UNIVERSITY OF BUCHAREST
FACULTY OF MATHEMATICS AND
COMPUTER SCIENCE
MASTERS OF ARTIFICIAL
INTELLIGENCE

Dissertation Thesis

Spatial-Temporal Neural Networks for Traffic Systems

Coordinator
Conf. dr. Ichim Bogdan

Graduate
Bomher Sebastian

BUCHAREST,
February 2022

Abstract

In the present days vehicles are getting faster and more affordable to common people. This causes people to buy more vehicles, especially cars which slows down traffic. This causes a new modern problem to arise, traffic. Traffic optimization is a modern problem for urban areas to try and minimize waiting time.

With a rise of Artificial Intelligence in the past decade new solutions can be found for previous unresolvable problems. 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 taking.

This experiment will build a traffic prediction model based on information acquired from the public roads from the city of Los Angeles. The dataset is made public on the governmental site PeMS developed by the California Department of Transportation. The roads are only in district 7 of Los Angeles and are mainly on the freeway.

For this experiment a better insight is offered through data visualization of the extracted data. Insight about speed, traffic flow and lane occupancy are acquired and analyzed. A geographical view of the sensor is created to improve the perspective. The dataset is cleaned and only data which is existent and useful will be used for classification. Three subsets of the dataset will be used to gain insight about how the volume of information will affect the model's predictions.

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

The final chapter will cover the experiments results, data visualization on the experiments results on different loss functions and the three different datasets as well as conclusions and future development proposals.

Table of Contents

I. 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 for traffic flow theory and what it represents, as long as for the machine learning algorithms such as Artificial Neural Networks, Deep Learning and Geometric Deep Learning.

The second half of the paper will include the methodology used for this project and how it was built as well as the experiment along with its results. Related work in the domain of machine learning with traffic flow will be included be included as well. The methodology will explain tools and software programs used for the experiment, data preparation as well as data visualization. Data is acquired from Los Angeles traffic sensors mounted on District 7's freeways.

The experiment part will cover what prediction models were used as well as their results on different variables of the experiment as well as insights and comments. This paper will conclude with the conclusion and future development ideas which will explain how the experiment was conducted and what could be improved.

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 which requires insight and solutions.

1. Motivation

As technology evolves so do the means of transportation. In a span of 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 19^{th} century humanity main transportation method where horses for thousands of years, the next century locomotives and cars totally replaced horses. Over time cars were starting to be available to the public and as more time passes, they are becoming increasingly affordable. This means that more people will use cars and the more cars there are the bigger the traffic congestion will be.

With time cities become increasingly bigger in population as well as area occupied and more difficult to navigate. In the year 1300 London had a population of 80 000 and in 2021 it has a population of 8 million people. This increased compression of cities combined with people buying and using more and more cars cause heavy traffic congestions with huge waiting times.

A study made in 2019 [1] showed that for a person driving in New York City they have to spend on average 92 hours of traffic. That is almost four days of wasted time. The average person in the USA spends around 54 hours a year in traffic. This situation will only get worse. It is estimated that, if the infrastructure capacity will not improve, the USA congestion cost will grow by 19% and wasted fuel will increase be 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 for 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 own economy.
- Lower traffic means less 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 or road rage and even vehicle accidents. These events can be lowered with a lower traffic.
- If multiple cities have lower traffic, it can even boost an entire country's economy.

Traffic congestion is a global problem which affects densely packed urban areas the most. According to a web site [2] the top four most congested cities are:

- Moscow region (oblast), Russia with a congestion level of 54%
- Mumbai, India with a congestion level of 53%
- Bogota, Colombia with a congestion level of 53%
- Manila, Philippines with a congestion level of 53%

Three out of the top four are poorer, resulting in a worse infrastructure. Bucharest is ranked 18th and Los Angeles is ranked 85th.

Multiple solutions exist to solve this problem, the first one is to share a vehicle among commuters and reducing the number of vehicles in traffic. For this solution many people choose a personal car as it is more comfortable than the public bus as well as waiting times for public busses. For a car a person does not have to wait to start to go towards its destination. Only the metro system is better than cars but it is the hardest method to implement as it requires underground work. Smaller vehicles like bikes and motorcycles are rarely used due to fear of injury if an accident happens. A car protects its passenger better. One of the few solutions is to know in time what waiting time is on each individual lane such that a better route planning can be made.

II. THEORY

1. Traffic Flow

Traffic Flow represents the study of interaction between travelers of different types, including vehicles and pedestrians with infrastructure having the goal to create an optimal transport network with efficient and optimal traffic movement such that traffic congestion is reduced to a minimum.

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

The traffic system is composed of roads/pavement on which the participants in traffic travel at a certain speed to reach their individual goal. The size can vary and is objective, it can range from a neighborhood to a countries entire traveling system. There are traffic systems which are unique to a single type vehicle, for example the railway system in which just the train travels. Traffic flow 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 transportation). 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 efficiency and can take thousands of passengers. Another example is a car which can take up to a maximum of 5 passengers legally. Although a car can travel to any point but a metro system has predetermined stations.

The main variables that are considered in traffic flow theory are density (k, number of vehicles per unit of space), speed (v) and flow (q, number of vehicles per unit of time). [3]

Speed is defined as the distance traveled per unit of time. Due to the high number of vehicles in a traffic system it is impractical to measure the speed of every vehicle, and the average speed is measured. There are 2 main definitions of speed.

The first one is time mean speed, which is the average speed of a vehicle over a period of time, is computed by measuring the distance traveled by a car 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.

$$v_t = \frac{1}{m}\sum_{i=1}^{m} v_i \tag{2.1}$$

Where m = Number of vehicles passing through a fixed point; v_i= speed of vehicle i

The second method is called space mean speed. This method is measured over the entire segment. Consecutive recordings of passing vehicles over the entire segment are considered to track individual vehicle speed, after which the speed is calculated. It is considered more accurate than the first method

$$v_s = \left(\frac{1}{m}\sum_{i=1}^{m}\frac{1}{v_i}\right)^{-1} \tag{2.2}$$

Where m = Number of vehicles passing through the roadway segment; v_i= speed of vehicle i

Density represents the total number of vehicles per road segment. There are two types of densities: critical density and jam density. Critical density is considered to be the maximum density under free flow, while jam density is considered the maximum density under congestion. Density is measured as:

$$K(L, t_1) = \frac{m}{L} = \frac{1}{\bar{s}(t_1)} \tag{2.3}$$

Where K = density, L = length of a roadway, t_1 = time, m = number of vehicles

Flow is represented as the total number of vehicles passing through a point given a time period. Headway is the inverse flow, which is represented by the time elapsed between a vehicle passing a point and the next vehicle. Flow is measured as:

$$q(T, x_1) = \frac{m}{T} = \frac{1}{\bar{h}(x_1)} \tag{2.4}$$

Where q = flow, T= time interval, x_1 = point in space, m = number of vehicles

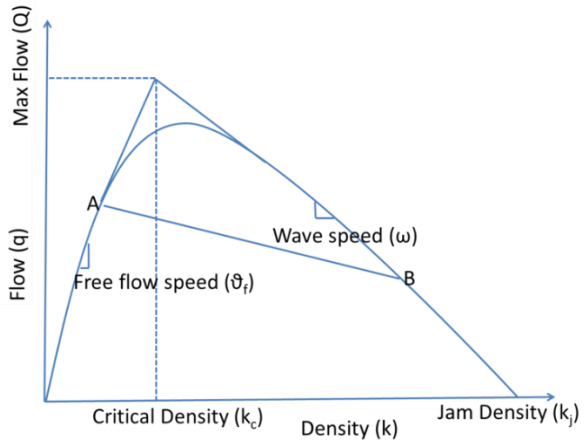


Figure 1 Flow Density Relationship

The goal of traffic flow analysis is to develop a model that will allow vehicles to reach their destinations within the shortest possible time. This is achieved in a four stages process: [4]

- Generation – Calculate how many trips to be generated by the needs of the passengers.
- Distribution – Generate the path between starting point and stopping point based on what has been generated at step 1
- Modal Split/Mode Choice – Decide the distribution of different type of vehicles for the number of passengers.
- Route Assignment – Assign each vehicle its route such as the entire system has a minimum travelling time.

