Spatio-Temporal Forecasting using Graph Gated Neural Networks

Arezou Amini

Department of Electrical & Computer Engineering, McGill University, Montréal

July 2021

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Science

ACKNOWLEDGEMENTS

I would like to appreciate everyone who has supported me during my Master's degree. In particular, I would like to express my gratitude toward my supervisor Professor Mark Coates. As his student, I was provided with multiple opportunities to learn and develop considerable research skills. I am also grateful for the insightful feedback and guidance he provided me throughout my studies. I thank Dr. Boris Oreshkin, for his collaboration and help to this work, and everyone from the computer networks research lab of McGill for fostering a supportive and stimulating environment. I also appreciate the support and guidance provided by the department of Electrical and Computer Engineering of McGill University. Lastly, I would like to thank my family especially my dear brother for all their help and encouragements.

ABSTRACT

Forecasting of multivariate time-series is an important problem that has potential applications in various domains such as intelligent transportation systems, energy and smart grids, and quantitative finance. A special case of the problem arises when there is a graph available that captures the relationships between the time-series. For this setting, the best existing approaches treat the problem as spatio-temporal forecasting, i.e., they strive to learn both temporal and spatial dependencies. In this work, a novel neural network learning architecture is presented that achieves performance competitive with or better than the best existing algorithms, without requiring knowledge of the graph. The key element of the architecture is the learnable fully connected hard graph gating mechanism that enables the use of a state-of-the-art, computationally efficient fully connected time-series forecasting architecture. Experiments conducted on two public graph-based datasets as well as two non-graph benchmark datasets illustrate the value of the proposed method, and ablation studies confirm the importance of each element of the architecture.

ABRÉGÉ

La prévision de séries chronologiques multivariées est un problème important qui a des applications potentielles dans divers domaines tels que les systèmes de transport intelligents, l'énergie et les réseaux intelligents, et la finance quantitative. Un cas particulier du problème survient lorsqu'il existe un graphe disponible qui capture les relations entre les séries chronologiques. Pour ce contexte, les meilleures approches existantes traitent le problème comme une prévision spatio-temporelle, c'est-à-dire qu'elles s'efforcent d'apprendre à la fois les dépendances temporelles et spatiales. Dans ce travail, une nouvelle architecture d'apprentissage par réseau de neurones est présentée qui permet d'obtenir des performances compétitives ou supérieures aux meilleurs algorithmes existants, sans nécessiter de connaissance sur le graphe. L'élément clé de l'architecture est le mécanisme d'apprentissage des portes d'un graphe entièrement connecté qui permet l'utilisation d'une architecture, à la pointe de la technologie et efficace en termes de calcul, entièrement connectée, pour la prévision de séries chronologiques. Des expériences menées sur deux ensembles de données publics basés sur des graphes ainsi que sur deux ensembles de données sans graphe pour référence illustrent la valeur de la méthode proposée, et les diverses expériences menées ont confirmé l'importance de chaque élément de l'architecture.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

Forecasting is among the oldest problems tackled by human civilization [2]. This task can be broadly related to self-supervised learning, also known as filling in the blanks, which is often considered the foundation of common sense and intelligent behaviour [3]. Therefore, improved forecasting algorithms have the potential to profoundly impact a wide array of fields in which planning and intelligent decision making are essential.

A multivariate time-series (TS) contains more than one time-dependent entity. The forecasting of future values for each entity depends on both its past values and other entities. In recent years, researchers have demonstrated the value of applying deep learning architectures for the multivariate TS problems [4–10]. Furthermore, many multivariate TS forecasting problems naturally admit a graphical model formulation. This is especially true when the entities whose past is observed and whose future has to be predicted affect each other through simple causal relationships. For example, introducing Pepsi products in a store will very likely decrease future sales of Coca-Cola in that store; car traffic congestion at one point on a highway is likely to slow down the traffic at preceding segments of the highway. Without graphical modeling, the model is blind to these nuances, making entity interactions a collection of confounding factors, extremely hard for the model to explain and predict. Equipped with a learnable model for entity properties (e.g., entity embeddings), a model for entity interactions (e.g., graph edge weights), and a mechanism to connect those to a TS model (e.g., a gating mechanism), we can attempt to learn the otherwise unknown entity interactions to improve forecasting accuracy.

Problems amenable to graphical TS modeling include forecasting demand for related products [11], electricity demand [12], road traffic [13], or passenger demand [14]. In this thesis, we focus first on forecasting road traffic, using a collection of sensors mounted on highways. This has obvious applications in traffic management and can have immediate positive effects through reducing CO_2 emissions as well as improving the life quality of the general population commuting to and from work every day. As the second application domain example, electricity demand is selected, which can be modelled using a spatio-temporal graph with nodes related via underlying dependent electricity consumption. The effective integration of green energy sources with the electrical grid depends on the accurate forecasting of short-term load and long-term demand. Tackling the electrical energy integration problem will have long ranging impact on making the world economy more sustainable and will help to address climate change.

Recent studies have shown that models that explicitly account for the underlying relationships across multiple time series outperform models that forecast each time series in isolation. As a result, the aim of this thesis is to tackle the spatio-temporal forecasting problem, which involves simultaneously forecasting multiple time series originating from entities related via an unknown underlying graph. Although the inclusion of graph modeling has proven to improve accuracy, current models have several serious limitations. First, the complexity and, therefore, the runtime of these models is significantly higher. Second, most models [14–17] rely on the definition of relationships between variables provided by a domain expert (e.g., an adjacency matrix is heuristically defined based on the geographical relationships between observed variables). Finally, existing models tend to rely on Markovian assumptions to make modeling the interactions across variables tractable.

To address these limitations we propose a novel architecture in this thesis, called FC-GAGA (Fully Connected Gated Graph Architecture), that is based on a combination of a fully-connected TS model, N-BEATS [1], and a graph gate mechanism. To produce the forecast for a single TS (node in the graphical model), it weighs the historical observations of all other nodes by learnable graph weights, gates them via a ReLU function and then stacks gated observations of all nodes to process them via fully connected residual blocks. The advantages of this architecture are threefold. First, the architecture does not rely on the knowledge of the underlying graph, and instead focuses on learning all the required non-linear predictive relationships. Second, the basic layer of the architecture is stackable, which allows every layer to learn its own graph structure. This endows the model with the ability to learn a very general non-Markovian information diffusion process. Finally, FC-GAGA is a very memory and computation efficient architecture, which we demonstrate via complexity analysis and profiling. Ablation studies indicate that when using the efficient fully-connected residual time-series prediction module, it is not sufficient to use standard graph attention since the sparsification achieved by the proposed novel graph gate is essential in achieving good predictive performance.

1.1 Thesis Organization and Contributions

The summary of the organization and contributions of the thesis is as following. The material presented in this thesis has been published in the Advancement of Artificial Intelligence Conference (AAAI), 2021 [18].

- *Chapter 2 - Background*

  Key material associated with the main topic of this thesis (spatio-temporal forecasting) and the proposed architecture is presented in the background chapter. I first provide a high-level overview of the most popular Graph Neural Networks that can be used to capture spatial dependencies. In the second part of the chapter, I present a summary of deep learning models that have been used for time-series prediction. Finally, I describe the computationally efficient N-BEATS model [1], because this model is exploited as the time-series block of the proposed architecture.

- *Chapter 3 - Literature Review*

  In this chapter, I review works that have addressed temporal or spatio-temporal forecasting. The literature review is divided into the following parts: (i) traditional and statistical approaches, (ii) deep learning based models that do not process an associated graph, and (iii) graph-based methods.

- *Chapter 4 - Fully Connected Gated Graph Architecture (FC-GAGA)*

  This chapter presents a novel graph gated neural network, called FC-GAGA, for spatio-temporal forecasting. The key element of the architecture is the learnable fully connected hard graph gating mechanism that enables the use of a state-of-the-art, computationally efficient fully connected time-series forecasting architecture. Experiments conducted on two public graph-based datasets and two non-graph benchmark datasets demonstrate performance competitive with or better than the best existing algorithms.

  I was a primary contributor in the development of the architecture and learning procedure, in collaboration with Dr. Boris Oreshkin (Researcher at ElementAI at the time when the research was conducted) and my supervisor, Prof. Mark Coates. I also conducted the majority of the numerical experiments. Some of the experiments were executed by Ms. Lucy Coyle, an undergraduate research intern working with Prof. Coates, and others were conducted by Dr. Oreshkin. I was co-first author (with Dr. Oreshkin) of the AAAI conference paper, in recognition of our joint effort in developing the methodology. Lucy Coyle and Mark Coates were co-authors.

- *Chapter 6 - Conclusion*

  This chapter summarizes the main contributions of the thesis and discusses the observed results.

CHAPTER 2
Background Material

This chapter presents the necessary background material for the spatio-temporal forecasting task: extraction of spatial dependencies and temporal prediction. Recently, Graph Neural Networks (GNNs) have been employed by multiple methods to capture spatial dependencies, so Section 2.1.1 is allocated to an overview of these models. The temporal prediction section commences with Feedforward Neural Networks (FNNs), which play an important role in the architecture proposed in this thesis. Subsequently, two popular variants of Recurrent Neural Networks (RNNs) are presented. Finally, we introduce a computationally efficient, fully connected time-series forecasting model called N-BEATS [1] in Section 2.2.3. This model forms the temporal component of our proposed architecture.

2.1 Extraction of Spatial Dependencies

2.1.1 Graph Neural Networks

Graphs are a general language for describing and modeling complex systems. Graph-structured data are irregular and non-euclidean, i.e., there is a variable number of unordered nodes, and each node may have a different number of neighbors. These features contrast with the core assumptions of most existing machine learning algorithms, which generally assume that instances are independent of each other, and often require that the data is Euclidean. Therefore, Graph Neural Networks

(GNNs) have emerged as an alternative solution. The key idea is to generate representations or embeddings of nodes that rely on the graph structure as well as any feature information [19]. In most of the GNN models, the adjacency matrix $\mathbf{A}^{N\times N}$ is used to describe the structure of a graph. The entries of this square matrix indicate whether pairs of vertices are adjacent or not in the graph. The entry has a non-zero value if the link is present in the graph; otherwise, the entry is 0.

One of the most commonly used GNN models is the Graph Convolutional Network (GCN) [20]. This has been used to extract the spatial features among the time series (nodes) in the spatio-temporal forecasting problem [15–17, 21–23]. This model treats the information of the neighbors and the information from the node itself in the same way. In a GCN, a layer-wise propagation rule consists of a weighted mean operator that determines the contribution of each neighbor to the node-level output, followed by multiplication by a learnable weight matrix. As the diagonal of an adjacency matrix $\mathbf{A}$ for a simple graph is zero, self-loops are introduced by adding the identity matrix to $\mathbf{A}$. This allows the neural network to take into account the feature of the node itself. The modified adjacency matrix is thus $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$. The weighted mean operator is based on the symmetric normalization of $\hat{\mathbf{A}}$, i.e., $\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}$ where $\hat{\mathbf{D}}$ is the diagonal node degree matrix of $\hat{\mathbf{A}}$. Consequently, the GCN computation at the ℓ-th layer having the learnable weight matrix $\mathbf{W}^{(\ell)}$ can be described as:

$$f\left(\mathbf{H}^{(\ell)}, \mathbf{A}\right) = \sigma\left(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(\ell)}\mathbf{W}^{(\ell)}\right). \tag{2.1}$$

2.2 Temporal Prediction

2.2.1 Feedforward Neural Networks

Feedforward networks are of extreme importance in machine learning since they are the stepping stone to developing more complex deep learning models, e.g., recurrent networks. A neural network model can be described as a directed acyclic graph comprised of many simple and connected processors called neurons. The neurons of middle layers (hidden layers) are activated via weighted connections from all neurons of the previous layer. For instance, a single hidden layer neural network with input layer of vector $\mathbf{x}$ and output layer with one neuron, $f(\mathbf{x})$, has the following formula:

$$\begin{aligned} \mathbf{h}^{(1)}(\mathbf{x}) &= g\left(\mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}\right), \\ f(\mathbf{x}) &= o\left(b^{(2)} + \mathbf{w}^{(2)^\top}\mathbf{h}^{(1)}(\mathbf{x})\right), \end{aligned} \tag{2.2}$$

where $\mathbf{W}^{(1)}$, $\mathbf{w}^{(2)}$, $\mathbf{b}^{(1)}$, and $b^{(2)}$ are the weights and biases of the network, respectively. g is the activation function of the hidden layer, and o is the output activation function.

2.2.2 Recurrent Neural Networks

Due to the limitations of traditional time series forecasting models, which often treat each time series independently and require manual feature selection, there has been a focus on RNNs, which are the extension of FNNs with feedback connections. The most effective RNNs are called gated RNNs. These include the Long Short-Term Memory (LSTM) [24] and the Gated Recurrent Unit (GRU) [25].

An LSTM network has internal memory allowing long-term dependencies to influence the output. The network consists of multiple gates and states. The input ($\mathbf{i}_t$), forget ($\mathbf{f}_t$), and output ($\mathbf{o}_t$) gates control the flow of information. The cell state

$\mathbf{c}_t$ maintains information about the input, and the hidden state $\mathbf{h}_t$ determines what passes through the output gate. The network can be described by the following equations:

$$
\begin{aligned}
\mathbf{i}^{(t)} &= \sigma\left(\mathbf{b}_i + \mathbf{U}_i\mathbf{h}^{(t-1)} + \mathbf{W}_i\mathbf{x}^{(t)}\right), \\
\mathbf{f}^{(t)} &= \sigma\left(\mathbf{b}_f + \mathbf{U}_f\mathbf{h}^{(t-1)} + \mathbf{W}_f\mathbf{x}^{(t)}\right), \\
\mathbf{o}^{(t)} &= \sigma\left(\mathbf{b}_o + \mathbf{U}_o\mathbf{h}^{(t-1)} + \mathbf{W}_o\mathbf{x}^{(t)}\right), \\
\widetilde{\mathbf{c}}_t &= tanh\left(\mathbf{b}_c + \mathbf{U}_c\mathbf{h}^{(t-1)} + \mathbf{W}_c\mathbf{x}^{(t)}\right), \\
\mathbf{c}^{(t)} &= \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \odot \widetilde{\mathbf{c}}^{(t)}, \\
\mathbf{h}^{(t)} &= \mathbf{o}^{(t)} \odot tanh\left(\mathbf{c}^{(t)}\right),
\end{aligned}
\tag{2.3}
$$

where $\mathbf{x}_t$ is the input at time step t, and $\mathbf{b}$, $\mathbf{U}$ and $\mathbf{W}$ respectively denote the biases, input weights, and recurrent weights in each gate. $\odot$ indicates the element-wise product.

A GRU network is designed to have fewer gates, thus reducing the number of learnable parameters. The main difference between these models is that the update gate ($\mathbf{u}_t$) in a GRU couples the forget and update gates of LSTM architecture. A GRU processes sequential input as follows:

$$
\begin{aligned}
\mathbf{u}^{(t)} &= \sigma\left(\mathbf{W}_u\mathbf{x}^{(t)} + \mathbf{V}_u\mathbf{s}^{(t-1)} + \mathbf{b}_u\right), \\
\mathbf{r}^{(t)} &= \sigma\left(\mathbf{W}_r\mathbf{x}^{(t)} + \mathbf{V}_r\mathbf{s}^{(t-1)} + \mathbf{b}_r\right), \\
\tilde{\mathbf{s}}^{(t)} &= tanh\left(\mathbf{W}_s\mathbf{x}^{(t)} + \mathbf{V}_s(\mathbf{r}^{(t)} \odot \mathbf{s}^{(t-1)}) + \mathbf{b}_s\right), \\
\mathbf{s}^{(t)} &= (1 - \mathbf{u}^{(t)}) \odot \mathbf{s}^{(t-1)} + \mathbf{u}_t \odot \tilde{\mathbf{s}}^{(t)},
\end{aligned}
\tag{2.4}
$$

where $\mathbf{u}_t$, $\mathbf{r}_t$ and $\tilde{\mathbf{s}}_t$ are the outputs of update gate, reset gate and pre-output gate respectively. $\mathbf{s}_{t-1}$ and $\mathbf{s}_t$ are the old and the new states. $\mathbf{W}$, $\mathbf{V}$, and $\mathbf{b}$ are the input weights, recurrent weights, and biases in each of the gates.

2.2.3 N-BEATS Architecture

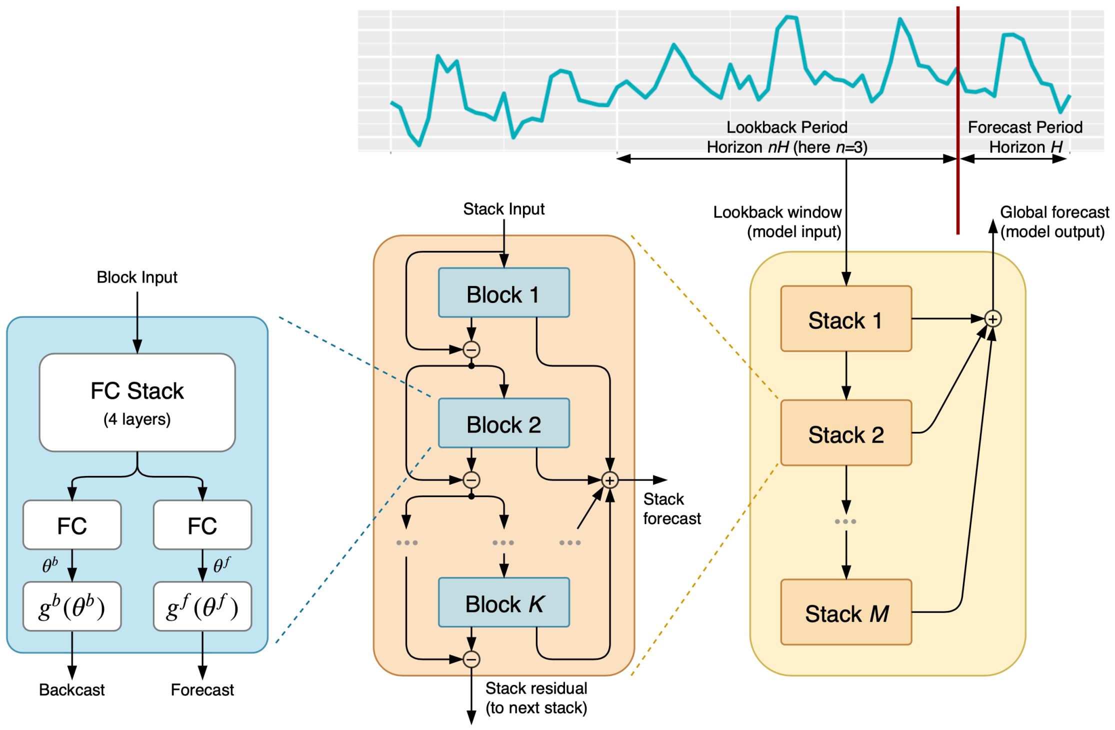


Figure 2–1: N-BEATS architecture taken from the original paper [1].

In this thesis, we take advantage of the N-BEATS architecture [1] in the time-series block of the proposed model. N-BEATS is one of the most recent literature on univariate time-series forecasting, which has achieved state-of-the-art performance on well-known forecasting datasets such as M3 [26], M4 [27] and Tourism [28] competition datasets including time series from diverse domains as well as Electricity [29] and Traffic [29] datasets, and it shows promising qualitative interpretability results.

