F/f(s_{N+1}, r_{\min})), r_{\max})$;

the current setup, any hole splitting due to a slowdown of traffic **at** the target sensor is not captured in this diagram. As such, we add a dummy sensor, whose speed is always the speed limit of the road. From this 2D point cloud we generate Vietoris-Rips complexes, filtrations and ultimately a persistence diagram, as described in subsection 2.4.4. The persistence diagrams corresponding to the traveltime diagrams described here are shown in figure 3.3. Note that point 2 in the persistence diagrams has a multiplicity of 4 in figure 3.3d and 5 in figure 3.3c. The holes and corresponding topological features are indicated with the red numbers.

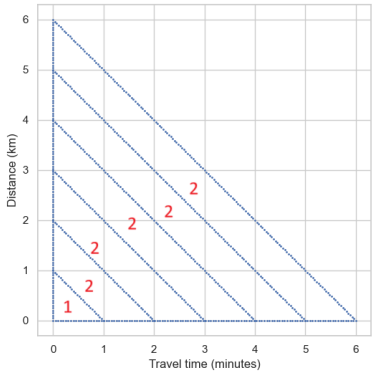
We now have a way to generate persistence diagrams from the data, but we wish to capture this information into a single number so that we may compare it to other metrics. We do this by taking the sum of lifetimes of one-dimensional features in the persistence diagram, so we summarize to

$$\sum_{(b,d) \in \mathrm{dgm}_t} d - b,$$

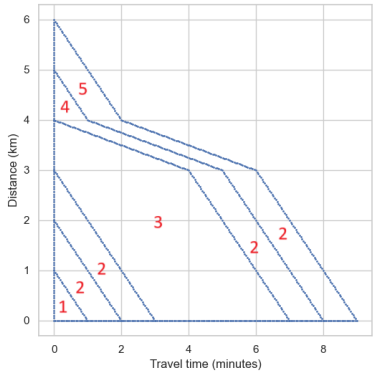
where $(b, d) \in \mathrm{dgm}_t$ denotes the birth and death coordinates of a point in the persistence diagram of the traveltime diagram at time t. We call this quantity the ”Travel Lifetime”.

3.2.3. Generalization to merging paths

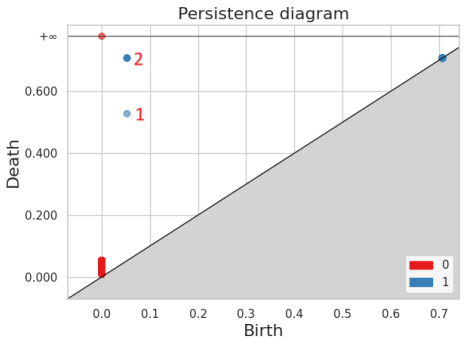
So far, a procedure has been proposed to study simple paths; a section of a road without junctions. This limits the useful scope of the pipeline, but it can be generalized to be applied to a road with an arbitrary number of merging junctions. A merging junction in the setting of studying congestion traveling upstream corresponds to a diverging junction when driving on the road.



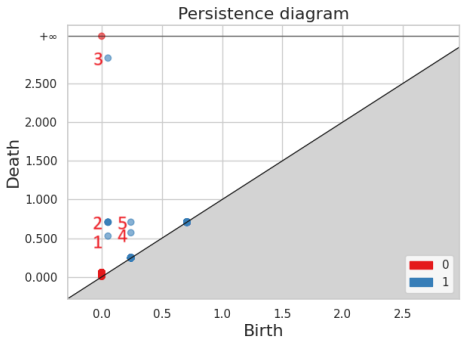
(a) Numbered holes in the traveltime diagram.



(b) Numbered holes in the traveltime diagram in case of a slowdown at sensor s_3.



(c) Persistence diagram corresponding to traveltime diagram with all speeds being $60\,\mathrm{km\,h^{-1}}$.



(d) Persistence diagram corresponding to traveltime diagram with a slowdown at sensor s_3.

Figure 3.3: Traveltime diagrams and their corresponding persistence diagrams.

Suppose there are two starting roads, which merge and then continue towards the target, like shown in figure 3.4, which merge at distance m from the target; 8. The intuition behind the generalization is that the distance traveled d_2 in the figure is a different kind of distance than d_1. As such, we use another dimension to separate their influences. As an example, generate the lines in traveltime diagrams for simple paths d_1 and d_2. Shift the lines of d_2 such that their points at distance m match the time coordinates of the lines of d_1 at distance m. We then embed the d_2 traveltime diagram in 3D, calling the new distance coordinate $\tilde{d}$, and rotate around the $(\cdot, m, 0)$ axis. After this, chop off everything lying in the half-space $\tilde{d} < 0$ and combine with the 3D embedding of the d_1 travel time diagram. The final step is closing the gaps. As such, an interpolated line between all end or start points of a line is added, including the point corresponding to where the roads have merged. We summarize this process in algorithm 3, and show its steps in figure 3.5.

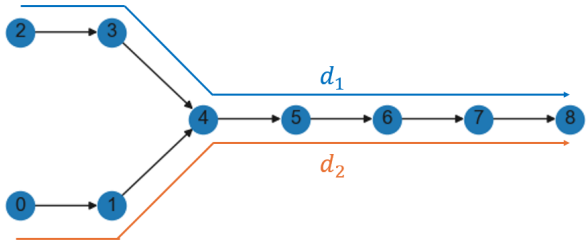


Figure 3.4: A simple example of a merging situation.

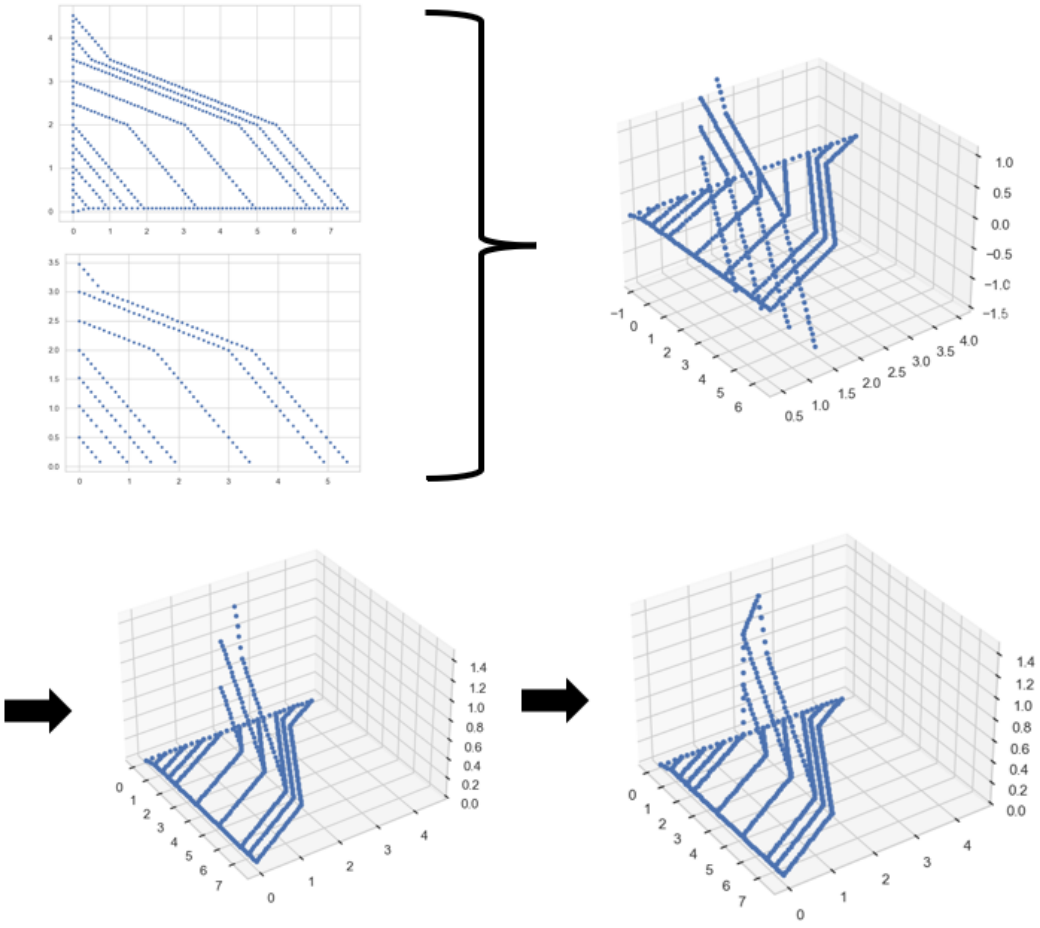


Figure 3.5: Algorithm 3; the process of combining the outputs of algorithm 2 of 2 paths which merge at distance 2 from the dummy sensor.

Algorithm 3: The algorithm which creates a flow-interpolated travel time diagram on a path with an arbitrary amount of merging roads. Line is an array of vectors, Line_j^i denotes the i-th element of the vector at position j in the Line array. The operator $\oplus$ denotes appending/concatenating arrays.

Data: We denote N the number of paths to the dummy sensor, M the number of sensors and take parameters R and F as used in algorithm 2. Let $v_{\max}$ be the speed limit at the target sensor, let $f(s_i)$ denote flow measures at sensor s_i and take travel distances between sensors $d(s_i, s_j)$. Let $0 < r_{\min} < r_{\max}$ be limits on the interpolation spacing.

