Spatial-Temporal Data Modeling with Graph Neural Networks

Thesis submitted in fulfilment of the requirements
for the degree of

Doctor of Philosophy

by

Zonghan Wu

under the supervision of

Guodong Long
Jing Jiang
Shirui Pan

to

University of Technology Sydney
Faculty of Engineering and Information Technology

August 2021

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, *Zonghan Wu* declare that this thesis, submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the *Australian Artificial Intelligence Institute*, *Faculty of Engineering and Information Technology* at the University of Technology Sydney, Australia. This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis. This document has not been submitted for qualifications at any other academic institution. This research is supported by the Australian Government Research Training Program.

SIGNATURE: Production Note: Signature removed prior to publication.

DATE: 19th August, 2021

PLACE: Sydney, Australia

Abstract

Spatial-temporal graph modeling is an important task to analyze the spatial relations and temporal trends of components in a system. It aims to model the dynamic node-level inputs by assuming inter-dependency between connected nodes. A basic assumption behind spatial-temporal graph modeling is that a node's future information is conditioned on its historical information as well as its neighbors' historical information. Therefore how to capture spatial and temporal dependencies simultaneously becomes a primary challenge. Current studies on spatial-temporal graph modeling face four major shortcomings: 1) Most graph neural networks only focus on the low frequency band of graph signals; 2) Current studies assume the graph structure of data reflects the genuine dependency relationships among nodes; 3) Existing studies on spatial-temporal graph neural networks are not applicable to pure multivariate time series data due to the absence of a predefined graph and lack of a general framework; 4) Existing approaches either model spatial-temporal dependencies locally or model spatial correlations and temporal correlations separately. The aim of this thesis is to study spatial-temporal data from the perspective of deep learning on graphs. I have studied the research objective in deep depth with four research questions: (1) How to coordinate the low, middle, and high frequency band of graph signals in graph convolution networks. (2) How to model spatial-temporal graph data effectively and efficiently; (3) How to handle spatial dependencies when a graph is totally missing, incomplete or inaccurate in spatial-temporal graph modeling; (4) In contrast to traditional spatial-temporal graph neural networks that handle spatial dependencies and temporal dependencies in separate, how to unify space and time as a whole in message passing. To address the aforementioned four research problems, I proposed four algorithms or models that can achieve satisfactory results. Specifically, I proposed an Automatic Graph Convolutional Network to learn graph frequency bands for graph convolution filters automatically; I introduced an efficient and effective framework that integrates diffusion graph convolution and dilated temporal convolution to capture spatial-temporal dependencies simultaneously. I developed a novel joint-learning algorithm that can capture spatial-temporal dependencies and learn latent graph structures at the same time; I designed a unified graph neural network that captures the inner spatial-temporal dependencies without compromising space-time integrity. To validate the proposed methods, I have conducted experiments on real-world datasets with a range of tasks including node classification, graph classification, and spatial-temporal graph forecasting. Experimental results demonstrate the effectiveness of the proposed methods.

Acknowledgments

I would like to express my deepest appreciation to my supervisors Dr. Shirui Pan, Dr. Guodong Long, Dr. Jing Jiang, and Prof. Chengqi Zhang for being my knowledgeable academic teachers and firm life supporters during my Ph.D study. Their unparalleled knowledge, practical suggestions, and constructive criticism built a solid foundation for my academic career.

A very special gratitude goes to Dr Huan Huo, who highly stimulated my motivation to go through higher research studies. Without Dr. Huo's encouragement and support, I could not stand where I am today.

I am also grateful to the University of Technology Sydney, who has provided me full research scholarships. Thanks to all staff members working behind the graduate research school. Their smooth and hard work provides me a comfortable working environment.

Finally, I would like to extend my deep gratitude to my family: my father Mr. Runwen Huang who always protect my mother and me when we are in bad situations, my mother Mrs. Jiangjing Wu who provides me endless loves in my life, and my wife Mrs. Yelu Yu who gives me full confidence to take difficult challenges.

List of Publications

Journal Papers

J-1. Wu Z., Pan S., Chen F., Long G., Zhang C. and Philip S.Y., 2020. A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems. [CORE A*, Google Citations: 1,969]

J-2. Wu Z., Pan S., Long G., Jiang J. and Zhang C., 2021. Beyond low-pass filtering: graph convolutional networks with automatic filtering. Submitted to IEEE Transactions on Knowledge and Data Engineering. [CORE A*, Under Review]

J-3. Wu Z., Zheng D., Pan S., Gan G., Long G. and Karypis G., 2021. TraverseNet: Unifying Space and Time in Message Passing. Submitted to IEEE Transactions on Neural Networks and Learning Systems. [CORE A*, Under Review]

Conference Papers

C-1. Wu Z., Pan S., Long G., Jiang J. and Zhang C., 2019. Graph wavenet for deep spatial-temporal graph modeling. In Proceedings of the 28th International Joint Conference on Artificial Intelligence (pp. 1907-1913). [CORE A*, Google Citations: 190]

C-2. Wu Z., Pan S., Long, G., Jiang J., Chang X. and Zhang C., 2020, August. Connecting the dots: Multivariate time series forecasting with graph neural networks. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 753-763). [CORE A*, Google Citations: 53]

Table of Contents

List of Figures

Figure **Page**

List of Tables

CHAPTER 1

INTRODUCTION

1.1 Background

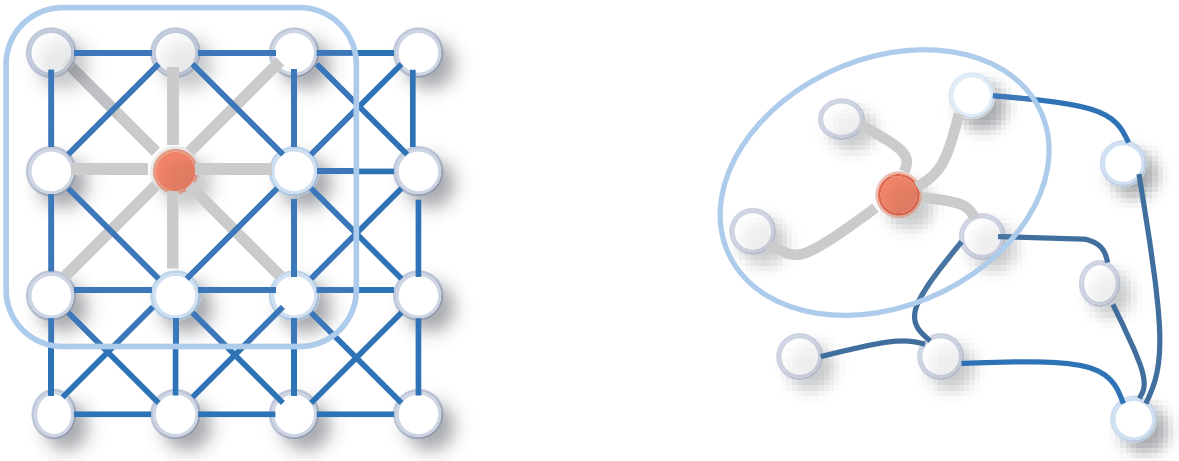
The recent success of neural networks has boosted research on pattern recognition and data mining. Many machine learning tasks such as object detection [4, 5], machine translation [6, 7], and speech recognition [8], which once heavily relied on handcrafted feature engineering to extract informative feature sets, has recently been revolutionized by various end-to-end deep learning paradigms, e.g., convolutional neural networks (CNNs) [9], recurrent neural networks (RNNs) [10], and autoencoders [11]. The success of deep learning in many domains is partially attributed to the rapidly developing computational resources (e.g., GPU), the availability of big training data, and the effectiveness of deep learning to extract latent representations from Euclidean data (e.g., images, text, and videos). Taking image data as an example, we can represent an image as a regular grid in the Euclidean space. A convolutional neural network (CNN) is able to exploit the shift-invariance, local connectivity, and compositionality of image data [12]. As a result, CNNs can extract local meaningful features that are shared with the entire data sets for various image analyses.

While deep learning effectively captures hidden patterns of Euclidean data, there is an increasing number of applications where data are represented in the form of graphs. For examples, in e-commence, a graph-based learning system can exploit the interactions between users and products to make highly accurate recommendations. In chemistry, molecules are modeled as graphs, and their bioactivity needs to be identified for drug

discovery. In a citation network, papers are linked to each other via citationships and they need to be categorized into different groups. The complexity of graph data has imposed significant challenges on existing machine learning algorithms. As graphs can be irregular, a graph may have a variable size of unordered nodes, and nodes from a graph may have a different number of neighbors, resulting in some important operations (e.g., convolutions) being easy to compute in the image domain, but difficult to apply to the graph domain. Furthermore, a core assumption of existing machine learning algorithms is that instances are independent of each other. This assumption no longer holds for graph data because each instance (node) is related to others by links of various types, such as citations, friendships, and interactions. Recently, there is increasing interest in extending deep learning approaches for graph data. Motivated by CNNs, RNNs, and autoencoders from deep learning, new generalizations and definitions of important operations have been rapidly developed over the past few years to handle the complexity of graph data. For example, a graph convolution can be generalized from a 2D convolution. As illustrated in Figure 1.1, an image can be considered as a special case of graphs where pixels are connected by adjacent pixels. Similar to 2D convolution, one may perform graph convolutions by taking the weighted average of a node's neighborhood information.

With the advance of graph neural networks, spatial-temporal graph modeling has received increasing attention. It aims to model the dynamic node-level inputs by assuming inter-dependency between connected nodes. Spatial-temporal graph modeling has wide applications in solving complex system problems such as traffic speed forecasting [13], taxi demand prediction [14], human action recognition [15], pedestrian trajectory prediction [16], person re-identification [17], and driver maneuver anticipation [18]. For a concrete example, in traffic speed forecasting, speed sensors on roads of a city form a graph where the edge weights are judged by two nodes' Euclidean distance. As the traffic congestion on one road could cause lower traffic speeds on its incoming roads, it is natural to consider the underlying graph structure of the traffic system as the prior knowledge of inter-dependency relationships among nodes when modeling time series data of the traffic speed on each road.

A basic assumption behind spatial-temporal graph modeling is that a node's future information is conditioned on its historical information as well as its neighbors' historical information. Therefore how to capture spatial and temporal dependencies simultaneously becomes a primary challenge. Recent studies on spatial-temporal graph modeling mainly follow two directions. They either integrate graph convolution networks (GCN) into



(a) 2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes the weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size.

(b) Graph Convolution. To get a hidden representation of the red node, one simple solution of the graph convolutional operation is to take the average value of the node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size.

Figure 1.1: 2D Convolution vs. Graph Convolution.

recurrent neural networks (RNN) [13, 19] or into convolution neural networks (CNN) [15, 20].

While having shown the effectiveness of introducing the graph structure of data into a model, these approaches face four major shortcomings.

1. Most graph neural networks only focus on the low frequency band of graph signals. However, the middle and high frequency band of graph signals should not be ignored because they may contain useful information as well. Besides, the bandwidth of current GNNs is fixed. Parameters of a graph convolutional filter only transform graph inputs without changing the curvature of a graph convolutional filter function. Therefore, the cut-off frequency or the bandwidth of a graph convolutional filter remains unchanged throughout learning. In reality, we are uncertain about whether we should retain or cut off the frequency at a certain point unless we have expert domain knowledge.

2. Current studies assume the graph structure of data reflects the genuine dependency relationships among nodes. However, there are circumstances when a connection does not entail the inter-dependency relationship between two nodes and when the inter-dependency relationship between two nodes exists but a connection is missing. To give

each circumstance an example, let us consider a recommendation system. In the first case, two users are connected, but they may have distinct preferences over products. In the second case, two users may share a similar preference, but they are not linked together. To a broader context, this problem is related to weakly-supervised learning where given labels are not always ground truth.

3. Existing studies on spatial-temporal graph neural networks are not applicable to pure multivariate time series data due to the absence of a predefined graph and lack of a general framework. Spatial-temporal graph neural networks take multivariate time series and an external graph structure as inputs, and they aim to predict future values or labels of multivariate time series. Spatial-temporal graph neural networks have achieved significant improvements compared to methods that do not utilize structural information. However, existing GNN approaches rely heavily on a pre-defined graph structure in order to perform time series forecasting. In most cases, multivariate time series does not have an explicit graph structure. The relationships among variables have to be discovered from data rather than being provided as ground truth knowledge.

4. Existing approaches either model spatial-temporal dependencies locally or model spatial correlations and temporal correlations separately. They prevent a node from being directly aware of its neighborhood long-range historical information. In fact, a node's current state may depend on its neighbors' previous states within a certain period of time. The rise of a node's curve may exert influence on its neighbors several time steps later because of physical distance. For example, traffic congestion of a road will cause another congestion of its nearby roads 15 minutes later. It suggests that treating spatial correlations and temporal correlations locally or separately is inappropriate.

1.2 Research Objectives

The main research objective of this research is to study spatial-temporal data from the perspective of deep learning on graphs: (1) How to coordinate the low, middle, and high frequency band of graph signals in graph convolution networks. (2) How to model spatial-temporal graph data effectively and efficiently; (3) How to handle spatial dependencies when a graph is totally missing, incomplete or inaccurate in spatial-temporal graph modeling; (4) In contrast to traditional spatial-temporal graph neural networks that handle spatial dependencies and temporal dependencies in separate, how to unify space and time as a whole in message passing.

To achieve the research objective, the following lists the conducted studies and

planned studies.

- conduct studies on the automatic learning of graph frequency bands for graph convolution filters. I propose an Automatic Graph Convolutional Network (AutoGCN) with three novel graph convolutional filters, a low-pass linear filter, a high-pass linear filter, and a middle-pass quadratic filter. While capturing the whole spectrum of graph signals, AutoGCN ends up with a spatial form without performing eigendecomposition. I enable the proposed graph convolution filters to control their bandwidth and magnitude automatically by updating the curvature and scope of filter functions during training. I empirically show that all three graph filters contribute to model performance. Experimental results show that AutoGCN achieves significant improvement over baseline methods that only function as low-pass filters on medium-scale datasets for both node classification and graph prediction tasks.

- conduct studies on an effective and efficient framework to capture spatial-temporal dependencies simultaneously. The core idea is to assemble the proposed graph convolution with dilated casual convolution in a way that each graph convolution layer tackles spatial dependencies of nodes' information extracted by dilated casual convolution layers at different granular levels. I evaluate my proposed model on traffic datasets and achieve state-of-the-art results with low computation costs.

- conduct studies on learning graph structure when the spatial dependency is unknown for multivariate time series data. I propose a novel graph learning module to learn hidden spatial dependencies among variables. My method opens a new door for GNN models to handle data without explicit graph structure. I present a joint framework for modeling multivariate time series data and learning graph structure. My framework is more generic than any existing spatial-temporal graph neural network as it can handle multivariate time series with or without a predefined graph structure. Experimental results show that my method outperforms the state-of-the-art methods on 3 of 4 benchmark datasets and achieves on-par performance with other GNNs on two traffic datasets which provide extra structural information.