This work demonstrates that a pure deep neural method can be effectively utilized in univariate time series forecasting compared to other machine learning approaches, which heavily rely on feature engineering or hybridization with classical statistical models.

The significant features of the N-BEATS design compared to previous architectures are threefold. First, the authors treat forecasting as a non-linear multivariate regression problem instead of a sequence-to-sequence problem. Therefore, the basic building block of the architecture, as shown in the left of Figure 2–1, is a fully-connected non-linear regressor that accepts the history of a time-series and predicts multiple points in the forecasting horizon. Second, a doubly residual principle is proposed to stack many basic blocks in a stack, as illustrated in the middle of Figure 2–1. To this end, the basic block generates both the future outputs, forecast, and contribution to the input decomposition, backcast. Finally, each stack can specialize in forecasting a particular type of output, e.g., trend or seasonality, making the output human interpretable. Due to the mentioned characteristics, we take advantage of the N-BEATS architecture to capture temporal correlations for the multivariate forecasting method proposed in this thesis.

With regard to the mathematical description, each block of N-BEATS (consider l as the block index) consists of a 4-layer fully connected network, which is followed by θ_l^b and θ_l^f projection layers, building a backcast/forecast (g_l^b/ g_l^f) fork at the end.

The equations of the basic block can be described as:

$$
\begin{aligned}
&\mathbf{H}_{\ell,1} = \mathrm{ReLU}\left(\mathbf{W}_{\ell,1}\mathbf{X}_{\ell} + \mathbf{b}_{\ell,1}\right),\ \mathbf{H}_{\ell,2} = \mathrm{ReLU}\left(\mathbf{W}_{\ell,2}\mathbf{H}_{\ell,1} + \mathbf{b}_{\ell,2}\right),\\
&\mathbf{H}_{\ell,3} = \mathrm{ReLU}\left(\mathbf{W}_{\ell,3}\mathbf{H}_{\ell,2} + \mathbf{b}_{\ell,3}\right),\ \mathbf{H}_{\ell,4} = \mathrm{ReLU}\left(\mathbf{W}_{\ell,4}\mathbf{H}_{\ell,3} + \mathbf{b}_{\ell,4}\right),\\
&\quad\theta_{\ell}^{b} = \mathbf{W}_{\ell}^{b}\mathbf{H}_{\ell,4},\ \theta_{\ell}^{f} = \mathbf{W}_{\ell}^{f}\mathbf{H}_{\ell,4},\\
&\quad\widehat{\mathbf{X}}_{\ell} = g_{\ell}^{b}\left(\theta_{\ell}^{b}\right),\ \widehat{\mathbf{Y}}_{\ell} = g_{\ell}^{f}\left(\theta_{\ell}^{f}\right),
\end{aligned}
\tag{2.5}
$$

where $\mathbf{X}_{\ell}$ is the input of ℓ-th block. $\widehat{\mathbf{X}}$ (backcast) and $\widehat{\mathbf{Y}}$ (forecast) are the outputs of the ℓ-th block. $\mathbf{W}_{\ell,1}$, $\mathbf{W}_{\ell,2}$, $\mathbf{W}_{\ell,3}$, $\mathbf{W}_{\ell,4}$, $\mathbf{W}_{\ell}^{b}$, $\mathbf{W}_{\ell}^{f}$, $\mathbf{b}_{\ell,1}$, $\mathbf{b}_{\ell,2}$, $\mathbf{b}_{\ell,3}$ and $\mathbf{b}_{\ell,4}$ are the learnable parameters of the network. In this thesis, we use the generic architecture of N-BEATS, in which g_{l}^{b} and g_{l}^{f} are linear projections of the previous layer outputs. As a result, the outputs of the ℓ-th block can be written as:

$$
\widehat{\mathbf{X}}_{\ell} = \mathbf{V}_{\ell}^{b}\theta_{\ell}^{b} + \mathbf{b}_{\ell}^{b},\ \widehat{\mathbf{Y}}_{\ell} = \mathbf{V}_{\ell}^{f}\theta_{\ell}^{f} + \mathbf{b}_{\ell}^{f}. \tag{2.6}
$$

The interpretability of the architecture can be enforced by considering the outputs of the block's penultimate layer as basis expansion coefficients θ_{ℓ}^{b} and θ_{ℓ}^{f} for the bases $\mathbf{V}_{\ell}^{b}$ and $\mathbf{V}_{\ell}^{f}$, respectively. These bases are learned through the network. In the mentioned generic configuration, the learned basis $\mathbf{V}_{\ell}^{f}$ does not have a human interpretable form. In contrast, the proposed interpretable configuration constrains the final layer of the block to be a fixed polynomial basis, or a fixed Fourier basis. This method constrains the outputs of each stack to appear as a trend or a repetitive pattern, a seasonality, providing the forecast with human interpretable components at the stack level.

Each stack of N-BEATS employs two residual recursions: (i) running over the entire input window (the backcast branch of each block) as described in Equation (2.7), which facilitates gradient backpropagation; and (ii) summing the forecast branch of blocks to generate its partial forecast as written in Equation (2.8).

$$\mathbf{X}_\ell = \mathbf{X}_{\ell-1} - \widehat{\mathbf{X}}_{\ell-1}. \tag{2.7}$$

$$\widehat{\mathbf{Y}} = \sum_{\ell} \widehat{\mathbf{Y}}_\ell. \tag{2.8}$$

The input of the first block is the model level input and $\widehat{\mathbf{X}}_0 \equiv \mathbf{0}$. For all other blocks, the input is the corresponding backcast $\mathbf{X}_\ell$ which can be considered as a sequential analysis of the input signal [1]. This design yields a hierarchical decomposition via aggregating the partial forecast of each block $\widehat{\mathbf{Y}}_\ell$ at the stack level as well as at the overall network level. Finally, the model output is the sum of all partial forecasts coming from the stack modules.

2.3 Summary

This chapter provides the fundamental material for the spatio-temporal forecasting topic. The following chapter will review algorithms that have been proposed to tackle the time-series forecasting problem, including traditional time-series methods, deep learning forecasting models, and spatio-temporal approaches.

CHAPTER 3
Literature Review

This chapter provides the literature review for time-series (TS) forecasting. First we introduce univariate TS statistical models, which consider each time-series as a single time-dependent variable. Subsequently we discuss multivariate time-series models, which strive to capture the underlying relationships across multiple TS. The idea of inferring a causal graphical model has also been explored as a mechanism for identifying the relationships between different TS. We summarize these classical and statistical methods in Section 3.1.

In recent studies, architectures utilizing the power of deep neural networks have been able to outperform state-of-the-art statistical methods [1,4–8,30–32]. We review these models in Section 3.2. In some settings, we are provided with a graph that is thought to extract the relationships between the time series. There has been an intensive research effort to derive architectures that can use this graph information to perform improved forecasting. Section 3.3 presents these graph-based models. The literature review provides more detail for the models that are more closely related to the methodology proposed in this thesis.

3.1 Statistical Approaches

Time-series forecasting has a very long history, so there are numerous approaches, many based on statistical models. Standard univariate methods include the

Auto-Regressive (AR) [33] and Auto-Regressive Integrated Moving Average (ARIMA) models [34]. The extension to the multivariate case leads to the Vector Auto-Regressive (VAR) and Vector Auto-Regressive Moving Average (VARMA) models [34]. One of the most comprehensive and accurate statistical models is Prophet [35], which includes components capturing the time-series trend, the seasonality, and special events, e.g., holidays. However, this model is vulnerable to a low number of observation data.

The graphical model formulation for capturing the relationships between the different variables can be used to derive sparse VAR predictive models [36–39] which enjoy better performance. However, as the linearity of these models can impede forecasting accuracy, kernel versions have been introduced [40, 41]. Although these architectures can outperform the neural network methods discussed below for shorter time-series, selecting appropriate kernels is a challenge.

In some settings, a graph that potentially captures the relationships is provided together with the TS. The graph VAR(MA) proposed in [42] is a linear model where the VAR(MA) coefficients are specified in terms of the Laplacian of the provided graph.

3.2 Neural Network Approaches without Graph

Historically, neural network approaches have struggled to compete in terms of prediction accuracy with state-of-the-art statistical forecasting models such as Prophet [35]. This has changed over the past few years. Several neural network architectures reviewed in this section, if trained on many time series, have eclipsed statistical approaches. The algorithms reviewed in this section do not consider the knowledge of any graph structure to generate predictions.

3.2.1 Point Forecasting Approaches

Some architectures form predictions for a single target TS based on its past history (and covariates) including [1, 30–32]. In contrast to the proposed method in this thesis, these architectures do not simultaneously form forecasts for multiple time series using past information from all of them. Smyl et al. [32] was the winner of the M4 competition by introducing a hybrid model which combines a state-space exponential smooth model (statistical model) with LSTM networks (deep learning model). The state-space model extracts the local information from an individual TS and the LSTM networks capture global information from all series. The main disadvantage of this model is that it must be hand-crafted for each specific forecasting horizon and dataset; therefore, this approach is hard to generalize. As a result, the Neural Basis Expansion Analysis for Interpretable Time Series (N-BEATS) model was proposed in [1] to show that a pure deep learning model is able to achieve promising results on a wide range of TS forecasting tasks without using any statistical approaches, hand-crafted feature engineering, or domain knowledge.

Other neural network based methods use multiple input time-series to predict a single variable [4–8]. For these architectures, several innovations have proven effective, including denoising via transforms e.g., wavelet and empirical mode decomposition, or stacked autoencoders [4, 10], employing Convolutional Neural Networks (CNNs) to extract short-term patterns [6], introducing memory components, implemented via LSTM encoders [8], and using attention mechanisms to determine which input variables (spatial attention) or time lags (temporal attention) to focus on [5, 7, 9, 10]. In the following, more details about these methods are provided.

Even though RNN based models can effectively extract temporal features, they have limitations when they are used to try to capture dependencies among multiple variables. Hence, there have been various attempts to solve this problem and improve performance. For instance, Bao et al. [4] propose a hybrid architecture that combines more than one deep learning model, and incorporates wavelet transforms, stacked auto-encoders, and LSTM networks, to perform prediction. The stacked autoencoder effectively extracts factors from the time series, and these are subsequently denoised by wavelet transforms and then fed into an LSTM. The results showed the superiority of the hybrid method over single RNN based models like the LSTM. Many works, including [10], are based on the ensemble decomposition framework to simplify the complex data. This results in capturing internal factors and improving forecasting accuracy. In [10] a prediction framework is presented that includes a stacked residual LSTM encoder-decoder with an attention module. This model design incorporates Ensemble Empirical Mode Decomposition (EEMD) due to its self-adaptability and time-frequency resolution features. K-means clustering is also used to reduce the number of input time series. Although this model is designed for the multivariate forecasting task, it focuses on the local past data of a single variable during prediction. Moreover, the proposed framework is not end-to-end since the decomposition and prediction steps are separate.

RNN-based approaches are essentially black-box methods with little interpretability. Moreover, they need much more data to fit reliably due to having many network parameters. They tend to exhibit poor performance in high-dimensional settings. Tank et al. [43] aim to overcome these deficiencies. Their proposed framework employs component wise architectures for multivariate time series prediction, training separate Multilayer Perceptrons (MLPs) or different variants of RNNs for each output time-series. By imposing group sparsity penalties on the first-layer weights, it becomes possible to estimate non-linear Granger Causality (GC) relationships between the time-series. This approach provides a more interpretable model, although it focuses primarily on relational inference and does not efficiently interpret the variability of causality effects through time.

As the typical RNN based approaches struggle to model seasonality patterns effectively, practitioners have used CNNs to capture local dependencies. For example, Lai et al. [6] develop a multivariate forecasting approach that uses a CNN to learn short-term dependencies between variables and then an RNN with skip connections, called a recurrent-skip layer, to capture long-term patterns in the time-series. The first layer is a convolutional component that learns the parameters of a set of filters across the entire set of time-series. The proposed method can accept multiple time-series in the input layer which allows the model to capture global properties during both training and prediction. However, this method has a scalability problem because of the growing size of the input layer. To tackle the scalability issue, an alternative model is proposed in [5], which substitutes the recurrent-skip layer with a temporal attention layer. Qin et al. [5] employ two time-varying attention modules. One module is applied on the input side, determining which input series to focus on; the other module, on the decoder side, determines which time lags (temporal hidden states of the encoder) are important. The temporal attention is point-wise attention which might not be suitable for modelling continuous periodical patterns. To address this problem, in [8], Chang et al. propose a Memory Time-series Network (MTNet). The encoder architecture involves applying a convolutional layer to extract short-term features, an attention mechanism to select among the features, and a GRU. Two such encoders are used to implement a memory, allowing the architecture to learn and store very long-term historical patterns. A third encoder is used to process the short-term history. Finally, outputs from the memory unit and the short-term encoder are combined to form the prediction. An advantage of this model is that it is more interpretable owing to its block-wise attention, although it requires a large amount of memory to detect abrupt events.

The attention mechanism in TS forecasting models is mainly applied to hidden states across time steps, as in [5], allowing the model to focus on capturing temporally important information. However this approach fails to consider the different importance of input variables. To handle this issue, Guo and Lin [7] propose an approach that is based on a multivariable LSTM with tensorized hidden states. Temporal and variable attention is derived from the hidden states. This mimics a probabilistic mixture of experts model, where each variable and lag corresponds to an expert, and the weighting of the experts is derived via the attention mechanism. In [9], in order to demonstrate the different importance of input variables and lags, Munkhdalai et al. append an adaptive input selection architectural component to LSTM or GRU networks for multivariate forecasting. An encoding network (RNN) is used to encode each time series and lag.

Unlike the RNN-based methods, the Transformer [44] model has access to any part of the history regardless of distance, making it potentially more suitable for extracting patterns with long-term dependencies. Moreover, the Transformer does not need to take the sequential data recursively; therefore, the model can be parallelized, which makes the training and prediction process more efficient. However, in the Transformer model, scaling attention to extremely long sequences is computationally expensive since the complexity of self-attention is quadratic with respect to sequence length. This poses a serious problem in forecasting time series that have strong long-term dependencies. In this vein, a modification of the original Transformer architecture is designed in [45] to address TS forecasting. Li et al. [45] propose the use of convolutional self-attention to leverage local context and LogSparse self-attention to enable learning from a long history without excessive memory usage.

Since the existing temporal Matrix Factorization (MF) approaches like [46] use graph regularization to incorporate temporal dependencies, which cannot support negatively correlated dependencies, the Temporal Regularized Matrix Factorization framework (TRMF) [47] is introduced to extend graph-based regularization approaches. This method aims to integrate temporal dependencies and matrix factorization models. It uses a principled time-series model to formulate relationships between latent temporal embeddings. These temporal dependencies are integrated into the standard matrix factorization approach by designing a new temporal regularizer. The proposed temporal dependency learning is data-driven since the strength of dependencies is learned from data. By taking advantage of MF properties, the

TRMF model is capable of imputing missing values to handle data noise and is scalable to high-dimensional time series datasets. Additionally, this model has better forecasting results since it can support negative correlations, and the weights can be automatically learned from data. However, the TRMF method can only formulate linear temporal dependencies. During prediction, the model only takes into account the global patterns, and this can lead to approximation errors [48].

In contrast to most deep learning models which only focus on an individual time-series during prediction, such as [30, 31], a hybrid model, called DeepGLO, is proposed in [48] that exploits both local and global patterns during training and prediction. DeepGLO combines a global model and a local prediction. The global model is formed using a matrix factorization model that is regularized by a temporal convolution network. The local prediction is then constructed by utilizing the factors derived from the global model as covariates in a Temporal Convolution Network (TCN) [49]. Although the authors argue that DeepGLO can deal with the problem of variation in scale, the proposed model cannot completely solve this problem since the proposed Deep Leveled Network only separates the mean and the residual of the time series. However, the mean and the residual scale can still be large, which makes a problem for neural networks.

3.2.2 Probabilistic Forecasting Methods

Although the approaches mentioned so far provide promising results in point forecasting, these models have a severe disadvantage as they cannot measure the uncertainty in their predictions. The availability of confidence intervals is useful when making decisions based on forecasting. Therefore, several powerful multivariate forecasting algorithms that are capable of providing uncertainty characterization have been proposed. These include DeepAR [31], DeepState [30], DeepFactors [50], and a hybrid model [32].

DeepAR is a probabilistic forecasting model proposed in [31] that is inspired by an autoregressive recurrent network. The model distribution is parameterized by a multi-layer LSTM network which takes covariates of the target TS and the ground truth value at the previous time step as inputs. By incorporating adaptive likelihood functions based on the statistical properties of the data, the DeepAR model is able to be data flexible. This model requires minimal feature engineering, as it learns seasonal behaviour across time series based on the provided time covariates. One of the main drawbacks of DeepAR is that as the scale of each time series is pre-specified in the model, it cannot properly handle the variation scales problem.

In [50], Wang et al. propose a more general approach called DeepFactors. DeepAR is a specific version of the DeepFactors framework. This method is a local–global method that obtains lower uncertainty and more accurate predictions compared to the DeepAR approach. DeepFactors is based on the decomposition of time series into global and local parts. RNNs are used to extract global dynamics, and the local component consists of a probabilistic graphical model that captures the uncertainty of each individual time series. The local models explored in [50] are white noise processes, linear dynamical systems, and Gaussian processes. In contrast to the DeepAR model, the scale of each time series is automatically estimated in DeepFactors.

Rangapuram et al. [30] present the DeepState model, which is the fusion of a linear state-space model and an RNN. This multivariate forecasting model uses a multi-layer RNN with LSTM cells for mapping from the covariates related to a target TS to the time-dependent parameters of a linear state-space model. This method provides model interpretability, can exploit assumptions about temporal smoothness, and exhibits scalability to high-dimensional datasets. However, the DeepState method can only model a Gaussian likelihood with time varying state-space model parameters for each time series.

All of the above-mentioned deep learning based forecasting algorithms do not account for the spatial relationships in the data. The probabilistic forecasting methods, such as the models in [31,50], mainly focus on modeling the forecasting distributions and do not perform as well in the point forecasting task.

3.3 Graph-based Methods

Most of the existing graph-based approaches require a pre-defined graph, although Graph Wavenet [22], Adaptive Graph Convolutional Recurrent Network (AGCRN) [51], and Structure Learning Convolutional Neural Network (SLCNN) [52] have the capacity to learn adjacency matrices from the temporal data, and the method in [53] uses dynamic time warping as a pre-processing step to learn a graph. In the following discussion, we categorize methods based on their architectures, including (i) Combination of GNNs and RNNs, (ii) Using Attention Mechanisms, (iii) Adaptive Graphs, and (iv) Advanced Architectures.

3.3.1 Combination of GNNs and RNNs

Zhao et al. propose a Temporal Graph Convolutional Network (T-GCN) in [54] that combines GRU networks and GCNs by replacing the feedforward neural networks inside the standard GRU with the graph convolutional network. Following the method proposed in [54], a more advanced architecture called the Diffusion Convolutional Recurrent Neural Network (DCRNN) is introduced in [15], which was a breakthrough in the traffic forecasting task. This architecture combines a diffusion graph convolutional network using a bidirectional graph random walk and a sequence-to-sequence encoder-decoder with scheduled sampling [55] to fuse spatial and temporal information. The matrix multiplications in each GRU cell are replaced by graph diffusion convolutions, leading to a Diffusion Convolutional Gated Recurrent Unit (DCGRU). Huang et al. [56] extend the DCRNN model to include more flexible spatial aggregation, using a rank influence learning mechanism similar to a graph attention network. The rank influence learning provides a more interpretable output with a tolerable runtime increase. One potential problem with the encoder-decoder structure used in [15, 56] is that since the latent space vector has a fixed size regardless of the length of the input and output sequences, for long-term input sequences, some information will be lost.