Result: Pointcloud in $\mathbb{R}^{N+1}$; the flow-interpolated travel time diagram.

s_{M+1} is the dummy sensor;
$v(s_{M+1}) \leftarrow v_{\max}$;
$f(s_{M+1}) \leftarrow \max_i f(s_i)$;
$\mathrm{Paths} \leftarrow$ `sort`$(\mathrm{Paths}, \texttt{by} = \text{"travel distance"}, \texttt{order} = \mathrm{desc})$;
$\mathrm{Initial\_lines} \leftarrow$ apply algorithm 2 to the longest path in Paths;
Pad all lines in $\mathrm{Initial\_lines}$ with zeros such that each point lies in $\mathbb{R}^N$;
$\mathrm{Interp\_lines} \leftarrow$ interpolated lines between all endpoints and starting points of $\mathrm{Initial\_lines}$;
$\mathrm{Pointcloud} \leftarrow$ empty array;
$\forall_{i \le M} \mathrm{Sensor\_locations}_{s_i} \leftarrow d(s_i, s_M) \oplus \mathbf{0}^{N-1}$ # Keeps track of the sensor location in $\mathbb{R}^N$;
for $i \in \{2, ..., N\}$ **do**
  $\mathrm{New\_lines} \leftarrow$ apply algorithm 2 to the path Paths_i;
  $\mathrm{Merge\_path}, s_{\mathrm{merge}} \leftarrow$ `get_merge_information`$(\texttt{path} = \mathrm{Paths}_i, \texttt{with\_paths} = \mathrm{Paths}_{<i})$;
  $\mathrm{Merge\_path} \leftarrow$ find path in $\mathrm{Paths}_{<i}$ with which Paths_i merges first;
  s_{merge} find sensor at which the new path merges with paths in $\mathrm{Paths}_{<i}$;
  for *Sensors s_j on path i up to the merging sensor* **do**
    Set $\mathrm{Sensor\_locations}_{s_j}$ to $\mathrm{Sensor\_locations}_{s_{\mathrm{merge}}}$ plus $(0, ..., d(s_j, s_{\mathrm{merge}}), ..., 0)^T$ where the i-th dimension is non-zero;
  end
  for $j \in \{1, 2, ..., |New\_lines|\}$ **do**
    $\mathrm{Line} \leftarrow \mathrm{New\_lines}_j$;
    $\mathrm{Line} \leftarrow$ `shift_down_in_distance`$(\mathrm{Line}, \texttt{by} = d(s_{\mathrm{merge}}, s_{M+1}))$;
    Pad Line with zeros such that each point lies in $\mathbb{R}^{N+1}$, with only dimensions 1 (time) and $i+1$ (new distance) being nonzero;
    $\mathrm{Line} \mathrel{+}= (0, \mathrm{Sensor\_locations}_{s_{\mathrm{merge}}}^T)^T$;
    Chop off all points in Line with any negative component;
    if *Line not empty* **then**
      Attach Line in dimension 1 (time) to j-th line from lines due from $\mathrm{Merge\_path}$ which start at distance greater than $\|\mathrm{Sensor\_locations}_{s_{\mathrm{merge}}}\|_1$;
    end
    $\mathrm{Pointcloud} \leftarrow \mathrm{Pointcloud} \cup \mathrm{Line}$;
  end
  $\mathrm{Interp\_lines} \leftarrow \mathrm{Interp\_lines} \cup$ Interpolate between all starting points of $\mathrm{New\_lines}$ with spacing $\min(\max(F/f(s_{N+1}, r_{\min})), r_{\max})$;
end
$\mathrm{Pointcloud} \leftarrow \mathrm{Pointcloud} \cup \mathrm{Interp\_lines}$;

4

Results

This chapter provides results of the developed methods in chapter 3. We repeat the idea of local information flows incorporated into the models in this chapter. This idea consists of two parts:

1. The manner in which information flows, mainly the speed with which it moves, depends on the direction of the flow. (**Direction**)
2. Relevant information for making a prediction with horizon h ahead is contained within an interval of size h in time. (**Locality**)

Both parts are first tested for viability with a relatively simple neural network in section 4.1, after which results from incorporation into a state-of-the-art GNN in section 4.2 are shown.

4.1. Preliminary study

To test the premise of the intuitions above, we set up a preliminary study concerned with predicting congestion on highways during the morning rush. The target location is the eastward direction of the A20 at the hectometer marker 41.3. This target location is named 'RWS01_MONIBAS_0201hrr0413ra' in the NDW data. The aim is to predict traffic speed in peak hours, such as the morning rush, 10 minutes ahead. Traffic speed and intensity are extracted every minute from 05:30:00 to 09:59:59 on all business days between 2023-03-06 and 2024-06-07.

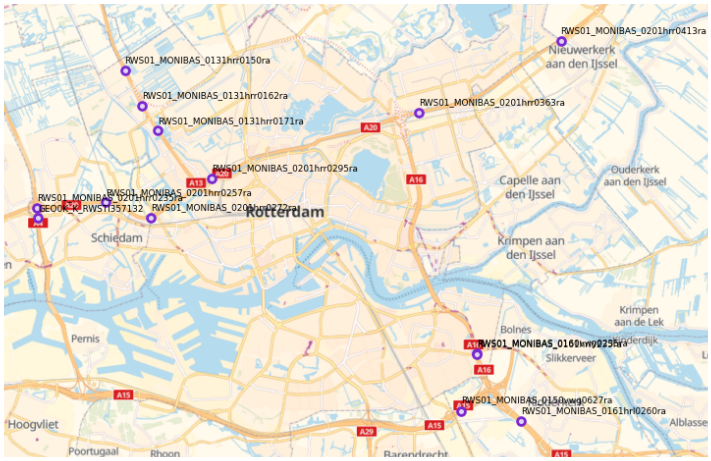
Two neural network models for upstream and downstream information directions are trained. These are also combined with and compared to much simpler predictors; the naive estimator and historic mean. Both will be specified further. The combination is explained in subsection 3.1.1. We also compare the split neural network models with a model trained directly on all data.

4.1.1. Submodels

As mentioned, 5 predictors are distinguished:

NN upstream The upstream neural network model has as input sensors the measured speed and flow at the target location and sensors between $11.8\,\mathrm{km}$ and $16.6\,\mathrm{km}$ upstream. This, combined with the prediction horizon of $10\,\mathrm{min}$, is in accordance with traffic traveling at speeds between $70\,\mathrm{km\,h^{-1}}$ and $100\,\mathrm{km\,h^{-1}}$. There is an additional selected location right after where the A16 and A20 merge, to include information on routing from both roads to the target location. Minutely measurements of the last $25\,\mathrm{min}$ were used, so that the input data is of size 14 locations $\times$ 25 minutes $\times$ 2 features per sample.

NN downstream The downstream neural network model has as input sensors the measured speed and flow at the target location and sensors between $3.1\,\mathrm{km}$ and $6.5\,\mathrm{km}$ ahead, as shown in figure 2.1b. This was chosen in accordance with the typical speed of traffic waves propagating backward, which is about $20\,\mathrm{km\,h^{-1}}$[24] and the prediction horizon of $10\,\mathrm{min}$. Minutely measurements



(a) The locations used for the ML upstream model

(b) The locations used for the ML downstream model.

Figure 4.1: Data locations used for the ML upstream and ML downstream models.

of the last $25\,\mathrm{min}$ were used, so that the input data is of size 4 locations $\times\, 25$ minutes $\times\, 2$ features per sample.

NN direct The direct neural network model uses all input sensors. Minutely measurements of the last $25\,\mathrm{min}$ were used, so that the input data is of size 17 locations $\times\, 25$ minutes $\times\, 2$ features per sample.

Naive The naive predictor receives only one input: the measured speed at the target location. There is no arithmetic, the model predicts the speed at time h in the future $\hat{v}_h$ as simply being the measured speed v_0, so $\hat{v}_h = v_0$.

Historic mean The historic mean receives as input the time of day and the day of the week. It will return the historic average of the requested minute of the week over the training data. That is: $\hat{v}_h = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} v_i$ where $\mathcal{I}$ is the set of all measurements at the same minute and day of the week.

The types and sizes of layers used are shown in table 4.1.

Table 4.1: Machine learning architectures used in the ML backward and ML forward models.

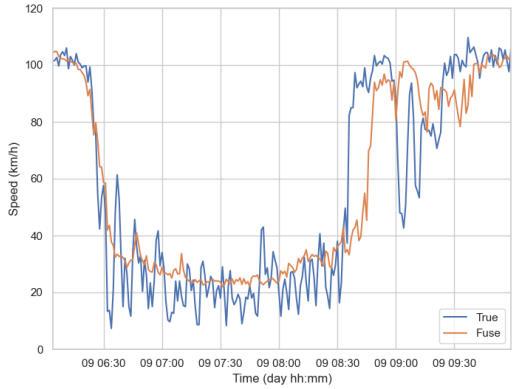
Layer	ML upstream	ML downstream
1	16 ReLu nodes	4 GRU nodes
2	16 ReLu nodes	4 GRU nodes
3	4 GRU nodes	4 GRU nodes
4	4 GRU nodes	4 ReLu nodes
5	4 GRU nodes	1 linear node
6	4 ReLu nodes	
7	1 linear node	

4.1.2. Results

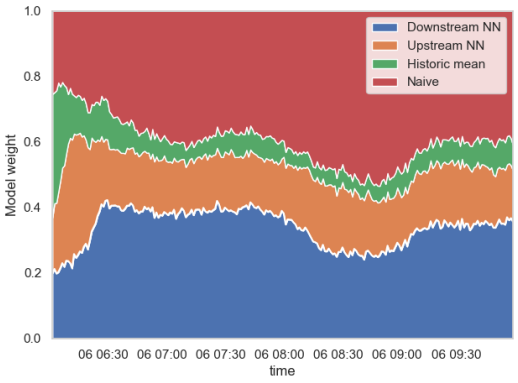
As stated in the case introduction, we considered the models with a prediction horizon of $10\,\mathrm{min}$. The combined models are compared with all submodels which make it up and with the direct NN model. In figure 4.2a true and predicted speeds are shown for a day on which congestion occurred. Table 4.2 shows the mean absolute error and mean squared error scores for all models. The results for two fusing models are shown: one combining the upstream and downstream NN models and the naive and historic mean baselines, and another only combining the upstream and downstream NN models. The latter combination is shown to isolate the premise of the **Direction** intuition. Both fusing models outperform their regression models. Interestingly, the direct ML approach did not perform better than the regression models. From the result we can conclude that within this preliminary study, creating split

models for each information flow direction and then combining outputs increases model performance.

We may also inspect the output probabilities/weights of the decision model. Figure 4.2b shows the predicted probability that some regression model is the best-performing model at any time of the morning rush hour, averaged over all days in the data. Congestion usually starts around 6:15-6:30 for the morning rush hour. During this period we see that the upstream NN model is the best predictor, which indicates that congestion usually starts further upstream and then propagates against traffic to the sensor location which is studied.



(a) True and predicted traffic speeds using the fuse model based on all four submodels and a prediction horizon of $10\,\mathrm{min}$ on data of one day in the testing set.



(b) Decision model output: predicted probability that some regression model is the best-performing model at any time of the morning rush hour, averaged over all days in the data.

Table 4.2: Results of the switch and fuse models and the individual regression models from subsection 4.1.1.

	Training			Testing	
Model	MAE	MSE		MAE	MSE
NN upstream	10.7	317.8		13.3	456.4
NN downstream	9.9	268.6		10.4	288.9
Historic mean	21.5	790.4		22.3	838.7
Naive	10.0	307.6		11.2	359.6
Switch	6.7	162.6		9.7	300.1
Fuse	6.8	146.3		9.2	236.2
NN direct	10.4	296.6		10.9	342.5
Fuse (NN up- & downstream)	8.1	184.9		10.1	264.1

4.2. Extending STNN

In this section, the STNN model introduced in chapter 2 is compared to the preliminary study. Later, it is extended first to a propagation-STNN (P-STNN) by implementing a new method for dynamic local neighborhood selection and by splitting, modeling and combining information flows. Another extension to a selfless-propagation-STNN (SP-STNN) is later implemented. This second extension improves how the model can be used in the real world.

