University of Southampton

Faculty of Physical Sciences and Engineering
Electronics and Computer Science

A Deep Learning Method for Traffic Prediction

by **Yanzhe S. Zhang**

April 30, 2024

Project Supervisor: Professor Lie-Liang Yang
Second Examiner: Mr. Iain McNally

A project report submitted for the award of MEng
Electronic Engineering with Artificial Intelligence

-

University of Southampton

Abstract

Faculty of Physical Sciences and Engineering
Electronics and Computer Science

A Deep Learning Method for Traffic Prediction

by Yanzhe S. Zhang

This is the project report of the Part III Project.

In contemporary urban environments, efficient urban mobility is imperative for daily routines. Traffic congestion stands as a pervasive challenge in urban areas, leading to delays, frustration, and inefficiencies in transportation systems. This project aims to explore traffic predictions based on historical data using deep learning techniques. This provides insights into improving urban mobility and effectively managing traffic congestion.

Various neural network architectures were developed and evaluated, with a comprehensive analysis of the influence of different hyper-parameters on performance. Specifically, the project investigates two prediction approaches: localised predictions for specific locations within the urban map and global predictions covering the entire area.

The Tensorflow 2.15 library with Keras API is used to construct and train models.

Contents

Definitions and Abbreviations

$\mathbb{R}^n$	Euclidean n-Spaces
$b, \mathbf{b} \| \vec{b}, \mathbf{B}$	Scalers, Vectors, Tensors (Matrices)
$\mathbf{B}_{ij}$	Element at i row and j rolumn in Tensor $\mathbf{B}$
$\mathbf{B} \in \mathbb{R}^{M \times N}$	Tensor $\mathbf{B}$ have a dimension of $(M \times N)$
$\mathbf{B}^T$	Transpose of Tensor $\mathbf{B}$
G	Graph
$\mathbf{A}$	Adjacency matrix
$\mathbf{D} \in \mathbb{R}^{N \times N},\ D_{ii} = \sum_j \mathbf{A}_{ij}$	Degree matrix
$\mathbf{I}_n$	n^{th} order Identity matrix
$x, \mathbf{x}, \mathbf{X}$	Inputs
$\omega, \mathbf{w}, \mathbf{W}$	Trainables
$y, \mathbf{y}, \mathbf{Y}$	Targets
A	Activation function
σ	Sigmoid function
η	Learning rate
$\odot$	Hadamard product (Elementwise multiplication)
$*$	Convolution
∇	Differential operator

Chapter 1: Introduction

In modern, fast-paced lives, urban mobility is an integral aspect that significantly impacts our daily routines. One of the most pervasive challenges in urban areas is the issue of traffic congestion, causing delays, frustration, and inefficiencies in transportation systems. This poses high requirements for traffic management and navigation systems addressing these problems. In the topic of traffic management and planning, accurate prediction of vehicles and proactively suggesting optimal routes in urban road networks is one of the important tasks to improve traffic efficiency and safety. This project seeks to explore the realm of traffic prediction, employing deep learning techniques to provide accurate insights into future traffic conditions.

The project focuses on near-future predictions, more specifically, predicting the traffic condition in half-hour advance. There are two approaches to the aims that are covered, one is to train a model for a single location, predicting traffic at individual points on the map. This is referred to as the Localised designs in this report. By focusing on specific locations within the map, localised predictions offer detailed insights into traffic patterns and congestion levels at particular points of interest. It allows for targeted interventions and optimizations at specific locations. It can also provide a source for navigation apps with real-time traffic information and suggest routes to avoid congested areas.

The other approach is to consider an area of roads as a whole, predicting the traffic for all roads of that area at the same time, namely the Globalised designs. It offers a holistic view of traffic conditions across a certain area, enables city authorities to manage traffic flow, and implements politics to alleviate congestion on a broader scale.

Chapter 2: Theoretical Background

2.1 Machine Learning

Machine learning focuses on developing algorithms and models that enable computers to learn patterns from data and make predictions or decisions without being explicitly programmed for the task [12]. The use of statistical techniques such as linear regression allows it to improve its performance over time as it is exposed to more information. There are three commonly used machine learning approaches [1] including supervised learning, where the algorithm is trained on labeled data; unsupervised learning, where the algorithm discovers patterns in unlabeled data; and reinforcement learning, where the algorithms learn through trial and error based on feedback from its actions.

2.1.1 Performance Measure

To let computers learn the patterns from data, minimising the cost function is often the target. This function measures how close the predictions are made to the actual targets, and gives an idea of how well the performance is. In this project, mean squared error (MSE) and mean absolute percentage error (MAPE) have been considered as cost functions. The functions of the two indicators are shown below:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{2.1}$$

The MSE calculates the square of the difference between the predicted value $\hat{y}_i$ and the actual value y_i, and takes the mean of it [1]. The value is in the range of $[0, +\infty)$, when the predicted value equals the real value, MSE is 0. As the difference gets larger, MSE increases.

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{\hat{y}_i - y_i}{y_i}\right| \tag{2.2}$$

Instead of the actual value of the loss, MAPE is focused on the relative difference [5]. The range of MAPE is also $[0, +\infty)$, where a 0% MAPE means a perfect match, and above 100% would be considered as bad. In general, if the MAPE is between 5% and

10%, it can be considered a good prediction accuracy. Note that when the real value is in the denominator, which means it is not usable for any data set that contains a real value of 0.

2.1.2 Gradient Descent

Gradient descent is a widely used optimisation algorithm used to minimise the cost function iteratively. It is also a key methodology in both machine learning and deep learning. The gradient or derivative of the cost function at a point tells the direction of the steepest increase of the function. Hence if the parameters repeatedly update in the opposite direction of the gradient, the model will be closer to, and potentially reach the minimum cost. The size of a step of each update is specified by the learning rate.

For example in linear regression, we shell fit the data using a polynomial function [1]:

$$\hat{y}(x, \mathbf{w}) = \omega_0 + \omega_1 x^1 + \omega_2 x^2 + \cdots + \omega_n x^n = \sum_{i=0}^{n} \omega_i x^i \tag{2.3}$$

where the function has an order of n, with $n+1$ parameters ω_i indicating the significance of each term. These parameters are also denoted as vector $\mathbf{w}$ for future convenience. The cost function using MSE hence can be denoted as:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}(x_i, \mathbf{w}) - y_i)^2 \tag{2.4}$$

A step of gradient descent can be represented as below, where η is the learning rate. Repeat the process until it achieves a certain level of accuracy, or other convergence criteria, the polynomial will finally fit the data.

$$\omega_{new} = \omega_{old} - \eta \cdot \nabla \text{MSE} \tag{2.5}$$

During gradient descent, local minimum points are often encountered. A local minimum point refers to a point in the cost function which is smaller than its adjacent points, but not the minimum point of the entire cost function. When using gradient descent for optimisation, the algorithm may converge to such a local minimum. This situation usually occurs when the function has multiple local minima points, and the gradient descent algorithm may be limited by the choice of initial values and the direction of the local gradient. In order to solve this problem, some strategies can be adopted, such as running the algorithm multiple times and selecting the smallest result as the final result, or using improved gradient descent algorithms, such as stochastic gradient descent (SGD), momentum method (Momentum), Adam, etc.

2.1.3 Overfitting & Underfitting

Overfitting and underfitting are common problems in machine learning. They both refer to the situation where the model fails to generalise well to unseen data during training.

Overfitting stands for the situation when the performance of training data is better than the unseen data. In this case, the model learns noise and subtle patterns in training data, so that it cannot perform the same with another dataset. Such a model is overly sensitive to specific examples in the training set. Underfitting, on the other hand, refers to the situation when the model performs poorly on both training data and unseen data. The model does not capture patterns and relationships in the data well. This may be because the model is too simple for such a problem. Underfit models cannot adapt well to the complexity of the data.

Overfitting could be eased by regularisation [16], which adds a penalty term to the cost function; increasing the diversity of the training data and reducing the dependence on specific data; and early stopping which monitors the performance of the model and stops training when the performance on unseen data no longer improve. The solution to underfitting is rather simple. It could increase the model's complexity, add more features, and increase the amount of training data.

2.2 Deep Learning

Deep learning is a subset of machine learning, focused on large and complex datasets. Deep learning is the process of learning the inherent patterns of data, extracting information from text, images, sounds and so on. The goal is to enable machines to have a similar analytical learning capabilities as humans. Popular deep learning architectures include convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers.

Deep learning methods are used in this project due to it enables the computer to learn the patterns from historical traffic data, and make predictions based on those patterns.

2.2.1 Neural Network

At the core of deep learning are artificial neural networks (ANN), which are inspired by the structure and functioning of the human brain. The structure of a neural network is illustrated in Figure 2.1. It is composed of layers of interconnected nodes (neurons). The input layer is where the data is fed into the network, and the output layer produces the final result. Between the input and output layers, there are one or more hidden layers each node in the hidden layers has a usually non-linear activation function A_k. Section 2.2.2 explains them more. Lines between layers represent linear transformations, with the parameter vectors **w**, and bias b.

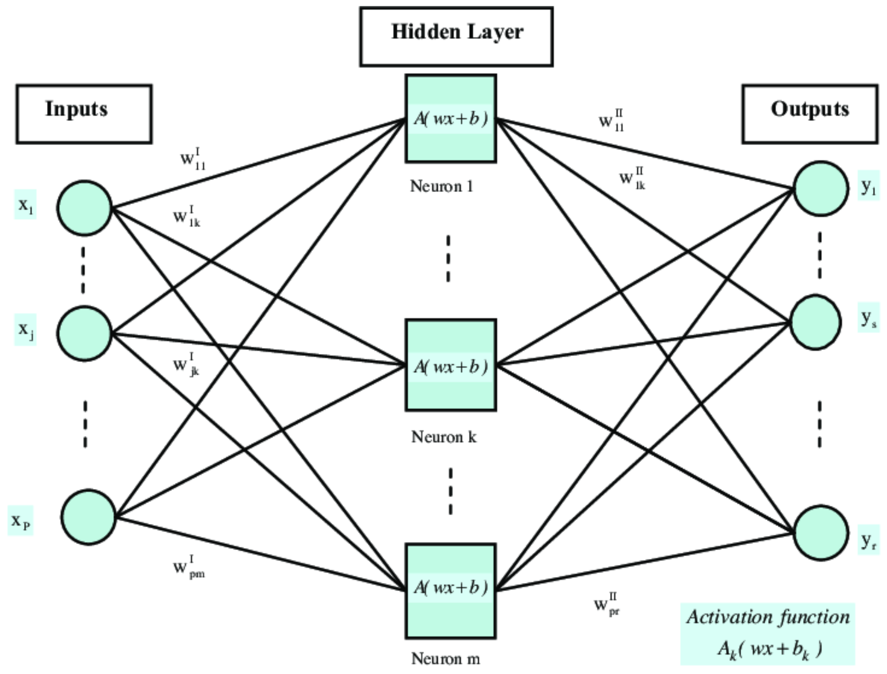


FIGURE 2.1: A simple ANN structure with one hidden layer (Adapted from [8])

What the network is trying to do is the same as machine learning: find a function that best fits the data, so it can make predictions. The process could be expressed as:

$$\begin{aligned} \mathbf{h} &= A_k(\mathbf{w^I x} + b_k^I), \quad k = 1, 2, ..., m \\ \mathbf{y} &= \mathbf{w^{II}} \cdot \mathbf{h} + b^{II} \end{aligned} \tag{2.6}$$

The inputs fed into the network first go through a linear transformation and then map to an activation function. At last, go through a linear transformation again and output. By altering the parameters and the number of hidden layers, it is possible to construct any curve.

Deep learning emphasises the use of deep neural networks, which means networks with multiple hidden layers. These networks are capable of learning complex datasets with non-intuitive characteristics. Equation 2.6 describing a single layer is constant along certain parallel hyperplanes [13], hence cannot be used to approximate high-order discontinuous complex functions. Whereas a two-layer network theoretically can form any complex functions. For a particular activation function, it's possible to construct a two-hidden-layer neural network that can approximate any continuous function defined on the unit cube in $\mathbb{R}^n$ to arbitrary precision. Specifically, this neural network would have $2n + 1$ units in the first hidden layer and $4n + 3$ units in the second hidden layer. Here n is the dimension. The proof could be found in [4].

The model with more than two hidden layers are not unnecessary. A deep network can reduce the need for the numbers of nodes in each layer [14].

2.2.2 Activation Functions

The choice of activation function is vital in neural networks. It introduces non-linearity into the model [3], and enables the network to learn complex patterns and relationships in the data. Not all non-linear functions are suitable as activation functions. The process of gradient descent requires the activation function to be differentiable so that the gradients can be calculated. Furthermore, in order to better train the neural network, the activation function is preferably monotonically increasing or monotonically decreasing, which can ensure the stability of the gradient. ReLU is a frequently used example that meets those requirements.

2.2.2.1 Rectified Linear Unit (ReLU)

ReLU is one of the most commonly used activation functions. It replaces all negative values in the inputs with zero, which can be represented by:

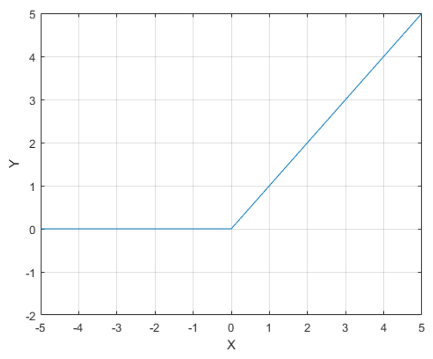
$$A(x) = \max(0, x) \tag{2.7}$$

During the training, it will not activate all neurons at the same time, which gives an advantage that it is computationally efficient. However, when $x < 0$, the gradient is zero. As training progresses, neurons may become inactive and weights fail to update. To address this issue, a variant of ReLU, Leaky ReLU, could be used instead. Leaky ReLU allows a small, non-zero gradient when the input is negative.

$$A(x) = \max(\alpha x, x) \tag{2.8}$$

where α is a small positive constant. A plot of both functions is shown in Figure 2.2

It is also possible to use the exponential linear units (ELU) to address the problem of inactive neurons. It uses $e^x - 1$, which gives a similar curve, but has a small negative value at $x < 0$.


(A) ReLU

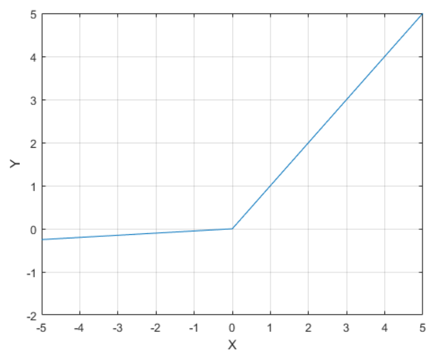

(B) Leaky ReLU

FIGURE 2.2: ReLU activation functions

2.2.2.2 Other Functions

There are many other activation functions such as sigmoid, hyperbolic tangent (Tanh), Softmax, etc. Those are suitable for different purposes. For example, sigmoid have a limited output range, suitable to use before output. However, the curve gets too smooth at two ends, which causes a low learning efficiency. It is more used in classification problems.