2. Machine Learning

Machine Learning is a subset of Artificial Intelligence which creates and develop software which is able to learn and improve from data over a period of time exposed to data (experience) without being explicitly programmed [5]. This method is used for tasks which are mathematically impossible to program iteratively. For example, it is impossible to explicitly program a function which can classify cats and dogs. The process of learning starts with observing data, experience (time observing data) and instruction in order to look for patterns in data for better future decisions. The goal is to let the software learn without human intervention.

There are three categories which divide machine learning algorithms based on the task and based on the nature of the data: [6]

- Supervised learning: The data is presented with a label and the task is to predict the correct label. An example is classifying 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 and 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 with a goal to perform. 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.

2.1. Multiple Linear Regression

One of the most popular machine learning algorithm is Linear Regression. Regression is a method of learning a target value based on independent predictors. This method is mostly used in forecasting or finding cause and effect relationships between variables. There is more than one method of regression but mostly differ on the number of independent variables and the type of relationships between independent and dependent variables.

Linear Regression is a type of regression where there is a linear relationship between independent and dependent variables [7]. Multiple Linear Regression, or Multiple Regression is when we observe the relationship between more than one independent variable and one dependent

variable. For each independent variable is assigned a weight, then they are summed together with a bias to produce a result:

$$Y = b + \sum_{i=1}^{n} w_i X_i \tag{2.5}$$

Where Y is the dependent variable, X_i is the independent variables, n is the total number of independent variables and b is the bias.

2.2. Artificial Neural Networks

One of the algorithms used in Supervised learning are called Artificial Neural Networks (ANN's). They are simply called neural networks (NN's). Neural Networks are a method based on the anatomy of the nervous system and the brain. These networks are based on the electrical activity of the nervous system.

The base element of the neural networks is a neuron, which is based on the fundamental unit of the brain with the same name, which receives an input, multiplies it by a weight, adds a bias and the returns the result for further processing by a software or another neuron. The function for a neuron looks like this:

$$y = x * w + b \tag{2.6}$$

Where x is the input, w is the weight, b is the bias and y is the output. Normally the neurons are placed in a layer, with the output from the previous layer being the input of the previous layer, thus simulating the synaptic connections of the brain.

After computation a scalar function is used to aggregate a layer into a single input value. Once it is calculated a transfer function, or activation function is used to calculate the output of the layer. 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. while Sigmoid is used for non-linear classification type tasks such as dogs vs. cats.

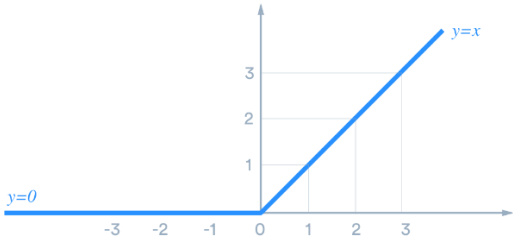


Figure 2 ReLU Activation Function

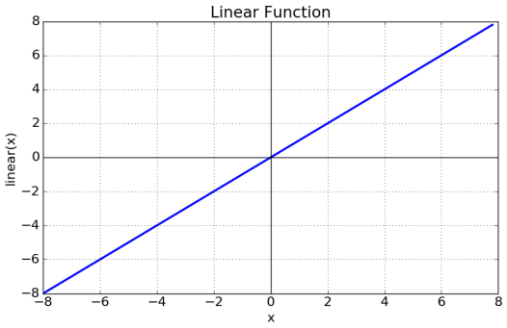


Figure 3 Linear Activation Function

$$ReLU : f(x) = \max(0, x) \qquad (2.7)$$

$$Linear : f(x) = x \qquad (2.8)$$

Generally, the artificial neural network is divided in tree named parts known as the input layer, the hidden layer and the output layer. [8]

- The input layer is responsible for receiving the data,
- The hidden layer is composed of neurons that are responsible for extracting patterns and relevant information for the task. This layer has the most of the computational work of the network.
- The output layer is responsible for giving the result of the neural networks.

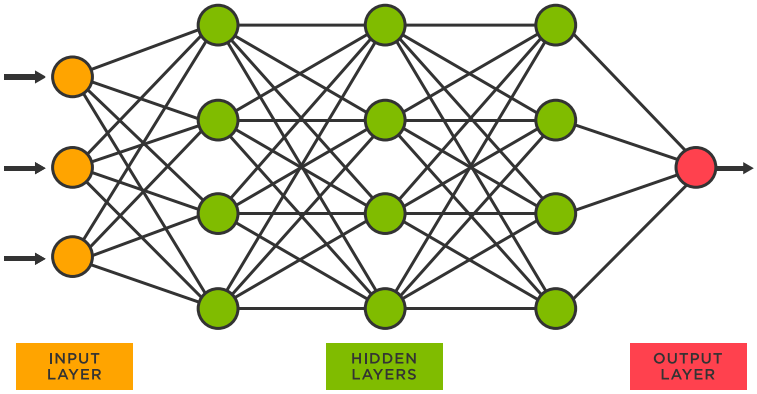


Figure 4 Neural Network Architecture

There are multiple types of neural networks: [8]

- The single layer feedforward architecture has an input layer and an output layer.

- The multiple layer feedforward architecture is composed of an input layer, an output layer and one or more hidden layer. This type of neural network is also called deep neural networks.
- The recurrent or feedback architecture is using the outputs of the neurons as the feedback input of the same neurons. This feature is good for dynamic information processing, such as timeseries.

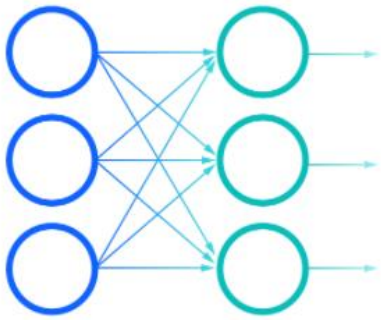


Figure 5 Feed Forward Neural Network

2.2.1. 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 of data samples and retain the patterns. After the training the network is able to create a function which maps the inputs and outputs, such that in future when exposed to unknown data it can generalize solutions which are close if not the desired output for the task given.

The training step is applying the algorithm for tunning the synaptic weights and biases of the neurons with the purpose to generalize a future solution better than the previous training step. In practice the data is split in 2 datasets: the training and test data. The test data is used to measure the performance of the network after the training is finished and 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 [8]. For each complete exposure of the training dataset for adjusting the weights are 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) increases 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 and depending on an increase or decrease will adjust the weights accordingly.

The learning rate defines the dimension of the correction in the learning process to adjust the weights. A high learning rate makes the training time 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 a higher accuracy but will lengthen the training time. In practice an adaptive learning rate is used in which 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.

2.2.2. Cost Functions

There are multiple ways to compute the loss function depending if the prediction is a classification or a regression type. The notation is as follows: n = number of values, y = predicted values and x = 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 n predicted values is computed thus MAE is calculated as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - y_i| \tag{2.9}$$

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

$$MAPE = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{x_i - y_i}{y_i}\right| \tag{2.10}$$

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

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.11}$$

2.2.3. Backward Propagation

Given a cost function and a neural network architecture (also called a model) there is a method which calculates the gradient of the cost function with respect to the model's weights. This function is called the backpropagation [9], a short term for backward propagation. The algorithm starts backward, in which the gradient of the final layer is calculated first and finishing with the first layer last. As the gradients are calculated the previous computation of gradients are also reused. This gives this method computational efficiency, as gradients computations are reused rather than calculating each layer individually as a forward propagation.

During the training process of a neural network with gradient descent the gradient of the error function is calculated $E(X, \theta)$ with respect to the weights and biases. Each iteration of gradient descent updates the weights and biases as:

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial E(X, \theta^t)}{\partial \theta} \tag{2.12}$$

Where $E(X, \theta)$ is the cost function, θ^t are the weights and biases at an iteration t of the training process and α is the learning rate.

