Graph Convolutional LSTM Model for Traffic Delay Prediction with Uncertainty

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of

Master of Science (Research) in Statistics

at

The University of Waikato

by

Dale Townsend

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2021

Abstract

Traffic flow in an urban environment exhibits a complex spatio-temporal interaction. The propogation of traffic flow through a transportation network depends on a number of factors, including the structure of the network and the time of day. Current analysis of this data by road controlling authorities is often simplified and lacks a detailed understanding of how traffic moves through the network. A deep learning model which models both the spatial and temporal interactions present in the data is able to capture complex patterns present in the data and allows for a more detailed understanding of traffic flow. A GC-LSTM model is explored for Hamilton City to predict traffic delay. It is found to have improved prediction accuracy over a standard LSTM by incorporating the spatial structure of the Hamilton road network. Additionally, Bayesian layers are integrated into the model to obtain a probability distribution over each prediction. By quantifying the uncertainty over each prediction, the decision making process based on the analysis can be carried out with a higher degree of confidence than a single point prediction from the model.

Acknowledgements

I would like to thank my supervisor Chaitanya Joshi for guidance in producing this dissertation. His expertise and motivation was instrumental throughout my research.

Thank you to staff in the City Transportation Unit at Hamilton City Council for your knowledge and expertise. A particular thanks to John Kinghorn for guidance in the uses cases of the model.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Hamilton City Council (HCC) has placed growing importance in the use of data in its Infrastructure Operations strategy - 'Data is turned into visible information that is openly available to our community to unlock opportunities for fast growth, efficiency and optimisation' (Hamilton City Council, internal document). As a result, there has been a significant increase in the collection of transportation data and consequently a need for efficient and accurate analysis of the data. The HCC data warehouse will continue to grow over the next year with the addition of datasets from the CCTV camera network, micro-mobility and pedestrian monitoring, and cellular data. With the analysis of this data comes the need to quantify uncertainty in the resulting models. The results of data analysis in local government have a significant influence on investment in infrastructure. The ability to understand models from a probabilistic perspective is valuable in both justifying the spend and better targeting that infrastructure to give a better outcome to the community. Bayesian deep learning models have the ability to provide this for transportation data analysis through the construction of credibility intervals over model parameters. The deep learning framework allows for large datasets with thousands of parameters, and Bayesian layers in the network construct probability distributions

over each output to quantify the prediction uncertainty. In the event of predictions with wide credible intervals, less weighting can be given to the particular prediction over those with a narrower interval.

Given the vast amount of transportation data collected by HCC, there is a growing need to apply novel techniques for handling big data. Much of the data is collected from thousands of sensors in the transport network, often recording data at small time intervals. It is often the case in these scenarios that the data is not utilised to its full potential, which is realised through the use of detailed analytics and machine learning models. Analysis of the city's traffic data shows that Hamilton is undergoing a change in travel patterns and increasing congestion. Inter-peak traffic volumes, defined as the hours outside of the morning and evening peaks, are increasing year on year and resulting in higher delays in certain areas of the city [13]. As the population of the city grows - 2.4% to 176,000 in 2020 [14] - traffic in the school peak continues to worsen. The ability to understand the spatio-temporal interaction of traffic will allow HCC to understand how people travel at different times of the day and how traffic on a street impacts the traffic nearby. When planning changes to infrastructure this information is vital for correct implementation. For example, traffic signal optimisation is a complex process which involves a deep understanding of where vehicles originated to use a certain road corridor, and where they then depart to after using it. When optimising corridors with multiple sets of signals, it is helpful to know the degree to which delay at one set of signals affects the next downstream set, which then has a branching effect to nearby sites and streets. This origin-destination (O-D) and travel time data informs the timings of each approach to an intersection in order to minimise the travel time on main routes.

In recent years, travel delay in Hamilton has increased across both peak and off-peak times [15] Note: 2020/21 result is 49%, to be published later in the

year]. Intersection Level of Service, as defined by the HCM guidelines [16] show that several intersections are operating at a poor level of service. Figure 1.1 (page 3) shows the average delay in seconds for streets in central Hamilton during the morning peak. High levels of delay up to 120 seconds can be seen on inbound links to the CBD, caused by an increase in traffic volume and a high number of trips ending in the CBD. The reason for a poor level of service at these intersections consists of several factors, which cannot be fully understood from analysis of the data without a spatio-temporal context. By viewing inbound links to an intersection and how delay propagates from their upstream links, delay can be analysed spatially and the concentration of delay across time in the peaks can be found.

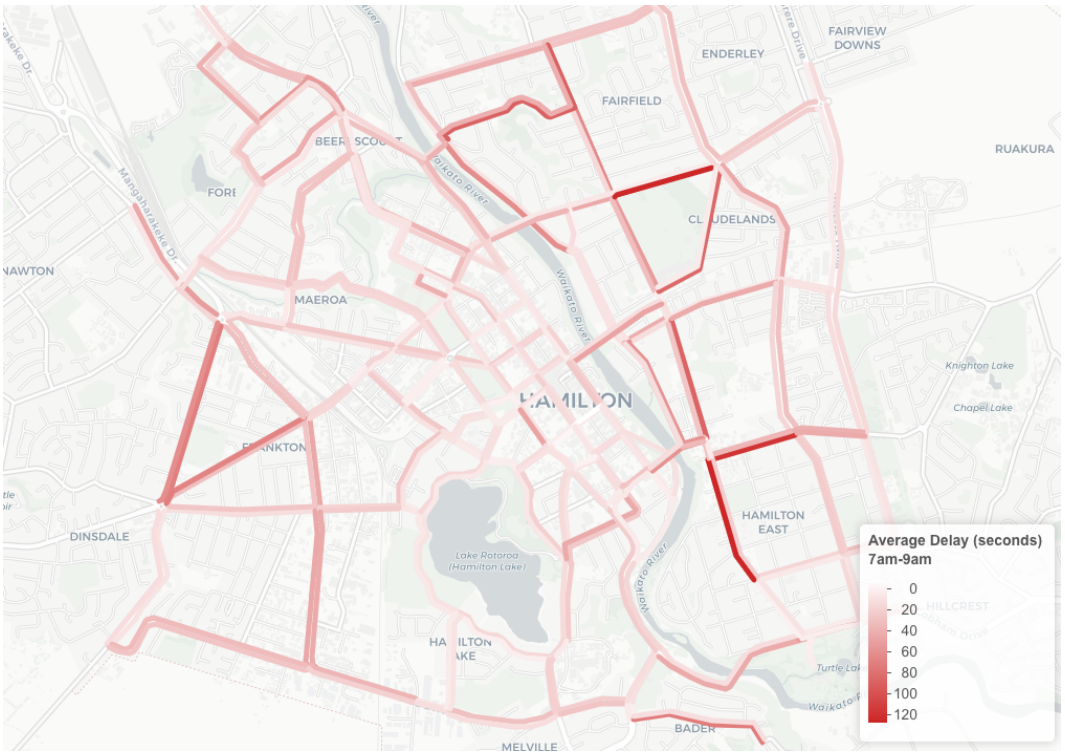


Figure 1.1: Traffic delay in central Hamilton from 7am to 9am

A graph convolution LSTM model (GS-LSTM) is explored in this thesis to model traffic delay data in a spatio-temporal manner. This model can be generalised for a number of applications and current issues in transportation analysis. One is in predicting delay on the transport network given historical data and observed delay in the surrounding area. It is often the case that

unexpected delay on a given street will have a significant flow on effect to nearby streets, as traffic is pushed onto parts of the network that operate on a previously expected level of traffic. Small increases in volume on these areas of the network can have significant effects on travel delay, which then propagates downstream through the network. The ability to use spatial and temporal information allows for detailed analysis of these effects when unusual delay is encountered, and therefore a better understanding of how traffic flows in a dense urban environment. A second application is in the analysis of road closures and prediction of their effects. By analysing the change in traffic flows in the immediate area of a closed road, transport planners are able to better understand how a change in one part of the network affects nearby streets.

A GC-LSTM model may also model variances in mode usage across the network, known as *mode share*. The temporal aspect of the model is able to capture long term trends in the proportion of usage by each mode, while the spatial component utilises the inherent graph structure of the transport network to capture shifts in mode proportion between areas of the city. The ability to calculate and understand mode shift is a key objective by HCC as defined in the 2021-31 long-term plan (LTP) [17]. By capturing the correct data and analysing it in a spatio-temporal manner, insights can be gathered to view patterns and trends in mode share as they implement projects to encourage a shift in mode usage towards bikes, walking, and public transport.

1.2 Background

Google Maps is a widely used tool by councils to analyse delay on the network, provide alerts to high delay, and predict future delay on the network. However, the alerts provided by Google Maps are only for high levels of delay, not 'unusual' delay. Another Intelligent Transport System (ITS) deployed in Hamilton is AddInsight. AddInsight creates a profile of 'expected' delay for

each road, given the observed historical data. This allows for alerts to unusual delay to transport operators, where the traffic is not just high in the peak periods but above what is expected in that period [18]. AddInsight can be installed at a much smaller cost and adds several tools designed for analysing the road network. One such tool is origin-destination analysis, which allows for viewing the routes taken by vehicles and a deeper understanding of how people choose to travel around the network. As a result, several city councils in New Zealand have either deployed or are planning to deploy AddInsight over the next year. This provides further benefits in seeing trips between these cities, as well as trips within cities.

INLA (Integrated Nested Laplace Approximation) is a model for Bayesian inference. It is an established alternative to methods such as Markov Chain Monte Carlo (MCMC) due to its speed and ability to model a wide range of complex problems [19]. It allows for an adjacency matrix to be defined to account for the spatial interaction in the data. INLA has previously been used for predicting traffic volume at signalised intersections in Hamilton City [20]. However, the ability for neural network models to learn is a valuable feature which can be used to understand changes in traffic flow given the changes seen in other parts of the city. While INLA was seen to accurately predict traffic volume and make use of the spatial structure in the road network, it is more difficult to generalise this to 'learn' how changes in one part of the network then influence the observed traffic flow in other areas. Another issue is in modelling the graph structure of the traffic network in a directional manner. While INLA allows for an adjacency matrix to be used in analysing spatial dependence, this adjacency is bi-directional and therefore does not capture the upstream and downstream nature of links in the traffic network, as well as the variance in time dependency and the distance of the influencing link. Upstream links have influence on their downstream links, but have very little effect on its own upstream links. Approximate Bayesian Computation (ABC) has also

been used in Hamilton City to model pedestrian flows in the CBD. ABC is a simulation based model which is able to bypass the likelihood function necessary for Bayesian models such as MCMC [21]. When comparing observed data with simulated data, a rejection algorithm is used with an appropriate distance function to accept or reject parameter values to the posterior distribution. An ABC model was used to predict pedestrian movements in the Hamilton CBD. It was found to accurately predict posterior probabilities, although its simulation based approach is computationally expensive for large datasets such as travel time data and an extension to the entire Hamilton City network.

Neural networks are often used in transportation data analysis - object detection in satellite imagery, classification of modes in camera feeds, and time series prediction of traffic volumes. A system was recently deployed in Hamilton to count forms of micro-mobility and pedestrians using an object detection algorithm. The core of this model is a Convolutional Neural Network (CNN) which was found to accurately classify four different forms of micro-mobility, given a sufficient quantity of labelled training data. An issue with the output of such models lies in the prediction uncertainty. Probabilities can be produced for a given prediction, however beyond this there is no ability to understand the parameters which contribute to this uncertainty or the probability distribution. This is what Bayesian models provide - a probability distribution is constructed over each parameter in the model. This allows for a greater understanding of the uncertainty in the model and construction of a credible interval over each output. This is valuable in the decision making process where the investment into infrastructure requires a certain degree of confidence in the analysis. It also allows for models which will not output a prediction under certain conditions [22].

This thesis looks at a Bayesian deep learning approach to modelling the spatio-temporal interaction of traffic flows. Deep learning models are able to learn

complex non-linear features in a dataset by utilising multi-layered architectures. Long-short term memory (LSTM) models in this field are capable of learning long term dependencies in sequenced datasets such as time series. However, predictions from LSTM's do not take into account the spatial interaction between elements of a dataset. Convolutional Neural Networks (CNN's) are able to model such a spatial structure, and are commonly used in image recognition where features of images can be identified and related to particular categories. When analysing traffic flows there is a complex interaction present between the sequence of time series observations on a particular street and the observations on surrounding streets. This interaction also changes based on the time of day, where traffic tends to travel towards key destinations in the morning and away from those destinations in the evening. Therefore, to obtain an accurate estimation of traffic flows on a given street, a model must be able to model both the spatial and temporal patterns present in the data. For a transport network the spatial structure is best defined by a graph model (graph convolutional), in which directionality and connectivity can be defined. This is a natural fit for a transport network, particularly to constrain observed data to discrete edges and nodes on the network rather than over a continuous field [23].

The use of both a graph convolutional model and an LSTM will enable traffic flows to be modelled over defined intervals of space and time, and will develop a greater understanding of how traffic behaves in Hamilton. A Graph-Convolutional LSTM (GC-LSTM) model is used to predict the travel delay on all links in the network, given the prior sequence on a given link and the observed delay on a series of upstream links at n prior time steps, where n is defined based on the link reachability within a given time interval t and a parameter k to limit the receptive field. Bayesian layers are then integrated into the model to quantify the uncertainty in both the weights (parameters) of the model and the model output. This will allow for defining credible intervals

in which we can be confident of the prediction lying within a certain range with $x\%$ probability.

A GC-LSTM model has an extensive number of uses for HCC. During the model training process it constructs influence weights for each upstream link leading to a given street. This allows for viewing the streets which most influence delay on their downstream links, and as a result a better understanding of how traffic moves through the city at different times of the day. Another use case is during events at venues such as FMG Stadium. During the matches at FMG, several roads around the stadium are closed. This results in flow changes in the CBD, which in turn has a flow on effect for traffic outside of the CBD. Understanding these flows in more detail for major events will help with traffic management, signal phasing, and planning of new infrastructure.

1.3 Aims of this thesis

This thesis aims to explore the use of a Bayesian deep learning model to capture the spatial and temporal patterns present in traffic flow. Such a model is applied in predicting the level of traffic delay across Hamilton City, with uncertainty quantified by the addition of Bayesian layers. The three aims of this thesis are as follows:

- Predict delay on collector and arterial roads in Hamilton city using a spatio-temporal deep learning model
- Obtain probability distributions on delay predictions using Bayesian layers in the network
- Explore the prediction accuracy of a GC-LSTM model against an LSTM model and varying levels of receptive fields

1.4 Thesis Structure

This thesis is structured as follows:

2. Deep Learning: An overview of deep learning is given, including hyper-parameters, activation functions and types of models. Case studies are given for CNN and RNN models on traffic delay analysis for Hamilton City. An analysis of hyper-parameter choice is conducted along with interactions between these parameters.

3. Spatio-Temporal Deep Learning for Transportation Networks: An overview is given of spatio-temporal models, graph networks, and the GC-LSTM model used for delay prediction in Hamilton City.

4. Bayesian Inference on Deep Learning: Methods are described for capturing uncertainty in deep learning models. The Bayes by Backprop algorithm is introduced for modelling uncertainty in deep learning models using variational inference.

5. Transportation Data: The structure of a traffic network is described, along with methods for transportation data collection, and an overview of the AddInsight system used to record traffic delay in Hamilton City.

6. Data & Measures: The traffic delay data as input to the model, and measures used to evaluate the model accuracy.

7. Model Data & Results: Model results are given for the GC-LSTM applied to the prediction of delay in Hamilton City, and a comparison is shown to an LSTM model to highlight the improvement in accuracy when spatial dependencies are considered.

8. Summary and Future Work: A summary is given for the GC-LSTM model to predict traffic delay in Hamilton City. Future applications of the model are given with context of current problems facing transportation in Hamilton City.

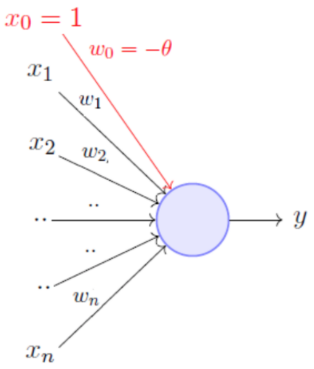
Chapter 2

Deep Learning

Deep Learning is a subset of machine learning which encompasses neural network models 'learning' from large amounts of data [24]. Multiple layers are often used in these models to account for non-linearity in the data. They are able to process a wide variety of data, such as unstructured text and images. In contrast, many machine learning algorithms leverage labelled and structured data in a tabular form. Deep learning models utilise an algorithm known as back-propagation and gradient descent to learn parameters in the model and optimise prediction accuracy on a validation dataset. Below is an overview of deep learning and the hyper-parameters used to train such a model.

2.1 Perceptron

The simple perceptron is a general computational model inspired by biological neurons in the human brain. It consists of one or more inputs, a single layer to aggregate the inputs, and a single input. It is the simplest form of a neural network. Figure 2.1 shows the structure of a simple perceptron.



A more accepted convention,

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

$$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$$

Figure 2.1: The structure of a simple perceptron [1]

A perceptron first multiplies its inputs by a vector of weights, which are then added together to produce a weighted sum. This sum is then passed through a chosen activation function to map the output to a desired range (such as [-1,1] or [0,1]) [2]. An example of this is classifying the gender of a set of people given their weight and height. Here, x^1 is the height of one person, and x^2 is their weight. The simple perceptron will fit a decision boundary (hyperplane) based on these two input vectors, as shown in Figure 2.2:

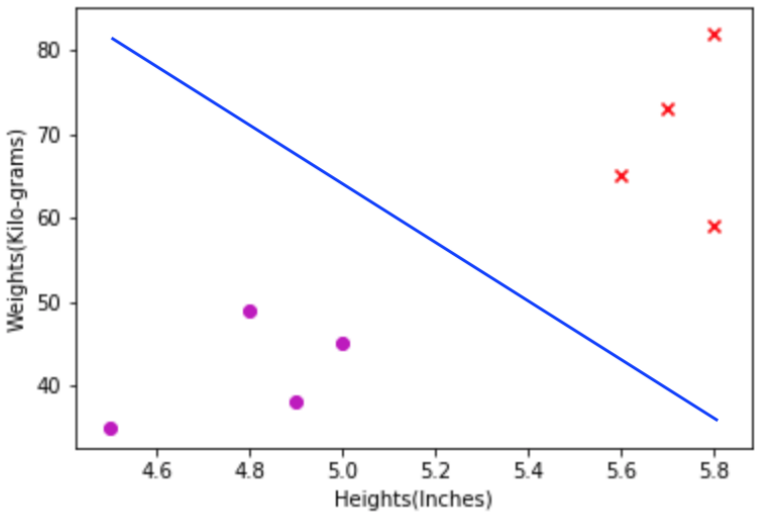


Figure 2.2: The decision boundary between height and weight data points [2]

The hyperplane in a simple perceptron consists of a weight vector $w^1 \in \mathbb{R}^{n \times 1}$,

and a single bias term b that determines the distance of the hyperplane from the origin. Formally, the classification of points as below, on, or above the hyperplane is as follows:

$$Above : -w'^T x' < b \rightarrow w'^T x' + b > 0 \tag{2.1}$$

$$On : -w'^T x' = b \rightarrow w'^T x' + b = 0 \tag{2.2}$$

$$Below : -w'^T x' > b \rightarrow w'^T x' + b < 0 \tag{2.3}$$

The algorithm for the simple perceptron is as follows [14]:

Algorithm 2.1. *1: Initialise with a set of random weights* $w \in \mathbb{R}^{(n+1)\times 1}$

2: For each data point x^i*, calculate a predicted class. If* $w^T x^{(i)} > 0$*, then* $y_p^{(i)} = 1$*, else* 0.

3: Update the weight vector w *as:*

4: *If* $y_p^{(i)} = 0$ *and* $y^{(i)} = 1$, $w = w + x^{(i)}$.

5: *If* $y_p^{(i)} = 1$ *and* $y^{(i)} = 0$, $w = w - x^{(i)}$.

6: *If* $y_p^{(i)} = y^{(i)}$, $w = w$.

7: Repeat from Step 2

8: Stop when all data points have been correctly classified.

The simple perceptron algorithm will only converge if there exists a weight vector w that can linearly separate the two classes. If the true function is non linear, it will fail to converge. In order to construct a deep neural network using such non linear functions, a more complex model known as an MLP *multilayer perceptron* is necessary.

2.2 Multilayer Perceptron

A multilayer perceptron (MLP), also known as a feedforward neural network, forms the basis of many deep learning models. The goal of an MLP is to approximate some function, f. Most commonly, we want to map this function of f and an input x to some output y, in the form $y = f(x)$. The MLP defines

this mapping based on a set of parameters θ, learning the values of θ that best approximates the function:

$$y = f(x; \theta) \tag{2.4}$$

MLP networks are typically represented by composing multiple functions, or layers, together [25]. This layer structure forms the foundation of deep learning models. They are designed to approximate any continuous function and can solve highly non-linear problems. As a result, they are often used in pattern classification, language translation, and image recognition. Through each hidden layer, the values from the previous layer are transformed from the previous layer with a weighted summation $w_1x_1 + w_2x_2 + ... + w_mx_m$. More formally, it is written as:

$$a = \phi(\sum_j w_j x_j + b), \tag{2.5}$$

where w_j is the weights vector, x_j are the input values, b is a bias vector, ϕ is a non-linear activation function, and a is the neuron's activation. An MLP requires the tuning of a number of hyperparameters. This is commonly done by evaluating the model results in the validation set, with the model accuracy being evaluated on the test dataset. An overview of the hyperparameters in a neural network model is given below.

2.3 Hyperparameters

Loss Function

To quantify how close the predictions are to the training labels, a loss function is defined. This is a function used to evaluate a candidate set of weights in relation to the objective of the model. Typically, we seek to minimise the error. Maximum likelihood, or MLE, is one such framework which seeks to find optimum parameter values through which the observed data is most

probable, through the maximisation of a likelihood function. For example, for a Gaussian distribution the maximum likelihood estimate for the mean μ is found by solving the likelihood function with respect to μ [26]:

$$f(x_i; \mu, \sigma^2) = \frac{1}{\mu\sqrt{2\pi}} \exp[-\frac{(x_i - \mu)^2}{2\sigma^2}] \tag{2.6}$$

$$L(\mu, \sigma) = \sigma^{-n}(2\pi)^{-n/2} \exp[-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (x_i - \mu)^2] \tag{2.7}$$

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} X_i = \bar{X} \tag{2.8}$$

Although a neural network model does not directly calculate maximum likelihood, we can define a function to measure the error between the observed data and the model output. One example of a loss function is the number of images correctly classified by the network. In practice, making small changes to the weights and biases won't change this function, making it difficult to know how much to change them. A common loss function is the quadratic cost:

$$MSE = \frac{1}{k} \sum_{i=1}^{k} (y_i - x_i)^2 \tag{2.9}$$

where y_i is the predicted value at i and x_i is the equivalent observed value. This function is commonly used in regression. Cross entropy loss is a more suitable function for classification problems where the set of possible values is more restricted. It more appropriately punishes incorrect classifications and usually results in a higher prediction accuracy. As the predicted probability diverges from the true label, the cross-entropy loss increases. In a binary classification setting it is defined as:

$$L = -\sum_{i=1}^{2} y_i \log(p_i) \tag{2.10}$$

$$= -[y \log(p) + (1 - y) \log(1 - p)], \tag{2.11}$$

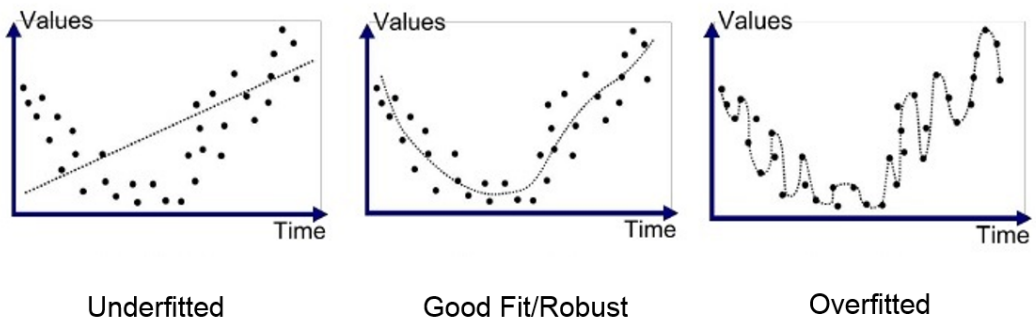
where y_i is the true class in the range [0,1], and p is the predicted Softmax probability for the i^{th} class [27]. Various factors must be considered when

choosing an appropriate loss function. The choice of function is highly context dependent. Regression models typically use the MSE (Mean Squared Error) or MAE (Mean Absolute Error), while classification problems use a score such as the hinge loss or cross-entropy loss.

