UNIVERSITY OF BUCHAREST

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

MASTERS OF ARTIFICIAL

INTELLIGENCE

**Dissertation Thesis**

**Spatial-Temporal Neural Networks for Traffic Systems**

Coordinator Graduate

Conf. dr. Ichim Bogdan Bomher Sebastian

BUCHAREST,

January 2022`

Abstract

In the present day, vehicles are getting faster and more affordable to common people. This causes people to buy more vehicles, especially cars which increases the number of cars on the road. This causes a new modern problem to arise, traffic congestion. Traffic optimization is a modern problem for urban areas which tries to minimize waiting time.

With the rise of Artificial Intelligence in the past decade, new solutions can be found for problems that were thought to be unsolvable. New methods in deep learning are used in multiple areas and they can be used in traffic prediction. With traffic information available, a machine learning model can be made to predict waiting times and impact route planning.

This paper will propose several experiments which will build traffic prediction models based on information acquired from the public roads in the city of Los Angeles. The dataset is made public on a governmental site [1] developed by the California Department of Transportation. The roads represented are found in the 7th district of Los Angeles and they are freeways.

In this paper, better insights are offered through data visualization of the extracted data. Insights about speed, traffic flow, and lane occupancy are acquired and analyzed. A geographical view of the road sensors is created to improve the perspective. The dataset is cleaned and only valid data will be used for classification. Four subsets of the dataset will be used to gain insights into how the volume of information and map generation will affect the model’s predictions.

Different models are used to predict individual sensor level speed. Linear Regression and statistical models are used as a reference to predict individual sensors without taking into account neighboring information. To take into account spatial information, three Graph Neural Networks models will be used. Graphs are useful for retaining nodes and neighboring nodes' information. These models will take into account the whole graph state, not just a single node prediction at a time.

The final chapter will cover the results of the experiment, data visualization, and different comparisons of the multiple subsets used for each experiment, as well as conclusions and future development proposals.

Table of Contents

[I. INTRODUCTION 6](#_Toc103796286)

[1. Motivation 6](#_Toc103796287)

[II. THEORY 9](#_Toc103796288)

[1. Traffic Systems 9](#_Toc103796289)

[2. Machine Learning 12](#_Toc103796290)

[2.1. Multiple Linear Regression 12](#_Toc103796291)

[2.2. Artificial Neural Networks 13](#_Toc103796292)

[2.2.1. Learning Process 15](#_Toc103796293)

[2.2.2. Cost Functions 16](#_Toc103796294)

[2.2.3. Backward Propagation 17](#_Toc103796295)

[2.3. Deep Learning 17](#_Toc103796296)

[2.3.1. Regularization 17](#_Toc103796297)

[2.3.2. Optimization 21](#_Toc103796298)

[2.3.3. Convolutional Neural Networks 23](#_Toc103796302)

[2.3.4. Geometric Deep Learning 24](#_Toc103796303)

[2.3.5. Recurrent Neural Network 28](#_Toc103796304)

[III. RELATED WORK 31](#_Toc103796305)

[IV. METHODOLOGY & EXPERIMENT 33](#_Toc103796306)

[1. Methodology 33](#_Toc103796307)

[1.1. Technical Implementation 33](#_Toc103796308)

[1.2. Dataset 34](#_Toc103796309)

[2. Experiments 36](#_Toc103796310)

[2.1. Data Preparation 37](#_Toc103796311)

[2.2. Map Generation 37](#_Toc103796312)

[2.2.1. Geodesic 38](#_Toc103796313)

[2.2.2. Open Source Routing Machine 38](#_Toc103796314)

[2.2.3. Linear Regression 39](#_Toc103796315)

[2.2.4. Generative 40](#_Toc103796316)

[2.3. Data Visualization 40](#_Toc103796317)

[2.3. Models 46](#_Toc103796318)

[2.4. Hyper Parameter Tuning 48](#_Toc103796319)

[2.5. Results 49](#_Toc103796320)

[2.5.1. Graph Neural Network Results 49](#_Toc103796321)

[2.5.2. Non-GNN Results 56](#_Toc103796322)

[V. CONCLUSIONS & DEVELOPMENT 60](#_Toc103796323)

[VI. BIBLIOGRAPHY 61](#_Toc103796324)

# INTRODUCTION

This paper represents a research project which uses machine learning algorithms and data science methods in the domain of traffic flow theory to gain information and predict traffic speed.

The first half of this paper will cover the theoretical aspect of traffic flow theory and what it represents, as well as the machine learning theoretical background including Neural Networks, Deep Learning, and Geometric Deep Learning.

After the theoretical part, this paper will include the methodology used for this project and how it was built, as well as the experiments with their results. Related work in the domain of machine learning with traffic flow will be included as well. The methodology will present the tools and software programs used for the experiment, data preparation, and data visualization. Data is acquired from Los Angeles traffic sensors mounted on the freeways located in the 7th district.

The last segment will cover the experiment settings, data preparation, and the machine learning models together with their results on different variables. Lastly, observations and insights will be provided based on the results. This paper will conclude with future development ideas, which will cover the limitations of the experiment and what could be improved or expanded.

The next part will present the motivation for why the subject has been chosen for this paper and why traffic flow is a modern problem that requires insights and solutions.

## Motivation

As technology evolves so do the means of transportation. In 66 years, humanity has progressed from the first-ever flight to the landing on the moon. This logic can be applied to terrestrial vehicles. Until the 19th-century humanity's main transportation method was horses for thousands of years. In the next century locomotives and cars replaced horses. Over time cars were starting to become more affordable to the public. This means that more people will use cars. The more cars there are, the bigger the traffic congestion will be.

With time cities become increasingly bigger in population and surface occupied. This had made them more difficult to navigate. In the year 1300 London had a population of 80 000 while in 2022 it has a population of 8 million people. This increased compression of cities combined with people buying and using more cars causes heavy traffic congestion with huge waiting times.

A study made in 2019 [2] showed that a person driving in New York City has to spend on average 92 hours in traffic. This means almost four days. The average person in the USA spends around 54 hours a year in traffic. The waiting time in traffic will only increase. It is estimated that, if the infrastructure capacity will not improve, the USA congestion cost will grow by 19% and wasted fuel will increase by 9%.

There are multiple benefits to reducing traffic congestion which affects not only the population, but an entire city, or even a country:

* Reduced traffic congestion lowers waiting time for transportation of goods which boosts the economy of the city. This in turn attracts investors to the city.
* A lower time in the daily commute helps the population save time and money on fuel, which boosts the free time of the average citizen as well as its economy.
* Lower traffic means fewer vehicles in the traffic, which reduces air and noise pollution, increasing the citizens' health.
* High traffic can cause frustration in drivers, which in term leads to neglect while driving, road rage, and even vehicle accidents. These events can be reduced with lower traffic.
* If multiple cities have lower traffic, it can even boost an entire country’s economy.

Traffic congestion is a global problem that affects densely packed urban areas the most. According to a website [3], the top four most congested cities are:

* Istanbul, Turkey with a congestion level of 62%
* Moscow region (oblast), Russia with a congestion level of 61%
* Kyiv, Ukraine with a congestion level of 56%
* Bogota, Colombia with a congestion level of 55%

The first one, Istanbul, is the most populated city in Europe. Bucharest is ranked 8th and Los Angeles is ranked 59th.

Multiple solutions exist to solve this problem, the first one is to share a vehicle among commuters and reduce the number of vehicles in traffic. For this solution, many will choose a personal car as it is more comfortable than the public transportation system. The waiting times are a detriment as well. For a car, a person does not have to wait to start going towards its destination. Only the metro system is better than cars but it is the hardest method to implement as it requires underground construction work. Smaller vehicles like bikes and motorcycles are rarely used due to the higher chance of injury in case of accidents. A car protects its passengers better. One of the few solutions is to know in time what waiting time is on the road such that better route planning can be made.

# THEORY

## Traffic Systems

Traffic Flow Theory studies the interaction between traffic participants of different types, including vehicles and pedestrians, with the infrastructure. It is having the goal to create an efficient travel space with minimum waiting times and optimal traffic movement such that traffic congestion is reduced.

Traffic networks are complex and nonlinear, depending on external factors such as time of day, season, and weather, and also on internal factors such as infrastructure, the number of vehicles, their types, and unexpected vehicle accidents. Since traffic is based on human interaction or rather individual reactions to other pedestrians or vehicles, the public system does not follow any rules and is rather chaotic.

The traffic system is composed of roads or the pavement on which the participants in traffic travel at a certain speed to reach their individual goals. The size of the system can vary and is objective. It can range from a neighborhood to a country's traveling system. There are traffic systems that are unique to a single type of vehicle. For example, the railway system, in which just the train travels. Traffic flow theory is focused on city-sized traffic systems for cars, which exhibit the biggest congestion.

Participants in traffic are pedestrians and vehicles (bikes, motorbikes, cars, or public vehicles). Different types of vehicles can be more effective than others. An example is the underground metro system, which is supervised by an organization. This way the participants travel restrictively to preserve the entire system's efficiency and can take thousands of passengers. In parallel, a car can take up to a maximum of 5 passengers. A car can travel to any point whereas a metro system has predetermined stations.

The main variables that are considered in traffic flow theory are density, speed, and flow. Density is represented as and it represents the number of vehicles occupying a space unit. Speed is represented as and it represents the velocity with which a vehicle moves. Lastly, flow is represented as and it is the number of vehicles passing two points in a time frame. [4]

Speed can be defined as the length traveled in a time unit. Due to the high number of vehicles in a traffic system, it is impractical to quantify the velocity of every vehicle, and the mean velocity is measured. There are two main definitions of speed.

The first one is the time average speed. It is the mean velocity of a vehicle over time and is computed by measuring the distance traveled by the vehicle over a certain reference point. This method is not accurate because the average speed of a vehicle over a wide range does not take into account the difference between the speeds of different vehicles.

|  |  |  |
| --- | --- | --- |
|  |  | (2.1) |

In Equation 2.1 represents the vehicle count moving through a point; is the speed of vehicle .