4.2.1. Comparison with preliminary work

As a comparison with the preliminary work, the STNN model was also evaluated on the dataset from the previous section. Note that the preliminary dataset already had inputs situated at around $10\,\mathrm{min}$ of travel time away from the target. As such, all nodes were used in a static way (no dynamic travel time information). Results are shown in table 4.3, where the input models for the decision model were constrained to forward, backward and naive models. The individual STNN models and their combined model outperform their respective preliminary models, though the improvement is smaller. This could be due to the upstream and downstream STNN models predicting closer to each other, so that there is

less freedom for the decision model to work with. Interestingly; combining not just both STNN models, but also the naive and historic mean predictors worsens the scores, we will comment on this later in the discussion chapter. A direct NN model is not included, as both the preliminary results and crude testing with STNN showed it performing worse than the best of individual upstream and downstream NN models.

Table 4.3: Results of the switch and fuse models and the individual regression models from subsection 4.1.1, where STNN is used as NN model.

	Training			Testing	
Model	MAE	MSE		MAE	MSE
STNN upstream	9.4	226.9		10.4	251.7
STNN downstream	9.0	212.0		9.5	224.8
Historic mean	21.7	790.1		22.4	836.0
Naive	10.0	309.4		11.2	360.3
Switch	7.4	191.7		9.9	297.8
Fuse	7.3	168.8		9.3	235.2
Fuse (STNN up- & downstream)	8.1	193.5		9.3	224.7

4.2.2. Larger datasets

To test the capabilities of the P-STNN and SP-STNN models, more challenging datasets are used. Both are concerned with predicting traffic in the city of Groningen, but their levels of detail vary. The datasets are described in table 4.4. Note the properties 'Selected sensors' and 'Usable data size'. Selected sensors indicates that some sensors were omitted due to limited data availability, both missing flow data and having too few data points in time. Some of the selected sensors still missed data at specific points in time, which caused parts of the data to be too small in a temporal sense to be input for the model and have an output to compare against, which limited the final amount of data available; the usable data size. The NDW dataset consists of NDW data only, whereas the MVVS dataset is an aggregation of multiple data sources. The graph corresponding to each dataset and its usable sensor locations are shown in figure 4.3 and, in the Appendix due to the size of the figure, A.2. Figure 4.3 also shows an example of how algorithm 1 selects nodes to use as input.

Table 4.4: Datasets used for the city of Groningen.

Properties	Dataset	
	NDW	MVVS
Spanned time	Morning rush hour (5:30-10:00) of workdays between 3-6-2024 and 23-6-2024	Days between 27-7-2024 and 21-08-2024
Temporal resolution	1 minute	1 minute
Usable sensors	28	56
Usable data size	3551 samples of 56 variables	19312 samples of 112 variables

We now test the developed algorithm 1 which was shown in subsection 3.1.2. This algorithm implements the **Locality** intuition. The test is evaluated on the NDW dataset by testing the performance of downstream P-STNN models on sensor locations on which the model has been trained and on which it has not been trained. We distinguish a static P-STNN model (denoted s-P-STNN), in which travel times between nodes are calculated from speed limits and which thus has a static adjacency matrix $A^t = A$ for all t, and a dynamic P-STNN model (denoted d-P-STNN), in which travel times are calculated from current speeds at the time of inference. Note that the speed to be used in an upstream model is based on the traffic wave speed described in section 2.2. Dynamic estimation of this wave speed between two minutely measurements proved unusable as will be explained in the Discussion and conclusion, so the downstream models were only run with a static adjacency matrix. The evaluation is carried out on nodes $10, 11, 12$ and 13 of the NDW graph, highlighted in figure A.1 in the Appendix. The out-of-sample results are generated using leave-one-out cross-validation (LOOCV) and then averaged. The in-sample results are generated using the same models as in the out-of-sample evaluation, so trained on three nodes, but evaluated on all nodes on which it was trained and then averaged. The results

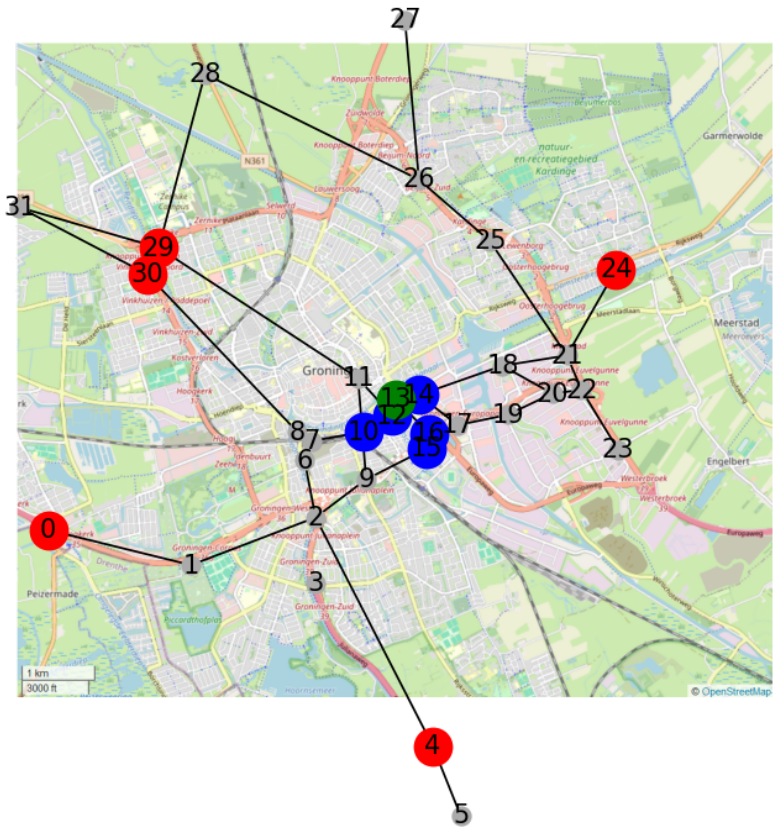


Figure 4.3: Network for the NDW dataset. Nodes are colored according to selection with algorithm 1. Target node 13 is shown in green. 5 near nodes were selected, shown in blue: $10, 12, 14, 15, 16$. Also, 5 far nodes were selected, shown in red: $0, 4, 24, 29, 30$.

are shown in table 4.5a. Note that the in-sample results are nearly indistinguishable, but that the dynamic P-STNN model transferred better with a 5.6% improvement in MSE. The static P-STNN model transferred badly, which could be due to its input containing confusing data.

Also shown in table 4.5 are results from downstream prediction performance, as well as flow prediction performance in both directions. Note that the downstream model is based on traffic waves propagating against the flow of traffic with a fixed speeds of $20\,\mathrm{km\,h^{-1}}$, so there is no dynamic P-STNN model.

4.2.3. SP-STNN

Recall from subsection 2.1 that speed data is available at arbitrary locations in a network thanks to Floating Car Data (FCD). Data on flow however is only available at sensor locations. All models described so far have used past measurements of the predicted variable as input, limiting the locations at which predictions can be made. A model which does not take past measurements of the predicted variable is free to predict anywhere within a road network. The change in code to P-STNN is small; when selecting near sensors to use, omit the target sensor itself as input. This gives a ”selfless” model, which we call SP-STNN (Selfless Propagation STNN). SP-STNN predicting flow is tested on both the NDW and the MVVS datasets. The results are shown in tables 4.6 and 4.7 respectively. Note that, as the model is ”selfless”, no in-sample test is performed as this is not representative of real-world use of the model. Fusing and switching results on the NDW dataset are found in the Appendix in table A.2 and results for the MVVS dataset are shown in table 4.7. The best-performing fuse model on the MVVS dataset is the model using both static models. The predictions of this fuse model and both static models are shown in figure 4.4.

Table 4.5: Testing performance of upstream or downstream STNN and P-STNN models on speed or flow at target nodes $11, 12, 13$ and 14 in the NDW dataset. s-P-STNN stands for static P-STNN, d-P-STNN stands for dynamic P-STNN.

(a) Speed, upstream

Model	In sample		Out of sample	
	MSE	MAE	MSE	MAE
STNN	40.7	4.52	62.5	6.03
s-P-STNN	38.9	4.30	91.5	7.46
d-P-STNN	40.3	4.45	59.0	5.95

(b) Flow, upstream

Model	In sample		Out of sample	
	MSE	MAE	MSE	MAE
STNN	$2.17 \cdot 10^5$	356	$2.6 \cdot 10^5$	386
s-P-STNN	$2.15 \cdot 10^5$	354	$2.98 \cdot 10^5$	407
d-P-STNN	$2.17 \cdot 10^5$	356	$2.63 \cdot 10^5$	405

(c) Speed, downstream

Model	In sample		Out of sample	
	MSE	MAE	MSE	MAE
STNN	39.2	4.38	58.9	5.66
s-P-STNN	38.8	4.33	116	9.11

(d) Flow, downstream

Model	In sample		Out of sample	
	MSE	MAE	MSE	MAE
STNN	$2.16 \cdot 10^5$	354	$3.66 \cdot 10^5$	480
s-P-STNN	$2.17 \cdot 10^5$	356	$2.72 \cdot 10^5$	403

Table 4.6: SP-STNN results on flow prediction on the NDW dataset.

Model	Upstream		Downstream	
	MSE	MAE	MSE	MAE
Static-SP-STNN	$3.13 \cdot 10^5$	432	$4.01 \cdot 10^5$	508
Dynamic-SP-STNN	$5.14 \cdot 10^5$	580		

4.2.4. Running time

Important for implementation is how fast the models train and evaluate. All ML tasks in this thesis were run on an Intel i7-12700H CPU. Training (SP-)STNN takes $9(3)\,\mathrm{ms\,epoch^{-1}\,sample^{-1}\,node^{-1}}$ on an Intel i7-12700H. This corresponds to a mean training time of approx. $2.5\,\mathrm{h}$ for the NDW dataset. Data preparation and random forest training times were negligible compared to the training task. Inference takes $5(1)\,\mathrm{ms\,sample^{-1}\,node^{-1}}$.

4.3. Previous work on TDA for time series analysis

We now shift focus from prediction results to the summarization of a local traffic state through the traveltime diagram introduced in chapter 3.