2.2.3 Backpropagation

Backpropagation[18] is a way to update **w** in the correct direction. It is a supervised learning algorithm that adjusts the parameters **w** by propagating the error backward from the output layer to the input layer.

It first calculates the error $\vec{\delta} = \mathbf{y} - \hat{\mathbf{y}}$, which represents the difference between output values and target values. Then, the error of each neuron in the network is calculated. In the previous example Figure 2.1, errors for hidden layers are calculated by Equation 2.9.

$$\vec{\delta_h} = \mathbf{w^{II}} \cdot \vec{\delta} \tag{2.9}$$

When the error for each neuron is computed, weights could be updated:

$$\begin{gathered}\mathbf{w^{I}_{new}} = \mathbf{w^{I}_{old}} + \eta\vec{\delta_h}\frac{\mathrm{d}A_k(\mathbf{w^{I}_{old}}\mathbf{x} + b^I_k)}{\mathrm{d}(\mathbf{w^{I}_{old}}\mathbf{x} + b^I_k)}\mathbf{x} \\ \mathbf{w^{II}_{new}} = \mathbf{w^{II}_{old}} + \eta\vec{\delta}\mathbf{h}\end{gathered} \tag{2.10}$$

where $\frac{\mathrm{d}A_k(\mathbf{w^{I}_{old}}\mathbf{x}+b^I_k)}{\mathrm{d}(\mathbf{w^{I}_{old}}\mathbf{x}+b^I_k)}$ is the derivative of the activation function. The learning rate η affects the learning speed. The initial values of $\mathbf{w^I}$ and $\mathbf{w^{II}}$ could be set randomly.

The random initial values of **w** bring randomness to the training process, making the results of each training different. This may help to overcome the local minimum, since every time the route of updates is different, there is a chance that the model will reach lower points next time.

2.3 Network Architectures / Layers

Targeting different problems, various structures of networks have been designed [1], i.e. network architectures. Different network architectures are designed to solve specific types of tasks or to address particular challenges. A network architecture contains one or several different layers. Each layer focuses on extracting patterns of a certain aspect of the data.

Figure 2.1 shows a structure of a simple Feedforward Neural Networks (FNNs), with a fully connected layer connecting every node to every node. By choosing the connections between nodes, ways to update parameters, and the activation functions, different types of layers are formed. Different layers may have their own specialisation. For example, connecting the nodes convolutionally (CNN layer) would make it particularly well-suited for grid-like data, such as images [10]. Including a CNN layer in the network is beneficial for extracting spatial patterns of the data.

By combining different layers, e.g. CNN and RNN layers, each layer responsible for extracting certain types of patterns and features, the network would be able to have a good accuracy on complex problems such as traffic prediction. The following sections will discuss more about different designs of network architectures and layers.

2.3.1 Convolutional Neural Networks (CNNs)

CNNs are widely used in tasks related to image recognition and classification. In traffic prediction, its idea could be used to extract the geographical characteristics of roads. Compared to fully connected networks, CNNs have significantly fewer nodes and connections, hence rapidly reducing the computational power required. An example of a CNN network's overall structure is shown in Figure 2.3.

The core of CNNs is the convolutional layers. They consist of a set of learnable kernels as parameters to train. Each kernel is small spatially but has the same depth as the input vectors. Kernels are convolved with the input data to produce feature maps. Let the size of the filter **w** being $(m_f \times n_f \times \dots)$, the $(a \times b)$th node in the output feature

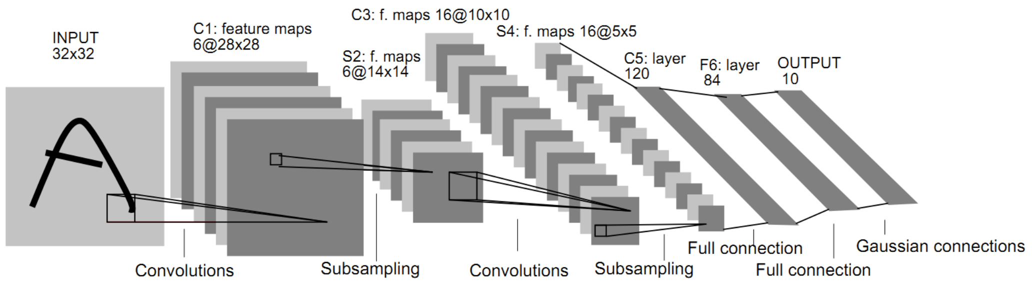


FIGURE 2.3: Architecture of LeNet-5, a convolutional NN, here used for digits recognition. Each plane is a feature map, i.e., a set of units whose weights are constrained to be identical. (Adapted from [10])

map **h** could be calculate by:

$$\mathbf{H}_{ab} = A\left(\sum_{i=a}^{m_f+a} \sum_{j=b}^{n_f+b} (\mathbf{X}_{ij} \odot \mathbf{W}_{ij})\right) \tag{2.11}$$

The activation function A is also needed to bring non-linearity.

Repeat this process step by step from the top left corner to the bottom right, spatial information is extracted. If some area of input vectors have values similar to the kernel, the corresponding value on the feature map will be large. Implies that the feature interest has been detected. To update the kernel, backpropagation is used.

As the input vectors convolute, the result will be smaller, especially when there are multiple convolution layers. The size of the vector can be maintained appropriately through padding. Padding refers to the process of adding additional values around the boundary of the input vectors, normally with zeros, before convolution.

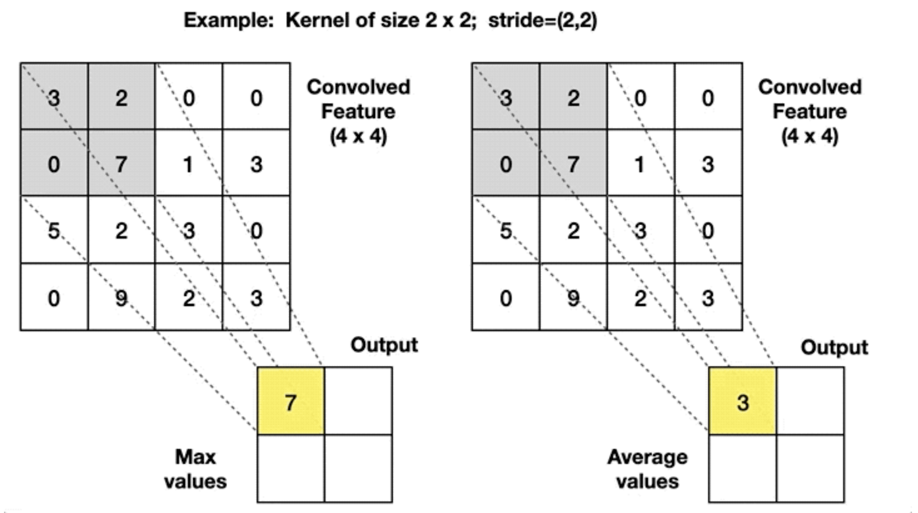


FIGURE 2.4: Illustration of Max Pooling: Take the highest value from the area; Average Pooling: Calculate the average value from the area. (Adapted from [2])

The use of pooling layers is common after convolution to reduce the spatial dimensions. It is like converting a high-resolution image into a lower-resolution one. The retained image will not have much impact on the understanding of the content. This can be done by averaging neighbourhoods or selecting the maximum value among neighbourhoods.

2.3.2 Recurrent Neural Networks (RNNs)

RNNs are a type of architecture that is designed for sequenced data where the order of the data points is crucial [11]. This makes RNN a good approach for traffic prediction. Similar to fully connected neural networks, RNNs are constructed by multiple RNN nodes in one or more RNN layers. The equation of a RNN layer is given by:

$$\mathbf{h}_t = A(\mathbf{w}^{\mathbf{I}}\mathbf{x}_t + \mathbf{w}^{\mathbf{II}}\mathbf{h}_{t-1} + b) \tag{2.12}$$

where $\mathbf{x}_t$ is the input vector at time t, $\mathbf{h}_t$ is the output vector at t, hence, $\mathbf{h}_{t-1}$ is the output at time $t-1$. Comparing this to Equation 2.6, the difference is that the RNN layer includes a term containing information from previous samples in the sequence, $\mathbf{w}^{\mathbf{II}}\mathbf{h}_{t-1}$. The essence of RNN is that the output of the current moment will take place in the calculation of the next moment as one of the inputs. Figure 2.5 illustrates the idea of RNN.

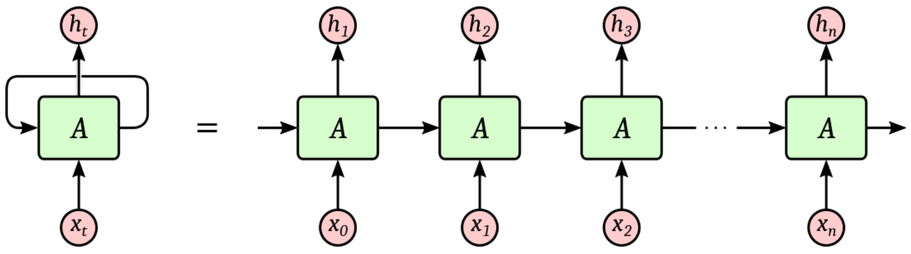


FIGURE 2.5: RNN as a neural network very deep in time (Adapted from [9])

RNNs are susceptible to the vanishing/exploding gradient problem [15]. Since the RNN uses the backpropagation to minimise the cost function, and the error has been calculated from outputs going back through the network to update the weights, those weights in the feedback loop ($\mathbf{w_{rec}}$) are multiplied a lot of times during the backpropagation. If $\mathbf{w_{rec}} < 1$, the gradient will be vanishing, since it approaches 0. In contrast, If $\mathbf{w_{rec}} > 1$, then the weight will tend to be very large, i.e. exploding.

2.3.2.1 Long Short-Term Memory (LSTM)

To address the vanishing/exploding gradient problem, multiple variations of RNN have been proposed. Long Short-Term Memory (LSTM) networks are one of the popular architectures [17]. The structure of LSTM is shown in Figure 2.6.

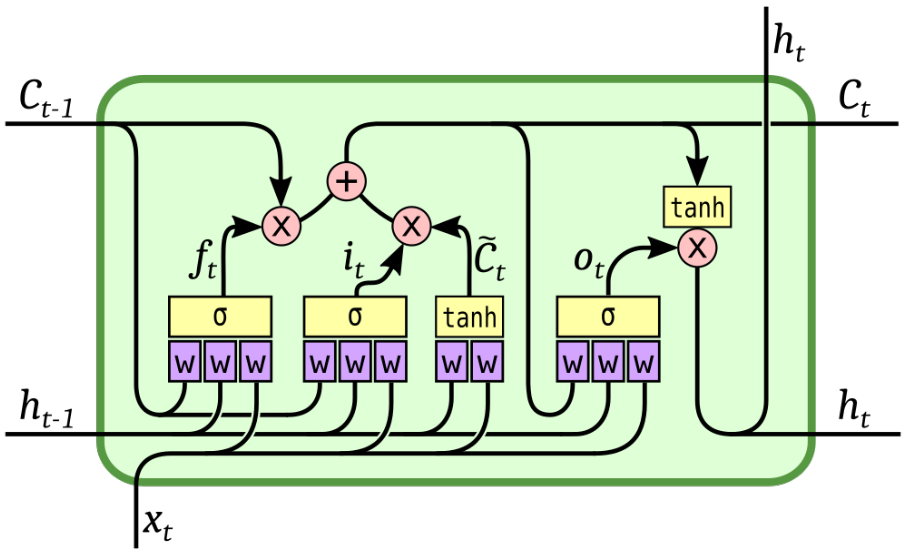


FIGURE 2.6: LSTM with peephole connections (Adapted from [9])

The LSTM network introduces three gates to manage the weights of samples in the sequenced data. f is the forget gate, i is input gate, and o is forget gate. σ represents a sigmoid function that gives a value between 0 and 1, which can act like a switch. **h** and **c** stands for hidden state and cell state. Each carries a memory path to pass down information. The equations of the model are given by:

$$\begin{aligned}
i_t &= \sigma(\mathbf{w}_{xi}\mathbf{x}_t + \mathbf{w}_{hi}\mathbf{h}_{t-1} + \mathbf{w}_{ci}\mathbf{c}_{t-1} + b_i)\\
f_t &= \sigma(\mathbf{w}_{xf}\mathbf{x}_t + \mathbf{w}_{hf}\mathbf{h}_{t-1} + \mathbf{w}_{cf}\mathbf{c}_{t-1} + b_f)\\
\tilde{\mathbf{c}}_t &= \tanh(\mathbf{w}_{xc}\mathbf{x}_t + \mathbf{w}_{hc}\mathbf{h}_{t-1} + b_c)\\
\mathbf{c}_t &= f_t \odot \mathbf{c}_{t-1} + i_t \odot \tilde{\mathbf{c}}_t\\
o_t &= \sigma(\mathbf{w}_{xo}\mathbf{x}_t + \mathbf{w}_{ho}\mathbf{h}_{t-1} + \mathbf{w}_{co}\mathbf{c}_t + b_o)\\
\mathbf{h}_t &= o_t \odot \tanh(\mathbf{c}_t)
\end{aligned} \tag{2.13}$$

These gates, or rather sigmoid functions, let LSTM choose to ignore a sample if the current sample has been considered as not important. Otherwise, the LSTM will discard the information before the sample, and only retain the information of current time t.

In the first version of LSTM, there is no output gate [15], and the equations are listed below:

$$\begin{aligned}
i_t &= \sigma(\mathbf{w}_{xi}\mathbf{x}_t + \mathbf{w}_{hi}\mathbf{h}_{t-1} + b_i)\\
f_t &= \sigma(\mathbf{w}_{xf}\mathbf{x}_t + \mathbf{w}_{hf}\mathbf{h}_{t-1} + b_f)\\
\tilde{\mathbf{h}}_t &= \tanh(\mathbf{w}_{xc}\mathbf{x}_t + \mathbf{w}_{hc}\mathbf{h}_{t-1} + b_c)\\
\mathbf{h}_t &= f_t \odot \mathbf{h}_{t-1} + i_t \odot \tilde{\mathbf{h}}_t
\end{aligned} \tag{2.14}$$

It is clearer that in this set of equations, $\mathbf{i}$ controls the weight of short-term memories, and f controls longer memories. Those two gates are independent. While training, it can find suitable parameters for $\mathbf{i}$ and f, which means $\mathbf{i}$ and f will use different $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ to give different control strategies.

The addition of the output gate and use of $\mathbf{c}$ is to keep the longer memories more effectively. $\mathbf{c}$ does not exit the output gate, hence it is not affected if the current output gate is approaching 0. Even though the current f is tend to 0, which means $\mathbf{c}_{t-1}$ has been forgotten, $\mathbf{h}_{t-1}$ still contains information about $\mathbf{c}_{t-1}$. This allows the long-term memories to pass down through the time sequence.