The second method is called space average speed. With this method, the speed is collected on the entire section. Successive recordings of passing vehicles over the entire segment are considered to track individual vehicle speed, after which the speed is computed. This method is regarded as more precise than the first method.

|  |  |  |
| --- | --- | --- |
|  |  | (2.2) |

where is the vehicle count going through the roadway segment; = speed of vehicle .

Density is the total count of vehicles on a road segment. Density can be categorized into two types: jam density and critical density. Critical density is considered to be the superior density under free flow, while jam density is considered, under congestion, the maximum density. Density is measured as:

|  |  |  |
| --- | --- | --- |
|  |  | (2.3) |

where is density; is the length of a roadway; = time; m = vehicle count.

Flow can be represented as the total vehicle count going through a mark given a temporal period. Headway is the flow inverted, which is represented by the period elapsed between a vehicle going through a mark and the next vehicle. Flow is measured as:

|  |  |  |
| --- | --- | --- |
|  |  | (2.4) |

where is the flow; is the time interval; = point in space; m = number of vehicles.



Figure 1 Flow Density Relationship

The goal of traffic flow theory is to analyze and develop a network that will allow vehicles to arrive at their destinations with minimal time loss. This is achieved in a four stages process: [5]

* Generation – Calculate how many expeditions are to be produced by the requirements of the passengers.
* Distribution – Generate the path between the starting and stopping point based on what has been generated in the first step.
* Modal Split/Mode Choice – Decide the distribution of different types of vehicles for the number of passengers.
* Route Assignment – Assign each vehicle its route such as the entire system has a minimum traveling time.

## Machine Learning

Machine Learning belongs to the area of Artificial Intelligence. It creates and develops software that can learn and improve by being exposed to data (experience) over time without being explicitly programmed [6]. This method is used for tasks that are mathematically impossible to program iteratively. For example, it is impossible to explicitly program a function that can classify cats and dogs in an image. The process of learning starts with observing data, experience (time observing data), and data pattern recognition for better future decisions. The goal is to let the program learn without human intervention.

Three categories divide machine learning algorithms depending on the task and the nature of the data: [7]

* Supervised learning: The data is presented with a label. The task is to predict the correct label. An example is the classification of an animal inside a photo as a cat or a dog. The image contains a correct label and the objective is to predict which animal is inside the photo.
* Unsupervised learning: The data has no label. The task is to structure the data to find similarities or patterns in the given data and label it for future unknown data.
* Reinforcement learning: There exists a dynamic environment to perform a specific task. As the problem space is explored the program receives positive or negative feedback based on its current decision such that positive behavior is reinforced. Self-driving vehicles are an example.

### Multiple Linear Regression

A commonly used machine learning algorithm is Linear Regression. It is a method of learning which predicts a dependent value based on an independent value. Linear Regression is commonly used in forecasting or determining cause and effect relationships between values. There is more than one method of regression. Most of them differ in the independent value count and the type of relationships between independent and dependent values.

Linear Regression is a variant of regression where there is a linear relationship between independent and dependent values [8]. Multiple Linear Regression is when there is a relationship between multiple independent values and one dependent value. Each independent variable is assigned a weight, then they are summed together with a bias to produce a result:

|  |  |  |
| --- | --- | --- |
|  |  | (2.5) |

represents the dependent value; is the independent values; is the total independent values count; is the bias.

### Artificial Neural Networks

One of the algorithms used in Supervised learning is called Artificial Neural Networks (ANN’s). They are simply called neural networks (NN’s). Neural Networks have their name based on the neuron of the brain, which is responsible for processing information.

The base element of the neural network is a neuron, which receives an input, multiplies it by weight, adds a bias, and then returns the result for further processing usually to another neuron. The function of a neuron is:

|  |  |  |
| --- | --- | --- |
|  |  | (2.6) |

In Equation 2.6 represents the input; represents the bias; represents the weight and is the output. Normally the neurons are placed in a layer, with the result from the layer before being the input of the previous layer.

After computing the weight, a scalar function is used to aggregate a layer into a single input value. Once multiple weights are calculated and summed an activation function is used to transform the output. A few examples are Linear, ReLU (Rectified Linear Unit), or Sigmoid activation. The Linear function is typically used as the last activation function for regression-type tasks such as price prediction. The Sigmoid is used for non-linear classification type tasks such as dogs vs. cats.

|  |  |
| --- | --- |
|  |  |
| Figure 2 ReLU Activation Function | Figure 3 Linear Activation Function |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | (2.7) |  |  | (2.8) |

Generally, the artificial neural network is divided into three named parts known as the input layer, the hidden layer, and the output layer. [9]

* The first layer is responsible for receiving the data,
* The hidden layers are made up of neurons that are extracting patterns and relevant information. This layer has most of the computational work of the network.
* The last layer is giving the result of the neural networks.



Figure 4 Neural Network Architecture

There are multiple types of neural networks: [9]

* The single-layer feedforward architecture has only an input layer and an output layer.
* The multiple layer feedforward architecture is made up of the input layer, the output layer, and one or more hidden layers. This type of neural network is also called a deep neural network.
* The feedback or recurrent architecture is using the result of the neurons as the feedback input of the same neurons. This feature is good for dynamic information processing, such as time series forecasting.



Figure 5 Feed Forward Neural Network

### Learning Process

The process of learning in neural networks is called training. The basis of neural networks is that they are capable of learning from exposure to data samples and retaining the patterns in them. After the training, the network can create a function that maps the inputs and outputs, such that in the future when exposed to unknown data it can generalize solutions that are close to, if not the desired output for the task given.

The training step is applying the algorithm for tunning the weights and the biases of the neurons with the purpose to generalize a future solution better than the previous training step. In practice, the data is split into two datasets: the training and test data. The test data is used to measure the accuracy of the model after the training is finished. It is not seen by the neural network during the training phase. For this reason, the training data is the bulk of the information, containing between 60%-90% of the data [9]. Each complete exposure of the training data to adjust the weights and the biases is called an epoch.

The learning algorithm consists of minimizing the observed errors between the predicted output and the desired output. If the error decreases as the epoch (time) increase, then the neural network is considered to be learning. The error rate will never reach 0, no matter how much the network learns, it cannot be perfect. The error rate of the network is given by a cost function which is evaluated at every epoch. Depending on its increase or decrease, it will adjust the weights accordingly.

The learning rate represents the dimension of the correction in the learning process to modify the weights. A higher learning rate makes the training period shorter but can have a smaller accuracy by taking too big a step in correcting the data and “jumping” over the correct output. A smaller learning rate will have higher accuracy but will lengthen the training time. In practice, an adaptive learning rate is used. At the start of the training, the learning rate is higher and after hitting a plateau in the error rate the learning rate is decreased. This process is repeated until a minimum learning rate is reached.

### Cost Functions

There are multiple ways to compute the loss function. Depending on the prediction type, if it is a classification or a regression type. The notation is as follows: = number of values, = predicted values and = correct values. For regression there are several ways of calculating the error:

* The Mean Absolute Error (abbreviated MAE) is measured as the absolute difference between predicted values and correct values. The mean between predicted values is computed, thus MAE is calculated as:

|  |  |  |
| --- | --- | --- |
|  |  | (2.9) |

* The Mean Absolute Percentage Error (abbreviated MAPE) is measured as a percentage version of MAE and is calculated as:

|  |  |  |
| --- | --- | --- |
|  |  | (2.10) |

* The Root Mean Squared Error (abbreviated RMSE) is measured as the squaring of the difference between the predicted and correct values, calculating the mean for values predicted and then calculating the root squared. It is essentially the root squared of the Mean Squared Error (abbreviated MSE).

|  |  |  |
| --- | --- | --- |
|  |  | (2.11) |

### Backward Propagation

Given a cost function and a neural network architecture (also called a model) there is a method that calculates the gradient of the error function considering the model’s weights. This method is named the backpropagation [10], a short-term for backward propagation. The algorithm starts backward, in which the derivate of the last layer is computed first and finishes with the first layer last. As the gradients are calculated, the previous computation of gradients is reused. This gives this method computational efficiency, as gradients computations are reused rather than calculating each layer individually like in a forward propagation manner.

In the time of the training procedure of a neural network with gradient descent, the gradient of the error function is calculated considering the weights and biases. Each cycle of gradient descent modifies the weights and biases as:

|  |  |  |
| --- | --- | --- |
|  |  | (2.12) |

where is the cost function, are the biases and the weights at a cycle of the training procedure and is the learning rate.

### Deep Learning

### Regularization

A problem often found in machine learning is making a model predict accurately the training data and also new, unseen data. This term is also called generalization. There are situations in which the error function in the training and the test phase are too high. This behavior is called underfitting, a situation in which for the given data the model is too simple and not enough weights exist to capture the relation between the input and target value and the model has too much bias.

There is another situation, in which the error for the test phase is high but the error in the training phase is low. This behavior is called overfitting. This makes the model learn “too much” [11] on the training set and is unable to generalize well on unseen data, such as the test set. In this case, the model has a high variance and can variate much more than is needed. Overfitting is a recurrent problem in machine learning. It is because the purpose of supervised machine learning models is to be able to generalize well on unseen data, in a more realistic situation.



Figure 6 Underfitting, Robust, and Overfitted Model

The perfect situation is to have a low test error, where although the model has some training error it can label new data as good as it labels its training data. There are a few solutions to adjust how well a model learns. In the case of underfitting is to increase the model complexity. An overfit model is harder to resolve, but it can be done with regularization. Regularization represents any part of the model, training, or prediction procedure that is accounted for by the constraints of the training data, including its finiteness. There are a few methods of regularization:

* Modifying the loss function, in which the loss function is directly modified to consider the normalization of the learned parameters
* Modifying data sampling. Sometimes the data is not enough to create a relation between input and target label. These methods try to manipulate the little amount of data to create a better representation of the actual input distribution.
* Changing the training approach.