Epochs

In deep learning, one epoch is a single pass through the entire training set [28]. This includes forward and back propagation so that the model has updated the parameters once. As the number of epochs increases the model fit improves, although overfitting can occur with a large number of epochs. To determine the optimal number of epochs, a small number is often chosen for the first model run. The accuracy reached after each epoch is plotted, with the goal to find a point where the improvements in accuracy become minimal. Further epochs after this point will increase the run time of the model with negligible improvements in prediction accuracy. There is also the risk of overfitting the model, although this can be reduced with techniques such as regularisation. Figure 2.3 shows this effect without the use of regularisation. As with most other hyper-parameters in a deep learning model, the ideal number of epochs is chosen by evaluating the accuracy of the validation set.

Figure 2.3: Curve fitting as the number of epochs decreases [3]



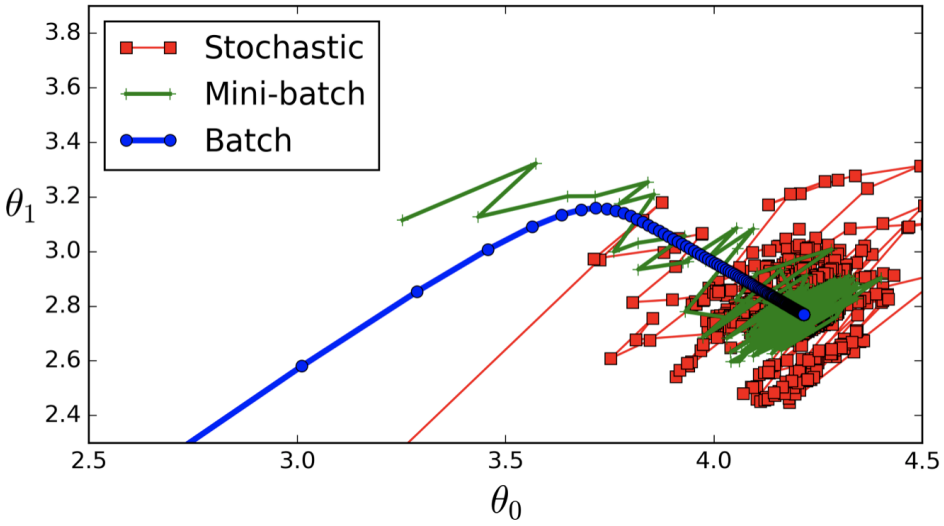
Batch Size

A batch is a subset of the data passed through the network. A full forward and backward pass is made through every batch in the dataset before one epoch is reached. The batch size is the number of training examples used in each pass (batch). When all batches have been passed through the network, one epoch is complete. Using a small batch size will use less memory, which becomes important when working with large datasets. Neural networks will typically train faster with smaller batches, although the network tends to traverse with more variance across the parameter landscape. However, with a sufficient epoch size it will likely converge to the same point as a model using a larger batch size. Using a larger batch size may restrict the model from exploring other areas of the parameter space, although an optimal learning rate can mitigate this. Figure 2.4 below shows the effect of different batch sizes on the exploration of the parameter landscape θ.

There are three main approaches to choosing a batch size [29]:

- Batch Gradient Descent - uses the full training set as one batch
- Stochastic Gradient Descent - each observation in the dataset is a complete batch
- Mini-Batch Gradient Descent - The size of the dataset n is divided into a number of batches b, where $1 < b < n$

Figure 2.4: Traversal of a parameter landscape with different choices of batch size [4]



Learning Rate

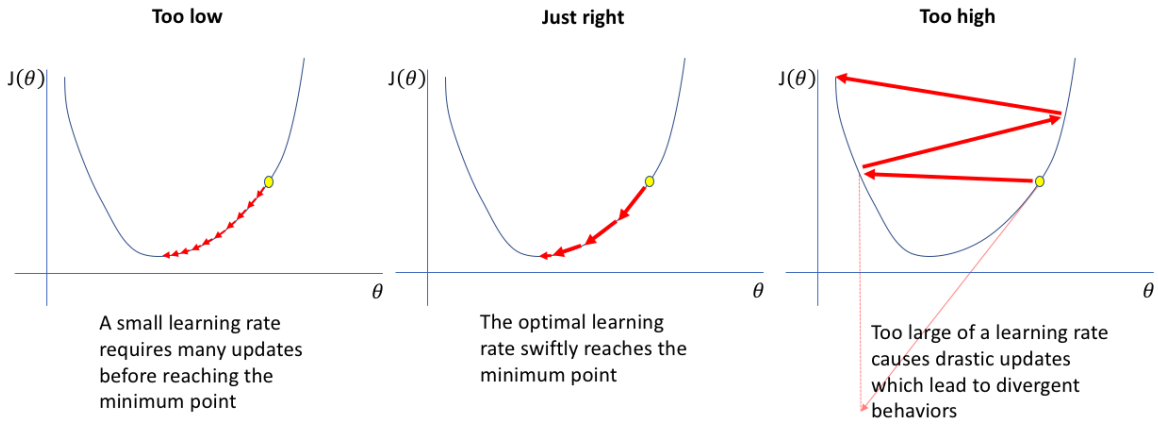
The learning rate is a hyper-parameter used in the gradient descent algorithm. The algorithm estimates the gradient of the error and updates the weights of the model accordingly, with the goal of moving through the parameter landscape to minimise the loss function.

The scale to which the weights are adjusted by is known as the learning rate. It is a small positive value typically between 0 and 1. This choice of value for the learning rate has an impact on both the speed and accuracy of the model convergence. If it is set too low, the model will require more epochs to converge. It may also fail to converge or get 'stuck' in a sub-optimal local minima of the landscape. If it is too large, the model requires fewer epochs but the performance of the model will vary significantly and it may finish at a sub-optimal set of weights [5].

The learning rate for a given model is typically set through trial and error. Different values can be iterated through between 0 and 1 and the accuracy

evaluated on the validation dataset. The default value for many packages in R and Python, including Keras, is 0.01.

Figure 2.5: Traversal of a parameter landscape with different choices of learning rate [5]



2.4 Backpropagation

Backpropagation is an algorithm used in deep learning to optimise the weights in the network. Based on the error rate introduced in the previous epoch, the algorithm proceeds backwards through the network. It calculates the gradient of the loss function with respect to all weights in the network.

Gradient Descent

Gradient Descent is an optimisation algorithm for finding a local minimum of a function. In the context of neural networks, there are often thousands of parameters each contributing to a single loss function. The goal of the gradient descent algorithm is to minimise this loss function, meaning that the error of the model is minimised and a high level of prediction accuracy is attained. To find this minimum, the parameters of the model must be adjusted. Each of this parameters in a neural network contributes to the classification step (the fully connected layer). By using the loss function as a measurement of

the prediction accuracy, the parameters can be subsequently tweaked with the goal of reducing the loss. The Gradient Descent algorithm enables this process to happen in an efficient manner. Figure 2.6 shows this process on a complex parameter landscape. In general terms, the algorithm proceeds as follows:

- For each training example, calculate the gradient of the cost function with respect to every weight and bias parameter
- Calculate the average gradient for all weights and biases
- Update the weights and biases using the updating rule:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{2.12}$$

Stochastic Gradient Descent performs an update of the weights for each mini-batch of n training examples. This generally leads to more stable convergence and allows for efficient use of matrix computation [30].

$$\theta = \theta - \eta * \nabla \theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{2.13}$$

As a first step in the back-propagation algorithm, an equation for the error in the output layer is defined as:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) \tag{2.14}$$

$$\delta^l = \Delta_a C \circ \sigma'(z^L) \tag{2.15}$$

The error δ_l in the next layer is defined as:

$$\delta_l = ((w^{l+1})^T \delta^{l+1}) \circ \sigma'(z^l), \tag{2.16}$$

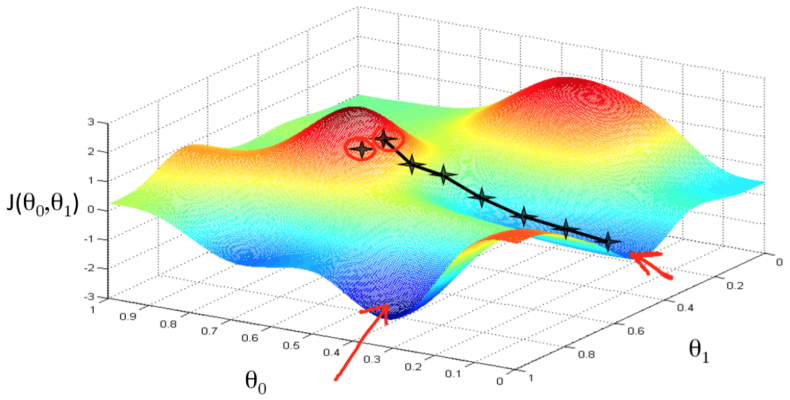
where $(w^{l+1})^T$ is the transpose of the weight matrix w^{l+1} for the $(l+1)^{\text{th}}$ layer. Intuitively, this moves the error back through the network and gives a measure of the output at the l^{th} layer. By using the above two equations, an equation

for the rate of change of the cost, with respect to any weight in the network, can be defined.

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.17}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \tag{2.18}$$

Figure 2.6: A representation of the gradient descent function [6]



Momentum

A technique commonly used in gradient descent is known as Momentum. Here, the gradients of past steps are used as an additional input into the direction and rate of the next step [31].

$$v_j = \eta \times v_j - \alpha \times \Delta_w \sum_1^m L_m(w) \tag{2.19}$$

$$w_j = v_j + w_j \tag{2.20}$$

In the above equation, η retains the history of previous gradients with a time window set by the momentum parameter. When the parameter is zero, η is not used and the algorithm becomes traditional gradient descent. As it is increased, more of the prior gradients are used as input into the algorithm. It

often 'accelerates' the model, allowing for larger jumps across the landscape where appropriate and converging the model in a shorter time span [31].

Activation Functions

An activation function is a mathematical function which determines the output. Each neuron in the network has an activation function attached to it, and decides if that neuron should be 'fired' (activated) given its input. They also normalise the output to a standard range such as 0 to 1 [32].

Because every data point fed into a neural network is passed through a large number of neurons, these activation functions must be computationally efficient. In some cases this can be simply activating the neuron based on a rule or threshold, while in other cases a mathematical transformation is applied. Increasingly these take the form of a non-linear function. This allows for the learning of complex data and more accurate predictions.

There are several such functions commonly used in networks today. The most commonly used functions are described briefly below.

Sigmoid Activation Function

The sigmoid activation function limits the output to a range between 0 and 1, expressed along a sigmoid curve. It is a commonly used function for training a neural network due to its capability to output probability with respect to a given class for a binary classification task. The function's non-linearity allows the model to learn complex features. However, it is only able to output a binary classification, and is computationally expensive. It also suffers from the vanishing gradient problem, where extremely small or large inputs give increasingly small derivatives (changes in the output). This is a common problem with activation functions which map the output to a small range.

$$z = w^T x + b \tag{2.21}$$

$$y = \frac{1}{1 + e^{-z}} \tag{2.22}$$

$$\tag{2.23}$$

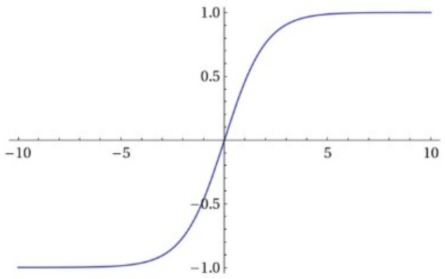


Figure 2.7: Sigmoid Activation Function

SoftMax Activation Function

The SoftMax function is generalised version of the sigmoid function, which is suited for classification tasks with multiple classes. It is the preferred function for the classification (output) layer, and gives predictions between 0 and 1 for each possible class.

If there are k output classes and the weight vector for the ith class is $w^{(i)}$, then the predicted probability for the ith class given the input vector $x \in \mathbb{R}^{nx1}$ is:

$$P(y_i = 1/x) = \frac{e^{w^{(i)T} x + b^{(i)}}}{\sum_{j=1}^{k} e^{w^{(j)T} x + b^{(j)}}} \tag{2.24}$$

Rectified Linear Unit (ReLU) Activation Function

In a ReLU function the output is equal to 0 if the input is less than or equal to zero, or else it is equal to the input. It has multiple desirable properties - it is computationally efficient, eliminates the 'vanishing gradient' problem, and

introduces non linearity. However, when inputs approach zero, the network cannot perform backpropagation due to the gradient of the function becoming zero.

$$y = max(0, w^T x + b) \tag{2.25}$$

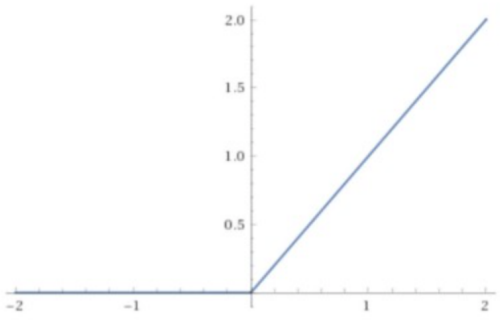


Figure 2.8: ReLU Activation Function

Leaky ReLU

An issue seen in the ReLU function is that over the course of the algorithm input nodes become inactive. The leaky ReLU is a variant of the ReLU function which allows for a small positive gradient when the input is below the threshold, which acts to keep all input nodes alive and increases performance. The prediction results are however not consistent for negative input values.

$$y = \begin{cases} w^T x + b, & \text{if } x > 0 \\ 0.01x, & \text{otherwise} \end{cases}$$

Choice of Activation Function

The choice of activation function for a given model is highly dependent on the use case. Recently, variations of ReLU and Swish have been used in the top performing models in the convolutional layers [33]. Softmax is typically only

used in the final layer due to its transformation of an input vector into a set of probabilities between zero and one for each class.

2.5 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a family of deep learning algorithms commonly used for data with a grid like topology. For example, images are arranged in a 2D grid of pixels. They are based on a technique known as *convolution*. This technique computes a weighted average of data points in order to extract the high level features from the input data [34]. Multiple convolution layers in the network compute different feature sets of the data - for example the first may find edges in an image, the second finds corners and combinations of edges, while the final layers finds higher level features such as faces and objects.

2.5.1 CNN Layers

There are three types of layers in a CNN (Convolutional Neural Network). A description of these layers is given below. Figure 2.9 shows the general structure.

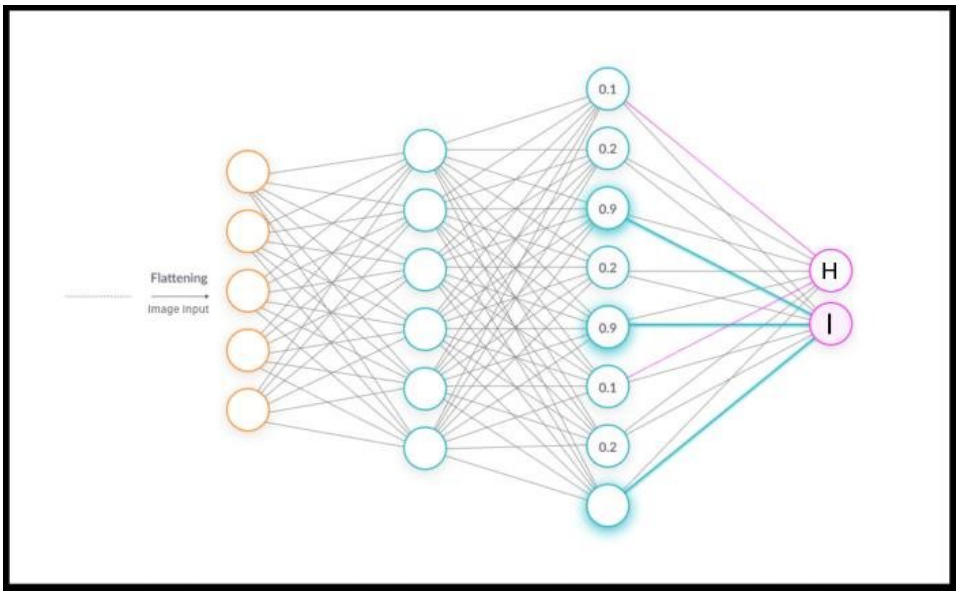
- Convolution Layer - Performs an operation known as a convolution. This is a linear set of operations that multiplies the values of each pixel in the input images by a set of filters (weights). Because the filters are smaller than the the dimension of the input images, a dot product is taken of the array of pixels in the current patch, resulting in a single value for each window. The result is know as a feature map (image). This operation allows the network to detect different types of features anywhere in the image.

- Pooling Layer - Summarises the output of the convolutional layers and retains the most important features. This layer down-samples the feature

maps, commonly using either average or max pooling. Average pooling calculates the average value of each window of the input, while max pooling takes the maximum value. It is typically applied in a 2x2 grid, reducing the dimensionality of the images by 75%.

- Fully Connected Layer - Classifies the images using a non-linear function. The images are flattened into a column vector and fed into the a feed-forward network and back-propagated through each iteration. The chosen activation function is applied to predict the output.

Figure 2.9: Structure of the fully connected layers, showing neuron weights used in determining the output class [7]



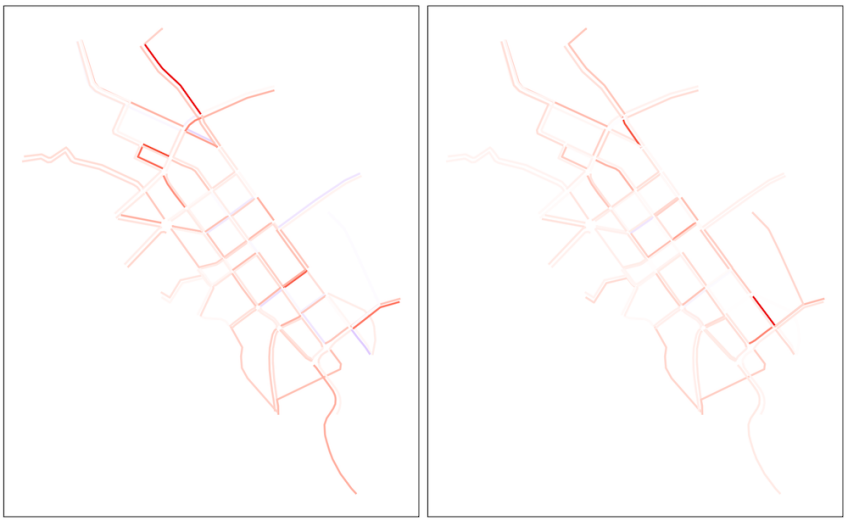
CNN's are widely used in image classification and object detection of traffic modes. Hamilton City has recently deployed micro-mobility and pedestrian counter utilising the MobileNet algorithm, which efficiently detects objects in a video feed and classifies them into the classes of pedestrian, cyclist or scooter. The resulting data is valuable for understanding usage of footpaths, shared paths and cycle lanes, including the overall trend of cyclists in the city and the impact of new infrastructure to encourage these active modes of transportation.

2.5.2 CNN Case Study - Traffic Peak Classification

A CNN model was developed to classify between the two significant traffic peaks in the Hamilton CBD. The AM peak is defined as the period between 7am and 9am, while the PM peak is the period between 4pm and 6pm. Within these peaks, a high level of traffic volume and delay is experienced. In the dataset, delay is calculated for each road and directions of travel, defined as a *link*, within each peak. Data from 2019 to 2020 was used, excluding the Level 3 and 4 COVID-19 lockdown periods. One heatmap for each day and peak was created, resulting in 480 images across 2 classes. The prediction accuracy was evaluated across 48 of these images as the validation dataset.

The model correctly predicted 94% of images from either class correctly. At first glance, it is difficult for the human eye to distinguish between heatmaps of the AM and PM peak. It is often not the case that the reverse flows experience delay in the PM peak. This can be seen at Victoria St at the top of the heatmaps. In the AM peak there is high delay on the southbound approach, cause by high volumes of traffic coming from Fairfield Bridge and stopping at the Mill/Victoria traffic signals. However, the northbound direction does not experience similar delay in the PM peak. This is due to optimisation of traffic signals at Fairfield Bridge allowing more vehicles to turn right, in addition to a large number of vehicles able to continue straight to the Victoria/Te Rapa traffic signals. Due to this, the delay is shifted further north and is not shown in the heatmap. A CNN model is able to capture the areas of the network which consistently experience high delay in each peak, and accurately predicts the peak based on this spatial variation.

Figure 2.10: Delay heatmaps of the AM Peak (left) and PM peak (right) in Hamilton



2.5.3 Hyperparameter Turning

In the training of a neural network, the optimal combination of hyperparameters is often accomplished by trial and error. Two different models may each perform best under two different sets of hyperparameters. This choice is highly dependent on the data. Varying sets of hyperparameter combinations were tested for the traffic peak classification model described above. Note that this optimisation of hyper-parameters is specific to the peak classification model, and results will differ depending on the specific model and dataset used.

Epochs vs Batch Size

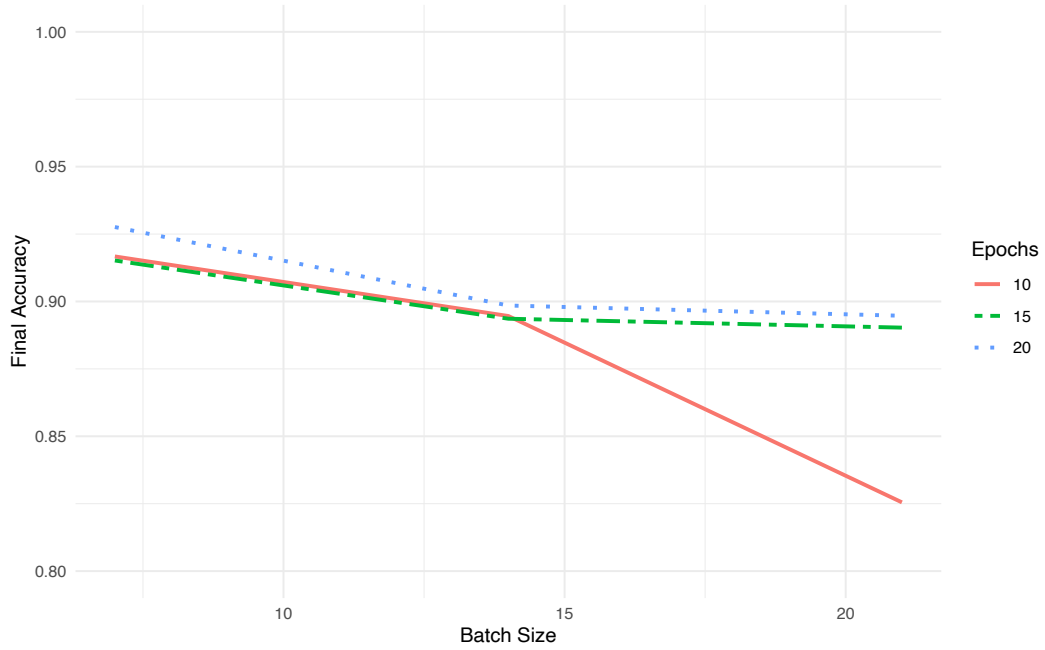


Figure 2.11: Batch Size vs Epochs

As seen in the above plot, 20 epochs performs best across all three batch sizes. Ten epochs performs similarly to 15 when the batch size is set to 7 or 14, but reduces significantly in accuracy at a batch size of 21.