2.3.3 Graph Neural Networks (GNNs)

GNNs are the type of architecture that is designed to work with data that can be represented as graphs, where a graph consists of vertices and edges that connect vertices. Each vertex in the graph could be a vector of features. In the context of traffic prediction, road networks could easily convert into graphs, with sensors on the road being the vertices and roads between sensors being the edges. Suppose exists spatio-temporal series data $\mathbf{X} = \{\mathbf{X}_t \in \mathbb{R}^{N\times F} | t = 0, \ldots, T\}$, where N is the number of vertices, F is the dimension of the features. The graph at time t could be represent as $G_t = (\mathbf{V}, \mathbf{E}_t, \mathbf{A}_t)$. $\mathbf{V}$ contains all vertices, $\mathbf{E}_t$ and $\mathbf{A}_t$ are the edge set and adjacency matrix at time t. $\mathbf{E}_t$ and $\mathbf{A}_t$ may not vary with time, in that case, the graph $G = (\mathbf{V}, \mathbf{E}, \mathbf{A})$.

If the topology of the graph is given, e.g. the road network information is given, the adjacency matrix could construct based on the topology by:

$$a_{ij}^t = \begin{cases} 1, & \text{if } \mathbf{V}_\mathrm{i} \text{ connects to } \mathbf{V}_\mathrm{j} \\ 0, & \text{otherwise} \end{cases} \tag{2.15}$$

where a_{ij}^t is the element in the adjacency matrix $\mathbf{A}$ at time t. If the topology is not given, it is possible to construct $\mathbf{A}$ by calculating the distance between vertices.

$$a_{ij}^t = \begin{cases} \dfrac{\exp(-\left\|d_{ij}^t\right\|_2)}{\delta}, & \text{if } d_{ij}^t \le \epsilon \\ 0, & \text{otherwise} \end{cases} \tag{2.16}$$

where d_{ij}^t is the distance between vertices i and j at time t, ϵ is a threshold that controls the sparsity of $\mathbf{A}$, and δ is a parameter that controls the distribution.

In the traffic prediction problem, an architecture capturing both spatial and temporal characteristics is desired. To do this, one could merge spatial graph convolutional network (GCN) used to capture spatial info, with RNN methods to capture temporal characteristics.

2.3.3.1 Spatial Graph Convolutional Networks (Spatial GCNs)

The input of the GCN network is the adjacency matrix **A** and the features of each vertex **X**. Intuitively, it is possible to construct the GCN layer by calculating the dot product of **A** and **X**, then dot with the weights **W**. Thus the equation of a GCN layer could be:

$$\mathbf{H} = A(\mathbf{AXW}) \tag{2.17}$$

However, there are some limitations to using this equation [6]. Due to the vertices of the graph not connecting to themselves, the diagonal of **A** is all zeros. Hence when it multiplies with **X**, the features of the vertex itself will be ignored. It is easy to solve by adding an identity matrix **I** to **A**, so that the diagonal becomes all ones. Furthermore, **A** has not been normalised. This will change the original scale and distribution of the feature vectors after multiplication. The idea of normalisation is explained in Section 2.4.2. The normalisation of **A** can be done by $\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$. Combining these two changes, the equation is now been [7]:

$$\mathbf{H} = A\left[\mathbf{D}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})\mathbf{D}^{-\frac{1}{2}}\mathbf{XW}\right] \tag{2.18}$$

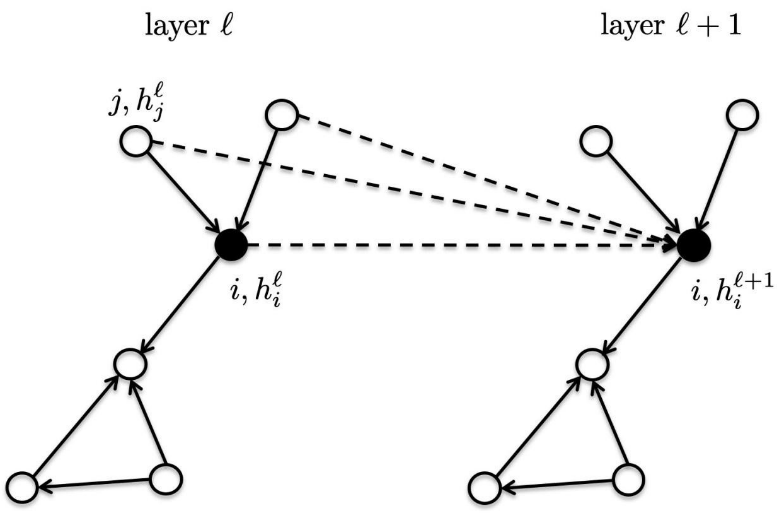


FIGURE 2.7: Illustration of the connections of vertex *i* in a GCN layer.

Refer to Figure 2.7, all the vertices that connect to the vertex *i* are considered to calculate the output of $\mathbf{h}_i^{l+1}$ The graph in this example is a directional graph, where the connections only hold in one direction. The GCN layer only considers the vertices that connect to *i* and *i* itself, but not the ones that *i* connects to.

2.4 Data Process

2.4.1 One-hot Encoding

Sometimes the data used contains categorical variables, which represent categories or labels, such as weekdays, and the class of a road. Each value of these variables is independent, the size does not matter. To let the model understand this better, one-hot encoding could be introduced.

One-hot encoding is a technique used to represent those categorical variables as binary vectors. The length of the vectors is equal to the number of unique categories, and each position in the vector corresponds to a specific category. If the data is in a category, the corresponding position is set to be 1, otherwise, fill with 0. With this implemented, each category is treated as an independent entity and does not impose any ordinal relationship.

2.4.2 Normalisation

Different features in the dataset often have different units and scales. This makes in some of the dimensions, the data more concentrated than others. In the process of gradient descent, it is hence more likely to deviate from the minimum and result in longer training time.

To minimise the effect of them, normalise data before training could be done. Normalisation is the process of limiting the preprocessed data to a certain range (such as $[0, 1]$ or $[-1, 1]$). There are many ways to normalise data. In this project, Min-Max Normalisation is used.

$$\mathbf{x}' = \frac{\mathbf{x} - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})} \tag{2.19}$$

Min-Max Normalisation converts the data into $[0, 1]$ range, the minimum value being 0, and the maximum being 1.

Due to the values after normalisation depending on the min and max values in the dataset, normalise the whole dataset will leak some subsequent information of the data to the previous samples. Consequently, normalisation of the validation set and the test set will use the factor generated by the training set, as presented in Equation 2.20.

$$\mathbf{x_{validation}}' = \frac{\mathbf{x_{validation}} - \min(\mathbf{x_{train}})}{\max(\mathbf{x_{train}}) - \min(\mathbf{x_{train}})}, \quad \mathbf{x_{test}}' = \frac{\mathbf{x_{test}} - \min(\mathbf{x_{train}})}{\max(\mathbf{x_{train}}) - \min(\mathbf{x_{train}})} \tag{2.20}$$

In this way, the training set does not contain any information about the test and validation set, and the three sets still remain consistent.

Chapter 3: Data Preprocess

3.1 Dataset Explained

The dataset was found and downloaded on the internet [19], which contains real data gathered in the city of Guiyang, China. The traffic information was processed and integrated by collecting users' geographical locations from the navigation app anonymously in real-time. The dataset is separated into three datasheets.

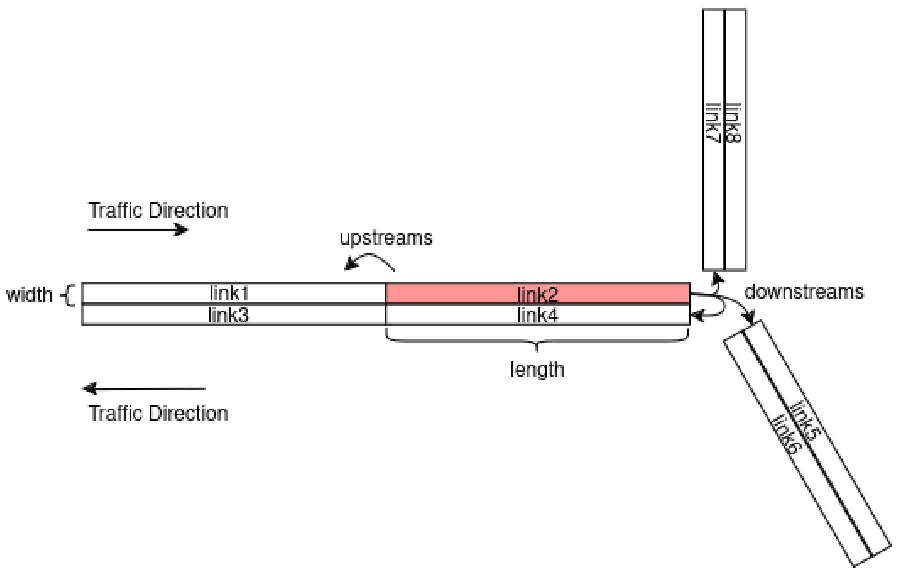
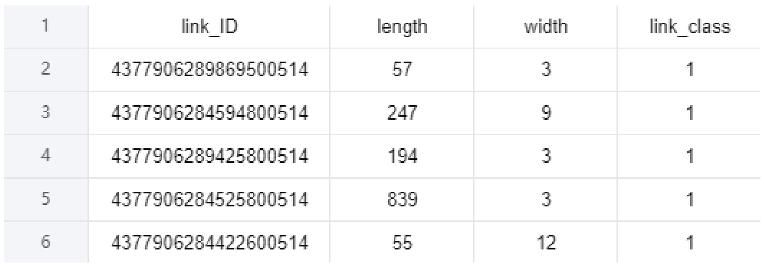


FIGURE 3.1: A diagram of links and its upstreams and downstreams

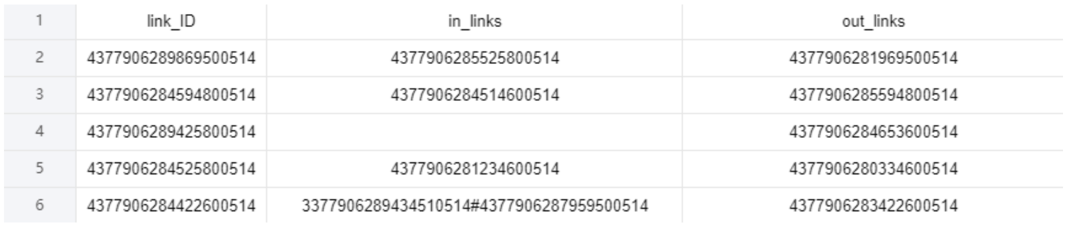
In the dataset, each direction of the traffic of a road is composed of one or multiple "links". An example of a link could be found in Figure 3.1. The first datasheet contains static information of a total of 132 links. First a few lines of the datasheet are shown in Figure 3.2. `link_ID` is a unique string ID assigned to each link; `length` and `width` of the links are integer values in meters; `link_class` is the classification of the road.

The second datasheet records the upstream and downstream relations of each link, forming the topology of the map. The headers are shown in Figure 3.3. `in_links` records the upstream `link_ID`. If there are multiple of them, a # symbol is used to separate them. Similarly, `out_links` records all the downstream roads.



1	link_ID	length	width	link_class
2	4377906289869500514	57	3	1
3	4377906284594800514	247	9	1
4	4377906289425800514	194	3	1
5	4377906284525800514	839	3	1
6	4377906284422600514	55	12	1

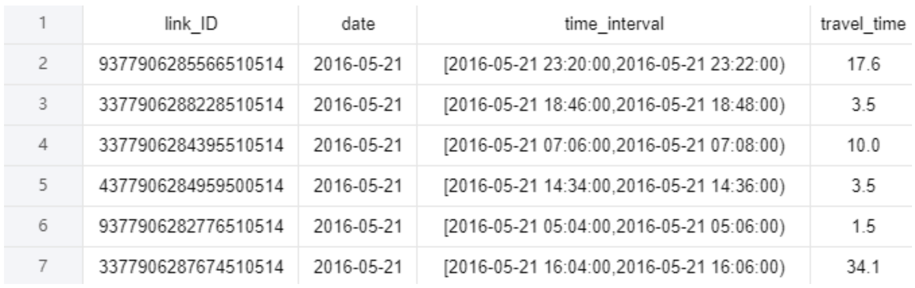
FIGURE 3.2: First five samples of link info



1	link_ID	in_links	out_links
2	4377906289869500514	4377906285525800514	4377906281969500514
3	4377906284594800514	4377906284514600514	4377906285594800514
4	4377906289425800514		4377906284653600514
5	4377906284525800514	4377906281234600514	4377906280334600514
6	4377906284422600514	3377906289434510514#4377906287959500514	4377906283422600514

FIGURE 3.3: First five samples of link topology

The third datasheet is the records of the `travel_time`, which is the temporal data. The data is sampled in a two-minute period. `travel_time` averages the time of vehicles that stay on the link within that two-minute time frame in seconds(Figure 3.4). It is a float variable that has a range of values between 0 to 120. A higher number reflects a more congested road, by predicting `travel_time`, it is able to have an idea of the traffic condition on the road. Alongside with `travel_time`, the datasheet also includes the record time interval and the date of the record. There are a total of 25,999,603 records, distributed from March to June of 2016 and from March to July of 2017.



1	link_ID	date	time_interval	travel_time
2	9377906285566510514	2016-05-21	[2016-05-21 23:20:00,2016-05-21 23:22:00)	17.6
3	3377906288228510514	2016-05-21	[2016-05-21 18:46:00,2016-05-21 18:48:00)	3.5
4	3377906284395510514	2016-05-21	[2016-05-21 07:06:00,2016-05-21 07:08:00)	10.0
5	4377906284959500514	2016-05-21	[2016-05-21 14:34:00,2016-05-21 14:36:00)	3.5
6	9377906282776510514	2016-05-21	[2016-05-21 05:04:00,2016-05-21 05:06:00)	1.5
7	3377906287674510514	2016-05-21	[2016-05-21 16:04:00,2016-05-21 16:06:00)	34.1

FIGURE 3.4: First six samples of records

3.2 Data Analysis & Preparation

3.2.1 Static Data

All links in the dataset have a `link_class` of 1, hence it is abandoned from further usage. The relation of the `length` and `width` of links and the `travel_time` has been studied, and the plot is shown in Figure 3.5. In the plot, length has a strong positive

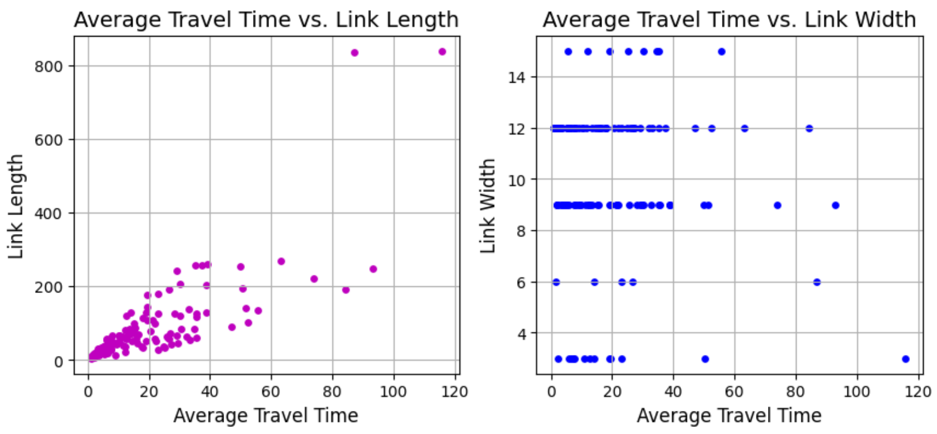


FIGURE 3.5: Length & Width vs. Average Travel Time

correlation with travel time, whereas width seems to have a range of distribution. As shown in the plot, most of the links have a length under 300 meters. Two minutes is more than enough for vehicles to pass the length. Hence generally the longer the links, the longer time it takes for vehicles to pass it. Both `length` and `width` can be used as features, this forms a static input shape of $\mathbb{R}^{132\times 2}$.