Since RNN-based networks are widely known to be hard to train and computationally burdensome, the Spatio-Temporal Graph Convolutional Network (STGCN) of [16] uses a CNN on the time axis to learn temporal relationships. Additionally, it employs the ChebNet graph convolution network to learn spatial relationships. Chen et al. [57] adapt residual-shortcut structures, gates, and hop connections, and thus achieve improved predictive performance. The hop connections allow learning of daily and weekly periodic behaviour.

3.3.2 Using Attention Mechanisms

The RNN models discussed thus far struggle with long-term temporal dependencies. An important extension is to use an attention mechanism on the time axis which enables the model to efficiently handle the long-term sequence data. To further improve the performance of the STGCN model [16], the Attention Based Spatial-Temporal Graph Convolutional Network (ASTGCN) is proposed in [21]. This method incorporates three components to capture recent dependencies, daily seasonality, and weekly seasonality. Each component includes spatial and temporal attention mechanisms followed by a graph convolution to capture the spatial dependencies and a convolution to extract the temporal relationships. The Spatial-Temporal Graph to Sequence (STG2Seq) model in [14] combines a long-term encoder and a short-term encoder, both based on gated graph convolutional modules (GGCMs), where the outputs of multiple GCNs, operating on different time windows, are passed through gates with learnable parameters. At the output, there are attention mechanisms operating both temporally and per-channel. The proposed stacked gating mechanism of [14] is more effective than an RNN, as it prevents vanishing gradients and the accumulation of errors in the prediction process to a certain extent; also, it enjoys faster training speed. The Graph Multi-Attention Network (GMAN) [17] incorporates multiple spatio-temporal attention blocks to provide greater flexibility inside an encoder-decoder model. Moreover, a transform attention layer is used to convey the learned features to the decoder. GMAN heavily depends on a pre-defined graph as it takes advantage of the node2vec algorithm [58] to preserve node structural information while performing attention mechanisms.

None of the mentioned approaches consider both dynamically adjusting attention weights and the graph structure information. The Spatio-Temporal Graph Attention Network (STGRAT) architecture [59] uses more sophisticated attention mechanisms and achieves improved prediction performance. The proposed approach is an encoder-decoder model using the positional encoding method of the Transformer [44] to capture features of long sequences and node attention to capture spatial correlations. Shi et al. [13] present an alternative attention-based framework that allows the encoder-decoder architecture to focus on temporal, spatial, and periodic correlations. A recurrent structure with temporal skip-connections allows long-term dependencies to be incorporated in the predictions. The spatial and periodic dependencies are captured through spatial attention in the encoder, and the decoder extracts the temporal correlations via temporal attention. However, one potential issue of the proposed model is that there are considerably more parameters than competing methods like ASTGCN [21]. As a result, this model usually requires more data before it exhibits improved performance.

One of the significant disadvantages of the above-mentioned approaches is that the performance of these models heavily relies on how the graph is defined, which requires domain knowledge. The pre-defined graph structures are generally fixed. In a spatio-temporal problem, it is often the case that spatial dependencies vary (slowly) over time.

3.3.3 Adapting or Learning the Graph

To avoid the need for a pre-defined graph, Zhang et al. [52] propose the Structure Learning Convolutional Neural Network (SLCNN) architecture, which incorporates a capacity to learn a better graph topology than the one derived from geographical considerations. The method in [53] also involves learning the graph; in this case, dynamic time warping is used to evaluate similarities between the time-series. The graph construction is a pre-processing step; it is not adapted during learning of the subsequent graph convolutional network. The use of dynamic time-warping leads to detection of correlations; there is no restriction to predictive or causal relationships, which are of more interest in the forecasting context. The Traffic Graph Convolutional Recurrent Neural Network (TGCRNN) of [60] integrates physical constraints in its construction of the graph adjacency matrix. These constraints are based on an assessment as to whether traffic at one node can influence traffic at another node within a specific time frame. Diao et al. [61] learn an adaptive Laplacian by maximizing the smoothness of the signals after removing a low-rank component. As in [53], this estimation is a pre-processing step applied before prediction. Formulating the estimation problem in terms of smoothness maximization means that neither causality nor predictive capability is considered.

3.3.4 Advanced Architectures

The pixel-wise prediction task in image processing and computer vision can be rendered analogous to multi-scale modeling of a dynamic graph by considering an image pixel as a graph node. Building on this insight, Yu et al. [62], inspired by the U-shaped networks in U-Net [63], propose the Spatio-Temporal U-Network (ST-UNet) which adds spatial pooling and temporal downsampling operations (followed by unpooling). This allows the architecture to learn representations at multiple scales and to model both the local dependencies and the global structure.

In contrast to most existing spatio-temporal approaches [15, 17, 56], the Graph WaveNet model introduced in [22] does not require a pre-defined graph. Graph WaveNet captures spatial correlations by using a graph convolution that generalizes the diffusion convolution proposed in [15]. This architecture also incorporates a module that can learn an adaptive graph topology, denoted as $\mathbf{A}_{\mathrm{adp}}$ in (3.1). For a directed graph with adjacency matrix $\mathbf{A}$, the diffusion process with K finite steps is specified by forward and backward transition matrices. The forward transition matrix is $\mathbf{P}_f = \frac{\mathbf{A}}{\sum_i \mathbf{A}_{:,i}}$ and the backward transition matrix is $\mathbf{P}_b = \frac{\mathbf{A}^T}{\sum_i \mathbf{A}^T_{:,i}}$. The proposed graph convolution layer has the following formulation:

$$\begin{aligned} \mathbf{A}_{adp} &= \mathrm{softmax}(\mathrm{ReLU}(\mathbf{E_1}\mathbf{E_2}^T))\,, \\ \mathbf{Z} &= \sum_{k=0}^{K} \mathbf{P}_f^k \mathbf{X}\mathbf{W}_f^k + \mathbf{P}_b^k \mathbf{X}\mathbf{W}_b^k + \mathbf{A}_{adp}^k \mathbf{X}\mathbf{W}^k\,, \end{aligned} \tag{3.1}$$

where $\mathbf{E_1}$, $\mathbf{E_2}$, $\mathbf{W}_f$, $\mathbf{W}_b$, and $\mathbf{W}$ are learnable parameters. Graph WaveNet employs dilated causal convolution [49] (also known as TCN) to extract temporal relationships. This allows the architecture to learn from a larger receptive field. Having a 1D input $\mathbf{x}$ and a kernel $\mathbf{F} \in \mathbb{R}^K$, the dilated causal convolution of $\mathbf{x}$ with $\mathbf{F}$ at time step t, using the dilation factor d, can be mathematically written as:

$$\mathbf{x} \star \mathbf{F}(t) = \sum_{k=0}^{K-1} \mathbf{F}(k)\mathbf{x}(t - d \times k)\,. \tag{3.2}$$

A gated mechanism for the TCNs allows the architecture to extract complex temporal dependencies. Although this model can learn the graph topology, the model assumes fixed spatial dependencies among nodes; it computes spatial dependencies once and uses these for predictions at all times.

Song et al. [23] propose Spatial-Temporal Synchronous Graph Convolutional Networks (STSGCNs). This approach is different from most other methods in that it constructs a single spatio-temporal graph and then applies multiple GCNs over this graph. This contrasts with most other methods [15,16,21] that use GCNs in the spatial domain and combine with LSTM or GRU networks in the temporal domain. Spatial dependencies are fixed once trained.

Xu et al. [64] introduce Spatial-Temporal Transformer Networks which do not apply a pre-defined graph structure and can model dynamical spatial dependencies. In this architecture, node features are enhanced with spatial and temporal positional embeddings. Spatial information is extracted by a combination of fixed graph convolution and dynamic graph convolution (with spatial dependencies learned from the traffic patterns). A temporal transformer is used to capture dynamic long-range temporal dependencies.

In [51], the Adaptive Graph Convolutional Recurrent Network (AGCRN) approach is proposed. Inspired by the matrix factorization method, Bai et al. modify the original GCN with a node adaptive parameter learning module, which allows the model to learn the node-specific features considering all time-series. For a graph of order N with adjacency matrix $\mathbf{A}$ and degree matrix $\mathbf{D}$, the modified GCN has the following formulation:

$$\mathbf{Z} = \left(\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right)\mathbf{x}\mathbf{E}_{\mathcal{G}}\mathbf{W}_{\mathcal{G}} + \mathbf{E}_{\mathcal{G}}\mathbf{b}_{\mathcal{G}}, \tag{3.3}$$

where $\mathbf{x}$ and $\mathbf{Z}$ are the input and the output of the GCN, respectively. $\mathbf{E}_{\mathcal{G}} \in \mathbb{R}^{N\times d}$ is the node-embedding matrix with embedding dimensionality of d ($d \ll N$). $\mathbf{W}_{\mathcal{G}}$ and $\mathbf{b}_{\mathcal{G}}$ are the weights and biases of the network, respectively. Instead of generating an adjacency matrix and then calculating the Laplacian matrix, this model uses a data adaptive graph generation module ($\hat{\mathbf{A}}$ in 3.4). Finally, the proposed model applies the adapted GCN instead of MLP layers in a GRU to capture both spatial and temporal correlations. The formulation of each AGCRN layer can be represented as:

$$\begin{aligned}
\hat{\mathbf{A}} &= \mathrm{softmax}(\mathrm{ReLU}(\mathbf{E}\mathbf{E}^T))\,, \\
\mathbf{r}^{(t)} &= \quad \sigma\left(\hat{\mathbf{A}}[\mathbf{x}^{(t)}, \mathbf{s}^{(t-1)}]\mathbf{E}\mathbf{W}_r + \mathbf{E}\mathbf{b}_r\right)\,, \\
\mathbf{u}^{(t)} &= \quad \sigma\left(\hat{\mathbf{A}}[\mathbf{x}^{(t)}, \mathbf{s}^{(t-1)}]\mathbf{E}\mathbf{W}_u + \mathbf{E}\mathbf{b}_u\right)\,, \\
\tilde{\mathbf{s}}^{(t)} &= \tanh\left(\hat{\mathbf{A}}[\mathbf{x}^{(t)}, \left(\mathbf{r}^{(t)} \odot \mathbf{s}^{(t-1)}\right)]\mathbf{E}\mathbf{W}_s + \mathbf{E}\mathbf{b}_s\right)\,, \\
\mathbf{s}^{(t)} &= \mathbf{u}^{(t)} \odot \mathbf{s}^{(t-1)} + \left(1 - \mathbf{u}^{(t)}\right) \odot \tilde{\mathbf{s}}^{(t)}\,,
\end{aligned} \tag{3.4}$$

where $\mathbf{r}^{(t)}$ and $\mathbf{u}^{(t)}$ are reset and update gates of a GRU cell at time step t, respectively. $\mathbf{E}$, $\mathbf{W}_r$, $\mathbf{W}_u$, $\mathbf{W}_s$, $\mathbf{b}_r$, $\mathbf{b}_u$ and $\mathbf{b}_s$ are learnable parameters of the model, and $[\cdot,\cdot]$ indicates the concatenation. In summary, the AGCRN layer can be represented as $\mathbf{s}^{(t)} = AGCRN(\mathbf{x}^{(t)}, \mathbf{s}^{(t-1)})$, where $\mathbf{x}^{(t)}$ and $\mathbf{s}^{(t)}$ are the input and the output of the layer respectively. The AGCRN model does not rely on a pre-specified graph. This approach can be potentially applied to prediction problems wherever pair-wise correlation is important. One of the main drawbacks of this model is that it is not clear how to use AGCRN for large-scale evolving graphs.

3.4 Summary

This literature review provides a summary of the approaches that have been proposed to tackle the time series forecasting problem. The review focuses on recent methods based on neural networks because these have been demonstrated to provide superior performance to statistical techniques, except in cases where very little training data is available.

Although traditional time series forecasting models, such as State Space and Autoregressive models as presented in Section 3.1 are still widely used, one of the main limitations is that most of the statistical approaches require specification of a much more restricted model. The neural network approaches can learn a model from the data. Deep neural networks have been employed, as discussed in Section 3.2, and various kinds of RNNs, including LSTM and GRU networks, have been used to model time series. However, due to the problems associated with vanishing and exploding gradients, RNNs remain challenging to train [65].

In recent years, spatio-temporal forecasting has received an increasing amount of attention. The goal is to extract the temporal patterns existing in the data as well as to learn and exploit the spatial relationships among the time series. The proposed spatio-temporal models (Section 3.3), which take into account (or learn) a graph describing dependencies between the time series, have outperformed the non-graph temporal models. These methods apply various GNNs to capture spatial dependencies. Most of the algorithms rely on a pre-specified graph, and can struggle to perform well if the graph does not accurately capture the dependencies. This is a key weakness of these approaches since the pre-defined graph often omits some of the important spatial relationships [51]. The manual specification of the graph heavily relies on domain knowledge, so the techniques are not readily applicable to data from other domains. To address this, we propose a novel neural network architecture that does not rely on any pre-specified graph, but instead learns multiple graphs that capture different types of dependencies. Our architecture is presented in the following chapter.

CHAPTER 4
Fully Connected Gated Graph Architecture (FC-GAGA)

This chapter introduces a novel pure deep learning architecture, called FC-GAGA, for spatio-temporal forecasting. The first section provides a more formal statement of the problem. Section 4.2 describes the details of the FC-GAGA architecture. It includes a discussion of a variety of designs that were investigated for the FC-GAGA model to improve its performance. Finally, in Section 4.3, we report the results of experiments on two non-graph datasets and two graph-based datasets. These experiments allow us to compare the performance of the proposed method with the state-of-the-art baselines. Moreover, ablation studies are provided to validate the effectiveness of different parts of the model.

4.1 Problem Setting

The problem of spatio-temporal forecasting involves simultaneously forecasting multiple time series originating from related entities. We model these relationships via an unknown underlying graph. Let a graph $G = (V, E)$ be defined as an ordered collection of vertices, $V = 1, \ldots, N$, and edges, $E \subset V \times V$. We are interested in the multivariate TS forecasting problem defined on this graph. Each vertex v in the graph is assumed to generate a sequence of observations, $\mathbf{y}_v = [y_{v,1} \ldots, y_{v,T}] \in \mathbb{R}^T$, governed by an unknown stochastic random process. The graph connectivity encoded in E is assumed to capture unknown relations between the vertices. For instance, considering the roads in a road network as graph vertices, the graph edges E may reflect the connectivity of the roads, and $\mathbf{y}_v$ may be the sequence of observations of traffic velocity. The task is to predict the vector of future values $\mathbf{y}_v \in \mathbb{R}^H = [y_{T+1}, y_{T+2}, \dots, y_{T+H}]$ for every vertex v based on the observations generated by all the vertices in the graph up to time T. The model has two inputs: (1) the input of length $w \leq T$ at vertex v, ending with the last observed value $y_{v,T}$, denoted as $\mathbf{x}_v \in \mathbb{R}^w = [y_{v,T-w+1}, \dots, y_{v,T}]$; and the corresponding time covariate input, $\mathbf{c}_v \in \mathbb{R}^{w \times d_c}$, where d_c is the number of available time covariate features. Example covariates are the time of the day and the day of the month. The output of the model is the point forecast of $\mathbf{y}_v$ at vertex v, and is indicated as $\widehat{\mathbf{y}}_v$.

4.2 FC-GAGA Architecture

The block diagram of a single layer of FC-GAGA is presented in Figure 4–1. The layer contains a graph gate, a time gate, and a fully connected time-series model. One of the important parts of the model design is a node embedding matrix $\mathbf{E}$ which is utilized in the graph and time gates. In our model, each node i is represented as an embedding vector of dimensionality d, $\mathbf{e}_i = [e_{i,1}, \dots e_{i,d}]$. The collection of all such vectors comprises the node embedding matrix $\mathbf{E} \in \mathbb{R}^{N \times d}$, defined as a set of learnable parameters and learned as part of the backpropagation procedure.

The input to the FC-GAGA layer is a matrix $\mathbf{X} \in \mathbb{R}^{N \times w}$ containing the history of length w of all nodes in the graph, together with the corresponding time covariate matrix $\mathbf{C} \in \mathbb{R}^{N \times w}$. Here, we only utilize the time-in-day feature of each node at a time; therefore, d_c is equal to one. The model generates predictions for all the nodes

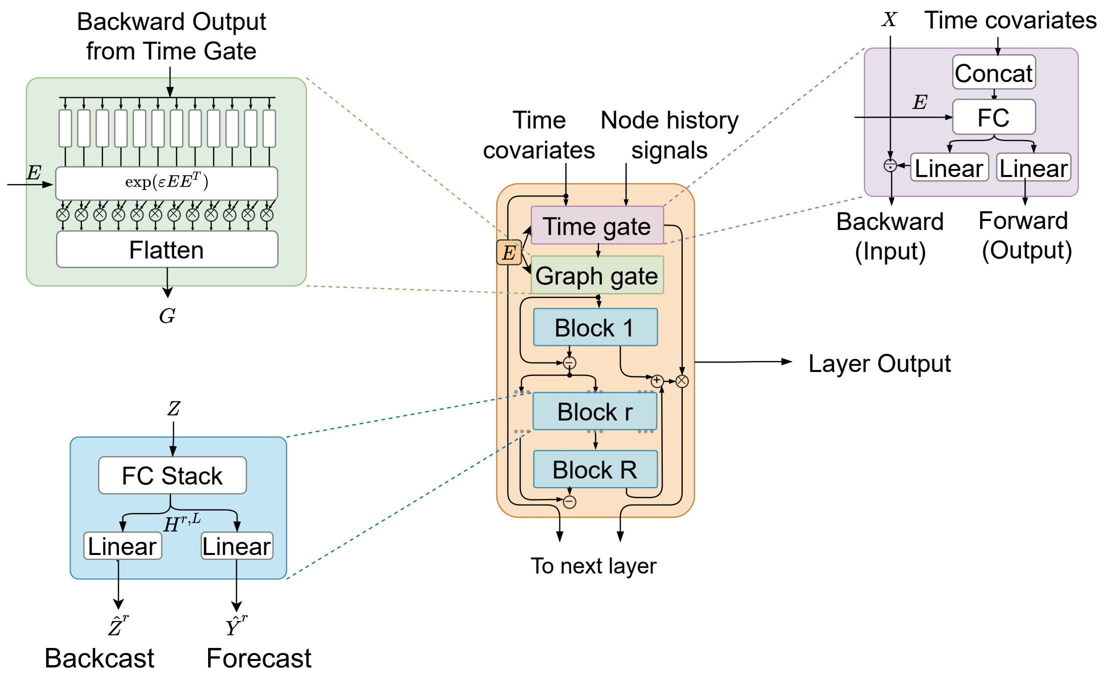


Figure 4–1: Diagram of a single layer of FC-GAGA, which includes a graph gate, a time gate and a fully connected time-series model.

in one shot; the predictions are denoted as $\widehat{\mathbf{Y}} \in \mathbb{R}^{N \times H}$, where H is the horizon of prediction. We now describe the operation of a single layer of FC-GAGA in detail.