In modifying the loss function, it is modified the cost function to include the weighted (Lasso Regression) and (Ridge regression) [12] normalization of the weights that are being optimized. This method prevents the weights from becoming over-variate and avoids overfitting. For example, the MAE loss function with the addition of the L1 and L2 regularization.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | (2.13) |  |  | (2.14) |
|  |  | | | | (2.15) |
|  |  | | | | (2.16) |

Where = number of values; = number of neurons; = correct value; = predicted value with respect to weight . The regularization sums a squared parameter as a penalty to the cost function while the regularization adds an absolute value parameter as a penalty.

In modifying the sampling method, there are two options: Data augmentation and Cross-Validation [13]. Data augmentation is done by increasing the quantity of available data by artificially augmenting existing data to create more input data. This is done by adding a random set of functions, differing by the type of data. For image data, it can be cropping, dilating, or rotating the image. For sound data, it can be noise pitching or time stretching.

The previous chapter there has been stated that there are two subsets for the data: train data and test data. To implement cross-validation a third subset of data is created. It is called the validation data. This portion has about 20% of the overall data. At each epoch’s end, the validation data is used as test data. This means that the training error will not be taken into account, but the validation error. This prevents overfitting because the error considered is not on training data but new unseen data.

The model architecture can be altered by adding a regularization layer called Dropout [14]. This type of regularization is frequently used and has the best results in deep learning. At every iteration, Dropout randomly selects some nodes with a probability and removes them, along with their input and output connectors as well. This method reduces the variance that the output can produce since there are fewer nodes.



Figure 7 Neural Network model without and with Dropout

It is often encountered in machine learning a huge disproportion between features. For example, there is the age and income as features. While age usually does not go higher than 100, income can be 100 times bigger than the age. This will cause the income to have a bigger influence on the outcome than the age, although they are equally important.

To overcome this disproportion of features, normalization must be applied. Normalization is a technique used in machine learning as a part of data preparation. The goal is to transform numerical features to a common scale without distorting the true value. The values end up in a range between 0 and 1. It is also called Min-Max [15] scaling and is defined as:

|  |  |  |
| --- | --- | --- |
|  |  | (2.17) |

Where is a feature; is the minimum value of a feature; is the maximum value of a feature.

### Optimization

The biggest problem with deep neural networks is the time required for calculating the gradients of the error function considering the model’s weight and bias. Since deep neural networks have at least one hidden layer they have more gradients to compute rather than single feedforward architecture. The backpropagation algorithm does not specify how the gradient is used to modify the model’s weights.

The first method for calculating the error function’s gradients is Gradient Descent. It is the first and most basic gradient algorithm which you can apply to train a deep neural model [16]. Sometimes it is named “batch gradient descent” or “deterministic gradient descent”. That is because the parameters are updated after seeing an entire batch of the training data and the gradient is deterministically computed. Gradient descent updates the parameter with a small step towards the minimum. The Gradient Descent algorithm is defined as:

|  |  |  |
| --- | --- | --- |
|  |  | (2.18) |

Where = learning rate; = model’s weights and biases at time , = predicted values for input values and model parameters ; = true values, = loss function for predicted values and true values. Gradient Descent reaches convergence with a fixed learning rate as the gradients get smaller.

Many algorithms are based on Gradient Descent to optimize the gradient computation such that convergence can be reached faster. While Gradient Descent will eventually reach convergence it can be pretty slow, which is the reason why it is not used as much in deep learning anymore. There are some situations when the gradients can take a bigger step. For this purpose, Momentum has been created.

Momentum is intended to accelerate the learning process and is inspired by physics. It is just like an object which gains momentum when it is descending on a slope. The longer the slope has the same inclination, the object will gain speed. If the slope becomes steeper the object will accelerate even more. The same logic can be applied to the Gradient Descent algorithm:

|  |  |  |
| --- | --- | --- |
|  |  | (2.19) |

The velocity will take into account previous calculated gradients and, if they consistently are the same, the momentum will grow.

The learning rate is a hyperparameter that is pretty difficult to set because it can have a high impact on the accuracy of the network. While momentum has mitigated some of the problems of the learning rate, its adjustment is still difficult and needs more tuning.

For this purpose, adaptive learning rate algorithms have been made [17]. The first algorithm introduced is AdaGrad [18]. Although a few variants of Gradient Descent algorithms have emerged, the problem with fine-tuning the hyperparameters still exists. The hyperparameters are to remake the training procedure of the neural network in concordance with the requirements of the dataset. AdaGrad adaptively scales the learning rate for all parameters of the model by scaling them inversely proportional to the sum of squared partial derivates from the training epochs. This will make the parameters with a greater gradient have a higher learning rate decrease while the parameters with a lower gradient will have a lower learning rate decrease. The downside of AdaGrad is that for training deep neural networks the cumulation of squared gradients will result in a decrease in the learning rate’s effectivity.

RMSprop [19] addresses AdaGrad’s problem by shifting the gradient accumulation into an exponential weighted moving average. Adam [20] is another adaptive learning algorithm and is mostly viewed as RMSprop with momentum. The distinction is that Adam is directly integrated into the algorithm as an estimate of the first-order momentum of the gradient.

Further optimization can be achieved by using parallel computing. Weight adjusting in a neural network is done by using matrix multiplications in which there is no need to wait for every element-to-element multiplication and can be done in parallel to speed up the training process. A small deep learning model can take from a few minutes or even an hour to train but a deep learning model with millions of trainable parameters can take days to train.

A graphics processing unit (abbreviated GPU) is specialized to accelerate the creation of images and uses parallelization for efficient computing. Although it is used most commonly in video games it has been recently used in deep learning instead of the central processing unit (abbreviated CPU). Although it has a smaller memory than the CPU, a GPU computes more efficiently the weights for deep learning training.



### Convolutional Neural Networks

For more complex and bigger-sized data a simple linear function with weight and bias is not enough. For this, the traditional function of a neuron is replaced with a convolution. It represents the application of a matrix filter to the weights. Since the convolution operation takes multiple dimensions, the CNN is the best choice for image classification, but it can be used in other areas such as text classification.

The Convolutional neural network (abbreviated CNN) is a neural network that contains at least one convolutional layer. In a convolution, the data input is represented as a tensor with the following shape: count, height, width, and channels. The tensor is transformed into a feature map, widely known as an activation map with the same shape. The convolution operation is analogous to how a brain neuron is reacting in the visual cortex to a stimulus. A convolutional neuron takes a part of the image, like a receptive field. A convolutional network can withhold temporal and spatial information in its input through the application of filters. Instead of operating on a normal linear function, a convolution uses multidimensional filters that “hover” over the image to extract information. In the case of a two-dimensional convolution, it would be a matrix of a small area, usually three by three or five by five.



Figure 8 Convolutional Neural Network

In real situations, images have high resolution, which can make the CNN model heavier with weights. For example, an image with the size of one hundred square pixels will have ten thousand weights for each neuron. It is no need for that much information. The essential information can be extracted with a down-sized picture such that the CNN model can be deeper and have relevant weights. Information down-sizing can be applied to most data that will be used in a CNN model.

### Geometric Deep Learning

The best way to represent a spatial dependency is with Graphs. A graph is an abstract data type that can be used to represent non-linear relationships between objects and locations. A graph is made up of nodes (sometimes called vertices) that are interconnected with edges (sometimes called arcs, lines, or links). There are two types of graphs depending on the type of edges: directed graphs in which the edges have a direction (are asymmetrical) and undirected graphs in which the edges representation do not have a direction (are symmetrical).

A graph can be represented as [21] where represents the nodes and the edges. The function is a mapping function which attributes an edge to an unordered pair of edges. This representation of is different for a directed graph where is a mapping function which attributes an edge for an ordered pair of edges. In the above definition, there can be a situation for the mapping function in which the nodes are identical, which is called a loop. A loop allows an edge to start and end in the same node. The graph can be restricted to exclude loops by adding the in the mapping function.



Figure 9 Graph Example

Graphs have a multitude of real-life applications, they can be used to model processes in biological, physical, and information systems [22]. Graphs have been receiving more attention in machine learning due to their great expressive power. They can be used to describe systems across various domains such as social networks, physical systems, protein interaction systems, or even knowledge graphs.

There are multiple types of graph classifications. There is node classification, in which the nodes contain information to be classified. Edge classification is where the edges contain information to be classified. There are also two types of graphs from a temporal point of view: dynamic and static graphs. Dynamic graphs may have their nodes and edges changed but static graphs remain the same. As with the signal received in the graph, they can be dynamic or static as well.

Graph neural networks (called GNNs) are a deep learning method that operates on graphs. To use a graph in convolutions, the theorems for the Laplacian matrix and Adjacency matrix will be presented. The adjacency matrix is a square matrix of edges, where is an edge belonging to the graph:

|  |  |  |
| --- | --- | --- |
|  |  | (2.20) |

For the situation of an undirected graph, an edge connects both nodes, so the adjacency matrix is symmetrical. For directed graphs, an edge comes from a node and goes into another but not necessarily the other way around. The graph can be defined as a weighted graph in which the edges have a weight defined as a number to mark the edge as more or less important. In this case, the adjacency matrix can be redefined as follows, where is the weight of edge :

|  |  |  |
| --- | --- | --- |
|  |  | (2.21) |

The Laplacian matrix is represented as in which represents the degree matrix. The degree matrix is diagonal and its values are the degree of each node. The degree of a node is given by counting connecting edges. . The Laplacian matrix may take three forms:

* Combinatorial Laplacian:

|  |  |  |
| --- | --- | --- |
|  |  | (2.22) |

* Symmetric normalized Laplacian:

|  |  |  |
| --- | --- | --- |
|  |  | (2.23) |

* Random Walk Normalized Laplacian:

|  |  |  |
| --- | --- | --- |
|  |  | (2.24) |

To further understand convolution on a graph a notion for eigenvectors, eigenvalues, and the Fourier transformation must be defined.