3.2.2 Spatial Data

Since vehicles cannot vanish on the road, the history traffic status of upstreams of a link is highly related to its future condition. An adjacency matrix $\mathbf{A} \in \mathbb{R}^{132\times 132}$ is created based on the topology of the 132 links. Number i row in $\mathbf{A}$ represent link i, and if the link i has a downstream of j, then $\mathbf{A}_{ij} = 1$. This is the method described in Equation 2.15.

Aside from the graph approach, it is also possible to include the spatial information by adding the travel time of the last time frame of the upstreams as features. There are a maximum of 4 upstreams a link can have in the given map. Hence 4 additional features are added to each sample. The features contain `travel_time` at the last time frame of the upstream links, 0 is initialised if not present.

3.2.3 Temporal Data

As mentioned in Section 3.2.1, the `travel_time` should be the target to predict. The datasheet contains other temporal information: year, month, date, and time of the day. Figure 3.6.A illustrates that different months and years do have different average travel time, hence they could be included as features.

The temporal features i.e. years, months, etc. are integer encoded. This might lead to the model incorrectly assuming magnitude relationships between categories. To eliminate this misunderstanding, one-hot encoding is used. `month` is encoded into 12 features, each feature represents a month. On the other hand, although `year` in this dataset contains only 2 values, 2016 and 2017, considering the extensibility of the model, `year` stays as integer values.

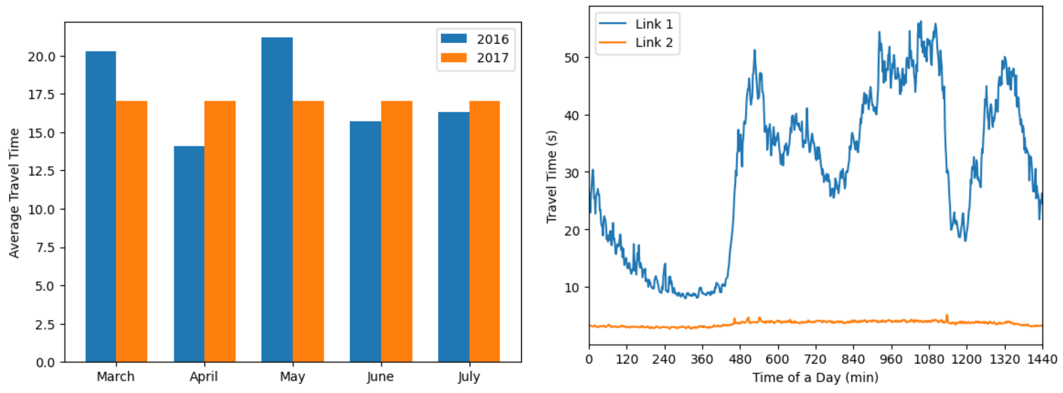


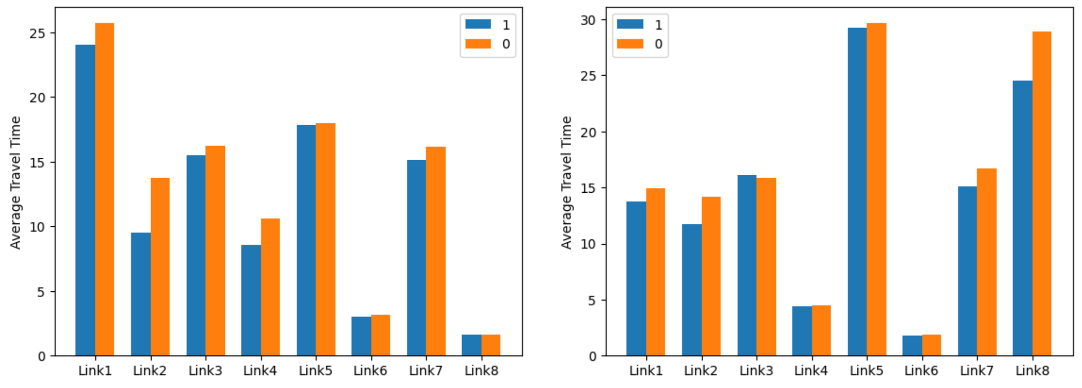
FIGURE 3.6: (A) Average travel time varies with different month and year; (B) Time of day vs. average travel time of two randomly choosed links

The time of the day also reveals characteristics. Figure 3.6. B shows the plot of two randomly chosen links. Link1 varies a lot at different times. In contrast, Link2 gives a much flattened curve. It implies that the time of a day of different links has a different impact on the travel time. The time of day is appropriate to be a feature. Due to the fact that the value of the time is not ordinal, the one-hot encoding should be used. However, this way one-hot encoding needs to create too many features that require large graphics memory in the GPU. To reduce the high memory consumption, weather it is rush hour is used as a substitution. From this plot, rush hours when the curve changes the most could be determined. 7-9 am. is considered the morning rush hour, and 5-7 pm. is the evening rush hour.

Based on the information given in the dataset, some other features can be obtained using external sources. For example, holiday and weekday information. Weekdays are determined using `datetime` library and holidays can be determined by a library called `chinese_calendar`, which records the local holidays of the given dataset.

Whether it is a holiday affects people's behaviour, and weekdays are selected since some of the links reveal periodicity throughout the week. By plotting them with the travel time on Figure 3.7, it can be seen that the weekdays are generally higher in terms of travel time than holidays and weekends.

It has been noticed that most links have missing data at various time segments. No matter what method is used to try to fill these missings, there will always be varying degrees of distortion. Also in actual applications, the model is likely to encounter



(A) Holiday (1); Working day (0) (B) Weekend (1); Weekday (0)

FIGURE 3.7: 8 randomly choosed links are used to compare (A) holiday (B) weekend with normal working day

missing values. If the model does not learn how to deal with missing values, the performance in real applications may be affected. Hence, a separate feature `is_missing` telling if the sample is missing is used. Let the model learn the missing values itself. For samples that are missing, all other features are 0 expect `is_missing`.

3.3 Train-Test Set Split

The dataset is separated into three sets: training set, validation set, and test set. The whole dataset contains samples from March, 1, 2016 to June, 1, 2016, and March, 1, 2017 to July, 1, 2017. The test set includes the last 10 days of both periods. It is used to evaluate the final performance of the trained model, providing an unbiased estimate of the performance on unseen data. The validation set takes another 20% of samples. It is used along with the training set while training the model. It helps in selecting the best-performing hyperparameters of the model, preventing overfitting. The samples left are the training set. This is the set that is used to train the model. It consists of features as input data along with their corresponding target labels, `travel_time`.

3.4 Sequence Generation

A sequence has to be generated as input of the RNN layers. It uses a certain amount of previous samples to predict future ones. An hour (30 samples) of previous samples is decided to be used to predict the following half-hour (15 samples) targets. This will give an input dimension of $\mathbf{X} \in \mathbb{R}^{N \times S \times F}$, where F is the number of features, $S = 30$ refers to the sequence length which is the 30 previous samples, and N being the number of sequences that generated from the training set. The targets, on the other hand, have a dimension of $\mathbf{Y} \in \mathbb{R}^{N \times H \times 1}$. $H = 15$ is the output length. Features expect the travel time is not included in the output sequence.

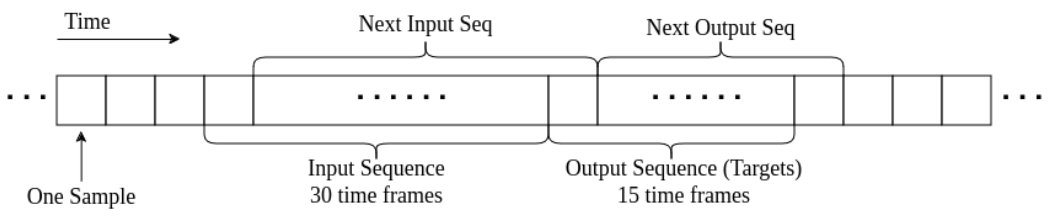


FIGURE 3.8: Illustration of sequences

For each different set of 45 continuous samples, a unique input sequence can be constructed. Hence for a training set continuous in time, B could be calculated by: $N = M - S - H$ where there are M samples in the training set. It is worth noting that the features of the targets are not included in the input in this design. This may cause the model to miss some vital information about the target time frame, for example, whether the target value is missing, whether the targets are in rush hours, etc. In other words, when training the model with this sequence generator, it not only captures the pattern of the features on the `travel_time` of target time frames but also has to capture the patterns of features and make predictions on future features.

Chapter 4: Model Designs

Tensorflow 2 with Keras API is chosen to accomplish the design of neural networks and train them. GPU is used to accelerate the training process. Detailed development tools and versions are listed in Table B.1.

The process of training is to use the input data and the targets in the training set to adjust the weight vectors and make the prediction results as close as possible to the target value. The model will iterate through the training set a number of times. Each time is called an epoch. In theory, the performance of each traversal will be better than the last epoch. In each epoch, the training data is divided into batches, and each batch contains B input sequences, B being the batch size. The reason to divide it into batches is to enable GPU to do parallel training, significantly saving time consumption. Each batch is randomly sampled from the entire training set, which can increase the randomness of the data, and reduce the model's dependence on specific batches of data.

The early stopping method is used throughout the training process. It is a process to monitor some indicators and stop training when a certain condition is met. For all the model designs, the MSE is monitored and the training will stop when the loss stops decreasing for 10 epochs.

4.1 Localised Designs

4.1.1 Pure Temporal Design

A simple LSTM model is designed to extract temporal information. A total of 19 features for each sample are used as input. The static features are not included since they are constant for a single location. Table 4.1 gives all the features used and the positions they are at. In the process of training, all the features are treated as `float32`. The types in the table are just an indication of what kind of values they might be.

Figure 4.1 shows the structure of the model. It is a sequential model in which each layer is connected one after another.

features	position	type	features	position	type
`year`	0	float	`is_morning_rush_hour`	15	bool
`month`	1-12	bool	`is_evening_rush_hour`	16	bool
`is_weekday`	13	bool	`is_missing`	17	int
`is_workday`	14	bool	`travel_time`	18	float

TABLE 4.1: Dimensions of samples for a single link

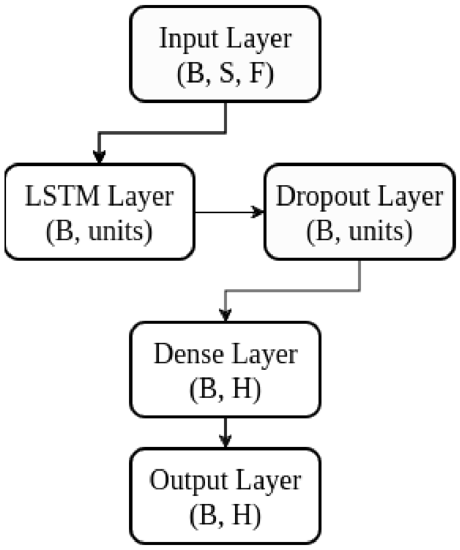


FIGURE 4.1: Basic structure of the simple LSTM model, with the notes of the dimension changes

The input layer feeds the input sequence into the LSTM layer, with a dimension of $\mathbf{X} \in \mathbb{R}^{B\times30\times19}$. The LSTM layer with different values of units will then extract temporal information for each sequence, and change the dimension to $(B \times \text{units})$. The unit is a parameter that indicates the number of LSTM nodes in the layer. It is the length of the weight vector $\mathbf{w}$. Only the units in the last time frame of the sequence will be output from the LSTM layer. Those values are used to form the final targets.

To give a more powerful model, more LSTM layers could be added in series before the dropout layer. Multiple LSTM layers could increase the capacity of the model, the layer hence could capture the long-term patterns better. Each layer can learn feature representations at different levels of abstraction. Instead of returning the final $\mathbf{h}_t$, every output of the sequence should be returned except the last LSTM layer. This way, the layers could be connected sequentially.

The dropout layer randomly discards a certain amount of neurons, thereby forcing the model to have a more robust performance. The discard ratio can be adjusted by tuning the dropout rate.

A dense layer is added to the end to give the correct output dimension. It is fully connected and hence has a great amount of weights to train. The activation function of this layer is the exponential ReLU function.

This results in a total of 49915 trainable parameters. With this design, the effects on the final performance by the number of LSTM nodes, the dropout rate, and the batch size have been studied.

4.1.2 LSTM with Spatial Features

As explained in Section 3.2.2, features of the speed at the last time frame from the upstreams are added to the inputs. The resulting input dimension would now be $\mathbf{X} \in \mathbb{R}^{B\times 30\times 23}$. Everything else stays the same as the pure temporal model, the increase in performance by adding the spatial features would be quantified.

4.2 Globalised Designs

Another approach to predict the traffic speed is to consider the map as a whole and make predictions of every link at the same time. Two complicated models compared to the localised one were designed.

The main difference between this approach from the localised designs is the sequenced data. Now the model is dealing with a total of 132 links, each link contains some statical and temporal features. The statical features are now in use, giving 2 additional features for every link. The temporal data are the features from Table 4.1, does not change.

Combining all those together, the models now have three inputs in total: temporal inputs with a dimension of $\mathbf{X_{temp}} \in \mathbb{R}^{N\times V\times F_t}$ where V stands for the number of links, F_t is the feature count of the temporal data; statical inputs $\mathbf{X_{stat}} \in \mathbb{R}^{V\times F_s}$, where F_s is the feature count of static data; and finally, the spatial inputs $\mathbf{A} \in \mathbb{R}^{V\times V}$, which is the adjacency matrix of the map.