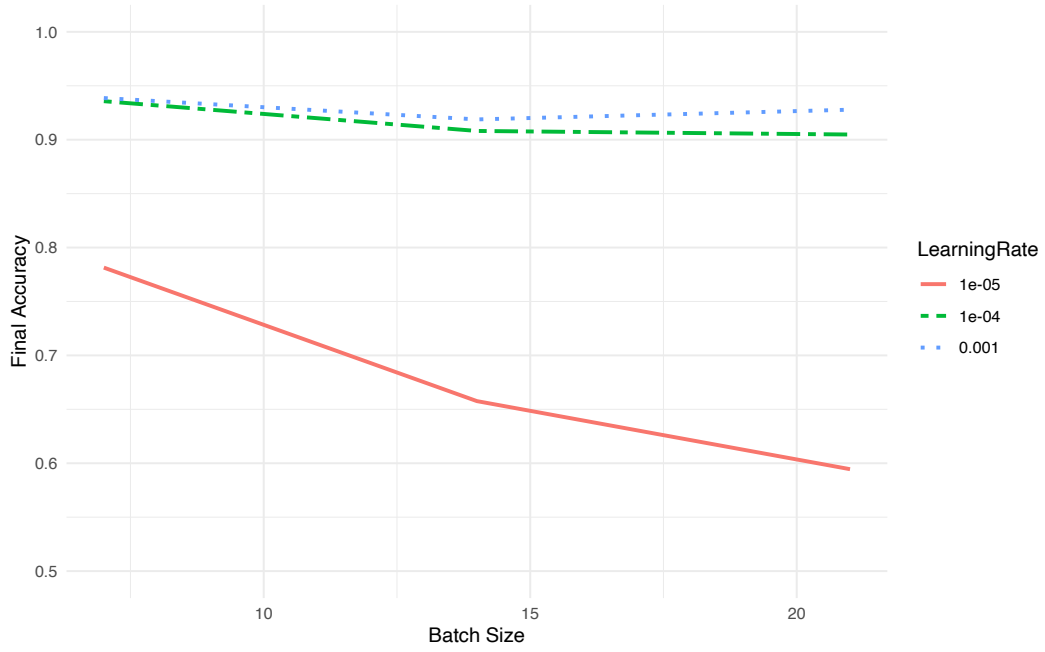


Figure 2.12: Batch Size vs Learning Rate

The number of epochs was fixed to 20 for subsequent models. It is observed that a learning rate of 1e-05 yields a significantly lower accuracy than higher learning rates. The accuracy decreases as the batch size increases. Across batch sizes, the other two learning rates result in similar accuracy. The highest accuracy was achieved with a learning rate of 0.001, but only marginally. The difference in accuracy between the larger two learning rates appears to increase as the batch size increases.

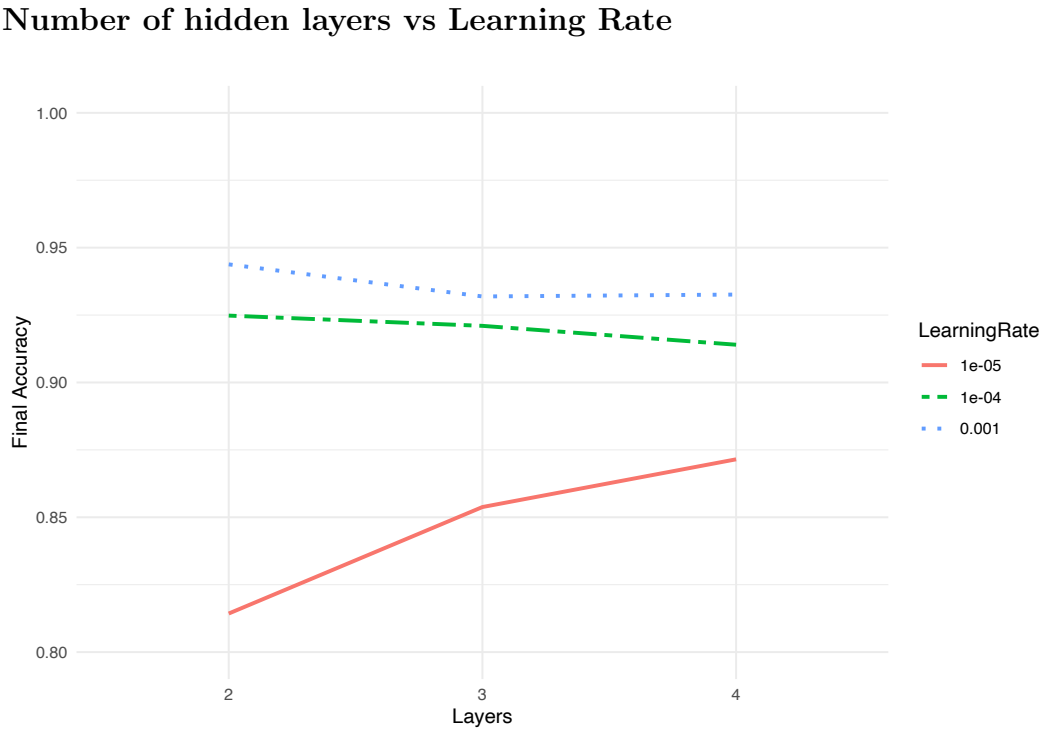


Figure 2.13: Hidden Layers vs Learning Rate

The batch size was fixed to seven for subsequent models. For the smallest learning rate of 1e-05, the accuracy improves significantly as the number of hidden layers increases. There is no significant change in accuracy across layers for the other two learning rates. The learning rate of 0.001 yields the highest accuracy across all iterations.

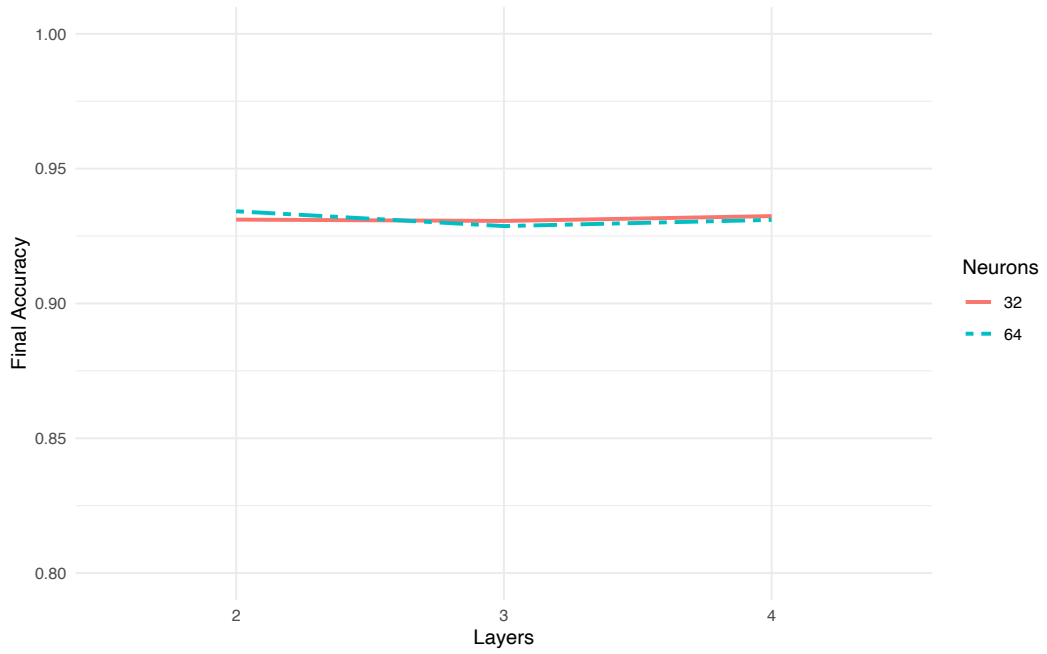


Figure 2.14: Hidden Layers vs Neurons

The learning rate was fixed to 0.001 for subsequent models. There was no significant difference in accuracy across all combinations of layers and neurons.

2.6 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a type of deep learning algorithm suited for a sequence of values, typically in the form $\mathbf{x} = x_1, x_2, ..., x_n$. They are able to use an internal memory state to learn and process sequences of inputs. One of the more common variants is an LSTM - 'Long Short Term Memory'. It is suitable for a variety of sequence based tasks, including handwriting recognition, anomaly detection, and time series analysis. RNN's operate on a sequence of vectors $x^{(t)}$ with a time step index ranging from 1 to T. These sequences are typically batched into smaller sequences.

2.6.1 LSTM Cell

LSTM's are a variant of the RNN model which use a short memory unit with the ability to 'forget' a part of its previously stored memory and add part of the new information from the current input. The cells consists of three main components: an input gate, forget gate, and output gate. This architecture is shown in Figure 2.15.

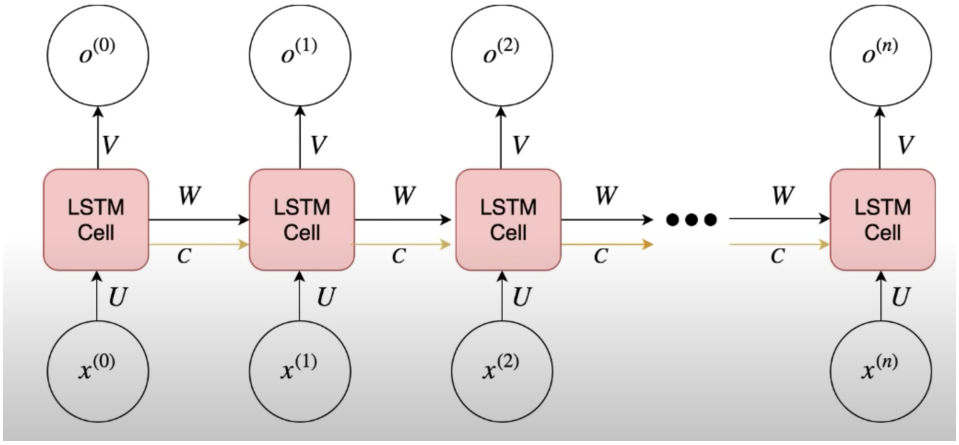


Figure 2.15: LSTM Architecture [8]

Each gate at time step t is calculated as:

$$i^t = \sigma(W^i[h^{(t-1)}, x^t] + b^i \tag{2.26}$$

$$f^t = \sigma(W^f[h^{(t-1)}, x^t] + b^f \tag{2.27}$$

$$\sigma^t = \sigma(W^o[h^{(t-1)}, x^t] + b^o \tag{2.28}$$

Figure 2.16 shows the architecture of a single LSTM cell. The output, h_t, is calculated from the input at previous time step h_{t-1} and the observation x_t at the current time step. These inputs are passed through a set of gates to determine the output, including an activation function tanh.

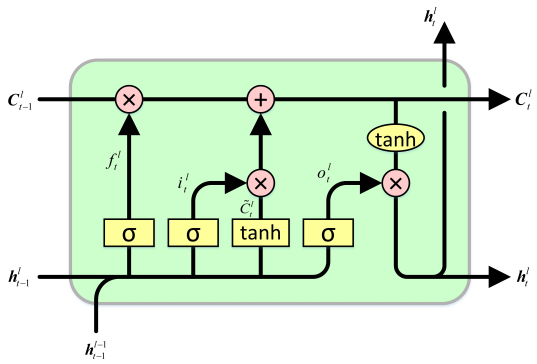


Figure 2.16: An LSTM Cell [8]

2.6.2 GRU Cell

A GRU cell (Figure 2.17) modifies the LSTM cell by using two gates - an Update Gate and a Reset Gate. The result is an architecture which is typically faster to train. However, the accuracy will differ between LSTM and GRU cells depending on the specific use case, and often both are used in the model training process.

$$Z^t = \sigma(W^z[h^{(t-1)}, x^t] \tag{2.29}$$

$$r^t = \sigma(W^r[h^{(t-1)}, x^t] + b^f \tag{2.30}$$

$$\tilde{h}^t = \tanh(W[r^t \times h^{t-1}, x^t]) \tag{2.31}$$

$$h^t = (1 - z^t) \times h^{t-1} + z^t \times \tilde{h}^t \tag{2.32}$$

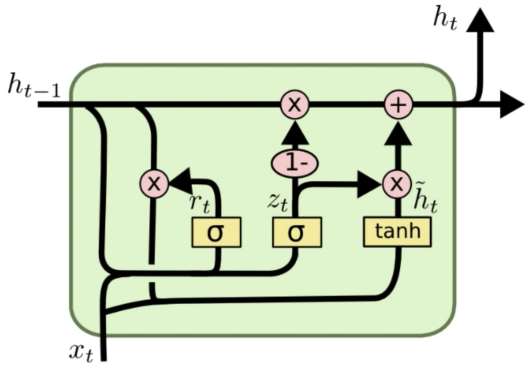


Figure 2.17: A GRU Cell [8]

Backpropagation through time

The Backpropagation through time (BTT) algorithm computes the gradients for updating the weights of the network, in proportion with the derivative of the error at each pass [35]. Using the standard backpropagation process for an RNN results in the exploding and vanishing gradient problems, where the gradients become close to zero and the algorithm stops updating. The hidden state $h_t \in R^h$ and the output $o_t \in R^q$ are calculated as:

$$h_t = W_{hx}x_t + W_{hh}h_{t-1}, \tag{2.33}$$

$$o_t = W_{qh}h_t \tag{2.34}$$

Figure 2.16 below shows the computational graph of the BTT architecture.

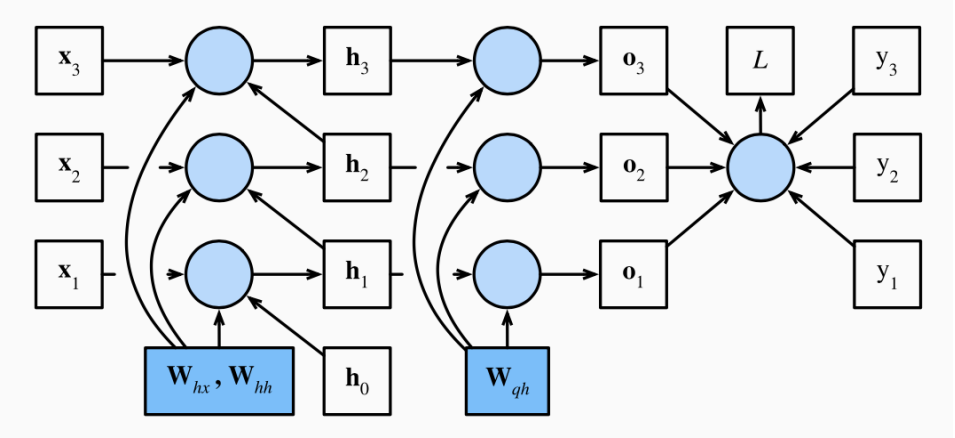


Figure 2.18: BTT computational graph, showing dependencies between model variables and parameters during computation [9].

2.6.3 Case Study - Prediction of Traffic Delay

An LSTM model is used in Python to predict delay on a section of Wairere Drive in Hamilton. One month of data is used from September 2, 2019 to September 27, 2019. The first three weeks of the dataset are used for training, with the following six days for validation and the final day for testing. The data is in fifteen minute intervals. The model is run for 1000 epochs with a

constant learning rate of 1e-5.

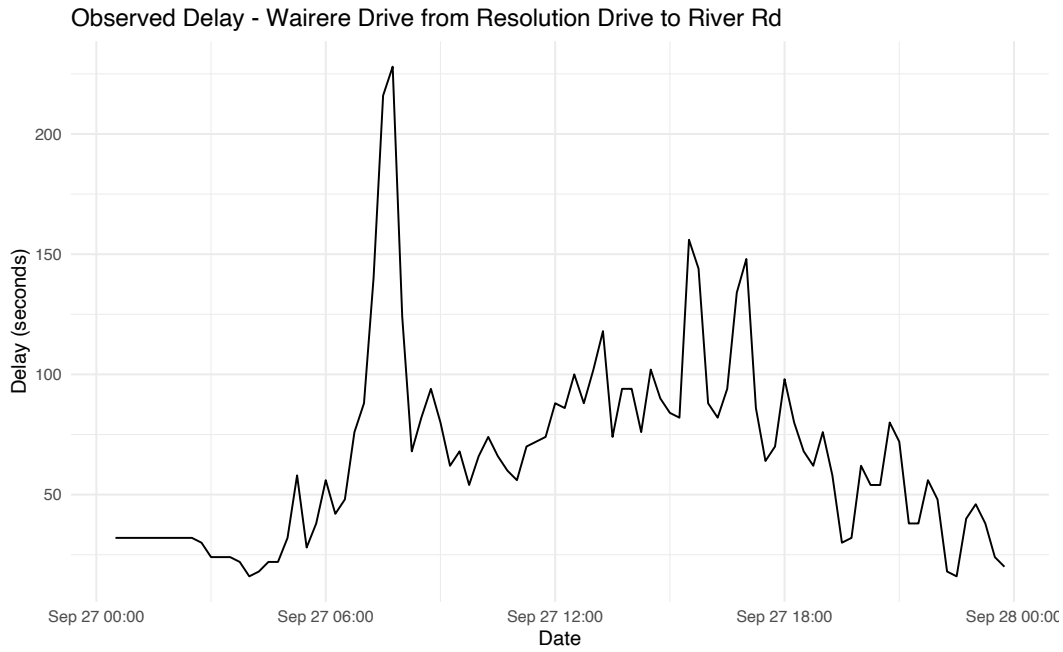


Figure 2.19: Delay on Wairere Drive on September 2, 2019

The delay on the link for Friday, September 27th 2019 is shown in Figure 2.17. It exhibits a sharp morning peak, followed by a constant delay throughout the day, until short peaks in the school and PM periods. This AM peak is generally seen on all weekdays in the data, although there are less occasions of small increases in delay at other times in the day. The LSTM is expected to capture the general pattern of the morning peak, although anomalous changes in delay at other times of the day may not be fully captured without looking at the spatial relationship with surrounding links.

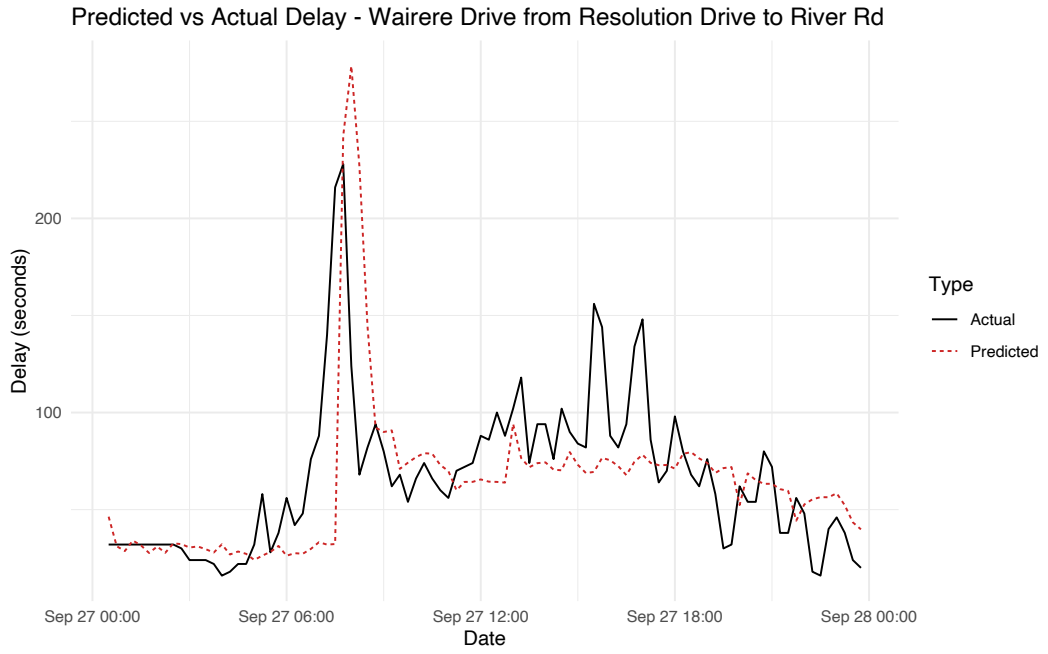


Figure 2.20: Predicted vs Actual Delay for a section of Wairere Drive, using an LSTM model

The MAPE for the model is 4.29%. The delay in the morning peak is mostly captured, although is is predicted slightly later in the day. The data shows an increase in delay in the afternoon and evening, although this is not captured by the model.. Because these peaks were not historically seen in the training data, the model fails to predict them It may have been caused by changes in traffic flow on other parts of the network, such as increased volume or an incident. It is expected that a model accounting for the spatial dependencies will perform better on data such as the above, where delay on the network does not always follow a predictable pattern and flow on effects are seen throughout the network.

2.7 Summary

Deep Learning algorithms are appropriate for modelling transportation data, given their ability to handle large quantities of multi-dimensional data. Variants on the traditional multilayer perceptron allow spatial data to be captured through the use of a convolutional process, as well as sequences of time series observations with a recurrent structure. These models have been successfully deployed in the collection and analysis of data in major cities, such as CNN models in camera systems counting and classifying different modes of transportation across the network.

Chapter 3

Spatio-Temporal Deep Learning for Transportation Networks

3.1 Transportation Data and Analysis

Local road controlling authorities (RCA's) collect large quantities of data to record what is happening on their transport network [36]. Up until recently, the majority of traffic data has been in the form of surveys, pneumatic tubes, or signal detectors. Surveys are typically conducted by setting up cameras or people at at a given location and recording the level of traffic. This is done to give a point in time baseline of traffic flow in the area. The area may be surveyed again following infrastructure changes in the area, to understand the change in traffic flow and measure if a certain goal was achieved, such as a higher number of pedestrians and a lower level of traffic. Pneumatic tubes are another method to collect traffic volumes. They are black tubes deployed across the roadway. By using two tubes at a short distance apart, they are able to measure speed and classification (type of vehicle) in addition to volume. In the case of infrastructure projects which aim to lower speeds or discourage heavy vehicles, tubes are an effective tool to gather such data. Signal detectors are another widely deployed system in cities. They are installed in each lane of a signalised intersection, and primarily act to control the flow of traffic

through the changing of signals. The counts recorded from detectors are sent to a central system to be analysed for trends in traffic volume and variances in volume at each intersection approach, both spatially and temporally.

Recently, there has been an increase in collection of data for other transport modes. This includes counts for pedestrians and cyclists, scooter data through shared micro-mobility providers, and public transport utilisation through card tag on/off systems such as the Bee Card [37]. Having data on every transport mode enables RCA's to analyse spatial and temporal variances in the usage of each mode, and make more informed decisions on projects as a result.

Traditionally, the analysis of transport data by RCA's has been in the form of basic aggregations such as daily totals, year-on-year change, or mode share proportions collected by surveys. Statistical models such as ARIMA forecasting are used on occasion [38], but fail to capture the inherent complexities in the network on both a spatial and temporal manner. While good results can be obtained by such forecasting on traffic volume data, it is unable to show the propagation of volume through a network or how volume on a given road influences the surrounding area. This understanding allows for better decision making upon analysis of the data, and reduces unforeseen events caused by infrastructure changes made as a result of an analysis. It is often the case that a small change to traffic signals at a given intersection can have major flow on effects in the immediate area. This effect may be captured by modelling the data in more complex models which can more accurately capture the network structure.

Given their complexity and computation time, deep learning models have historically not been widely used by local road controlling authorities. Development of such models requires local domain knowledge and expertise in the models themselves. As a result, much of the transportation data collected

in urban environments is not explored in detail beyond KPI calculations and basic aggregations. As traffic sensors become more precise, cheaper to deploy, and expand in scope, there is a growing need to analyse the data in a spatio-temporal manner to better understand how traffic flows in a city. Traditionally, forecasting of traffic volume or delay has been done by either finding historical averages in the dataset or with time series models such as ARIMA [39]. This algorithm is commonly used in business intelligence (BI) software via a 'one-click' approach. This presents dangers where a prediction from the auto analysis is used to make decisions without an understanding of how the algorithm has come to its prediction, or if it is an appropriate algorithm to use.