Earlier in this thesis the use of a so-called Takens embedding is briefly explained. The method of using a Takens embedding is particularly powerful when applied to time series which show some periodicity. This is shown in the following example.

Consider the traffic flow measured every 5 minutes on Mondays on some sensor location. We use $\mu = 6$ and $\tau = 40$, such that $(\mu + 1)\tau = 280$, an entire day, so that we can capture the expected periodicity of a day. The most important features of the embedding can be shown in a 2D figure through embedding once more with principal component analysis (PCA). This is purely illustrative, its result is shown in figure 4.5b. Again, a periodic signal with a period of a day ($280 \cdot 5$ minutes) would be expected. The time of day can be encoded as the path of 24-hour clock: a circle, $t \mapsto (\cos(t/(24{\cdot}60)), \sin(t/(24{\cdot}60)))$, with t in minutes. A circle has a hole in the middle; a 1-dimensional topological feature, so a prominent 1-dimensional topological feature is expected when studying the persistence diagram, which is also shown in figure 4.5c. In this example we already knew what period to expect and we set μ and τ accordingly. There exist automated methods for estimating appropriate μ and τ [51].

Another method has been mentioned; using a sliding window. This method was tested for viability in the traffic problem. We show an example of such a test; we take a window of 25 measurements, each at a single minute. This gives 25 points for a speed and flow measurement. Each window is transformed to a persistence diagram and these diagrams are compared using the Wasserstein distance. Note that we are taking overlapping windows, so the first datapoint in the sequence of Wasserstein distances is the Wasserstein distance between diagrams generated from measurements $1-25$ and measurements $2-26$. The result is shown in figure 4.6. No correlation to past, current of future traffic states could

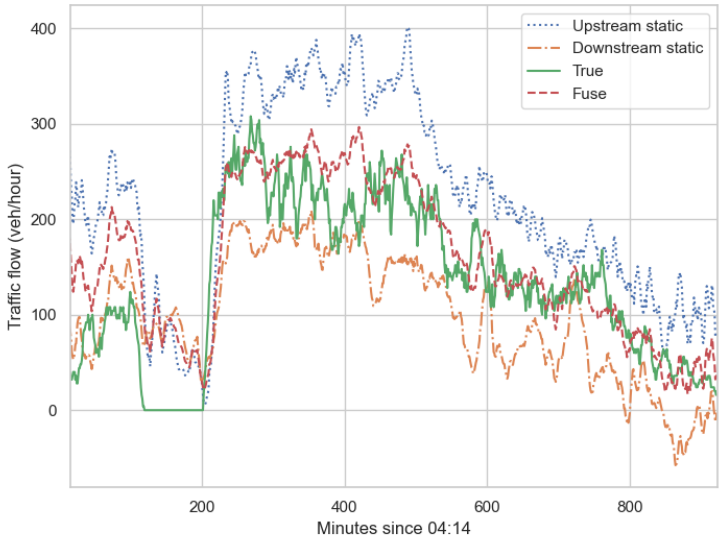


Figure 4.4: Predicted traffic flow on the 5th of August 2024 at some location in the city of Groningen. The fusing model, the predictions being fused and the measured flow are shown.

Table 4.7: SP-STNN results on flow prediction on the MVVS dataset.

Model	MSE	MAE
Upstream static-SP-STNN	$4.85 \cdot 10^4$	165
Upstream dynamic-SP-STNN	$5.83 \cdot 10^4$	180
Downstream static-SP-STNN	$5.01 \cdot 10^4$	172
Fuse (static upstream, static downstream)	$4.56 \cdot 10^4$	162
Fuse (dynamic upstream, static downstream)	$4.80 \cdot 10^4$	164

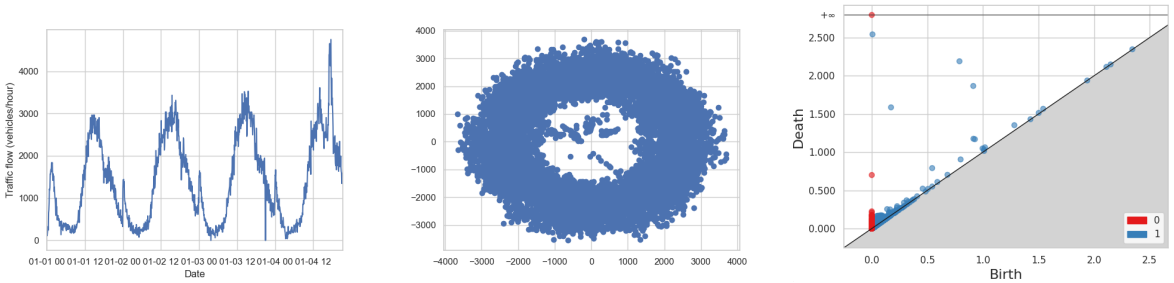
be found, nor was the sequence of Wasserstein distances useful as an input to a neural network.Thus, though these methods have shown to be useful in some applications, they have not been of use in the scope of this thesis.

4.4. Travel lifetime

In section 3.2 we introduced the traveltime diagrams and travel lifetime. Using this summarization, we study a data example. We take a highway section close to what was used in section 4.1 on the 27th of August 2024. The sensor placement and their distances can be found in figure A.3. We study the road to track information propagating upstream; congestion propagating upstream at $20\,\mathrm{km\,h^{-1}}$ [24] would take $8.4\,\mathrm{min}$ to traverse the section.

We show the results for the entire day in figure 4.7a. We also zoom in on the time when congestion starts upstream by showing some of the sensor speeds in figure 4.7b and the zoomed-in target speed and the travel lifetime in figure 4.7c. It is most important to note that the travel lifetime starts to increase before the congestion hits our target, and that it is low when the road is calm (between 00:00 and 05:00). As such, we conclude that the design of the travel lifetime succeeds in capturing both the incoming congestion wave and the extremely calm period.

We call the ”Travel Lifetime” of a traffic state on a simple path the resulting number of the pipeline: traveltime diagram to persistence diagrams to sum of lifetimes. In figure 4.7, this travel lifetime is compared to a graph metric; the diameter of the graph. The diameter of a graph is given by the length of the longest shortest paths, where the length of a path is determined by its travel time as calculated from the measured speeds. The windowing method was already shown in figure 4.6 and is not included



(a) Traffic flow on Sundays on some sensor location.

(b) 2D PCA embedding of the Takens embedding of the flow in figure 4.5a with parameters $\mu = 5, \tau = 48$.

(c) Persistence diagram of the Takens embedding in the previous figure.

Figure 4.5: Process of time series TDA through a Takens embedding.

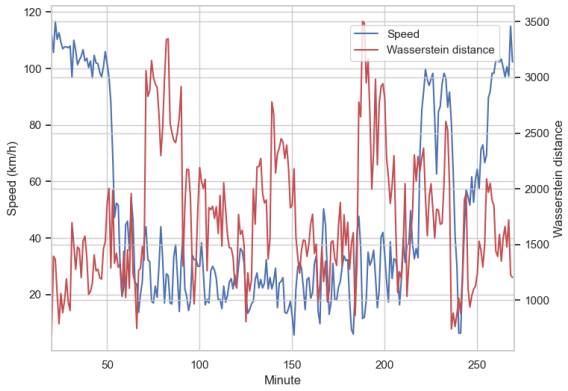


Figure 4.6: Windowing and Wasserstein distance method applied to traffic data.

here because it did not yield noteworthy results.

4.4.1. Summarization value

A method for visualizing and summarizing local traffic information has been presented. We may ask now whether the travel lifetime captures a ”new” aspect from the data as opposed to what the neural networks used could already extract. A simple test is designed, with the same sensors as in section 3.2.1 but the data now spans an entire month. Of this month, we split the data like we did earlier: 70% training, 15% validation and 15% testing data. NN models are evaluated with four different input groups:

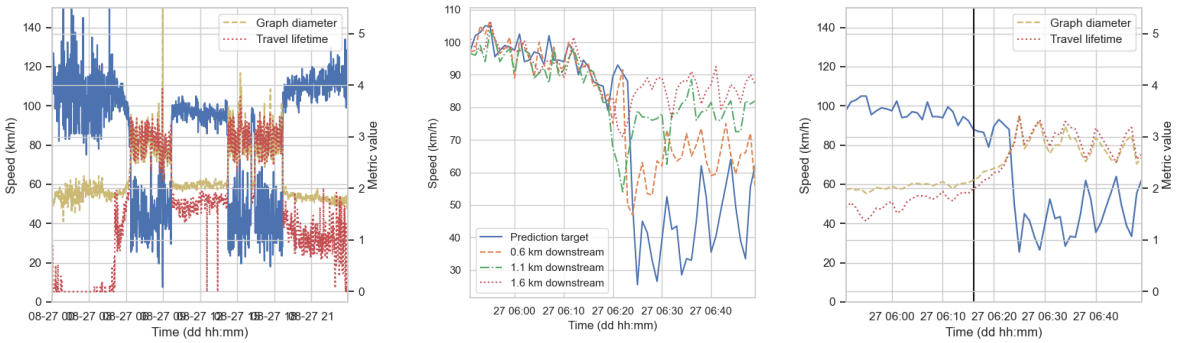
Travel lifetime This input group contains only the travel lifetime, so the model attempts to predict from only the travel lifetime as input.

Sensor data The most straightforward usage of the data is feeding the raw data to the NN. This group consists of measured speed and flow at all sensor locations.

All data This group combines all sensor data and the travel lifetime.

Target sensor only A final group included to compare how a local state condensed into a single number might compare to the measured speed and flow at just the target. This group can also be compared to the **sensor data** group to test whether measurements at other sensors can be ”confusing”.

All models are set up to predict speed at the target sensor location with a prediction horizon of 5 minutes. The test was carried out over three locations and on workweek and weekend separately, to prevent drawing conclusions from a single case. The tests on workweeks are shown in table 4.8, weekend



(a) Target speed, travel lifetime and graph diameter as calculated over an entire day

(b) Speeds recorded at target sensor and its three nearest sensors around a congestion occurrence. Note that congestion appears to travel upstream.

(c) Speed recorded at target sensor, the travel lifetime and graph diameter around a congestion occurrence.

Figure 4.7: Data study results of the process described in section 3.2.1

tests are shown in table 4.9. The three locations used were the situation around Gouda as shown in the preliminary study (see fig A.3), the two new locations are Rijnsweerd and Zoeterwoude-dorp, which were chosen because both locations offer a balanced mix between congestion during rush hours and free-flowing traffic. The sensors used are shown in figures A.4 and A.5 in the Appendix. The best-performing group varies per location and part of the week, but the group **Both** seems to be better than or comparable to the **Sensor data** group.