The model is also trained by the number of batches in each epoch, hence all three inputs need to add a dimension of batches. The statical and spatial inputs are broadcasted to match the dimensions. Similar to the localised designs, the temporal inputs also need to be in sequences for LSTM layers to train. This makes the input shape of temporal inputs $\mathbf{X_{temp}} \in \mathbb{R}^{B\times S\times V\times F_t}$. This process creates a lot of duplicates of the data. Due to the memory usage restraint, a generator generates data used in each batch is used, rather than feeding the whole dataset into the model at once.

The outputs also contain the predictions of all links. The output shape is hence $\mathbf{H} \in \mathbb{R}^{B\times H\times V}$. For each time frame, a prediction of the speed of every link is structured.

4.2.1 Spatio-Temporal LSTM (ST-LSTM)

The structure of one design that can deal with spatial and temporal information is illustrated in Figure 4.2. It is a combination of the use of CNN layers and RNN layers.

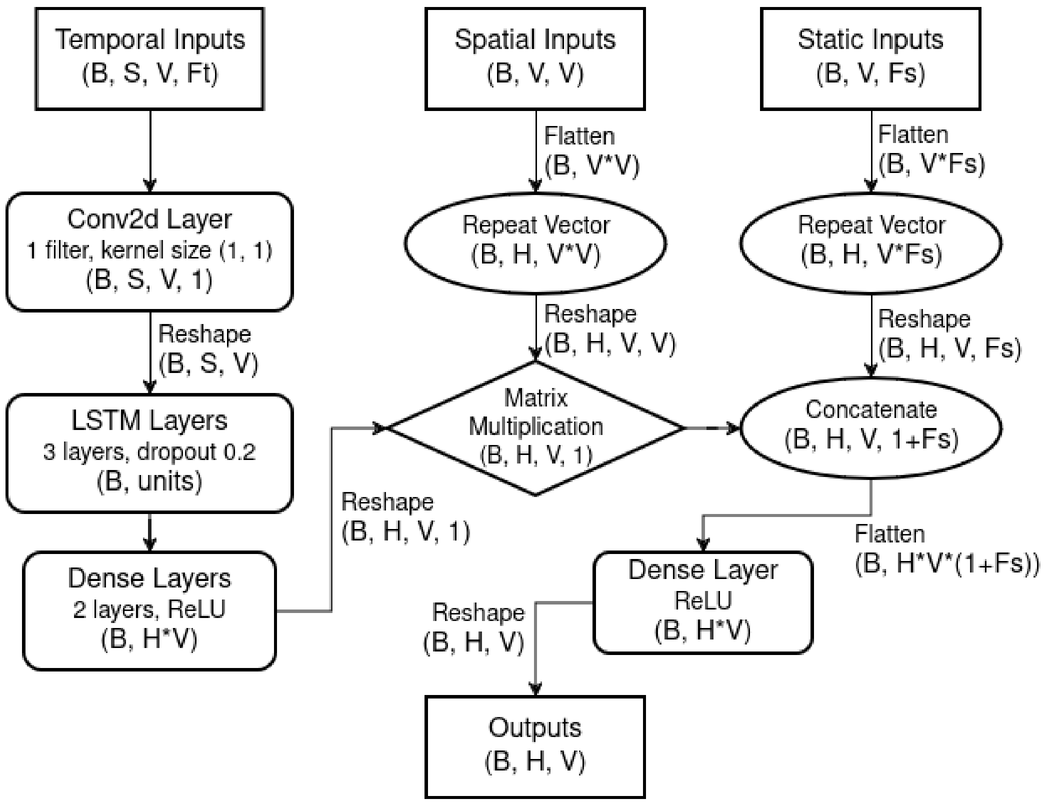


FIGURE 4.2: Structure of the Spatio-Temporal LSTM design illustrated in a flowchart

In the sequences of the temporal data, each time frame contains two-dimensional features: V and F_t. However, the LSTM layer can only handle one-dimensional features for one time step. This could be sorted by using a two-dimensional convolutional layer with 1 filter and a kernel size of (1, 1), to compress the information contained by features in F_t into 1 feature. So now each link is related to only one feature. The LSTM layer hence can take that feature of a total of 132 links as features of the time frame.

Now the LSTM layer is dealing with the whole map of data. It needs to be more powerful compared to ones of the localised designs. A three-layer LSTM sequence is used to extract temporal information. The three LSTM layers are connected sequentially, each containing 100 nodes, The ones after using the output sequence generated from the previous layer as inputs. The final layer outputs a vector of 100 units.

Inherit from the localised designs, two dense layers are used to match the dimension for future usage. The spatial inputs and static inputs are repeated first to match the target length, they are all the same across the time frames of the target. The processed temporal data is then multiplied with the spatial input, to merge the two data together.

This is when the temporal information is combined with the spatial information. The outcome of this stage will give a dimension of $\mathbb{R}^{B \times H \times V \times 1}$.

The static data is then concatenated with it to extend the last dimension. Finally, a dense layer learning the statical and spatial data added is then used before the final output. The model contains a total of 31365780 trainable parameters, even considering 237620 parameters per link, it is still much more than the localised design.

4.2.2 GCN-LSTM

In the ST-LSTM design, the spatial information is trained by a dense layer at the end. A dense layer is more effective for extracting high-level features and is suitable for data that is not affected by internal structures. It works to extract spatial information, but it is certainly not the best choice. Also, the input is merged into the sequenced temporal data after the extraction of temporal information by LSTM layers. This may cause the model to lose its vision of the spatial perspective when dealing with temporal data.

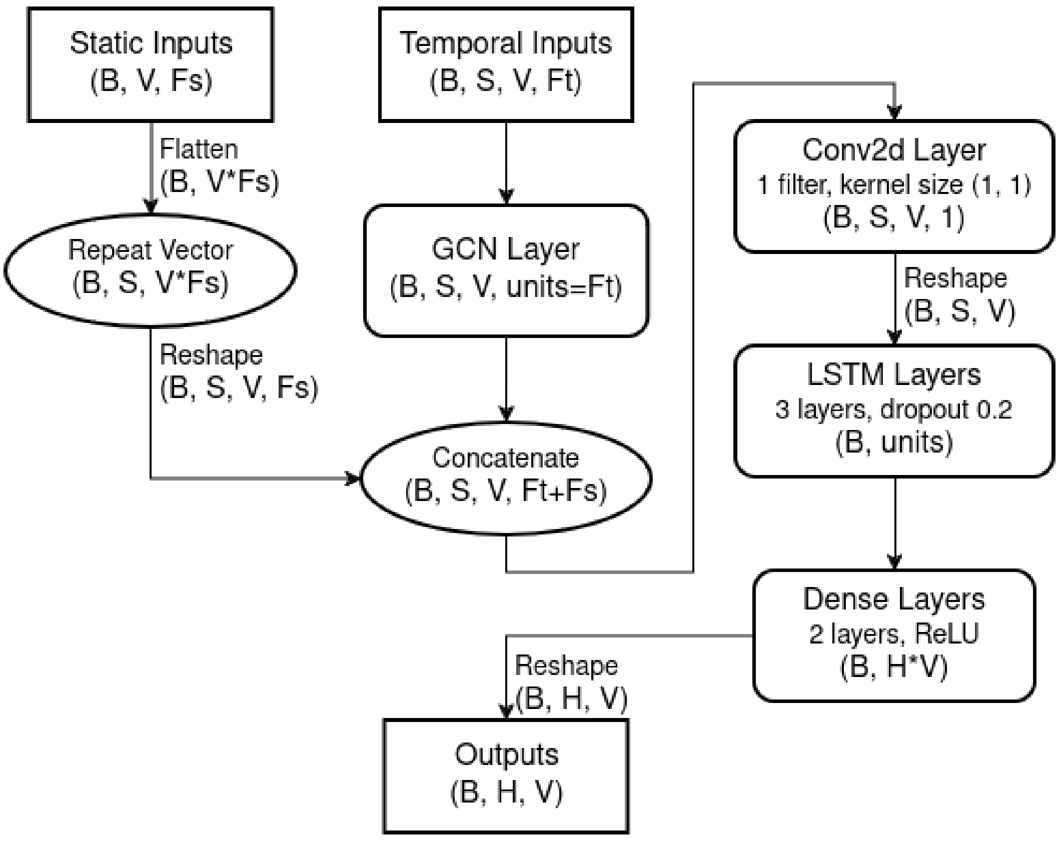


FIGURE 4.3: Structure of the GCN-LSTM design illustrated in a flowchart

GCN layer is a better choice here. Since the adjacent matrix is not changing over time frames, the spatial information could be implemented into the GCN layer before training the data. Refering to Equation 2.18, $\mathbf{D}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})\mathbf{D}^{-\frac{1}{2}}$ is given as the model is defined.

Figure 4.3 shows the structure of the new GCN-LSTM design. This design is inspired by the STGCN design from [20], which uses two temporal gated convolution layers to sandwich a spatial graph convolution layer. In this design, the spatial inputs no longer exist since the matrix is implanted into the GCN layer.

Before the convolutional layer and LSTM layers, the GCN layer with 19 units (which is the same amount as F_t) is used. It hence does not change the dimension of $\mathbf{X_{temp}}$, but the features are updated based on the road topology. Along with the spatial information, the statical inputs are also merged before the extraction of the temporal information.

In this way, it should be expected to get a better performance. In addition, there are only 2337381 trainable parameters used in total. It is significantly less than the STLSTM design, only 7.45%.

Chapter 5: Model Trainings & Results

5.1 Localised Designs

Using the simple LSTM design, the number of the LSTM units, the dropout rate, and the batch size are the parameters studied in this section. The model has the initial values of 100 LSTM units, a 0.2 dropout rate, and a batch size of 360.

5.1.1 Selecting LSTM Units

Firstly, use 10, 30, 50, 100, 150, 200, and 500 LSTM units to train the model respectively, and record the MSE loss of each epoch. The results are plotted in Figure 5.1.

In general, the plot shows that the more the LSTM units, the quicker the loss reduces, and the final loss also reduces. In theory, larger units result in an LSTM layer with more parameters, which gives the model more capacities to be better at learning complex patterns. The LSTM has been designed to effectively capture longer-term dependences, more parameters could also enhance this ability of the model. Hence, would improve the model's performance.

Aside from the LSTM layer, an increase in LSTM units also increases the number of connections in the dense layer afterward. It not only increases the capability of the LSTM layer but also the dense layer.

For both the LSTM layer and the dense layer, the time taken for each epoch increases as the number of LSTM units increases. This is straightforward since there are more variables to train each time. Increasing time consuming is negligible for first a few numbers of units, but it has a tendency to increase exponentially referring to Figure 5.2.

Small units like 10, clearly does not have the ability to extract common patterns of the data. The validation loss skyrocketed as the training loss slowly decreased. The model is overfitting the training data and is not able to adapt unseen data. On the other hand, for larger numbers of units, there is a sudden rise in the training loss. It is a sign of the learning rate being too high. The severe fluctuations of the validation loss are another

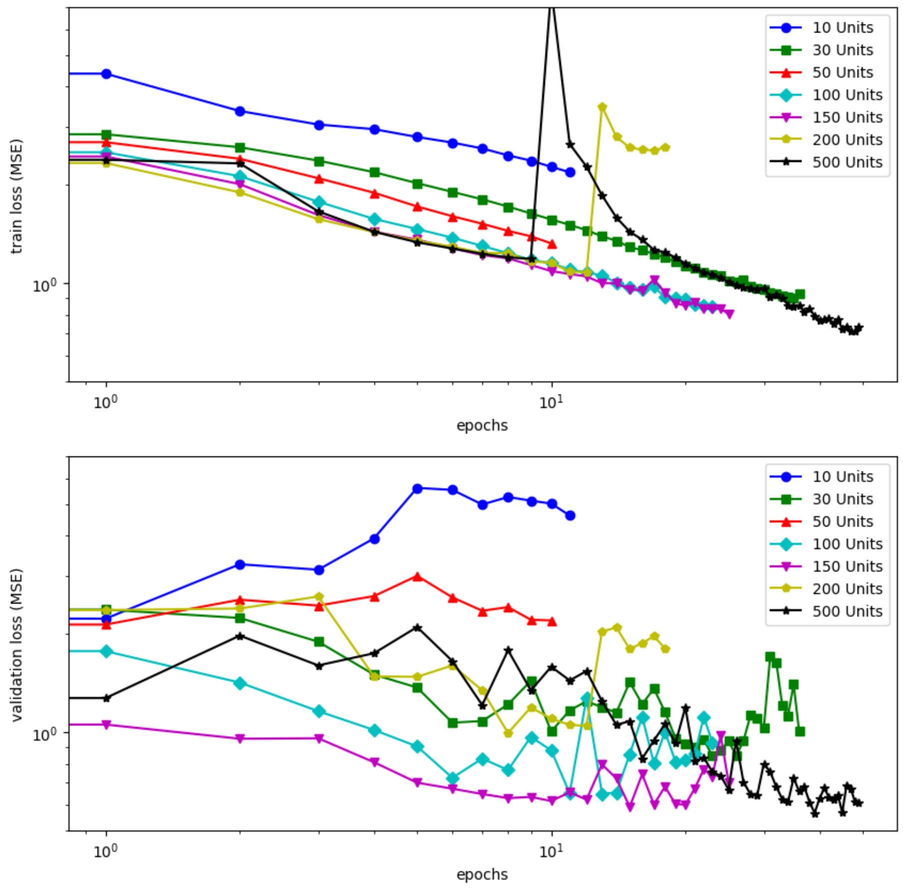


FIGURE 5.1: MSE losses vs. Epochs of different numbers of LSTM units for both train and validation set

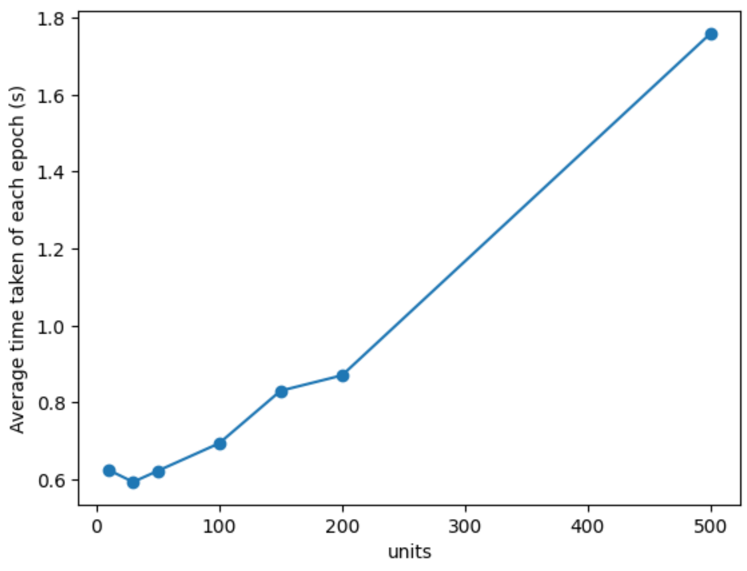


FIGURE 5.2: Average time taken of each epoch vs. number of LSTM units

symptom. A learning rate that is either too high or too low may lead to fluctuations in loss.