Eigenvectors (also called characteristic vectors) of a linear transformation are described in linear algebra as the non-zero vector which changes only at a scalar factor when said linear transformation is used. The value of which the eigenvector is scaled is called the eigenvalue, denoted with . A formal definition can be written as:

|  |  |  |
| --- | --- | --- |
|  |  | (2.25) |

where is a linear transformation, is the eigenvector of and is the eigenvalue.

A Fourier transformation is a decomposition of a function that depends on either time or space into multiple functions depending on spatial or temporal frequency. The Fourier transformation of a function is formally denoted . There are more ways of defining a Fourier transformation of an integrable function [23], for any real number . One of them is:

|  |  |  |
| --- | --- | --- |
|  |  | (2.26) |

A key role in graph convolutional networks that rely on the spectral method is the eigendecomposition of the Laplacian Matrix. The Laplacian matrix can also be written as:

|  |  |  |
| --- | --- | --- |
|  |  | (2.27) |

Where is a matrix made of eigenvectors ordered by eigenvalues. It is also known that the matrix is an orthogonal matrix that has .

The Convolution Theorem says that under some conditions the Fourier transformation of a convolution of two functions is the elemental product of their Fourier transformation [24]. It can be implemented by the same logic on graphs and implement the Fourier transforms on graphs. The graph convolution will be defined on input signal and filter as [25]:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | (2.29) |  |

Where ⊙ is the element-wise product

Graph Convolutional Network [26] (also abbreviated GCN) is based on and simplifies ChebNets architecture and uses graph convolutions. The output of the convolution is:

|  |  |  |
| --- | --- | --- |
|  |  | (2.30) |

GCN is spatially localized which combines benefits from spatial-based and spectral-based methods. Each output is based on the weighted aggregation of the node itself and the neighboring nodes. There are 2 disadvantages to this architecture: the weights assigned to a node in the vicinity are the same, thus limiting the architecture's ability to capture correlation in spatial information. During training, the weights computing increases exponentially with the number of the architecture’s layers.



Figure 10 Graph Convolutional Network

### Recurrent Neural Network

One of the presented types of neural network architectures is recurrent neural networks. A recurrent neural network (abbreviated RNN) is a special type of neural network having neurons that receive previous inputs and affect the current input and output. In short, they are feeding themselves information. They are commonly used for temporal or ordinal problems such as time series prediction, language translation, or natural language processing (abbreviated NLP).

There are multiple types of recurrent neural networks. A widely used type is the Long short-term memory (abbreviated LSTM). LSTM [27] has been introduced in 1997 as a solution to a classic RNN problem.



Figure 11 Recurrent Neural Network

Theoretically, RNNs can keep an arbitrary long-term dependency in their input but it has a flaw: when training classical RNNs with backpropagation the long-term gradients tend to reach zero or infinity. It is also named the vanishing gradient problem (for zero gradients) and exploding gradient (for infinity gradients). LSTM partially solves this problem by allowing gradients to propagate unchanged and solves the vanishing gradient problem but can still be affected by exploding gradient problem.



Figure 12 Long short-term memory cell

LSTM operates like classical RNN in a chain-like structure but the module has a different structure. An LSTM cell can be observed in figure 12. The mechanism behind LSTM is made up of 4 stages:

The first update in LSTM is the “forget gate” layer. It is the leftmost part of the cell represented in Figure 12. The previous output will be taken and current input and apply a sigmoid layer to have a number between 0 and 1 which will determine the magnitude of how much will be discarded, with 0 being completely discarded. The forget function can be written as:

|  |  |  |
| --- | --- | --- |
|  |  | (2.31) |

The second step is to get new information with the “input gate” layer. This represents the middle part of the cell. The input gate takes input and applies a sigmoid layer to it. Next, another layer is created   with possible candidate values as a hyperbolic tangent (tanh) layer of the input. The input gate and candidate values can be expressed as:

|  |  |  |
| --- | --- | --- |
|  |  | (2.32) |
|  |  | (2.33) |

The third step is to modify the cell state with the prior three values computed earlier. This step represents the top half of the LSTM cell. The forget gate is multiplied by the prior state to discard the information. Then the input gate and possible values are multiplied as well. Finally, the two are summed and the function looks like this:

|  |  |  |
| --- | --- | --- |
|  |  | (2.34) |

The final step is to create a new output. This is represented by the right part of the cell. On the previous output is applied a sigmoid function. In the newly created cell state, a hyperbolic tangent function is applied and is multiplied with the output. The final part looks like this:

|  |  |  |
| --- | --- | --- |
|  |  | (2.35) |

# RELATED WORK

The act of transportation has been in attendance to humankind since the beginning of civilization. In addition to making vehicles faster and more effective scientists have been and still are looking for ways to optimize traffic flow and create better-optimized traffic systems. This chapter will illustrate several scientific papers of research and advances in the traffic optimization domain.

This subject was approached as early as 1959 [28]. More recent work includes a paper in 2007 [29] that used a version of the random forest algorithm using Adaboost. This paper used data collected by Jerusalem Traffic Flow management. It also used optimization to traffic lights, which heavily influences traffic.

One of the first modern approaches [30] used Regression, Historical average, “Auto-Regressive Integrated Moving Average” (abbreviated ARIMA), and “Seasonal Auto-Regressive Integrated Moving Average” (SARIMA) using data from a section of Melbourne’s Eastern Freeway as data. This paper has seen significant differences between a weekday and a weekend day, but very little difference between normal days and holidays.

A more recent study [31] used Autoencoder and stacked autoencoders (abbreviated SAE). An autoencoder is a neural network that tries to reproduce its input. A stacked autoencoder is made by stacking several autoencoders. This study used the same source for the dataset as this paper but the year represented was 2013. It can discover latent traffic flow and does not consider only a shallow structure of traffic data.

Another study [32] compared multiple methods including random forest, Support Vector Regression (adaptation for Support Vector Machines for regression problems), Multilayer perceptron, and Multiple Linear Regression with neural networks having the lowest error rate.

For this task, it is not sufficient to represent only through a temporal point of view but it must be represented from a spatial point of view. This can be achieved with Convolutions and Graph Neural Networks. One study [33] used both temporal and spatial dependencies of this task to create a Spatial-Temporal Convolution block based on a recurrent graph neural network and showed significant improvement over traditional methods.

A study [34] took the approach of predicting not just the speed but also other parameters of the same dataset as this paper, flow, and occupancy. It took multiple numbers of hidden units inside a GC-LSTM model crossed with different time aggregation. The models have performed better using smaller aggregations on speed, flow, and occupancy.

More studies on graph neural networks include “Attention Based Spatial-Temporal Graph Convolutional Networks” (ASTGCN) and its Multi-Component variant [35], Graph Multi-Attention Network (GMAN) [36], a Multivariate Time Series Graph Neural Network (MTGNN) or even hybrid recurrent and graph neural network called Graph convolutional LSTM (GCLSTM) [37].

In recent years the most promising results are in deep learning and graph neural networks due to their capabilities to encompass both temporal and spatial problems and not to rely on regression for each station which captures traffic flow and speed. This paper will focus as well on deep learning and graph neural networks.

# METHODOLOGY & EXPERIMENT

This chapter will include the methodology of this experiment, technologies, and libraries used. Further will be presented the experiments taken in this paper with the data, data processing, and experiments results and findings.

## Methodology

### Technical Implementation

For the entire experiment, a single programming language has been applied, Python 3.8. Python is the most widely used programming language for data science projects, machine learning projects, or everything in the field of Artificial Intelligence. In this experiment, it is the best choice for a programming language since it contains a variety of libraries for data manipulation, data visualization, machine learning, and hyper parametrization.

The Dataset for this project can be found on the USA governmental website PeMS [1]. More details on the dataset will be covered in the Dataset chapter. To access this dataset an account is required and then the user can access the Data Clearinghouse section and select the sector desired. After the entire dataset is downloaded it is saved in a folder called Data for further processing. The dataset is contained in CSV files, each file representing 1 day of data. Pandas [38] has been used to read the information required from the CSV. For any array manipulation such as sorting, grouping, or filtering the NumPy [39] library has been used. NumPy uses some of the best array manipulations in python and is the best choice for this type of experiment.

For most machine learning projects, Keras [40] is the choice for training. In this project and all its deep learning models, the library PyTorch [41] has been used. PyTorch contains methods for classical convolutions and all the required tools for machine learning as well as implementation for Graph Neural Network training through PyTorch Geometric [42] and a tool for Graph Recurrent Neural Networks with PyTorch Geometric Temporal [43]. PyTorch is harder to use since the training step needs to be manually implemented. But with more flexibility, it has more to offer.

Python’s machine learning libraries often support GPU integration for faster training. PyTorch offers GPU integration as well. Only Nvidia graphics cards are supported for GPUs. To enable GPU, CUDA [44] and cuDNN [45] have to be installed on the machine on which the project was conducted. The machine has an NVIDIA GeForce GTX 1650 graphics card and an Intel Core i5 10th generation. Besides training, the rest of the computing power comes from the CPU and the CPU is just as important.

In most cases, there are too many parameters to be adjusted so in practice a hyper parametrization library is used for training deep learning models. For this task, Ray Tune [46] is the tool needed. It contains a grid for hyperparameters to be chosen at random. The number of samples to be trialed can also be set. Ray Tune also provides checkpointing by saving the models’ weights state. Information such as loss is also stored such that everything is saved for later use.

Data visualization is also important for bigger datasets to gain insights. For a better understanding of the dataset, a data visualization library is used called Plotly [47]. Plotly is used to plot diverse information about the dataset to help form a better grasp of what information is processed along with gaining insight through plots. Although Plotly is used in Python in this experiment, it is also available in R or Matlab.

Libraries tend to have dependencies and conflicts. Because of this, a library manager is needed to create a good working environment. For this Anaconda [48] Framework is used. Anaconda offers support for most data science and machine learning tasks and not just for Python but R as well. Anaconda has been used to install and update every library in the development of this experiment.