2.3. Deep Learning

2.3.1. Regularization

A common problem in machine learning is to make a model perform well on the training data and also on new, unseen data. This term is also called generalization. There are situations in

which the error function in the training step is lower than in the validation/test step. This behavior is called underfitting, a situation in which for the given data the model is not complex enough to capture the relation between the input and target value and the model has too much bias. The opposite, in which the validation/test error function is lower than the training phase error is called overfitting. This makes the model learn “too much” [10] 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 is able to variate much more than it is needed. Overfitting is a recurrent problem in machine learning, because the purpose of supervised machine learning models is to be able to generalize well on unseen data, in a more realistic situation.

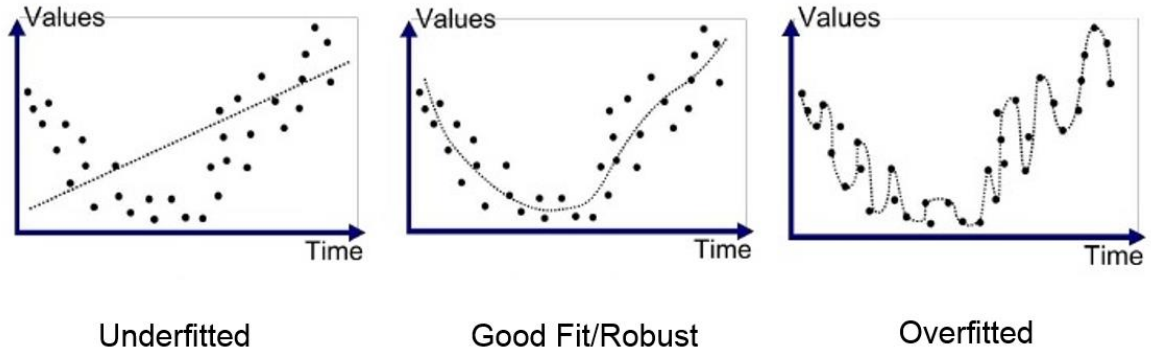


Figure 6 Underfitting, Robust and Overfitted Model

The perfect situation is between overfitting and underfitting, where although the model has some training error it is able to label new data as good as it labels its training data. There are a few solutions to adjust how well a model learns, for underfitting is to increase the model complexity. An overfit model is harder to resolve, but it can be done with regularization. Regularization is any component of the model, training process or prediction procedure which is included to account for limitations 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 L1 (Lasso Regression) and L2 (Ridge regression) [11] normalization of the weights that are being optimized. This method prevents the weights becoming to variate and avoids overfitting. For example, the MAE loss function and add the L1 and L2 regularization.

$$L1 = \lambda \sum_{j=1}^{p} \beta_j^2 \qquad (2.13) \qquad L2 = \lambda \sum_{j=1}^{p} |\beta_j| \qquad (2.14)$$

$$MAE\ L1 = \frac{1}{n}\left(\sum_{i=1}^{n} |x_i - \sum_{j=1}^{p} y_{ij}\beta_j| + \lambda \sum_{j=1}^{p} \beta_j^2\right) \qquad (2.15)$$

$$MAE\ L2 = \frac{1}{n}\left(\sum_{i=1}^{n} \left|x_i - \sum_{j=1}^{p} y_{ij}\beta_j\right| + \lambda \sum_{j=1}^{p} |\beta_j|\right) \qquad (2.16)$$

Where n = number of values p = number of neurons, x = correct value, y = predicted value with respect to weight β. The L1 regularization adds a squared coefficient as a penalty to the cost function while the L2 regularization adds an absolute value coefficient as penalty.

In modifying the sampling method, there are two options: Data augmentation and Cross-Validation [12]. 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, for sound data it can be noise pitching or time stretching.

In 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 called validation data. This portion has about 20% of the remaining training data. At the end of each epoch the validation data is used as a 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 on unseen data.

The model architecture can be altered by adding a regularization layer called Dropout [13]. This type of regularization is frequently used and has the best results in deep learning. At every iteration Dropout randomly selects some nodes with probability p and removes them along with their input and output connection as well. This method reduces the variance which the output can produce since there are less nodes.

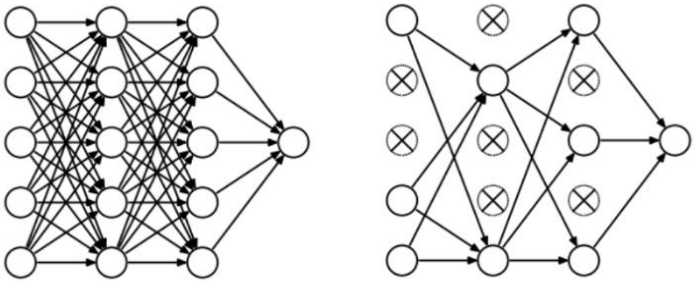


Figure 7 Neural Network model without and with Dropout

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 [14] scaling and is defined as:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{2.17}$$

Where X is a feature, X_{min} is the minimum value of a feature, X_{max} is the maximum value of a feature.

2.3.2. Optimization

The biggest problem with deep neural networks is the time required for computing the gradients of the cost function with respect to the model's weight and bias. Due to the fact that 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 update the weights of the model.

The first method for calculating the error function's gradients is Gradient Descent. Gradient Descent is the first and most basic gradient algorithm which you can apply to train a deep neural model [15]. It is also called "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 deterministic computed. Gradient descent updates the parameter θ with a small step towards the minima. The Gradient Descent algorithm looks like this:

$$\theta^{t+1} = \theta^{t} + \alpha \nabla_{\theta} \sum_{i=1}^{n} L(f(x_i, \theta), y_i) \tag{2.18}$$

Where α = learning rate, θ = model's weights and biases at time t, $f(x_i, \theta)$ = predicted values for value x_i input values and model parameters θ, y_i = true values, L = loss function for predicted values and true values. Gradient Descent reaches convergence with a fixed learning rate as the gradients get smaller.

There are many algorithms which 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 it 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 and if the slope becomes steeper the object will accelerate even more. The same logic can be applied to the Gradient Descent algorithm:

$$v = \alpha v + n \nabla_{\theta} \sum_{i=1}^{n} L(f(x_i, \theta), y_i) \; ; \theta = \theta + v \tag{2.19}$$

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

The learning rate is one of the hyperparameters that is the most difficult to set because it can have a high impact on the performance of the model. While momentum has mitigated some of the problem the learning rate has, a new hyperparameter that is just as difficult to set as the learning rate still has to be added.

For this purpose, adaptive learning rate algorithms has been made [16]. The first algorithm introduced is AdaGrad [17]. Though several variants of Gradient Descent algorithms have evolved, the challenge lies in fine-tuning the hyper-parameters to redefine the algorithm to train the network as per the requirements of the dataset. AdaGrad adaptively scales the learning rate for of all model's parameters by scaling them inversely proportional to the sum of squared partial derivates from the training epochs. This will make the parameters with greater gradient have a higher learning rate decrease while the parameters with 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 of learning rate's effectivity.

RMSprop [18] addresses AdaGrad's problem by shifting the gradient accumulation into an exponential weighted moving average. Adam [19] 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 efficient the weights for deep learning training.

2.3.3. Convolutional Neural Networks

For more complex and bigger sized data a simple linear function with a weight and bias is not enough. For this the traditional function for a neuron is replaced with a convolution. A Convolution is the simple application of a filter to an input that results in an activation. Since the convolution operation takes multiple dimensions, the CNN is the best choice for image classification, but it cand be used in other areas such as text classification.

The Convolutional neural network (CNN for short) is a type of neural network which contains at least one convolutional layer. In a convolution, the input is a tensor with shape: number of inputs x input height x input width x input channels and the input is transformed into a feature map, also known as activation map with the same number of dimensions. The convolution operation is similar to the response of a neuron in the visual cortex to a certain stimulus. A convolutional neuron processes a part of the image, like a receptive field. A convolutional network is able to capture spatial and temporal dependencies in its input through application of filters. Instead of operating on a normal linear function a convolution uses multidimensional filters which "hover" over the image in order to extract information. In the case of a 2d convolution it would be a matrix of a small area, usually 3x3 or 5x5.