- conduct studies on a simple and powerful framework named TraverseNet that captures the inner spatial-temporal dependencies without compromising space-time integrity. I propose a message traverse layer, effectively unifying space and

time in message passing by traversing information of a node's neighbors' past to the node's present. I construct TraverseNet with message traverse layers and validate the significance of the message traverse mechanism with experimental studies.

1.3 Content and Organization

This thesis is organized as follows:

- Chapter 2: This chapter presents a survey of deep learning on graphs.
- Chapter 3: This chapter introduces a novel graph convolutional network that can adapt graph frequency automatically.
- Chapter 4: This chapter proposes a convolution-based spatial-temporal graph neural network that is both efficient and effective.
- Chapter 5: This chapter addresses the problem that the graph is unknown in spatial-temporal modeling.
- Chapter 6: This chapter introduces a spatial-temporal graph neural network that unifies space and time.
- Chapter 7: This chapter provides a brief summary of the thesis contents as well as its contributions.
- Chapter 8: This chapter points out several future directions of research on spatial-temporal graph modeling.

CHAPTER 2

LITERATURE REVIEW

There are a limited number of existing reviews on the topic of graph neural networks (GNNs). Using the term *geometric deep learning*, Bronstein et al. [12] give an overview of deep learning methods in the non-Euclidean domain, including graphs and manifolds. Although it is the first review on GNNs, this survey mainly reviews convolutional GNNs. Hamilton et al. [21] cover a limited number of GNNs with a focus on addressing the problem of network embedding. Battaglia et al. [22] position *graph networks* as the building blocks for learning from relational data, reviewing part of GNNs under a unified framework. Lee et al. [23] conduct a partial survey of GNNs which apply different attention mechanisms. In summary, existing surveys only include some of the GNNs and examine a limited number of works, thereby missing more recent development of GNNs. This chapter provides a comprehensive overview of GNNs, for both interested researchers who want to enter this rapidly developing field and experts who would like to compare GNN models. To cover a broader range of methods, this chapter considers GNNs as all deep learning approaches for graph data.

This chapter makes notable contributions summarized as follows:

- **New taxonomy** I propose a new taxonomy of graph neural networks. Graph neural networks are categorized into four groups: recurrent graph neural networks, convolutional graph neural networks, graph autoencoders, and spatial-temporal graph neural networks.

- **Comprehensive review** I provide the most comprehensive overview of modern

deep learning techniques for graph data. For each type of graph neural network, I provide detailed descriptions of representative models, make the necessary comparison, and summarise the corresponding algorithms.

- **Abundant resources** I collect abundant resources on graph neural networks, including state-of-the-art models, benchmark data sets, open-source codes, and practical applications. This chapter can be used as a hands-on guide for understanding, using, and developing different deep learning approaches for various real-life applications.

- **Future directions** I discuss theoretical aspects of graph neural networks, analyze the limitations of existing methods, and suggest four possible future research directions in terms of model depth, scalability trade-off, heterogeneity, and dynamicity.

2.1 Background

A brief history of graph neural networks (GNNs) Sperduti et al. (1997) [24] first applied neural networks to directed acyclic graphs, which motivated early studies on GNNs. The notion of graph neural networks was initially outlined in Gori et al. (2005) [25] and further elaborated in Scarselli et al. (2009) [26], and Gallicchio et al. (2010) [27]. These early studies fall into the category of recurrent graph neural networks (RecGNNs). They learn a target node's representation by propagating neighbor information in an iterative manner until a stable fixed point is reached. This process is computationally expensive, and recently there have been increasing efforts to overcome these challenges [28, 29].

Encouraged by the success of CNNs in the computer vision domain, a large number of methods that re-define the notion of *convolution* for graph data are developed in parallel. These approaches are under the umbrella of convolutional graph neural networks (ConvGNNs). ConvGNNs are divided into two main streams, the spectral-based approaches and the spatial-based approaches. The first prominent research on spectral-based ConvGNNs was presented by Bruna et al. (2013) [30], which developed a graph convolution based on the spectral graph theory. Since this time, there have been increasing improvements, extensions, and approximations on spectral-based ConvGNNs [1, 31–33]. The research of spatial-based ConvGNNs started much earlier than spectral-based ConvGNNs. In 2009, Micheli et al. [34] first addressed graph mutual dependency by architecturally composite non-recursive layers while inheriting ideas of

message passing from RecGNNs. However, the importance of this work was overlooked. Until recently, many spatial-based ConvGNNs (e.g., [35–37]) emerged. The timeline of representative RecGNNs and ConvGNNs is shown in the first column of Table 2.2. Apart from RecGNNs and ConvGNNs, many alternative GNNs have been developed in the past few years, including graph autoencoders (GAEs) and spatial-temporal graph neural networks (STGNNs). These learning frameworks can be built on RecGNNs, ConvGNNs, or other neural architectures for graph modeling. Details on the categorization of these methods are given in Section 2.3.

Graph neural networks vs. network embedding The research on GNNs is closely related to graph embedding or network embedding, another topic which attracts increasing attention from both the data mining and machine learning communities [21, 38–42]. Network embedding aims at representing network nodes as low-dimensional vector representations, preserving both network topology structure and node content information, so that any subsequent graph analytics task such as classification, clustering, and recommendation can be easily performed using simple off-the-shelf machine learning algorithms (e.g., support vector machines for classification). Meanwhile, GNNs are deep learning models aiming at addressing graph-related tasks in an end-to-end manner. Many GNNs explicitly extract high-level representations. The main distinction between GNNs and network embedding is that GNNs are a group of neural network models which are designed for various tasks while network embedding covers various kinds of methods targeting the same task. Therefore, GNNs can address the network embedding problem through a graph autoencoder framework. On the other hand, network embedding contains other non-deep learning methods such as matrix factorization [43, 44] and random walks [45].

Graph neural networks vs. graph kernel methods Graph kernels are historically dominant techniques to solve the problem of graph classification [46–48]. These methods employ a kernel function to measure the similarity between pairs of graphs so that kernel-based algorithms like support vector machines can be used for supervised learning on graphs. Similar to GNNs, graph kernels can embed graphs or nodes into vector spaces by a mapping function. The difference is that this mapping function is deterministic rather than learnable. Due to a pair-wise similarity calculation, graph kernel methods suffer significantly from computational bottlenecks. GNNs, on one hand, directly perform graph classification based on the extracted graph representations and therefore are much more efficient than graph kernel methods. For a further review of graph kernel methods, I refer the readers to [49].

Table 2.1: Commonly used notations.

Notations	Descriptions
$\|\cdot\|$	The length of a set.
$\odot$	Element-wise product.
G	A graph.
V	The set of nodes in a graph.
v	A node $v \in V$.
E	The set of edges in a graph.
e_{ij}	An edge $e_{ij} \in E$.
$N(v)$	The neighbors of a node v.
$\mathbf{A}$	The graph adjacency matrix.
$\mathbf{A}^T$	The transpose of the matrix $\mathbf{A}$.
$\mathbf{A}^n, n \in Z$	The n^{th} power of $\mathbf{A}$.
$[\mathbf{A}, \mathbf{B}]$	The concatenation of $\mathbf{A}$ and $\mathbf{B}$.
$\mathbf{D}$	The degree matrix of $\mathbf{A}$. $\mathbf{D}_{ii} = \sum_{j=1}^{n} \mathbf{A}_{ij}$.
n	The number of nodes, $n = \|V\|$.
m	The number of edges, $m = \|E\|$.
d	The dimension of a node feature vector.
b	The dimension of a hidden node feature vector.
c	The dimension of an edge feature vector.
$\mathbf{X} \in \mathbf{R}^{n \times d}$	The feature matrix of a graph.
$\mathbf{x} \in \mathbf{R}^n$	The feature vector of a graph in the case of $d = 1$.
$\mathbf{x}_v \in \mathbf{R}^d$	The feature vector of the node v.
$\mathbf{X}^e \in \mathbf{R}^{m \times c}$	The edge feature matrix of a graph.
$\mathbf{x}^e_{(v,u)} \in \mathbf{R}^c$	The edge feature vector of the edge (v, u).
$\mathbf{X}^{(t)} \in \mathbf{R}^{n \times d}$	The node feature matrix of a graph at the time step t.
$\mathbf{H} \in \mathbf{R}^{n \times b}$	The node hidden feature matrix.
$\mathbf{h}_v \in \mathbf{R}^b$	The hidden feature vector of node v.
k	The layer index
t	The time step/iteration index
$\sigma(\cdot)$	The sigmoid activation function.
$\sigma_h(\cdot)$	The tangent hyperbolic activation function.
$\mathbf{W}, \Theta, w, \theta$	Learnable model parameters.

2.2 Definition

Throughout this chapter, I use bold uppercase characters to denote matrices and bold lowercase characters denote vectors. Unless particularly specified, the notations used in this chapter are illustrated in Table 2.1. Now I define the minimal set of definitions required to understand this chapter.

Definition 1 (Graph)**.** *A graph is represented as $G = (V, E)$ where V is the set of n vertices or nodes (I will use nodes throughout this work), and E is the set of m edges. Let $v_i \in V$ to denote a node and $e_{ij} = (v_i, v_j) \in E$ to denote an edge pointing from v_j to v_i. The neighborhood of a node v is defined as $N(v) = \{u \in V | (v, u) \in E\}$. The adjacency matrix $\mathbf{A}$ is a $n \times n$ matrix with $A_{ij} = 1$ if $e_{ij} \in E$ and $A_{ij} = 0$ if $e_{ij} \notin E$. A graph may have node attributes $\mathbf{X}$ [1], where $\mathbf{X} \in \mathbf{R}^{n \times d}$ is a node feature matrix with $\mathbf{x}_v \in \mathbf{R}^d$ representing the feature vector of a node v. Meanwhile, a graph may have edge attributes $\mathbf{X}^e$, where $\mathbf{X}^e \in \mathbf{R}^{m \times c}$ is an edge feature matrix with $\mathbf{x}^e_{v,u} \in \mathbf{R}^c$ representing the feature vector of an edge (v, u).*

Definition 2 (Directed Graph)**.** *A directed graph is a graph with all edges directed from one node to another. An undirected graph is considered as a special case of directed graphs where there is a pair of edges with inverse directions if two nodes are connected. A graph is undirected if and only if the adjacency matrix is symmetric.*

Definition 3 (Spatial-Temporal Graph)**.** *A spatial-temporal graph is an attributed graph where the node attributes change dynamically over time. The spatial-temporal graph is defined as $G^{(t)} = (\mathbf{V}, \mathbf{E}, \mathbf{X}^{(t)})$ with $\mathbf{X}^{(t)} \in \mathbf{R}^{n \times d}$.*

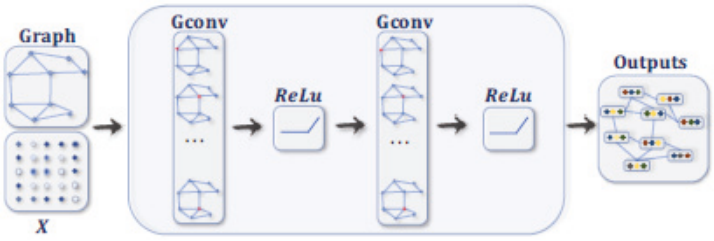
2.3 Categorization and Frameworks

In this section, I present my taxonomy of graph neural networks (GNNs), as shown in Table 2.2. I categorize graph neural networks (GNNs) into recurrent graph neural networks (RecGNNs), convolutional graph neural networks (ConvGNNs), graph autoencoders (GAEs), and spatial-temporal graph neural networks (STGNNs). Figure 2.1 gives examples of various model architectures. In the following, I give a brief introduction of each category.

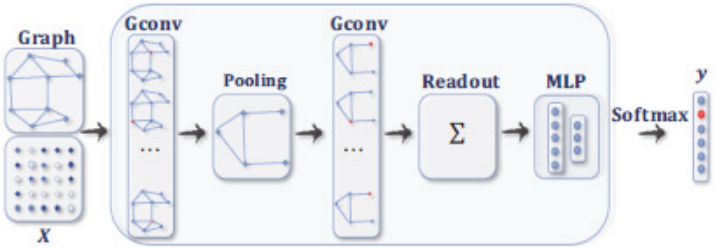
2.3.1 Taxonomy of Graph Neural Networks (GNNs)

Recurrent graph neural networks (RecGNNs) mostly are pioneer works of graph neural networks. RecGNNs aim to learn node representations with recurrent neural architectures. They assume a node in a graph constantly exchanges information/message with its neighbors until a stable equilibrium is reached. RecGNNs are conceptually

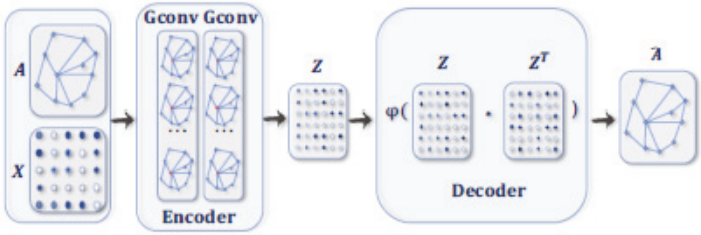
[1] Such graph is referred to an *attributed graph* in literature.



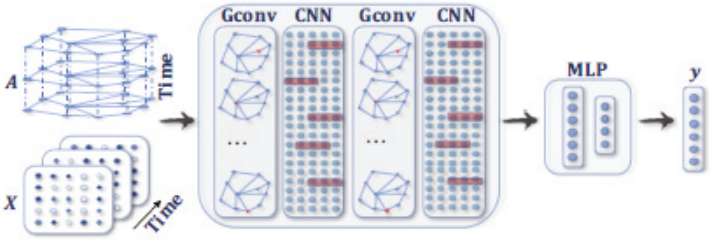
(a) A ConvGNN with multiple graph convolutional layers. A graph convolutional layer encapsulates each node's hidden representation by aggregating feature information from its neighbors. By stacking multiple layers, the final hidden representation of each node receives messages from a further neighborhood.



(b) A ConvGNN with pooling and readout layers for graph classification [32]. A graph convolutional layer is followed by a pooling layer to coarsen a graph into sub-graphs so that node representations on coarsened graphs represent higher graph-level representations. A readout layer summarizes the final graph representation by taking the sum/mean of hidden representations of sub-graphs.



(c) A GAE for network embedding [69]. The encoder uses graph convolutional layers to get a network embedding for each node. The decoder reconstructs the graph adjacency matrix given network embeddings. The network is trained by minimizing the discrepancy between the real adjacency matrix and the reconstructed adjacency matrix.



(d) A STGNN for spatial-temporal graph forecasting [79]. The graph convolutional layer operates on A and $X^{(t)}$ to capture the spatial dependency, while the 1D-CNN layer slides over X along the time axis to capture the temporal dependency. The output layer is a linear transformation, generating a prediction for each node, such as its future value at the next time step.

Figure 2.1: Different graph neural network models built with graph convolutional layers.