Time gate block. The goal of the time gate block is to model the time covariate features (*e.g.*, time-of-day and day-of-week) that may be available together with the node observations. In general, data with strong seasonal patterns require model components that are able to explicitly capture these effects. We extract time related features using a multiplicative gate model. The gate normalizes the input of the FC-GAGA layer using a fully connected network as depicted in Figure 4–1. The normalization allows the other blocks to focus on relationships that do not depend on the time covariates. These normalization effects are reversed at the output of the layer.

There are two key features in the design of the time gate block. First, the input time feature matrix $\mathbf{C}$ is concatenated with the node embedding matrix $\mathbf{E}$ for each node to account for the fact that each node may have a different seasonality pattern. This is equivalent to removing a node-specific multiplicative seasonality from the input of the block and applying it again at the output of the block. Second, the model design allows the input and output time effects to be decoupled via separate linear projection layers. This is important because the input and output occur at different times, and the time effects are therefore usually different.

Graph edge weights. In the proposed architecture, the strengths of node links are encoded in a weight matrix $\mathbf{W} \in \mathbb{R}^{N \times N}$ derived from node embeddings:

$$\mathbf{W} = \exp(\epsilon \mathbf{E}\mathbf{E}^T). \tag{4.1}$$

Here ϵ is a hyperparameter that is set to allow for the decoupling of the scaling of $\mathbf{E}$ from the scaling that is required to achieve a suitable dynamic range in $\mathbf{W}$. Each row of $\mathbf{E}$ stores an embedding of a node in the graph; it is designed to capture its semantic properties. Then, the product $\mathbf{E}\mathbf{E}^T$ provides a matrix of node relations, and $\mathbf{W}$ in Equation (4.1) models the strengths of the relationships across node pairs. As a result, we expect that the magnitudes of edge weights $\mathbf{W}_{i,j}$ will reflect the extent of mutual influence between the pair of nodes (i, j) at a given FC-GAGA layer.

Graph gate block. The input of the graph gate block comes from the backward output of the time gate block, as illustrated in Figure 4–1. The graph gating operation, as in Equation (4.2), produces a matrix $\mathbf{G} \in \mathbb{R}^{N \times Nw}$. The first dimension (row) of the gated matrix corresponds to the target node i. The second dimension corresponds to all the nodes which the target node interacts with. It contains all the information accumulated by the graph during the past w steps. The entry $\mathbf{G}_{i,jw+k}$ reflects the anticipated influence of the $\mathbf{X}_{j,k}$, the k-th entry of the time series $\mathbf{X}$ associated with node j on the prediction of $\mathbf{x}_i$. The graph gate calculation is

$$\mathbf{G}_{i,jw+k} = \text{ReLU}[(\mathbf{W}_{i,j}\mathbf{X}_{j,k} - \widetilde{\mathbf{x}}_i)/\widetilde{\mathbf{x}}_i], \tag{4.2}$$

where $\widetilde{\mathbf{x}}$ is the maximum of the input values over the time dimension, $\widetilde{\mathbf{x}}_i = \max_j \mathbf{X}_{i,j}$.

In the proposed graph gate design, there are several important elements at play. First, the scaling operation, i.e., the division by $\widetilde{\mathbf{x}}$, the maximum value of in the input window, was found through experiments to be effective for the fully connected time-series block. Some form of scaling is necessary, since unscaled input variables can have very different ranges, and this can result in a slow or unstable learning process. If we scale the time series of each node with respect to its own maximum value, the relative magnitude information between different input time series will be lost. Therefore, we propose scaling by the maximum for the target node, which results in the stability of scaling but also retains the relative magnitudes of different inputs.

Second, the measurements in nodes i and j have to be aligned and related to each other. This is accomplished by the operation inside the ReLU where the level of node j is aligned with the level of the target node i Furthermore, the ReLU operation has the function of shutting off the irrelevant i, j pairs while not affecting the scale alignment achieved via $\mathbf{W}_{i,j}$. The magnitude of $\mathbf{W}_{i,j}$ affects the probability of opening the hard gate.

Fully connected time-series block. We use a fully connected residual architecture with L hidden layers, R residual blocks and weights shared across nodes. Its input for the target node i, $\mathbf{Z}_i$, is conditioned on the node embedding and its own history: $\mathbf{Z} = [\mathbf{E}, \mathbf{X}/\widetilde{\mathbf{x}}, \mathbf{G}]^T$, $\mathbf{Z} \in \mathbb{R}^{N(w+1)+d \times N}$. Using residual block and layer superscripts and denoting the fully connected layer with weights $\mathbf{A}^{r,\ell}$ and biases $\mathbf{b}^{r,\ell}$ as $\mathrm{FC}_{r,\ell}(\mathbf{H}^{r,\ell-1}) \equiv \mathrm{ReLU}(\mathbf{A}^{r,\ell}\mathbf{H}^{r,\ell-1} + \mathbf{b}^{r,\ell})$, the operation of the fully connected residual TS modeling architecture is described as follows:

$$\begin{aligned} \mathbf{Z}^r &= \mathrm{ReLU}[\mathbf{Z}^{r-1} - \widehat{\mathbf{Z}}^{r-1}], \\ \mathbf{H}^{r,1} &= \mathrm{FC}_{r,1}(\mathbf{Z}^r),\ \ldots,\ \mathbf{H}^{r,L} = \mathrm{FC}_{r,L}(\mathbf{H}^{r,L-1}), \\ \widehat{\mathbf{Z}}^r &= \mathbf{B}^r\mathbf{H}^{r,L},\ \widehat{\mathbf{Y}}^r = (\mathbf{H}^{r,L})^T\mathbf{F}^r. \end{aligned} \tag{4.3}$$

The first formula in Equation (4.3) indicates the residual process, and the second formula describes the L hidden layers mathematically. This block produces backcast ($\widehat{\mathbf{Z}}^r$) and forecast ($\widehat{\mathbf{Y}}^r$). We assume $\widehat{\mathbf{Z}}^0 \equiv \mathbf{0}$ and $\mathbf{Z}^0 \equiv \mathbf{Z}$. The projection matrices have dimensions $\mathbf{B}^r \in \mathbb{R}^{N(w+1)+d \times d_h}$ and $\mathbf{F}^r \in \mathbb{R}^{d_h \times H}$. The final forecast of the FC-GAGA layer is the sum of forecasts of all residual blocks, $\widehat{\mathbf{Y}} = \sum_r \widehat{\mathbf{Y}}^r$.

FC-GAGA layer stacking. One of the effective features of our model design is that the basic layer explained above is stackable (see Figure 4–2) based on the following three principles. First, the next layer accepts the sum of forecasts of previous layers as input. Second, each FC-GAGA layer has its own set of node embeddings and thus its own graph gate. Thus each layer is provided the freedom

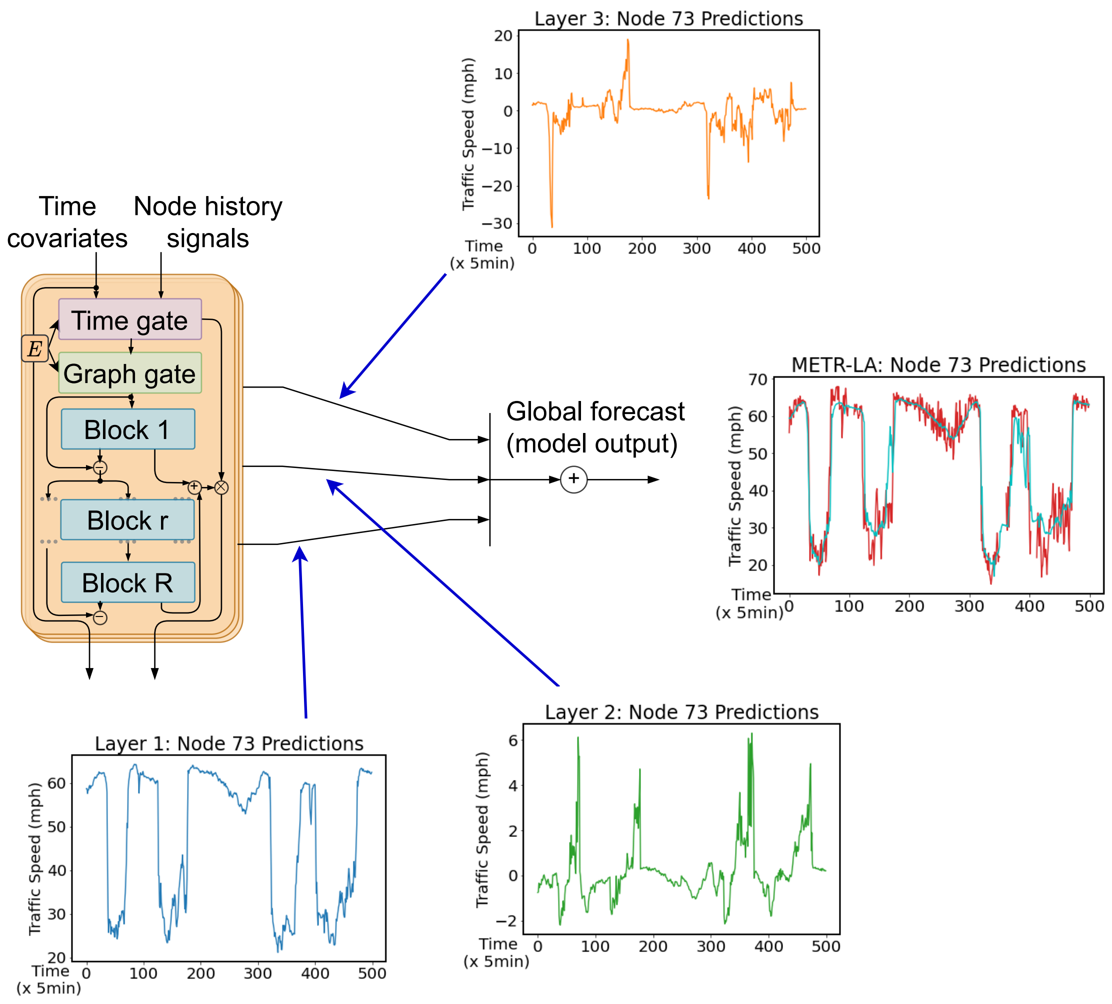

Layer 3: Node 73 Predictions
Traffic Speed (mph)
Time (x 5min)
Time covariates
Node history signals
Time gate
Graph gate
Block 1
Block r
Block R
E
Global forecast (model output)
METR-LA: Node 73 Predictions
Layer 1: Node 73 Predictions
Layer 2: Node 73 Predictions

Figure 4–2: FC-GAGA layer stacking with 3 layers.

to gate the information flow across nodes after taking into account the processing already accomplished by the previous layer(s). For example, in the first FC-GAGA layer, for node id 5, it may be optimal to focus on the histories of node ids [10, 200, 500]. However, since the first FC-GAGA layer updates the states of all nodes, in the second layer, node 5 may no longer need the information provided by nodes [10, 200, 500], nor that provided by their neighbours; and instead may wish to focus on node ids [3, 15], as they now provide more important information for adjusting the forecast. Finally, the final model output is equal to the average of the layer forecasts.

Complexity analysis. In the following analysis, we skip the batch dimension and compute the complexity involved in creating a single forecast of length H for all nodes N in the graph when the input history is of length w, the node embedding width is d and the hidden layer width is d_h. The graph gate block has complexity $\mathcal{O}(N^2(w+d))$, as is evident from Equation (4.2), which involves the $N \times N$ node interaction matrix derived from the $N \times d$ embedding matrix and gating of the $N^2 w$ input values. The time gate mechanism produces a seasonality factor for each node using its associated time feature, so its complexity scales linearly with the number of nodes, the hidden dimension, the input history length and the forecast horizon, i.e., $\mathcal{O}(N(d+w)d_h + NHd_h)$. In most practical situations we have $d+w > H$. Finally, the fully-connected TS model with L FC layers and R residual blocks that accepts the flattened $N \times Nw$ output of the graph gate scales as follows. The first and the last layers of the residual block scale as $\mathcal{O}(N^2 w d_h)$ (recall that the last linear layer is doing a backcast from d_h to Nw). The hidden layers scale as $\mathcal{O}(Nd_h^2)$. This results in the total fully-connected TS model complexity $\mathcal{O}(R(2N^2 w d_h + (L-2)Nd_h^2))$. In most practical configurations, the total complexity of the model will be dominated by $\mathcal{O}(N^2 R w d_h)$. According to the presented complexity analysis, our graph gate design has N times smaller complexity, $O(N^2)$, compared to the approaches known in the literature that are based on matrix multiplication in the graph diffusion step $O(N^3)$ (e.g. DCRNN [15] and Graph WaveNet [22]).

4.2.1 Variant Designs of FC-GAGA Architecture

In this section, different cases of the FC-GAGA model are investigated in order to better understand the effects of each part of the model and improve the design of the model. These cases are examined on two public graph-based datasets, and the results are presented in Section 4.3.5.

- *Asymmetric Graph Embedding Matrix*

  A.1. In the original architecture, the graph weights matrix is defined as $\mathbf{W} = \exp(\epsilon \mathbf{E}\mathbf{E}^T)$, where $\mathbf{E}$ is the node embedding matrix. Instead we consider the graph weights as $\mathbf{W} = \exp(\mathbf{W}')$, where $\mathbf{W}'$ is a matrix of learnable parameters, learned as part of the backpropagation procedure. The idea is to understand the impact on the model performance of learning the graph weight matrix directly instead of using node embeddings.

  A.2. Using the graph weights matrix $\mathbf{W} = \exp(\mathbf{W}')$ similar to the case A.1, but instead of utilizing the node embedding matrix $\mathbf{E}$ for the time gate, an extra fully connected layer is added to the time gate block to learn the relationships between the nodes.

A.3. In this case, the graph weight matrix is defined as $\mathbf{W} = \exp(\epsilon \mathbf{E}_1 \mathbf{E}_2^T)$, where $\mathbf{E}_1$ and $\mathbf{E}_2$ are two separate node embedding matrices. This provides more freedom for asymmetric predictive relationships.

- *Layer Constraints*

  In the original model, all layers have the freedom to learn the graph edge weights matrix $\mathbf{W}$ using information from all nodes. Hence, the model generates the predictions for a target node using the history information that corresponds to all the nodes with which the target node interacts. In this case, we constrain some layers by setting their weight matrices to the identity matrix. This constraint forces a layer to make predictions for the target node based only on its own history information.

4.3 Experiments

In this section, the proposed architecture in this thesis is assessed via conducting thorough experiments on two graph-based and two non-graph datasets. First, the utilized datasets for the evaluation are described in Section 4.3.1, followed by preprocessing steps. Then the baselines used for the comparison are presented. After explaining the experimental details, the quantitative and qualitative results are provided. The chapter will be finalized by the ablation studies and profiling results.

4.3.1 Datasets

The proposed FC-GAGA architecture is evaluated on the graph-based datasets, which are METR-LA and PEMS-BAY [15,66] and the non-graph datasets Electricity and Traffic [29]. The summary statistics of the datasets are provided in Table 4–1 and Table 4–2.

- *Graph-based datasets*
  - **METR-LA:** consists of the data of 207 sensors collected from loop detectors in the highway of Los Angeles County for four months from March 1st, 2012 to June 30th, 2012.
  - **PEMS-BAY:** contains the data of 325 sensors in the Bay Area for six months from January 1st, 2017 to May 31th, 2017.

In both datasets, the traffic speed readings of sensors are aggregated into 5 minute windows. The sensor distributions of the METR-LA dataset, for instance, are visualized in Figure 4–3. To construct the sensor graph, the pairwise road network distances between sensors are computed.

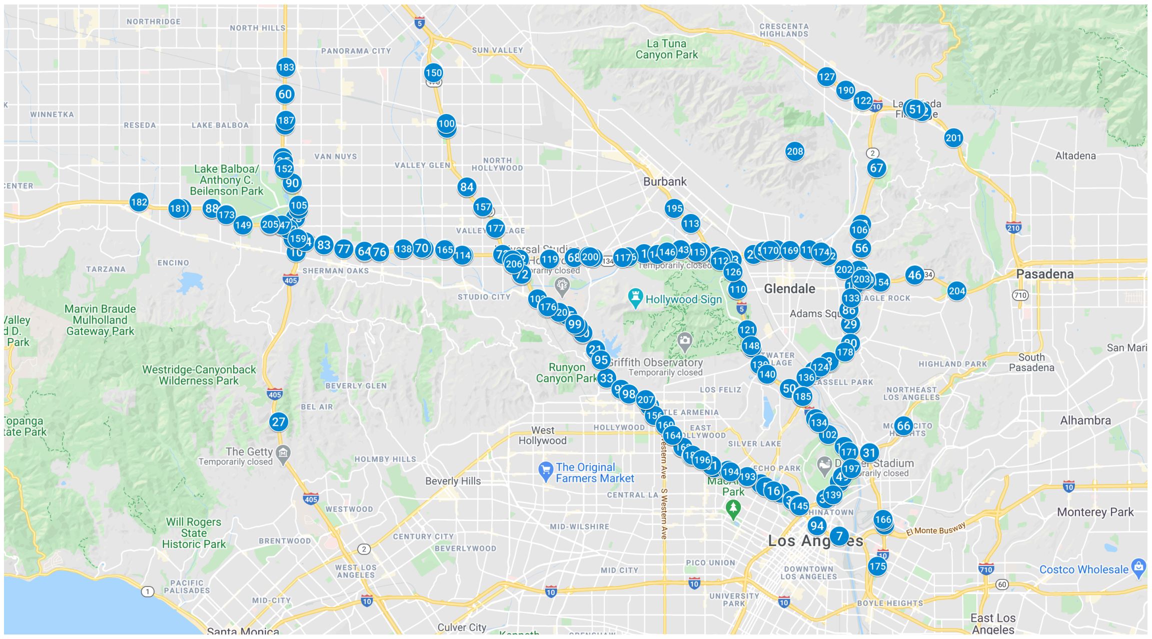


Figure 4–3: The visualization of sensor distributions in the METR-LA dataset.

- *Non-graph datasets*

– **Electricity:** The electricity usage in kW was recorded every 15 minutes for 370 Portuguese clients from 2011 to 2014. The data is converted to reflect hourly consumption by aggregating every 4 points of the original Electricity dataset.

– **Traffic:** This is a collection of 15 months of daily data from the California Department of Transportation. The data describes the occupancy rate of different car lanes of San Francisco freeways. The data was sampled every 10 minutes. Therefore, the hourly traffic occupancy dataset on 963 roads is obtained by taking the average over every 6 points in the original Traffic dataset.

Table 4–1: Statistics of non-graph datasets.

Dataset	**#Time series**	**Domain**	**Freq.**	**#Time steps**	**Prediction horizon**
Electricity	370	$\mathbb{R}^+$	Hourly	26,304	24
Traffic	963	(0, 1)	Hourly	10,560	24

Table 4–2: Statistics of graph-based datasets.