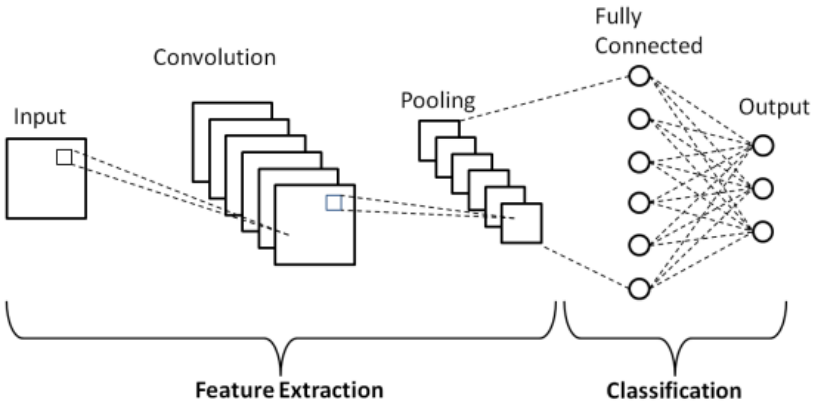


Figure 8 Convolutional Neural Network

In real situations images have high resolution, which can make the CNN model heavier with weights. For example, an image with 100 x 100 size will have 10 000 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 which will be used in a CNN model.

2.3.4. Geometric Deep Learning

The best way to represent a temporal dependency is with Graphs. A graph is an abstract data type which can be used to represent non-linear relationships between objects and locations. A graph is made up from nodes (sometimes called vertices) which 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 $G = (V, E, \phi)$ [20] where V is the set of nodes, E is the set of edges and $\phi : E \to \{ \{x, y\} \mid x, y \in V \}$ is a mapping function which attributes an edge for an unordered pair of edges. This representation of ϕ is different for a directed graph where $\phi : E \to \{ (x, y) \mid x, y \in V^2 \}$ 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 by permitting loops by adding the $x \neq y$ in the mapping function.

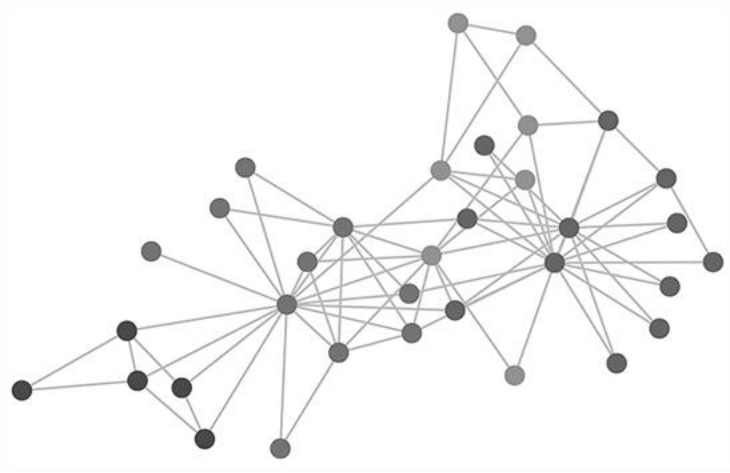


Figure 9 Graph Example

Graphs have a multitude of real-life applications, they can be used to model processes in biological, physical and information systems [21]. Graphs have been receiving more attention in machine learning due to the great expressive power of graphs. They can be used to describe systems across various domains such as social networks, physical systems, protein interactions systems or even knowledge graphs.

There are multiple types of graph neural networks classifications. There are nodes classification, in which the nodes contain information to be classified or edge classification in which 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 change nodes and edges 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 GNN's) are a deep learning method which operate on graphs. To use graph in convolutions the Laplacian matrix and Adjacency matrix. The adjacency matrix A is an n x n matrix of edges where:

$$A_{ij} = 1 \; if \; e_{ij} \in E, 0 \; otherwise \tag{2.20}$$

Where e is an edge belonging to the graph. In the case of undirected graphs an edge connects both nodes, so the adjacency matrix is symmetrical. In the case of 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:

$$A_{ij} = w_{ij}\ if\ e_{ij} \in E, 0\ otherwise \tag{2.21}$$

Where w is the weight of edge e

The Laplacian matrix is defined as L = D – A where D is the degree matrix. The degree matrix is diagonal and its elements are the degree of each node. The degree of a node is given by the number of connecting edges. $D_{ii} = \sum_j A_{ii}$. The Laplacian matrix usually has three forms:

- Combinatorial Laplacian:

$$L = D - A \tag{2.22}$$

- Symmetric normalized Laplacian:

$$L^{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tag{2.23}$$

- Random Walk Normalized Laplacian:

$$L^{rw} = D^{-1} L = I - D^{-1} A \tag{2.24}$$

To further understand convolution on graph a notion for eigenvectors and eigenvalues as well as Fourier transformation must be defined

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

$$T(v) = \lambda v \tag{2.25}$$

Where T is a linear transformation, v is the eigenvector of T and λ is the eigenvalue

A Fourier transformation is a decomposition of a function which depends on either time or space into multiple functions depending on spatial or temporal frequency. The Fourier transform may be formally defined as an improper Riemann integral, making it an integral transform. The Fourier transformation of a function f is formally denoted $\hat{f}$. There are more ways of defining a Fourier transformation of an integrable function [22], for any real number ξ one of them is:

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \xi}\, dx \tag{2.26}$$

A key role in graph convolutional networks relied on spectral method is the eigen-decomposition of the Laplacian Matrix. The Laplacian matrix can also be written as:

$$L = U \cap U^{-1} \tag{2.27}$$

Where $U = [u_{i<n}] \in R^{NxN}$ is a matrix made of eigenvectors sorted by eigenvalues. It is also known that U is orthogonal matrix and has $U^{-1} = U^T$

The Convolution Theorem states that under some conditions the Fourier transformation of a convolution of two functions is the pointwise product of their Fourier transformation [23]. It can be done, by analogy to the graph and implement the Fourier transforms on graphs. The graph convolution will be defined on input signal x and filter $g \in R^N$ as [24]:

$$x * g = U\left((U^T x) \odot (U^T g)\right) \tag{2.29}$$

Where $\odot$ is the element wise product

Graph Convolutional Network [25] (also abbreviated GCN) is based and simplifies ChebNets architecture and is uses graph convolutions. The output of the convolution is:

$$y = x * g = \sigma(\theta(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})x) \tag{2.30}$$

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

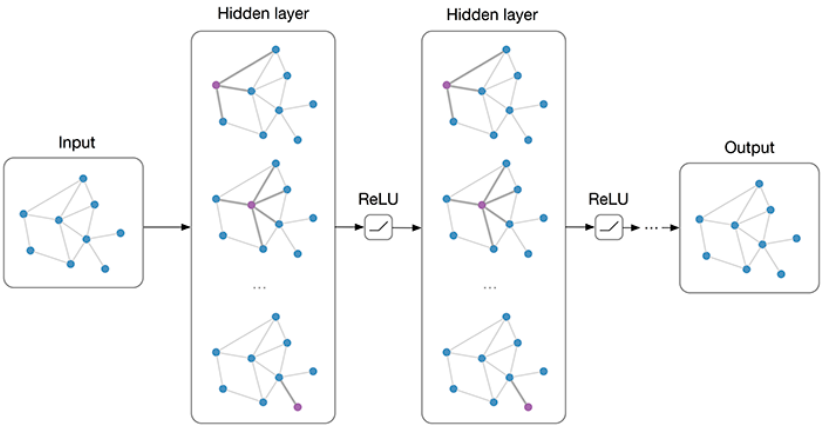


Figure 10 Graph Convolutional Network

2.3.5. Recurrent Neural Network

One of the presented types of neural networks architectures are recurrent neural networks. A recurrent neural network (abbreviated RNN) is a special type of neural network which contains neurons that receive prior inputs and influence 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)

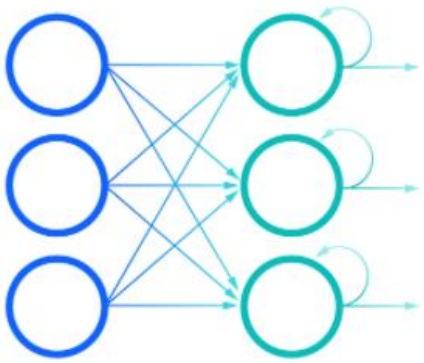


Figure 11 Recurrent Neural Network

There are multiple types of recurrent neural networks, one of the most popular is the Long short-term memory (abbreviated LSTM). LSTM [26] have been introduced in 1997 as a problem to classic RNN's problem. Theoretically RNN's can keep an arbitrary long-term dependency in its input but it has a flaw: when training classical RNN's with backpropagation the long-term gradients tend to zero or infinity. This is also called 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 buy can still be affected by exploding gradient problem.