3.2 Deep Learning Models

Deep learning models have proven to be capable in capturing the highly non-linear spatio-temporal effects in traffic forecasting [40]. Simple feed forward neural networks were first explored for travel time estimation, and have since been extended to several other deep learning based models including fuzzy NN's, recurrent, deep belief networks, auto-encoders and generative adversarial networks (GAN's). Recurrent neural networks in various forms, including LSTM and GRU variants, have been successfully applied to traffic forecasting due to their ability to capture temporal dependencies in the data. They have been used in forecasting traffic speeds, travel time, and volume.

The Structural-RNN [41] identifies that while Recurrent Neural Network (RNN) models are capable of modelling temporal sequence data, they lack a spatio-temporal structure necessary for considering dependencies between space and time. The example given in the paper is in modelling human motion where the next sequence is a complex interaction between the previous spatial state and the sequence of motions leading to that state.Several applications of this

model are in the fields of human motion, video frame and image generation. The common structure used is known as an st-graph. It is represented with $G = (V, E_S, E_T)$, where V is the set of nodes in the network, E_S is the set of the edges, and E_T is the set of edges over time. It is recognised that a vast quantity of data fits into this structure, particularly in modelling human activity and interactions.

DeepTransport [42] explores spatio-temporal prediction in traffic forecasting. An end to end framework is proposed with a combined CNN-RNN approach to obtain spatio-temporal information within a traffic network graph structure. It is noted that traditional approaches to traffic forecasting (ARIMA, Deep Belief Networks and Stack Autoencoders) ignore the spatial relationship present between edges in a transport network. The model takes the approach of an intricate topological graph, identifying a set of upstream and downstream edges which each influence each other over a series of time steps t. The DeepTransport model is relatively intuitive and simple to understand. It achieved good results in traffic condition forecasting and outperforms all other models tested.

In recent years, several novel approaches have been proposed as modifications to the standard LSTM model. These include the bidirectional LSTM, deep LSTM, shared hidden LSTM, and nested LSTM [43]. They are typically created through the restructuring of an LSTM to better capture the complex temporal dependencies present in traffic data. For example, a bidirectional LSTM uses two independent RNN's such that there is both backward and forward information in the sequence at every time step. Many of these models also incorporate additional data such as crashes, geographical attributes, and weather to enhance the prediction performance.

While the LSTM based models perform well in capturing temporal depen-

dencies on a road, modelling the spatial relationships present in the network is necessary for many applications in traffic forecasting. CNN's have been applied to traffic networks to extract spatial features, however the inherent structure of a traffic network means that the resulting images in a CNN model have a high amount of noise. The areas we are interested in modelling are only present in a small proportion of the 2D images, and therefore CNN based models for traffic forecasting often result in spurious spatial relationships. There have been attempts to convert traffic data into three-dimensional matrices to reduce the issues present in 2D models, although they are still unable to deal with the network structure or physical attributes of a traffic network.

A more intuitive approach in traffic forecasting is to learn the network as a graph structure. This is a natural fit for common transport analysis such as shortest path routing, dynamic lane assignment in peak times, and analysis of unusual delay [44]. In deep learning these models have taken the form of graph convolutional networks (GNN's). GNN's typically make use of an adjacency matrix or Laplacian matrix to describe the relationships between road segments. The Laplacian matrix $L_{n\times n}$ is defined as:

$$L = D - A, \tag{3.1}$$

where D is the degree matrix describing the number of edges attached to each node, and A is the adjacency matrix [45]. The elements of L are given by

$$L_{i,j} = \begin{cases} \deg(v_i) & \text{if } i = j \\ -1 & \text{if } i \neq j \text{and } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

It is a symmetric positive semi-definite matrix such that it can be diagonalised via eigen-decomposition:

$$L = U\Delta U^T \tag{3.2}$$

,where Δ is a diagonal matrix containing the eigenvalues, U contains the eigenvectors of the matrix, and U_T is the transpose of U. The spectral convolution on the traffic graph is defined as the multiplication of a signal $x_t \in \mathbb{R}^N$, with a filter $h_\theta = \text{diag}(\theta)$ parameterised by $\theta \in \mathbb{R}^N$ [46]. The spectral graph convolution operation is defined as:

$$h_\theta *_G x_t = U h_\theta U^T x_t = U \text{diag}(\theta) U^T x_t \tag{3.3}$$

The Laplacian matrix provides a way to investigate the connectedness of a graph. Graph convolution models utilising the Laplacian matrix are based on *spectral graph theory*, which is the study of graph properties in relation to its eigenvalues and eigenvectors [47]. One such model extension to spectral graph convolution is localized spectral graph convolution (LSGC), which reduces the learning complexity by using localised convolution operations learned from the data [10]. An LGSC only has K parameters equal to the number of hops from a given node, and does not need eigen-decomposition. Each convolution operation on a centred vertex extracts the summed weighted feature of the vertex's K-hop neighbours.

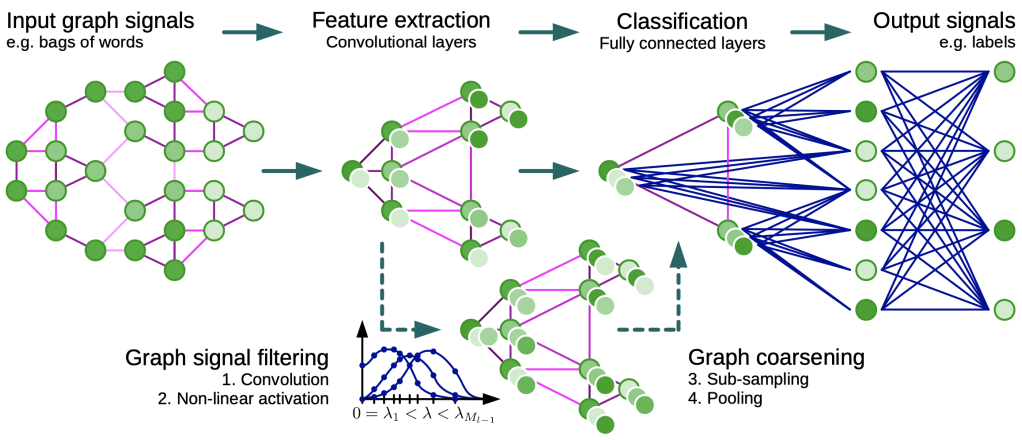


Figure 3.1: Spectral Graph Convolution [10]

A Graph Convolutional LSTM (GC-LSTM) conducts convolution of a datasets' graph structure in order to calculate a weight for a given edge. This weight is then used as an additional input into an LSTM model, enabling prediction based on both the spatial and temporal structure present in the data. The remainder of this chapter is largely based upon the paper [12].

3.3 Traffic Graph

A graph is a mathematical structure describing a set of objects and how they are connected. The objects are known as *nodes*, while the connections between nodes are *edges*. They can be either directed or undirected [48].

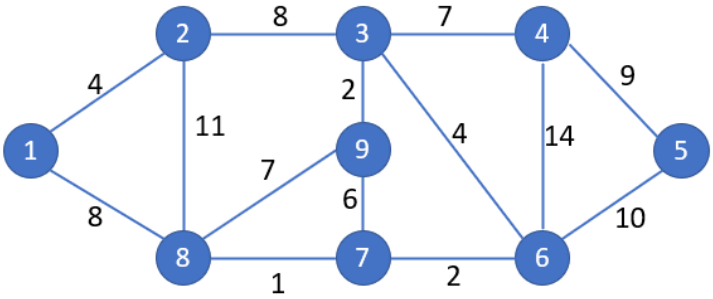


Figure 3.2: A simplified graph, showing nodes as blue circles and edges as lines. The numbers adjacent to lines represent a known characteristic of the network, such as distance or time [11].

A graph representing a transport network is distinct in that there are no isolated nodes, and the network rarely changes in structure. Node or edge attributes may change dynamically in the case of traffic volume or delay, or they may stay relatively static. A transport graph may have meaningful characteristics describing the infrastructure, such as road type, length, posted speed or number of lanes. The GC-LSTM model uses nodes to represent the traffic sensing locations, while edges represent the intersections connecting each node (road segment). It can be represented by an undirected graph $G = V, E$ with

N nodes $v_i \in V$ and edges $(v_i, v_j) \in E$. The connections between nodes are defined with an adjacency matrix $A \in R^{N \times N}$, in which each element $A_{i,j} = 1$ if there is an edge connecting node i and node j. The model uses these connections to find the number of hops between edges in the network. A parameter is set in the model as k, defined as the maximum number of hops to use in the adjacency graph when calculating the spatial weight. This is shown in Figure 3.3 for a link on Wairere Drive in Hamilton. The adjacency matrix for each k from L can be found as $A * A^k$.

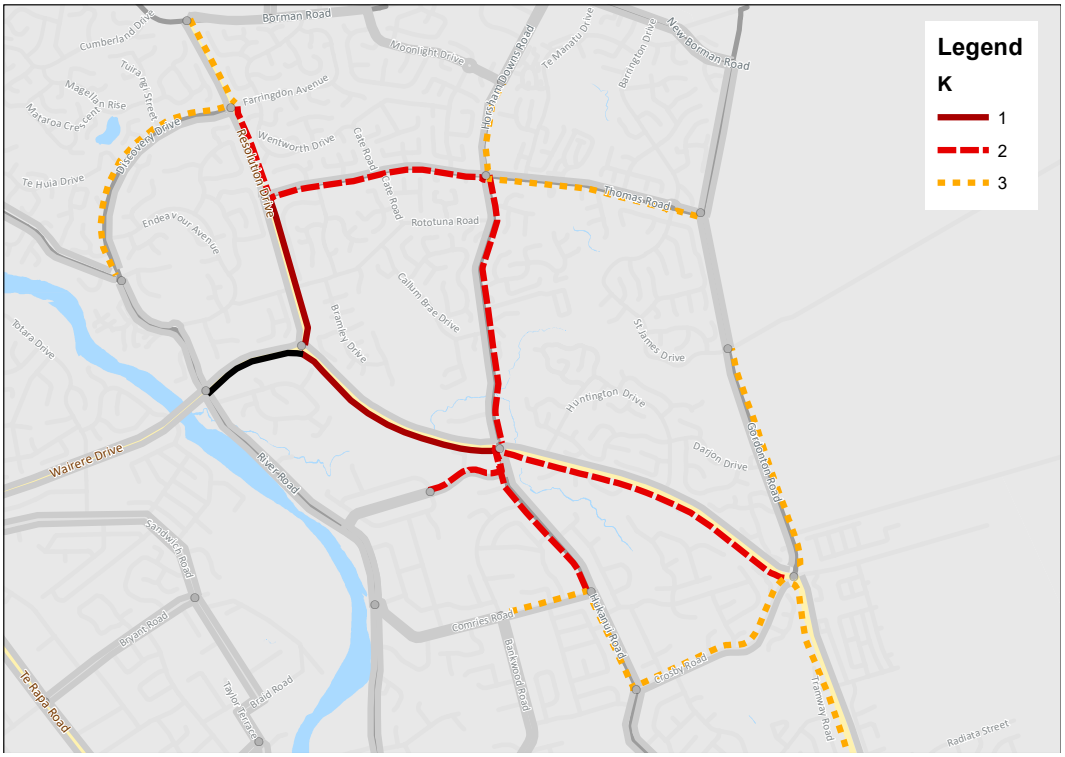


Figure 3.3: The number of hops from each node (road segment) to the link in black

An adjacency matrix is defined to represent the connectivity of nodes, $A \in R^{N \times N}$. Each element $A_{i,j}$ represents 1 if there is an edge connecting node i and node j, and 0 if not. Given that the current traffic state on a link will influence the future state on that link, all links are considered self influenced. An identity matrix I is added to A to obtained the one hop neighbourhood of the network:

$$\tilde{A} = A + I \tag{3.4}$$

The k-hop neighbourhood of the graph is defined as the number of edge traversals to travel from node i to node j. This is characterised as $(A+I)^k$, where k is the number of edge traversals. However, it is not necessary to weight a nodes k-hop neighbours by the number of hops, and as such all values of $(A + I)^k$ are clipped to $0, 1$. Values in each of the k-hop matrices are calculated as

$$\tilde{A}^k_{i,j} = min((A + I)^k_{i,j}, 1) \tag{3.5}$$

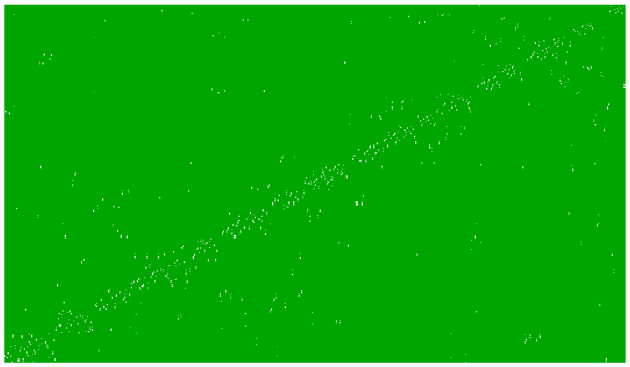


Figure 3.4: Adjacency matrix for the Hamilton traffic network at $K = 1$

Figure 3.3 shows the adjacency matrix for the AddInsight network in Hamilton City. An adjacency between each row and column at $k = 1$ hop is shown as a white dot. Green dots indicate no adjacency at one hop. By multiplying this matrix by itself, an additional set of matrices are created to limit the field of influence to links which are reachable within the time span used in the prediction, known as the *free-flow reachable matrix* (FFR). Each element $\text{FFR}^t_{i,j}$ is 1 if the link j is reachable from link i within time t, under free flow conditions. Free flow is defined as the time taken to traverse the full length of a link at the posted speed limit, with no delay encountered on the journey. Each road is considered self reachable and thus, each diagonal element is set to one.

Notations for the traffic graph are defined in the following list:

1. G: Traffic Network Based Graph $G = V, E$
2. V: Set of vertices in G with the size of $|V| = N$
3. E: Set of edges in G with the size of $|E|$
4. $A \in \mathbb{R}^{N \times N}$: Adjacency matrix of G
5. $FFR \in \mathbb{R}^{N \times N}$: Free-flow reachable matrix
6. $x_t \in \mathbb{R}^N$: Vector of delay of all graph nodes at time t

3.4 Traffic Graph Convolution

The graph convolutional layer in the model extracts localised features from the traffic network's graph structure. The region which affects the results of the convolution operation is known as a *receptive field*. The product of the neighborhood matrix A, the input data x_t, and the trainable weight matrix W, are used in the graph convolution operation to extract features from the one hop neighborhood. The operation is defined as

$$GC_t^k = (W_{gck} \odot A^k \odot FFR)x_t \tag{3.6}$$

where $\odot$ is element wise matrix multiplication (the Hadamard operator), and $x_t \in \mathbb{R}_N$ is the vector of traffic delay states of all nodes at time t. W_{gck} is a trainable weight matrix for the kth order traffic graph convolution and GC^k is the extracted kth order traffic graph convolution feature. The result of this operation is a sparse matrix, as A^k and FFR are both sparse matrices. The trained weight matrix measures the interaction between edges in the graph and subsequently allows for examination of how links in an area interact.

The features extracted from different orders of k-hops from 1 to K are concatenated into a vector, defined as:

$$GC_t^{\{K\}} = [GC_t^1, GC_t^2, ..., GC_t^K] \tag{3.7}$$

A comparison of the traffic graph convolution (TGC) to SGC and LGSC is shown below. The TGC performs better for spatial localisation, as it is able to extract local features from the FFR matrix based on physical attributes and reachable distances within set time ranges. SGC and LSGC need multiple convolution layers, resulting in a loss of model interpretability. The TGC operation only uses one convolutional layer allowing for easier interpretation of its parameters.

Graph Convolution	TGC	SGC	LSGC
Graph convolution on signal x_t	$W_{gck} \odot \tilde{A}^K \odot FFR$	$U\text{diag}(\theta)U^T$	$\sum_{j=0}^{K-1} \theta'_j L^j$
Weight Parameters	$W_{gck} \in \mathbb{R}^{N \times N}$	$\theta \in \mathbb{R}^N$	$\theta' \in \mathbb{R}^K$
Computation time	$O(N^2)$	$O(N)$	$O(K\|\xi\|)$

3.5 Traffic Graph Convolutional LSTM

The weights calculated by the convolution step are used an additional input into the LSTM model, and are optimised in the gradient descent algorithm in the same manner as the LSTM weights. If a link has a high influence on one if its downstream links, this will be captured in the weights and used alongside the time series sequence in the LSTM to enhance the prediction.

The final GC-LSTM model learns the complex spatio-temporal dependencies present in the data. The gate structure of the LSTM and hidden states are unchanged, while the input is replaced the the graph convolution features reshaped into a vector $GC^K \in \mathbb{R}^{KN}$ [5]. The gate structure of the LSTM is retained, and the input to each gate is replaced by the graph convolutional

features, reshaped into a vector $GC^K \in \mathbb{B}^{KN}$. The gates in the network are then defined as follow:

$$f_t = \sigma_g(W_f \cdot GC_t^K + U_f \cdot h_{t-1} + b_f) \tag{3.8}$$

$$i_t = \sigma_g(W_i \cdot GC_t^K + U_i \cdot h_{t-1} + b_i) \tag{3.9}$$

$$o_t = \sigma_g(W_o \cdot GC_t^K + U_o \cdot h_{t-1} + b_o) \tag{3.10}$$

$$C_t = \tanh(W_f \cdot GC_t^K + U_C \cdot h_{t-1} + b_C) \tag{3.11}$$

where $\cdot$ is the matrix multiplication operator, W are the weight matrices mapping the input for each of the three gates and the input cell state, U are the weight matrices for the preceding hidden state, and b are the four bias vectors. Each of the three gates use the sigmoid activation function σ while the cell state uses the hyperbolic tangent function. The loss during the training process is defined as:

$$\text{Loss} = L(y_T, \hat{y}_T) = L(x_{T+1}, h_T) \tag{3.12}$$

where $L(\cdot)$ is the residual between the predicted and true value.

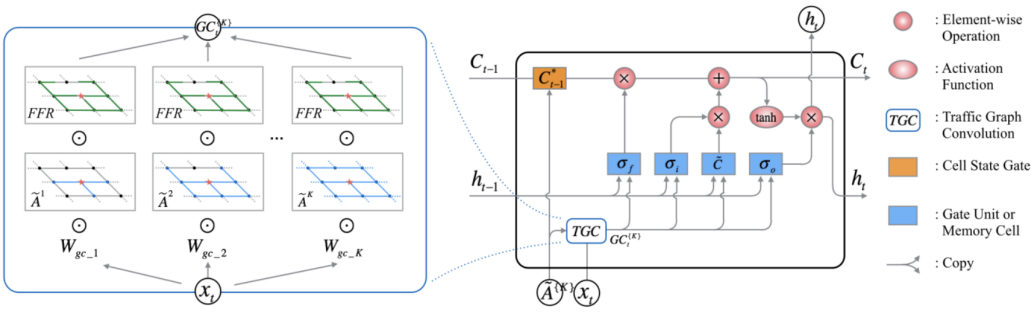


Figure 3.5: Architecture of the GC-LSTM model [12]

The architecture of the GC-LSTM model can be seen above. On the left, the convolution operation is performed to calculate a weight for each connected link and all k-hops. This vector of weights is then used as input into the standard LSTM architecture on the right. The full algorithm for the GC-LSTM is shown below in Algorithm 3.1.

Algorithm 3.1. *1:* ***Inputs:*** $X_t = [x_1, ..., x_T], \{\tilde{A}^1, ..., \tilde{A}^K\}, FFR$

2: ***Parameters:*** $W_{gc_1}, ..., W_{gc_K}, W_S, U_S, b_S, W_N$

3: ***Initialize:*** $h_0 = \boldsymbol{0} \in \mathbb{R}^N, C_0 = \boldsymbol{0} \in \mathbb{R}^N$

4: *For* $t = 1$ *to* T *do:*

5: *For* $k = 1$ *to* K *do:*

6: $GC_t^k \leftarrow (W_{gck} \odot \tilde{A}^k \odot FFR)x_t$

7: ***end for***

8: $GC_k^{\{K\}} \leftarrow [GC_t^1, GC_t^2, ..., GC_t^K]$

9: $h_t, C_t = TGC\text{-}LSTM(x_t, GC_t^{\{K\}}, h_{t-1}, C_{t-1}$

10: ***end for***
Return: h_T

3.6 Regularisation

In order to make the generated set of features and weights more interpretable, the model uses two regularisation operations. An L1-norm of the weight matrices to the loss function is used as a regularisation term:

$$R^1 = ||W_{gc}||_1 = \sum_{i=1}^{K} |W_{gci}| \tag{3.13}$$

A second regularisation operation is introduced on the features in the graph convolution operation. It considers the impact of neighbouring nodes with respect to a specific node must be transmitted through all nodes, between the node of interest and the influencing node. To restrict the difference between features extracted from adjacent hops in the convolution operation, the following term is added to the loss function at each step:

$$R^2 = ||GC_T^K||_2 = \sqrt{\sum_{i=1}^{K-1} (GC_T^i - GC_T^{i+1})^2} \tag{3.14}$$

This term ensures that features extracted from different k-hops should not differ dramatically and thus better reflect the physical realities of the network. The final loss function is defined as

$$\text{Loss} = L(h_T - x_{T+1}) + \lambda_1 R^1 + \lambda_2 R^2 \tag{3.15}$$

3.7 Summary

There are several deep learning models able to capture spatio-temporal dependencies in transportation data. A common approach is to model the network as a graph, in order to define the connectivity between roads and the distance between them. A grid based approach to the convolution of a transport network encounters issues with sparsity. Spectral graph convolution and traffic graph convolution are two approaches which model the network in a natural way, and allow for detailed analysis of how traffic flows propagate through the network.

Chapter 4

Bayesian Inference on Deep Learning

While traditional neural networks perform well in regions with large amounts of data, they are not able to express uncertainty with regions with little data. This results in overly confident predictions in these regions. Various approaches of Bayesian learning in neural networks have been proposed to solve this issue.

4.1 Bayesian Inference

Bayesian inference is a method of statistical inference grounded in Bayes Rule. It states that the probability of some event A, given that event B has occurred, is given by [49]:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{4.1}$$

In the context of Bayesian parameter estimation, we aim to find the posterior distribution of the parameters given the data:

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}, \tag{4.2}$$

where $p(\theta|x)$ is the posterior distribution, $p(x|\theta)$ is the likelihood, $p(\theta)$ is the prior, and $p(x)$ is the probability of the data. Often the denominator $p(x)$

involves computing an integral to normalise the posterior to a probability distribution:

$$p(x) = \int p(x|\theta)p(\theta)d\theta \tag{4.3}$$

In some cases, such as models where a conjugate prior is available, this is straightforward to calculate numerically. If no conjugate prior is available, there are a wide range of computational methods for sampling from the posterior. One of these is Markov chain Monte Carlo (MCMC). MCMC constructs a Markov chain with the target posterior distribution as its equilibrium distribution. By sampling from this chain for a long period of time, a sample from the posterior is obtained. A common algorithm for this is known as Metropolis Hastings, developed by Metropolis, Rosenbluth, Rosenbluth, Teller and Teller (1953). Formally, the algorithm uses a proposal distribution $g(x^{'}|x)$ as the conditional probability of accepting a state $x^{'}$ given x, and an acceptance ratio $A(x^{'}|x)$ as the probability of accepting the proposed state $x^{'}$ [50].

$$P(x^{'}|x) = g(x^{'}|x)A(x^{'}|x), \text{where} \tag{4.4}$$

$$A(x^{'}|x) = \min\left(1, \frac{P(x^{'})}{P(x)}\frac{g(x|x^{'})}{g(x^{'}|x)}\right) \tag{4.5}$$

MCMC can be used for parameter inference in any class of statistical model where the likelihood $g(x^{'}|x)$ can be evaluated. Consider a simple generalised linear model of the form $\pi = \frac{1}{1+\exp[-(\beta_0+\beta_1 x_i)]}$. Here the parameters are the intercept β_0 and a single coefficient β_1. The likelihood follows a Binomial distribution of the form

$$L(p|x) = \frac{n!}{x!(n-x)!}p^x(1-p)^{n-x} \tag{4.6}$$

While MCMC is a popular method for Bayesian inference models, it is often infeasible to use in deep learning models due to it's computational complexity [51].