Table 2.2: Taxonomy and representative publications of Graph Neural Networks (GNNs)

Category		Publications
Recurrent Graph Neural Networks (RecGNNs)		[26–29]
Convolutional Graph Neural Networks (ConvGNNs)	Spectral methods	[1, 30–33, 50, 51]
	Spatial methods	[2, 34–37, 52, 53] [54–60] [3, 61–66]
Graph Autoencoders (GAEs)	Network Embedding	[67–72]
	Graph Generation	[73–78]
Spatial-temporal Graph Neural Networks (STGNNs)		[13, 15, 18, 19, 79–81]

important and inspired later research on convolutional graph neural networks. In particular, the idea of message passing is inherited by spatial-based convolutional graph neural networks.

Convolutional graph neural networks (ConvGNNs) generalize the operation of *convolution* from grid data to graph data. The main idea is to generate a node v's representation by aggregating its own features $\mathbf{x}_v$ and neighbors' features $\mathbf{x}_u$, where $u \in N(v)$. Different from RecGNNs, ConvGNNs stack multiple graph convolutional layers to extract high-level node representations. ConvGNNs play a central role in building up many other complex GNN models. Figure 2.1a shows a ConvGNN for node classification. Figure 2.1b demonstrates a ConvGNN for graph classification.

Graph autoencoders (GAEs) are unsupervised learning frameworks which encode nodes/graphs into a latent vector space and reconstruct graph data from the encoded information. GAEs are used to learn network embeddings and graph generative distributions. For network embedding, GAEs learn latent node representations through reconstructing graph structural information such as the graph adjacency matrix. For graph generation, some methods generate nodes and edges of a graph step by step while other methods output a graph all at once. Figure 2.1c presents a GAE for network embedding.

Spatial-temporal graph neural networks (STGNNs) aim to learn hidden patterns from spatial-temporal graphs, which become increasingly important in a variety of applications such as traffic speed forecasting [13], driver maneuver anticipation [18], and human action recognition [15]. The key idea of STGNNs is to consider spatial dependency and temporal dependency at the same time. Many current approaches integrate graph convolutions to capture spatial dependency with RNNs or CNNs to model the temporal dependency. Figure 2.1d illustrates a STGNN for spatial-temporal graph forecasting.

2.3.2 Frameworks

With the graph structure and node content information as inputs, the outputs of GNNs can focus on different graph analytics tasks with one of the following mechanisms:

- **Node-level** outputs relate to node regression and node classification tasks. RecGNNs and ConvGNNs can extract high-level node representations by information propagation/graph convolution. With a multi-perceptron or a softmax layer as the output layer, GNNs are able to perform node-level tasks in an end-to-end manner.
- **Edge-level** outputs relate to the edge classification and link prediction tasks. With two nodes' hidden representations from GNNs as inputs, a similarity function or a neural network can be utilized to predict the label/connection strength of an edge.
- **Graph-level** outputs relate to the graph classification task. To obtain a compact representation on the graph level, GNNs are often combined with pooling and readout operations. Detailed information about pooling and readouts will be reviewed in Section 2.5.3.

Training Frameworks. Many GNNs (e.g., ConvGNNs) can be trained in a (semi-) supervised or purely unsupervised way within an end-to-end learning framework, depending on the learning tasks and label information available at hand.

- **Semi-supervised learning for node-level classification.** Given a single network with partial nodes being labeled and others remaining unlabeled, ConvGNNs can learn a robust model that effectively identifies the class labels for the unlabeled nodes [1]. To this end, an end-to-end framework can be built by stacking a couple of graph convolutional layers followed by a softmax layer for multi-class classification.
- **Supervised learning for graph-level classification.** Graph-level classification aims to predict the class label(s) for an entire graph [61, 63, 82, 83]. The end-to-end learning for this task can be realized with a combination of graph convolutional layers, graph pooling layers, and/or readout layers. While graph convolutional layers are responsible for exacting high-level node representations, graph pooling layers play the role of down-sampling, which coarsens each graph into a substructure each time. A readout layer collapses node representations of each graph into a graph representation. By applying a multi-layer perceptron and a softmax layer to graph representations, we can build an end-to-end framework for graph classification. An example is given in Fig 2.1b.

- **Unsupervised learning for graph embedding.** When no class labels are available in graphs, we can learn the graph embedding in a purely unsupervised way in an end-to-end framework. These algorithms exploit edge-level information in two ways. One simple way is to adopt an autoencoder framework where the encoder employs graph convolutional layers to embed the graph into the latent representation upon which a decoder is used to reconstruct the graph structure [69, 70]. Another popular way is to utilize the negative sampling approach which samples a portion of node pairs as negative pairs while existing node pairs with links in the graphs are positive pairs. Then a logistic regression layer is applied to distinguish between positive and negative pairs [52].

In Table 2.3, I summarize the main characteristics of representative RecGNNs and ConvGNNs. Input sources, pooling layers, readout layers, and time complexity are compared among various models. In more detail, I only compare the time complexity of the message passing/graph convolution operation in each model. As methods in [30] and [31] require eigenvalue decomposition, the time complexity is $O(n^3)$. The time complexity of [55] is also $O(n^3)$ due to the node pair-wise shortest path computation. Other methods incur equivalent time complexity, which is $O(m)$ if the graph adjacency matrix is sparse and is $O(n^2)$ otherwise. This is because in these methods the computation of each node v_i's representation involves its d_i neighbors, and the sum of d_i over all nodes exactly equals the number of edges. The time complexity of several methods is missing in Table 2.3. These methods either lack a time complexity analysis in their papers or report the time complexity of their overall models or algorithms.

2.4 Recurrent Graph Neural Networks

Recurrent graph neural networks (RecGNNs) are mostly pioneer works of GNNs. They apply the same set of parameters recurrently over nodes in a graph to extract high-level node representations. Constrained by computational power, earlier research mainly focused on directed acyclic graphs [24, 84].

Graph Neural Network (GNN*[2]) proposed by Scarselli et al. extends prior recurrent models to handle general types of graphs, e.g., acyclic, cyclic, directed, and undirected graphs [26]. Based on an information diffusion mechanism, GNN* updates nodes' states

[2]As GNN is used to represent broad graph neural networks in this chapter, I name this particular method GNN* to avoid ambiguity.

Table 2.3: Summary of RecGNNs and ConvGNNs. Missing values ("-") in pooling and readout layers indicate that the method only experiments on node-level/edge-level tasks.

Approach	Category	Inputs	Pooling	Readout	Time Complexity
GNN* (2009) [26]	RecGNN	A, X, X^e	-	a dummy super node	$O(m)$
GraphESN (2010) [27]	RecGNN	A, X	-	mean	$O(m)$
GGNN (2015) [28]	RecGNN	A, X	-	attention sum	$O(m)$
SSE (2018) [29]	RecGNN	A, X	-	-	-
Spectral CNN (2014) [30]	Spectral-based ConvGNN	A, X	spectral clustering+max pooling	max	$O(n^3)$
Henaff et al. (2015) [31]	Spectral-based ConvGNN	A, X	spectral clustering+max pooling		$O(n^3)$
ChebNet (2016) [32]	Spectral-based ConvGNN	A, X	efficient pooling	sum	$O(m)$
GCN (2017) [1]	Spectral-based ConvGNN	A, X	-	-	$O(m)$
CayleyNet (2017) [33]	Spectral-based ConvGNN	A, X	mean/graclus pooling	-	$O(m)$
AGCN (2018) [50]	Spectral-based ConvGNN	A, X	max pooling	sum	$O(n^2)$
DualGCN (2018) [51]	Spectral-based ConvGNN	A, X	-	-	$O(m)$
NN4G (2009) [34]	Spatial-based ConvGNN	A, X	-	sum/mean	$O(m)$
DCNN (2016) [35]	Spatial-based ConvGNN	A, X	-	mean	$O(n^2)$
PATCHY-SAN (2016) [36]	Spatial-based ConvGNN	A, X, X^e	-	sum	-
MPNN (2017) [37]	Spatial-based ConvGNN	A, X, X^e	-	attention sum/set2set	$O(m)$
GraphSage (2017) [52]	Spatial-based ConvGNN	A, X	-	-	-
GAT (2017) [2]	Spatial-based ConvGNN	A, X	-	-	$O(m)$
MoNet (2017) [53]	Spatial-based ConvGNN	A, X	-	-	$O(m)$
LGCN (2018) [54]	Spatial-based ConvGNN	A, X	-	-	-
PGC-DGCNN (2018) [55]	Spatial-based ConvGNN	A, X	sort pooling	attention sum	$O(n^3)$
CGMM (2018) [56]	Spatial-based ConvGNN	A, X, X^e	-	sum	-
GAAN (2018) [57]	Spatial-based ConvGNN	A, X	-	-	$O(m)$
FastGCN (2018) [58]	Spatial-based ConvGNN	A, X	-	-	-
StoGCN (2018) [59]	Spatial-based ConvGNN	A, X	-	-	-
Huang et al. (2018) [60]	Spatial-based ConvGNN	A, X	-	-	-
DGCNN (2018) [61]	Spatial-based ConvGNN	A, X	sort pooling	-	$O(m)$
DiffPool (2018) [63]	Spatial-based ConvGNN	A, X	differential pooling	mean	$O(n^2)$
GeniePath (2019) [64]	Spatial-based ConvGNN	A, X	-	-	$O(m)$
DGI (2019) [65]	Spatial-based ConvGNN	A, X	-	-	$O(m)$
GIN (2019) [66]	Spatial-based ConvGNN	A, X	-	sum	$O(m)$
ClusterGCN (2019) [3]	Spatial-based ConvGNN	A, X	-	-	-

by exchanging neighborhood information recurrently until a stable equilibrium is reached. A node's hidden state is recurrently updated by

$$\mathbf{h}_v^{(t)} = \sum_{u \in N(v)} f(\mathbf{x}_v, \mathbf{x^e}_{(v,u)}, \mathbf{x}_u, \mathbf{h}_u^{(t-1)}), \tag{2.1}$$

where $f(\cdot)$ is a parametric function, and $\mathbf{h}_v^{(0)}$ is initialized randomly. The sum operation enables GNN* to be applicable to all nodes, even if the number of neighbors differs and no neighborhood ordering is known. To ensure convergence, the recurrent function $f(\cdot)$ must be a contraction mapping, which shrinks the distance between two points after

projecting them into a latent space. In the case of $f(\cdot)$ being a neural network, a penalty term has to be imposed on the Jacobian matrix of parameters. When a convergence criterion is satisfied, the last step node hidden states are forwarded to a readout layer. GNN* alternates the stage of node state propagation and the stage of parameter gradient computation to minimize a training objective. This strategy enables GNN* to handle cyclic graphs. In follow-up works, Graph Echo State Network (GraphESN) [27] extends echo state networks to improve the training efficiency of GNN*. GraphESN consists of an encoder and an output layer. The encoder is randomly initialized and requires no training. It implements a contractive state transition function to recurrently update node states until the global graph state reaches convergence. Afterward, the output layer is trained by taking the fixed node states as inputs.

Gated Graph Neural Network (GGNN) [28] employs a gated recurrent unit (GRU) [85] as a recurrent function, reducing the recurrence to a fixed number of steps. The advantage is that it no longer needs to constrain parameters to ensure convergence. A node hidden state is updated by its previous hidden states and its neighboring hidden states, defined as

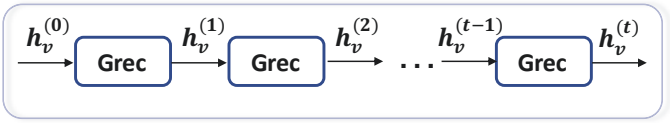
$$\mathbf{h}_v^{(t)} = GRU(\mathbf{h}_v^{(t-1)}, \sum_{u \in N(v)} \mathbf{W}\mathbf{h}_u^{(t-1)}), \tag{2.2}$$

where $\mathbf{h}_v^{(0)} = \mathbf{x}_v$. Different from GNN* and GraphESN, GGNN uses the back-propagation through time (BPTT) algorithm to learn the model parameters. This can be problematic for large graphs, as GGNN needs to run the recurrent function multiple times over all nodes, requiring the intermediate states of all nodes to be stored in memory.

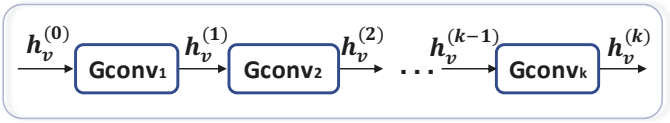
Stochastic Steady-state Embedding (SSE) proposes a learning algorithm that is more scalable to large graphs [29]. SSE updates node hidden states recurrently in a stochastic and asynchronous fashion. It alternatively samples a batch of nodes for state update and a batch of nodes for gradient computation. To maintain stability, the recurrent function of SSE is defined as a weighted average of the historical states and new states, which takes the form

$$\mathbf{h}_v^{(t)} = (1-\alpha)\mathbf{h}_v^{(t-1)} + \alpha \mathbf{W_1} \sigma(\mathbf{W_2}[\mathbf{x}_v, \sum_{u \in N(v)} [\mathbf{h}_u^{(t-1)}, \mathbf{x}_u]]), \tag{2.3}$$

where α is a hyper-parameter, and $\mathbf{h}_v^{(0)}$ is initialized randomly. While conceptually important, SSE does not theoretically prove that the node states will gradually converge to fixed points by applying Equation 2.3 repeatedly.



(a) Recurrent Graph Neural Networks (RecGNNs). RecGNNs use the same graph recurrent layer (Grec) in updating node representations.



(b) Convolutional Graph Neural Networks (ConvGNNs). ConvGNNs use a different graph convolutional layer (Gconv) in updating node representations.

Figure 2.2: RecGNNs v.s. ConvGNNs

2.5 Convolutional Graph Neural Networks

Convolutional graph neural networks (ConvGNNs) are closely related to recurrent graph neural networks. Instead of iterating node states with contractive constraints, ConvGNNs address the cyclic mutual dependencies architecturally using a fixed number of layers with different weights in each layer. This key distinction is illustrated in Figure 2.2. As graph convolutions are more efficient and convenient to composite with other neural networks, the popularity of ConvGNNs has been rapidly growing in recent years. ConvGNNs fall into two categories, spectral-based and spatial-based. Spectral-based approaches define graph convolutions by introducing filters from the perspective of graph signal processing [86] where the graph convolutional operation is interpreted as removing noises from graph signals. Spatial-based approaches inherit ideas from RecGNNs to define graph convolutions by information propagation. Since GCN [1] bridged the gap between spectral-based approaches and spatial-based approaches, spatial-based methods have developed rapidly recently due to its attractive efficiency, flexibility, and generality.