### Dataset

The dataset is found online as mentioned. It represents data traffic from the city of Los Angeles, California. The website offers multiple sectors of the city. In this experiment, the 7th sector was chosen. Multiple type choices will impact what data is offered and for this experiment, the “Station 5-Minute” is chosen. This means that data is aggregated every 5 minutes throughout the day for each station. The first timestamp is 00:00 and goes throughout the whole day until 23:55 and the cycle repeats.

The data is available for multiple years and every day of the year with some exceptions. In the Available Files section, there are the files for every day of the year or up until the present if the year selected is still ongoing. At the time this experiment was started, the latest completed month was selected, which is July 2021. The heaviest congestion happens during the weekdays. To gain better insights into traffic congestion and to predict speed time during the most critical periods only the weekdays are selected.

Caltrans PeMS collects data from multiple types of vehicle detectors stations (abbreviated VDS) [49], some including side-fire radar, magnetometers, or inductive loops. A VDS sends data to its District TMC every 30 seconds.

Each file is contained in a compressed file. After the extraction, a text file remains. Data is split into rows and each column is separated by a comma. A row contains data for a station at a point in time. The station row orders are consistent throughout the file. This means that for a point in time the station will be presented in the same order as the rest. There is a total of 4904 unique stations each having a unique identifier. To measure the amount of information for a day of measurements 288 individual timestamps are recorded, multiplied by 4904 unique stations it can be seen that there are over 1.4 million rows in a single day. This amount of information will make training very slow but in the following chapters data preparation will be made and empty information will be removed, allowing the deep learning models to learn useful information promptly.

In each row the following information can be found:

* Timestamp: The date and time at which the information is retrieved.
* Station: Unique station number identifier
* District: District number
* Freeway: Freeway number
* The direction of travel: north, south, east, or west
* Lane Type: a two-character string indicating the type of the lane
* Station Length: The length of the segment covered by the station measured in miles
* Samples: Number of samples received for all lanes
* Observed: Percentage of lane points at the station location which was observed
* Flow: The sum of vehicles passing through the station over five minutes across all lanes.
* Occupancy: Percentage of occupancy across all lanes in five minutes as a decimal number.
* Speed: Average flow-weighted speed in five minutes across all lanes.

The next rows represent the columns samples, observed, occupancy and speed but for each lane.

In the type section of the PeMS website, there is an option for Station Metadata. This dataset is contained as the five-minute station one, within a single text file with columns separated by row. This file contains metadata about the station sensors. In this project, there are three variables used from metadata: the station's unique identifier, latitude, and longitude. They are used to accurately place the stations.

As mentioned in the theoretical chapter the main variables are Flow, Occupancy (Density), and Speed. This experiment will revolve around these three variables. The timestamp, as well as the station’s unique identifier, will be applied with the three variables. For a better understanding, the other variables will be used as well to get better insight through data visualization.

## Experiments

The experiment will consist of using five models: a linear regression model and two statistical models for each sensor node and three graph neural networks, one based on Spatial-Temporal Convolutions, and one with Graph Convolutional Neural Networks together with Graph Embedded LSTM. The last one is Diffusion Convolutional Gated Recurrent Unit with Graph Convolutional Neural Networks.

The dataset will be split into multiple sizes based on what sensors will be included. This means there will be taken into consideration a fixed number of sensors. The sensors will always be the same as the time-axis moves. There are four different datasets based on the information from this dataset:

* Experimental dataset, containing eight hand-picked sensors in an intersection.
* Tiny dataset, containing 50 hand-picked sensors in an intersection
* Small dataset, containing 120 randomly picked sensors.
* Medium dataset, containing 480 randomly picked sensors.

### Data Preparation

For every deep learning experiment, the information being processed needs to be prepared to have only useful information. For this purpose, the python script passes through each row and checks the correctitude of the information by checking if the corresponding station to the row contains information about speed, occupancy, and flow. If all three are present it is considered a good node and it is saved. After this step from 4904 nodes, only 2800 are left. The occupancy and flow variables are normalized using the min-max method explained in the previous chapter. After removing empty rows, the station which will be used in training is saved for later use. In this way, for all five datasets, the information for the station nodes will always be the same.

### Map Generation

Three out of the six models proposed for this project use graphs as inputs and for this reason, the graphs for training need to be constructed. The Station Metadata does not contain any information about how each sensor is connected or if they are on the same roadside. Because of this, the graph needs to be constructed manually or procedurally.

For the datasets Experimental and Tiny, the graph connectivity can be created manually. For the Small and Medium datasets, there are too many nodes for manual graph connectivity to be feasible.

There are two things to consider while constructing the graph: the adjacency (connectivities) and the weight of the edges. As such the generation task can be split into two parts. The graph generating its adjacency, how nodes are connected, a step which will be shortened as Generation. The second is the graph setting the edges' weight, now called Distance.

There are three ways for the Generation step: Manual, Generative, and Linear Regression. The manual one is the hand-picked connections and thus will not be covered. The other two will be covered in this chapter. For the distance, there are two ways of computing: Geodesic and Open Street Resource Project, abbreviated OSRM. Each type of Generation will be crossed with every type of distance to compute a graph, resulting in six possible combinations.

### Geodesic

The geodesic method is using geolocation data from metadata. The latitude and longitude of each node will be measured against every other station present in the dataset and used to compute its geodesic distance. The geodesic distance between two points is calculated with the below formula:

|  |  |  |
| --- | --- | --- |
|  |  | (3.1) |

Where P1 and P2 are the two points having latitude and longitude for P1 and latitude and longitude for P2. is the absolute difference in longitude for the points and represents the absolute difference in latitude between the two points.

### Open Source Routing Machine

The second method of measuring distance is using the actual distance between two points on the road instead of the geodesical distance. This means that the distance is the path a vehicle would take on the road, not a straight line between two points. This distance can be measured using the Open Source Routing Machine project (abbreviated OSRM) [50].

A more clear difference between the two distances can be observed in figure 13. The blue line is the OSRM distance and the red line is the Geodesic distance.



Figure 13 OSRM and Geodesic Distances

Using the geological latitude and longitude in the dataset as the input for the two points, the website computes the road distance needed to go from the first point to the second.

### Linear Regression

Linear Regression uses coefficients to determine a feature’s weight. Based on equation 2.5 the bigger the weight is the bigger an influence a feature will have on the result. In the experiment’s graph situation, a previous node in the same direction of travel should have the biggest impact on the outcome of the current node. More informally, what happens behind a node should matter the most.

Based on this principle a node selection can be used using a linear regression’s coefficients to determine the most influential nodes. Instead of using flow and occupancy as features, the outcome of the dataset’s other nodes will be used as features and the outcome of the current node will be used as labels. The top three highest coefficients will be used as the node’s predecessor in the graph. This procedure is repeated until all nodes have three nodes to be marked as previous nodes. This method will further be denoted as LR.

### Generative

The distance is measured between every station. Not every station can be connected to every other station because the temporal information will be lost so the connection of the nodes must be computed. There are several ways to consider which nodes will be connected, the following formula will be used:

|  |  |  |
| --- | --- | --- |
|  |  | (3.2) |

Where represents the weighted distance among stations and . and represents the distance among the stations. There are two new hyperparameters introduced in equation 3.2, and . These two parameters are thresholds to control the sparsity and distribution of the graph’s edges. The values for the two parameters are:

|  |  |  |
| --- | --- | --- |
|  |  | (3.3) |
|  |  | (3.4) |

These parameters will be used in deep learning hyper parametrization to obtain insights into how sparse a graph should be in its representation.

### Data Visualization

The sensors are located in the 7th district of Los Angeles and there are a total number of 4809 sensors. The selected month is July 2021 and contains 22 days of data. Weekend days were removed since most work is done on weekdays and the results would be altered.

In figure 14 all sensors are presented. It can be observed that in the city the sensors are more tightly packed together. A reason may be that the inner city’s roads are more important and more heavily circulated. It can also be noted that only the highly circulated roads are taken into consideration like freeways since it would be redundant to calculate every single road in the city.



Figure 14 All Map Sensors

In Figure 15 the sensors selected for the experimental dataset are presented. The Experimental dataset is handpicked and was selected such that the nodes selected are close to one another. There are a total of eight sensors that are placed at the intersection of Marina Freeway and San Diego Freeway. It is located a few miles from the sea and near a commercial center, in the middle of the city and it is one of the busiest intersections.

|  |  |
| --- | --- |
|  |  |
| Figure 15 Experimental Dataset | Figure 16 Marina and San Diego Freeways Intersection |

Like the experimental dataset, the Tiny Dataset was also hand-picked but a different intersection was chosen. The intersection between Santa Monica Freeway and Harbor Freeway is considered for the Tiny dataset. In this case, the nodes stretch more outside the intersection, almost reaching the nearby intersections. This will give more information on surrounding traffic. The nodes representing the Tiny dataset can be found in figure 17 as well as a view from the overpassing Santa Monica Freeway in figure 18.

|  |  |
| --- | --- |
|  |  |
| Figure 17 Tiny Dataset Nodes | Figure 18 Santa Monica Freeway Overpass |

The Medium dataset is represented in figure 19 and the small dataset is represented in figure 20. For these two datasets the nodes are chosen randomly and they do not have to be near one another. It can be noted that the small dataset is sparser than the medium dataset. As stated in the data preparation the two hyperparameters and will be a major factor in the results of the experiment as the medium dataset will be more well connected.

|  |  |
| --- | --- |
|  |  |
| Figure 19 Medium Dataset | Figure 20 Small Dataset |

The obtained adjacency matrices are obtained using equation 3.2 and they have a size of 8x8, 50x50, 120x120, and 480x480. There is a total of 16 different adjacency matrices for each dataset. Four of the represented matrices are represented in figures 21, 22, 23, and 24. It can be observed as sigma increases the connectivity of the graph is increased. On the other hand, as epsilon increases, it restricts connectivity. For a lower epsilon and a higher sigma, there are more different weights, not just 1 or 0. In figure 19 as it can be observed in more middle ranged colors. Not all the nodes are connected. This behavior is normal since not all the roads’ node points are interconnected between them.