4.2 Quantifying Uncertainty

The point estimate approach of traditional deep learning models may result in over-confident predictions of data points outside of the training distribution. This has implications in fields such as medical applications or autonomous driving, where a failure to make a prediction or an over-confident prediction can have undesirable consequences. A growing field in deep learning theory is in quantifying uncertainty in model predictions by capturing the uncertainty in the model parameters. Generally, there are two types of uncertainty present in a model: *epistemic uncertainty* measures the lack of knowledge in the data, measured by $p(\theta|D)$. This can be improved with more data. *Aleatoric uncertainty* is due to the inherent variability, and hence uncertainty, present in the data itself and measured as $p(y|x, \theta)$ [52]. This can be improved in the model selection and training process. The use of Bayesian inference in deep learning models allows us to distinguish between these two types of uncertainty.

Bayesian models specify a predictive distribution over yet unobserved data:

$$p(y|D, x) = \int p((y|w, x)p(w|D)dw \tag{4.7}$$

The above defines the probability for class label y given new input x and dataset D [33]. In practice, this integral is either numerically intractable or computationally infeasible. As a result, sampling methods must be used to approximate the posterior distribution and hence the predictive distribution over new data points. One such approach is *Monte Carlo dropout* [53]. This provides stochasticity in a neural network by randomly 'turning off' weights in each layer. This can be viewed as a Bayesian approximation to represent uncertainty.

An alternative solution to capturing uncertainty is *variational inference*. The aim is to approximate the true posterior distribution of each weight in the

network using an equivalent distribution function q and a latent variable Z. This approximating distribution $q(x)$ is compared to the true posterior $p(x)$ using a form of KL divergence.

4.3 Bayes via Dropout

Dropout is a procedure in the back-propagation algorithm. It randomly drops nodes in each layer of the network when re-calculation weights, which helps to prevent over-fitting. It also acts as a way to quantify prediction uncertainty when used during the evaluation phase as a form of ensemble learning. By combining the results of multiple models, the ensemble average can be found along with the variance in predictions. It is a convenient technique for quantifying uncertainty, given its simplicity to learn and little knowledge required to implement. However, it does not fully capture uncertainty in the model and only serves as an indication of uncertainty. It also lacks some flexibility compare to other methods described below which aim to provide a fully Bayesian approach to back-propagation [54].

4.4 Stochastic Gradient Descent

The goal of Stochastic Gradient Descent is to converge to a point estimate of the objective function while utilising noisy estimates of the gradient [55]. It is commonly used in mini batch gradient descent where noise is commonly seen in single batches. The parameter update rule for the movement can be written as:

$$\Delta\theta_t = \frac{\epsilon_t}{2}(\frac{N}{2}\Delta\log(p(D_t, \theta_t)) + \Delta\log(p(\theta_t))), \tag{4.8}$$

where D_t is a minibatch sampled at time t from the complete dataset, D. ϵ_t is the learning rate at time t, N is the size of the entire dataset and n is the size of the current minibatch.

There are MCMC algorithms based on the SGD algorithm, which aim to approximate the posterior distribution. It is able to find each mode of a complex posterior landscape, given sufficient running time. However, these methods suffer from the same computational drawbacks as MCMC, where convergence to the posterior often takes a large memory footprint and an infeasible length of computation. For simpler deep learning models it is a common approach.

4.5 Bayes by Backprop

Proposed in 2015, Bayes by Backprop introduces an algorithm for learning the probability distributions over the weights of a neural network [56]. The algorithm is based on a form of variational inference, where the expected lower bound (ELBO) is found to approximate the true posterior in an optimisation based approach. The Bayes by Backprop algorithm is described in the following section.

4.5.1 KL Divergence

The Kullback-Leibler Divergence (KL divergence) is a measure of similarity between two probability distributions. In Bayesian terms it is a measure of information gained when revising ones belief from a prior probability distribution Q to the posterior probability distribution P, or the amount of information lost when Q is used to approximate P. To find a distribution Q which is closest to the posterior P, we aim to minimise the KL divergence [57]. The measure originated from information theory, where a common metric in information encoding is Entropy:

$$H = -\sum_{i=1}^{N} p(x_i) \times \log p(x_i) \tag{4.9}$$

The above formula is modified to add an approximating distribution q, and the log difference is taken between each:

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \times (\log p(x_i) - \log q(x_i)) \tag{4.10}$$

This can be rewritten as the expectation of the log difference between p & q:

$$D_{KL}(p||q) = E[\log p(x) - \log q(x)] \tag{4.11}$$

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \times \log \frac{p(x_i)}{q(x_i)} \tag{4.12}$$

Take two distributions P and Q. P is a binomial distribution with parameters $N = 2$ and $p = 0.4$. Q is a discrete uniform distribution with parameter $p = 1/3$. The KL divergence between the two distributions is calculated as follows:

$$D_{KL}(P||Q) = \sum_{x \in X} P(x) \ln(\frac{P(x)}{Q(x)}) \tag{4.13}$$

$$= \frac{1}{25}(32\ln(2) + 55\ln(3) - 50\ln(5)) \approx 0.0852996 \tag{4.14}$$

$$D_{KL}(Q||P) = \sum_{x \in X} Q(x) \ln(\frac{Q(x)}{P(x)}) \tag{4.15}$$

$$= \frac{1}{3}(-4\ln(2) - 6\ln(3) + 6\ln(5)) \approx 0.097455 \tag{4.16}$$

The above shows that the KL divergence is not a symmetrical formula and depends on the order in which P and Q are specified.

4.5.2 Variational Inference

As we cannot model the posterior $P(w|D)$ directly, we instead find the parameters θ of a distribution q on the weights, denoted as the variational posterior $q(w|\theta)$. We aim to minimise the KL divergence with the true posterior [58]. This is expressed as:

$$= \arg\min_{\theta} KL[q(w|\theta)||P(w)] - E_{q(w|\theta)}[\log P(D|w)] \tag{4.17}$$

This is known as the variational free energy. The first term is the KL divergence between the variational distribution $q(w|\theta)$ and the prior on the weights $p(w)$, and is known as the complexity cost. The second term is the expected value of the likelihood w.r.t. the variational distribution, and is known as the likelihood cost. We can rearrange the first term to obtain the loss function as follows, defined as the *expected lower bound (ELBO)*:

$$F(D,\theta) = KL[q(w|\theta)||P(w)] - E_{q(w|\theta)}[\log P(D|w)] \tag{4.18}$$

$$= F(D,\theta) = E_{q(w|\theta)} \log(q(w|\theta) - E_{q(w|\theta)} \log p(w) - E_{q(w|\theta)} \log p(D|w) \tag{4.19}$$

In practice, we sample from the variational distribution $q(w|\theta)$, as all three terms are expectations w.r.t. the variational distribution. Monte Carlo sampling is used to draw samples $w^{(i)}$ from the variational posterior:

$$F(D,\theta) \approx \sum_{i=1}^{N} [\log q(w^{(i)}|\theta) - \log p(w^{(i)}) - \log p(D|w^{(i)})] \tag{4.20}$$

4.5.3 Reparameterisation

The local reparameterisation trick moves the parameters to be learned, μ and θ, out of the distribution function for any weight w. A new parameter ϵ is defined as a sample of a standard Gaussian distribution [56]. It is then multiplied by σ and μ is added:

$$\theta = (\mu, \sigma^2) \tag{4.21}$$

$$\epsilon \approx N(0,1) \tag{4.22}$$

$$f(\epsilon) = w = \mu + \sigma \times \epsilon \tag{4.23}$$

The above two parameters of interest are incorporated into every weight value in the network. They are learned according to the following:

$$\Delta\mu = \frac{\partial f}{\partial w} + \frac{\partial f}{\partial \mu} \tag{4.24}$$

$$\Delta\sigma = \frac{\partial f}{\partial w}\frac{\epsilon}{\sigma} + \frac{\partial f}{\partial \sigma} \tag{4.25}$$

$$\mu \tag{4.26}$$

4.5.4 Gaussian Variational Posterior

A sample of the weights w can be obtained by sampling a unit Gaussian, shifting it by mean μ and standard deviation σ. The standard deviation is parameterised by $\theta = log(1 + \exp(p))$, thus ensuring that it is always positive. The variational parameters are $\theta = (\mu, p)$. The sample of weights hence becomes $w = t(\theta, \epsilon) = \mu + \log(1 + \exp(p)) \cdot \epsilon$, where $\cdot$ is pointwise multiplication. The algorithm proceeds as follows to update the variational parameters μ and p [56]:

1. Sample ϵ $N(0, 1)$
2. Let $w = \mu + \log(1 + \exp(p)) \circ \epsilon$
3. Let $\theta = (\mu, p)$
4. Let $f(w, \theta) = \log q(w|\theta) - \log P(w)P(D|w)$
5. Calculate the gradient with respect to the mean as $\Delta_u = \frac{\partial f(w,\theta)}{\partial w} + \frac{\partial f(w,\theta)}{\partial \mu}$
6. Calculate the gradient with respect to the standard deviation parameter p as $\Delta_p = \frac{\partial f(w,\theta)}{\partial w}\frac{\epsilon}{1+\exp(-p)}\frac{\partial f(w,\theta)}{\partial \epsilon}$
7. Update the variational parameters as $\mu \leftarrow \mu - \alpha\Delta_\mu, p \leftarrow p - \alpha\Delta_p$

4.5.5 Prior Distribution

A simple Gaussian distribution can be used to define the prior:

$$P(w) = \prod_i N(w_i|0, \sigma_p^2) \tag{4.27}$$

$$\log P(w) = \sum_i \log N(w_i|0, \sigma_p^2) \tag{4.28}$$

Alternatively, [56] recommends a Gaussian scale mixture prior, defined as the weighted sum of two zero-centered Gaussian distributions. The variance of the second mixture component is smaller than the first, giving a heavier tail in the prior density than a plain Gaussian prior. This is found to concentrate many of the weights around zero.

$$P(w) = \prod_i (\pi N(w_i|0, \sigma_1^2) + (1-\pi)N(w_i|0, \sigma_2^2)) \tag{4.29}$$

$$\log P(w) = \sum_i \log(\pi N(w_i|0, \sigma_1^2) + (1-\pi)N(w_i|0, \sigma_2^2)) \tag{4.30}$$

$\pi, \sigma_1^2, \sigma_2^2$ are hyperparameters and are not learned during training. The variational posterior is a Gaussian distribution with mean vector μ and variance vector σ^2:

$$q(w|\theta) = \prod_i N(w_i|\mu, \sigma^2) \tag{4.31}$$

$$\log q(w|\theta) = \sum_i \log N(w_i|\mu, \sigma^2) \tag{4.32}$$

4.5.6 Network Training

During each forward pass (epoch) in the model, a sample is drawn from the variational posterior distribution. The cost function is evaluated against this sample, with the likelihood term being evaluated at the end of the forward pass. During the backpropagation process, the gradients of μ and σ are calculated so that their values can be updated. For numeric stability, the network is instead parametrized with p and transformed with the softplus function to obtain $\sigma = \log(1 + \exp(p))$. This step ensures that σ is always positive.

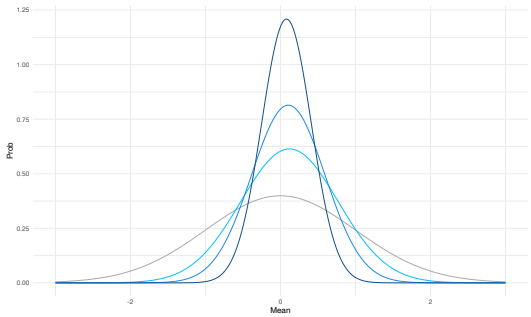


Figure 4.1: Weight One

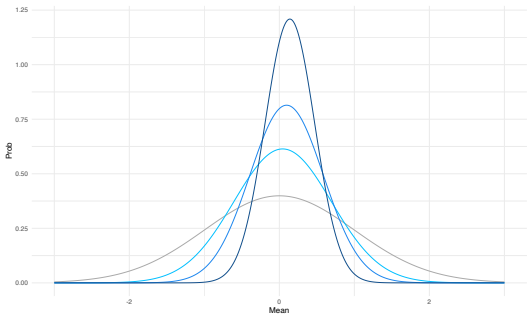


Figure 4.2: Weight Two

Above is an example of two randomly sampled weights, showing the initial prior in dark grey against their subsequent distributions across each batch of 100 epochs in increasing shades of blue. The final model at 300 epochs is shown in dark blue. Each weight can be seen to converge to its posterior distribution along with a decreasing standard deviation.

4.5.7 Prediction Uncertainty

To obtain a credible interval over the prediction, samples are drawn from the variational posterior distribution, which approximates the true posterior. For each of these samples a prediction is calculated and the mean and standard deviation is found. The standard deviation is used to calculate a credible interval over each point prediction. In regions of the dataset with high uncertainty, either through a large variation in observed data or a lack of data points, this uncertainty is represented through a wide credible interval. This effect is shown in Figure 4.3.

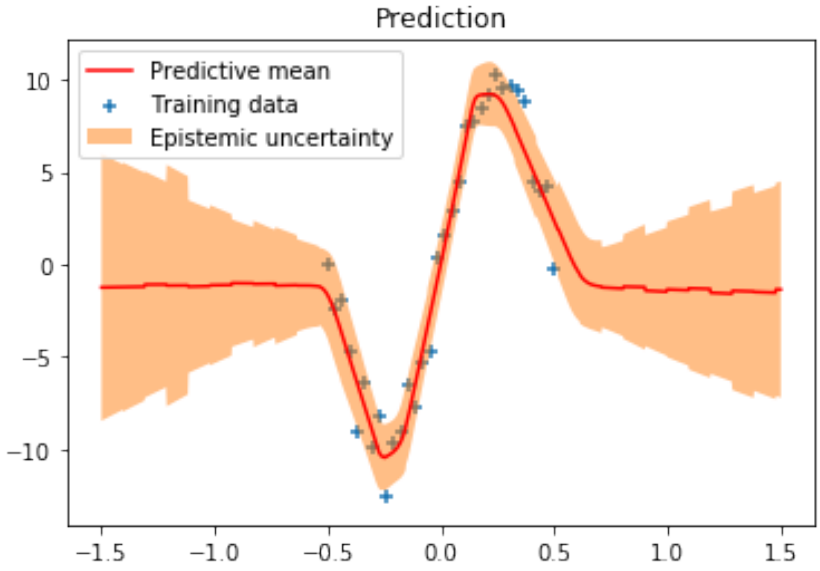


Figure 4.3: The variation in prediction uncertainty across regions of a dataset

4.6 Bayesian Deep Learning in PyTorch

There are several Python libraries which use the Bayes by Backprop algorithm to create a distribution over weights in the model. One such library is *Blitz* [59].The library provides an example of an LSTM model utilising Bayesian layers, shown below. The closing stock price for IBM is predicted for the final 750 days given 11 years of historical data. The results are shown in Figure 4.4.

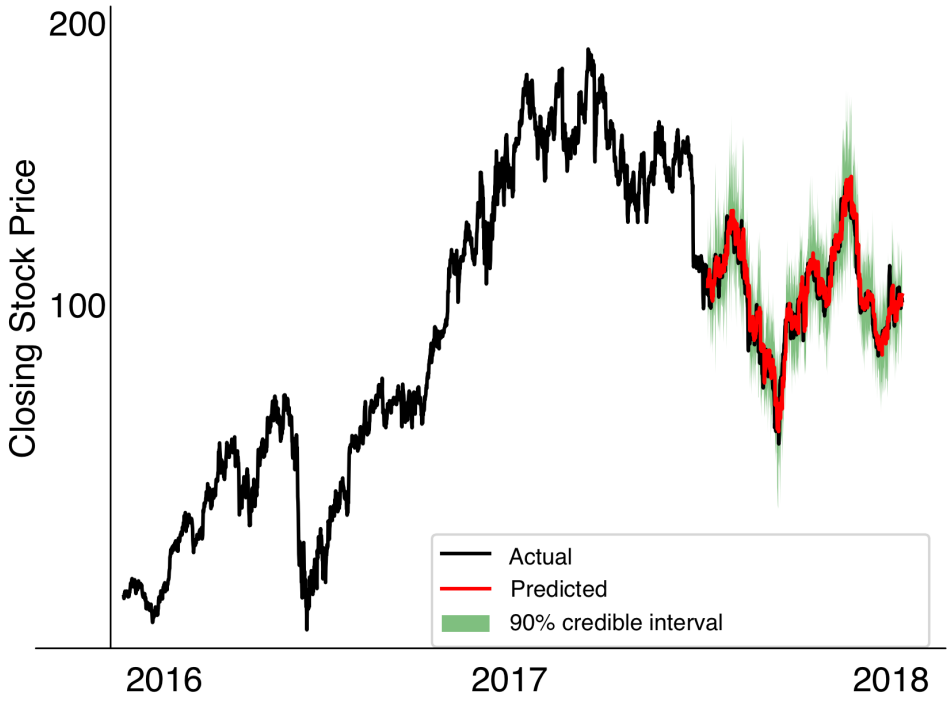


Figure 4.4: The predicted vs actual stock price for IBM across 750 days, and the 90% credible interval shown in green

4.6.1 LSTM Model on Delay Data using Blitz

Delay for the section of Wairere Drive in Hamilton City (from Resolution Drive to River Rd) was evaluated with a 90% credible interval. The model was run across 300 epochs with a batch size of 8. These parameters were chosen after testing the validation dataset across a range of hyper-parameters and choosing an optimal point between computation time and model accuracy.

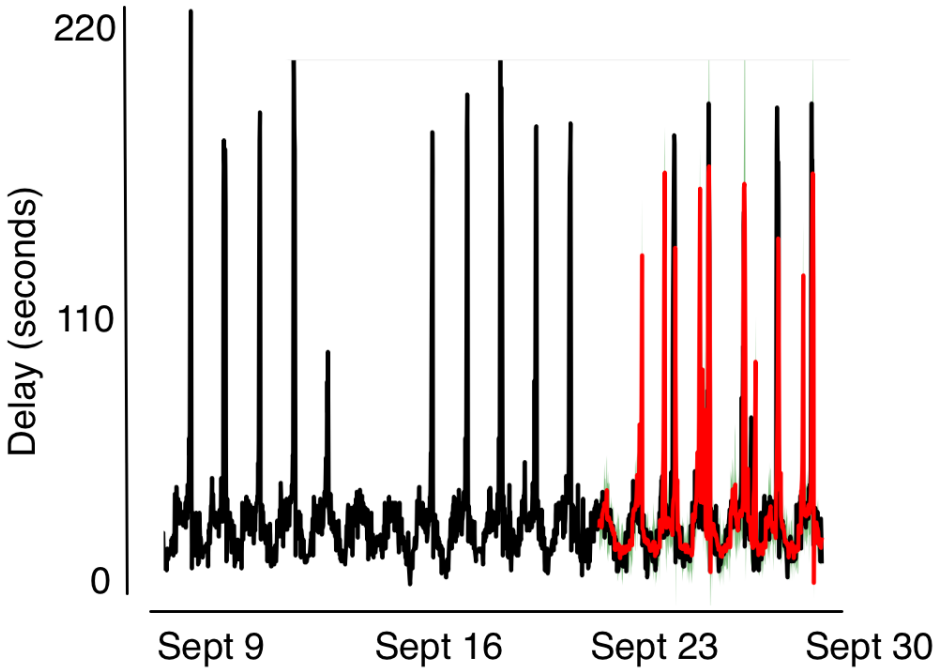


Figure 4.5: Predicted vs actual delay for Wairere Drive, with predicted delay shown in red

The model accurately predicts inter-peak volumes and manages to capture the general trend in the data. However, it under-predicts the peaks. It also appears to predict a significant PM peak on Sunday, although it accurately predicts the low delay seen on Saturday. The 90% credible interval is sufficiently narrow and correctly models greater uncertainty in the peak periods where delay can vary by significant levels.

4.7 Summary

Bayesian Inference in Deep Learning can be accomplished with several models, including Dropout ensembles, MCMC, and Variational Inference. In practice, dropout is a common method to easily obtain an idea of model uncertainty through ensemble averaging. However, it does not fully capture model uncertainty. MCMC is commonly used in statistical inference models, but when applied to deep learning models quickly becomes infeasible computationally. Variational Inference allows uncertainty to be modelled in a deep learning framework through an optimisation approach. While it does not model the true posterior in the case of MCMC, it offers a sufficient approximation in most deep learning models and can be achieved with less computation time.

Chapter 5

Transportation Data

A transport network consists of a number of intersections and roads connecting the intersections. It carries different modes of traffic to and from zones, defined as *origins* and *destinations*. Hamilton City collects a variety of data on the use of the network. One such dataset is traffic volumes. This is collected in a variety of ways. One method is using sensors embedded in lanes of signalised intersections, also know as *detectors* [60]. These detectors record the count of vehicle passing over them. Typically, this data is aggregated to a count of vehicles at a given intersection (*site*) or road (*link*). Another method for recording volumes is with pneumatic tubes [61]. These typically take the form of rubber tubes laid across a roadway, and are also able to record vehicle classification via the gap between axles as well as speeds. It is best practice to collect data on every main transportation mode - heavy vehicle, bus, light vehicle, cyclist, pedestrian, or micro-mobility device. This data is commonly used to inform *mode share*, defined as the proportion of each mode using a given part of the network. In recent times there has been an increasing focus on cities to change the observed mode share to see increased volumes of certain modes such as cyclists and scooters. Following an infrastructure intervention such as cycleway or shared path, the goal is to increase the volume of cyclists and/or pedestrians, resulting in a mode share shift towards these modes and away from vehicles. In areas where pedestrians are prioritised such as a central business

district, measures such as raised platforms, signalised pedestrian crossings, and off street pathways are used to encourage walking while discouraging vehicular thoroughfare [59].

Another dataset commonly used in transportation analysis is travel times. This data can be collected in a variety of ways including cameras, Bluetooth sensors and number plate recognition. HCC uses a system called AddInsight [18] which utilises Bluetooth sensors to detect devices in vehicles and record the travel time between sites in the network. This data is recorded as the travel time on a given link. An attribute called *delay* is then calculated based on the observed travel time. It is typically the difference between an expected travel time and the observed travel time. If a vehicle is expected to take x seconds to travel along a link and it is observed to have taken y seconds, the delay d is calculated as $d = y - x$. Travel time data is an important data source in major cities and provides vital information on the performance of the transport network. It is used to alert operators to high levels of delay, which may be caused by a variety of factors such as a crash, unusually high traffic volume, or a traffic signal fault. The historical data allows for analysis of the trends in delay on each part of the network, which when paired with equivalent traffic volume data provides insight into how the network is changing in accordance with increased population growth and new developments for both housing and employment. By identifying parts of the network with high delay, action such as traffic signal optimisation, construction of roundabouts or signals, and future road planning can be carried in out a more informed manner.

5.1 AddInsight Delay Data

Addinsight is an Intelligent Transport System (ITS) that collects floating car data from road network deployed hardware and performs vehicle re-identification, allowing for travel time calculation of individual vehicles and for road segments

(links). It then performs real time travel time prediction based on historical data so that unusual congestion is reported as a possible incident. This travel time data is saved to a database to allow for analysis of trends and patterns. As of June 2021, HCC has deployed 170 sensors across all collector, arterial and state highway roads (Figure 5.2). The connections between these sensors follow road segments in a directional manner. There are approximately 570 links within the city boundary, with the majority recording travel time data since November 2018. In addition to travel time and delay data, the identification of vehicles allows for tracking journeys across the network, including their origin and destination to a block granularity. This data provides a better understanding of how vehicles travel around the transport network. For each link in the network, the AddInsight software is able to show which roads were used to get to the given link, as well as where those journeys subsequently travelled to. This data can be potentially be used in a spatio-temporal deep learning model for analysing the effects of interruptions on the network such as closures or crashes. Refer to Section 7 for more detail and model results.