2.5.1 Spectral-based ConvGNNs

Background Spectral-based methods have a solid mathematical foundation in graph signal processing [86–88]. They assume graphs to be undirected. The normalized graph Laplacian matrix is a mathematical representation of an undirected graph, defined as $\mathbf{L} = \mathbf{I_n} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, where $\mathbf{D}$ is a diagonal matrix of node degrees, $\mathbf{D}_{ii} = \sum_j(\mathbf{A}_{i,j})$. The

normalized graph Laplacian matrix possesses the property of being real symmetric positive semidefinite. With this property, the normalized Laplacian matrix can be factored as $\mathbf{L} = \mathbf{U}\Lambda\mathbf{U}^T$, where $\mathbf{U} = [\mathbf{u_0}, \mathbf{u_1}, \cdots, \mathbf{u_{n-1}}] \in \mathbf{R}^{n \times n}$ is the matrix of eigenvectors ordered by eigenvalues and Λ is the diagonal matrix of eigenvalues (spectrum), $\Lambda_{ii} = \lambda_i$. The eigenvectors of the normalized Laplacian matrix form an orthonormal space, in mathematical words $\mathbf{U}^T\mathbf{U} = \mathbf{I}$. In graph signal processing, a graph signal $\mathbf{x} \in \mathbf{R}^n$ is a feature vector of all nodes of a graph where x_i is the value of the i^{th} node. The *graph Fourier transform* to a signal $\mathbf{x}$ is defined as $\mathscr{F}(\mathbf{x}) = \mathbf{U}^T\mathbf{x}$, and the inverse graph Fourier transform is defined as $\mathscr{F}^{-1}(\hat{\mathbf{x}}) = \mathbf{U}\hat{\mathbf{x}}$, where $\hat{\mathbf{x}}$ represents the resulted signal from the graph Fourier transform. The graph Fourier transform projects the input graph signal to the orthonormal space where the basis is formed by eigenvectors of the normalized graph Laplacian. Elements of the transformed signal $\hat{\mathbf{x}}$ are the coordinates of the graph signal in the new space so that the input signal can be represented as $\mathbf{x} = \sum_i \hat{x}_i \mathbf{u}_i$, which is exactly the inverse graph Fourier transform. Now the graph convolution of the input signal $\mathbf{x}$ with a filter $\mathbf{g} \in \mathbf{R}^n$ is defined as

$$\begin{aligned} \mathbf{x} *_G \mathbf{g} &= \mathscr{F}^{-1}(\mathscr{F}(\mathbf{x}) \odot \mathscr{F}(\mathbf{g})) \\ &= \mathbf{U}(\mathbf{U}^T\mathbf{x} \odot \mathbf{U}^T\mathbf{g}), \end{aligned} \tag{2.4}$$

where $\odot$ denotes the element-wise product. If we denote a filter as $\mathbf{g}_\theta = diag(\mathbf{U}^T\mathbf{g})$, then the spectral graph convolution is simplified as

$$\mathbf{x} *_G \mathbf{g}_\theta = \mathbf{U}\mathbf{g}_\theta\mathbf{U}^T\mathbf{x}. \tag{2.5}$$

Spectral-based ConvGNNs all follow this definition. The key difference lies in the choice of the filter $\mathbf{g}_\theta$.

Spectral Convolutional Neural Network (Spectral CNN) [30] assumes the filter $\mathbf{g}_\theta = \Theta_{i,j}^{(k)}$ is a set of learnable parameters and considers graph signals with multiple channels. The graph convolutional layer of Spectral CNN is defined as

$$\mathbf{H}_{:,j}^{(k)} = \sigma\Big(\sum_{i=1}^{f_{k-1}} \mathbf{U}\Theta_{i,j}^{(k)}\mathbf{U}^T\mathbf{H}_{:,i}^{(k-1)}\Big) \quad (j = 1, 2, \cdots, f_k), \tag{2.6}$$

where k is the layer index, $\mathbf{H}^{(k-1)} \in \mathbf{R}^{n \times f_{k-1}}$ is the input graph signal, $\mathbf{H}^{(0)} = \mathbf{X}$, f_{k-1} is the number of input channels and f_k is the number of output channels, $\Theta_{i,j}^{(k)}$ is a diagonal matrix filled with learnable parameters. Due to the eigen-decomposition of the Laplacian matrix, Spectral CNN faces three limitations. First, any perturbation to a graph results in a change of eigenbasis. Second, the learned filters are domain dependent, meaning

they cannot be applied to a graph with a different structure. Third, eigen-decomposition requires $O(n^3)$ computational complexity. In follow-up works, ChebNet [32] and GCN [1] reduce the computational complexity to $O(m)$ by making several approximations and simplifications.

Chebyshev Spectral CNN (ChebNet) [32] approximates the filter $\mathbf{g}_\theta$ by Chebyshev polynomials of the diagonal matrix of eigenvalues, i.e, $\mathbf{g}_\theta = \sum_{i=0}^{K} \theta_i T_i(\tilde{\Lambda})$, where $\tilde{\Lambda} = 2\Lambda/\lambda_{max} - \mathbf{I_n}$, and the values of $\tilde{\Lambda}$ lie in $[-1, 1]$. The Chebyshev polynomials are defined recursively by $T_i(\mathbf{x}) = 2\mathbf{x}T_{i-1}(\mathbf{x}) - T_{i-2}(\mathbf{x})$ with $T_0(\mathbf{x}) = 1$ and $T_1(\mathbf{x}) = \mathbf{x}$. As a result, the convolution of a graph signal $\mathbf{x}$ with the defined filter $\mathbf{g}_\theta$ is

$$\mathbf{x} *_G \mathbf{g}_\theta = \mathbf{U}(\sum_{i=0}^{K} \theta_i T_i(\tilde{\Lambda}))\mathbf{U}^T\mathbf{x}, \tag{2.7}$$

where $\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{max} - \mathbf{I_n}$. As $T_i(\tilde{\mathbf{L}}) = \mathbf{U}T_i(\tilde{\Lambda})\mathbf{U}^T$, which can be proven by induction on i, ChebNet takes the form,

$$\mathbf{x} *_G \mathbf{g}_\theta = \sum_{i=0}^{K} \theta_i T_i(\tilde{\mathbf{L}})\mathbf{x}, \tag{2.8}$$

As an improvement over Spectral CNN, the filters defined by ChebNet are localized in space, which means filters can extract local features independently of the graph size. The spectrum of ChebNet is mapped to $[-1, 1]$ linearly. CayleyNet [33] further applies Cayley polynomials which are parametric rational complex functions to capture narrow frequency bands. The spectral graph convolution of CayleyNet is defined as

$$\mathbf{x} *_G \mathbf{g}_\theta = c_0\mathbf{x} + 2Re\{\sum_{j=1}^{r} c_j(h\mathbf{L} - i\mathbf{I})^j(h\mathbf{L} + i\mathbf{I})^{-j}\mathbf{x}\}, \tag{2.9}$$

where $Re(\cdot)$ returns the real part of a complex number, c_0 is a real coefficent, c_j is a complex coefficent, i is the imaginary number, and h is a parameter which controls the spectrum of a Cayley filter. While preserving spatial locality, CayleyNet shows that ChebNet can be considered as a special case of CayleyNet.

Graph Convolutional Network (GCN) [1] introduces a first-order approximation of ChebNet. Assuming $K = 1$ and $\lambda_{max} = 2$, Equation 2.8 is simplified as

$$\mathbf{x} *_G \mathbf{g}_\theta = \theta_0\mathbf{x} - \theta_1\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{x}. \tag{2.10}$$

To restrain the number of parameters and avoid over-fitting, GCN further assume $\theta = \theta_0 = -\theta_1$, leading to the following definition of a graph convolution,

$$\mathbf{x} *_G \mathbf{g}_\theta = \theta(\mathbf{I_n} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{x}. \tag{2.11}$$

To allow multi-channels of inputs and outputs, GCN modifies Equation 2.11 into a compositional layer, defined as

$$\mathbf{H} = \mathbf{X} *_G \mathbf{g}_\Theta = f(\bar{\mathbf{A}}\mathbf{X}\Theta), \tag{2.12}$$

where $\bar{\mathbf{A}} = \mathbf{I_n} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ and $f(\cdot)$ is an activation function. Using $\mathbf{I_n} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ empirically causes numerical instability to GCN. To address this problem, GCN applies a normalization trick to replace $\bar{\mathbf{A}} = \mathbf{I_n} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ by $\bar{\mathbf{A}} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ with $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I_n}$ and $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$. Being a spectral-based method, GCN can be also interpreted as a spatial-based method. From a spatial-based perspective, GCN can be considered as aggregating feature information from a node's neighborhood. Equation 2.12 can be expressed as

$$\mathbf{h}_v = f(\Theta^T(\sum_{u \in \{N(v) \cup v\}} \bar{A}_{v,u}\mathbf{x}_u)) \quad \forall v \in V. \tag{2.13}$$

Several recent works made incremental improvements over GCN [1] by exploring alternative symmetric matrices. Adaptive Graph Convolutional Network (AGCN) [50] learns hidden structural relations unspecified by the graph adjacency matrix. It constructs a so-called residual graph adjacency matrix through a learnable distance function which takes two nodes' features as inputs. Dual Graph Convolutional Network (DGCN) [51] introduces a dual graph convolutional architecture with two graph convolutional layers in parallel. While these two layers share parameters, they use the normalized adjacency matrix $\bar{\mathbf{A}}$ and the positive pointwise mutual information (PPMI) matrix which captures nodes co-occurrence information through random walks sampled from a graph. The PPMI matrix is defined as

$$\mathbf{PPMI}_{v_1,v_2} = max(\log(\frac{count(v_1,v_2) \cdot |D|}{count(v_1)count(v_2)}), 0), \tag{2.14}$$

where $v_1, v_2 \in V$, $|D| = \sum_{v_1,v_2} count(v_1,v_2)$ and the $count(\cdot)$ function returns the frequency that node v and/or node u co-occur/occur in sampled random walks. By ensembling outputs from dual graph convolutional layers, DGCN encodes both local and global structural information without the need to stack multiple graph convolutional layers.

2.5.2 Spatial-based ConvGNNs

Analogous to the convolutional operation of a conventional CNN on an image, spatial-based methods define graph convolutions based on a node's spatial relations. Images can be considered as a special form of graphs with each pixel representing a node. Each pixel is directly connected to its nearby pixels, as illustrated in Figure 1.1a. A filter is

applied to a 3×3 patch by taking the weighted average of pixel values of the central node and its neighbors across each channel. Similarly, the spatial-based graph convolutions convolve the central node's representation with its neighbors' representations to derive the updated representation for the central node, as illustrated in Figure 1.1b. From another perspective, spatial-based ConvGNNs share the same idea of information propagation/message passing with RecGNNs. The spatial graph convolutional operation essentially propagates node information along edges.

Neural Network for Graphs (NN4G) [34], proposed in parallel with GNN*, is the first work towards spatial-based ConvGNNs. Distinctively different from RecGNNs, NN4G learns graph mutual dependency through a compositional neural architecture with independent parameters at each layer. The neighborhood of a node can be extended through incremental construction of the architecture.

NN4G performs graph convolutions by summing up a node's neighborhood information directly. It also applies residual connections and skip connections to memorize information over each layer. As a result, NN4G derives its next layer node states by

$$\mathbf{h}_v^{(k)} = f(\mathbf{W}^{(k)^T}\mathbf{x}_v + \sum_{i=1}^{k-1}\sum_{u\in N(v)} \Theta^{(k)^T}\mathbf{h}_u^{(k-1)}), \tag{2.15}$$

where $f(\cdot)$ is an activation function and $\mathbf{h}_v^{(0)} = \mathbf{0}$. Equation 2.15 can also be written in a matrix form:

$$\mathbf{H}^{(k)} = f(\mathbf{X}\mathbf{W}^{(k)} + \sum_{i=1}^{k-1}\mathbf{A}\mathbf{H}^{(k-1)}\Theta^{(k)}), \tag{2.16}$$

which resembles the form of GCN [1]. One difference is that NN4G uses the unnormalized adjacency matrix which may potentially cause hidden node states to have extremely different scales. Contextual Graph Markov Model (CGMM) [56] proposes a probabilistic model inspired by NN4G. While maintaining spatial locality, CGMM has the benefit of probabilistic interpretability.

Diffusion Convolutional Neural Network (DCNN) [35] regards graph convolutions as a diffusion process. It assumes information is transferred from one node to one of its neighboring nodes with a certain transition probability so that information distribution can reach equilibrium after several rounds. DCNN defines the diffusion graph convolution as

$$\mathbf{H}^{(k)} = f(\mathbf{W}^{(k)} \odot \mathbf{P}^k\mathbf{X}), \tag{2.17}$$

where $f(\cdot)$ is an activation function and the probability transition matrix $\mathbf{P} \in \mathbf{R}^{n \times n}$ is computed by $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$. Note that in DCNN, the hidden representation matrix $\mathbf{H}^{(k)}$ remains the same dimension as the input feature matrix $\mathbf{X}$ and is not a function of its previous hidden representation matrix $\mathbf{H}^{(k-1)}$. DCNN concatenates $\mathbf{H}^{(1)}, \mathbf{H}^{(2)}, \cdots, \mathbf{H}^{(K)}$ together as the final model outputs. As the stationary distribution of a diffusion process is a summation of power series of probability transition matrices, Diffusion Graph Convolution (DGC) [13] sums up outputs at each diffusion step instead of concatenation. It defines the diffusion graph convolution by

$$\mathbf{H} = \sum_{k=0}^{K} f(\mathbf{P}^k \mathbf{X} \mathbf{W}^{(k)}), \tag{2.18}$$

where $\mathbf{W}^{(k)} \in \mathbf{R}^{D \times F}$ and $f(\cdot)$ is an activation function. Using the power of a transition probability matrix implies that distant neighbors contribute very little information to a central node. PGC-DGCNN [55] increases the contributions of distant neighbors based on shortest paths. It defines a shortest path adjacency matrix $\mathbf{S}^{(j)}$. If the shortest path from a node v to a node u is of length j, then $\mathbf{S}^{(j)}_{v,u} = 1$ otherwise 0. With a hyperparameter r to control the receptive field size, PGC-DGCNN introduces a graph convolutional operation as follows

$$\mathbf{H}^{(k)} = \|_{j=0}^{r} f((\tilde{\mathbf{D}}^{(j)})^{-1} \mathbf{S}^{(j)} \mathbf{H}^{(k-1)} \mathbf{W}^{(j,k)}), \tag{2.19}$$

where $\tilde{D}^{(j)}_{ii} = \sum_l S^{(j)}_{i,l}$, $\mathbf{H}^{(0)} = \mathbf{X}$, and $\|$ represents the concatenation of vectors. The calculation of the shortest path adjacency matrix can be expensive with $O(n^3)$ at maximum. Partition Graph Convolution (PGC) [15] partitions a node's neighbors into Q groups based on certain criteria not limited to shortest paths. PGC constructs Q adjacency matrices according to the defined neighborhood by each group. Then, PGC applies GCN [1] with a different parameter matrix to each neighbor group and sums the results:

$$\mathbf{H}^{(k)} = \sum_{j=1}^{Q} \bar{\mathbf{A}}^{(j)} \mathbf{H}^{(k-1)} \mathbf{W}^{(j,k)}, \tag{2.20}$$

where $\mathbf{H}^{(0)} = \mathbf{X}$, $\bar{\mathbf{A}}^{(j)} = (\tilde{\mathbf{D}}^{(j)})^{-\frac{1}{2}} \tilde{\mathbf{A}}^{(j)} (\tilde{\mathbf{D}}^{(j)})^{-\frac{1}{2}}$ and $\tilde{\mathbf{A}}^{(j)} = \mathbf{A}^{(j)} + \mathbf{I}$.