|  |  |
| --- | --- |
|  |  |
| Figure 21 Graph Heatmap for epsilon 0.1 and sigma 1 | Figure 22 Graph Heatmap for epsilon 0.1 and sigma 10 |

|  |  |
| --- | --- |
|  |  |
| Figure 23 Graph Heatmap for epsilon 0 Graph Heatmap for epsilon 0.7 and sigma 1 | Figure 24 Graph Heatmap for epsilon 0 Graph Heatmap for epsilon 0.7 and sigma 10 |

The procedural methods of obtaining the graph map tend to have more node connections than in reality. For the Generative method, the intersections would be misinterpreted because all nodes are near each other so all nodes in an intersection would be interconnected, which is not true. This method would be better for roads without intersections, as it would choose the closest nodes only, based on sigma and epsilon values. The Linear Regression node selection method is more suited for intersections as it chooses the most influential nodes only. Figure 25 is presenting the graph heatmap for the linear regression node selection method and in figure 26 the manual node selection method is presented. Both figures are for the Tiny dataset.

|  |  |
| --- | --- |
|  |  |
| Figure 25 Linear Regression Node Selection for Tiny Dataset | Figure 26 Manual node selection for Tiny Dataset |

There are specific time intervals in which traffic is heavier, usually called rush hour. Figure 27 is presenting a speed map of the traffic at 18:00. Six in the afternoon is considered a rush hour since most work schedules end at this hour and every commuter goes home. Figure 28 is a speed map of the traffic at 22:00.

With the increase in traffic participants so does the waiting time. In figure 27 the center part of the district is colored more in blue signifying heavier traffic with some blue parts along with the outer areas. In figure 28 most of the roads are covered in red, as the roads are clearer during the night.

|  |  |
| --- | --- |
|  |  |
| Figure 27 Traffic speed at 18:00 | Figure 28 Traffic speed at 22:00 |

In figure 29 the daily average Speed is presented. The maximum speed limit for a freeway in California is 65 miles per hour (mph). On all days the median is at 65 mph while the upper standard deviation is at around 70 and the lower at 60.



Figure 29 Daily Average Speed

There are exceptions in both upper and lower deviation that are normal. There are traffic participants that do not respect the speed limit and drive over the speed limit. It can also be observed that the lower deviation is higher because of traffic. There are no major discrepancies between the days of the week.



Figure 30 Road Types

Multiple types of roads can be fitted with a sensor. They aren’t placed only on the inner freeway of the city but in multiple places. The types distribution of road types is presented in figure 30. The majority of sensors are placed on the mainline of the freeway (39%) while the secondary is on-ramp roads (22%) and the third is the High occupancy vehicle (HOV) lanes (17%). High occupancy lanes are lanes restricted to vehicles with a higher occupancy rate than cars. It is expected for the largest number of sensors to be on the mainline of the freeways. There are other types of roads such as Off ramp, Freeway intersection connectors, or Community college district (noted Coll/Dist).

### Models

There will be six models tested. The first model represents a Linear Regression model used as a baseline. The model takes each sensor data for the entire month of July and makes its prediction using the technique explained in the theoretical chapter. Note that this model will not include the data from its neighbors.

The next two models are statistical models, namely Auto-Regressive Integrated Moving Average (ARIMA) and Seasonal Autoregressive Integrated Moving Average (SARIMA). These models have only temporal data, just like the previous model.

The fourth model is a temporal geometric neural network model based on an existing model [33] presented as managing to encapsulate spatial dependencies as well as temporal dependencies.



Figure 31 STConv Structure

It consists of three blocks Spatial-Temporal Convolutions separated with a ReLU activation. Each block is made of two Temporal Gated convolutions with a Spatial Graph Convolution between them. Each Temporal Gated Convolution contains a gated linear unit (GLU) with a one-dimensional convolution. This infrastructure is represented in figure 31. This model will be denoted as STCONV.



Figure 32 GC-LSTM

The fifth model is composed of two elements. The first element is an LSTM embedded Graph convolution based on the GC-LSTM [37]. The second part is a simple Graph Convolution. The GC-LSTM block is represented in figure 32. At each time step, a graph convolution is passed to an LSTM cell. The LSTM cell keeps track of previous information as well. This model will be denoted as GCLSTM.

The sixth model is composed of two elements like the previous LSTM-based model. Instead of using LSTM, this model will use Diffusion Convolutional Gated Recurrent Unit [51] together with Graph Convolutions. This model will be denoted as DCRNN.

### Hyper Parameter Tuning

There are multiple parameters to consider in creating a model for the three Graph Neural Network models. To test and measure different combinations hyperparameter optimization is used. The parameters taken into consideration are as follows:

* Sigma: [1, 3, 5, 10]
* Epsilon: [0.1, 0.3, 0.5, 0.7]
* K: [1, 3, 5, 7]
* Hidden Channels: [4, 8, 16, 32]
* Optimizer Type: [Adam, AdamW, Adamax, RMSprop]

Each takes one choice from the array they are configured in. Sigma and Epsilon are used in map generation. K and the hidden channels are used in model building and the optimizer type is used at the training step.

For the Tiny and Experimental datasets, there are three generation types: Manual, Generative, and Linear Regression node selection. The small and medium datasets have only the Generative. Manually picking hundreds of node connections is not feasible. For all datasets, there are two distance generation types: Geolodesic and ORM. This brings a total of eight possible dataset configurations.

For each possible combination of the three GNN models and each dataset configuration, a hyper-parameter tuning is made. In the case of manual and Linear Regression node selection, the sigma and epsilon parameters are used for the graph’s weights. There are taken only 16 samples of combination to be tested. The total number of combinations possible is possible combinations which is too much.

For each trial of combinations, a total of 30 epochs are granted. Not all trials reach 30 epochs, some are stopped by Early stopping to save time if the model does not improve after 5 epochs on the validation step. If a trial does not perform better than the top trial after 10 epochs an early stop is called as well, also called a grace period.

### Results

This chapter will be split into two subcategories: Graph neural network results and non-Graph neural network results. In the first section, the overall results are presented, including the non-GNN results but with a focus on Graph neural networks models. In the second subchapter the Linear Regression, ARIMA, and SARIMA will be covered more.

### Graph Neural Network Results

The results are split into multiple categories. The first one is the model type which was introduced in the previous Models chapter crossed with the dataset size category, totaling four different sizes explained in the data visualization chapter. Next will be the type of distance used in the graph’s weights. It is described in the map generation and finally, the type of graph generation, which is also described in the map generation chapter. These categories are crossed with each other as specified in the previous chapters, except for those that are not Graph Neural Networks. The Losses are specified in double precision.

Over the results specified earlier the losses will be calculated with the following functions: Root Mean Squared Error (RMSE), Mean Average Percentage Error (MAPE), Mean Average Error (MAE), and Mean Squared Error (MSE), Their equations are a part of the theoretical chapter under equations 2.9, 2.10 and 2.11. The results are shown in figure 33.

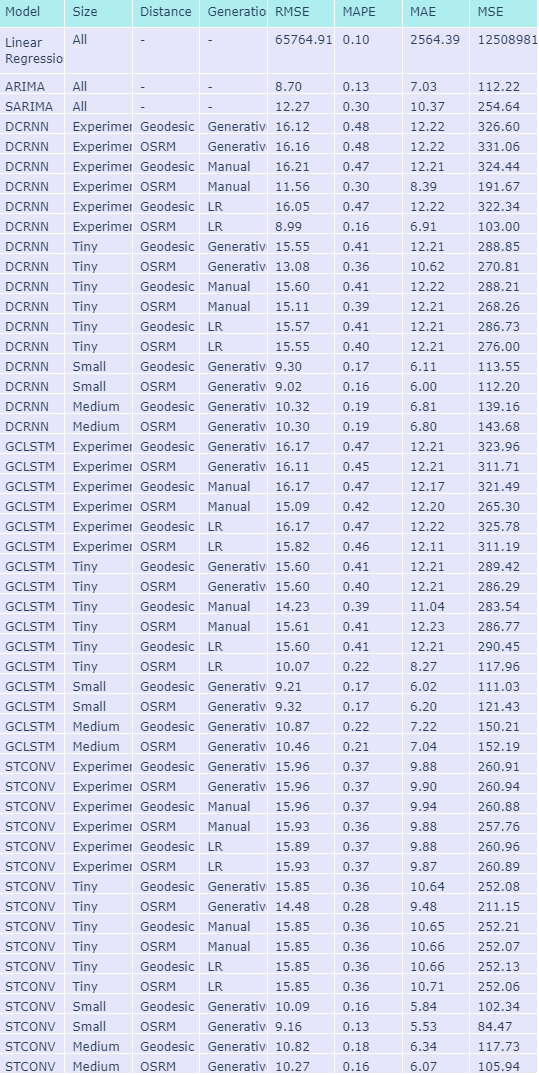


Figure 33 Final Results

First, the three models which are not Graph Neural Networks models are covered (the Linear Regression, ARIMA, and SARIMA models). These models have been trained individually over every single node which contains valid data and have not taken into consideration any other nodes' data. For them, there will be no distance or generation data. The Linear Regression models perform overall the poorest, having 2564 Mean Average Error. That is because there are results for some nodes which have very high losses and bring the average error up. This subject will be expanded more later when figure 42 will be covered.

The ARIMA and SARIMA models performed very well, for some cases even better than the Graph Neural Networks. For ARIMA the Mean Average Error is 7.03 and for SARIMA is 10.37. These statistical models take into consideration the moving average and the period over time that is produced, unlike the Linear Regression which takes only into consideration the data it is provided.

The results covered next are all made by the three Graph Neural Network models which will include the distance and generation data because the models use graphs as data. Next, the distance used in map generation will be covered. For most, if not all the real distance between two road points, the OSRM type, has performed better than the Geodesic counterpart. The difference can range from a few decimals to a few points. For example, in the STCONV model and the Small dataset, the Geodesic result is 5.84 while the OSRM result is 5.53.