Recording travel times requires sensors mounted along the road where each vehicle/probe can be detected and report back a unique identifier. When a probe is detected again at a different location (re-identified) the travel time and speed can be calculated by comparing the time at which the probe was detected by both sensors. Addinsight sensors in Hamilton City supports MAC addresses from Bluetooth devices such as car stereos, with new detection types such as BT-LE (Bluetooth Low Energy) and Wi-Fi currently being explored. Analysis of the capture rate currently shows 20% of vehicles on the network are being detected with AddInsight sensors. This capture rate is shown in Figure 5.1. The red bars show the count of vehicles captured using AddInsight sensors, with the teal bars showing the total volume of vehicles captured using in ground detectors (via the SCATS system). The technology is able to capture a consistent rate of vehicles across a week.

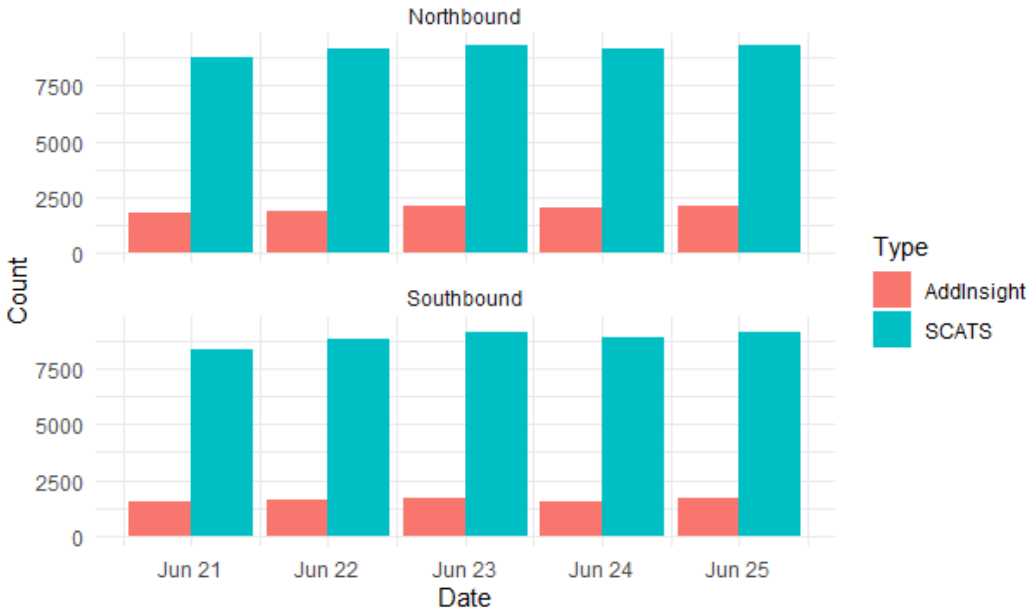


Figure 5.1: The capture rate of two links on Tristram St in the Hamilton CBD

Date	Link_ID	Travel_Time	Delay
1/6/2021 16:45	304	273	195
1/6/2021 17:00	304	316	238
1/6/2021 17:15	304	389	311

Table 5.1: An example of travel time and delay data recorded by AddInsight

Table 5.1 shows a sample of recorded travel time and delay for the link between Pukete Rd and River Rd, via Wairere Drive. Both travel time and delay are in seconds. Note that the free flow travel time for this link is 78 seconds, with the recorded delay calculated as the free flow time subtracted from the observed delay.

Figure 5.2: AddInsight Network, showing links (blue lines) and sensors (orange dots)

5.2 AddInsight Traffic Graph

The site and link structure of the AddInsight network in Hamilton City is naturally represented as a graph $G = V, E$ with N nodes $v_i \in V$ and edges $(v_i, v_j) \in E$. The connections between nodes are defined with an adjacency matrix $A \in R^{N \times N}$, in which each element $A_{i,j} = 1$ if there is an edge connecting node i and node j. See Figure 5.3 for an example of a generic graph. The number of edges that must be traversed to travel from node a to node b is called the number of *hops* - for example node 3 below is 2 hops from node 1, as there is first a traversal from node 1 to node 2, followed by a traversal from node 2 to node 3. The set of nodes which required k traversals from a given node a is known as the set of k-hops for node a. Links which feed into a given link are denoted as *upstream* links, while those which follow on from a given link are denoted as *downstream* links. Each node a has a set of i nodes at $k = 1...n$ traversals, where n is the furthest number of traversals required to reach any other node in the network, for node a. In terms of spatial influence,

it is expected that nodes at set $k = 1$ will influence node a at the next recorded time interval, with increasing values of k will influence the node at increasing time interval lengths. Additionally, if data is not recorded for a certain set, a model which utilises the graph structure will lack information that could be used to inform a prediction y for a given node at time t. This effect is explored further in Section 7.

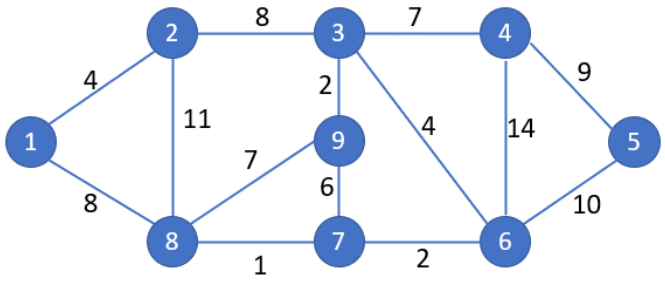


Figure 5.3: A graph of nodes and edges, with an attribute attached to each edge.

5.3 Northern Links Model

To analyse the effect of links which lack spatial information in the GC-LSTM, a subset of links in Hamilton City was first used in the GC-LSTM model.This subset was taken as the set of links above the *cross-city connector*, or the section of Whatawhata Rd beginning at Newcastle Rd to the western end of Fifth Ave meeting Wairere Drive (Figure 5.4). The data for the links bordering the cross-city connector does not include any links to their south, or at $k = 1$ hop. Some links at one traversal north from this set are lacking data from links at $k = 2$ hops. As the value of k increases, the influence of the missing data will decrease due to higher weighting given to direct upstream and downstream links. Figure 5.5 shows the set of links in the northern model with missing data at $k = 2$ hops. Section 7.3 provides an analysis of the northern links model against the performance of the full model, in which the links directly

above the cross-city connector have additional spatial information.

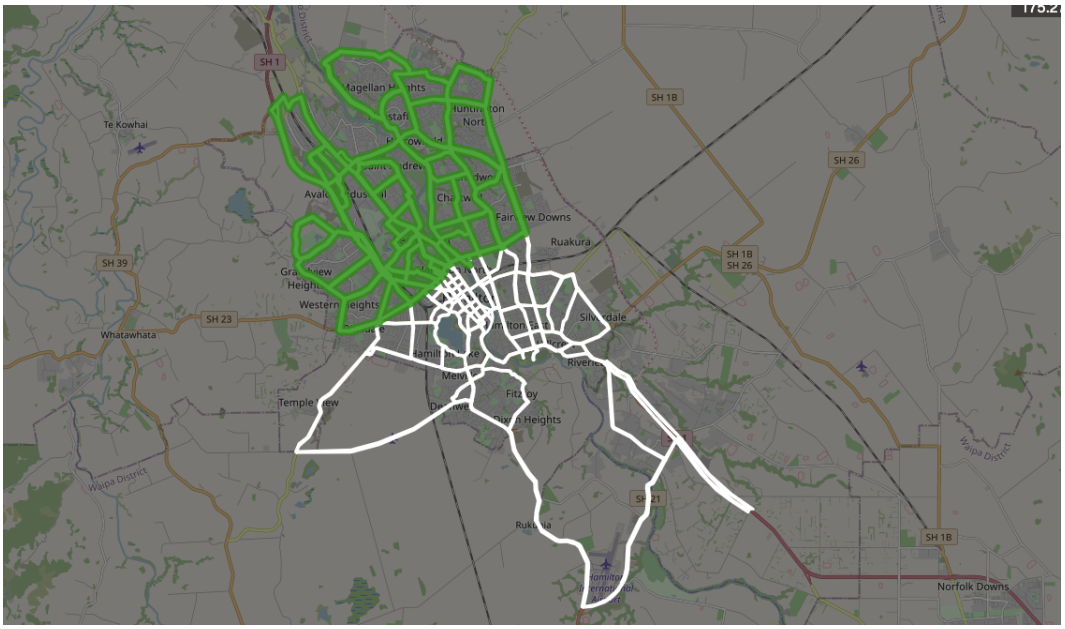


Figure 5.4: The green links are the spatial extent of the northern model, while links in white are excluded

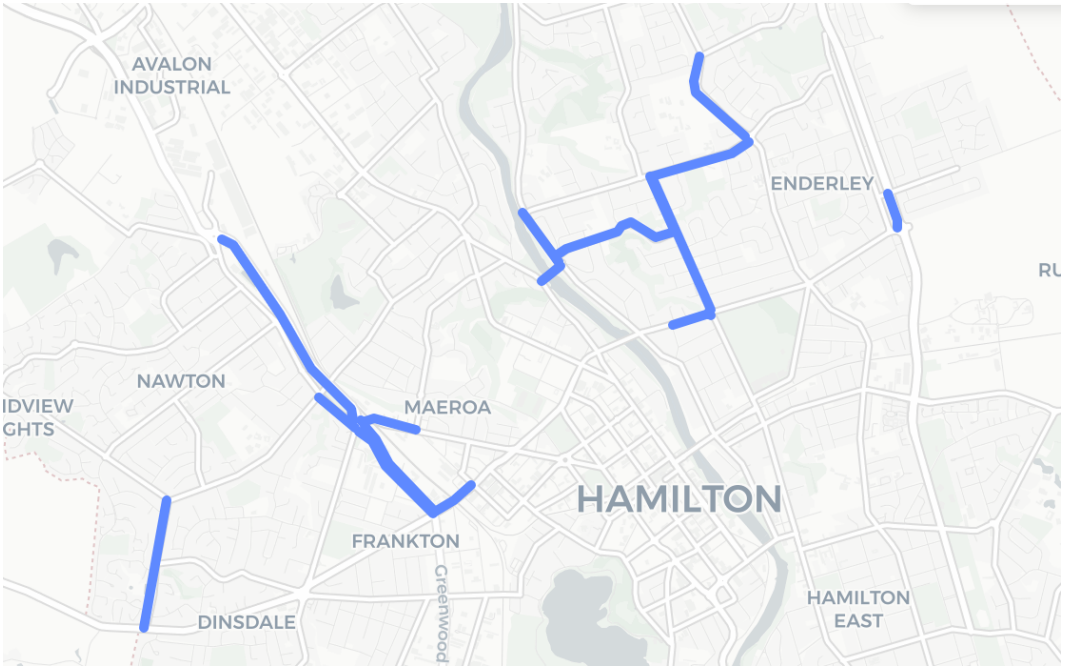


Figure 5.5: Links with missing data at k = 2 hops

Chapter 6

Data & Measures

6.1 Delay Data

The delay on a link (road segment) is dependent on a number of factors, due to the complex spatio-temporal interaction of traffic. A given link is influenced by its set of upstream links, and in turn influences its own downstream links. Each upstream link has a different weighting which varies across the day.

In addition to the spatial correlation present in the road network, the delay at a given time t_i at a link L_n is determined by the sequence of delay at times $T = t_{i-1}, t_{i-2}, ..., t_{i-n}$. To accurately determine the delay at a given link and time of day, both the graph structure and short term time series data need to be considered. This can be achieved by the use of a neural network which integrates both convolutional and recurrent layers.

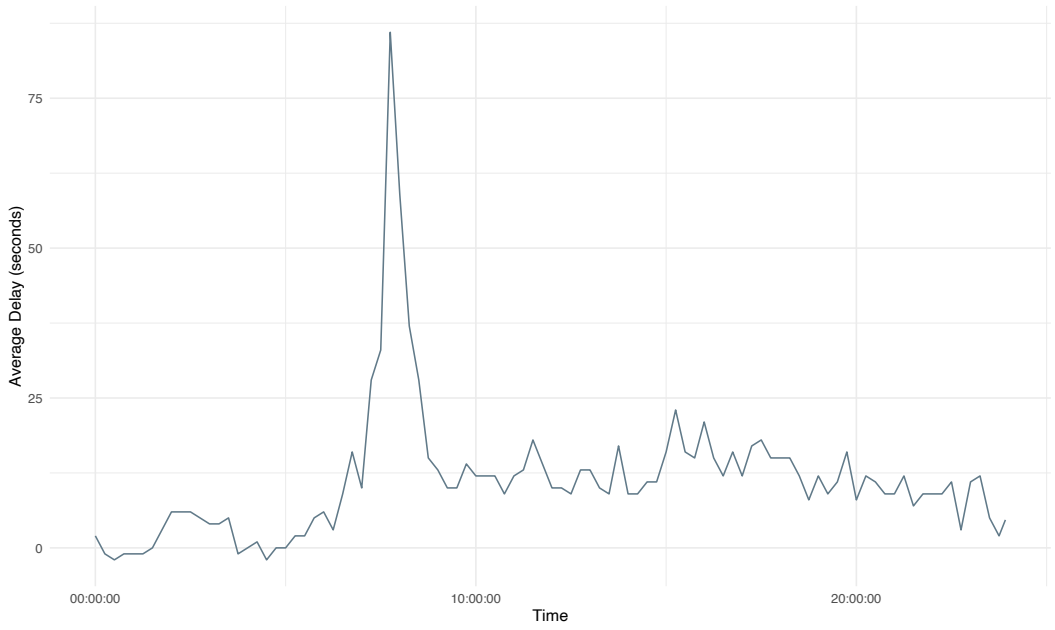


Figure 6.1: Observed delay on Wairere Drive, showing a large morning peak

Eight weeks of data were extracted for the entire network of sensors in Hamilton City, from August 1, 2019 to September 30, 2019. The data is in fifteen minute intervals for each link, for a total of 533 links. The variables present in the model are the time interval, link ID, travel time (seconds) and delay (seconds), with the latter two being an average of all observations in the given interval.

Date	ID	TravelTime	AverageDelay
01/07/2019 07:00	442	110	21
01/07/2019 07:15	442	112	23
01/07/2019 07:30	442	118	29
01/07/2019 07:45	442	121	32
01/07/2019 08:00	442	120	31

Table 6.1: An example of recorded data from the AddInsight system used in the GC-LSTM model

The average delay is subsequently adjusted to a new variable named 'pro-

portional delay', which scales the observed delay according to an adjusted expected delay. This expected delay is calculated as the median of all minimum travel times over a month in the given fifteen minute period. This gives a more reasonable value for a 'baseline' travel time, due to the expected travel time present in the data showing erroneous values in times with little traffic. The delay variable is then recalculated as the difference between the observed travel time and adjusted expected travel time. The proportional delay is then calculated as a scaled variant of the delay:

$$d_p = \frac{t}{t-d} - 1 \tag{6.1}$$

where d = the adjusted delay based on the re-calculated expected travel time, and t = the observed travel time in the given time period. This adjusted delay in shown in Table 6.2.

Date	ID	PropDelay
01/07/2019 07:00	371	0.462
01/07/2019 07:15	371	0.477
01/07/2019 07:30	371	0.492
01/07/2019 07:45	371	0.543
01/07/2019 08:00	371	0.527

Table 6.2: Adjusted delay (proportional delay) for Link 371

6.2 Accuracy Measures

Two accuracy measures are defined to evaluate the results of the model. The first is the MPE, or *mean percentage error*. This is defined as:

$$MPE = \frac{1}{N}\sum_{i=1}^{n}\frac{y_t - \hat{y}_t}{y_t} * 100\%, \tag{6.2}$$

where y_t is the observed delay and $\hat{y}_t$ is the predicted delay. A positive value indicates a trend towards over-prediction in the model, while a negative value indicates under-prediction. Additionally, an additional measure is defined as the percentage of predictions within 10% of the observed delay. 10% is the approximate and acceptable error rate for the recording of traffic volume and delay, and therefore is chosen as the target range for which the majority of predictions should be within. This accuracy measure is referred to as P.

Chapter 7

Results

The optimal number of epochs was set by running the model for 500 epochs and plotting the validation loss. The convergence point was determined to be 300 epochs, at which point the decrease in validation loss is outweighed by the increased computation time. Each epoch took between 55 seconds and one minute to run, for a total running time of approximately 190 minutes. The total dataset used was Monday September 2, 2019 to Friday September 27, 2019. Weekends were included, as the model is able to pick up the recurring pattern of the difference in delay between weekdays and weekends. The prediction range was the final day of the four week range, with the first two weeks used to train the model and the following thirteen days used as a validation set. The learning rate was set to 1e-5, which was chosen through running the model over a range of learning rates and evaluating the validation set accuracy. The batch size was set to 48 after running the model through a number of batch sizes from 16 to 96, and evaluating the balance of validation loss against computation time. See Section 7.3 for more detail on batch size optimisation.

Model	Peak	MPE	P
LSTM	AM	0.22	74.3%
LSTM	Inter	0.06	87.1%
LSTM	PM	0.13	74.2%
GC-LSTM	AM	0.12	**80.4%**
GC-LSTM	Inter	0.05	**88.6%**
GC-LSTM	PM	0.11	**79.7%**

Table 7.1: Results for the full GC-LSTM model

Table 7.1 shows the results for both the LSTM model and the full GC-LSTM. Overall, the model performed well across all times of day, with an MPE close to zero and the majority of the predictions within an acceptable +/- 10% error rate. The GC-LSTM model outperformed the LSTM for all time ranges, highlighting the increased precision when incorporating the spatial dependence of the traffic network. The difference in accuracy between the LSTM and GC-LSTM is highest during the peaks, showing that the links have a higher influence on their downstream links during these time periods than in off-peak periods.

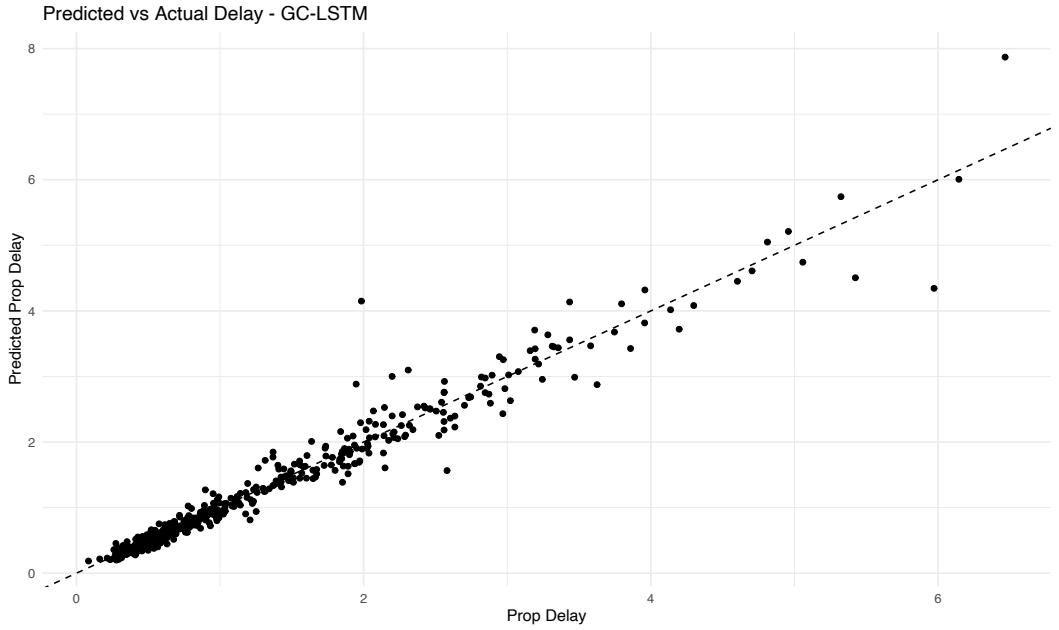
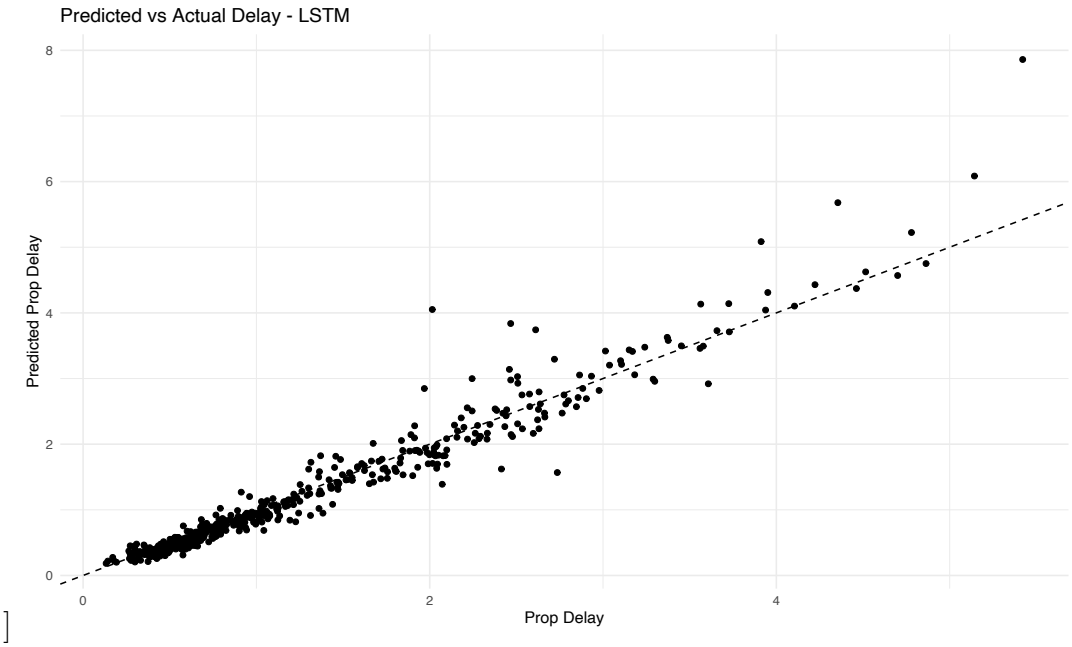


Figure 7.1: Predicted vs actual values for the GC-LSTM



]

Figure 7.2: Predicted vs actual values for the LSTM

A sample of the observed vs predicted delay is shown in Figure 7.1, with the equivalent results for an LSTM shown in Figure 7.2. There is little evidence of increasing variance and the observations are closely bound to the dotted line, indicating high accuracy at all levels of delay. The point in the upper right corner appears to be an anomaly caused by an incident on the road network.