Table 4.8: MSE scores of NN models given the four input groups described earlier, during the workweek and at different locations as shown in the appendix.

(a) Location Gouda.

	Travel Lifetime	Sensor data	Both	Target sensor only
GRU(1)→Dense(1)	129±1	130±2	**125**±2	133±1
GRU(2)→Dense(2)	132±1	128±2	**125**±3	130±1
GRU(4)→Dense(4)	131±1	125±3	**121**±2	132±1
GRU(8)→Dense(8)	131±1	**111**±2	**111**±2	127±1
GRU(16)→Dense(16)	129±1	115±2	**111**±2	128±1

(b) Location Rijnsweerd.

	Travel Lifetime	Sensor data	Both	Target sensor only
GRU(1)→Dense(1)	**94**±1	110±3	112±2	100±1
GRU(2)→Dense(2)	**96**±2	112±3	112±3	102±2
GRU(4)→Dense(4)	**103**±2	119±3	113±2	105±2
GRU(8)→Dense(8)	**105**±2	118±2	117±2	110±2
GRU(16)→Dense(16)	**104**±1	123±3	120±2	108±2

(c) Location Zoeterwoude.

	Travel Lifetime	Sensor data	Both	Target sensor only
GRU(1)→Dense(1)	112±2	113 ±3	**100**±2	123±2
GRU(2)→Dense(2)	114±3	110±3	**106**±3	126±3
GRU(4)→Dense(4)	113±2	114±3	**107**±3	136±2
GRU(8)→Dense(8)	117±3	113±3	**109**±3	133±2
GRU(16)→Dense(16)	114±2	115±3	**101**±3	135±2

Table 4.9: MSE scores of NN models given the four input groups described earlier, during the weekend and at different locations as shown in the appendix.

(a) Location Gouda.

	Travel Lifetime	Sensor data	Both	Target sensor only
GRU(1)→Dense(1)	122±6	**89**±3	93±4	107±5
GRU(2)→Dense(2)	97±5	93±2	**91**±3	100±5
GRU(4)→Dense(4)	95±2	**94**±2	**94**±2	96±2
GRU(8)→Dense(8)	**95**±2	98±2	98±2	96±2
GRU(16)→Dense(16)	96±1	**94**±2	96±2	100±1

(b) Location Rijnsweerd.

	Travel Lifetime	Sensor data	Both	Target sensor only
GRU(1)→Dense(1)	72±5	**57**±4	**57**±3	72±5
GRU(2)→Dense(2)	60±4	**53**±1	**53**±1	59±4
GRU(4)→Dense(4)	57±3	**52**±1	**52**±1	55±1
GRU(8)→Dense(8)	53±1	**52**±1	**52**±1	54±1
GRU(16)→Dense(16)	55±1	**52**±1	**51**±1	54±1

(c) Location Zoeterwoude.

	Travel Lifetime	Sensor data	Both	Target sensor only
GRU(1)→Dense(1)	68±7	**30**±5	32±5	50±6
GRU(2)→Dense(2)	33±5	28±2	**27**±2	28±4
GRU(4)→Dense(4)	**25**±1	28±2	30±2	**25**±1
GRU(8)→Dense(8)	**25**±1	28±2	28±2	26±1
GRU(16)→Dense(16)	**26**±2	31±2	32±2	27±1

5

Discussion and conclusion

This thesis set out to present answers to the following questions:

1. How can knowledge of traffic characteristics be used to improve GNNs?
2. How can GNNs be used for short-term traffic prediction in a practical setting?
3. How can the traffic state of a local region be summarized?

The answers to these questions presented in this thesis are discussed and summarized in this chapter.

5.1. Discussion

In this section we identify and argue over some possible shortcomings of the models and methods developed in this thesis.

Local neighborhood intuition

The inputs for the individual models are selected from a local neighborhood, which is constructed by considering where information traveling at some speed can reach within the prediction horizon. This allows for linear scaling of model training and inference complexity with the size of the network. However, this determination of a local neighborhood is a heuristic based on the short-term prediction horizon. The basis of this heuristic may break when considering longer prediction horizons due to two reasons:

1. Traffic state changing multiple times within the prediction horizon. Imagine making a prediction given a congested current state. This congestion could resolve, making information from outside the currently considered neighborhood more relevant as traffic speeds rise. The considered neighborhood could then not be large enough to capture for example a new congested state developing.
2. Information relevant to the prediction may not exist yet in the network; cars that might cause congestion in an hour may not be on the road yet.

Both issues are in part mitigated by keeping the prediction horizon short. In practice, the issue of not capturing a traffic state changing multiple times within a prediction horizon of $10\,\mathrm{min}$ was not observed in the inspected predictions of this thesis work.

Lastly, the local neighborhood can be rather large. When considering a prediction horizon of 10 minutes, an average speed of $50\,\mathrm{km\,h^{-1}}$ results in a travel distance of $8.3\,\mathrm{km}$, which might well be larger than the size of a relatively large city. Groningen just fits, being inscribed by a circle with a radius of approximately $5\,\mathrm{km}$. This could form a problem if no or very few traffic sensors are present outside the city.

Wave speed

The input selection is implemented with static and dynamic adjacency information, where earlier the difficulty in calculating the wave speed of traffic information traveling upstream was mentioned. The speed of this wave can theoretically be calculated on a road section as a function of traffic flow q and speed v. The speed of this wave ω is then given by

$$\omega = \frac{q_1 - q_2}{q_1/v_1 - q_2/v_2}, \tag{5.1}$$

where the subscripts denote different sensor locations [24]. In practice however, calculating this quantity is not useful for predicting the start of congestion. This is because prediction would require capturing small deviations, but small deviations give a $0/0$ situation, making ω a very unstable quantity, as can be seen in figure 5.1. We can also reason statistically that if $q_1 \approx q_2$ and $v_1 \approx v_2$, and rewriting 5.1 as

$$\omega = v_1 v_2 \frac{q_1 - q_2}{q_1 v_2 - q_2 v_1}, \tag{5.2}$$

then the fraction in equation 5.2 is approximated by a Cauchy distribution[1], possibly explaining the instability. The approximate relation to a Cauchy distribution breaks when $q_1 \not\approx q_2$ or $v_1 \not\approx v_2$. The Cauchy approximation is also shown in figure A.6 in the Appendix using Monte Carlo sampling.

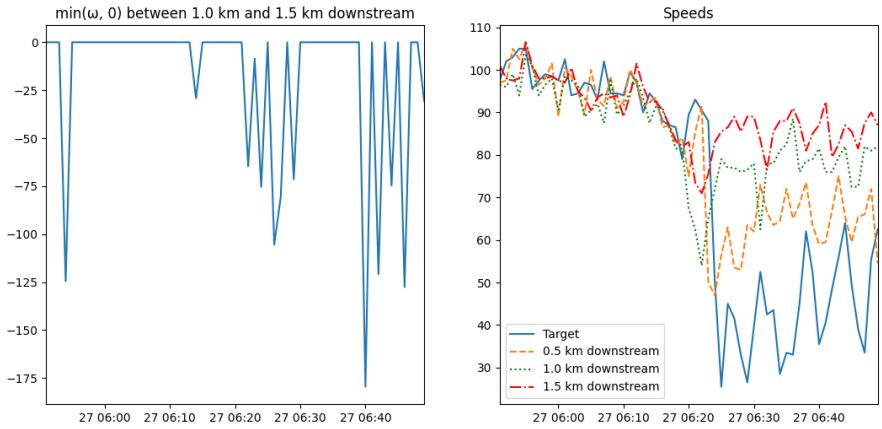


Figure 5.1: Calculated negative wave speeds ω and the congestion as visible from dropping speeds. Note that $\min(\omega, 0)$ is shown instead of ω, to limit positive values which are often large.

The broader availability of features like adaptive cruise control has inspired estimating such a wave speed on a vehicle level, which could be reported through FCD services [52].

Long range dependencies

We also feel the need to point out that predicting using local information loses explicit access to long-range effects. To paint a picture, assume a prediction horizon of 10 minutes for predicting at location A and assume there is a road closure at location B some 20 minutes away. The effect of the road closure can only be captured once it has reached the local neighborhood. This does not form a problem for driving a route which originally passed location B, as the speed at that location can be predicted using a neighborhood local to B. It does however raise the question of how well a local predictor can capture long-range effects, given that its access to long-range events is implicit.

Decision model

We also remark that a combination of split models using a random forest improves performance, but that including more predictors does not necessarily improve performance further. It seems that as outputs

[1] Using the fact that a normal with mean 0 divided by a normal with mean 0 gives a Cauchy distribution and that the difference of two χ^2 distributions is approximately normal with mean 0. We are skipping some concerns over dependencies and just indicate the type of relation we are dealing with.

of models converge on a similar answer, the decision model's performance worsens slightly. This is corroborated by a decrease in performance as the Naive estimator and historical mean are included during freely flowing traffic conditions, where traffic behaves "as normal". The regression models to be used should be seen as hyper-parameters of the model and finding the best model is an exercise in exchanging inputs until the best model is found.

Local information wave endpoint

In chapter 3 a way of visualizing and summarizing the traffic state of a local neighborhood is introduced. The visualization transforms a large number of variables into an insightful diagram which admits further summarization into a single quantity. Both the visualization and summarization allow incoming congestion waves to be captured before they reach the target and show extremely calm periods.

The method is shown in a thesis focused on short-term predictions, but the prediction horizon is not used in the presented algorithms. This gives rise to the question: why would the use of the presented method be limited to short-term prediction? The answer is two-fold:

1. Computation cost. Considering a large distance from where paths may start means that the paths are long and that a large number of paths are considered. Though software implementations for TDA are efficient in the sense that they execute their task efficiently, the task which they are solving is cubic in complexity with respect to the point cloud they are operating on [41, 53].

2. A traffic event moving in the traveltime diagram may stop or disappear before it reaches the target.

Local information method

The method of summarizing a traveltime diagram to the travel lifetime is a TDA-based transformation. The transformation works well and is based on, in my opinion, very elegant mathematics. However, it may seem TDA is an excessively powerful tool for the job which comes with the drawback of being computationally expensive. One might imagine finding the largest inscribed balls in the holes. This will ultimately rely on taking all combinations of 3 points in the diagram and fitting a circle, scaling with $\binom{n}{3} = \mathcal{O}(n^3)$ with n the number of points in the diagram: the same scaling as TDA [41, 53]. A different method summarizing the data leading to the traveltime diagram is possible, but the TDA method seems to me a very natural method to analyze the traveltime diagram.