A high learning rate may cause parameter updates too far, which could easily jump over the minimum point, and make it difficult to converge; while a low learning rate may cause the convergence speed to be too slow, causing the model to fall into a local minimum. This could be optimised by adding a learning rate scheduler in callback functions that update the learning rate dynamically. The scheduler starts with a larger value of learning rate, 0.001. This learning rate does not change throughout the first 5 epochs. Starting at the 6^{th} epoch, the learning rate will decrease by a factor of 0.9 every epoch.

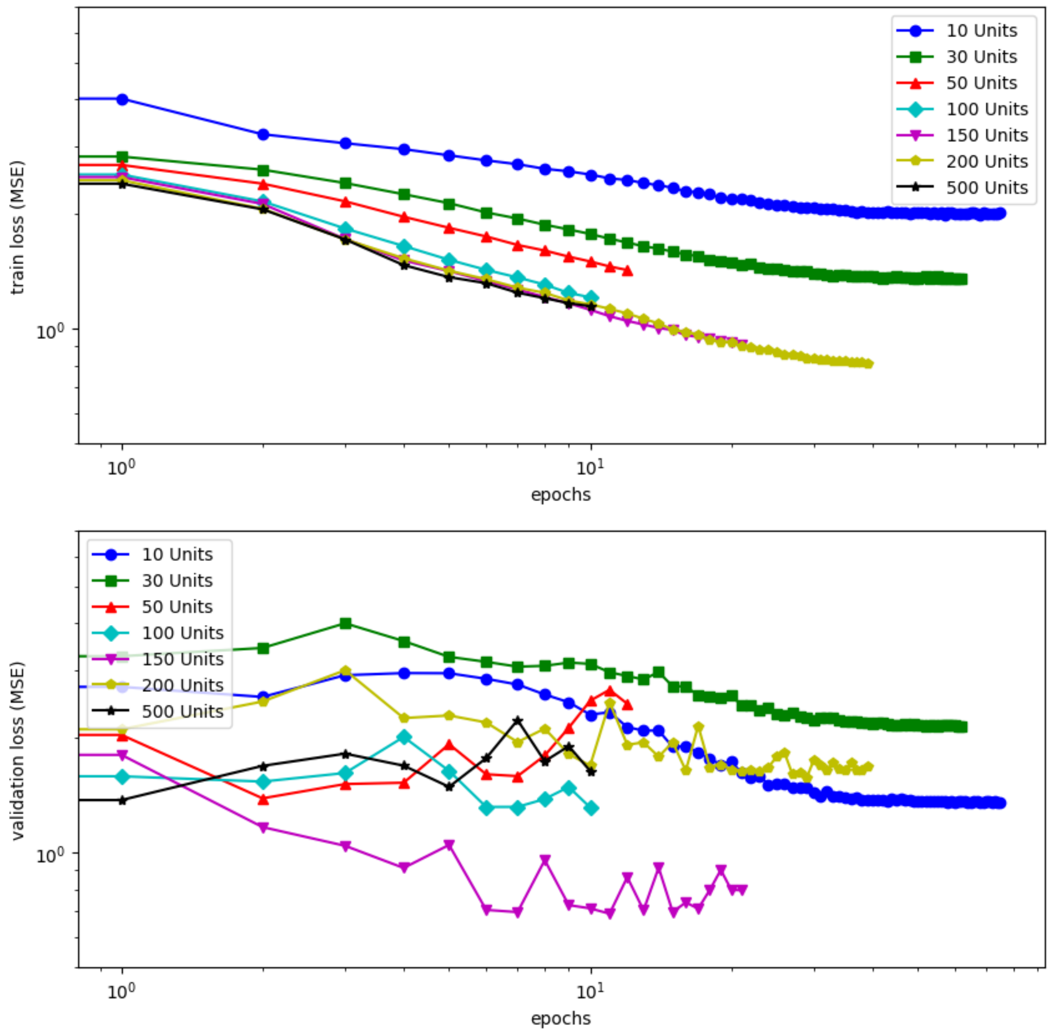


FIGURE 5.3: MSE losses vs. Epochs of different numbers of LSTM units with the learning rate scheduler

After applying the learning rate scheduler, the results are shown in Figure 5.3. This time the curves of both train losses and validation losses are smoother, and there is no sudden rise in training loss anymore. The largest fluctuations are at about 5^{th} to 10^{th} epoch, which is the starting point of the learning rate starts decreasing. This tells that the scheduler indeed improves the model as expected.

For larger numbers of units, the training loss does not improve since 100 LSTM units. Some of them end earlier due to there being no significant drop in validation loss. This implies that it is the best performance this architecture given the same input can reach. On the other hand, 10, 30, and 50 units underfit the data, the losses are high in both training and validation.

It has been noticed that each time the curve of the training and validation loss over epochs is different, especially for the validation set. Even the final validation loss is different every time the model. This is due to several reasons. Firstly, at the beginning of training, model parameters are initialised randomly. Therefore, even if the same architecture and hyperparameters are used, the initial state of the model is different, which will result in different training outcomes. Secondly, in each epoch, the model is updated based on a different batch of data. The sequences in each batch are randomly selected, which will lead to different perturbations for each training. This affects the updated trajectory of the model. 150 units are chosen due to it gives more stable results without having a massive increase in time consumption.

5.1.2 Selecting Dropout Rates

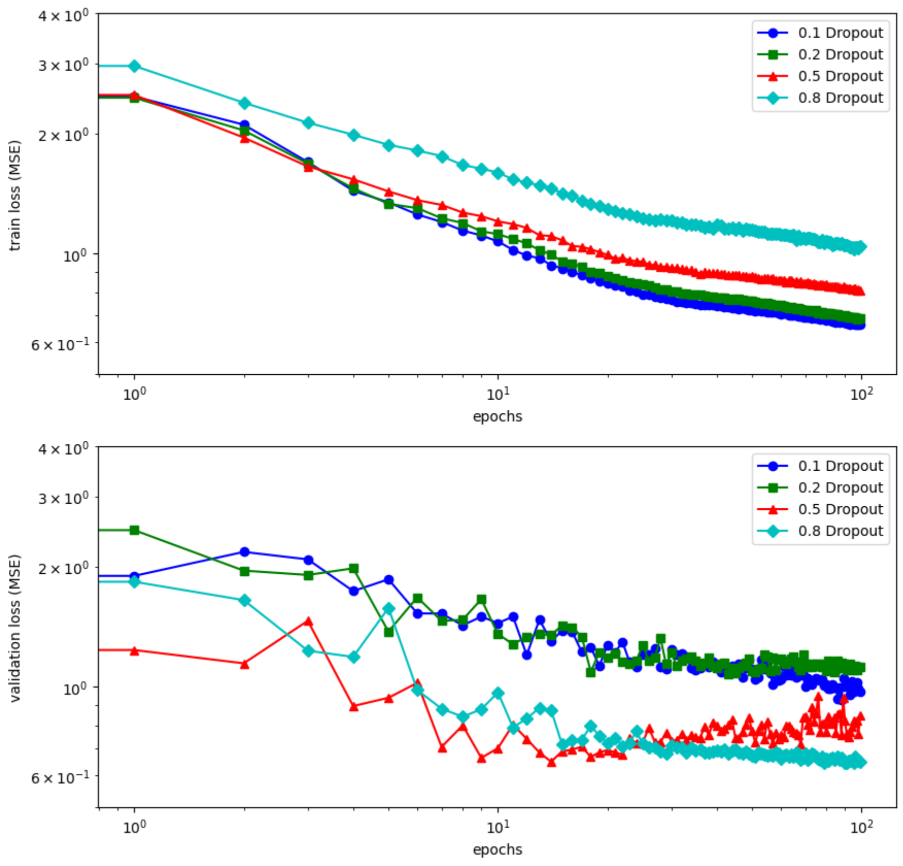


FIGURE 5.4: MSE losses vs. Epochs of different dropout rates for both train and validation set

The dropout rates used in this study are 0.1, 0.2, 0.5 and 0.8. Each takes out 10%, 20%, 50%, and 80% nodes from the network. This is designed to prevent overfitting, hence the early stopping strategy is not used to have a better view of the effects caused by the dropout rate. Figure 5.4 contains the results of them.

A higher dropout rate can more strongly reduce the overfitting of the model. It discards more nodes in the network, making the network less dependent on specific nodes. However, a higher dropout rate also reduces the capacity of the model and causes the performance of the training set to decrease. On the plot, the lowest dropout rates (0.1 and 0.2) give the lowest training loss, whereas the validation loss of them is higher.

An appropriate dropout rate is often a trade-off between overfitting and underfitting, thereby improving the overall performance of the model. The dropout rate of 0.5 gives roughly the same training and validation loss, it is hence the best choice.

5.1.3 Selecting Batch Sizes

Batch sizes are chosen from 10, 30, 120, 360, 720, and 1440. A large batch size could improve the training speed because each parameter update requires less calculation. Using a larger batch size allows the process of gradient descent to parallelise the computation to a greater extent because the sequences could be split across different computation units on the GPU and computed at the same time. Figure 5.5 shows an inverse proportional relationship between the time taken for one epoch and the batch size.

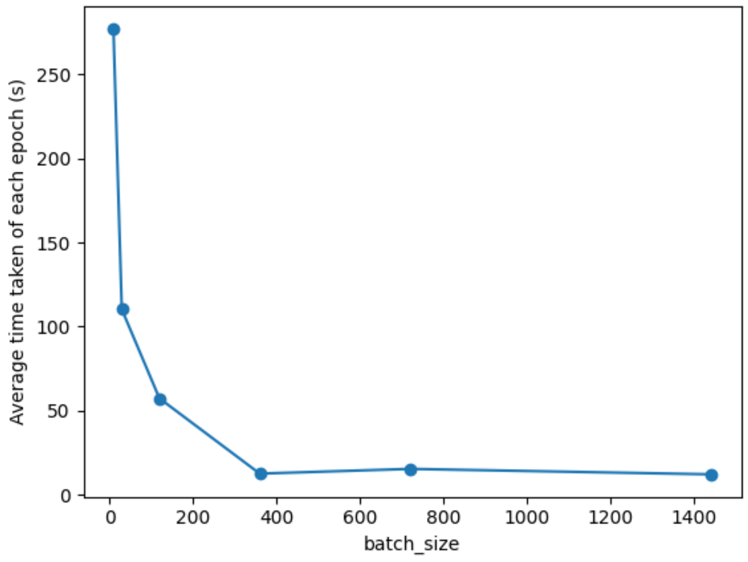


FIGURE 5.5: Average time taken of each epoch vs. batch sizes

Different batch sizes may require different learning rates for optimal performance. Typically, larger batch sizes allow for larger learning rates, while smaller batch sizes may require smaller learning rates to avoid exploding or vanishing gradient problems. To prevent overfitting, the learning rate scheduler explained in Section 5.1.1 and the early stopping strategy are applied. Figure 5.6 gives the resulting losses of different batch sizes.

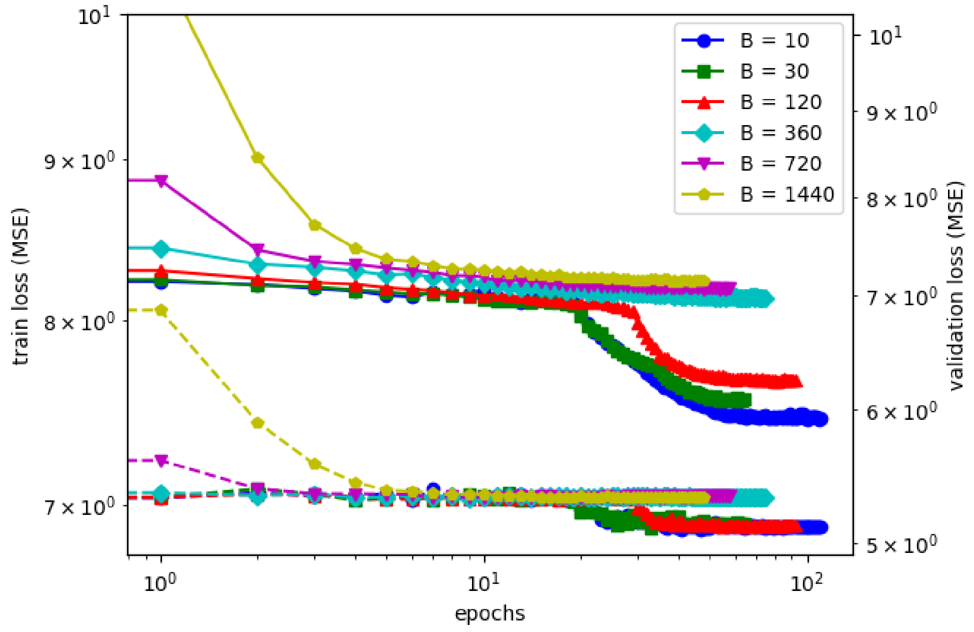


FIGURE 5.6: MSE losses vs. Epochs of different batch sizes for both train and validation set

In general, the number of iterations required to run an epoch is reduced as the batch size increases. To achieve the same accuracy, more epochs are needed. Memory usage is also a key factor when selecting the batch size. Smaller batch sizes can reduce memory consumption because fewer intermediate results need to be stored for each batch.

A larger batch size can improve the stability of the model and reduce the randomness of parameter updates. However, if the batch size is too large, it may cause the model to fall into a local minimum and be difficult to escape. In the experiment, batch sizes beyond 120 fall into a local minimum, in contrast, small batch sizes have a significant drop during later epochs. The drop happened both in train loss and validation loss, showing that this is indeed a better performance rather than overfitting.

5.1.4 Addition of Spatial Features

By adding the spatial features, with 150 LSTM units, 0.5 dropout rate, 120 batch size, and the implementation of a learning rate scheduler, the training loss of the model drops significantly. A plot comparing the previous pure temporal model and the new model is presented in Figure 5.7. Both models used the same set of hyperparameters.