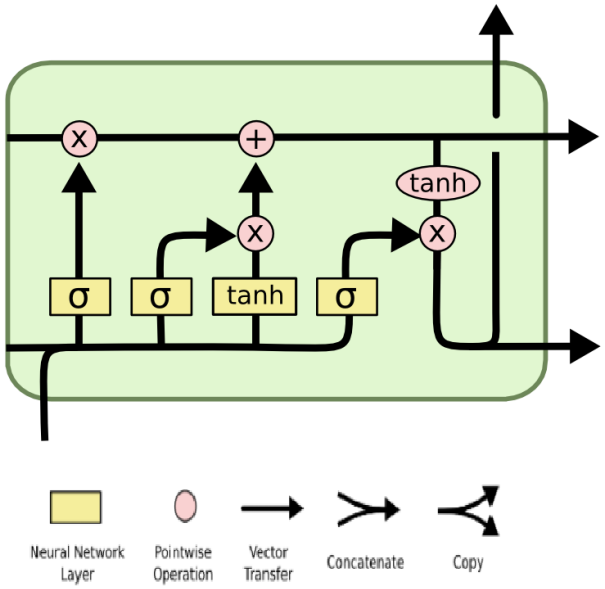


Figure 12 Long short-term memory cell

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

The first thing updated in LSTM is the "forget gate" layer. It is the leftmost part of the cell represented in Figure 12. The previous output h_{t-1} will be taken and x_t 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 f_t can be written as:

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \tag{2.31}$$

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 h_t and applies a sigmoid layer to it. Next another layer is created $\tilde{C}_t$ with possible candidate values as a tanh layer of the input. The input gate and candidate values can be expressed as:

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{2.32}$$

$$\tilde{C}_t = \tanh\left(W_c * [h_{t-1}, x_t] + b_c\right) \tag{2.33}$$

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

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.34}$$

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 tanh function is applied and is multiplied with the output. The final part looks like this:

$$h_t = \sigma(W_o[h_{t-1}, x_t] + b_o) * \tanh\left(C_t\right) \tag{2.35}$$

III. RELATED WORK

The act of transportation has been in attention 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. In this chapter there will be illustrated several scientific researches and advances in traffic optimization domain.

This subject being approached as early as 1959 [27]. More recent work includes a paper in 2007 [28] which used a version of random forest 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 [29] 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 week day and a weekend day, but very little difference between normal days and holydays.

A more recent study [30] used Autoencoder and stacked autoencoders (abbreviated SAE's). An autoencoder is a neural network which tries to reproduce its input and a stacked auto encoder is made by stacking several autoencoders. This study used the same dataset as this paper but for the year 2013. It can discover latent traffic flow, and does not consider only a shallow structure of traffic data.

Another study [31] 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 pe represented as a spatial potin of view. This can be achieved with Convolutions and Graph Neural Networks. One study [32] used both temporal and spatial dependencies of this task to create a Spatial Temporal Convolution block based on recurrent graph neural network and showed significant improvement over traditional methods.

More studies on graph neural networks include Attention Based Spatial-Temporal Graph Convolutional Networks (ASTGCN) and its Multi-Component variant [33], Graph Multi-Attention Network (GMAN) [34], a Multivariate Time Series Graph Neural Network (MTGNN) or even hybrid recurrent and graph neural network called Graph convolutional LSTM (GCLSTM) [35].

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.

IV. METHODOLOGY & EXPERIMENT

This chapter will include the methodology used for this experiment, technologies and libraries used and how they were used as well as what data has been used, data processing and experiments results and findings.

1. Methodology

1.1. Technical Implementation

For the entire experiment a single programming language has been used, 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. For this experiment 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 a USA governmental website PeMS [36]. More details on the dataset will be covered on the Dataset chapter. To access this dataset an account is required and then 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 [37] has been used to read the information required in the CSV. For any array manipulation like sorting, grouping or filtering the NumPy [38] library has been used. NumPy uses some of the best array manipulation in python and is the best choice for this type of experiment.

For most machine learning projects Keras [39] is the choice to go. For this project and all its deep learning models the library PyTorch [40] 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 [41] and a tool for Graph Recurrent Neural Networks with PyTorch Geometric Temporal [42]. Although PyTorch is harder to use, the training step needs to be implemented, it has more to offer.

Python’s machine learning libraries often support GPU integration for faster training. PyTorch offers GPU integration as well. For Python only Nvidia graphics card are supported for

GPU's. To enable GPU, CUDA [43] and cuDNN [44] has 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 10^{th} 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 [45] is used. It contains a grid for hyper parameters to be chosen at random, a number of samples can be set and a patience for other builds to produce better results than the previous ones. Ray Tune also provides checkpointing by saving the models weights state as well as information such as loss such that everything is saved for later use.

Data visualization is also important for bigger datasets to gain insight. For a better understanding of the dataset a data visualization library is used called Plotly [46]. Plotly is used to plot diverse information about the dataset to help form a better understanding of what information is processed as well as gaining insight through plots. Although Plotly is used in Python in this experiment it is available in R or Matlab as well.

Libraries tend to have dependencies and conflicts and such a library manager is needed to create a good working environment. For this Anaconda [47] 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 used in the development of this experiment.

1.2. 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. For this experiment was chosen sector seven. There are multiple type choices and will impact what data is offered and for this experiment the "Station 5-Minute" is chosen. This means that for every 5 minutes throughout the day are record of data of 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 throughout 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 present if the year selected is still ongoing. At the time this experiment was started the latest completed month was selected, July 2021. The heaviest congestion happens during the week days. In order to find better insight of 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) [48], 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 .gz extension file. After extracting a .txt 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 repeat meaning that for a point in time the station will be presented in the same order as for 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 there are 720 individual timestamps which are recorded, multiplied with 4904 unique stations it can be seen that there are over 3 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 in a timely manner.

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
- 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 were observed

- Flow: Sum of vehicle passing through the station over a five-minute period across all lanes.
- Occupancy: Percentage of occupancy across all lanes in a five-minute period as a decimal number.
- Speed: Average flow-weighted speed in a five-minute period across all lanes.

The next rows represent the columns samples, observed, occupancy and speed but for each individual 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 .txt file with columns separated by row. This file contains metadata about the station sensors. In this experiment there are three variables used from metadata: the station 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 with their timestamp as well as the station unique identifier but for a better understanding the other variables will be used as well to get better insight through data visualization.

2. Experiment

The experiment will consist of using three models: a linear regression model for each individual sensor node and two graph neural networks, one based on Spatial Temporal Convolutions and the other based on Graph Convolutional Neural Networks together with Graph Embedded LSTM.

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 three different datasets based on the information from this dataset:

- Experimental dataset, containing eight hand-picked sensors in an intersection.
- Small dataset, containing 120 random picked sensors.
- Medium dataset, containing 480 random picked sensors.

2.1. 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 are saved for later us, in that way for all three datasets the station will always be the same.

Two out of the tree models proposed for this experiment use graphs as inputs and for this reason the graphs for training needs to be constructed. The Station Metadata does not contain any information about how each sensor are connected to one another or if they are on the same road side. Because of this the graph needs to be constructed manually using geolocation data from metadata. The latitude and longitude of each node will be measured against every other station present in the dataset and compute its geodesic distance. The geodesic distance between two points is calculated with the below formula:

$$GeoD(P1, P2) = 2\ arcsin\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \cos\phi_2 \sin^2(\frac{\Delta\lambda}{2})} \qquad (3.1)$$

Where P1 and P2 are the two points having ϕ_1 latitude and λ_1 longitude for P1 and ϕ_2 latitude and λ_2 longitude for P2. $\Delta\lambda$ represents the absolute difference in longitude for the two points and $\Delta\phi$ represents the absolute difference in latitude between the two points.