5.2. Conclusions

With these possible shortcomings discussed, we summarize what has been achieved in this thesis.

Model improvements

First, an improved model for data-driven traffic prediction is developed. The improvements were made on the basis of leveraging traffic characteristics and discarding far-reaching network relations. Further, the model also admits a "selfless" form, which trades some performance for removal of the restriction to locations with sensors, allowing for greater flexibility.

Existing knowledge of traffic characteristics shows that information on some traffic state may propagate downstream or upstream. When traffic is flowing freely, information moves along with the vehicles. The other direction corresponds to congestion which travels upstream like a wave. This thesis has shown that training individual ML models for both directions in which information can flow and then combining both outputs can improve model performance. This has been shown in the Results chapter, and can additionally be seen in tables A.1, A.2 and A.3. We have also seen (fig 4.2b) that analyzing the output of the decision models used can bring insight into the origin of congestion. Another model performance improvement has been made using information from a local neighborhood. This thesis has shown that selecting inputs according to algorithm 1 can result in better model performance.

Local information summarization method

A method for visualizing and summarizing local traffic information is developed and called a traveltime diagram. Though the value to a human interpreter is subjective, it can be objectively stated that the form in which the data is presented in the traveltime diagram is easier to digest than the raw data. The

design of the method allows for similar interpretation of traveltime diagrams of two roads with equal speed limits. The traveltime diagram is condensed into a single number; the travel lifetime. A neural network model analysis is used as a proxy to determine the added value of the travel lifetime. This test suggests that the Travel Lifetime is informative when used in addition to raw sensor data. Perhaps the NN models used could not perform the same transformation that TDA could.

5.3. Closing points

Future work

Building on the work of this thesis, we can identify some things which are left to do in order to bring data-driven prediction models further. We distinguish improvements to the workings of the model in a more academic setting and improvements in application.

First, how could the presented model be improved further? SP-STNN works well for short-term predictions, but literature has shown that network-level models tend to perform better in longer-term predictions. As an example of such a model, the original STNN [9] was compared against (among others) a network-level model called GMAN [8], which outperformed STNN on time scales of longer than 30 minutes. A possible approach could be attempting to incorporate long-range dependencies in an efficient manner.

The problem of only having implicit long-range dependence seems to be a limitation of message-passing GNNs focused on in recent publications (mostly this year; 2024). Approaches to tackling this problem have been introduced in the form of subsampling long-range dependence paths [54] or by tackling a problem locally and then combining the local outputs [55], effectively subsampling in space. Especially the latter, passing information up to larger groupings, might be of interest for improving a traffic prediction neural network if the direction of information passing is reversed (or even made bidirectional); a network level NN may predict vague quantities such as how busy traffic will be for a coarse version of the road network, which can then be used as input for a locally predicting GNN. A basis for this idea has already been laid in [53], which used TDA to show how the Louvain algorithm can be used to make a graph coarser.

Long-range dependence is also studied extensively in physics, such as the statistical mechanics of the Ising or XY model, see [38, 56] for definition. Statistical mechanics tells us that if a zoomed-out view of a system behaves similarly to the original, we can exploit this rescaling to analyze the problem, this is called renormalization theory. Using this as inspiration, one could also apply a locally predicting GNN to a coarse version of the network and then refine it until a desired resolution has been achieved.

Secondly, for implementation more experience is useful. The model presented is quite ready to be applied, but practical questions are likely to arise:

- How well will a model trained on one city transfer to another city?
- How does the performance of the model change when the prediction horizon is taken to be long?
- If regulations like maximum speed change, does the model need to be retrained, or will it capture the new situation well?

These questions are best answered through gaining experience by simply trying and evaluating the outcome. Crucial to being able to gain this experience is the availability of data like the MVVS dataset of more cities/networks. As such, we recommend first gathering datasets like the MVVS dataset on more networks.

Summary

This thesis presents localized methods for traffic prediction and analysis. The prediction method presents an extension of a state-of-the-art Graph Neural Network inspired by traffic flow characteristics on a local level. This inspiration from traffic flow characteristics consists of two parts. The first intuition is that state at some location and at time T in the future will be not influenced by information which is further than traveling time T away. The second intuition is that traffic information traveling with or against the stream of traffic behaves differently. The developed model leverages these intuitions to increase model prediction performance. Further, a modification is made which allows the model to

be applied at an arbitrary location in a network, at the cost of performance. Alongside these model extensions, a novel method of visualizing a local traffic state is presented through constructing a novel traveltime diagram. This diagram can be used as a visual tool for analyzing traffic locally. Further, the traveltime diagram is designed to be summarized using Topological Data Analysis to a quantity called the Travel Lifetime which can represent traffic states ranging from extremely calm to imminent congestion to a congested state in a single number. The newly proposed Travel Lifetime is tested as an input to a Neural Network model for predicting traffic speed showing that its use as an input can improve model performance.

Bibliography

1. Rijkswaterstaat (Dutch Ministery of Infrastructure and Water Management). *Tunneldoseren* https://www.rijkswaterstaat.nl/wegen/wegbeheer/tunnels/tunneldoseren.
2. *MT-ITS : 2015 International Conference on Models and Technologies for Intelligent Transportation Systems : Budapest University of Technology and Economics (BME), Faculty of Transport Engineering and Vehicle Engineering, Department of Transport Technology and Economics : 3-5 June 2015, Budapest* 191. ISBN: 9789633131428 (Institute of Electrical and Electronics Engineers, 2015).
3. Oh, S., Byon, Y. J., Jang, K. & Yeo, H. Short-term travel-time prediction on highway: A review on model-based approach. *KSCE Journal of Civil Engineering* **22,** 298–310. ISSN: 19763808 (Jan. 2018).
4. Alghamdi, T., Elgazzar, K., Bayoumi, M., Sharaf, T. & Shah, S. *Forecasting Traffic Congestion Using ARIMA Modeling* in *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)* (IEEE, June 2019), 1227–1232. ISBN: 978-1-5386-7747-6.
5. Anand, N. C., Scoglio, C. & Natarajan, B. *GARCH — non-linear time series model for traffic modeling and prediction* in *NOMS 2008 - 2008 IEEE Network Operations and Management Symposium* (IEEE, 2008), 694–697. ISBN: 978-1-4244-2065-0.
6. Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* **65,** 386–408. ISSN: 1939-1471 (1958).
7. Park, W. J. & Park, J.-B. Park and Park: Artificial intelligence in dentistry. *European Journal of Dentistry* (2018).
8. Zheng, C., Fan, X., Wang, C. & Qi, J. *GMAN: A Graph Multi-Attention Network for Traffic Prediction* in *The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)* (2020). https://github.com/zhengchuanpan/GMAN..
9. Yang, S., Liu, J. & Zhao, K. Space Meets Time: Local Spacetime Neural Network For Traffic Flow Forecasting. http://arxiv.org/abs/2109.05225 (Sept. 2021).
10. Zhu, J., Song, Y., Zhao, L. & Li, H. A3T-GCN: Attention Temporal Graph Convolutional Network for Traffic Forecasting. http://arxiv.org/abs/2006.11583 (June 2020).
11. Liu, Z., Wan, G., Prakash, B. A., Lau, M. S. & Jin, W. *A Review of Graph Neural Networks in Epidemic Modeling* in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Association for Computing Machinery, Aug. 2024), 6577–6587. ISBN: 9798400704901.
12. Tsaban, T. *et al.* Harnessing protein folding neural networks for peptide–protein docking. *Nature Communications* **13.** ISSN: 20411723 (Dec. 2022).
13. Abadi, M. *et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems* tech. rep. (). www.tensorflow.org..
14. Cao, M., Li, V. O. & Chan, V. W. *A CNN-LSTM Model for Traffic Speed Prediction* in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)* (IEEE, 2020). ISBN: 9781728152073.
15. Kumar, K., Parida, M. & Katiyar, V. K. Short term traffic flow prediction in heterogeneous condition using artificial neural network. *Transport* **30,** 397–405. ISSN: 16483480 (Oct. 2015).
16. Li, Y., Yu, R., Shahabi, C. & Liu, Y. *Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting* in (July 2017). http://arxiv.org/abs/1707.01926.
17. Feng, J., Chen, Y., Li, F., Sarkar, A. & Zhang, M. *How Powerful are K-hop Message Passing Graph Neural Networks* in *36th Conference on Neural Information Processing Systems (NeurIPS 2022)* (2022). ISBN: 2205.13328v4.

18. Tilson, S. *Applications for Homology* Dec. 2010. https://math.stackexchange.com/a/13668.
19. Frederik. *Applications of homological algebra in the sciences and engineering* Feb. 2023. https://math.stackexchange.com/a/4645951.
20. Bellman, R. *Dynamic Programming* ISBN: 9780691079516 (Princeton University Press, 1957).
21. Majumdar, S. & Laha, A. K. Clustering and classification of time series using topological data analysis with applications to finance. *Expert Systems with Applications* **162.** ISSN: 09574174 (Dec. 2020).
22. Curto, C. & Itskov, V. Cell groups reveal structure of stimulus space. *PLoS Computational Biology* **4.** ISSN: 15537358 (2008).
23. Palm, H. & Papjes, E. Groningen implementeert multimodaal én voorspellend verkeersmodel. *NM magazine* (2022).
24. Dr. ir. R. van Nes. in. Chap. 4 (Technische Universiteit Delft). https://ocw.tudelft.nl/course-readings/deel-4-verkeersstroomtheorie-en-verkeersmanagement/.
25. Contreras, J., Espínola, R., Nogales, F. J. & Conejo, A. J. ARIMA models to predict next-day electricity prices. *IEEE Transactions on Power Systems* **18,** 1014–1020. ISSN: 08858950 (Aug. 2003).
26. Bauwens, L., Laurent, S. & Rombouts, J. V. *Multivariate GARCH models: A survey* Jan. 2006.
27. Kontopoulou, V. I., Panagopoulos, A. D., Kakkos, I. & Matsopoulos, G. K. *A Review of ARIMA vs. Machine Learning Approaches for Time Series Forecasting in Data Driven Networks* Aug. 2023.
28. Koza John R., Bennett, F. H., Andre David & Keane Martin A. in *Artificial Intelligence in Design '96* (eds Gero John S. & Sudweeks, F.) 151–170 (Springer Netherlands, Dordrecht, 1996). ISBN: 978-94-009-0279-4. https://doi.org/10.1007/978-94-009-0279-4_9.
29. Abiwinanda, N., Hanif, M., Hesaputra, S. T., Handayani, A. & Mengko, T. R. *Brain tumor classification using convolutional neural network* in *IFMBE Proceedings* **68** (Springer Verlag, 2019), 183–189.
30. Akhtar, M. & Moridpour, S. *A Review of Traffic Congestion Prediction Using Artificial Intelligence* 2021.
31. Bahdanau, D., Cho, K. & Bengio, Y. *NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE* in ().
32. Rickard, M. *Mixture of Experts: Is GPT-4 Just Eight Smaller Models?* June 2023. https://matt-rickard.com/mixture-of-experts-is-gpt-4-just-eight-smaller-models.
33. Medar, R., Rajpurohit, V. S. & B., R. *Impact of Training and Testing Data Splits on Accuracy of Time Series Forecasting in Machine Learning* in *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)* (IEEE, 2017). ISBN: 9781538640081.
34. Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. http://arxiv.org/abs/1207.0580 (July 2012).
35. NASA. *NASA NEURAL NETWORK PROJECT PASSES MILESTONE* Sept. 2003. https://www.nasa.gov/news-release/nasa-dryden-flight-research-center-news-room-news-releases-nasa-neural-network-project-passes-milestone/.
36. John Hertz, Anders Krogh & Richard G Palmer. *Introduction to the theory of neural computation* 1st ed. (Addison-Wesley, Redwood, 1991).
37. Vaswani, A. *et al. Attention Is All You Need* tech. rep. (2023).
38. Thijssen, J. *LECTURE NOTES STATISTICAL PHYSICS AP3021* Delft, 2022. http://brightspace.tudelft.nl/underthecourseAP3021.
39. Cho, K. *et al.* Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. http://arxiv.org/abs/1406.1078 (June 2014).

40. Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani & Jonathan Taylor. *An Introduction to Statistical Learning* 1st ed. ISBN: 978-3-031-38746-3 (Springer, 2023).
41. Chazal, F. & Michel, B. An Introduction to Topological Data Analysis: Fundamental and Practical Aspects for Data Scientists. *Frontiers in Artificial Intelligence* **4.** ISSN: 26248212 (Sept. 2021).
42. Botnan, M. B. *Topological Data Analysis Mastermath* tech. rep. (2024). `https://www.few.vu.nl/~botnan/`.
43. Takens, F. *Detecting strange attractors in turbulence* in *Dynamical Systems and Turbulence, Warwick 1980* (eds Rand Davidand Young & Lai-Sang) (Springer Berlin Heidelberg, Berlin, Heidelberg, 1981), 366–381. ISBN: 978-3-540-38945-3.
44. Dirafzoon, A., Lokare, N. & Lobaton, E. *ACTION CLASSIFICATION FROM MOTION CAPTURE DATA USING TOPOLOGICAL DATA ANALYSIS* in *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)* (IEEE, 2016).
45. Gidea, M. & Katz, Y. Topological data analysis of financial time series: Landscapes of crashes. *Physica A: Statistical Mechanics and its Applications* **491,** 820–834. ISSN: 03784371 (Feb. 2018).
46. Ansel, J. *et al. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation* in *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS* **2** (Association for Computing Machinery, Apr. 2024), 929–947. ISBN: 9798400703850.
47. Guillame-Bert, M., Bruch, S., Stotz, R. & Pfeifer, J. *Yggdrasil Decision Forests: A Fast and Extensible Decision Forests Library* in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Association for Computing Machinery, Aug. 2023), 4068–4077. ISBN: 9798400701030.
48. The GUDHI Project. *GUDHI User and Reference Manual* 3.10.1 (GUDHI Editorial Board, 2024).
49. Bui, K. H. N., Cho, J. & Yi, H. Spatial-temporal graph neural network for traffic forecasting: An overview and open research issues. *Applied Intelligence* **52,** 2763–2774. ISSN: 15737497 (Feb. 2022).
50. Treiber, M. & Kesting, A. *Traffic Flow Dynamics* ISBN: 978-3-642-32459-8 (Springer, 2013).
51. Kennel, M. B., Brown, R. & Abarbanel, H. D. I. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Physical Review A* **45,** 3403–3411. ISSN: 1050-2947 (Mar. 1992).
52. Huang, D., Shere, S. & Ahn, S. *Dynamic Highway Congestion Detection And Prediction Based On Shock Waves* in *Proceedings of the seventh ACM international workshop on VehiculAr InterNETworking* (ACM Digital Library, 2010). ISBN: 9781450301459.
53. Carmody, D. R. & Sowers, R. B. Topological analysis of traffic pace via persistent homology. *Journal of Physics: Complexity* **2.** ISSN: 2632072X (June 2021).
54. Chen, D., Schulz, T. H. & Borgwardt, K. Learning Long Range Dependencies on Graphs via Random Walks. `http://arxiv.org/abs/2406.03386` (June 2024).
55. Hariri, A. & Vandergheynst, P. *Graph learning for capturing long-range dependencies in protein structures* tech. rep. (2024).
56. Dijkhuizen, R. *An exploration of the Kuramoto model* PhD thesis (Technische Universiteit Delft, Delft, 2021).

A

Appendix

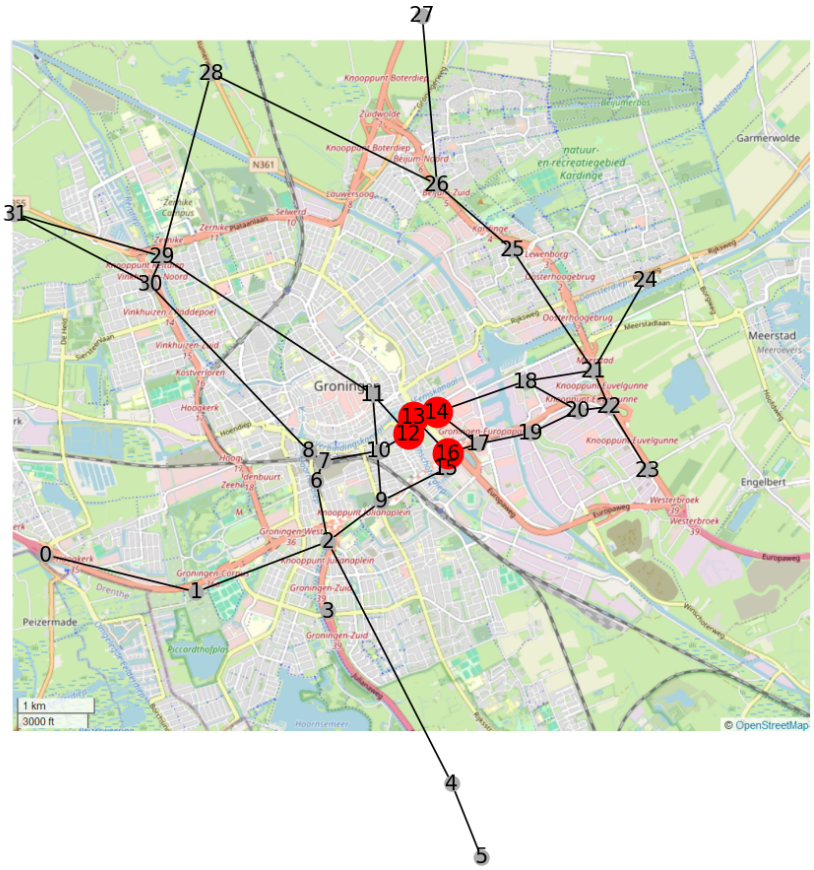


Figure A.1: Network for the NDW dataset. Sensors used in the LOOCV study are marked red and slightly larger.

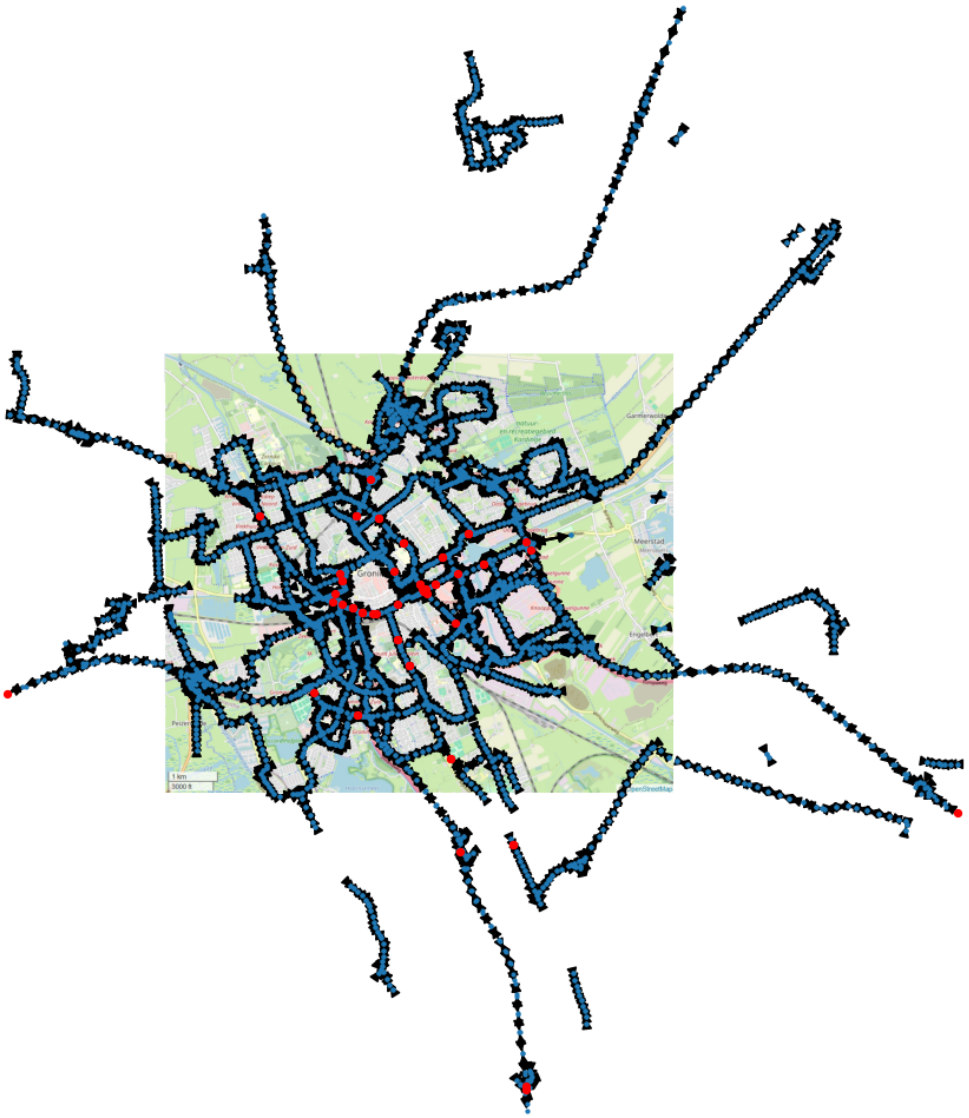
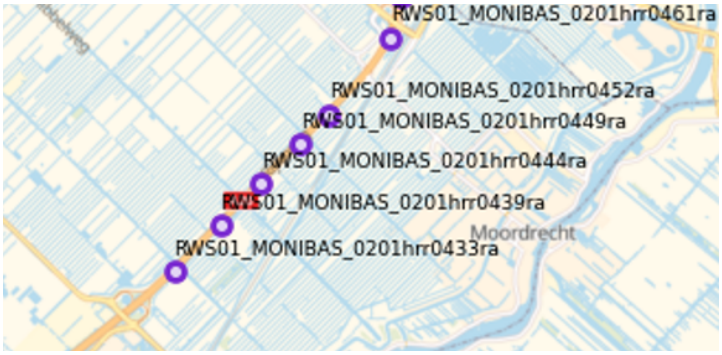


Figure A.2: Network for the MVVS dataset. Sensor locations which reported sufficient flow data are shown with red dots.