Message Passing Neural Network (MPNN) [37] outlines a general framework of spatial-based ConvGNNs. It treats graph convolutions as a message passing process in which information can be passed from one node to another along edges directly. MPNN runs K-step message passing iterations to let information propagate further. The message passing function (namely the spatial graph convolution) is defined as

$$\mathbf{h}_v^{(k)} = U_k(\mathbf{h}_v^{(k-1)}, \sum_{u \in N(v)} M_k(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathbf{x}^e_{vu})), \tag{2.21}$$

where $\mathbf{h}_v^{(0)} = \mathbf{x}_v$, $U_k(\cdot)$ and $M_k(\cdot)$ are functions with learnable parameters. After deriving the hidden representations of each node, $\mathbf{h}_v^{(K)}$ can be passed to an output layer to perform node-level prediction tasks or to a readout function to perform graph-level prediction tasks. The readout function generates a representation of the entire graph based on node hidden representations. It is generally defined as

$$\mathbf{h}_G = R(\mathbf{h}_v^{(K)} | v \in G), \tag{2.22}$$

where $R(\cdot)$ represents the readout function with learnable parameters. MPNN can cover many existing GNNs by assuming different forms of $U_k(\cdot), M_k(\cdot)$, and $R(\cdot)$, such as [1, 89–91]. However, Graph Isomorphism Network (GIN) [66] finds that previous MPNN-based methods are incapable of distinguishing different graph structures based on the graph embedding they produced. To amend this drawback, GIN adjusts the weight of the central node by a learnable parameter $\epsilon^{(k)}$. It performs graph convolutions by

$$\mathbf{h}_v^{(k)} = MLP((1+\epsilon^{(k)})\mathbf{h}_v^{(k-1)} + \sum_{u \in N(v)} \mathbf{h}_u^{(k-1)}), \tag{2.23}$$

where $MLP(\cdot)$ represents a multi-layer perceptron.

As the number of neighbors of a node can vary from one to a thousand or even more, it is inefficient to take the full size of a node's neighborhood. GraphSage [52] adopts sampling to obtain a fixed number of neighbors for each node. It performs graph convolutions by

$$\mathbf{h}_v^{(k)} = \sigma(\mathbf{W}^{(k)} \cdot f_k(\mathbf{h}_v^{(k-1)}, \{\mathbf{h}_u^{(k-1)}, \forall u \in S_{\mathcal{N}(v)}\})), \tag{2.24}$$

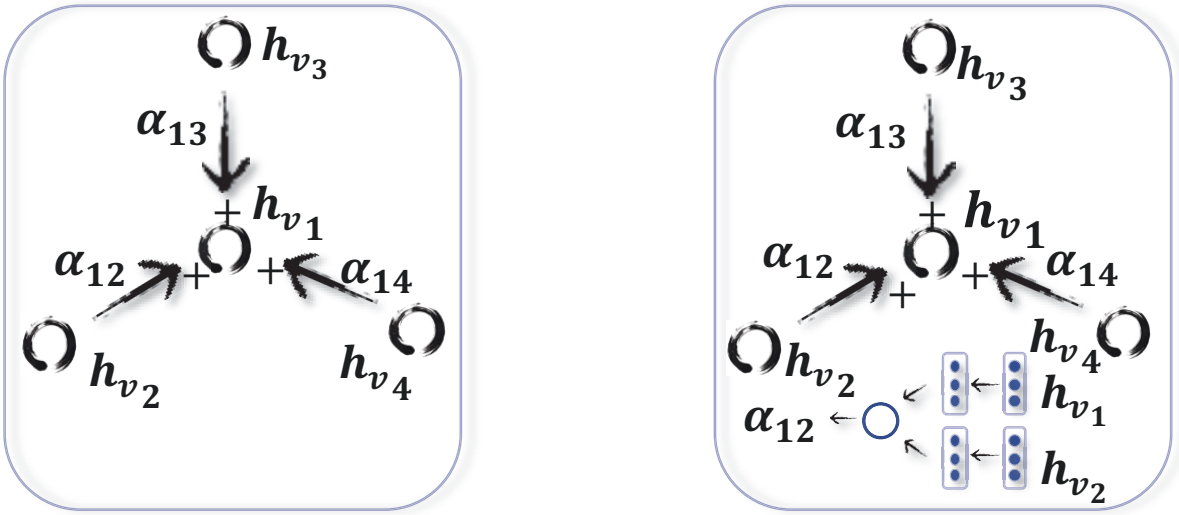
where $\mathbf{h}_v^{(0)} = \mathbf{x}_v$, $f_k(\cdot)$ is an aggregation function, $S_{\mathcal{N}(v)}$ is a random sample of the node v's neighbors. The aggregation function should be invariant to the permutations of node orderings such as a mean, sum or max function.

Graph Attention Network (GAT) [2] assumes contributions of neighboring nodes to the central node are neither identical like GraphSage [52], nor pre-determined like GCN [1] (this difference is illustrated in Figure 2.3). GAT adopts attention mechanisms to learn the relative weights between two connected nodes. The graph convolutional operation according to GAT is defined as,

$$\mathbf{h}_v^{(k)} = \sigma(\sum_{u \in \mathcal{N}(v) \cup v} \alpha_{vu}^{(k)} \mathbf{W}^{(k)} \mathbf{h}_u^{(k-1)}), \tag{2.25}$$

where $\mathbf{h}_v^{(0)} = \mathbf{x}_v$. The attention weight $\alpha_{vu}^{(k)}$ measures the connective strength between the node v and its neighbor u:

$$\alpha_{vu}^{(k)} = softmax(g(\mathbf{a}^T[\mathbf{W}^{(k)}\mathbf{h}_v^{(k-1)} || \mathbf{W}^{(k)}\mathbf{h}_u^{(k-1)})), \tag{2.26}$$



(a) GCN [1] explicitly assigns a non-parametric weight $a_{ij} = \frac{1}{\sqrt{deg(v_i)deg(v_j)}}$ to the neighbor v_j of v_i during the aggregation process.

(b) GAT [2] implicitly captures the weight a_{ij} via an end-to-end neural network architecture, so that more important nodes receive larger weights.

Figure 2.3: Differences between GCN [1] and GAT [2]

where $g(\cdot)$ is a LeakyReLU activation function and $\mathbf{a}$ is a vector of learnable parameters. The softmax function ensures that the attention weights sum up to one over all neighbors of the node v. GAT further performs the multi-head attention to increase the model's expressive capability. This shows an impressive improvement over GraphSage on node classification tasks. While GAT assumes the contributions of attention heads are equal, Gated Attention Network (GAAN) [57] introduces a self-attention mechanism which computes an additional attention score for each attention head. Apart from applying graph attention spatially, GeniePath [64] further proposes an LSTM-like gating mechanism to control information flow across graph convolutional layers. There are other graph attention models which might be of interest [92, 93]. However, they do not belong to the ConvGNN framework.

Mixture Model Network (MoNet) [53] adopts a different approach to assign different weights to a node's neighbors. It introduces node pseudo-coordinates to determine the relative position between a node and its neighbor. Once the relative position between two nodes is known, a weight function maps the relative position to the relative weight between these two nodes. In such a way, the parameters of a graph filter can be shared across different locations. Under the MoNet framework, several existing approaches for manifolds such as Geodesic CNN (GCNN) [94], Anisotropic CNN (ACNN) [95], Spline

Table 2.4: Time and memory complexity comparison for ConvGNN training algorithms (summarized by [3]). n is the total number of nodes. m is the total number of edges. K is the number of layers. s is the batch size. r is the number of neighbors being sampled for each node. For simplicity, the dimensions of the node hidden features remain constant, denoted by d.

Complexity	GCN [1]	GraphSage [52]	FastGCN [58]	StoGCN [59]	Cluster-GCN [3]
Time	$O(Kmd+Knd^2)$	$O(r^K nd^2)$	$O(Krnd^2)$	$O(Kmd+Knd^2+r^K nd^2)$	$O(Kmd+Knd^2)$
Memory	$O(Knd+Kd^2)$	$O(sr^K d+Kd^2)$	$O(Ksrd+Kd^2)$	$O(Knd+Kd^2)$	$O(Ksd+Kd^2)$

CNN [96], and for graphs such as GCN [1], DCNN [35] can be generalized as special instances of MoNet by constructing nonparametric weight functions. MoNet additionally proposes a Gaussian kernel with learnable parameters to learn the weight function adaptively.

Another distinct line of works achieves weight sharing across different locations by ranking a node's neighbors based on certain criteria and associating each ranking with a learnable weight. PATCHY-SAN [36] orders neighbors of each node according to their graph labelings and selects the top q neighbors. Graph labelings are essentially node scores which can be derived by node degree, centrality, and Weisfeiler-Lehman color [97, 98]. As each node now has a fixed number of ordered neighbors, graph-structured data can be converted into grid-structured data. PATCHY-SAN applies a standard 1D convolutional filter to aggregate neighborhood feature information where the order of the filter's weights corresponds to the order of a node's neighbors. The ranking criterion of PATCHY-SAN only consider graph structures, which require heavy computation for data processing. Large-scale Graph Convolutional Network (LGCN) [54] ranks a node's neighbors based on node feature information. For each node, LGCN assembles a feature matrix which consists of its neighborhood and sorts this feature matrix along each column. The first q rows of the sorted feature matrix are taken as the input data for the central node.

Improvement in terms of training efficiency Training ConvGNNs such as GCN [1] usually is required to save the whole graph data and intermediate states of all nodes into memory. The full-batch training algorithm for ConvGNNs suffers significantly from the memory overflow problem, especially when a graph contains millions of nodes. To save memory, GraphSage [52] proposes a batch-training algorithm for ConvGNNs. It samples a tree rooted at each node by recursively expanding the root node's neighborhood by K steps with fixed sample size. For each sampled tree, GraphSage computes the root node's hidden representation by hierarchically aggregating hidden node representations from

bottom to top.

Fast Learning with Graph Convolutional Network (FastGCN) [58] samples a fixed number of nodes for each graph convolutional layer instead of sampling a fixed number of neighbors for each node like GraphSage [52]. It interprets graph convolutions as integral transforms of embedding functions of nodes under probability measures. Monte Carlo approximation and variance reduction techniques are employed to facilitate the training process. As FastGCN samples nodes independently for each layer, between-layers connections are potentially sparse. Huang et al. [60] propose an adaptive layer-wise sampling approach where node sampling for the lower layer is conditioned on the top one. This method achieves higher accuracy compared to FastGCN at the cost of employing a much more complicated sampling scheme.

In another work, Stochastic Training of Graph Convolutional Networks (StoGCN) [59] reduces the receptive field size of a graph convolution to an arbitrarily small scale using historical node representations as a control variate. StoGCN achieves comparable performance even with two neighbors per node. However, StoGCN still has to save intermediate states of all nodes, which is memory-consuming for large graphs.

Cluster-GCN [3] samples a subgraph using a graph clustering algorithm and performs graph convolutions to nodes within the sampled subgraph. As the neighborhood search is also restricted within the sampled subgraph, Cluster-GCN is capable of handling larger graphs and using deeper architectures at the same time, in less time and with less memory. Cluster-GCN notably provides a straightforward comparison of time complexity and memory complexity for existing ConvGNN training algorithms. I analyze its results based on Table 2.4.

In Table 2.4, GCN [1] is the baseline method which conducts the full-batch training. GraphSage saves memory at the cost of sacrificing time efficiency. Meanwhile, the time and memory complexity of GraphSage grows exponentially with an increase of K and r. The time complexity of Sto-GCN is the highest, and the bottleneck of the memory remains unsolved. However, Sto-GCN can achieve satisfactory performance with very small r. The time complexity of Cluster-GCN remains the same as the baseline method since it does not introduce redundant computations. Of all the methods, Cluster-GCN realizes the lowest memory complexity.

Comparison between spectral and spatial models Spectral models have a theoretical foundation in graph signal processing. By designing new graph signal filters (e.g., Cayleynets [33]), one can build new ConvGNNs. However, spatial models are preferred over spectral models due to efficiency, generality, and flexibility issues. First, spectral

models are less efficient than spatial models. Spectral models either need to perform eigenvector computation or handle the whole graph at the same time. Spatial models are more scalable to large graphs as they directly perform convolutions in the graph domain via information propagation. The computation can be performed in a batch of nodes instead of the whole graph. Second, spectral models which rely on a graph Fourier basis generalize poorly to new graphs. They assume a fixed graph. Any perturbations to a graph would result in a change of eigenbasis. Spatial-based models, on the other hand, perform graph convolutions locally on each node where weights can be easily shared across different locations and structures. Third, spectral-based models are limited to operating on undirected graphs. Spatial-based models are more flexible to handle multi-source graph inputs such as edge inputs [26, 37, 90, 99, 100], directed graphs [13, 35], signed graphs [101], and heterogeneous graphs [102, 103], because these graph inputs can be incorporated into the aggregation function easily.

2.5.3 Graph Pooling Modules

After a GNN generates node features, we can use them for the final task. But using all these features directly can be computationally challenging, thus, a down-sampling strategy is needed. Depending on the objective and the role it plays in the network, different names are given to this strategy: (1) the pooling operation aims to reduce the size of parameters by down-sampling the nodes to generate smaller representations and thus avoid overfitting, permutation invariance, and computational complexity issues; (2) the readout operation is mainly used to generate graph-level representation based on node representations. Their mechanism is very similar. In this chapter, I use pooling to refer to all kinds of down-sampling strategies applied to GNNs.

In some earlier works, the graph coarsening algorithms use eigen-decomposition to coarsen graphs based on their topological structure. However, these methods suffer from the time complexity issue. The Graclus algorithm [104] is an alternative of eigen-decomposition to calculate a clustering version of the original graph. Some recent works [33] employed it as a pooling operation to coarsen graphs.

Nowadays, mean/max/sum pooling is the most primitive and effective way to implement down-sampling since calculating the mean/max/sum value in the pooling window is fast:

$$\mathbf{h}_G = mean/max/sum(\mathbf{h}_1^{(K)}, \mathbf{h}_2^{(K)}, ..., \mathbf{h}_n^{(K)}), \tag{2.27}$$

where K is the index of the last graph convolutional layer.

Henaff et al. [31] show that performing a simple max/mean pooling at the beginning of the network is especially important to reduce the dimensionality in the graph domain and mitigate the cost of the expensive graph Fourier transform operation. Furthermore, some works [28, 37, 55] also use attention mechanisms to enhance the mean/sum pooling.

Even with attention mechanisms, the reduction operation (such as sum pooling) is not satisfactory since it makes the embedding inefficient: a fixed-size embedding is generated regardless of the graph size. Vinyals et al. [105] propose the Set2Set method to generate a memory that increases with the size of the input. It then implements an LSTM that intends to integrate order-dependent information into the memory embedding before a reduction is applied that would otherwise destroy that information.