The geodesic 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 connectiveness of the nodes must be computed. There are several ways to consider which nodes will be connected, the following formula will be used:

$$w_{ij} = \exp\left(-\frac{d_{ij}^2}{\sigma^2}\right) if\ i \neq j\ and\ \exp\left(-\frac{d_{ij}^2}{\sigma^2}\right) \geq \epsilon \tag{3.2}$$

Where w_{ij} represents the weighted distance between station i and j. and d_{ij} represents the geodesic distance between station i and j. There are two new hyper parameters introduced in equation 3.2, ϵ and σ. These two parameters are thresholds to control the sparsity and distribution of the graph's edges. The values for the two parameters are:

$$\sigma \in \{1, 3, 5, 10\} \tag{3.3}$$

$$\epsilon \in \{0.1, 0.3, 0.5, 0.7\} \tag{3.4}$$

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

The graph information is saved in two NumPy arrays, the first one represented in a two-row array representing which station are connected. The station will not be saved as their unique identifier but as their index of order instead. Two stations are considered connected if their respective index of order is on the same column. The second array is a single row array containing the weight for each edge described in the previous array. This arrangement is necessary for PyTorch Geometric training.

The last step in data preparation is grouping data together for training. A PyTorch Geometric model uses input data and graph information called edge index array and edge weight array. The Spatial Temporal model uses a batch size of 8 while the custom does not use batch sizes and takes one temporal step at a time. The data is arranged to allow training and groups together the information for each type of dataset corresponding to what nodes are present in that dataset. The next part of this experiment is to train the three models against the three datasets.

2.2. 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 June 2021 and contains 22 days of data. Weekend days were removed since there most work is done on week days and the results would be altered.

In figure 13 all sensors and there can 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 13 All Map Sensors

In Figure 14 there are the sensors selected for the experimental dataset. The Experimental dataset is handpicked and was selected such that the station selected are in close proximity to one another. There are a total of eight sensors 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 14 Experimental Dataset

Figure 15 Marina and San Diego Freeways Intersection

The Medium dataset is represented in figure 16 and the small dataset is represented in figure 17. For these two datasets the nodes are chosen randomly and they do not have to be in close proximity to one another. It can be noted that the small dataset is sparser than the medium dataset. As stated in the data preparation the two hyper parameters ϵ and σ will be a major factor in the results of the experiment as the medium dataset will be more well connected.

Figure 16 Medium Dataset

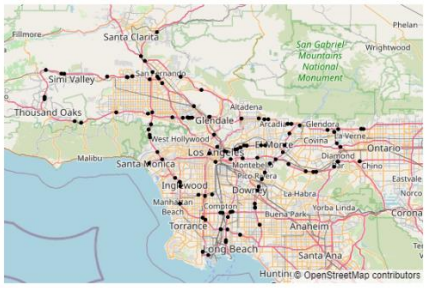


Figure 17 Small Dataset

The obtained adjacency matrices are obtained using equation 3.2 and they have a size of 8x8, 120x120 and 480x480. There is a total of 16 different adjacency matrices. Four of the represented matrices are represented in figures 18, 19, 20 and 21. It can be observed as sigma is being increased the connectivity of the graph is increased. On the other hand, as epsilon increases it restricts the 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 more middle ranged colors. Not all the nodes are connected to one another. This behavior is normal since not all the roads are interconnected at all points between them.

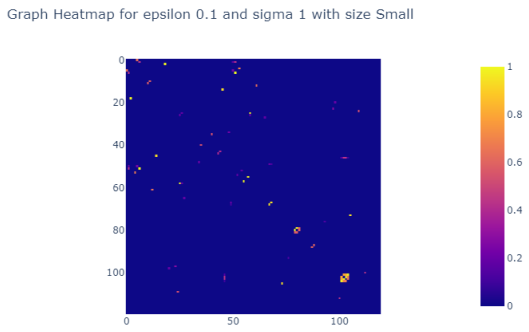


Figure 18 Graph Heatmap for epsilon 0.1 and sigma 1

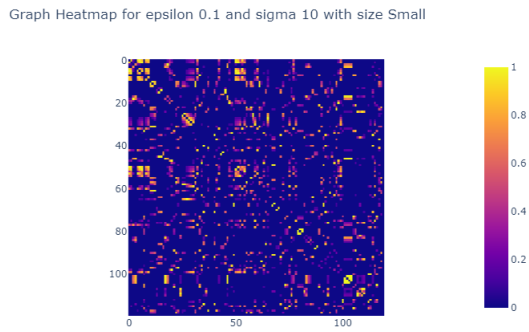


Figure 19 Graph Heatmap for epsilon 0.1 and sigma 10

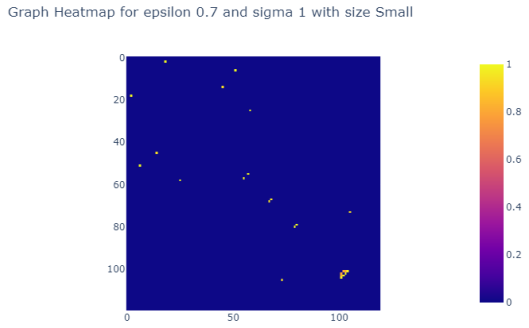


Figure 20 Graph Heatmap for epsilon 0 Graph Heatmap for epsilon 0.7 and sigma 1

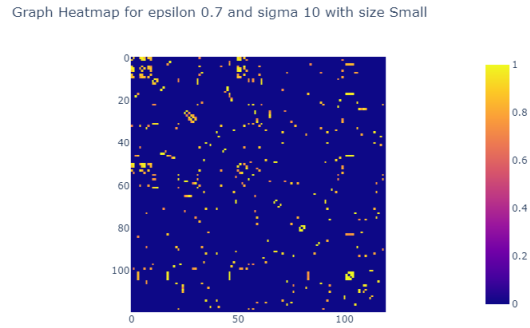
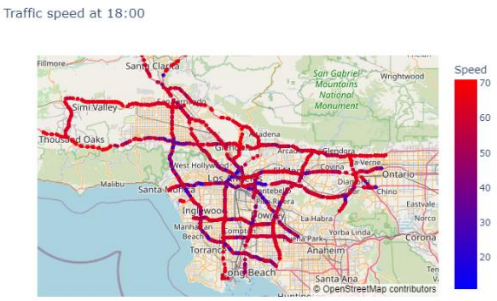


Figure 21 Graph Heatmap for epsilon 0 Graph Heatmap for epsilon 0.7 and sigma 10

There are specific time intervals in which traffic is heavier called rush hour. In figure 22 it is presented a speed map of the traffic at 18:00 while in figure 23 a speed map of the traffic at 22:00. Six in the afternoon is considered a rush hour since most work schedules end at this hour and every commuter goes home.

With the increase in traffic participants so does the waiting time. In figure 22 the center part of the district is colored more in blue signifying heavier traffic with some blue parts along the outer areas. In figure 23 most of the roads are covered in red, as the roads are clearer during the night.



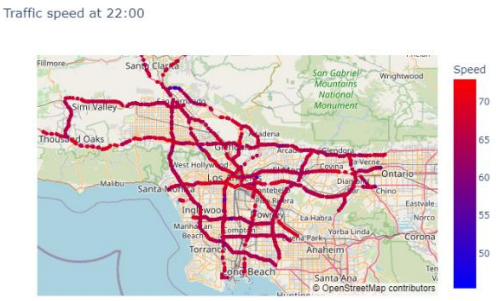


Figure 22 Traffic speed at 9:00

Figure 23 Traffic speed at 22:00

In figure 24 the daily average Speed is presented. The maximum speed limit for a freeway in California is 65 miles per hour (mph). In all days the median is at 65 mph while the upper standard deviation is at around 70 and the lower at 60. There are exceptions in both upper and lower deviation which is normal, there are traffic participants which 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.