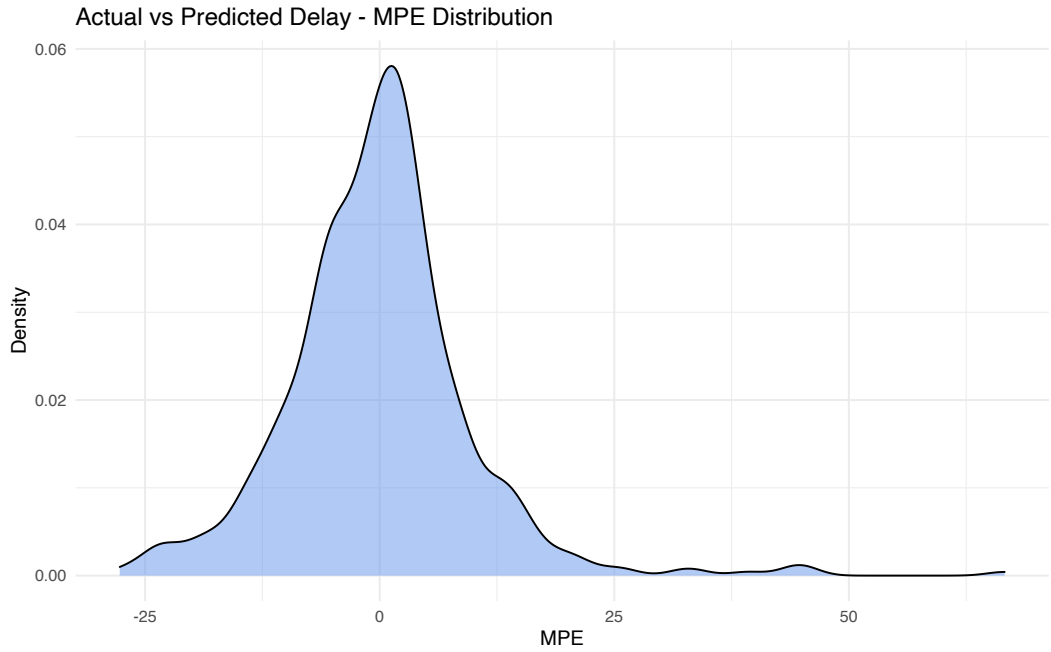


Figure 7.3: Distribution of the MPE across all links

Even at this high level of delay, the predicted value is extremely close to the observed delay. In comparison, the LSTM also exhibits high accuracy but has a higher variance and error than the GC-LSTM. As observed delay increases, the LSTM model tends to overpredict, while the GC-LSTM model is still able to predict accurately without clear over or under-prediction. These periods of unusually high delay tend to occur as a flow on effect of an incident on the network. This shows the value in using a model which is able to capture such spatial dependencies. The GC-LSTM is able to capture delay seen on upstream links, which then propagates through the network to the given link on Wairere Drive. As described in Section 3.4, the model achieves this by assigning weights to upstream links defined by the adjacency matrix and restricted by the K parameter and FFR matrix. As these weights are another input into the LSTM model, they can be trained through the back-propagation process, and the model is subsequently able to capture the influence of upstream links on any given link.

The distribution of the percentage error (MPE) between observed and pre-

dicted values (Figure 7.3) shows a normal distribution centred close to zero. The majority of points are within the acceptable level of +/-10%. There are a few observations with an over-prediction up to 73%, although this error occurs predominantly for predictions on low levels of existing delay. In practice, such a model is primarily used for prediction at high levels of delay, as the majority of traffic delay analysis at HCC is conducted during peak times.

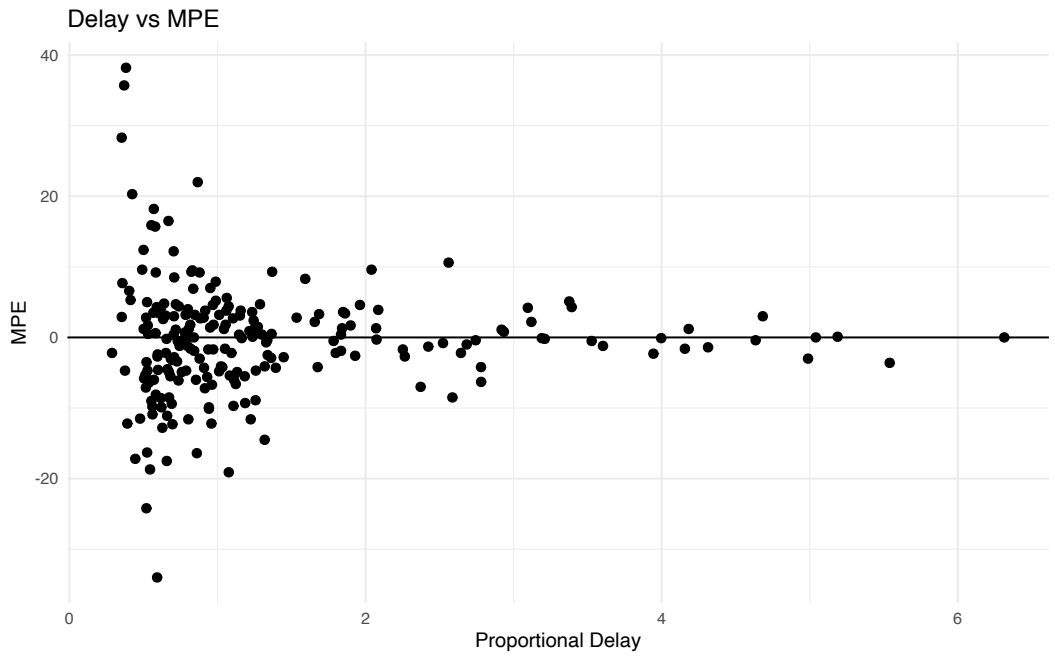


Figure 7.4: Observed proportional delay vs MPE

Figure 7.4 shows the observed delay against the MPE. High variance in the MPE is seen at low levels of delay. This is to be expected, given the tendency for the MPE measure to take on extreme values when the input is close to zero. The absolute difference in observed vs predicted delay in this area is quite low, and within the range of a few seconds in a real-world context. At high values of observed delay, the MPE remains centered at zero. The observations with high error values also occur for links with low levels of traffic and short roadway lengths. Due to these attributes, these links experience a high variance in observed delay.

7.1 Prediction on Wairere Drive

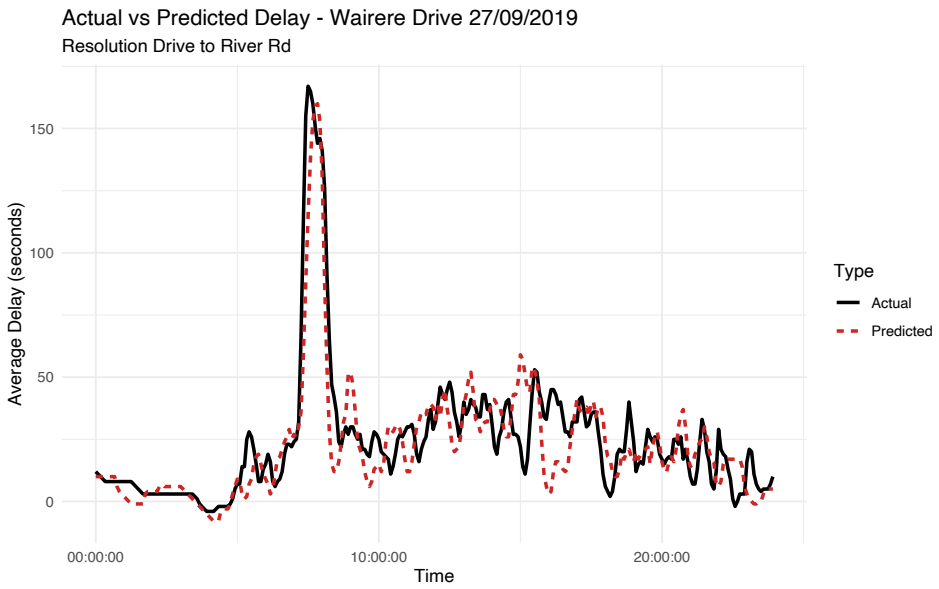


Figure 7.5: Predicted vs actual delay on Wairere Drive, between Resolution Drive and River Rd, on 27/09/2019

Prediction accuracy is evaluated on a key link (371) in the north of Hamilton City (Figure 7.5). The link runs westbound from Resolution Drive to River Rd. Its destination intersection, Wairere/River, is the busiest intersection in Hamilton with a peak of 70,000 vehicles a day. The other approaches at this intersection also experience high traffic volumes throughout the day, led by significant population growth in the north of the city as well as proximity to key shopping centres and employment zones. As a result, Link 371 experiences significant delay in the morning peak.

The GC-LSTM model is able to capture this morning peak, as well as much of the variance in delay in the remainder of the day. This is due to the spatial information captured from downstream links where a morning peak was observed at $t - n$ minutes, $n \in [5, 10, 15]$. This demonstrates the improvements in prediction accuracy in cases where there are occurrences on the network not typically seen, caused by a number of factors in traffic flow such as an

unusually high volume, incidents, or road closures. This information can only be captured in a model taking into account the spatial dependencies in the network.

7.2 Quantifying Uncertainty

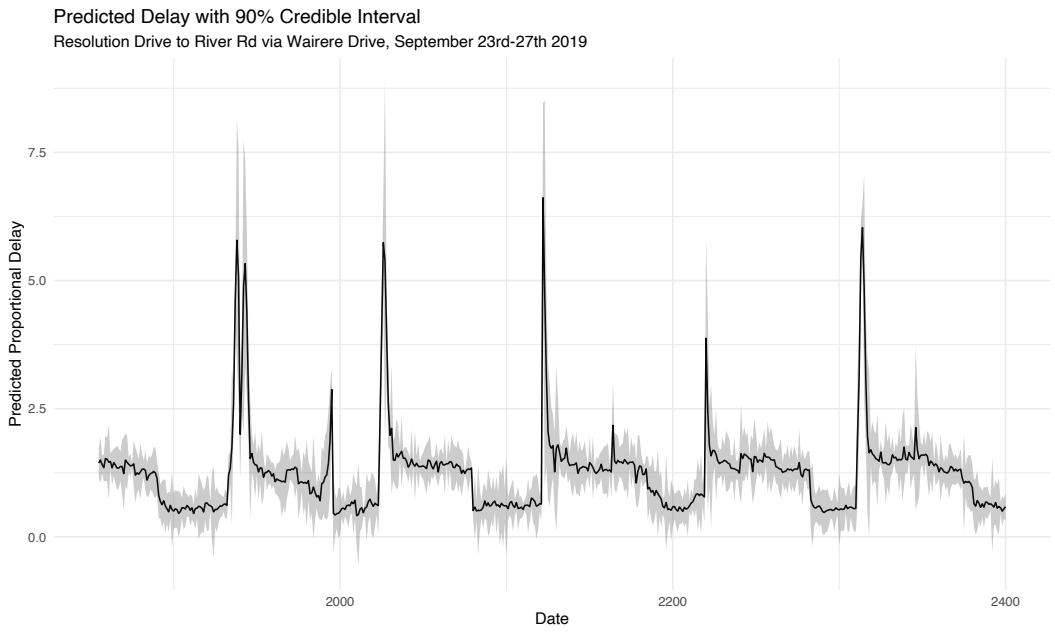


Figure 7.6: Predicted delay with 90% credible intervals shaded in grey

For any given link in the model, the upper and lower bounds at a specified credible interval can be obtained. Above is the 90% credible interval for Link 371 on Wairere Drive, shown in grey shading. This link typically experiences high delay in the school and PM peak. As seen above, the credible interval remains at a similar width throughout the day, indicating high prediction certainty even during peak periods. As the vast majority of traffic analysis is carried out for these peak periods, having a narrow credible interval allows for more certain decision making and accounting for possible variation around the model prediction.

7.3 Northern Model

Table 7.2 shows the prediction accuracy of links in the northern model, for links with missing spatial data at $k \in (1, 2, 3)$, against the full GC-LSTM model of all links in the network. There is a clear improvement in accuracy for these links across all time periods, further highlighting the benefits of the graph convolution operation.

Model	Peak	P
Northern	AM	77.4%
Northern	Inter	87.2%
Northern	PM	76.8%
Full	AM	**80.4%**
Full	Inter	**88.6%**
Full	PM	**79.7%**

Table 7.2: The P measure for the northern model and the full model, across periods of the day.

7.4 Hyperparameters

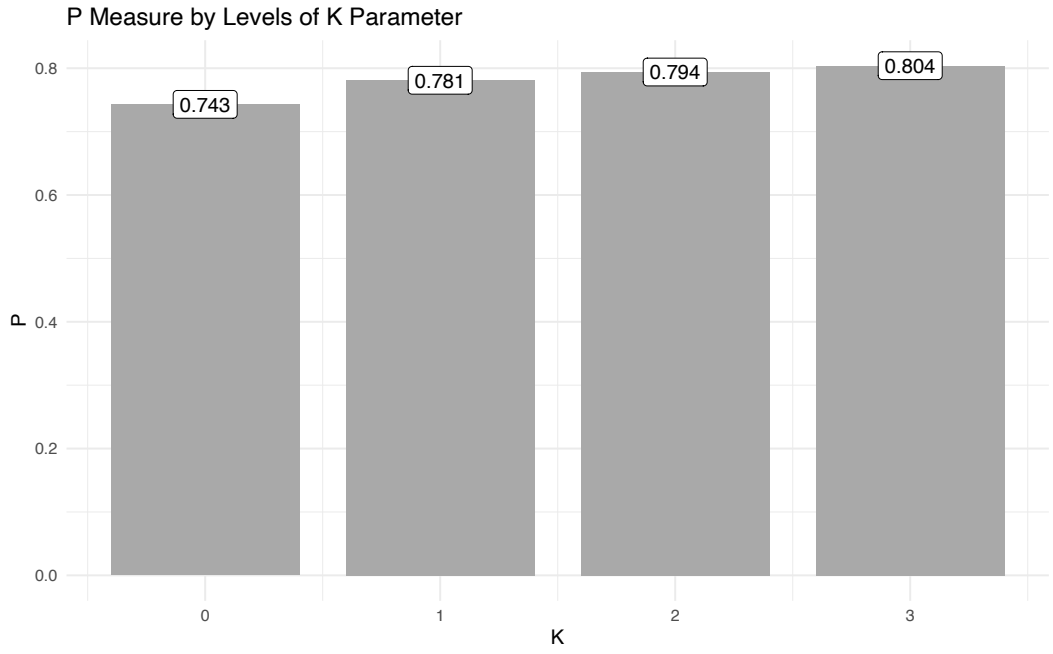


Figure 7.7: Model accuracy by levels of the K receptive field parameter. $K = 0$ is a standard LSTM.

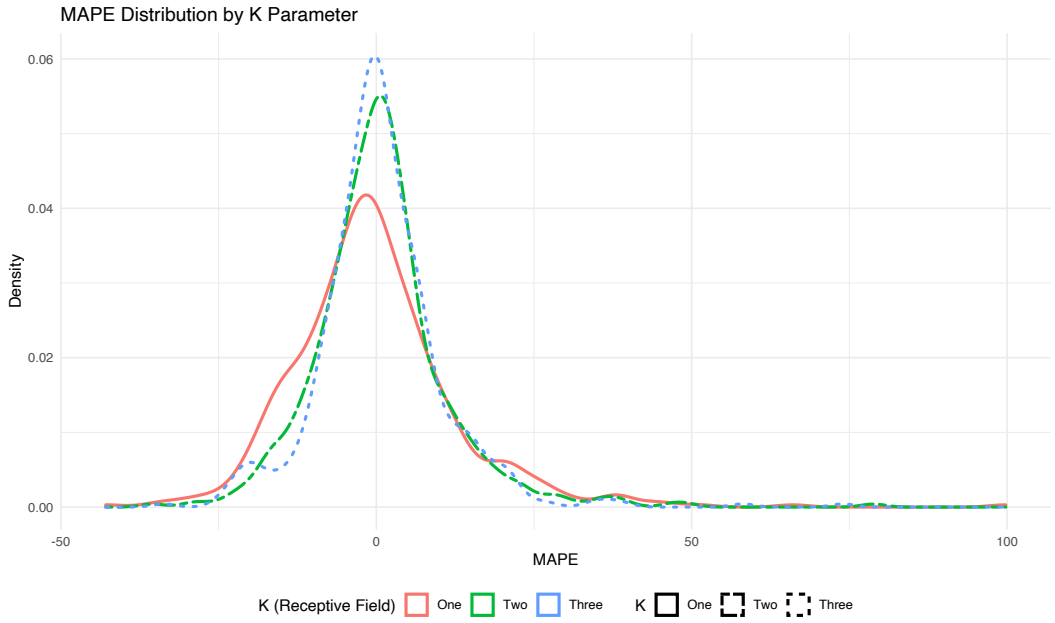


Figure 7.8: The MPE distribution by levels of the K parameter

Figure 7.7 (above) shows the change in P measured as the proportion of predictions within 10% of the observed delay, against varying levels of the K

parameter. K controls the receptive field in the model, and is equal to the number of hops from a given link that the model is using to calculate spatial weights. At $K = 1$, the model is using links directly connected to a given link. At $K = 2$, the model is also including links at 2 hops from the link, and so on. There is a large improvement in accuracy from one to two hops, and a smaller gain at three hops. Beyond three, there is negligible improvement in the model, and this is offset by a significantly longer running time. The number of links used to model spatial dependency for a given link increases exponentially with each increase in K, and therefore a balance is needed between model accuracy and the size of the receptive field.

Figure 7.8 (above) shows the distribution of the MPE measure across each link in the model. As K increases, the distribution decreases in variance and moves closer to a mode value of zero.

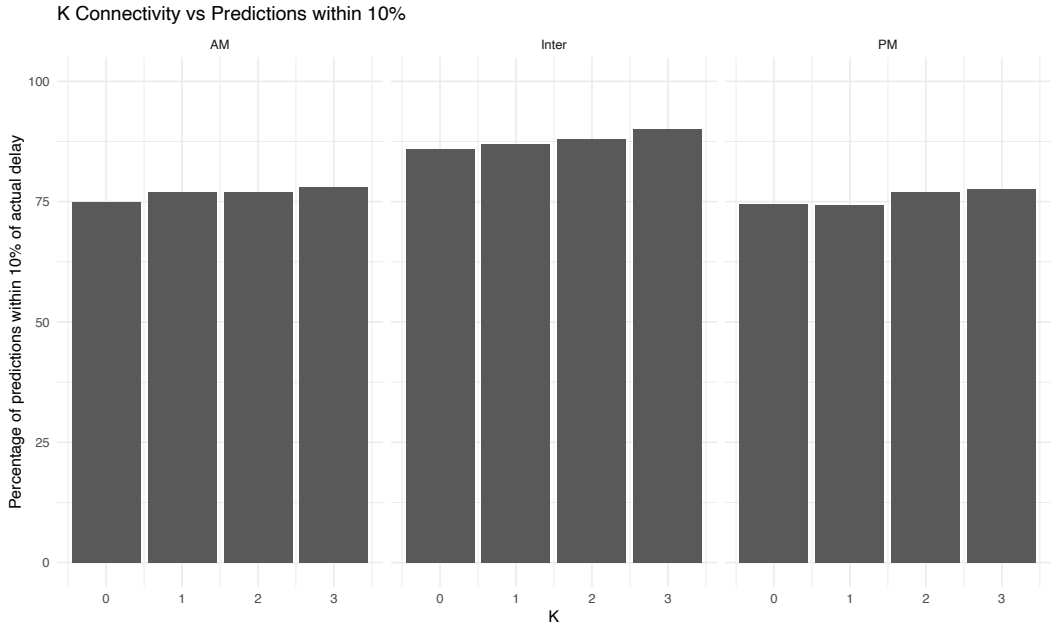


Figure 7.9: P viewed against k-connectivity and peak periods

Figure 7.9 shows P against levels of K-connectivity, where k = 0 indicating a link missing a direct upstream link, and k = 3 indicating a link missing an upstream link at 3 or more k-hops. The inter-peak period of 10-11am is the

most accurate of the three peaks, while AM and PM exhibit similar accuracy. As k-connectivity increases, the prediction accuracy slightly increases across all three peaks, further highlighting the benefits of the GC-LSTM model over the standard LSTM.

Iterations of the model were run from a batch size of 16 to 64. Smaller batch sizes have a longer running time than larger batch sizes, although they typically converge at a faster rate. Finding an optimal batch size must be a balance between convergence time and model running time, and can vary substantially between datasets and models. Figure 7.10 shows the convergence of rate of three batch sizes.

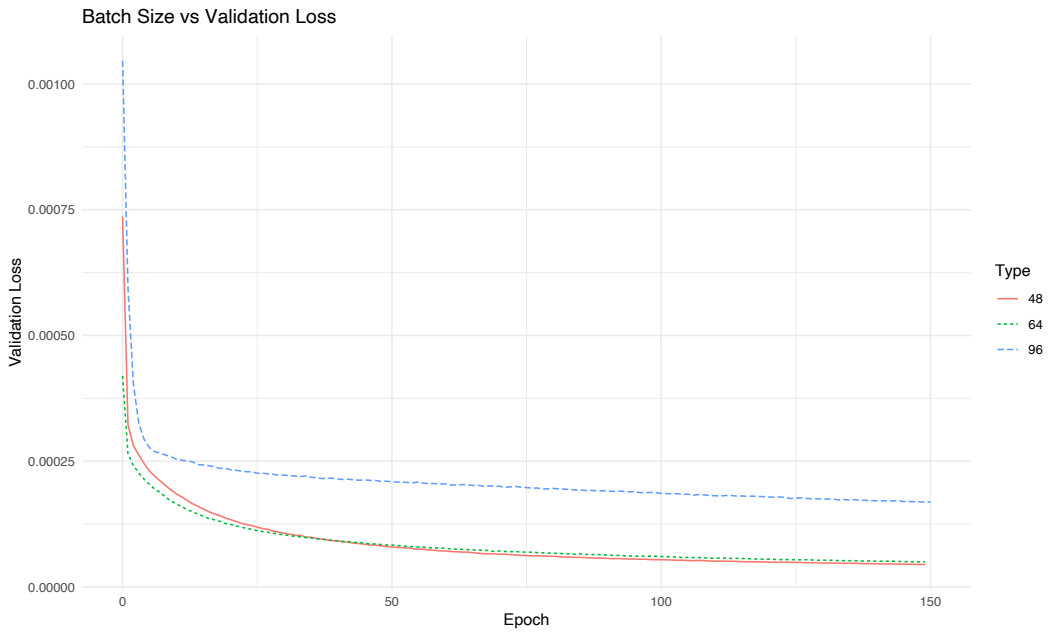


Figure 7.10: Validation loss against three batch sizes of 48, 64 and 96

Additionally, the K parameter to set the size of the receptive field also has a significant effect on model running time. A K value of three was determined to be the optimal level, as increasing it further has minimal effect on accuracy while substantially increasing the running time of the model.

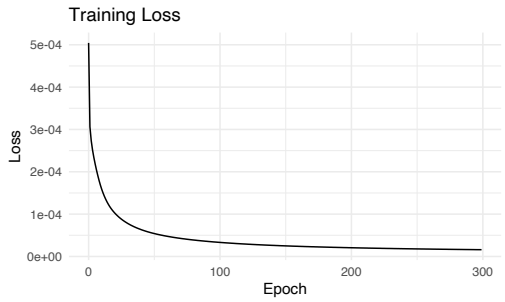


Figure 7.11: Training loss over 100 epochs

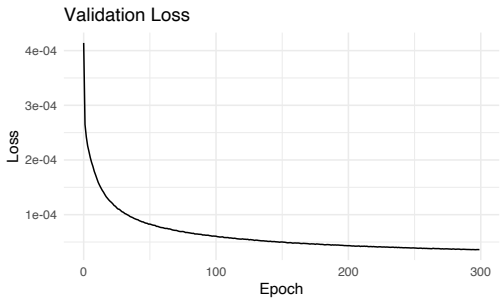


Figure 7.12: Validation loss over 100 epochs

Figures 7.11 and 7.12 show the validation loss of the model across 300 epochs. The model had total running time of 335 minutes or 5 1/2 hours. Increasing the size of the training dataset and reducing the batch size substantially increases the learning time but only marginally improves the test accuracy of the model.

7.4.1 Summary

The GC-LSTM model demonstrates a noticeable improvement in prediction accuracy over a standard LSTM. This highlights the strong spatial dependencies present in the dataset, and corresponds with the intuition of traffic flow propagation through the network. Links up to 3 hops from a given link have an influence on delay, with links further upstream having a negligible effect. The model is able to accurately capture high levels of delay in the AM and PM peak, while the LSTM model is unable to capture the full extent of the observed delay. In addition, periods of high delay which are not usually observed on a link are able to be captured through the spatial structure of the network, where this delay is first seen on links upstream in the network. A credible interval can be constructed through the integration of Bayesian layers in the network, which are able to produce probability distributions over each weight and subsequent prediction in the model.

Chapter 8

Summary & Future Work

8.1 Summary

There is a vast quantity of transportation data recorded by RCA's across all modes of transport. In particular, traffic delay and volume is continuously analysed in order to detect trends in traffic flow and identify areas of the network with high levels of delay. Due to the complex spatio-temporal interaction seen in traffic flow, models which take this interaction into account are best suited to modelling traffic volume and delay on the network. This is demonstrated by the use of a GC-LSTM model for Hamilton City to predict delay. The model shows a clear improvement over a standard LSTM, and is able to predict unforeseen changes in delay on links. Links up to three hops from a given link are shown to have an influence on delay. By modelling the network in a graph structure, these spatial dependencies can be captured and used in the model.