Defferrard et al. [32] address this issue in another way by rearranging nodes of a graph in a meaningful way. They devise an efficient pooling strategy in their approach ChebNet. Input graphs are first coarsened into multiple levels by the Graclus algorithm [104]. After coarsening, the nodes of the input graph and its coarsened version are rearranged into a balanced binary tree. Arbitrarily aggregating the balanced binary tree from bottom to top will arrange similar nodes together. Pooling such a rearranged signal is much more efficient than pooling the original.

Zhang et al. [61] propose the DGCNN with a similar pooling strategy named SortPooling which performs pooling by rearranging nodes to a meaningful order. Different from ChebNet [32], DGCNN sorts nodes according to their structural roles within the graph. The graph's unordered node features from spatial graph convolutions are treated as continuous WL colors [97], and they are then used to sort nodes. In addition to sorting the node features, it unifies the graph size to q by truncating/extending the node feature matrix. The last $n-q$ rows are deleted if $n > q$, otherwise $q-n$ zero rows are added.

The aforementioned pooling methods mainly consider graph features and ignore the structural information of graphs. Recently, a differentiable pooling (DiffPool) [63] is proposed, which can generate hierarchical representations of graphs. Compared to all previous coarsening methods, DiffPool does not simply cluster the nodes in a graph but learns a cluster assignment matrix $\mathbf{S}$ at layer k referred to as $\mathbf{S}^{(k)} \in \mathbf{R}^{n_k \times n_{k+1}}$, where n_k is the number of nodes at the k^{th} layer. The probability values in matrix $\mathbf{S}^{(k)}$ are being generated based on node features and topological structure using

$$\mathbf{S}^{(k)} = softmax(ConvGNN_k(\mathbf{A}^{(k)}, \mathbf{H}^{(k)})). \tag{2.28}$$

The core idea of this is to learn comprehensive node assignments which consider both topological and feature information of a graph, so Equation 2.28 can be implemented

with any standard ConvGNNs. However, the drawback of DiffPool is that it generates dense graphs after pooling and thereafter the computational complexity becomes $O(n^2)$.

Most recently, the SAGPool [106] approach is proposed, which considers both node features and graph topology and learns the pooling in a self-attention manner.

Overall, pooling is an essential operation to reduce graph size. How to improve the effectiveness and computational complexity of pooling is an open question for investigation.

2.5.4 Discussion of Theoretical Aspects

I discuss the theoretical foundation of graph neural networks from different perspectives.

Shape of receptive field The receptive field of a node is the set of nodes that contribute to the determination of its final node representation. When compositing multiple spatial graph convolutional layers, the receptive field of a node grows one step ahead towards its distant neighbors each time. Micheli [34] proved that a finite number of spatial graph convolutional layers exists such that for each node $v \in V$ the receptive field of node v covers all nodes in the graph. As a result, a ConvGNN is able to extract global information by stacking local graph convolutional layers.

VC dimension The VC dimension is a measure of model complexity defined as the largest number of points that can be shattered by a model. There are few works on analyzing the VC dimension of GNNs. Given the number of model parameter p and the number of nodes n, Scarselli et al. [107] derive that the VC dimension of a GNN* [26] is $O(p^4n^2)$ if it uses the sigmoid or tangent hyperbolic activation and is $O(p^2n)$ if it uses the piecewise polynomial activation function. This result suggests that the model complexity of a GNN* [26] increases rapidly with p and n if the sigmoid or tangent hyperbolic activation is used.

Graph isomorphism Two graphs are isomorphic if they are topologically identical. Given two non-isomorphic graphs G_1 and G_2, Xu et al. [66] prove that if a GNN maps G_1 and G_2 to different embeddings, these two graphs can be identified as non-isomorphic by the Weisfeiler-Lehman (WL) test of isomorphism [97]. They show that common GNNs such as GCN [1] and GraphSage [52] are incapable of distinguishing different graph structures. Xu et al. [66] further prove if the aggregation functions and the readout functions of a GNN are injective, the GNN is at most as powerful as the WL test in distinguishing different graphs.

Table 2.5: Main characteristics of selected GAEs

Approaches	Inputs	Encoder	Decoder	Objective
DNGR (2016) [67]	A	a multi-layer perceptron	a multi-layer perceptron	reconstruct the PPMI matrix
SDNE (2016) [68]	A	a multi-layer perceptron	a multi-layer perceptron	preserve node 1st-order and 2nd-order proximity
GAE* (2016) [69]	A,X	a ConvGNN	a similarity measure	reconstruct the adjacency matrix
VGAE (2016) [69]	A,X	a ConvGNN	a similarity measure	learn the generative distribution of data
ARVGA (2018) [70]	A,X	a ConvGNN	a similarity measure	learn the generative distribution of data adversarially
DNRE (2018) [71]	A	an LSTM network	an identity function	recover network embedding
NetRA (2018) [72]	A	an LSTM network	an LSTM network	recover network embedding with adversarial training
DeepGMG (2018) [73]	A,X,X^e	a RecGNN	a decision process	maximize the expected joint log-likelihood
GraphRNN (2018) [74]	A	a RNN	a decision process	maximize the likelihood of permutations
GraphVAE (2018) [75]	A,X,X^e	a ConvGNN	a multi-layer perceptron	optimize the reconstruction loss
RGVAE (2018) [76]	A,X,X^e	a CNN	a deconvolutional net	optimize the reconstruction loss with validity constraints
MolGAN (2018) [77]	A,X,X^e	a ConvGNN	a multi-layer perceptron	optimize the generative adversarial loss and the RL loss
NetGAN (2018) [78]	A	an LSTM network	an LSTM network	optimize the generative adversarial loss

Equivariance and invariance A GNN must be an equivariant function when performing node-level tasks and must be an invariant function when performing graph-level tasks. For node-level tasks, let $f(\mathbf{A},\mathbf{X}) \in R^{n\times d}$ be a GNN and $\mathbf{Q}$ be any permutation matrix that changes the order of nodes. A GNN is equivariant if it satisfies $f(\mathbf{QAQ}^T,\mathbf{QX}) = \mathbf{Q}f(\mathbf{A},\mathbf{X})$. For graph-level tasks, let $f(\mathbf{A},\mathbf{X}) \in R^d$. A GNN is invariant if it satisfies $f(\mathbf{QAQ}^T,\mathbf{QX}) = f(\mathbf{A},\mathbf{X})$. In order to achieve equivariance or invariance, components of a GNN must be invariant to node orderings. Maron et al. [108] theoretically study the characteristics of permutation invariant and equivariant linear layers for graph data.

Universal approximation It is well known that multi-perceptron feedforward neural networks with one hidden layer can approximate any Borel measurable functions [109]. The universal approximation capability of GNNs has seldom been studied. Hammer et al. [110] prove that cascade correlation can approximate functions with structured outputs. Scarselli et al. [111] prove that a RecGNN [26] can approximate any function that preserves unfolding equivalence up to any degree of precision. Two nodes are unfolding equivalent if their unfolding trees are identical where the unfolding tree of a node is constructed by iteratively extending a node's neighborhood at a certain depth. Xu et al. [66] show that ConvGNNs under the framework of message passing [37] are not universal approximators of continuous functions defined on multisets. Maron et al. [108] prove that an invariant graph network can approximate an arbitrary invariant function defined on graphs.

2.6 Graph autoencoders

Graph autoencoders (GAEs) are deep neural architectures which map nodes into a latent feature space and decode graph information from latent representations. GAEs can be used to learn network embeddings or generate new graphs. The main characteristics of selected GAEs are summarized in Table 2.5. In the following, I provide a brief review of GAEs from two perspectives, network embedding and graph generation.

2.6.1 Network Embedding

A network embedding is a low-dimensional vector representation of a node which preserves a node's topological information. GAEs learn network embeddings using an encoder to extract network embeddings and using a decoder to enforce network embeddings to preserve the graph topological information such as the PPMI matrix and the adjacency matrix.

Earlier approaches mainly employ multi-layer perceptrons to build GAEs for network embedding learning. Deep Neural Network for Graph Representations (DNGR) [67] uses a stacked denoising autoencoder [112] to encode and decode the PPMI matrix via multi-layer perceptrons. Concurrently, Structural Deep Network Embedding (SDNE) [68] uses a stacked autoencoder to preserve the node first-order proximity and second-order proximity jointly. SDNE proposes two loss functions on the outputs of the encoder and the outputs of the decoder separately. The first loss function enables the learned network embeddings to preserve the node first-order proximity by minimizing the distance between a node's network embedding and its neighbors' network embeddings. The first loss function L_{1st} is defined as

$$L_{1st} = \sum_{(v,u)\in E} A_{v,u} ||enc(\mathbf{x}_v) - enc(\mathbf{x}_u)||^2, \tag{2.29}$$

where $\mathbf{x}_v = \mathbf{A}_{v,:}$ and $enc(\cdot)$ is an encoder which consists of a multi-layer perceptron. The second loss function enables the learned network embeddings to preserve the node second-order proximity by minimizing the distance between a node's inputs and its reconstructed inputs. Concretely, the second loss function L_{2nd} is defined as

$$L_{2nd} = \sum_{v\in V} ||(dec(enc(\mathbf{x}_v)) - \mathbf{x}_v) \odot \mathbf{b}_v||^2, \tag{2.30}$$

where $b_{v,u} = 1$ if $A_{v,u} = 0$, $b_{v,u} = \beta > 1$ if $A_{v,u} = 1$, and $dec(\cdot)$ is a decoder which consists of a multi-layer perceptron.

DNGR [67] and SDNE [68] only consider node structural information which is about the connectivity between pairs of nodes. They ignore nodes may contain feature information that depicts the attributes of nodes themselves. Graph Autoencoder (GAE*[3]) [69] leverages GCN [1] to encode node structural information and node feature information at the same time. The encoder of GAE* consists of two graph convolutional layers, which takes the form

$$\mathbf{Z} = enc(\mathbf{X},\mathbf{A}) = Gconv(f(Gconv(\mathbf{A},\mathbf{X};\Theta_{\mathbf{1}}));\Theta_{\mathbf{2}}), \tag{2.31}$$

where $\mathbf{Z}$ denotes the network embedding matrix of a graph, $f(\cdot)$ is a ReLU activation function and the $Gconv(\cdot)$ function is a graph convolutional layer defined by Equation 2.12. The decoder of GAE* aims to decode node relational information from their embeddings by reconstructing the graph adjacency matrix, which is defined as

$$\hat{\mathbf{A}}_{v,u} = dec(\mathbf{z}_v,\mathbf{z}_u) = \sigma(\mathbf{z}_v^T\mathbf{z}_u), \tag{2.32}$$

where $\mathbf{z}_v$ is the embedding of node v. GAE* is trained by minimizing the negative cross entropy given the real adjacency matrix $\mathbf{A}$ and the reconstructed adjacency matrix $\hat{\mathbf{A}}$.

Simply reconstructing the graph adjacency matrix may lead to overfitting due to the capacity of the autoencoders. Variational Graph Autoencoder (VGAE) [69] is a variational version of GAE to learn the distribution of data. VGAE optimizes the variational lower bound L:

$$L = E_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{A}|\mathbf{Z})] - KL[q(\mathbf{Z}|\mathbf{X},\mathbf{A})||p(\mathbf{Z})], \tag{2.33}$$

where $KL(\cdot)$ is the Kullback-Leibler divergence function which measures the distance between two distributions, $p(\mathbf{Z})$ is a Gaussian prior $p(\mathbf{Z}) = \prod_{i=1}^{n} p(\mathbf{z}_i) = \prod_{i=1}^{n} N(\mathbf{z}_i|0,\mathbf{I})$, $p(A_{ij} = 1|\mathbf{z}_i,\mathbf{z}_j) = dec(\mathbf{z}_i,\mathbf{z}_j) = \sigma(\mathbf{z}_i^T\mathbf{z}_j)$, $q(\mathbf{Z}|\mathbf{X},\mathbf{A}) = \prod_{i=1}^{n} q(\mathbf{z}_i|\mathbf{X},\mathbf{A})$, with $q(\mathbf{z}_i|\mathbf{X},\mathbf{A}) = N(\mathbf{z}_i|\mu_i, diag(\sigma_i^2))$. The mean vector μ_i is the i^{th} row of an encoder's outputs defined by Equation 2.31 and $\log\sigma_i$ is derived similarly as μ_i with another encoder. According to Equation 2.33, VGAE assumes the empirical distribution $q(\mathbf{Z}|\mathbf{X},\mathbf{A})$ should be as close as possible to the prior distribution $p(\mathbf{Z})$. To further enforce the empirical distribution $q(\mathbf{Z}|\mathbf{X},\mathbf{A})$ approximate the prior distribution $p(\mathbf{Z})$, Adversarially Regularized Variational Graph Autoencoder (ARVGA) [70, 113] employs the training scheme of a generative adversarial networks (GAN) [114]. A GAN plays a competition game between a generator

[3] I name it GAE* to avoid ambiguity in this chapter.

and a discriminator in training generative models. A generator tries to generate 'fake samples' to be as real as possible while a discriminator attempts to distinguish the 'fake samples' from real ones. Inspired by GANs, ARVGA endeavors to learn an encoder that produces an empirical distribution $q(\mathbf{Z}|\mathbf{X},\mathbf{A})$ which is indistinguishable from the prior distribution $p(\mathbf{Z})$.

Similar as GAE*, GraphSage [52] encodes node features with two graph convolutional layers. Instead of optimizing the reconstruction error, GraphSage shows that the relational information between two nodes can be preserved by negative sampling with the loss:

$$L(\mathbf{z}_v) = -log(dec(\mathbf{z}_v, \mathbf{z}_u)) - QE_{v_n \sim P_n(v)} \log(-dec(\mathbf{z}_v, \mathbf{z}_{v_n})), \tag{2.34}$$

where node u is a neighbor of node v, node v_n is a distant node to node v and is sampled from a negative sampling distribution $P_n(v)$, and Q is the number of negative samples. This loss function essentially enforces close nodes to have similar representations and distant nodes to have dissimilar representations. DGI [65] alternatively drives local network embeddings to capture global structural information by maximizing local mutual information. It shows a distinct improvement over GraphSage [52] experimentally.