The spatial features bring a 10% reduction in the final train loss, this is significant compared to the difference in previous experiments. The validation loss, on the other hand,

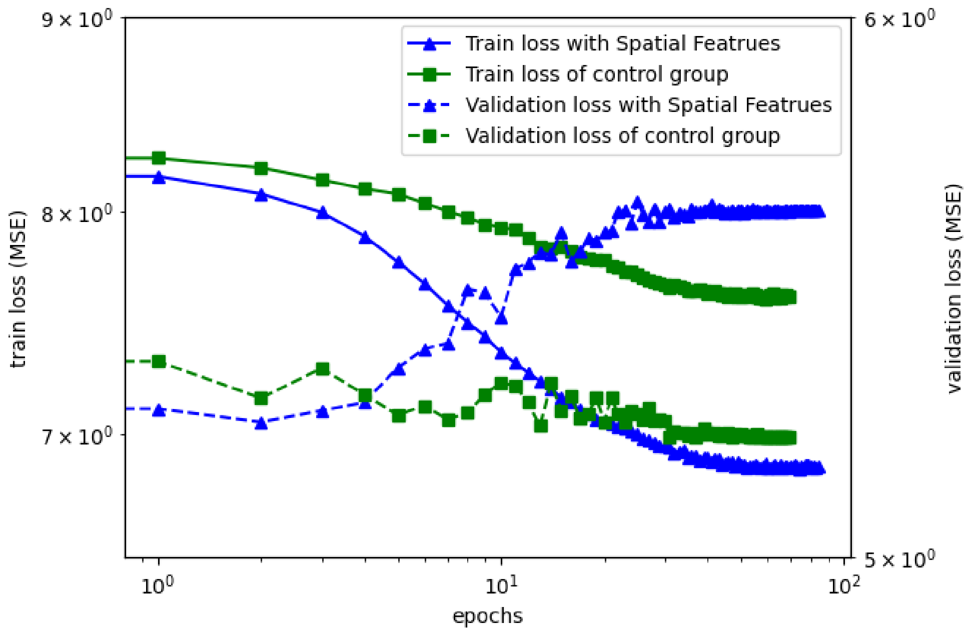


FIGURE 5.7: MSE losses vs. Epochs comparing the pure temporal model with the addition of spatial features

increases in later epochs, where the pure temporal design does not. Any feature change would affect the choice of hyperparameters, hence the process that has been done towards the pure temporal model should be done again.

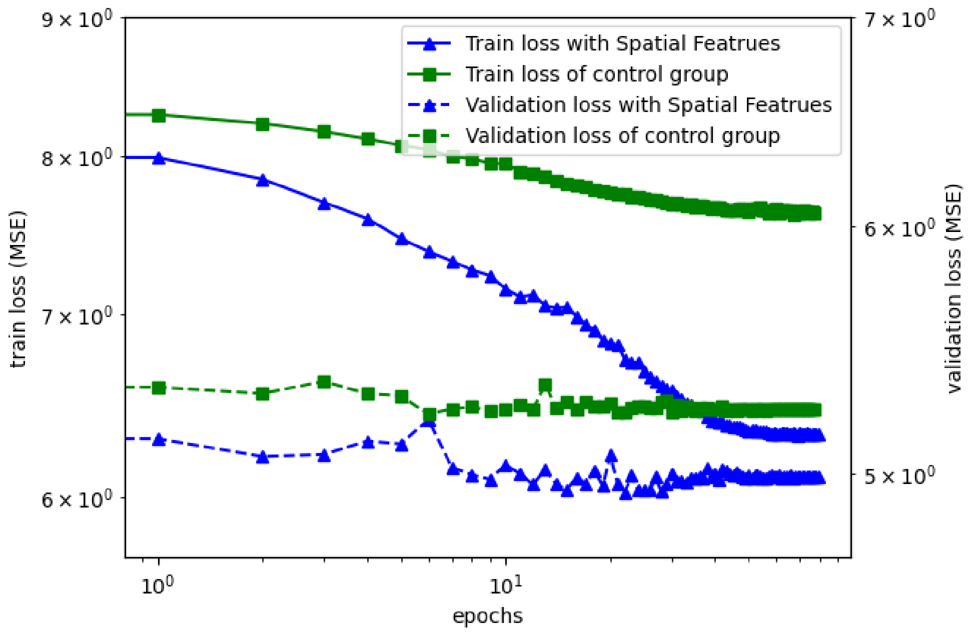


FIGURE 5.8: MSE losses vs. Epochs comparing the pure temporal model with the addition of spatial features

By choosing appropriate hyperparameters, namely 200 LSTM units, 0.2 dropout rate, and 120 batch size, the training loss reduced 20% from the control group, without a raise of validation loss. The graph has been plotted in Figure 5.8. This result shows a close relationship between the current speed of a link and the previous values of its upstreams. The new features provide a better representation of the data, making the model easier to learn the patterns and the structure.

Beyond this comparison, the stability of the model is explored. 8 randomly picked links are trained using the above settings, and the MAPE is used instead of MSE. This is because the MAPE focuses on the relative difference, making the loss comparable between different links. Table 5.1 shows the results.

ID	118	36	47	88	114	70	21	74
Train MAPE	72.45	78.88	78.15	78.88	79.55	78.39	77.61	70.22
Val MAPE	76.25	79.05	78.73	79.50	79.83	79.83	78.76	80.98

TABLE 5.1: Trian and validation MAPE loss of 8 randomly picked links

The result does show a good stability of the model. The train loss varies not much in the range between 70 to 80. The difference is caused by the number of upstreams the link has. There are four features representing the speed in the upstreams. For the links only contain one or even no upstreams, such features are wasted. The above discussion has proven that such features have a positive impact on its performance, different links may benefit differently from these features.

Also, based on the analysis of data in Section 3.2.3, each link seems to have a unique distribution of data. Even if the data is similar, the model is more sensitive to certain features, and the importance of these features may differ in different links.

5.2 Globalised Designs

5.2.1 ST-LSTM

Different from the localised designs, the ST-LSTM model minimise the loss by only using two epochs. Every time the loss reaches its minimum at the second epoch. It can be a good sign that the model has adapted well to the training set. The characteristics of data are learned in a short training period. Conversely, it can also be a sign of underfitting, that the model only captures the simple patterns of the data, and cannot further improve.

Instead of giving the whole training set and validation set to the model, those two sets are given by sequence generators. A total of three generators responsible for the training, validation, and test set were used, which means the data were separated before the sequences formed. This allows the training and validation set to separate completely.

By tuning the batch size, the training and test loss stays the same, however, the validation loss increases as the batch size increases. The results are gathered in Table 5.2. The validation loss increase is as expected since a larger batch size introduces more noise and makes the generalisation ability of the model decrease. The training loss, on the other hand, seems to fall into a local minimum and cannot escape. It is a sign of underfitting.

B	120	360	720	1440	5040
Train MSE	1.8701e-04	1.8701e-04	1.8701e-04	1.8701e-04	1.8701e-04
Val MSE	4.7873e-05	1.1068e-04	2.22036e-04	3.2226e-04	3.7096e-04
Test MSE	2.2528e-04	2.2528e-04	2.2528e-04	2.2528e-04	2.2528e-04

TABLE 5.2: Minimum MSE losses vs. Batch sizes of ST-LSTM model

This model can be considered as unsuccessful, the structure of the model designed is not suitable for the problem. To improve its performance, it is reasonable to change the architecture.

5.2.2 GCN-LSTM

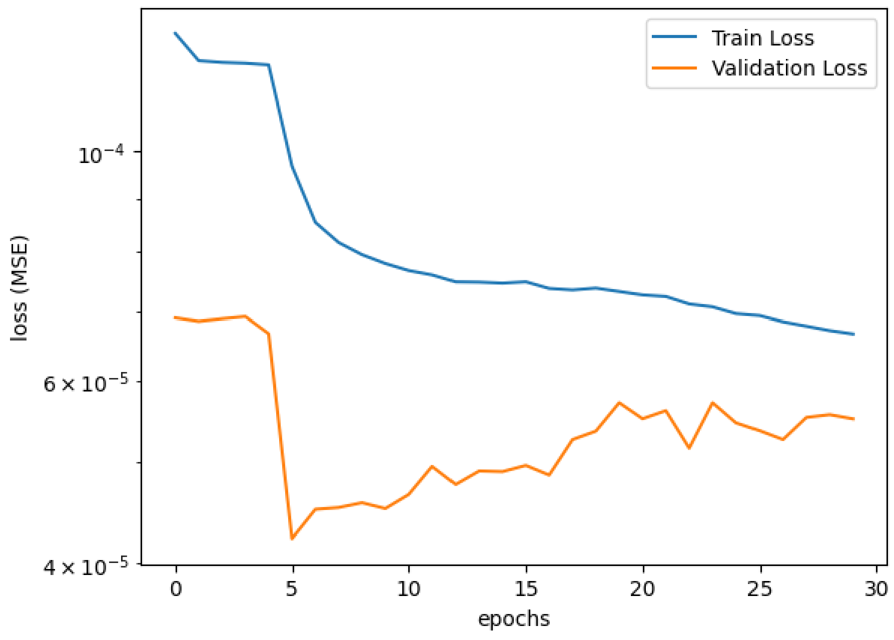


FIGURE 5.9: MSE losses vs. Epochs of GCN-LSTM model

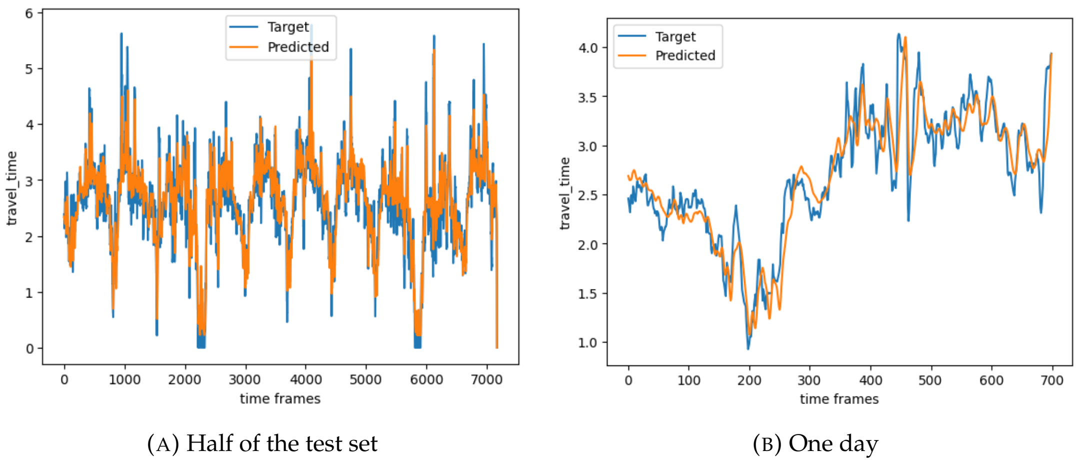
The model easily overcomes the local minimum point and trains further down the gradient. The performance has been improved by using the GCN layer. The path of the losses over epochs is shown in Figure 5.9.

Chapter 6: Evaluation

6.1 Localised Design

The test set is fed into the trained localised model to give a general sense of how well did the model predict. By collecting all of the predictions together, predictions and the actual target values are plotted in Figure 6.1. The graph shows a good prediction in long-term trends of the travel time, yet there are some biases.

In the actual values, there are large fluctuations of values, these rapid changes have not been well predicted by the model. The values predicted are rather conservative, focusing more on the long-term trends, and tend to ignore the high-frequency fluctuations.



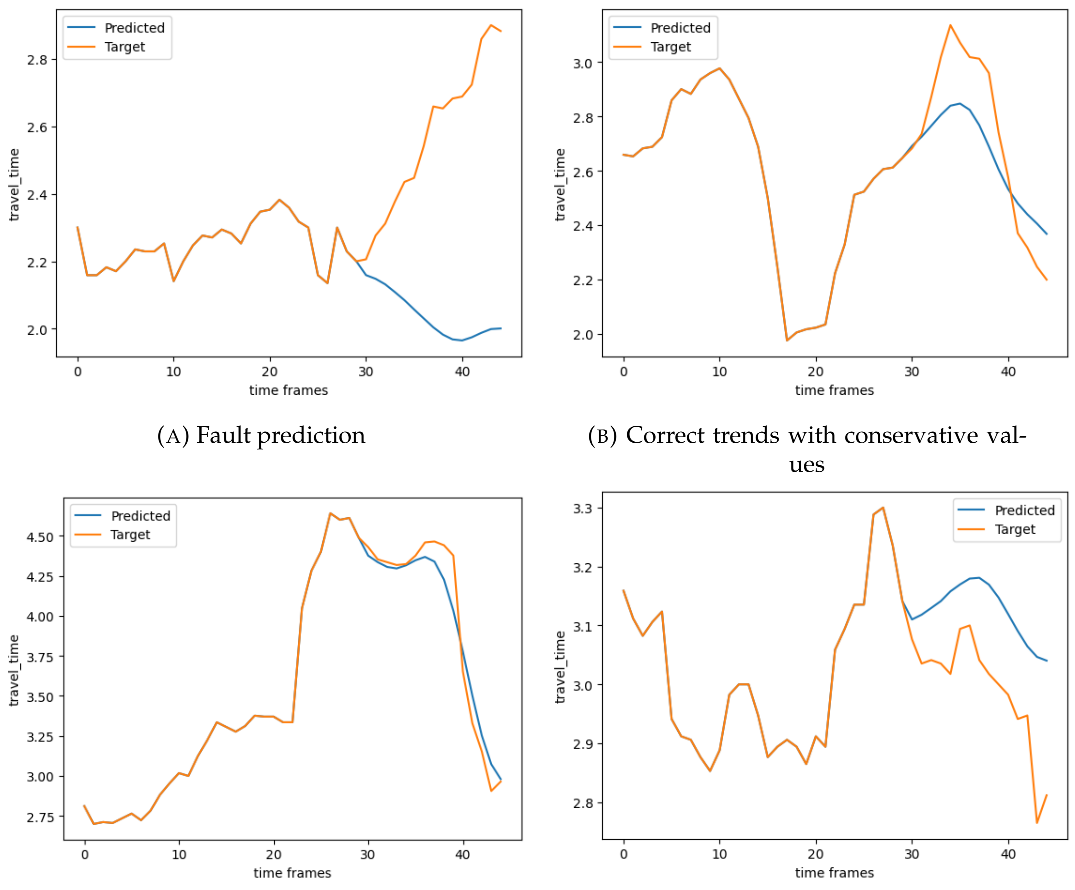
(A) Half of the test set

(B) One day

FIGURE 6.1: Predictions from the localised design joined together compared with the actual values

Due to the fact that the model only predicts the next 15 minutes in advance, such a graph does not tell much about the situation to use the model. Instead, each predictions are plotted individually with the true values. Some prepresentative examples are shown in Figure 6.2. The first 30 data points are the inputs that feed into the model, after that comes the predictions. Generally, the predictions show a smooth curve that has a starting point very close to the last time frame in the inputs. The first few data points predicted are more accurate compared to the later ones.

Figure 6.2a is an example of a prediction that has a trend that completely does not agree with the actual values. This is a fault prediction that happens more frequently when there is a significant increase in travel time in a short period. This may be due to unaccounted-for external factors and unexpected events, such as accidents. This does affect the applications of the model, like a navigation application. Predictions like this will affect the decision made on selecting roads and the expected time taken calculated.



(A) Fault prediction

(B) Correct trends with conservative values

(C) Correct trends and values

(D) Sudden change in values

FIGURE 6.2: Representative predictions in the test set

The prediction of Figure 6.2b predicts the trend correctly but with a lower amplitude. The LSTM model generally has the nature of smoothness, which predicts noises and sudden changes relatively stably. It implies that the model mainly learns the general trends and cyclical changes in the data, but fails to fully capture the sudden changes in the real values. Also, the model may filter out some of the fluctuations as noise or outliers, which also makes the predicted values relatively smooth.