Another great difference is the DCRNN model and the Experimental dataset and the LR Generation the geodesic result is 12.22 while the OSRM result is almost half at 6.91. This result is expected because the OSRM distance type represents how vehicles would travel on the road, whereas the Geodesic type represents a straight line on the geographic map and consider only the closest physical node, not the closest travelable node.

Next, the generation type is covered. Mostly the Generative part in combination with the Geodesic distance performed the worst, that is because it reflected the real traffic map the least. If two particular nodes are the closest that does not mean that they are accessible to one another. The difference between the results is a few decimal points, comparing it to the manual and Linear Regression types. An example is the GCLSTM model in combination with the Tiny dataset and the OSRM distance type the Generative result is 12.21, the Manual result is 12.20 and the Linear Regression is 8.27. The unexpected result is that for most results the Linear Regression result gave better results than the Manual type. In the Manual type, the graph map was hand-picked, not generated, and is reflecting the exact real-world traffic map. This may be because in the Linear Regression type the nodes considered neighbors are the ones with the highest weight in a Linear Regression prediction for the traffic speed. As such the nodes which will be linked will have the most impact on the result.

Lastly, the results for the dataset size will be covered. Between the Tiny and the Experimental, the losses do not differ too much. For DCRNN and GCLSTM model on Tiny and Experimental datasets the losses are mostly around 12 and for the STCONV model and the same datasets are around 9 to 10. The larger datasets like Small and Medium performed better than the smaller datasets, having around 5-6 loss for every model type. With more information about the entire traffic system, the models can create a more realistic map and imitate real-life traffic maps.

The best results for each dataset are as follows: For Experimental 6.91 with DCRNN model, Tiny 8.27 with GCLSTM model, Small 5.53 with STCONV model, and Medium 6.07 with the STCONV model. For the Experimental and Tiny both the best results have LR generation type and OSRM distance type. The overall best model from all scenarios is the Small dataset with the STCONV model with a 5.53 loss. The drawback of this good result was the training time necessary for this dataset, which took considerably longer than the smaller ones.

From a model type perspective, each model performed well in some area. The STCONV model performed worse on smaller datasets but better on the larger ones and was the slowest to train as well. The DCRNN and GCLSTM model performed exactly opposite, they have better results on the smaller dataset, Tiny and Experimental, but worse results for the larger ones. They were also trained faster than the STCONV model.

The results can be used in a real-world situation to predict traffic nodes at different times to try and use the information to find better alternatives in some situations. Considering that the information will be used only during rush hours because the only time the need for an alternative route is only when traffic is unexpectedly more heavy than usual. For the smaller datasets, a 12-mile-per-hour difference can mean the difference between 10 or 24 miles per hour, which will heavily impact alternate route computing. Although most of the time the distances traveled is greater than the smaller datasets and a vehicle would need to traverse most of the city. For such distances, the larger datasets are more suited for the job, both representing the entire city, with Medium having a denser node representation. In this situation, the results between 5-7 miles per hour difference can be more reliable and be used better. If a vehicle moves at 5 miles per hour or 10 it will still be mostly at a standstill and an alternative route should be found, which is the exact type of information that should be uncovered in pathfinding.

The next results to be covered will be the sigma and epsilon hyper-parameter possibilities. A table containing all Sigma possibilities on its column and all Epsilon possibilities on its row indices is presented in figure 34.

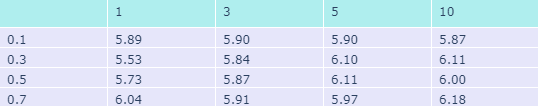


Figure 34 Sigma and Epsilon's best losses

The losses are higher going with sigma and epsilon having higher values. The best results are with an epsilon of 0.3 and sigma 1. This is also the best-performing model covered earlier. Also, the models using sigma 1 and 3 and with epsilon 0.3 and 0.5 performed similarly, having around 5.8 loss. This means that the graphs which had medium to low connectivity and did not eliminate the low weight connections had the most success since a higher sigma means a higher connectivity graph and a higher epsilon means eliminating low weight connections.

Figure 35 is presenting a boxplot for the three GNN models over the four types of dataset sizes and the three types of graph generation types. The only distance type considered is the OSRM distance type since it had the best results. The boxplot shows a Mean Average Error over all nodes during all training steps.

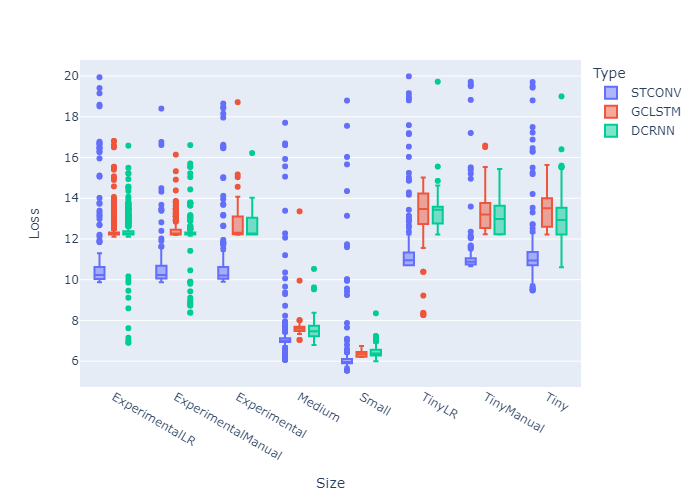


Figure 35 GNN MAE Boxplot

For most models, the values are tightly packed around the minimal value, with outliers on the upper part of the plot. That is normal since most models start with a higher loss but as they try to minimize it the loss becomes lower. Up to a certain point, the loss will start to decrease slower to try and not overshoot the minimum. This is due to a decrease in the learning rate. There is an exception to that rule, The Experimental dataset with a Linear Regression map generation type in combination with the DCRNN model has outliers lower than the interquartile mean. This may be caused by a more good hyperparameter combination than usual and decreasing drastically the overall loss.

The STCONV has the interquartile range between 10 to 11 for Experimental, 11 to 11.5 for Tiny but for Medium it is at 7, and for Small at around 6. The GCLSTM and DCRNN models have a higher interquartile range at around 12 to 14 but have a lot lower outliers. In general, the median is very close to the lower quartile or the minimum of the boxplot (not counting outliers).

In Figures 36 and 37 a line plot is shown for the training over time of two DCRNN and STCONV models.

|  |  |
| --- | --- |
|  |  |
| Figure 36 DCRNN Train | Figure 37 STCONV Train |

In both figures, the loss starts at 15 or 16 and then rapidly decreases. After a few epochs, the loss starts to not decrease with such a velocity and even starts to plateau. After this point, the learning rate is reduced. There can still be a chance that the loss will “overshoot” the minimum and starts to rise instead of decrease. This behavior is trying to be corrected but if after 5 epochs it fails to bring the loss back or lower than it was then the model will initiate an early stopping and the model with the best results on the validation data will be taken into consideration. The behavior can be observed in the DCRNN model at the 10th epoch, where the loss is higher and tries to correct itself but is unable to reduce the loss after 5 epochs. On the other hand, the STCONV model has been corrected successfully a few times, at the 7th, 10th, or 13th epoch.

To better see the chosen values a prediction for a single node, 760256, was represented in figure 36. The data chosen is the same test data chosen for the final results. The figure represents 5 days’ worth of data.



Figure 38 Actual and Predicted values for node 760256

The data has 5 rush hour peaks, each at a similar distance to one another. During the rush hour, there is no error and the model predicts accurately at what speed the vehicles are moving through this node. As stated before, during this period this information is the most valuable, being able to predict when a certain node will be more congested than usual will have higher importance than knowing a node has normal traffic. Still, it is important to know if a node is free so that it is chosen as a better alternative but still it is better to know which nodes to avoid first of all. During the free period of the day, there is a slight error of about 5 miles per hour. The difference between going at 65 or 70 miles per hour is not that important, the important information in this context is knowing that it is not congested and can be used as an alternative route. Additionally, this plot does not show the entirety of the 20 days of data, only a quarter. As such a path that would require a long time of traveling could still be effective.

### Non-GNN Results

Just as for the Graph neural network results, the three boxplot plots are represented in Figures 39, 40, and 41. They represent the losses on each model which is not a Graph neural network (Linear Regression, ARIMA, and SARIMA). The plots show the result of each node, rather than a specific dataset since these models do not use a specific dataset size but rather all valid nodes.

|  |  |
| --- | --- |
|  |  |
| Figure 39 ARIMA Boxplot | Figure 40 SARIMA Boxplot |

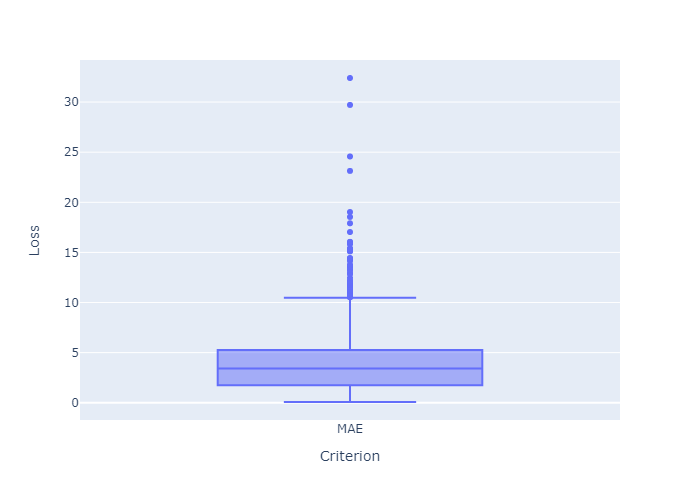


Figure 41 Linear Regression Boxplot

Through these plots, each node's loss can be better seen, rather than a Mean Average Error (MAE). For the Linear Regression plot, the higher values have been eliminated so that the plot is more visible (the MAE for Linear Regression is not 3 as in this plot but rather around 2000). For the ARIMA model the loss can go from 25 to 0, and for the SARIMA model from around 40 to 0 as well. The Linear Regression model did not perform that well generally and had losses in the millions for some nodes while most of them had acceptable losses.