Dataset	**#Nodes**	**Domain**	**Interval**	**#Time step**
METR-LA	207	$\mathbb{R}^+$	5 Minutes	34,272
PEMS-BAY	325	$\mathbb{R}^+$	5 Minutes	52,116

4.3.2 Preprocessing

For all datasets, missing values are substituted by the last known value in the same time-series. Furthermore, for non-graph datasets, all conducted preprocessing is aligned with the N-BEATS paper [1], which is as following. Both Electricity and Traffic datasets are converted to reflect hourly data in the Traffic dataset by taking the average over every 6 points and the Electricity dataset via aggregating every 4 points. The conversion to hourly is conducted so that all the points available in $(h-1:00, h:00]$ hours are aggregated to hour h, e.g., if the start of the original dataset is "2011-01-01 00:15" then the first point after conversion will be "2011-01-01 01:00". In the Electricity dataset, the first year from the training set is removed; therefore, the start date of this dataset is 2012-01-01, following the custom in the TRMF literature [47]. Furthermore, it is also checked that data points for both Electricity and Traffic datasets after conversion match those used in [47]. As we follow the protocol utilized by the TRMF model, in both datasets, the last 7 days are considered as the test set. These pre-processing decisions were conducted to facilitate comparison with earlier works. The graph-based datasets are chronologically split as 70% of data for training, 10% for validation, and 20% for testing.

4.3.3 Baselines and Proposed Architecture

- **Experiment 1:** For graph-based datasets, the proposed architecture is compared with both temporal models that do not require a pre-specified graph and spatio-temporal models that may rely on a pre-specified graph or have a learnable graph.

  The following temporal models are considered:

  - *ARIMA:* Auto-Regressive Integrated Moving Average model [67] implemented using a Kalman filter.
  - *SVR:* Linear Support Vector Regression model [68].
  - *FNN:* Feedforward Neural Network.

- *FC-LSTM:* The sequence-to-sequence model that uses fully connected LSTM cells in the encoder and decoder [69].
- *N-BEATS:* Neural Basis Expansion Analysis for Interpretable Time Series [1], an univariate model, built using backward and forward residual connections and a deep stack of fully-connected layers. Here, we utilize the generic architecture of N-BEATS without ensembling.

The spatio-temporal models include:

- *DCRNN:* Diffusion Convolutional Recurrent Neural Network [15], a graph convolutional network inside the sequence-to-sequence architecture.
- *STGCN:* Spatio-Temporal Graph Convolutional Network [16], merges graph convolutions with gated temporal convolutions.
- *Graph WaveNet:* fuses graph convolution and dilated causal convolution proposed in [22].
- *GMAN:* Graph Multi-Attention Network [17], an encoder-decoder model with multiple spatio-temporal attention blocks, and a transform attention layer between the encoder and the decoder.
- *STGRAT:* Spatio-Temporal Graph Attention Network for Traffic Forecasting [59], an encoder-decoder model using the positional encoding method of the Transformer [44] to capture features of long sequences and node attention to capture spatial correlation.
- *AGCRN:* Adaptive Graph Convolutional Recurrent Network [51], a combination of a GRU with a modified GCN which has a node-adaptive parameter learning module.

  - *FC-GAGA:* The proposed architecture having the number of layers $L = 3$.
  - *FC-GAGA (4 layers)‡:* The proposed model with 4 layers. In the fourth layer, the graph gate weights are set to the identity matrix, implying more reliance on the pure time-series component.

  Of the mentioned methods, only Graph Wavenet can generate predictions without a pre-specified graph. For DCRNN, we run the official code and report the results after a bug fix in the code, which are better than the originally reported results in the paper [15]. For STGRAT, we report the results from the original paper since the official code is not provided. We execute STGCN; however, in our experiments, STGCN achieves slightly worse performance than reported in the original paper; therefore, we cite the results published by the authors [16]. The results reported for Graph WaveNet, GMAN and AGCRN methods are produced via running the official codes provided by the authors.

- **Experiment 2:** For non-graph datasets, FC-GAGA is compared to the following deep learning based forecasting approaches:
  - *TRMF:* Temporal Regularized Matrix Factorization [47], which integrates temporal dependencies and matrix factorization models by introducing a temporal regularizer.
  - *DeepGLO:* The proposed method in [48], which is a global matrix factorization with regularization using temporal convolution networks.
  - *N-BEATS:* Neural Basis Expansion Analysis for Interpretable Time Series [1], Here, we utilize the ensemble of generic and interpretable architectures of N-BEATS.

; and compared to deep learning probabilistic forecasting models:

– *DeepAR:* The model presented in [31], which is an RNN based probabilistic method that uses a parametric likelihood for forecasting.

– *DeepState:* The probabilistic model proposed in [30], which is the fusion of a linear state-space model and an RNN.

– *DeepFactors:* The proposed model in [50] that uses RNNs for global dynamics along with a local probabilistic graphical model to account for uncertainty.

For the N-BEATS model, we report the best results of non-graph datasets provided in the original paper [1]. For TRMF, the results also are reported from the original paper [47]. Since the approaches proposed by Amazon Labs (DeepAR, DeepState, and DeepFactors) utilize different splits as their test sets compared to the TRMF and N-BEATS models, we cannot utilize their reported results. Therefore, we train these models using GluonTS framework [70] released by the authors. Note that the implementations of the DeepState and DeepFactors models are not completely aligned with the original methods, and some parts are missing. We optimized hyperparameters for these models. DeepGLO also used a different split of data as the test set. Thus, we train the model using the unnormalized version of datasets since DeepGLO achieves better performance in this case, and we utilize the provided settings released with the official code by the authors.

Descriptions of all of the baselines are provided in Chapter 3.

4.3.4 Experimental Details

Experiment 1. To adjust the hyperparameters properly, we examine the performance of the proposed model over various choices of different parameters: the best scaler ϵ in Equation 4.1 is selected among the set $\{1, 5, 10, 20\}$, the number of hidden layer width d_h for all fully connected layers is examined over the set $\{64, 128, 256\}$, and the number of blocks R and the weight decay are chosen over the sets $\{2, 3, 4\}$ and $\{1e^{-4}, 1e^{-5}, 1e^{-6}\}$ respectively. The number of epochs is tested over $\{60, 80, 100\}$ and the initial learning rate is chosen over the set $\{1e^{-3}, 1e^{-4}\}$. Fianlly the best hyperparameters are optimized as followed. The scalar ϵ is set to 10. The embedding dimensionality, d, is set to 64 and the hidden layer width d_h is set to 128. The number of blocks R in the fully-connected TS model is equal to 2. We use weight decay of 1e-5 to regularize fully-connected layers. The model is trained using the Adam optimizer with default TensorFlow settings and initial learning rate of 0.001 for 60 epochs. The learning rate is annealed by a factor of 2 every 6 epochs starting at epoch 43. One epoch consists of 800 batches of size 4, and the model takes the history of 12 time steps and predicts 12 time steps (60 min) ahead in one shot. Each training batch is assembled using 4 time points chosen uniformly at random from the training set and the histories of all nodes collected at each of the time points. METR-LA has 207 sensor nodes and in PEMS-BAY has 325, resulting in the batches consisting of $207 \cdot 4 = 828$ and $325 \cdot 4 = 1300$ time-series, respectively. The objective function used to train the network is Mean Absolute Error (MAE), averaged over all nodes and all forecasts within horizon $H = 12$:

$$\mathcal{L} = \frac{1}{HN} \sum_{i=1}^{H} \sum_{v=1}^{N} |y_{v,T+i} - \widehat{y}_{v,T+i}|. \tag{4.4}$$

We opted for MAE loss function since it achieves better results in our experiments. Unlike Mean Square Error (MSE), MAE is more robust to outliers. Moreover, it provides a balance between the accuracies of different horizons. The accuracy of our model is measured with three widely used metrics: Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Root Mean Squared Error (RMSE):

$$\begin{aligned} \text{MAE} &= \frac{1}{N} \sum_{v=1}^{N} |y_{v,T+H} - \widehat{y}_{v,T+H}|, \\ \text{MAPE} &= \frac{1}{N} \sum_{v=1}^{N} \frac{|y_{v,T+H} - \widehat{y}_{v,T+H}|}{|y_{v,T+H}|}, \\ \text{RMSE} &= \sqrt{\frac{1}{N} \sum_{v=1}^{N} (y_{v,T+H} - \widehat{y}_{v,T+H})^2}. \end{aligned} \tag{4.5}$$

Experiment 2. For non-graph datasets, we follow the common procedure utilized by the TRMF baseline model [47], which is to consider the last 7 days of data as the test set for both Electricity and Traffic datasets. Therefore, the number of training points for Electricity and Traffic datasets will be $26,136$ and $10,392$ respectively.

In this experiment, following standard practice in the literature, the proposed model takes the history of 24 time steps (one day) and predicts 24 time steps ahead in one shot. Analogous to the procedure used by existing baselines, the rolling-day prediction for seven days after the last point seen in training is conducted as evaluation, i.e., the prediction horizon is one day (24 points) and the start time of the forecast is shifted by one day after evaluating the prediction for the current day. Here, the 7 consecutive days right before the test set are utilized as the validation set. All the training settings is the same as *Experiment 1* except for the number of epochs and the number of batches which increase to 80 and 1600, respectively. The objective function used to train the model is MAE since this loss obtains the best performance. We measure accuracy for non-graph datasets via Normalized Deviation (ND) as in the TRMF literature [47]:

$$\mathrm{ND} = \frac{\sum_{v=1}^{N} |\widehat{y}_{v,T+H} - y_{v,T+H}|}{\sum_{v=1}^{N} |y_{v,T+H}|}. \tag{4.6}$$

4.3.5 Quantitative Results

Results of graph-based datasets:

Our key empirical results on two traffic datasets appear in Table 4–3. FC-GAGA compares favourably even against graph-based models that rely on additional external graph definitions on both METR-LA and PEMS-BAY datasets (DCRNN, STGCN, Graph WaveNet, and GMAN). Most of the time, FC-GAGA outperforms the Graph WaveNet[‡] model when the algorithms are trained and evaluated in the same conditions, i.e., both models only rely on the graph learned from the data. (In this setting, Graph WaveNet uses only the *adaptive* adjacency matrix that it learns from the data). FC-GAGA significantly outperforms the univariate models (ARIMA, SVR, FNN, FC-LSTM, and N-BEATS). Note that STGRAT heavily relies on the main ingredients of Transformer architecture such as positional encoding and attention mechanisms. Therefore, comparing FC-GAGA against it gives a good idea of how our approach stands against Transformer-based methods in terms of accuracy.

Results of non-graph datasets:

The proposed FC-GAGA architecture is also evaluated on the Electricity and Traffic datasets and the results are provided in Table 4–4. We observe that the FC-GAGA model outperforms the existing methods for the Electricity dataset. In the Traffic dataset, the univariate N-BEATS shows a better performance compared to the multivariate models. This indicates that most time-series in this dataset do not provide valuable information for generating predictions of other time-series. This result is the opposite of what was observed for the graph-based datasets, where the performance of N-BEATS was considerably worse than the multivariate algorithms. We conjecture that one reason for this is that the time-granularity of the Traffic dataset is much coarser (hourly rather than 5 minute intervals).

Results of FC-GAGA architectural variants:

In the following, we provide the results of experiments conducted to examine the FC-GAGA architectural variants, explained in Section 4.2.1.

- *Asymmetric Graph Embedding Matrix:*

  Based on Table 4–5, none of the alternative methods improve the results. Learning the graph weights matrix independently (A.1) yields worse performance than the original model (FC-GAGA). This result suggests that using the node embedding matrix in all parts of the architecture (both graph gate and temporal gate) causes a useful functional coupling during the learning procedure. Using an extra fully connected layer instead of the node embedding matrix $\mathbf{E}$ (A.2) leads to significantly worse performance, confirming that the

Table 4–3: Key empirical results on two traffic datasets. Error metrics computed by averaging over last time step of horizon with input window length 12. The best and the second best results in each column are shown in bold and marked with underline respectively. Lower numbers are better. ‡Graph Wavenet trained using official code by the authors using only adaptive matrix without the support of geographical adjacency matrix.

	METR-LA (15/30/60 min)		
Models	MAE	MAPE (%)	RMSE
DCRNN	2.67/3.08/3.56	6.84/8.38/10.30	5.17/6.30/7.52
STGCN	2.88/3.47/4.59	7.62/9.57/12.70	5.47/7.24/9.40
Graph WaveNet	2.69/3.07/3.53	6.90/8.37/10.01	5.15/6.22/7.37
GMAN	2.77/3.07/**3.40**	7.25/8.35/**9.72**	5.48/6.34/7.21
STGRAT	**2.60**/**3.01**/3.49	**6.61**/**8.15**/10.01	**5.07**/6.21/7.42
AGCRN	2.89/3.25/3.67	7.79/9.13/10.53	5.61/6.56/7.66
ARIMA	3.99/5.15/6.90	9.60/12.70/17.40	8.21/10.45/13.23
SVR	3.99/5.05/6.72	9.30/12.10/16.70	8.45/10.87/13.76
FNN	3.99/4.23/4.49	9.90/12.90/14.00	7.94/8.17/8.69
FC-LSTM	3.44/3.77/4.37	9.60/10.90/13.20	6.30/7.23/8.69
Graph WaveNet‡	2.80/3.18/3.57	7.45/9.00/10.47	5.45/6.42/7.29
N-BEATS	3.08/3.79/4.91	8.10/10.58/14.70	6.10/7.68/9.73
FC-GAGA	2.75/3.10/3.51	7.25/8.57/10.14	5.34/6.30/7.31
FC-GAGA(4 layers)‡	2.70/3.04/3.45	7.01/8.31/9.88	5.24/**6.19**/**7.19**

	PEMS-BAY (15/30/60 min)		
Models	MAE	MAPE (%)	RMSE
DCRNN	1.31/1.66/1.98	2.74/3.76/4.74	2.76/3.78/4.62
STGCN	1.36/1.81/2.49	2.90/4.17/5.79	2.96/4.27/5.69
Graph WaveNet	1.30/1.63/1.95	2.73/3.67/4.63	2.74/3.70/4.52
GMAN	1.34/1.62/1.86	2.81/**3.63**/**4.31**	2.82/3.72/**4.32**
STGRAT	**1.29**/**1.61**/1.95	**2.67**/**3.63**/4.64	**2.71**/**3.69**/4.54
AGCRN	1.38/1.70/2.01	2.98/3.88/4.68	2.90/3.87/4.66
ARIMA	1.62/2.33/3.38	3.50/5.40/8.30	3.30/4.76/6.50
SVR	1.85/2.48/3.28	3.80/5.50/8.00	3.59/5.18/7.08
FNN	2.20/2.30/2.46	5.19/5.43/5.89	4.42/4.63/4.98
FC-LSTM	2.05/2.20/2.37	4.80/5.20/5.70	4.19/4.55/4.96
Graph WaveNet‡	1.34/1.69/2.00	2.79/3.79/4.73	2.83/3.80/4.54
N-BEATS	1.47/2.01/2.77	3.07/4.54/6.76	3.21/4.64/6.37
FC-GAGA	1.36/1.68/1.97	2.87/3.80/4.67	2.86/3.80/4.52
FC-GAGA(4 layers)‡	1.34/1.66/**1.93**	2.82/3.71/4.48	2.82/3.75/4.40

Table 4–4: ND results of Traffic and Electricity datasets. The best and the second best results in each column are shown in bold and marked with underline respectively. Lower numbers are better. The results specified by * are the ones obtained via training the models.

Models	Traffic	Electricity
TRMF	0.187	0.255
DeepAR	0.178^*	0.698^*
DeepState	0.795^*	0.798^*
DeepFactors	0.408^*	1.205^*
DeepGLO	0.148	0.155^*
N-BEATS	**0.112**	<u>0.171</u>
FC-GAGA	<u>0.137</u>	**0.135**

node embedding matrix plays an important role. The results of using separate node embedding matrices to construct the graph weight matrix $\mathbf{W}$ (A.3) show that although there are probably directions of traffic flow, the node relationships in the traffic datasets either exhibit symmetry, or the additional flexibility of the model is outweighed by the removal of the regularization effect. Again, this result indicates that using the node embedding matrix is important since the difference between A.3 and FC-GAGA compared to the difference between A.1 and FC-GAGA is much lower. The reason is that the node embedding matrix identifies which one of the nodes is being targeted in both the time gate block and fully connected time-series block.

- *Layer constraint:* The experimental results are provided in Table 4–6. As it can be seen, considerable improvement for both datasets is obtained by constraining the graph edge weight matrix $\mathbf{W}$ of the 4-th layer to the identity matrix. The results suggest that since the 4-th layer is responsible for making corrections, it is better to base the corrections only on the node's own history.

Table 4–5: Experiment results on METR-LA and PEMS-BAY datasets for the *Asymmetric Graph Embedding Matrix* test. A.1: Learning the graph weights independently, A.2: Extra fully connected layer instead of node embedding, and A.3: Using separate node embeddings to build the graph weight.

	METR-LA (15/30/60 min)		
Models	MAE	MAPE (%)	RMSE
A.1	2.91/3.39/4.00	7.75/9.51/12.03	5.68/6.83/8.19
A.2	3.06/3.70/4.69	8.15/10.53/14.05	5.97/7.38/9.22
A.3	2.78/3.13/3.52	7.26/8.68/10.29	5.38/6.38/7.34
FC-GAGA	**2.75/3.10/3.51**	**7.25/8.57/10.14**	**5.34/6.30/7.31**
	PEMS-BAY (15/30/60 min)		
Models	MAE	MAPE (%)	RMSE
A.1	1.41/1.87/2.46	2.92/4.14/5.65	3.06/4.32/5.77
A.2	1.52/2.32/3.06	3.16/5.03/7.26	3.29/4.93/6.75
A.3	1.36/1.69/1.98	2.87/3.84/4.68	2.86/3.82/4.53
FC-GAGA	**1.36/1.68/1.97**	**2.87/3.80/4.67**	**2.86/3.80/4.52**

Table 4–6: Experiment results on METR-LA and PEMS-BAY datasets for the *Layer Constraint* test. The best and the second best results in each column are shown in bold and marked with underline respectively.