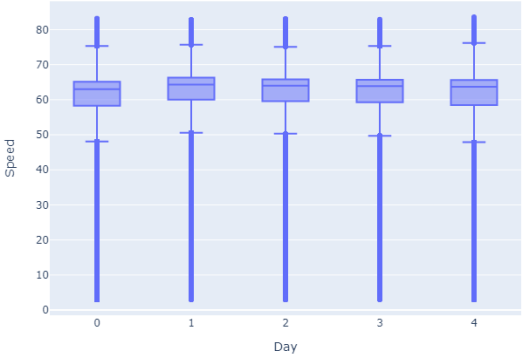


Figure 24 Daily Average Speed

In figure 24 the daily average Speed is presented. The maximum speed limit for a freeway in California is 65 miles per hour (mph). In all days the median is at 65 mph while the upper standard deviation is at around 70 and the lower at 60. There are exceptions in both upper and lower deviation which is normal, there are traffic participants which 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.

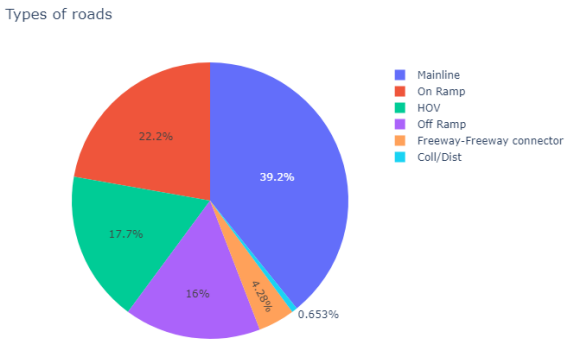


Figure 25 Road Types

There are multiple types of roads which could be fitted with a sensor. They aren't placed only on the inner freeway of the city but on multiple places. The types distribution of road types is presented in figure 25. The majority of sensors are placed on the mainline of the freeway (39%) while the secondary is on ramp roads (22%) and the third are 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).

2.3. Models

There will be 3 models tested. The first model is a Linear Regression model used as baseline. The model takes each individual sensor data for the entire month of June 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 second model is a temporal geometric neural network model based on an existing model [32] presented as managing to encapsulate spatial dependencies as well as temporal dependencies.

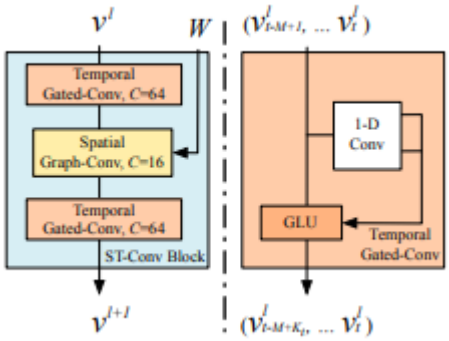


Figure 26 STConv Structure

It consists of three blocks Spatial Temporal Convolutions separated with an ReLU activation. Each block is made of two Temporal Gated Convolution 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 26.

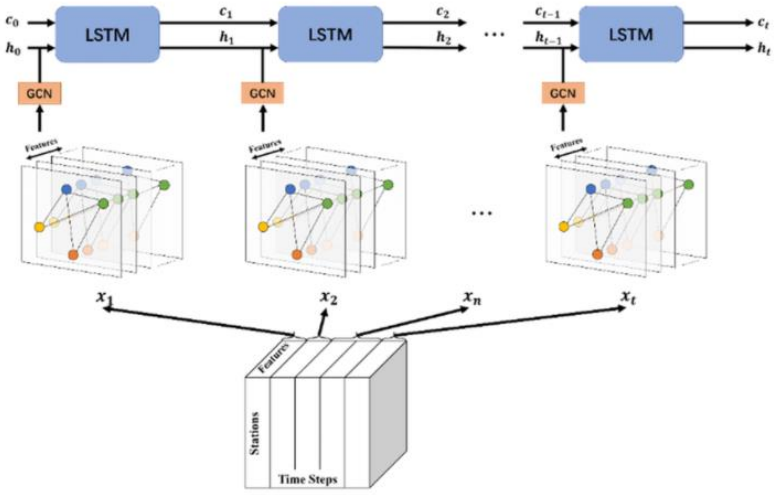


Figure 27 GC-LSTM

The third model is composed of two elements. The first element is a LSTM embedded Graph convolution based on the GC-LSTM [35]. The second part is a simple Graph Convolution. The GC-LSTM block is represented in figure 27. At each time step a graph convolution is passed to a LSTM cell. The LSTM cell keeps track of previous information as well.

2.4. Results

The three models were tested against the three databases and they were evaluated under four loss functions: Mean Average Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and Mean Average Percentage Error (MAPE). Their equations are a part of the theoretical chapter under equations 2.9, 2.10 and 2.11. The results are as follows in figure 28:

Type	Size	RMSE	MAPE	MAE	MSE
Linear Regression	All	170579.40	0.11	5016.04	43553409179
STCONV	Experimental	15.86	0.37	10.01	258.14
STCONV	Small	10.03	0.15	5.84	101.17
STCONV	Medium	11.70	0.21	6.90	137.46
Custom	Experimental	16.19	0.47	12.24	324.58
Custom	Small	9.56	0.17	6.20	112.36
Custom	Medium	10.94	0.22	7.22	145.33

Figure 28 Table Results

The results for Linear Regression are a mean across all nodes present in the medium dataset while for the GNN models the minimum is selected. For Linear regression the mean is higher because there are a few nodes with a really high error but in figure 29 it can be observed that most of the nodes have better prediction than even GNN.

For the GNN models the experimental dataset has the highest error, the second is the medium and the lowest error is in the small dataset. This can prove that some extra information about neighboring nodes may be more beneficial than more overall information. This is true to some extent, as the medium dataset does not have the lowest error meaning that more information at some point is better than spatial information. From a model point of view the LSTM embedded

custom model performs a little worse with a few miles difference in error than the STCONV model.

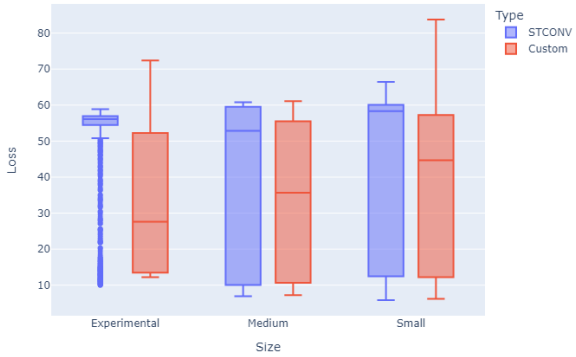


Figure 29 Boxplot for each size and model over loss

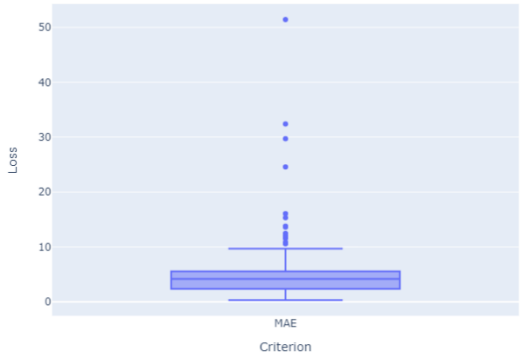


Figure 30 Boxplot for Linear Regression on MAE

In figure 29 and 30 a boxplot for Linear Regression and GNN over each dataset for the loss is represented. At each training epoch a copy of the model state is saved and evaluated. In figure 30 only the nodes with an error less than 100 are considered. This is because some of them have a error ranging in the millions and the boxplot would not be visible. For the linear regression in figure 30 all nodes are considered for the boxplot. The interquartile mean is below 10, meaning that some nodes will deliver accurate information, but the whole system is still inferior to a GNN system.

For GNN models the values have a smaller range. The interquartile mean is higher at around 40 for each model. It can be also observed that the interquartile range is closer to the minimum.