These results can show that not only complex Graph neural networks can provide great results. As for the time-loss efficiency, the two statistical models (ARIMA and SARIMA) performed overwhelmingly faster than their graph neural networks counterparts. If time is more important the statistical models are the better alternative.

To visualize where the models predicted better and where the losses are higher a plot with the heatmap of the losses for each valid node over the ARIMA model is presented in figure 42.

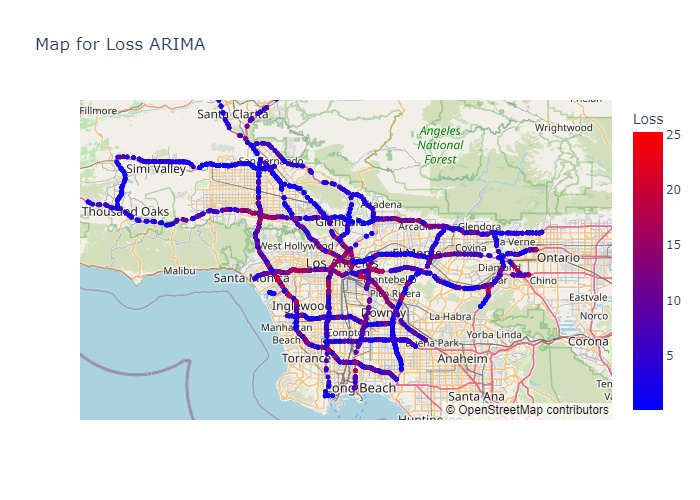


Figure 42 Heatmap loss for ARIMA

It can be observed just like in figure 39 that most of the model has a low loss, by the coloration of blue of the heatmap. The place where the model has the biggest issue is downtown intersections while the outlier freeways of the city have more blue nodes. This can indicate that the model has trouble predicting speed during congestion since most congestion happens at intersections. This can make the statistical models less viable since the critical information for pathfinding is finding where the nodes are congested. Although not all intersections are colored entirely with red, meaning there is some degree of accuracy for the congested intersections. This means that to have the statistical models trained faster some compromise must be made in terms of results for the congested nodes. Still, they can be used to find free portions of the freeway, as freeways that do not have an intersection are colored in blue.

# CONCLUSIONS & DEVELOPMENT

Traffic is a global problem whose effects can be seen on the general population or even a country’s economy. This problem has caught the attention of scientists but it is a recurrent issue for many countries' governments as infrastructure stagnates but traffic increases. Multiple countries have developed road sensors that capture traffic information for law enforcement as well as key factors for traffic flow minimization such as speed.

Recent evolvement in the field of machine learning with the rising popularity of deep learning could use such data to anticipate the speed at a certain sensor based on prior experience. With the improvement of Geometric Deep Learning and the usage of Graphs, the spatial dependencies can be resolved. The traffic system can be represented as a whole instead of calculating each sensor speed. This mimics real-life scenarios as the surrounding traffic impacts the traffic in a certain location.

Linear Regression and statistical models can still be a good baseline for traffic predictions in some situations. With enough data for each sensor, it can provide even better results than Graph Neural Networks. These results come at a cost since they cannot represent the whole system as well as Graph Neural Networks because some sensors may lack information and do not take into consideration neighboring sensors.

Due to the high amount of information the computational power needed to represent the entire system is too great. Because of this reason, only a part of the system is represented. Out of the 4000 sensors available, only 2800 contain information but at most 480 at a time can be used. With a better optimization of the training algorithms or faster GPUs and CPUs, the whole system can be taken into consideration as well as more information to be trained, not just a month.

Another setback for this experiment was the large amount of data preparation needed to experiment on more models. Some Temporal Geometric Neural Networks model needs data to be more heavily prepared for usage. Different models could be used such as ASTGCN or GMAN.

Different cities can be taken into consideration. Los Angeles does not have such heavy congestion as the top-heaviest traffic cities in the world. Data from cities with more densely packed populations and lower infrastructure like Bucharest could provide more insight.

# BIBLIOGRAPHY

|  |  |
| --- | --- |
| [1] | "Caltrans Performance Measurement System (PeMS)," California Deprtament of, [Online]. Available: https://pems.dot.ca.gov/. |
| [2] | B. E. a. T. L. D. Schrank, "2019 Urban Mobility Report," Texas A&M Transportation Institute, 2019. |
| [3] | "Tom Tom Traffic Index," [Online]. Available: https://www.tomtom.com/en\_gb/traffic-index/ranking/. |
| [4] | D. C.Gazis, Traffic Theory, 2006. |
| [5] | M. K. A. Treiber, Traffic Flow Dynamics, 2013. |
| [6] | A. L. Samuel, Some Studies in Machine Learning, 1959. |
| [7] | A. A. R. Sathya, "Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification," in *International Journal Of Advanced Artificial Intelligence*, 2013. |
| [8] | J. M. M. E. M. P. Mark Tranmer, Multiple Linear Regression, 2020. |
| [9] | D. H. S. R. A. F. H. B. L. S. F. d. R. A. Ivan Nunes da Silva, Artificial Neural Networks, Springer, 2017. |
| [10] | G. E. H. &. R. J. W. David E. Rumelhart, "Learning representations by back-propagating errors," 1986. |
| [11] | D. M. Hawkins, "The Problem of Overfitting," Minneapolis, 2003. |
| [12] | R. C. Moore and J. DeNero, "L1 and L2 regularization for multiclass hinge loss models," 2011. |
| [13] | D. Berrar, "Cross-validation," 2019. |
| [14] | P. Baldi and P. Sadowski, "Understanding Dropout," 2013. |
| [15] | S. K. Patro and K. K. sahu, "Normalization: A Preprocessing Stage". |
| [16] | S. Ruder, "An overview of gradient descent optimization," 2016. |
| [17] | Y. Li, Y. Fu, H. Li and S.-W. Zhang, "The Improved Training Algorithm of Back Propagation Neural Network with Self-adaptive Learning Rate," in *International Conference on Computational Intelligence and Natural Computing*, 2009. |
| [18] | J. Duchi, E. Hazan and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," 2011. |
| [19] | G. Hinton, N. Srivastava and K. Swersky, "Neural Networks for Machine Learning," 2012. |
| [20] | J. B. Diederik P. Kingma, "Adam: A Method for Stochastic Optimization," 2014. |
| [21] | E. A. Bender and S. G. Williamson, "Lists, Decisions and Graphs. With an Introduction to Probability," 2010. |
| [22] | T. Adali and A. Ortega, "Applications of Graph Theory [Scanning the Issue]," 2018. |
| [23] | G. Kaiser, " Friendly Guide to Wavelets," *Physics Today,* 1994. |
| [24] | C. D. McGillem and G. R. Cooper, "Continuous and Discrete Signal and System Analysis," 1984. |
| [25] | W. Cao, Z. Yan, Z. He and Z. He, "A Comprehensive Survey on Geometric Deep Learning," 2020. |
| [26] | M. W. Thomas N. Kipf, "Semi-Supervised Classification with Graph Convolutional Networks," 2016. |
| [27] | S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," 1997. |
| [28] | H. Greenberg, "An Analysis of Traffic Flow," 1959. |
| [29] | Y. R. Guy Leshem, "Traffic Flow Prediction using Adaboost Algorithm with Random Forests as a Weak Learner," 2007. |
| [30] | S. t. t. f. prediction, "E. Chung, N. Rosalion," in *24th Australian Transportation Research Forum, Hobart*, Tasmania, 2001. |
| [31] | Y. D. W. K. Z. L. a. F.-Y. W. Yisheng Lv, "Traffic Flow Prediction With Big Data: A Deep Learning Approach," 2015. |
| [32] | K. K. ,. J.-M. S. Charalampos Bratsas, "A Comparison of Machine Learning Methods for the Prediction of Traffic Speed in Urban Places," 2019. |
| [33] | H. Y. Z. Z. Bing Yu, "Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework," 2018. |
| [34] | F. I. Bogdan Ichim, "Predicting Multiple Traffic Features using a Spatio-Temporal Neural," 2022. |
| [35] | S. Guo, Y. Lin, N. Feng, C. Song and H. Wan, "Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting," 2019. |
| [36] | X. F. C. W. J. Q. Chuanpan Zheng, "GMAN: A Graph Multi-Attention Network for Traffic Prediction," Xiamen, 2019. |
| [37] | X. X. Y. W. H. Z. Jinyin Chen, "GC-LSTM: Graph Convolution Embedded LSTM for Dynamic Link Prediction," 2018. |
| [38] | "Pandas," [Online]. Available: https://pandas.pydata.org. |
| [39] | "NumPy," [Online]. Available: https://numpy.org. |
| [40] | "Keras," [Online]. Available: https://keras.io. |
| [41] | "PyTorch," [Online]. Available: https://pytorch.org. |
| [42] | "PyTorch Geometric," [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/. |
| [43] | "PyTorch Geometric Temporal," [Online]. Available: https://pytorch-geometric-temporal.readthedocs.io/en/latest/notes/introduction.html. |
| [44] | "CUDA," [Online]. Available: https://developer.nvidia.com/cuda-zone. |
| [45] | "cuDNN," [Online]. Available: https://developer.nvidia.com/cudnn. |
| [46] | "Ray Tune," [Online]. Available: https://docs.ray.io/en/ray-0.4.0/tune.html. |
| [47] | "Plotly," [Online]. Available: https://plotly.com. |
| [48] | "Anaconda," [Online]. Available: https://www.anaconda.com. |
| [49] | PeMS User Guide, 2020. |
| [50] | "Open Source Routing Machine Project," [Online]. Available: http://project-osrm.org/. |
| [51] | R. Y. C. S. Y. L. Yaguang Li, Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting, 2018. |