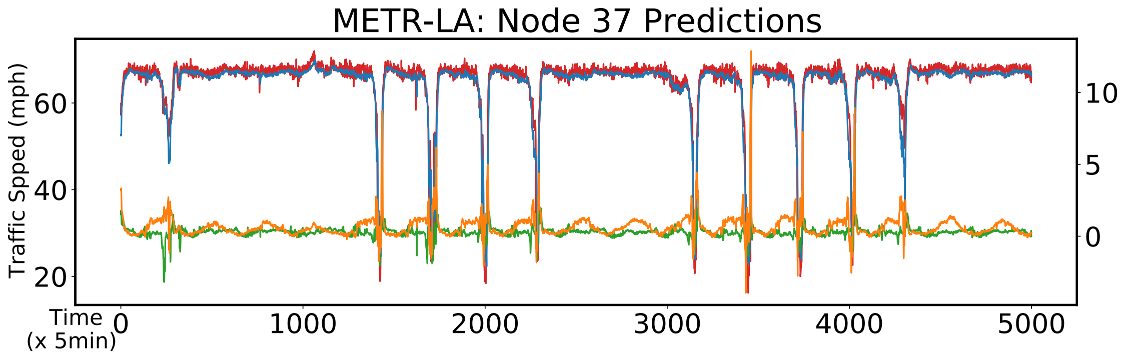
	METR-LA (15/30/60 min)		
Models	MAE	MAPE (%)	RMSE
1st layer	2.78/3.12/3.51	7.29/8.59/10.12	5.39/6.39/7.31
2nd layer	2.75/3.11/$\underline{3.50}$	7.22/8.59/10.07	5.34/6.33/$\underline{7.28}$
3rd layer	$\underline{2.75}$/$\underline{3.09}$/3.51	$\underline{7.13}$/$\underline{8.46}$/$\underline{9.98}$	$\underline{5.30}$/$\underline{6.30}$/7.29
1st & 2nd layers	2.81/3.15/3.54	7.46/8.86/10.28	5.47/6.45/7.33
1st & 3rd layers	2.80/3.15/3.55	7.44/8.81/10.38	5.47/6.43/7.35
2nd & 3rd layers	2.78/3.14/3.56	7.29/8.67/10.24	5.39/6.38/7.35
all 3 layers	2.95/3.48/4.21	7.76/9.70/12.18	5.80/7.05/8.57
4th layer	**2.70/3.04/3.45**	**7.01/8.31/9.88**	**5.24/6.19/7.19**
5th layer	2.73/3.07/3.47	7.03/8.29/9.91	5.28/6.23/7.21
FC-GAGA	2.75/3.10/3.51	7.25/8.57/10.14	5.34/6.30/7.31

	PEMS-BAY (15/30/60 min)		
Models	MAE	MAPE (%)	RMSE
1st layer	1.36/1.68/1.94	2.87/3.82/4.64	2.85/3.79/4.46
2nd layer	1.35/1.67/1.94	$\underline{2.84}$/$\underline{3.76}$/$\underline{4.55}$	**2.82/3.75**/$\underline{4.41}$
3rd layer	**1.33/1.66/1.92**	$\underline{2.84}$/3.79/4.56	**2.82**/3.76/$\underline{4.41}$
1st & 2nd layers	1.36/1.69/1.95	2.90/3.89/4.67	2.88/3.87/4.50
1st & 3rd layers	1.36/1.69/1.96	2.84/3.79/4.62	2.87/3.81/4.49
2nd & 3rd layers	1.36/1.68/1.95	2.92/3.91/4.69	2.88/3.87/4.50
all 3 layers	1.40/1.83/2.38	2.90/4.02/5.41	2.99/4.16/5.47
4th layer	$\underline{1.34}$/**1.66**/$\underline{1.93}$	**2.82/3.71/4.48**	**2.82/3.75/4.40**
5th layer	1.35/1.67/1.94	2.83/3.75/4.56%	2.86/3.79/4.46
FC-GAGA	1.36/1.68/1.97	2.87/ 3.80/4.67	2.86/3.80/4.52

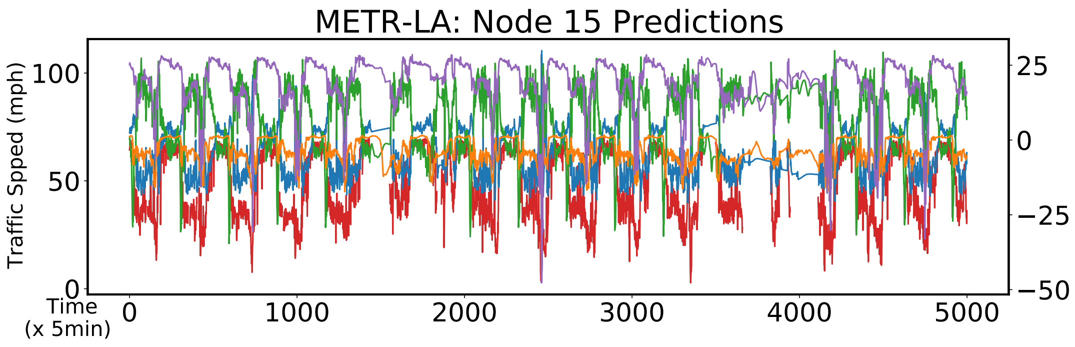
4.3.6 Qualitative Results

The final FC-GAGA forecast is composed of the average of the forecasts of individual layers. Figures 4–4 and 4–5 present examples of how the different layers in the architecture contribute to the final 15 min ahead prediction (after scaling by the averaging factor 1/3) for the METR-LA and PEMS-BAY datasets, respectively. Figures 4–6 and 4–7 further illustrate the layer contributions. Figures 4–8 and 4–9 show the same information, but focus on shorter time windows. We can see that the role of the first layer is mainly to provide a (relatively accurate) baseline forecast in all cases, while at the same time accounting for some seasonal effects, and then layers 2, 3 and 4 provide modifications to enhance the accuracy. These layers provide iterative correction terms to the original baseline produced by layer 1, based on the most recent data. This is especially evident for layer 3 and 4 whose outputs are inactive most of the time, becoming active when an abrupt change is required because the observed signals undergo significant changes (primarily during rush-hour on weekdays). For several of the nodes (e.g., METR-LA node 37 and 124, PEMS-BAY nodes 30 and 179), we can see that layer 2 is responsible for modeling a daily periodic fluctuation.

In Figures 4–8 and 4–9, clearer evidence of the compensation effects of layers 2 and 3 is demonstrated. For example, for node 31 in the METR-LA dataset, we see in Figure 4–8 that the layer 1 prediction lags behind the true signal after the sudden drop, and struggles to return to the same level for close to an hour. Layer 2 (orange) compensates for this by providing a significant positive component to the prediction only during this period when the layer 1 prediction is trying to recover. For PEMS-BAY node 182 in Figure 4–9 it is clear that layers 2 and 3 are compensating for the prediction lag of layer 1 whenever there are significant changes in the true signal.



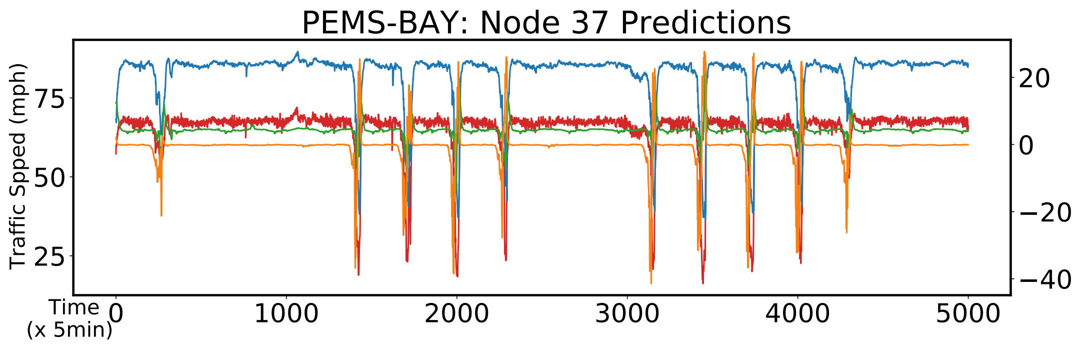
(a)



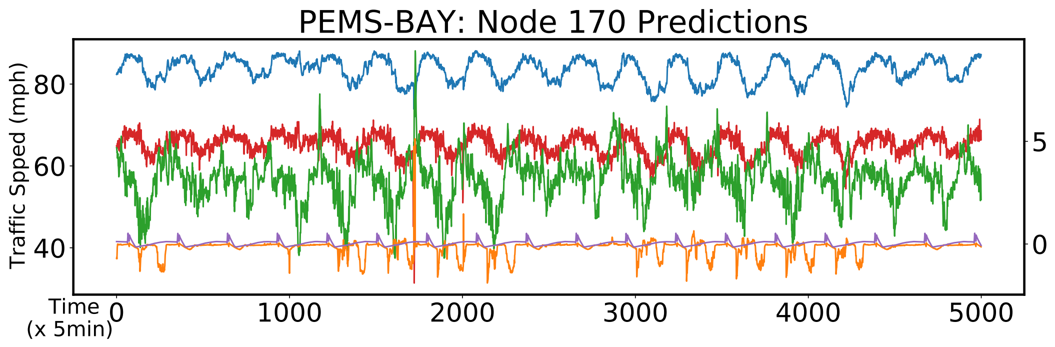
(b)

Figure 4–4: (a) 15 min ahead forecasts of FC-GAGA with 3 layers. (b) 15 min ahead forecasts of FC-GAGA(4 layers)‡. Blue, green, orange and purple lines depict the partial forecasts produced by layers 1, 2, 3, and 4 of the architecture respectively. Blue & red: left axis; orange & green & purple: right axis.

Next, we show in Figures 4–10 and 4–11 the geographical distribution of the weights $\mathbf{W}_{i,j}$ in the graph gate, specified in Equation (4.2), for layers 1–3, as learned for the METR-LA and PEMS-BAY datasets, respectively. Each layer is provided the



(a)



(b)

Figure 4–5: (a) 15 min ahead forecasts of FC-GAGA with 3 layers. (b) 15 min ahead forecasts of FC-GAGA(4 layers)‡. Blue, green, orange and purple lines depict the partial forecasts produced by layers 1, 2, 3, and 4 of the architecture respectively. Blue & red: left axis; orange & green & purple: right axis.

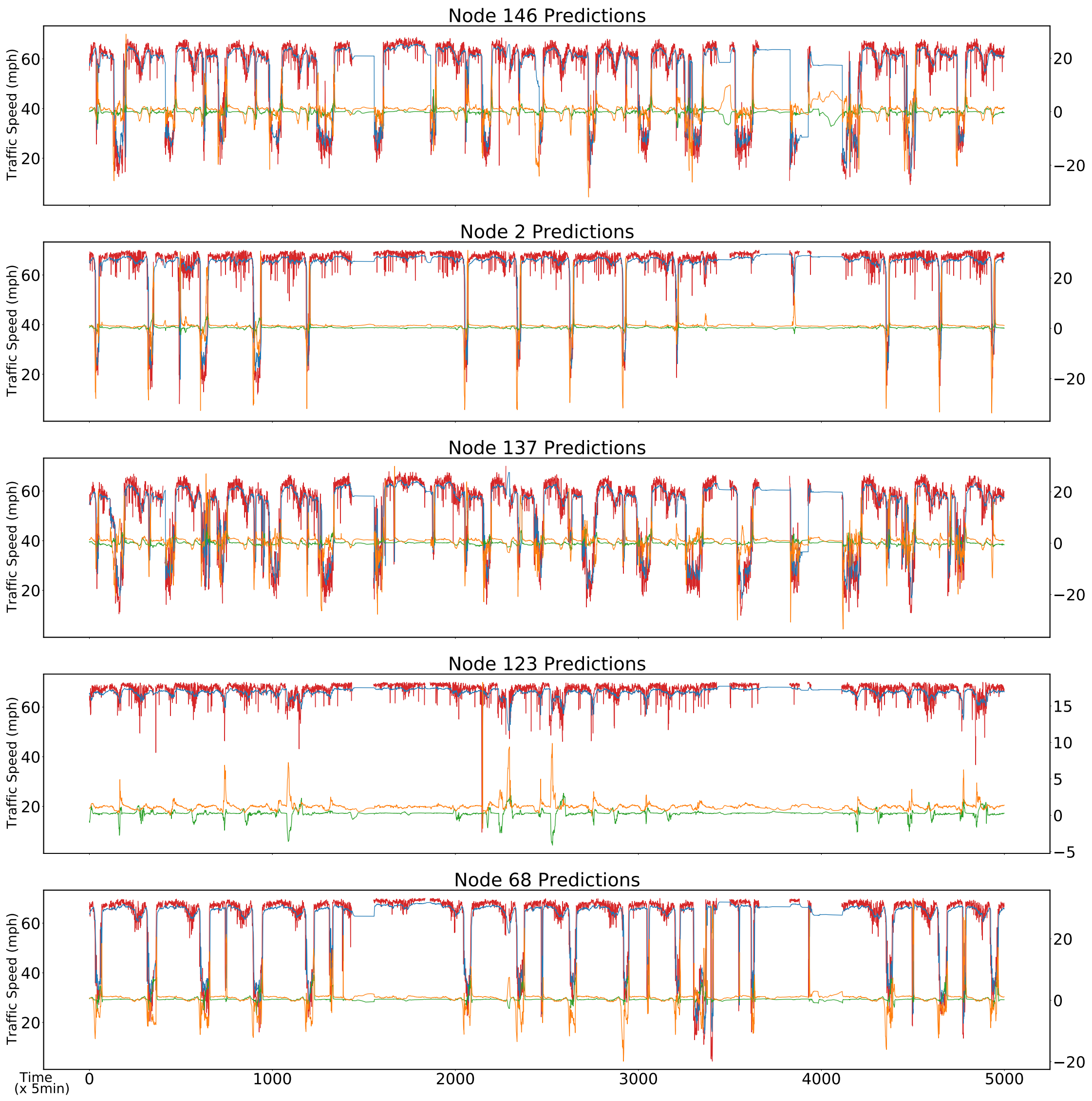


Figure 4–6: FC-GAGA 15 min ahead forecasts for different nodes in METR-LA dataset. Blue, green and orange lines depict the partial forecasts produced by layers 1, 2, and 3 of the architecture respectively. Magnitudes of blue and red lines are indicated by the left axis labels; magnitudes of orange and green lines are indicated by the right axis labels.

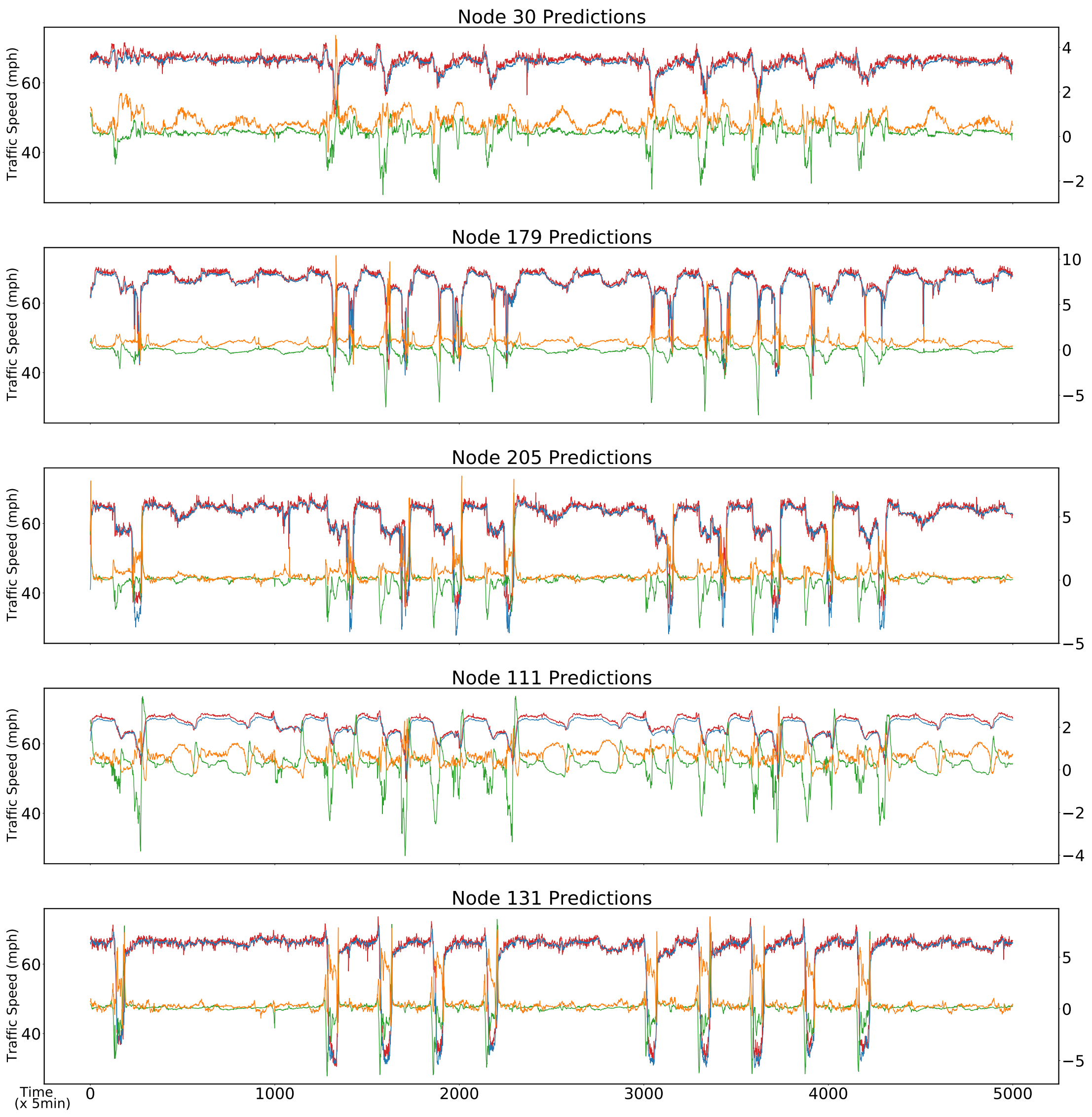


Figure 4–7: FC-GAGA 15 min ahead forecasts for different nodes in PEMS-BAY dataset. Blue, green and orange lines depict the partial forecasts produced by layers 1, 2, and 3 of the architecture respectively.Magnitudes of blue and red lines are indicated by the left axis labels; magnitudes of orange and green lines are indicated by the right axis labels.

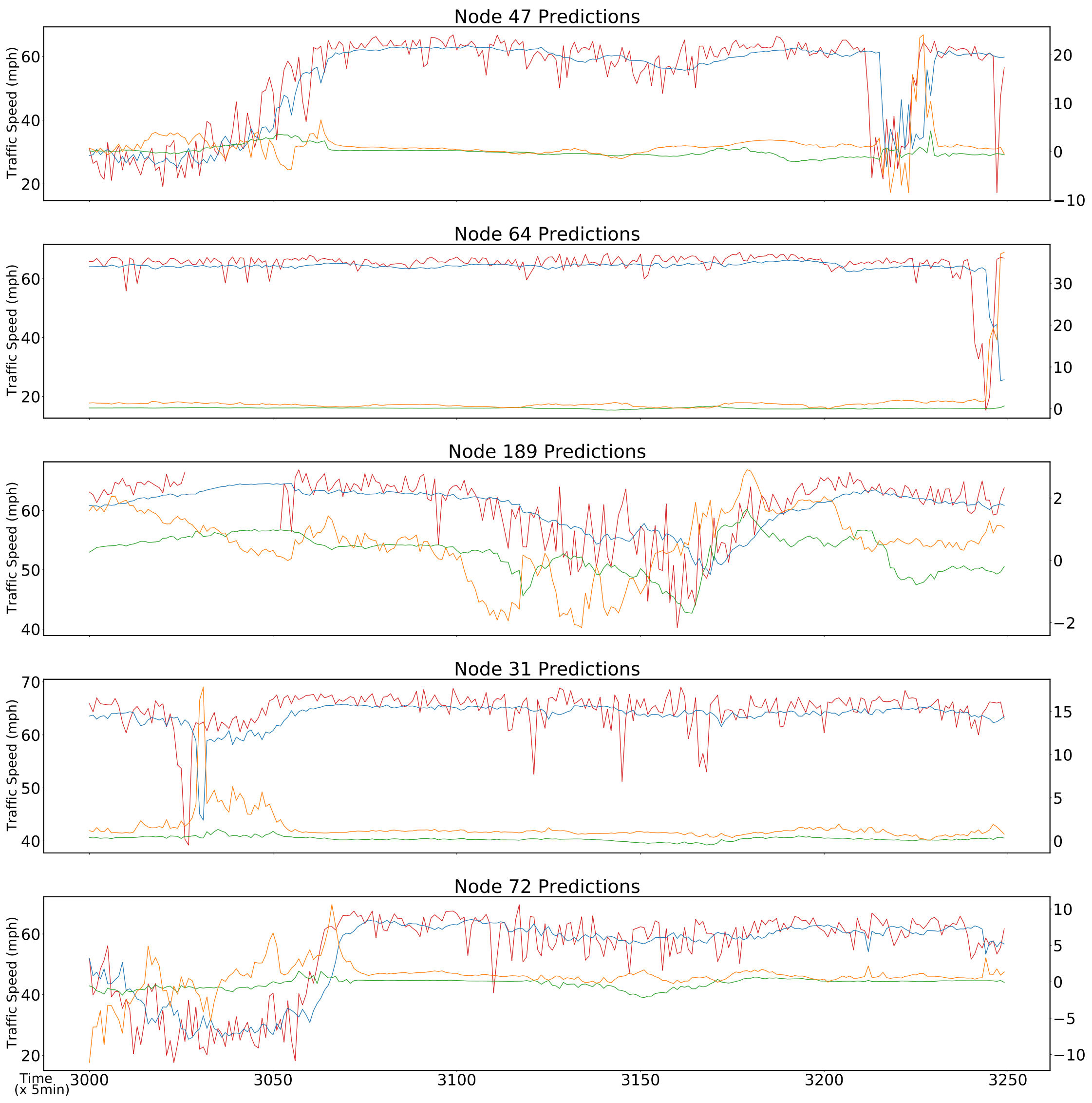


Figure 4–8: FC-GAGA 15 min ahead forecasts for different nodes in METR-LA dataset from time 3000 to 3250. Blue, green and orange lines depict the partial forecasts produced by layers 1, 2, and 3 of the architecture respectively. Magnitudes of blue and red lines are indicated by the left axis labels; magnitudes of orange and green lines are indicated by the right axis labels.