(a) Sensor locations used.

Sensor	461	452	449	444	439	433	Dummy
Distance to next	9 hm	3 hm	5 hm	5 hm	6 hm	5 hm	-

(b) Distances between sensors as shown in figure A.3a

Figure A.3: Spatial context used for the 1-dimensional travel lifetime data study at location Gouda.

(a) Sensor locations used.

Sensor	825	829	835	839	844	847	851	854	Dummy
Distance to next	4 hm	6 hm	4 hm	5 hm	3 hm	4 hm	3 hm	5 hm	-

(b) Distances between sensors as shown in figure A.4a

Figure A.4: Spatial context used for the 1-dimensional travel lifetime data study at location Rijnsweerd.

(a) Sensor locations used.

Sensor	362	367	371	375	380	384	388	391	Dummy
Distance to next	5 hm	4 hm	4 hm	5 hm	4 hm	4 hm	3 hm	5 hm	-

(b) Distances between sensors as shown in figure A.5a

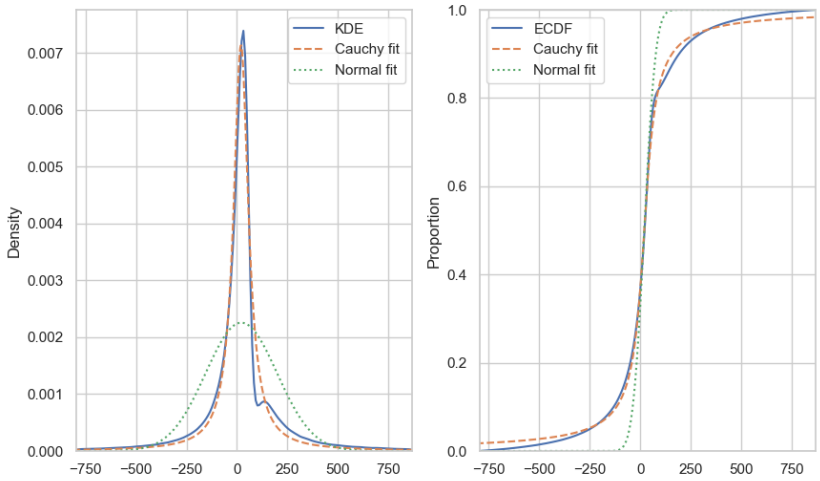
Figure A.5: Spatial context used for the 1-dimensional travel lifetime data study at location Zoeterwoude-dorp.

Table A.1: Results of (P)STNN models for speed on the NDW dataset. Indiv in sample denotes having trained a random forest for each sensor location, other columns apply a single general random forest for all sensor locations. The fuse and switch models in this table are combining output of two (P)STNN models, the historic mean and the naive estimator. Fusing and switching models are denoted with -,s or d for STNN, static-P-STNN and dynamic-P-STNN. An example: (d, s) denotes fusing upstream dynamic, downstream static, historic mean and naive estimators.

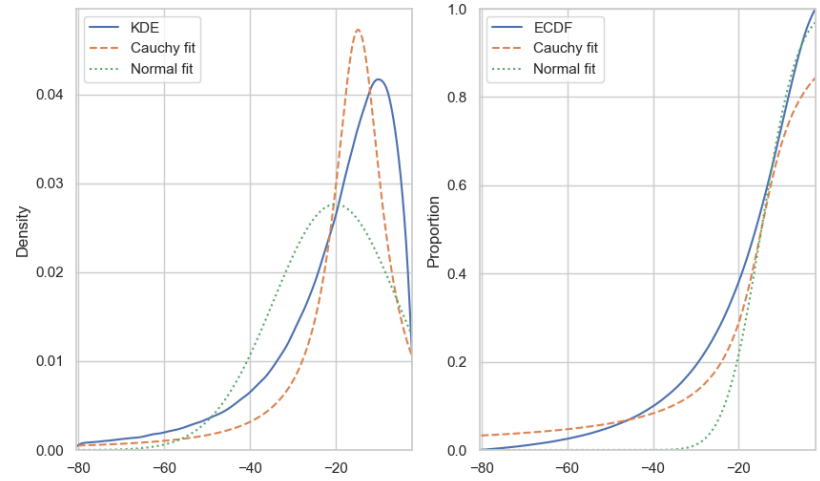
	in sample		indiv in sample		out of sample	
	MSE	MAE	MSE	MAE	MSE	MAE
Upstream dynamic-P-STNN	40.2	4.44			58.9	5.95
Downstream static-P-STNN	38.7	4.33			116	9.11
Upstream static-P-STNN	38.8	4.29			91.5	7.46
Upstream STNN	40.6	4.51			62.4	6.02
Downstream STNN	39.2	4.37			58.8	5.65
Historic mean	63.6	5.54				
Naive	61.2	5.53				
Fuse (d, s)	46.0	4.76	42.9	4.58	45.5	4.77
Switch (d, s)	63.6	5.54	63.5	5.54	63.4	5.53
Fuse (s, s)	46.4	4.79	42.9	4.56	44.8	4.74
Switch (s, s)	63.6	5.54	63.7	5.55	63.6	5.54
Fuse (-, -)	45.4	4.72	42.4	4.46	45.5	4.74
Switch (-, -)	63.6	5.54	63.7	5.55	63.7	5.54
Fuse (d, -)	45.6	4.74	42.5	4.56	44.9	4.70
Switch(d, -)	63.6	5.54	63.5	5.54	63.7	5.55
Fuse (s, -)	45.4	4.74	42.4	4.56	44.6	4.70
Switch (s, -)	63.6	5.54	63.6	5.54	63.7	5.54
Fuse (-, s)	45.4	4.73	42.5	4.56	46.1	4.80
Switch(-, s)	63.6	5.54	63.7	5.55	63.3	5.52

Table A.2: Results of (SP)STNN models for flow on the NDW dataset. The fuse and switch models in this table are combining output of two (SP)STNN models. Fusing and switching models are denoted with -,s or d for STNN, static-P-STNN and dynamic-P-STNN. An example: (d, s) denotes fusing upstream dynamic and downstream static models.

	In sample		Out of sample	
	MSE $(\cdot 10^3)$	MAE	MSE $(\cdot 10^3)$	MAE
Upstream dynamic-SP-STNN	226	364	514	580
Downstream static-SP-STNN	217	355	402	509
Upstream static-SP-STNN	218	355	313	433
Upstream STNN	216	353	272	406
Downstream STNN	217	354	491	570
Fuse (d, s)	220	354	303	417
Switch (d, s)	219	356	393	502
Fuse (s, s)	220	354	244	374
Switch (s, s)	216	353	304	429
Fuse (-, -)	222	354	265	390
Switch (-, -)	216	353	272	406
Fuse (d, -)	221	354	327	433
Switch(d, -)	221	357	436	532
Fuse (s, -)	221	354	251	377
Switch (s, -)	216	353	308	430
Fuse (-, s)	220	354	260	388
Switch(-, s)	216	354	272	406



(a) Situation where $q_1 \approx q_2$ and $v_1 \approx v_2$.



(b) Situation where $q_1 >> q_2$ and $v_1 >> v_2$

Figure A.6: Monte Carlo sampling of equation 5.1 with Cauchy and Normal distributions fit to the sampled data. Both figures were sampled with q_1, q_2, v_1, v_2 having means such that $\omega(q_1, q_2, v_1, v_2) \approx -17\,\mathrm{km\,h^{-1}}$.

Table A.3: Results of (SP)STNN models for speed on the NDW dataset. Indiv in sample denotes having trained a random forest for each sensor location, other columns apply a single general random forest for all sensor locations. The fuse and switch models in this table are combining output of two (SP)STNN models, the historic mean and the naive estimator. Fusing and switching models are denoted with -,s or d for STNN, static-P-STNN and dynamic-P-STNN. An example: (d, s) denotes fusing upstream dynamic, downstream static, historic mean and naive estimators.

	in sample		indiv in sample		out sample		selfless in sample		selfless out of sample	
	MSE ($\cdot 10^3$)	MAE	MSE ($\cdot 10^3$)	MAE	MSE ($\cdot 10^3$)	MAE	MSE ($\cdot 10^3$)	MAE	MSE ($\cdot 10^3$)	MAE
Upstream dynamic-SP-STNN	217	356			262	405	226	364	514	580
Downstream static-SP-STNN	217	355			272	403	217	355	402	509
Upstream static-SP-STNN	215	354			299	407	218	355	313	433
Upstream STNN	217	356			260	385	216	353	272	406
Downstream STNN	215	353			367	480	217	354	491	570
Historic mean	325	428								
Naive	840	737								
Fuse (d, s)	256	379	245	372	247	375	252	378	260	384
Switch (d, s)	325	428	324	428	325	428	325	428	325	428
Fuse (s, s)	254	380	245	373	253	378	255	380	251	377
Switch (s, s)	325	428	325	428	325	428	325	428	325	428
Fuse (-, -)	255	381	247	374	251	377	259	383	253	379
Switch (-, -)	325	428	325	428	325	429	325	428	325	428
Fuse (d, -)	253	379	245	372	247	376	252	378	264	387
Switch(d, -)	325	428	325	428	325	428	325	428	325	428
Fuse (s, -)	254	380	246	384	257	381	256	382	251	377
Switch (s, -)	325	428	325	428	325	428	325	428	325	428
Fuse (-, s)	254	380	246	373	251	377	256	381	255	380
Switch(-, s)	325	428	325	428	325	428	325	428	325	428