For the aforementioned methods, they essentially learn network embeddings by solving a link prediction problem. However, the sparsity of a graph causes the number of positive node pairs to be far less than the number of negative node pairs. To alleviate the data sparsity problem in learning network embedding, another line of works converts a graph into sequences by random permutations or random walks. In such a way, those deep learning approaches which are applicable to sequences can be directly used to process graphs. Deep Recursive Network Embedding (DRNE) [71] assumes a node's network embedding should approximate the aggregation of its neighborhood network embeddings. It adopts a Long Short-Term Memory (LSTM) network [10] to aggregate a node's neighbors. The reconstruction error of DRNE is defined as

$$L = \sum_{v \in V} ||\mathbf{z}_v - LSTM(\{\mathbf{z}_u | u \in N(v)\})||^2, \tag{2.35}$$

where $\mathbf{z}_v$ is the network embedding of node v obtained by a dictionary look-up, and the LSTM network takes a random sequence of node v's neighbors ordered by their node degree as inputs. As suggested by Equation 2.35, DRNE implicitly learns network embeddings via an LSTM network rather than using the LSTM network to generate network embeddings. It avoids the problem that the LSTM network is not invariant to the permutation of node sequences. Network Representations with Adversarially

Regularized Autoencoders (NetRA) [72] proposes a graph encoder-decoder framework with a general loss function, defined as

$$L = -E_{\mathbf{z}\sim P_{data}(\mathbf{z})}(dist(\mathbf{z}, dec(enc(\mathbf{z})))), \tag{2.36}$$

where $dist(\cdot)$ is the distance measure between the node embedding $\mathbf{z}$ and the reconstructed $\mathbf{z}$. The encoder and decoder of NetRA are LSTM networks with random walks rooted on each node $v \in V$ as inputs. Similar to ARVGA [70], NetRA regularizes the learned network embeddings within a prior distribution via adversarial training. Although NetRA ignores the node permutation variant problem of LSTM networks, the experimental results validate the effectiveness of NetRA.

2.6.2 Graph Generation

With multiple graphs, GAEs are able to learn the generative distribution of graphs by encoding graphs into hidden representations and decoding a graph structure given hidden representations. The majority of GAEs for graph generation is designed to solve the molecular graph generation problem, which has a high practical value in drug discovery. These methods either propose a new graph in a sequential manner or in a global manner.

Sequential approaches generate a graph by proposing nodes and edges step by step. Gomez et al. [115], Kusner et al. [116], and Dai et al. [117] model the generation process of a string representation of molecular graphs named SMILES with deep CNNs and RNNs as the encoder and the decoder respectively. While these methods are domain-specific, alternative solutions are applicable to general graphs by means of iteratively adding nodes and edges to a growing graph until a certain criterion is satisfied. Deep Generative Model of Graphs (DeepGMG) [73] assumes the probability of a graph is the sum over all possible node permutations:

$$p(G) = \sum_{\pi} p(G, \pi), \tag{2.37}$$

where π denotes a node ordering. It captures the complex joint probability of all nodes and edges in the graph. DeepGMG generates graphs by making a sequence of decisions, namely whether to add a node, which node to add, whether to add an edge, and which node to connect to the new node. The decision process of generating nodes and edges is conditioned on the node states and the graph state of a growing graph updated by a RecGNN. In another work, GraphRNN [74] proposes a graph-level RNN and an edge-level RNN to model the generation process of nodes and edges. The graph-level RNN

adds a new node to a node sequence each time while the edge-level RNN produces a binary sequence indicating connections between the new node and the nodes previously generated in the sequence.

Global approaches output a graph all at once. Graph Variational Autoencoder (GraphVAE) [75] models the existence of nodes and edges as independent random variables. By assuming the posterior distribution $q_\phi(\mathbf{z}|G)$ defined by an encoder and the generative distribution $p_\theta(G|\mathbf{z})$ defined by a decoder, GraphVAE optimizes the variational lower bound:

$$L(\phi,\theta;G) = E_{q_\phi(z|G)}[-\log p_\theta(G|\mathbf{z})] + KL[q_\phi(\mathbf{z}|G)||p(\mathbf{z})], \tag{2.38}$$

where $p(\mathbf{z})$ follows a Gaussian prior, ϕ and θ are learnable parameters. With a ConvGNN as the encoder and a simple multi-layer perception as the decoder, GraphVAE outputs a generated graph with its adjacency matrix, node attributes and edge attributes. It is challenging to control the global properties of generated graphs, such as graph connectivity, validity, and node compatibility. Regularized Graph Variational Autoencoder (RGVAE) [76] further imposes validity constraints on a graph variational autoencoder to regularize the output distribution of the decoder. Molecular Generative Adversarial Network (MolGAN) [77] integrates convGNNs [118], GANs [119] and reinforcement learning objectives to generate graphs with the desired properties. MolGAN consists of a generator and a discriminator, competing with each other to improve the authenticity of the generator. In MolGAN, the generator tries to propose a fake graph along with its feature matrix while the discriminator aims to distinguish the fake sample from the empirical data. Additionally, a reward network is introduced in parallel with the discriminator to encourage the generated graphs to possess certain properties according to an external evaluator. NetGAN [78] combines LSTMs [10] with Wasserstein GANs [120] to generate graphs from a random-walk-based approach. NetGAN trains a generator to produce plausible random walks through an LSTM network and enforces a discriminator to identify fake random walks from the real ones. After training, a new graph is derived by normalizing a co-occurrence matrix of nodes computed based on random walks produced by the generator.

In brief, sequential approaches linearize graphs into sequences. They can lose structural information due to the presence of cycles. Global approaches produce a graph all at once. They are not scalable to large graphs as the output space of a GAE is up to $O(n^2)$.

2.7 Spatial-temporal Graph Neural Networks

Graphs in many real-world applications are dynamic both in terms of graph structures and graph inputs. Spatial-temporal graph neural networks (STGNNs) occupy important positions in capturing the dynamicity of graphs. Methods under this category aim to model the dynamic node inputs while assuming interdependency between connected nodes. For example, a traffic network consists of speed sensors placed on roads where edge weights are determined by the distance between pairs of sensors. As the traffic condition of one road may depend on its adjacent roads' conditions, it is necessary to consider spatial dependency when performing traffic speed forecasting. As a solution, STGNNs capture spatial and temporal dependencies of a graph simultaneously. The task of STGNNs can be forecasting future node values or labels, or predicting spatial-temporal graph labels. STGNNs follow two directions, RNN-based methods and CNN-based methods.

Most RNN-based approaches capture spatial-temporal dependencies by filtering inputs and hidden states passed to a recurrent unit using graph convolutions [13, 19, 57]. To illustrate this, suppose a simple RNN takes the form

$$\mathbf{H}^{(t)} = \sigma(\mathbf{W}\mathbf{X}^{(t)} + \mathbf{U}\mathbf{H}^{(t-1)} + \mathbf{b}), \tag{2.39}$$

where $\mathbf{X}^{(t)} \in \mathbf{R}^{n \times d}$ is the node feature matrix at time step t. After inserting graph convolution, Equation 2.39 becomes

$$\mathbf{H}^{(t)} = \sigma(Gconv(\mathbf{X}^{(t)}, \mathbf{A}; \mathbf{W}) + Gconv(\mathbf{H}^{(t-1)}, \mathbf{A}; \mathbf{U}) + \mathbf{b}), \tag{2.40}$$

where $Gconv(\cdot)$ is a graph convolutional layer. Graph Convolutional Recurrent Network (GCRN) [19] combines a LSTM network with ChebNet [32]. Diffusion Convolutional Recurrent Neural Network (DCRNN) [13] incorporates a proposed diffusion graph convolutional layer (Equation 2.18) into a GRU network. In addition, DCRNN adopts an encoder-decoder framework to predict the future K steps of node values.

Another parallel work uses node-level RNNs and edge-level RNNs to handle different aspects of temporal information. Structural-RNN [18] proposes a recurrent framework to predict node labels at each time step. It comprises two kinds of RNNs, namely a node-RNN and an edge-RNN. The temporal information of each node and each edge is passed through a node-RNN and an edge-RNN respectively. To incorporate the spatial information, a node-RNN takes the outputs of edge-RNNs as inputs. Since assuming different RNNs for different nodes and edges significantly increases model complexity, it instead splits nodes and edges into semantic groups. Nodes or edges in the same semantic group share the same RNN model, which saves the computational cost.

RNN-based approaches suffer from time-consuming iterative propagation and gradient explosion/vanishing issues. As alternative solutions, CNN-based approaches tackle spatial-temporal graphs in a non-recursive manner with the advantages of parallel computing, stable gradients, and low memory requirements. As illustrated in Fig 2.1d, CNN-based approaches interleave 1D-CNN layers with graph convolutional layers to learn temporal and spatial dependencies respectively. Assume the inputs to a spatial-temporal graph neural network is a tensor $\mathscr{X} \in R^{T \times n \times d}$, the 1D-CNN layer slides over $\mathscr{X}_{[:,i,:]}$ along the time axis to aggregate temporal information for each node while the graph convolutional layer operates on $\mathscr{X}_{[i,:,:]}$ to aggregate spatial information at each time step. CGCN [79] integrates 1D convolutional layers with ChebNet [32] or GCN [1] layers. It constructs a spatial-temporal block by stacking a gated 1D convolutional layer, a graph convolutional layer and another gated 1D convolutional layer in a sequential order. ST-GCN [15] composes a spatial-temporal block using a 1D convolutional layer and a PGC layer (Equation 2.20).

Previous methods all use a pre-defined graph structure. They assume the pre-defined graph structure reflects the genuine dependency relationships among nodes. However, with many snapshots of graph data in a spatial-temporal setting, it is possible to learn latent static graph structures automatically from data. To realize this, Graph WaveNet [80] proposes a self-adaptive adjacency matrix to perform graph convolutions. The self-adaptive adjacency matrix is defined as

$$\mathbf{A}_{adp} = SoftMax(ReLU(\mathbf{E}_1 \mathbf{E}_2^T)), \tag{2.41}$$

where the SoftMax function is computed along the row dimension, **E1** denotes the source node embedding and **E2** denotes the target node embedding with learnable parameters. By multiplying **E1** with **E2**, one can get the dependency weight between a source node and a target node. With a complex CNN-based spatial-temporal neural network, Graph WaveNet performs well without being given an adjacency matrix.

Learning latent static spatial dependencies can help researchers discover interpretable and stable correlations among different entities in a network. However, in some circumstances, learning latent dynamic spatial dependencies may further improve model precision. For example, in a traffic network, the travel time between two roads may depend on their current traffic conditions. GaAN [57] employs attention mechanisms to learn dynamic spatial dependencies through an RNN-based approach. An attention function is used to update the edge weight between two connected nodes given their current node inputs. ASTGCN [81] further includes a spatial attention function and a

temporal attention function to learn latent dynamic spatial dependencies and temporal dependencies through a CNN-based approach. The common drawback of learning latent spatial dependencies is that it needs to calculate the spatial dependency weight between each pair of nodes, which costs $O(n^2)$.

2.8 Applications

As graph-structured data are ubiquitous, GNNs have a wide variety of applications. In this section, I summarize the benchmark graph data sets, evaluation methods, and open-source implementation, respectively. I detail practical applications of GNNs in various domains.

2.8.1 Data Sets

I mainly sort data sets into four groups, namely citation networks, biochemical graphs, social networks, and others. In Table 2.6, I summarize selected benchmark data sets. More details is given in Supplementary Material A.1.

Table 2.6: Summary of selected benchmark data sets.

Category	Data set	Source	# Graphs	# Nodes(Avg.)	# Edges (Avg.)	#Features	# Classes	Citation
Citation Networks	Cora	[121]	1	2708	5429	1433	7	[1, 2, 33, 35, 51, 53, 54] [58–60, 62, 65, 69, 70]
	Citeseer	[121]	1	3327	4732	3703	6	[1, 2, 51, 54, 59, 60, 62] [65, 69, 70]
	Pubmed	[121]	1	19717	44338	500	3	[1, 2, 29, 35, 51, 53, 54] [58, 60, 62, 64, 65, 69, 70] [78, 99]
	DBLP (v11)	[122]	1	4107340	36624464	-	-	[72, 78, 103]
Biochemical Graphs	PPI	[123]	24	56944	818716	50	121	[2, 29, 52, 54, 57, 59, 64] [3, 65, 72]
	NCI-1	[124]	4110	29.87	32.30	37	2	[35, 36, 55, 61, 66, 100, 102]
	MUTAG	[125]	188	17.93	19.79	7	2	[35, 36, 55, 61, 66, 100]
	D&D	[126]	1178	284.31	715.65	82	2	[36, 55, 61, 63, 100, 102]
	PROTEIN	[127]	1113	39.06	72.81	4	2	[36, 55, 61, 63, 66]
	PTC	[128]	344	25.5	-	19	2	[35, 36, 55, 61, 66]
	QM9	[129]	133885	-	-	-	-	[37, 77]
	Alchemy	[130]	119487	-	-	-	-	-
Social Networks	Reddit	[52]	1	232965	11606919	602	41	[52, 57–60, 65]
	BlogCatalog	[131]	1	10312	333983	-	39	[29, 64, 68, 72]
Others	MNIST	[132]	70000	784	-	1	10	[30, 32, 33, 53, 100]
	METR-LA	[133]	1	207	1515	2	-	[13, 57, 80]
	Nell	[134]	1	65755	266144	61278	210	[1, 51, 59]

2.8.2 Evaluation & Open-source Implementations

Node classification and graph classification are common tasks to assess the performance of RecGNNs and ConvGNNs.

Node Classification In node classification, most methods follow a standard split of train/valid/test on benchmark data sets including Cora, Citeseer, Pubmed, PPI, and Reddit. They reported the average accuracy or F1 score on the test data set over multiple runs. A summarization of experimental results of methods can be found in Supplementary Material A.2. It should be noted that these results do not necessarily represent a rigorous comparison. Shchur et al. identified [135] two pitfalls in evaluating the performance GNNs on node classification. First, using the same train/valid/test split throughout all experiments underestimates the generalization error. Second, different methods employed different training techniques such as hyper-parameter tuning, parameter initialization, learning rate decay, and early stopping. For a relatively fair comparison, I refer the readers to Shchur et al. [135].

Graph Classification In graph classification, researchers often adopt 10-fold cross validation (cv) for model evaluation. However, as pointed out by [136], the experimental settings are ambiguous and not unified across different works. In particular, [136] raises the concern of the correct usage of data splits for model selection versus model assessment. An often encountered problem is that the external test set of each fold is used both for model selection and risk assessment. [136] compare GNNs in a standardized and uniform evaluation framework. They apply an external 10 fold CV to get an estimate of the generalization performance of a model and an inner holdout technique with a 90%/10% training/validation split for model selection. An alternative procedure would be a double cv method, which uses an external k fold cv for model assessment and an inner k fold cv for model selection. I refer the readers to [136] for a detailed and rigorous comparison of GNN methods for graph classification.

Open-source implementations facilitate the work of baseline experiments in deep learning research. In Supplementary Material A.3, I provide the hyperlinks of the open-source implementations of the GNN models reviewed in this chapter. Noticeably, Fey et al. [96] published a geometric learning library in PyTorch named PyTorch Geometric [4], which implements many GNNs. Most recently, the Deep Graph Library (DGL) [5] [137] is released which provides a fast implementation of many GNNs on top of popular deep

[4] `https://github.com/rusty1s/pytorch_geometric`
[5] https://www.dgl.ai/

learning platforms such as PyTorch and MXNet.

2.8.3 Practical Applications

GNNs have many applications across different tasks and domains. Despite general tasks which can be handled by each category of GNNs directly, including node classification, graph classification, network embedding, graph generation, and spatial-temporal graph forecasting, other general graph-related tasks such as node clustering [138], link prediction [139], and graph partitioning [140] can also be addressed by GNNs. I detail some applications based on the following research domains.