Figure 6.2d have a similar condition when there is a sudden change. This time the change is given in the input sequence. The prediction shows a correct trend but the values are closer to the value of the last of the input sequence. As long as the the trend is correctly predicted, it will be fine to use since it will not affect much on decision makings.

Figure 6.2c gives an accurate prediction that it correctly predicts the trend and values. This is an ideal situation. These accurate predictions happen more when there is a drop in travel time. This may be because the characteristics of the drop are clearer than a rise in travel time in the dataset. The model successfully captures the pattern and makes good predictions.

6.2 Globalised Design

The predictions of a single link using the GCN-LSTM model are worse compared to the localised design. The benefit of it is that it predicts a total of 132 links together at once. Figure 6.3 shows the one-day predictions of one of the links compared with the targets.

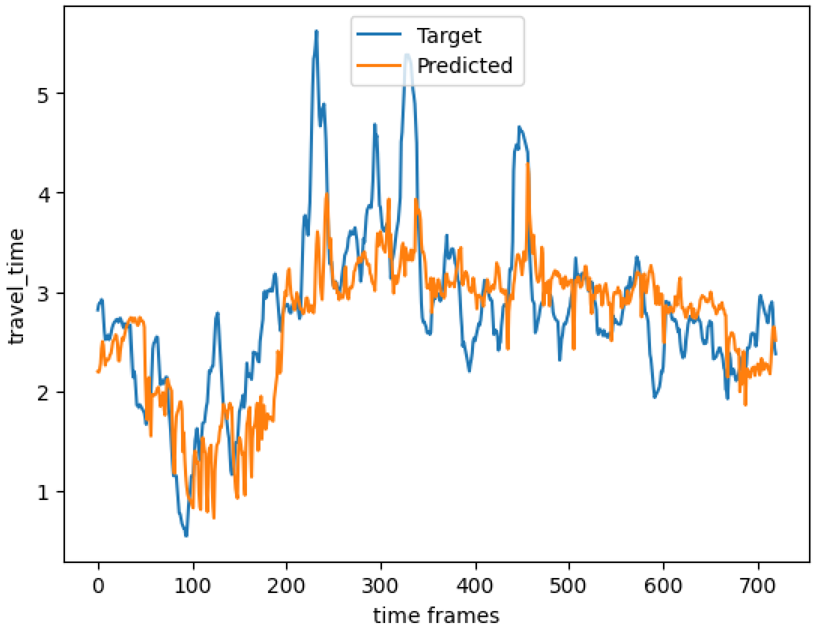


FIGURE 6.3: Predictions of one link from the GCN-LSTM model joined together compared with the actual values

The predictions are still not far from the actual values but are a lot more off than the localised ones. The predictions seem lagging behind, it needs some time to adapt the jumps of the actual values. This shows that the model does not have a good ability to predict the sudden changes. The LSTM layers are not able to handle all links at the same time to give the same level of accuracy as for only one link.

The current performance of the GCN-LSTM model with inputs used is not ideal to be used in applications that require accurate predictions in a short period of time. There are many more parameters and selections that could be made to optimise the performance of the model which have not been covered in the project.

Chapter 7: Conclusion & Proposal

7.1 Conclusion

Both localised and globalised designs showed their ability to predict future travel times based on historical data. The localised one gives a more accurate prediction focusing on a single location, whereas the globalised design uses less time and computational power to predict a whole area.

The hyperparameter optimisation such as the batch size, and learning rate do affect the final performance of the model, but not as much as the input features and the overall structures of the model.

Both architectures were designed with high flexibility. Adding or removing features will not break the models. In fact, choosing appropriate features would have a great boost in the performance of models. Different cities and areas may have different information collected, features could be selected based on specific locations. The performance could be enhanced by having a shorter time frame, which allows the model to learn the patterns of shorter periods of time. The information on traffic lights, accidents, and the number of vehicles would be critical to include in the inputs to further improve the performance. The target to predict is the travel time. However, no matter what value is given, including the average speed of vehicles, the number of vehicles on the road, etc., the architectures are expected to work in the same way.

The prediction could be used as input to predict travel times in further future, however, this would give an increasingly worse prediction.

7.2 Project Management

It was a challenge to come up with a plan for this project at the start when every topic and section related to the project required study and further research. Different architectures to be learnt, and libraries for codes to be familiar with. There was no good enough view of the big picture when stating the goals, and hence some of the goals and scopes have changed as progress. Faults, misunderstandings, and better approaches come up while designing and training models and some of them require redoing the designs completely. The plan has been reconsidered every time it happens. The Gantt chart is shown in Figure B.1. The expected and actual progress differ significantly as seen in the Gantt chart.

Some tasks are added to the project halfway through, and also tasks are abandoned. This was a risky move since it would change the plan for every task afterward. The new designs added took longer time to optimise and train than expected, which led to insufficient time to make the pathfinding algorithm based on the outputs of the models. It would be better if it is decided to stick with the plan, which will achieve the goals stated to a higher extent.

Jupyter notebook is used to code with, which enables to change parameters and alter structures without rerunning the whole process. Git is used as version control. It makes backtracking easier. Different libraries were tried and different operating systems were prepared to ensure the platform of designing is working.

The project brief states the issue ”Assuming that an accident has occurred at a certain location, is there a way to accurately anticipate the resulting traffic conditions on the adjacent roads?” However, due to the fact that a traffic dataset with information on accidents has not been found, there is no way that predictions are not able to reflect accidents.

7.3 Future Work

- Another dataset could be used to train the model. This would give an indication of the universality of the architecture. The ones containing information about accidents and traffic lights are preferred, due to they could potentially enhance performance.
- The hyperparameters require a great deal of time and effort to optimise, tools such as Bayesian optimisation, grid search, etc. could be used to automate the optimisation process.
- Different lengths of the input and output sequences could be considered. The effect of amount of historical data given to the model would potentially affect the performance of the model.
- The structure of the input sequence could be optimised to include some features of the target time frames. Most of the features such as the date, time, and holidays are known values, and including them would result in a better performance.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. springer, 2006.

[2] Saul Dobilas. Convolutional neural networks explained — how to successfully classify images in python, 2022. URL https://towardsdatascience.com/convolutional-neural-networks-explained-how-to-successfully-classify-images-in-python-df829d4ba761.

[3] A. Geron. Hands-on machine learning with scikit-learn, keras and tensorflow, 2019. URL https://www.vlebooks.com/Product/Index/1609613.

[4] ROBERT HECHT-NIELSEN. Iii.3 - theory of the backpropagation neural network**based on "nonindent" by robert hecht-nielsen, which appeared in proceedings of the international joint conference on neural networks 1, 593–611, june 1989. © 1989 ieee. In Harry Wechsler, editor, *Neural Networks for Perception*, pages 65–93. Academic Press, 1992. ISBN 978-0-12-741252-8. doi: https://doi.org/10.1016/B978-0-12-741252-8.50010-8. URL https://www.sciencedirect.com/science/article/pii/B9780127412528500108.

[5] R.J. Hyndman. *Forecasting: principles and practice, 2nd edition*. OTexts: Melbourne, Australia, 2012.

[6] Thomas Kipf. Graph convolutional networks, 2016. URL https://tkipf.github.io/graph-convolutional-networks/.

[7] Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ArXiv*, abs/1609.02907, 2016. URL https://api.semanticscholar.org/CorpusID:3144218.

[8] Ozan Kocadağlı and Barış Aşıkgil. Nonlinear time series forecasting with bayesian neural networks. *Expert Systems with Applications*, 41:6596–6610, 11 2014. doi: 10.1016/j.eswa.2014.04.035.

[9] Jakub Kvita. Visualizations of rnn units diagrams of rnn unrolling, lstm and gru., 2016. URL https://kvitajakub.github.io/2016/04/14/rnn-diagrams/.

[10] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[11] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning, 2015.

[12] Kevin P Murphy. *Machine Learning: A Probabilistic Perspective*. MIT press Cambridge, 2012.

[13] Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999. doi: 10.1017/S0962492900002919.

[14] Tomaso A. Poggio, Hrushikesh Narhar Mhaskar, Lorenzo Rosasco, Brando Miranda, and Qianli Liao. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review. *International Journal of Automation and Computing*, 14:503 – 519, 2016. URL https://api.semanticscholar.org/CorpusID:15562587.

[15] Antônio H. Ribeiro, Koen Tiels, Luis A. Aguirre, and Thomas B. Schön. Beyond exploding and vanishing gradients: analysing rnn training using attractors and smoothness, 2020.

[16] Claudio Filipi Gonçalves Dos Santos and João Paulo Papa. Avoiding overfitting: A survey on regularization methods for convolutional neural networks. *ACM Computing Surveys*, 54(10s):1–25, January 2022. ISSN 1557-7341. doi: 10.1145/3510413. URL http://dx.doi.org/10.1145/3510413.

[17] Hochreiter Sepp and Schmidhuber Jürgen. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

[18] Ryszard Tadeusiewicz. Sieci neuronowe. *Akademicka Oficyna Wydawnicza RM*, 110:236, 1993. URL https://www.academia.edu/30816962/Sieci_Neuronowe.

[19] Tianchi. Traffic prediction dataset, 2018. URL https://tianchi.aliyun.com/dataset/dataDetail?dataId=1079.

[20] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, IJCAI-2018. International Joint Conferences on Artificial Intelligence Organization, July 2018. doi: 10.24963/ijcai.2018/505. URL http://dx.doi.org/10.24963/ijcai.2018/505.

Appendix A: Project Brief

Project Brief

Project Title: Traffic Prediction via Deep Learning

Student Name*: Yanzhe (Solomon) Zhang* ***Supervisor Name****: Lie-Liang Yang*

Problem:

In our daily lives, we frequently encounter the challenge of traffic congestion. In such scenarios, it would be highly advantageous if we could reliably foresee traffic conditions and consequently make informed choices about the most optimal routes for our journeys. With this motivation, this project aims to delve into the realm of traffic prediction using deep learning techniques. Our particular focus is on a specific issue: assuming that an accident has occurred at a certain location, is there a way to accurately anticipate the resulting traffic conditions on the adjacent roads?

Goal:

Develop a model that could predict future traffic conditions of a map, based on historical data and environmental factors. The accuracy is planned to be evaluated. Build a pathfinding program that can use the predicted results to dynamically update route information in real-time.

Scope:

The model is expected to predict both traffic flow and speed using RNN-based method. It needs to respond to both conditions with and without an accident. Weather, time, and temperature may be considered as environmental factors. The pathfinding algorithm is expected to be simple since it is not the main focus. The dataset used would be found on the Internet.

Appendix B: Tables

Tools	Version
CPU	13th Gen Intel®Core^{TM} i7-13700KF $\times$ 24
GPU	NVIDIA GeForce RTX 4070 Ti
GPU Driver	NVIDIA UNIX x86_64 Kernel Module 535.171.04
CUDA	12.2
cuDNN	8.9
Operating System	Ubuntu 22.04.4 LTS 64-bits
Clang	16.0.0
Python	3.10.12
Tensorflow	2.15.0.post1
numpy	1.24.4

TABLE B.1: Development tools and version requirements

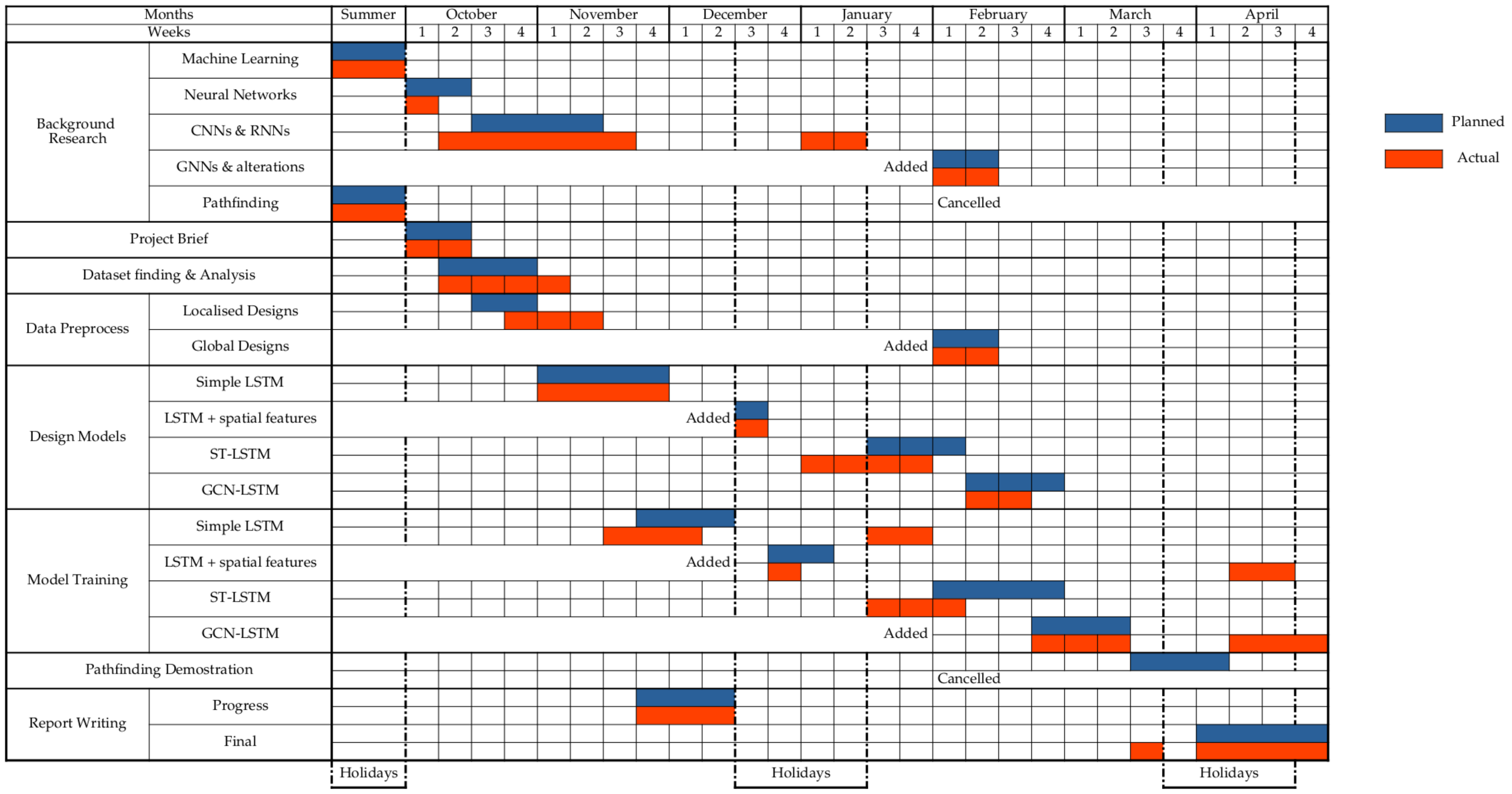


FIGURE B.1: Gantt Chart with planned progress and actual progress