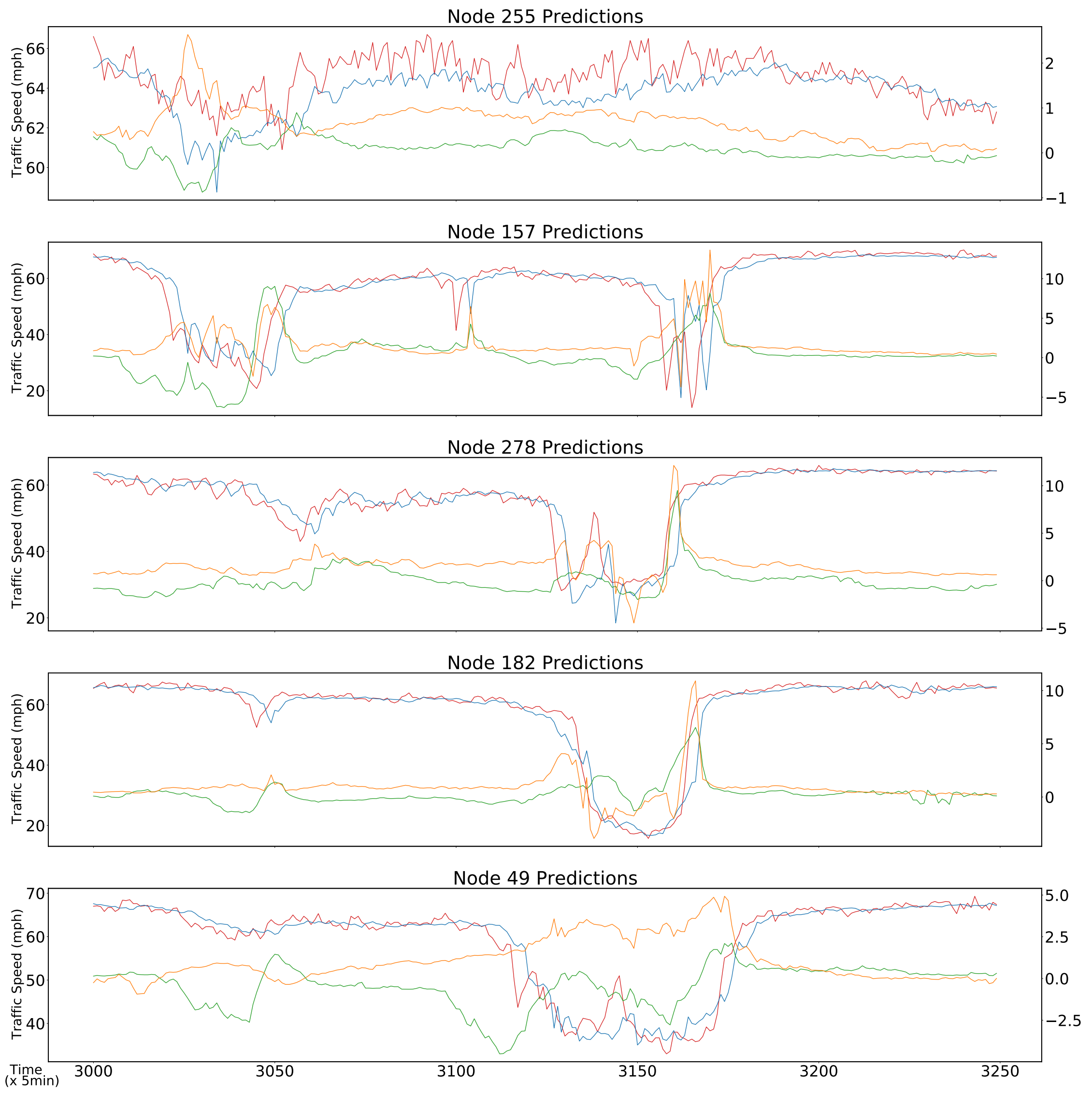


Figure 4–9: FC-GAGA 15 min ahead forecasts for different nodes in PEMS-BAY dataset from time 3000 to 3250. Blue, green and orange lines depict the partial forecasts produced by layers 1, 2, and 3 of the architecture respectively. Magnitudes of blue and red lines are indicated by the left axis labels; magnitudes of orange and green lines are indicated by the right axis labels.

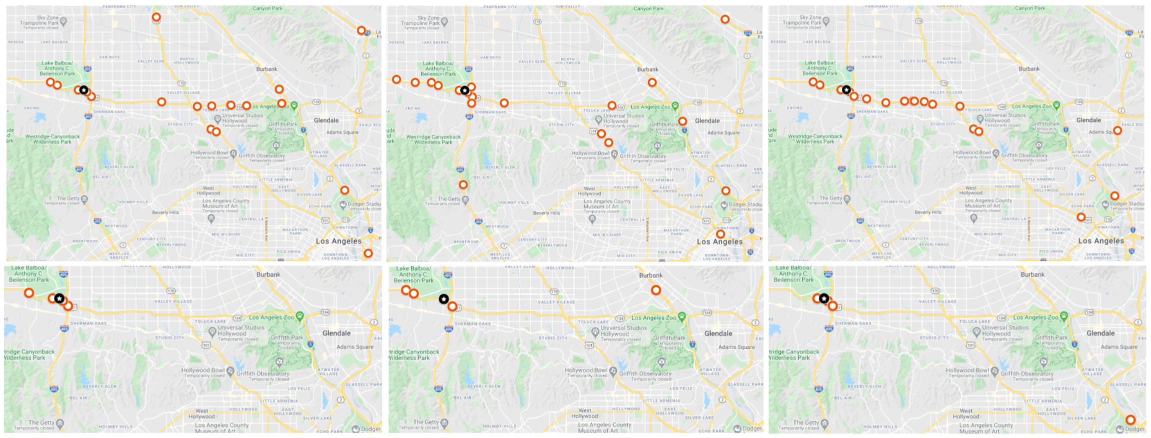


Figure 4–10: Spatial distribution of weights: maps of 20 and 4 highest weighted nodes for layers 1, 2, and 3 (left to right). The white star in the black circle is the forecasted node (node 146 in METR-LA).

freedom to learn its relationship across graph nodes; the learned relationships differ significantly across layers, indicating information aggregation from different spatial regions. There are usually fewer nodes with significant weight for the third layer, and they tend to be located closer to the target node for which the forecast is produced (see Figures 4–10 and 4–11, the maps of the four largest weights).

Figure 4–12 depicts the average weight by weight rank, i.e., for the largest weight as an example, what is the average value. The left panels of this figure show clearly that the gating is less strictly enforced in the first layer, as the weights for layer 1 are higher than those of layer 2, which are in turn higher than layer 3, implying stricter gating in Equation (4.2) for layers 2 and 3. This illustrates how layer 1 incorporates information from many nodes to form its prediction, whereas layers 2 and 3 use far fewer nodes (the weight gating blocks the contribution from many nodes). The left panels of Figures 4–10 and 4–11 support this intuition since the

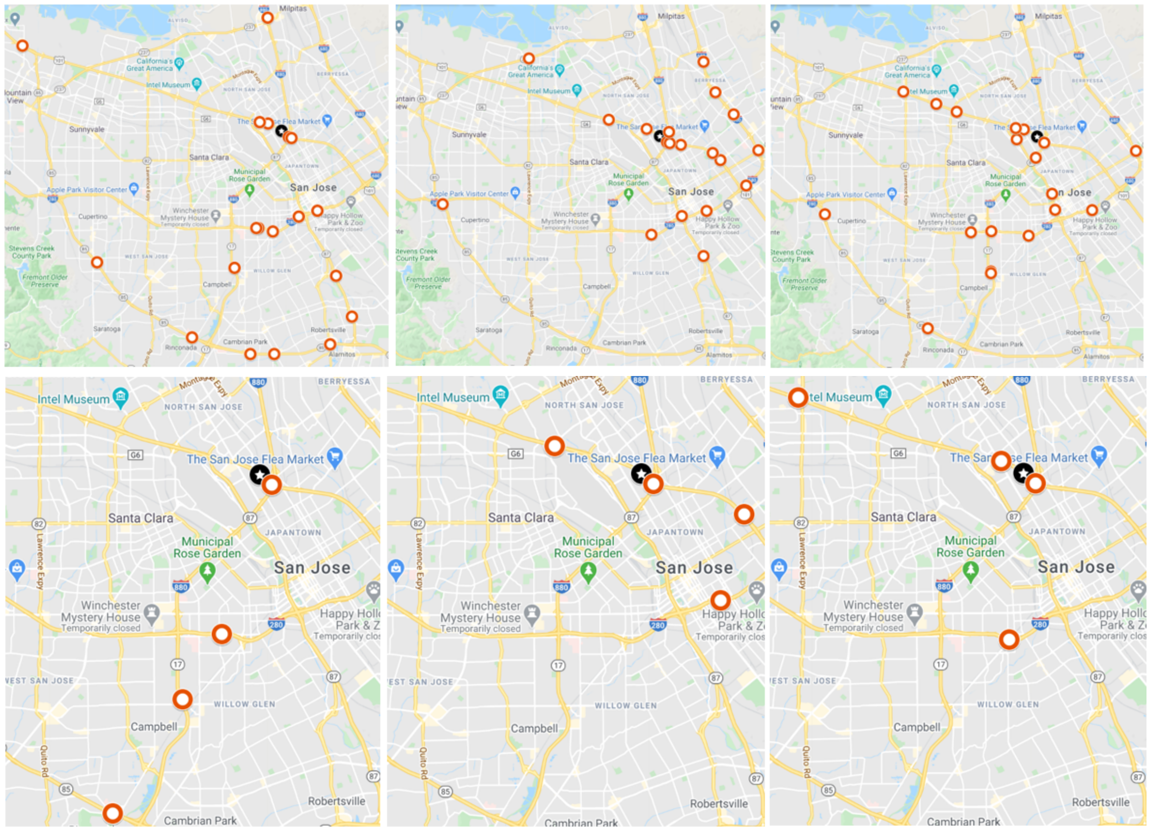
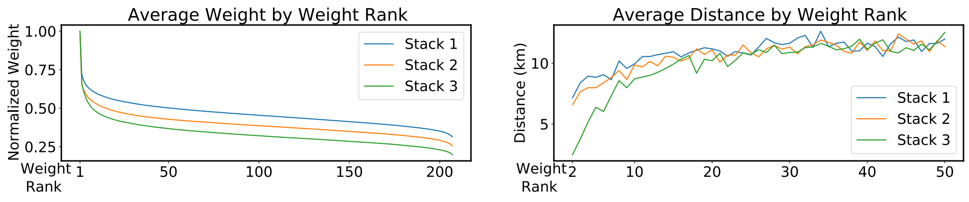


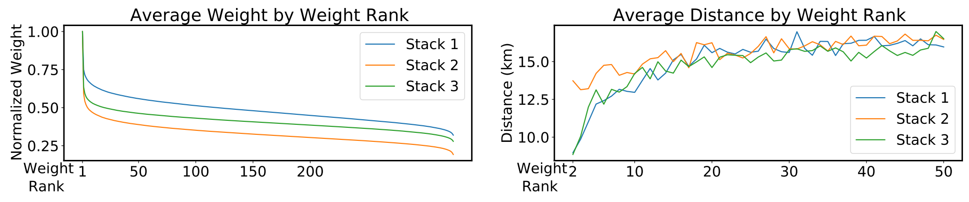
Figure 4–11: Spatial distribution of weights: maps of 20 and 4 highest weighted nodes for layers 1, 2, and 3 (left to right). The white star in the black circle is the forecasted node (node 216 in PEMS-BAY).

geographic distribution of values is more dispersed. We interpret this as indicating that in layer 1, FC-GAGA collects information across various nodes and geographical locations to construct a stable baseline forecast. As we move from layer 2 to layer 3, we can see that the nodes with highest graph weights more tightly concentrate around the target node (see Figures 4–10 and 4–11, middle and right panels, and the right panel of Figure 4–12 (a)).

In the right panel of Figure 4–12 (b), we show the average distance from the forecasted nodes for each weight rank for the PEMS-BAY dataset. The results have similarities with the results depicted in Figure 4–12 (a) for the METR-LA dataset; however, there are also differences. As for the METR-LA dataset, the average distance increases with the weight rank, especially for layers 1 and 3, particularly for the first 20 ranks. For the PEMS-BAY dataset, we see that the distance for layer 2 does not grow as rapidly. This suggests that for PEMS-BAY, layer 2 often incorporates information from nodes that are further away.



(a)



(b)

Figure 4–12: (a) Average of graph gate weights $\mathbf{W}_{i,j}$ normalized by the self-weight $\mathbf{W}_{i,i}$ (left) and their average distances from the forecasted node (right) per FC-GAGA layer in METR-LA dataset. (b) Average of graph gate weights $\mathbf{W}_{i,j}$ normalized by the self-weight $\mathbf{W}_{i,i}$ (left) and their average distances from the forecasted node (right) per FC-GAGA layer in PEMS-BAY dataset.

Table 4–7: Ablation study: the effectiveness of the FC-GAGA layer stacking. 4I‡ uses the identity for the fourth layer weight matrix.

	METR-LA (15/30/60 min)		
Layers	MAE	MAPE (%)	RMSE
1	2.80/3.17/3.63	7.36/8.82/10.55	5.44/6.44/7.48
2	2.77/3.13/3.54	7.30/8.74/10.41	5.37/6.39/7.36
3	2.75/3.10/3.51	7.25/8.57/10.14	5.34/6.30/7.31
4	2.75/3.10/3.52	7.21/8.54/10.19	5.34/6.34/7.34
4I‡	2.70/3.04/3.45	7.01/8.31/9.88	5.24/6.19/7.19
	PEMS-BAY (15/30/60 min)		
Layers	MAE	MAPE (%)	RMSE
1	1.35/1.69/2.00	2.85/3.85/4.78	2.85/3.83/4.61
2	1.36/1.68/1.97	2.87/3.80/4.64	2.86/3.81/4.52
3	1.36/1.68/1.97	2.87/3.80/4.67	2.86/3.80/4.52
4	1.35/1.69/1.98	2.83/3.78/4.66	2.86/3.83/4.57
4I‡	1.34/1.66/1.93	2.82/3.71/4.48	2.82/3.75/4.40

4.3.7 Ablation Studies

We conduct ablation studies with the goals of validating the effectiveness of the FC-GAGA layer stacking mechanism, the graph gating mechanism and the time gate block. Table 4–7 demonstrates the performance of FC-GAGA as a function of the number of layers. Increasing the number of layers leads to substantial improvement on the METR-LA dataset, while on PEMS-BAY, the number of layers does not affect performance significantly. Forecasting for the METR-LA dataset is known to be a more complex problem than forecasting for PEMS-BAY due to the more erratic nature of its time series. This implies that increasing the number of FC-GAGA layers to solve harder problems may bring additional accuracy benefits, while using only one FC-GAGA layer to solve an easier problem may benefit from the computational efficiency standpoint (the runtime scales approximately linearly with the number of layers). The final row in the table ($4I^{\ddagger}$) shows the performance when the fourth layer is set to the identity so that the layer focuses on forming a prediction using only the history of each node. This approach leads to a noticeable improvement; forcing one layer to learn univariate relationships can be beneficial.

Table 4–8: Ablation study: the effectiveness of the FC-GAGA graph gate and time gate.

		METR-LA (15/30/60 min)		
Models	Layers	MAE	MAPE (%)	RMSE
(1) MV N-BEATS	3	3.00/3.60/4.44	7.96/10.22/13.59	5.90/7.28/8.92
(2) add time gate	3	2.86/3.24/3.68	7.60/9.13/10.81	5.61/6.66/7.67
(3) add graph gate	3	2.81/3.21/3.67	7.33/8.75/10.51	5.36/6.36/7.45
FC-GAGA	3	2.75/3.10/3.51	7.25/8.57/10.14	5.34/6.30/7.31

		PEMS-BAY (15/30/60 min)		
Models	Layers	MAE	MAPE (%)	RMSE
(1) MV N-BEATS	3	1.41/1.86/2.40	2.94/4.20/5.90	3.05/4.26/5.48
(2) add time gate	3	1.37/1.70/1.99	2.87/3.83/4.68	2.89/3.86/4.57
(3) add graph gate	3	1.35/1.69/2.00	2.84/3.79/4.72	2.86/3.82/4.58
FC-GAGA	3	1.36/1.68/1.97	2.87/3.80/4.67	2.86/3.80/4.52

Table 4–8 provides the results of ablating the graph gate and time gate mechanisms with the original architecture, a 3-layer FC-GAGA network. Both the time gate and graph gate individually lead to improvements over a straightforward multivariate N-BEATS model, i.e., accepting concatenated histories, and then combine to offer further improvement. Table 4–9 examines different approaches for the graph

Table 4–9: Ablation study: the effectiveness of the FC-GAGA graph gate block.