To better see the chosen values a prediction for a single node, 760256, was represented in figure 31. The data chosen is the same test data chosen for the final results. The model is good at predicting rush hour, when the values drop, which is the best time period for the model to make right predictions. It is better to know that there is traffic ahead than that it is not. During the non-rush hours when the values are higher there is a slight error of about 5 miles per hour. The figure represents 5 days' worth of data

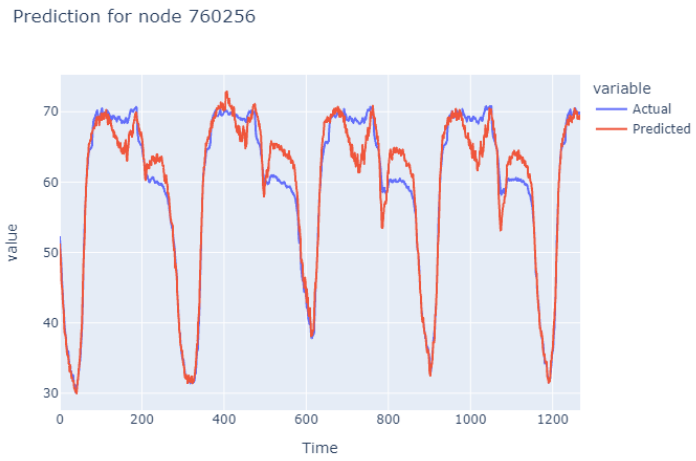


Figure 31 Actual and Predicted values for node 760256

In figure 32 a progression for each best result is shown. It can be seen that most of the training starts between 50-80 and, depending on the optimizer type, will converge at around 10. It can be seen on the red line that it converges at 12 but keeps having very small improvements and stops at 300 epochs. For the rest of the results the training stops at 50, 120 or 200 training epochs.

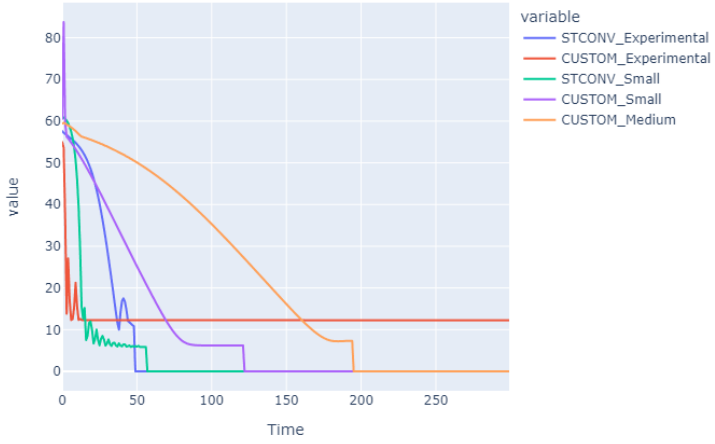


Figure 32 Training for each best result

V. CONCLUSIONS & DEVELOPMENT

Traffic is a global problem which effects can be seen on the general population or even a country's economy. This problem has caught the attention of scientist but it is a recurrent issue for many countries government as infrastructure stagnates but traffic increases. Multiple countries have developed road sensors which captures traffic information for traffic 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 for deep learning can be used on such data to predict speed values along the sensor at a moment of time based on prior experience. With the improvement of Geometric Deep Learning and 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 can still be a good baseline for traffic predictions in some situations. With enough data for each individual sensor, it can provide even better results than Graph Neural Networks. These results come at a cost since it cannot represent the whole system as good 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 GPU's and CPU's 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 do not have such a heavy time like the top-heaviest traffic cities in the world. Data from cities with more densely packed population and lower infrastructure like Bucharest could provide more insight.

VI. BIBLIOGRAPHY

[1] B. E. a. T. L. D. Schrank, "2019 Urban Mobility Report," Texas A&M Transportation Institute, 2019.

[2] "Tom Tom Traffic Index," [Online]. Available: https://www.tomtom.com/en_gb/traffic-index/ranking/.

[3] D. C.Gazis, Traffic Theory, 2006.

[4] M. K. A. Treiber, Traffic Flow Dynamics, 2013.

[5] A. L. Samuel, Some Studies in Machine Learning, 1959.

[6] A. A. R. Sathya, "Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification," in *International Journal Of Advanced Artificial Intelligence*, 2013.

[7] J. M. M. E. M. P. Mark Tranmer, Multiple Linear Regression, 2020.

[8] D. H. S. R. A. F. H. B. L. S. F. d. R. A. Ivan Nunes da Silva, Artificial Neural Networks, Springer, 2017.

[9] G. E. H. &. R. J. W. David E. Rumelhart, "Learning representations by back-propagating errors," 1986.

[10] D. M. Hawkins, "The Problem of Overfitting," Minneapolis, 2003.

[11] R. C. Moore and J. DeNero, "L1 and L2 regularization for multiclass hinge loss models," 2011.

[12] D. Berrar, "Cross-validation," 2019.

[13] P. Baldi and P. Sadowski, "Understanding Dropout," 2013.

[14] S. K. Patro and K. K. sahu, "Normalization: A Preprocessing Stage".

[15] S. Ruder, "An overview of gradient descent optimization," 2016.

[16] 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.

[17] J. Duchi, E. Hazan and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," 2011.

[18] G. Hinton, N. Srivastava and K. Swersky, "Neural Networks for Machine Learning," 2012.

[19] J. B. Diederik P. Kingma, "Adam: A Method for Stochastic Optimization," 2014.

[20] E. A. Bender and S. G. Williamson, "Lists, Decisions and Graphs. With an Introduction to Probability," 2010.

[21] T. Adali and A. Ortega, "Applications of Graph Theory [Scanning the Issue]," 2018.

[22] G. Kaiser, " Friendly Guide to Wavelets," *Physics Today*, 1994.

[23] C. D. McGillem and G. R. Cooper, "Continuous and Discrete Signal and System Analysis," 1984.

[24] W. Cao, Z. Yan, Z. He and Z. He, "A Comprehensive Survey on Geometric Deep Learning," 2020.

[25] M. W. Thomas N. Kipf, "Semi-Supervised Classification with Graph Convolutional Networks," 2016.

[26] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," 1997.

[27] H. Greenberg, "An Analysis of Traffic Flow," 1959.

[28] Y. R. Guy Leshem, "Traffic Flow Prediction using Adaboost Algorithm with Random Forests as a Weak Learner," 2007.

[29] S. t. t. f. prediction, "E. Chung, N. Rosalion," in *24th Australian Transportation Research Forum, Hobart*, Tasmania, 2001.

[30] Y. D. W. K. Z. L. a. F.-Y. W. Yisheng Lv, "Traffic Flow Prediction With Big Data: A Deep Learning Approach," 2015.

[31] K. K. ,. J.-M. S. Charalampos Bratsas, "A Comparison of Machine Learning Methods for the Prediction of Traffic Speed in Urban Places," 2019.

[32] H. Y. Z. Z. Bing Yu, "Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework," 2018.

[33] S. Guo, Y. Lin, N. Feng, C. Song and H. Wan, "Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting," 2019.

[34] X. F. C. W. J. Q. Chuanpan Zheng, "GMAN: A Graph Multi-Attention Network for Traffic Prediction," Xiamen, 2019.

[35] X. X. Y. W. H. Z. Jinyin Chen, "GC-LSTM: Graph Convolution Embedded LSTM for Dynamic Link Prediction," 2018.

[36] "Caltrans Performance Measurement System (PeMS)," California Deprtament of, [Online]. Available: https://pems.dot.ca.gov/.

[37] "Pandas," [Online]. Available: https://pandas.pydata.org.

[38] "NumPy," [Online]. Available: https://numpy.org.

[39] "Keras," [Online]. Available: https://keras.io.

[40] "PyTorch," [Online]. Available: https://pytorch.org.

[41] "PyTorch Geometric," [Online]. Available: https://pytorch-geometric.readthedocs.io/en/latest/.

[42] "PyTorch Geometric Temporal," [Online]. Available: https://pytorch-geometric-temporal.readthedocs.io/en/latest/notes/introduction.html.

[43] "CUDA," [Online]. Available: https://developer.nvidia.com/cuda-zone.

[44] "cuDNN," [Online]. Available: https://developer.nvidia.com/cudnn.

[45] "Ray Tune," [Online]. Available: https://docs.ray.io/en/ray-0.4.0/tune.html.

[46] "Plotly," [Online]. Available: https://plotly.com.

[47] "Anaconda," [Online]. Available: https://www.anaconda.com.

[48] PeMS User Guide, 2020.