By capturing uncertainty in the model predictions, greater confidence can be given to decisions made from the model. Uncertainty can be captured in a deep learning model by estimating the true posterior of each weight in the network. This is done through the use of the Bayes by Backprop algorithm, which uses a form of variational inference to optimise a variational distribution $q(x)$ against

the true posterior, $p(x)$. By maximising the evidence lower bound (ELBO), an approximation to the posterior distribution is obtained. The result is a probability distribution on each prediction in the model calculated from the posterior distribution of weights in the model. A desired credible interval can then be obtained.

The resulting GC-LSTM model with Bayesian layers offers a robust model for predicting traffic delay in Hamilton City. The results can be analysed to gain a better understanding of how traffic delay propagates through the network, in addition to the spatio-temporal interactions present in the data. The probability distributions obtained on the predictions offer a level of confidence when making decisions based on the data, and as a result better outcomes for the community when planning infrastructure.

8.2 Imputation of Traffic Volume Data

The GC-LSTM framework can be extended to several models at HCC which would benefit from a combined spatio-temporal approach. One such model is in the imputation of traffic counts. At signalised intersections in Hamilton City, inductive sensors in each lane records vehicle counts, in addition to algorithms which control the timing of signal phases based on gaps between vehicles, volumes, and approach prioritisation settings. The sensors will go offline on occasion, resulting periods of missing data at random intervals and duration. The data from these sensors is used to quantify the level of traffic on roads and intersections, including the identification of peak periods, daily total volumes, and long term trends in traffic volume. When analysing data from these sensors it is important to be aware of missing data, as counts may be under-reported if a daily total is calculated with the presence of missing data. It is expected that the volumes recorded by other sensors at the same intersection are correlated to some degree to the sensor with missing data. Additionally, sensors at nearby

intersections may have some degree of correlation and can be used to inform the imputation of the missing data. Data from these sensors combined with the historical data from the sensor with missing data can be used to improve the current LSTM based model. An adjacency matrix can be constructed such that sensors directly upstream or downstream from sensors at adjacent sites can be marked as adjacent, as well as sensors at the same intersection. The spatial weights generated from the GC-LSTM model can be used to further inform traffic patterns on top of the historical time series data at the sensor being imputed. The non-linearity of this model and complexity of the data indicate that a deep learning model will be more appropriate over other spatio-temporal prediction models. The data is recorded in five minute intervals at over 1100 sensors at 113 intersections, resulting in millions of data points per day. Each sensor may potentially use over a hundred parameters which are correlated, as complex intersections have at least 24 sensors and up to 96 sensors at surrounding sites.

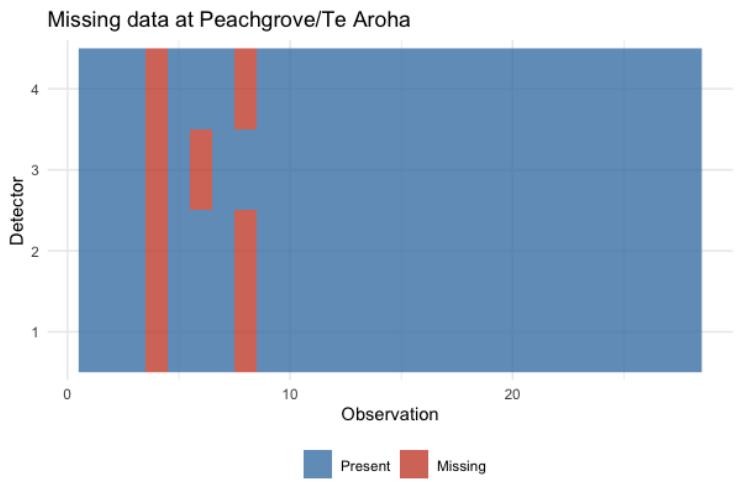


Figure 8.1: Periods of missing counts at an intersection in Hamilton, in five minute intervals

Figure 8.1 shows the periods of missing data from 5pm to 12am on one day at the Peachgrove/Te Aroha intersection. This intersection has 18 sensors, although sensors 5 to 18 had a complete dataset and thus have been left out of the chart for simplicity purposes. The second period of missing data at sensor

3 may be informed by the complete dataset at that time interval from the remaining detectors. The second period of missing data at sensors 1, 2 and 3 may be informed by the present data at detector 3 along with the remaining detectors 5 to 18. While the historical time series data at a given sensors may be sufficient in many cases to accurately impute the data, often the sensors will be offline for extended periods. In this case, an LSTM model is expected to underperform in comparison to a GC-LSTM model, which is able to utilise data from nearby sensors to enhance the prediction.

8.3 Prediction of the effect of road closures

Road closures are a recurring necessity in urban areas. Transportation infrastructure must be regularly maintained in order to keep a high standard of roads, footpaths and supporting infrastructure. It is often the case that a full road closure is the preferred option to minimise the duration of works and carry it out to an acceptable standard. When closures are carried out on vital roads in the network this has a significant flow on effect to surrounding roads [62]. In addition to the posted detour route, drivers will often seek out alternative routes if the closure is for an extended period of time. These alternative routes before the closure see a certain level of traffic volume and delay and often have little variation in these variables. A small change in volume can have a significant effect on resulting delay, and a further effect on delay for the remaining journey. Currently the effect of such infrastructure changes are determined by a land use transport model. However, this model is designed to work on a macro level and is only able to model large scale effects of new infrastructure. It also does not have the ability to use observed sensor data to capture what is currently happening on the network.

The ability to predict the change in traffic volume and delay and surrounding roads will be beneficial for HCC when planning road closures. For example, if

a certain increase in delay is expected on a road perpendicular to a planned closure (as it is likely a preferred alternative route), signals on that route can be adjusted in advance to prepare for the increased traffic volume. Determining the likely routes to be taken by vehicles in the presence of a road closure is often complex, as it is related to the destination of each individual's journey. People will often make a route choice based on a combination of expected traffic volume, delay, and the shortest distance to their destination. AddInsight data records individual vehicle journeys through the network including their origin and destination (to a block level). Given vehicles that travel through a certain section of road, the destination of these vehicles can be analysed and used as an input into the likely alternative routes. This data can also be used to plan the posted detour route. Given the ability of a GC-LSTM model to use data from upstream links to inform a given link's delay, this may extend to analysing the effects of closing a link (road). The direct upstream links to a closed link are expected to see a significant increase in delay. The volume from these links is now distributed across alternative downstream links to the one which is closed. The proportion of this traffic is often not shifted evenly to these links, and will vary by the time of day. If most traffic in the morning peak uses the closed link to travel into the CBD, and the CBD is located to the south-east of the link, the traffic is expected to shift to the links which will have less travel time to the CBD. Depending on the structure of the network, this may be the set of links directly east of the closed link, or it may be a different set of links with more capacity and higher speed limit. This complex interaction and shift in traffic volume may be modelled in a deep learning approach, with the spatio-temporal effects captured by a GC-LSTM model.

A future application of the GC-LSTM model described in this report will be adapting it to analyse and predict the effect of road closures. The model will be trained on the historical dataset of road and lane closures in order to capture the resulting change in delay on the network. The journey data will be

investigated in this process, and modelled in such a way that for a given link and time of day, the most common destinations can be captured and used as an input into the model. There are a large number of planned road closures in Hamilton City over the next few years, resulting from an increased focus in the long-term plan to shift mode share (the distribution of transport modes used for a journey) and increase maintenance on the road network. Several of these closures have resulted in negative feedback by the public, often alongside statements that sufficient analysis has not been carried out to quantify the effects of a closure [63], [62]. The ability to understand the likely increase in volume and subsequent delay on surrounding roads is hugely beneficial when consulting on such changes with the public. Additionally, the bayesian inference used in the delay prediction model can be applied to the road closure model to quantify the uncertainty on the predictions. Obtaining a probability distribution on traffic volume and delay change will allow staff to make decisions of road closures with more confidence, and allows for accounting any variance seen from the prediction following the closure. Currently, estimates on traffic volumes are provided as a single figure with no allowance for any variation around the estimate. A Bayesian model allows for the creation of a credible interval around each prediction and the formation of statements using a chosen probability percentage.

8.4 Mode Share Quantification

Waka Kotahi/NZTA has developed a plan to grow the usage of public transport, cycling and walking as alternative modes of transport [64]. This is known as *mode shift*. The percentages of journeys using a certain type of transportation is known as *mode share* [65]. Shifting mode share towards alternative modes (public transport, walking, cycling, and micro-mobility) has been passed on from Waka Kotahi to city councils to incorporate in their long term plans. As a result, shifting mode share is a priority for HCC and has been

incorporated in future transportation projects [66]. Cycleways, shared paths, bus lanes, and suitable walking infrastructure are now considered in all major infrastructure projects along with the ability to collect data on each mode to quantify the shift in mode choice. Two systems have been recently deployed to classify transportation modes on the network. The first, Briefcam, is able to classify different types of vehicles, pedestrians, and cyclists. The second, MAP counters, will be deployed roadside and on shared paths. They will count and classify pedestrians, cyclists, micro-mobility (scooters and skateboard), and wheelchair users. The rollout of these devices will result in over 100 devices on the network recording data on each transportation mode. This is in addition to SCATS detectors for recording vehicle volumes, AddInsight for recording delay and sample volumes, and before/after surveys done to support projects. This vast amount of data can be used to quantify mode share on the network, although this can only be done to a high level of accuracy in the locations where the devices are deployed. A GC-LSTM may be used to extend mode share estimation to parts of the network with only a subset of observed data. For example, areas of the transport network near those with high cyclist usage are expected to be correlated in some way, depending on the supporting infrastructure. Additionally, quantifying the uncertainty in mode share estimation is important when making decisions based on the data. A narrow credible interval allows for more confidence when reporting on observed mode share, and also allows for ruling out a non-significant change in the share of certain modes if a change is observed.

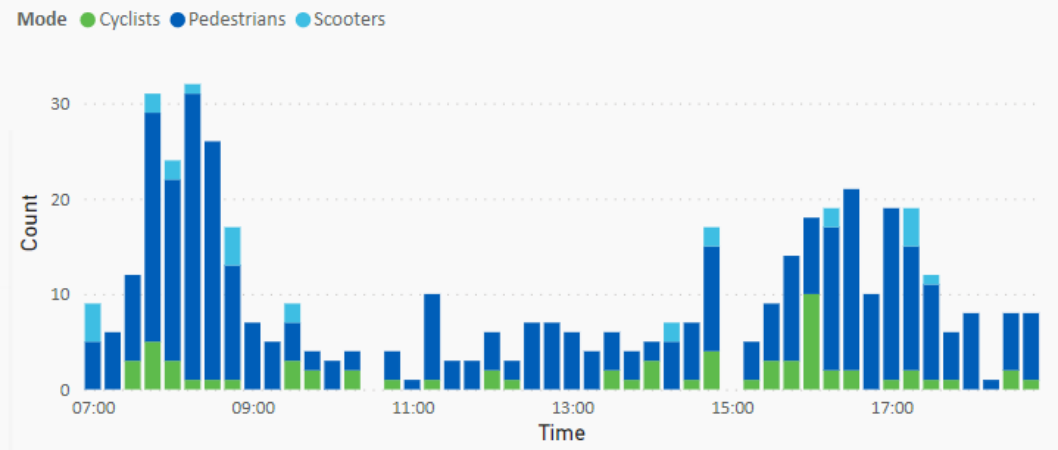


Figure 8.2: Mode share of active modes on Rostrevor St, during its closure in May and June

Figure 8.2 shows the mode share on a closed road in the Hamilton CBD over a Wednesday. The mode share for cyclists is highest during the morning and evening peaks, while pedestrians dominate at midday.

References

[1] Towards Data Science, "Perceptron learning algorithm: A graphical explanation of why it works," 2018. https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975.

[2] Srikumar, V., "The perceptron algorithm," 2018. https://www.cs.utah.edu/~zhe/pdf/lec-10-perceptron-upload.pdf.

[3] Bhande, A., "What is underfitting and overfitting in machine learning and how to deal with it," 2018. https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-w

[4] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition.* Newton: O'Reilly Media, Inc., 2019.

[5] Jordan, J., "Setting the learning rate of your neural network," 2018. https://www.jeremyjordan.me/nn-learning-rate/.

[6] Ojha, S., Santos, K., "Implementing parallelism using different architectures," https://shashank-ojha.github.io/ParallelGradientDescent/.

[7] Sreenivas, A., "Indian sign language communicator using convolutional neural network," 2020. https://www.researchgate.net/publication/343263135_Indian_Sign_Language_Communicator_Using_Convolutional_Neural_Network.

[8] Colah, "Understanding lstm networks," 2015. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[9] Dive into Deep Learning, "8.7. backpropagation through time," https://d2l.ai/chapter_recurrent-neural-networks/bptt.html.

[10] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," 2017.

[11] Real Statistics using Excel, "Graph theory," https://www.real-statistics.com/other-mathematical-topics/graph-theory/.

[12] Z. Cui, K. Henrickson, R. Ke, Z. Pu, and Y. Wang, "Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting," 2019.

[13] Hamilton City Council, "Transport choices and road safety," 2021. https://www.hamilton.govt.nz/our-services/transport/movingaround/Pages/default.aspx.

[14] Infometrics, "Hamilton city economic profile," 2020. https://ecoprofile.infometrics.co.nz/Hamilton%2BCity/Population/Growth.

[15] Hamilton City Council, "Annual report 2019/20," 2020. https://www.hamilton.govt.nz/our-council/council-publications/annualreport/Documents/Annual%20Report%202020-Final%20with%20initials.pdf.

[16] Wikipedia, "Level of service (transportation)," 2021. https://en.wikipedia.org/wiki/Level_of_service_(transportation).

[17] Hamilton City Council, "2021-31 long-term plan. (2021)," 2021. https://www.hamilton.govt.nz/our-council/10-year-plan/Pages/default.aspx.

[18] AddInsight, "Addinsight - traffic intelligence system," https://addinsight.com.au/.

[19] H. Rue, A. Riebler, S. H. Sørbye, J. B. Illian, D. P. Simpson, and F. K. Lindgren, "Bayesian computing with inla: A review," 2016.

[20] D. Townsend and C. Nel, "Traffic prediction at signalised intersections using integrated nested laplace approximation," 2021.

[21] D. Townsend, "Validation and inference of agent based models," 2021.

[22] L. Jospin, W. Buntine, F. Boussaid, H. Laga, and M. Bennamoun, "Hands-on bayesian neural networks – a tutorial for deep learning users," 07 2020.

[23] Wikipedia, "Transport network analysis," 2021. https://en.wikipedia.org/wiki/Transport_network_analysis.

[24] IBM, "Ibm - deep learning," https://www.ibm.com/cloud/learn/deep-learning.

[25] DeepAI., "Multilayer perceptron," 2018. https://deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron.

[26] Brooks-Bartlett, J., "Probability concepts explained: Maximum likelihood estimation," 2018. https://towardsdatascience.com/probability-concepts-explained-maximum-likelihood-estimation-c7b4342fdbb1.

[27] Michael A. Nielsen, "Neural networks and deep learning, determination press," 2015. Accessed: 2021-04-01.

[28] Towards Data Science, "Epochs vs batch size vs iterations," 2017. https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9.

[29] itdxer, "What is batch size in neural network? [online forum post]," 2018. https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network.

[30] Wikipedia, "Stochastic gradient descent," 2021. https://en.wikipedia.org/wiki/Stochastic_gradient_descent.

[31] Kathuria, A., "Intro to optimization in deep learning: Momentum, rmsprop and adam," 2018. https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/.

[32] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 2018.

[33] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017.

[34] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, 2017.

[35] Britz, D. , "Recurrent neural networks tutorial, part 3 – backpropagation through time and vanishing gradients," 2015. http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and

[36] NAP. , "Multimodal transportation planning data: Compendium of data collection practices and sources," 1997. https://www.nap.edu/read/6341/chapter/4.

[37] Waikato Regional Council, "Bee card - how it works," https://beecard.co.nz/Pages/HowItWorks.

[38] S. V. Kumar and L. Vanajakshi, "Short-term traffic flow prediction using seasonal ARIMA model with limited input data," *European Transport Research Review*, vol. 7, June 2015.

[39] Microsoft, "Forecasting with arima," https://appsource.microsoft.com/en-us/product/power-bi-visuals/wa104380888?tab=overview.

[40] Yin, X., Wu, G., Wei, J., Shen, Y., Qi, H., Yin, B. , "Deep learning on traffic prediction: Methods, analysis and future directions," 2020. https://arxiv.org/abs/2004.08555.

[41] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-rnn: Deep learning on spatio-temporal graphs," 2016.

[42] X. Cheng, R. Zhang, J. Zhou, and W. Xu, "Deeptransport: Learning spatial-temporal dependency for traffic condition forecasting," 2019.

[43] Eswarsai, "Exploring different types of lstms," 2021. https://medium.com/analytics-vidhya/exploring-different-types-of-lstms-6109bcb037c4.

[44] W. Jiang and J. Luo, "Graph neural network for traffic forecasting: A survey," 2021.

[45] Wikipedia, "Laplacian matrix," 2021. https://en.wikipedia.org/wiki/Laplacian_matrix.

[46] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.

[47] ERoughgarden, T., Valiant, G., "Cs168: The modern algorithmic toolbox lectures 11: Spectral graph theory, 1," 2021. https://web.stanford.edu/class/cs168/l/l11.pdf.

[48] Wikipedia, "Graph theory," 2021. https://en.wikipedia.org/wiki/Graph_theory.

[49] Gois, A., "Introduction to bayesian inference," 2020. https://towardsdatascience.com/introduction-to-bayesian-inference-18e55311a261.

[50] C. P. Robert, "The metropolis-hastings algorithm," 2016.

[51] T. Papamarkou, J. Hinkle, M. T. Young, and D. Womble, "Challenges in markov chain monte carlo for bayesian neural networks," 2021.

[52] Laumann, F., "What uncertainties tell you in bayesian neural networks," 2019. https://towardsdatascience.com/what-uncertainties-tell-you-in-bayesian-neural-networks-6fbd5f85648e.

[53] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," 2016.

[54] L. V. Jospin, W. Buntine, F. Boussaid, H. Laga, and M. Bennamoun, "Hands-on bayesian neural networks – a tutorial for deep learning users," 2020.

[55] C. Nemeth and P. Fearnhead, "Stochastic gradient markov chain monte carlo," 2019.

[56] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, p. 1613–1622, JMLR.org, 2015.

[57] Wikipedia, "Kl divergence," 2021. https://en.wikipedia.org/wiki/Kullback\OT1\textendashLeibler_divergence.

[58] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American Statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[59] piEsposito, "Blitz - bayesian layers in torch zoo," 2019. https://github.com/piEsposito/blitz-bayesian-deep-learning.

[60] NSW, "Traffic signal design," https://roads-waterways.transport.nsw.gov.au/business-industry/partners-suppliers/documents/guidelines/complementary-traffic-material/tsdsect11v13-i.pdf.

[61] Diamond Traffic Products., "Road tube," https://diamondtraffic.com/technicaldescription/117.

[62] Stuff, "Pedestrian friendly city street - genius or folly?," 2021. https://www.stuff.co.nz/national/124980332/pedestrianfriendly-city-street-closure--genius-or-folly.

[63] Stuff, "Five cross roads: Dozen of cross residents," 2021. https://www.stuff.co.nz/motoring/news/124889778/five-cross-roads-dozens-of-cross-residents-road-closure-possibility-for-ha

[64] NZTA, "Nzta - keep cities moving," 2020. https://www.nzta.govt.nz/walking-cycling-and-public-transport/keeping-cities-moving/.

[65] Wikipedia, "Modal share," 2021. https://en.wikipedia.org/wiki/Modal_share.

[66] Stuff, "Hamilton street to become cycle friendly," 2021. https://www.stuff.co.nz/waikato-times/news/125374725/student-highway-suburban-hamilton-street-to-become-cyclefriendly-in-34m-bi

Appendix

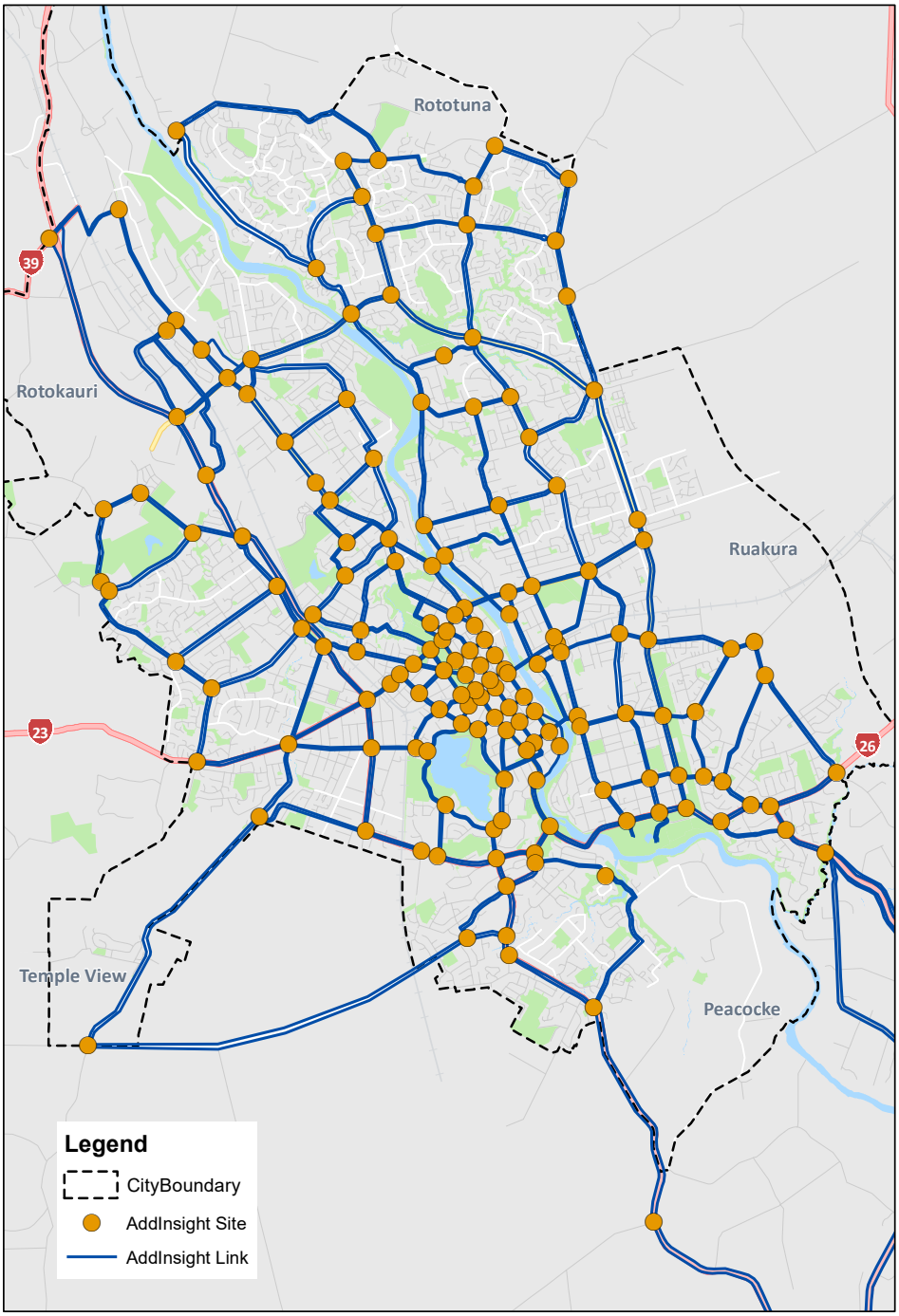


Figure 8.3: Complete map of the AddInsight Network within the Hamilton City boundary. Links are in dark blue, sites are in orange.