		METR-LA (15/30/60 min)		
Graph gate **W**	Layers	MAE	MAPE (%)	RMSE
graph attention	3	2.99/3.56/4.43	7.90/10.00/13.15	5.83/7.15/8.89
identity	3	2.97/3.54/4.35	7.80/9.88/12.77	5.87/7.25/8.93
ones	3	2.87/3.24/3.67	7.71/9.22/10.80	5.67/6.71/7.65
shared learnable	3	2.77/3.13/3.57	7.20/8.53/10.09	5.36/6.35/7.37
learnable first layer	3	2.77/3.13/3.55	7.28/8.67/10.23	5.40/6.41/7.44
FC-GAGA	3	2.75/3.10/3.51	7.25/8.57/10.14	5.34/6.30/7.31

		PEMS-BAY (15/30/60 min)		
Graph gate **W**	Layers	MAE	MAPE (%)	RMSE
graph attention	3	1.44/1.92/2.57	3.00/4.32/6.07	3.08/4.38/5.86
identity	3	1.41/1.86/2.44	2.92/4.10/5.58	3.05/4.27/5.66
ones	3	1.38/1.70/2.00	2.89/3.82/4.70	2.89/3.86/4.60
shared learnable	3	1.37/1.72/2.01	2.95/4.00/4.84	2.88/3.90/4.62
learnable first layer	3	1.36/1.69/1.99	2.87/3.82/4.70	2.86/3.83/4.57
FC-GAGA	3	1.36/1.68/1.97	2.87/3.80/4.67	2.86/3.80/4.52

gate. "Graph attention" is a standard graph attention approach that does not perform hard gating. We see that the sparsification provided by our proposed gate is essential; graph attention is even outperformed by the univariate FC-GAGA model ("identity"). The univariate FC-GAGA outperforms all univariate methods in Table 4–3 (ARIMA, SVR, FNN, FC-LSTM, and N-BEATS) by a large margin. When **W** is set to all ones ("ones"), FC-GAGA is provided with more information from the other graph nodes and can learn relationships between different nodes, but it cannot emphasize influential nodes. We examine three learnable options: "shared learnable" where all layers share a single learnable **W**, "learnable first layer" where

W associated with the first layer is learnable, and it is set to the ones matrix for other layers, and finally the fully learnable FC-GAGA approach. According to the results, allowing the architecture to learn a different weight matrix for each layer leads to the best prediction performance, and the additional computational overhead is minor.

Table 4–10: Profiling results: total training and evaluation runtime, and GPU memory utilization. Measured using official DCRNN, Graph WaveNet and GMAN codes and our non-optimized tensorflow implementation of FC-GAGA on NVIDIA P100 GPU in the default Google Colab environment.

Models	METR-LA	
	Runtime, min	GPU memory, GB
DCRNN	358	8.63
Graph WaveNet	90	2.14
GMAN	215	9.84
FC-GAGA, 3 layers	37	0.93
	PEMS-BAY	
	Runtime, min	GPU memory, GB
DCRNN	828	8.63
Graph WaveNet	192	2.75
GMAN	366	9.84
FC-GAGA, 3 layers	69	1.47

4.3.8 Profiling Results

To confirm the computational efficiency of FC-GAGA, a profiling experiment is conducted using a P100 GPU in the default Google Colab environment. We profiled the original codes provided by the authors of DCRNN [15], Graph Wavenet [22], and GMAN [17]. The profiling experiment uses our TensorFlow 2.0 implementation of FC-GAGA, relying on standard Keras layer definitions, with no attempt to optimize for memory or speed. Table 4–10 clearly shows that FC-GAGA is more computationally effective as it consumes approximately half the memory and compute time of Graph WaveNet, is about 10 times faster than DCRNN and 6 times faster than

GMAN, and about 5-10 times more memory efficient than these algorithms. We can also see that FC-GAGA scales well between METR-LA (207 nodes) and PEMS-BAY (325 nodes) datasets, which may be an important property for handling larger scale problems with thousands of nodes.

4.4 Summary

In this chapter, a novel gated graph neural architecture, called FC-GAGA, has been introduced. The efficacy of this model has been demonstrated through extensive experiments, including a thorough assessment of how the proposed approach compares to state-of-the-art baselines. The results show that FC-GAGA compares favourably even against graph-based models that rely on additional external graph definitions on the two graph-based datasets. Moreover, the model achieves comparable or better results for two non-graph datasets. In order to better understand the model, we inspect the contribution of each layer to the final prediction and examine the graph edge weight matrix $\mathbf{W}$. Further ablation studies have been conducted to validate the effectiveness of each part of the model. Finally, a profiling experiment establishes the computational efficiency of FC-GAGA.

CHAPTER 5
Conclusion

In this thesis, a novel neural architecture for spatio-temporal forecasting, which we call FC-GAGA (Fully Connected Gated Graph Architecture), is proposed and empirically validated. FC-GAGA combines a fully connected TS model with temporal and graph gating mechanisms that are generally applicable and computationally efficient. The proposed approach offers three advantages. First, the architecture does not depend on any knowledge of the underlying graph; instead, it focuses on learning all the required non-linear predictive relationships. Second, the basic layer of the architecture is stackable, allowing every layer to learn its own graph structure. This provides the model with the ability to learn a very general non-Markovian information diffusion process that can be learned effectively, as we showed empirically. Finally, FC-GAGA has a hard gate mechanism, which is efficient both in terms of memory and computation, as demonstrated via profiling.

We empirically show that the proposed model can be learned efficiently from the data to capture non-Markovian relations across multiple variables over layers in the architecture, resulting in excellent generalization performance. We further profile FC-GAGA's training and inference runtime and demonstrate that it is several times more efficient in its use of GPU memory and compute than existing models with comparable accuracy. Our results provide compelling positive evidence to stimulate the development of fully connected architectures for graph-based information processing.

Our empirical study supports that the magnitudes of the graph edge weights introduced in the graph gate block correlate well with the geographical proximity of nodes. The conducted experiments show that without the hard gating, the graph weighting procedure is much less effective. Ablation studies also demonstrate that when using the efficient fully-connected residual time-series prediction module, it is not sufficient to apply standard graph attention, i.e., the sparsification achieved by our proposed novel graph gate is essential in achieving good predictive performance. We conjecture that this is because, for each target node, there are only a few relevant nodes at a given layer; therefore, the input to the fully connected architecture should be sparse. Hard gating encourages sparsity, while soft gating provides input that is not sparse, which can lead to overwhelming a fully connected network with far too many low-magnitude inputs originating from many nodes in the graph. Additionally, based on the provided complexity analysis, the proposed graph gate design has N times smaller complexity, $O(N^2)$, compared to most state-of-the-art methods in the literature that exhibit $O(N^3)$ complexity (e.g., DCRNN, Graph WaveNet, GMAN, and TRMF).

Future Work. One of the potential extensions to FC-GAGA is to effectively have the graph edge weight matrix $\mathbf{W}$ that evolves over time. In the current model, the predictions are generated by focusing on the long-term dependencies. An alternative approach might then be an architecture that is a combination of the model arising from the long-term dependencies (current $\mathbf{W}$) and the one arising from the short-term relationships. The idea might be that we normally make predictions at node j based mainly on nodes $\{i, r, w\}$. However, for instance when there is very heavy traffic, it is valuable to also learn from other nodes $\{i, s, t\}$. Through this way, we can recognize the heavy traffic scenario by generating some kind of context vector.

Although the proposed method in this thesis shows promising results for point forecasts on both graph-based and non-graph datasets, the model has a serious disadvantage as it cannot measure the uncertainty in the predictions. Since one of the major benefits of time series forecasting is to make decisions based on predictions, it is essential to provide a prediction interval. Therefore, one important avenue to pursue is to extend the existing model to a probabilistic forecasting approach.

REFERENCES

[1] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, “N-BEATS: Neural basis expansion analysis for interpretable time series forecasting,” in *Proc. Int. Conf. Learning Representations (ICLR)*, 2020.

[2] A. A. Neale, “Weather forecasting: Magic, art, science and hypnosis.” *Weather and Climate*, vol. 5, no. 1, pp. 2–5, 1985.

[3] Y. LeCun, “Deep learning hardware: Past, present, and future,” in *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2019, pp. 12–19.

[4] W. Bao, J. Yue, and Y. Rao, “A deep learning framework for financial time series using stacked autoencoders and long-short term memory,” *PloS One*, vol. 12, no. 7, 2017.

[5] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. W. Cottrell, “A dual-stage attention-based recurrent neural network for time series prediction,” in *Proc. Int. Joint Conf. Artificial Intelligence*, 2017.

[6] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, “Modeling long-and short-term temporal patterns with deep neural networks,” in *Proc. Int. ACM SIGIR Conf. Research & Development in Information Retrieval*, 2018, pp. 95–104.

[7] T. Guo and T. Lin, “Multi-variable lstm neural network for autoregressive exogenous model,” *arXiv preprint arXiv:1806.06384*, 2018.

[8] Y.-Y. Chang, F.-Y. Sun, Y.-H. Wu, and S.-D. Lin, “A memory-network based solution for multivariate time-series forecasting,” *arXiv preprint arXiv:1809.02105*, 2018.

[9] L. Munkhdalai, T. Munkhdalai, K. H. Park, T. Amarbayasgalan, E. Erdenebaatar, H. W. Park, and K. H. Ryu, “An end-to-end adaptive input selection with dynamic weights for forecasting multivariate time series,” *IEEE Access*, vol. 7, pp. 99 099–99 114, 2019.

[10] F. Liu, Y. Lu, and M. Cai, "A hybrid method with adaptive sub-series clustering and attention-based stacked residual LSTMs for multivariate time series forecasting," *IEEE Access*, vol. 8, pp. 62 423–62 438, 2020.

[11] P. K. Singh, Y. Gupta, N. Jha, and A. Rajan, "Fashion retail: Forecasting demand for new items," *arXiv preprint arXiv:1907.01960*, 2019.

[12] D. Rolnick, P. L. Donti, L. H. Kaack, K. Kochanski, A. Lacoste, K. Sankaran, A. S. Ross, N. Milojevic-Dupont, N. Jaques, A. Waldman-Brown *et al.*, "Tackling climate change with machine learning," *arXiv preprint arXiv:1906.05433*, 2019.

[13] X. Shi, H. Qi, Y. Shen, G. Wu, and B. Yin, "A spatial-temporal attention approach for traffic prediction," *IEEE Trans. Intelligent Transportation Systems*, pp. 1–10, Apr. 2020.

[14] L. Bai, L. Yao, S. Kanhere, X. Wang, and Q. Sheng, "STG2Seq: Spatial-temporal graph to sequence model for multi-step passenger demand forecasting," in *Proc. Int. Joint Conf. Artificial Intelligence*, 2019, pp. 1981–1987.

[15] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2018.

[16] B. Yu, H. Yin, and Z. Zhu, "Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting," in *Proc. Int. Joint Conf. Artificial Intelligence*, 2018.

[17] C. Zheng, X. Fan, C. Wang, and J. Qi, "GMAN: A graph multi-attention network for traffic prediction," in *Proc. AAAI Int. Conf. Artificial Intelligence*, 2020.

[18] B. N. Oreshkin, A. Amini, L. Coyle, and M. J. Coates, "FC-GAGA: Fully connected gated graph architecture for spatio-temporal traffic forecasting," in *Proc. AAAI Int. Conf. Artificial Intelligence*, 2021.

[19] W. L. Hamilton, *Graph Representation Learning.* Morgan & Claypool, 2020.

[20] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. on Learning Representations (ICLR)*, 2017.

[21] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *Proc. AAAI Conf. Artificial Intelligence*, 2019.

[22] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph wavenet for deep spatial-temporal graph modeling," in *Proc. Int. Joint Conf. Artificial Intelligence*, 2019, pp. 1907–1913.

[23] C. Song, Y. Lin, S. Guo, and H. Wan, "Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting," in *Proc. AAAI Conf. Artificial Intelligence*, 2020.

[24] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comp.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[25] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[26] S. Makridakis and M. Hibon, "The m3-competition: results, conclusions and implications," *International Journal of Forecasting*, vol. 16, no. 4, pp. 451–476, 2000.

[27] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "The m4 competition: Results, findings, conclusion and way forward," *International Journal of Forecasting*, vol. 34, no. 4, pp. 802–808, 2018.

[28] G. Athanasopoulos, R. J. Hyndman, H. Song, and D. C. Wu, "The tourism forecasting competition," *International Journal of Forecasting*, vol. 27, no. 3, pp. 822–844, 2011.

[29] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[30] S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski, "Deep state space models for time series forecasting," in *Proc. Int. Conf. Neural Info. Process. (NIPS)*, 2018, pp. 7785–7794.

[31] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks," *International Journal of Forecasting*, 2019.

[32] S. Smyl, "A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting," *International Journal of Forecasting*, vol. 36, no. 1, pp. 75 – 85, 2020.

[33] G. U. Yule, "On a method of investigating periodicities disturbed series, with special reference to Wolfer's sunspot numbers," *Philosophical Trans. Royal Soc. London. Series A*, vol. 226, no. 636-646, pp. 267–298, 1927.

[34] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control.* John Wiley & Sons, 2015.

[35] S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018.

[36] F. R. Bach and M. I. Jordan, "Learning graphical models for stationary time series," *IEEE Trans. Signal Processing*, vol. 52, no. 8, pp. 2189–2199, 2004.

[37] J. Songsiri and L. Vandenberghe, "Topology selection in graphical models of autoregressive processes," *Journal of Machine Learning Research*, vol. 11, no. Oct, pp. 2671–2705, 2010.

[38] A. Bolstad, B. D. Van Veen, and R. Nowak, "Causal network inference via group sparse regularization," *IEEE Trans. Signal Processing*, vol. 59, no. 6, pp. 2628–2641, 2011.

[39] J. Mei and J. M. Moura, "Signal processing on graphs: Causal modeling of unstructured data," *IEEE Trans. Signal Processing*, vol. 65, no. 8, pp. 2077–2092, 2016.

[40] N. Lim, F. d'Alché Buc, C. Auliac, and G. Michailidis, "Operator-valued kernel-based vector autoregressive models for network inference," *Machine Learning*, vol. 99, no. 3, pp. 489–513, 2015.

[41] Y. Shen, G. B. Giannakis, and B. Baingana, "Nonlinear structural vector autoregressive models with application to directed brain networks," *IEEE Trans. Signal Processing*, vol. 67, no. 20, pp. 5325–5339, 2019.

[42] E. Isufi, A. Loukas, N. Perraudin, and G. Leus, "Forecasting time series with VARMA recursions on graphs," *IEEE Trans. Sig. Process.*, vol. 67, no. 18, pp. 4870–4885, Sept. 2019.

[43] A. Tank, I. Covert, N. Foti, A. Shojaie, and E. Fox, "Neural Granger causality for nonlinear time series," *arXiv preprint arXiv:1802.05842*, 2018.

[44] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Int. Conf. Neural Info. Process. (NIPS)*, 2017, pp. 5998–6008.

[45] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan, "Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting," in *Proc. Int. Conf. Neural Info. Process. (NIPS)*, 2019, pp. 5244–5254.

[46] A. J. Smola and R. Kondor, "Kernels and regularization on graphs," in *Learning Theory and Kernel Machines*, 2003, pp. 144–158.

[47] H.-F. Yu, N. Rao, and I. S. Dhillon, "Temporal regularized matrix factorization for high-dimensional time series prediction," in *Proc. Int. Conf. Neural Info. Process. (NIPS)*, 2016.

[48] R. Sen, H.-F. Yu, and I. S. Dhillon, "Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting," in *Adv. Neural Information Processing Systems*, 2019, pp. 4838–4847.

[49] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *Proc. Int. Conf. Learning Representations (ICLR)*, May 2016.

[50] Y. Wang, A. Smola, D. C. Maddix, J. Gasthaus, D. Foster, and T. Januschowski, "Deep factors for forecasting," in *Proc. Int. Conf. on Machine Learning (ICML)*, 2019.

[51] L. Bai, L. Yao, C. Li, X. Wang, and C. Wang, "Adaptive graph convolutional recurrent network for traffic forecasting," in *Proc. Int. Conf. Neural Info. Process. (NIPS)*, vol. 33, 2020, pp. 17 804–17 815.

[52] Q. Zhang, J. Chang, G. Meng, S. Xiang, and C. Pan, "Spatio-temporal graph structure learning for traffic forecasting," in *Proc. AAAI Int. Conf. Artificial Intelligence*, 2020.

[53] B. Yu, M. Li, J. Zhang, and Z. Zhu, "3d graph convolutional networks with temporal graphs: A spatial information free framework for traffic forecasting," *arXiv preprint arXiv:1903.00919*, 2019.

[54] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, "T-GCN: A temporal graph convolutional network for traffic prediction," *IEEE Trans. Intelligent Transportation Systems*, vol. 21, no. 9, pp. 3848–3858, 2020.

[55] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Proc. Int. Conf. Neural Info. Process. (NIPS)*, vol. 28, 2015.

[56] Y. Huang, Y. Weng, S. Yu, and X. Chen, "Diffusion convolutional recurrent neural network with rank influence learning for traffic forecasting," in *Proc. IEEE Int. Conf. Big Data Science And Engineering*, Aug 2019, pp. 678–685.

[57] C. Chen, K. Li, S. G. Teo, X. Zou, K. Wang, J. Wang, and Z. Zeng, "Gated residual recurrent graph neural networks for traffic prediction," in *Proc. AAAI Conf. Artificial Intelligence*, 2019, pp. 485–492.

[58] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 2016, pp. 855–864.

[59] C. Park, C. Lee, H. Bahng, K. Kim, S. Jin, S. Ko, J. Choo *et al.*, "STGRAT: A spatio-temporal graph attention network for traffic forecasting," *arXiv preprint arXiv:1911.13181*, 2019.

[60] Z. Cui, K. Henrickson, R. Ke, and Y. Wang, "Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting," *IEEE Trans. Intelligent Transportation Systems*, vol. 21, no. 11, pp. 4883–4894, 2020.

[61] Z. Diao, G. Wang, D. Zhang, Y. Liu, K. Xie, and S. He, "Dynamic spatial-temporal graph convolutional neural networks for traffic forecasting," in *Proc. AAAI Conf. Artificial Intelligence*, 2019.

[62] B. Yu, H. Yin, and Z. Zhu, "ST-UNet: A spatio-temporal U-network for graph-structured time series modeling," *arXiv e-prints, arXiv:1903.05631*, Mar. 2019.

[63] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Proc. Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015, pp. 234–241.

[64] M. Xu, W. Dai, C. Liu, X. Gao, W. Lin, G.-J. Qi, and H. Xiong, "Spatial-temporal transformer networks for traffic flow forecasting," *arXiv preprint arXiv:2001.02908*, 2020.

[65] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. Int. Conf. on Machine Learning (ICML)*, 2013, pp. 1310–1318.

[66] C. Chen, K. Petty, A. Skabardonis, P. Varaiya, and Z. Jia, "Freeway performance measurement system: Mining loop detector data," *Transportation Research Record*, vol. 1748, no. 1, pp. 96–102, 2001.

[67] S. Makridakis and M. Hibon, "ARMA models and the Box–Jenkins methodology," *Journal of Forecasting*, vol. 16, no. 3, pp. 147–163.

[68] C.-H. Wu, J.-M. Ho, and D. T. Lee, "Travel-time prediction with support vector regression," *IEEE Trans. Intelligent Transportation Systems*, vol. 5, no. 4, pp. 276–281, 2004.

[69] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Int. Conf. Neural Info. Process. (NIPS)*, 2014, pp. 3104–3112.

[70] A. Alexandrov, K. Benidis, M. Bohlke-Schneider, V. Flunkert, J. Gasthaus, T. Januschowski, D. C. Maddix, S. Rangapuram, D. Salinas, J. Schulz, L. Stella, A. C. Türkmen, and Y. Wang, "GluonTS: Probabilistic and Neural Time Series Modeling in Python," *Journal of Machine Learning Research*, vol. 21, no. 116, pp. 1–6, 2020.