Computer vision Applications of GNNs in computer vision include scene graph generation, point clouds classification, and action recognition.

Recognizing semantic relationships between objects facilitates the understanding of the meaning behind a visual scene. Scene graph generation models aim to parse an image into a semantic graph which consists of objects and their semantic relationships [141–143]. Another application inverses the process by generating realistic images given scene graphs [144]. As natural language can be parsed as semantic graphs where each word represents an object, it is a promising solution to synthesize images given textual descriptions.

Classifying and segmenting points clouds enables LiDAR devices to 'see' the surrounding environment. A point cloud is a set of 3D points recorded by LiDAR scans. [145–147] convert point clouds into k-nearest neighbor graphs or superpoint graphs and use ConvGNNs to explore the topological structure.

Identifying human actions contained in videos facilitates a better understanding of video content from a machine aspect. Some solutions detect the locations of human joints in video clips. Human joints which are linked by skeletons naturally form a graph. Given the time series of human joint locations, [15, 18] apply STGNNs to learn human action patterns.

Moreover, the number of applicable directions of GNNs in computer vision is still growing. It includes human-object interaction [148], few-shot image classification [149–151], semantic segmentation [152, 153], visual reasoning [154], and question answering [155].

Natural language processing A common application of GNNs in natural language processing is text classification. GNNs utilize the inter-relations of documents or words to infer document labels [1, 2, 52].

Despite the fact that natural language data exhibit a sequential order, they may also contain an internal graph structure, such as a syntactic dependency tree. A syntactic dependency tree defines the syntactic relations among words in a sentence. Marcheggiani et al. [156] propose the Syntactic GCN which runs on top of a CNN/RNN sentence encoder. The Syntactic GCN aggregates hidden word representations based on the syntactic dependency tree of a sentence. Bastings et al. [157] apply the Syntactic GCN to the task of neural machine translation. Marcheggiani et al. [158] further adopt the same model as Bastings et al. [157] to handle the semantic dependency graph of a sentence.

Graph-to-sequence learning learns to generate sentences with the same meaning given a semantic graph of abstract words (known as Abstract Meaning Representation). Song et al. [159] propose a graph-LSTM to encode graph-level semantic information. Beck et al. [160] apply a GGNN [28] to graph-to-sequence learning and neural machine translation. The inverse task is sequence-to-graph learning. Generating a semantic or knowledge graph given a sentence is very useful in knowledge discovery [161, 162].

Traffic Accurately forecasting traffic speed, volume or the density of roads in traffic networks is fundamentally important in a smart transportation system. [13, 57, 79] address the traffic prediction problem using STGNNs. They consider the traffic network as a spatial-temporal graph where the nodes are sensors installed on roads, the edges are measured by the distance between pairs of nodes, and each node has the average traffic speed within a window as dynamic input features. Another industrial-level application is taxi-demand prediction. Given historical taxi demands, location information, weather data, and event features, Yao et al. [14] incorporate LSTM, CNN and network embeddings trained by LINE [163] to form a joint representation for each location to predict the number of taxis demanded for a location within a time interval.

Recommender systems Graph-based recommender systems take items and users as nodes. By leveraging the relations between items and items, users and users, users and items, as well as content information, graph-based recommender systems are able to produce high-quality recommendations. The key to a recommender system is to score the importance of an item to a user. As a result, it can be cast as a link prediction problem. To predict the missing links between users and items, Van et al. [164] and Ying et al. [165] propose a GAE which uses ConvGNNs as encoders. Monti et al. [166] combine RNNs with graph convolutions to learn the underlying process that generates the known ratings.

Chemistry In the field of chemistry, researchers apply GNNs to study the graph struc-

ture of molecules/compounds. In a molecule/compound graph, atoms are considered as nodes, and chemical bonds are treated as edges. Node classification, graph classification, and graph generation are the three main tasks targeting molecular/compound graphs in order to learn molecular fingerprints [89, 90], to predict molecular properties [37], to infer protein interfaces [167], and to synthesize chemical compounds [73, 77, 168].

Federated learning Federated Learning is a new machine learning paradigm that can collaboratively learn an intelligent model with privacy preserving. It has been broadly applied to various industry sectors, such as finance [169, 170], healthcare [171, 172], and smartphones [173–175]. GCN can be applied to enhance the learning process of the FL system by leveraging the structural information among distributed clients [176] which used to be heterogeneous [177] and with hidden community structures [178, 179].

Others The application of GNNs is not limited to the aforementioned domains and tasks. There have been explorations of applying GNNs to a variety of problems such as program verification [28], program reasoning [180], social influence prediction [181], adversarial attacks prevention [182], electrical health records modeling [183, 184], brain networks [185], event detection [186], and combinatorial optimization [187].

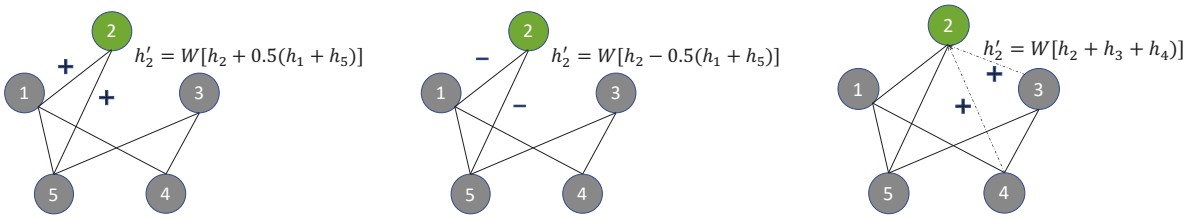
CHAPTER 3

BEYOND LOW-PASS FILTERING: GRAPH CONVOLUTIONALNETWORKS WITH AUTOMATIC FILTERING

3.1 Motivation

Over the past years, convolutional neural networks and recurrent neural networks have achieved great success on grid data such as images and sequences. Moving forward, the focus of research gradually shifts from regular grid data to irregular non-Euclidean data [12]. Being an important kind of non-Euclidean data, graphs are ubiquitous in the real world, such as cyber networks and social networks. Graph data describes the relationship, association, or interaction among different entities. To extract latent representations from data, graph neural networks [188, 189], in particular graph convolutional networks, are developed specifically for graphs. Graph neural networks not only have leveled up benchmarks on conventional graph-related tasks [37, 75, 113, 138, 139, 190–192], but also has been proven to be helpful in solving many deep learning problems where structural dependencies exist [80, 145, 156, 165, 193].

Among graph neural networks, the study of graph convolutional networks is built upon the foundation of graph signal processing. With the eigen-decomposition of the graph Laplacian matrix, a graph convolutional filter can be defined as a function of frequency (eigenvalue). Through proper design of the filter function, a graph convolutional filter can be viewed from the spatial domain - it smooths a node's inputs by



(a) An example of a low-pass filter. It smooths the node self information by taking the average of its neighboring information into account.

(b) An example of a high-pass filter. It computes the difference between the node self information with its neighboring information.

(c) An example of a middle-pass filter. It integrates a node self information with its second hop neighborhood information.

Figure 3.1: Illustration of low-pass, high-pass filters, and middle-pass filters.

aggregating information from the node's neighborhood. Due to the advantage of efficiency, generality, and flexibility, there is a trend of designing graph convolutional networks directly on spatial domains without considering their spectral properties. As a result, these spatial-designed approaches may only focus on the low frequency band of graph signals [194]. However, the middle and high frequency band of graph signals should not be ignored because they may contain useful information as well.

To consider low, middle, high frequency band of graph signals at the same time, we need to return back to spectral-based approaches. As a spectral-designed graph convolutional filter is defined by a function of frequency, it can theoretically extract information on any frequency band. Pioneer works of spectral graph convolutional networks are computationally inefficient due to eigen-decomposition of the graph Laplacian matrix [30]. Defferrard et al. [32] proposed ChebNet with the filter function defined as Chebyshev polynomials. With graph Fourier transform and inverse graph Fourier transform, the graph convolutional operation essentially is reduced to multiplying linear transformed graph inputs with non-linear transformed graph Laplacian matrix. In this way, eigen-decomposition is not necessarily required. The graph kernel of ChebNet involves computing higher orders of the graph Laplacian matrix. Kipf et al. [1] further propose GCN which is a first-order approximation of ChebNet with a renormalization trick. Despite that GCN improves over ChebNet in terms of efficiency, GCN is shown to be a low-pass filter [194]. Follow-up works of GCN also remain the same problem [50, 51, 195]. Most recently, Balcilar et al. [194] propose DSGCN with arbitrary graph convolutional filter functions, while taking low, middle, and high frequency band of graph signals into consideration. However, filter functions of DSGCN are customized differently according to datasets without any rules to follow. It requires tremendous effort to find the optimal

filter function manually for a new dataset. More importantly, the bandwidth of graph convolutional filters of DSGCN is fixed, which also applies to other existing spectral graph convolutions. Parameters of a graph convolutional filter only transform graph inputs without changing the curvature of a graph convolutional filter function. Therefore, the cut-off frequency or the bandwidth of a graph convolutional filter remains unchanged throughout learning. In reality, we are uncertain about whether we should retain or cut off the frequency at a certain point unless we have expert domain knowledge.

In this chapter, I aim to design a graph convolutional network that can capture the whole spectrum of graph signals in a more efficient and effective way. My method consists of three graph convolutional filters, a low-pass, a middle-pass, and a high-pass filter. To avoid eigen-decomposition, I limit the choice of filter functions within linear and quadratic forms of the graph Laplacian matrix. More specifically, the low-pass and high-pass filters are designed to be linear functions while the middle-pass filter has a quadratic form. The roles of low-pass, high-pass, and middle-pass filter are illustrated in Figure 3.1. Different from existing spectral-based methods, I introduce extra parameters to control the curvature and scope of all three filters. As a benefit, the bandwidth and magnitude of my graph convolutional filters can be adjusted automatically during training.

The main contributions of this chapter are summarized as follows,

- I propose an Automatic Graph Convolutional Network (AutoGCN) with three novel graph convolutional filters, a low-pass linear filter, a high-pass linear filter, and a middle-pass quadratic filter. While capturing the whole spectrum of graph signals, AutoGCN ends up with a spatial form without performing eigen-decomposition.

- I enable the proposed graph convolutional filters to control their bandwidth and magnitude automatically by updating the curvature and scope of filter functions during training. I empirically show that all three graph filters contribute to model performance.

- Experimental results show that AutoGCN achieves significant improvement over baseline methods that only function as low-pass filters on medium-scale datasets for both node classification and graph prediction tasks.

My source codes are publicly available at https://github.com/nnzhan/AutoGCN.git. The rest of this chapter is organized as follows. In Section 3.2, I summarize current works of graph convolutional networks. In Section 3.3, I formally define my problems. In Section 3.4, I provide the background knowledge about spectral-rooted spatial graph

convolution. In Section 3.5, I present my method named automatic graph convolution in detail. In Section 3.6, I report the experimental results of my method on both node classification and graph prediction tasks. Finally, in Section 3.7, I make a conclusion of this chapter.

3.2 Related Work to AutoGCN

The study of graph convolutional networks is rooted in graph signal processing or spectral graph theory [86]. By defining the graph Fourier transform and the inverse graph Fourier transform of a graph signal, the convolution between a graph signal and a filter can be derived by the convolution theorem where the Fourier transform of the convolution of two signals equals the elementwise product of their Fourier transforms. With this theorem, Bruna et al. [30] define graph convolutional filters as functions of eigenvalues of the graph Laplacian matrix. Eigen-decomposition of a graph Laplacian matrix is computationally expensive. To bypass this bottleneck, Defferrard et al. [32] show that a filter function defined on the eigenvalues is equivalent to the same function defined on the graph Laplacian matrix. Based on this result, various of filter functions which are defined on the graph Laplacian matrix directly have been proposed such as Chebyshev polynomials [32], a first-order approximation of Chebyshev polynomials [1], and Cayley polynomials [196]. Besides graph Fourier transform, another line of works defines graph convolution through graph wavelet transform [197, 198]. These methods are localized in the vertex domain and do not require eigen-decomposition as well. However, complex operations on the graph Laplacian matrix still impede model efficiency due to higher-order computation.

Concurrently, despite graph convolution, message passing has mostly dominated the recent development of graph neural networks due to its efficiency, generality, and flexibility. The basic idea is to propagate graph signals along graph structures. By iterating the propagation step multiple times, a node can broaden its neighborhood to the entire graph. Earlier schemes of graph message passing follow the recurrent architecture, where the parameters are shared across multiple propagation steps [26, 28, 29]. These methods update node states recursively until steady states are reached. On the other side, compositional schemes modularize message passing as a neural network layer to improve model pluggability and capacity [34]. Gilmer et al. [37] formalize the message passing framework with two components, a propagator and a updator. The propagator summarizes information for a node based on its neighborhood context. The updator

transforms the collected information with learnable parameters. The most intuitive propagate function is the mean function, which takes the average of the center node information and its neighborhood information. Various improvements are made over the mean aggregate function by refining edge weights through normalization [1] and diffusion [35], assigning learnable weights to neighbors [2, 53], introducing a scalar parameter to pass graph isomorphism test [199], teleporting back to a node's initial information to avoid the over-smoothing problem [200]. Some other advanced works focus on designing complex architectures of graph neural networks by increasing network depth and receptive field size [201–203]. Graph neural networks based on message passing do not consider spectral properties of the message passing operations, namely graph convolution. Wu et al. [195] prove that stacking multiple GCN with identity activation is a low-pass filter. Balcilar et al. [194] show that most graph neural networks are essentially low-pass filters. In parallel to graph convolution transforms, graph wavelet transforms decompose graph signals in multi-scale [204, 205]. These works also consider the high-frequency part of graph signals, but they lack the ability to automatically adjust the bandwidth of graph signals based on data.

3.3 Problem Formulation

Attributed graph. An attributed graph $G = (V, E, \mathbf{X})$ consists of a set of nodes V, a set of edges E, and a node feature matrix $\mathbf{X} \in \mathbf{R}^{n \times d}$ where n is the number of nodes for the graph G and d is the feature dimension. The structural information of the graph G can be encoded by a graph adjacency matrix $\mathbf{A} \in \mathbf{R}^{n \times n}$. If there exists a connection $(v_i, v_j) \in E$, then $\mathbf{A}_{ij} \neq 0$, otherwise $\mathbf{A}_{ij} = 0$. In this chapter, I only consider undirected attributed graphs. In this case, $\mathbf{A}$ is symmetric.

Node-level prediction. The node-level prediction task forecasts a label for each node in a graph. It aims to learn a mapping $f : (\mathbf{A}, \mathbf{X}) \rightarrow \mathbf{Y} \in \mathbf{R}^{n \times c}$. For node regression problems, $c = 1$. For node classification problems, c equals the number of classes.

Graph-level prediction. The graph-level prediction task forecasts a label for each graph in a data set. It aims to learn a mapping $f : (\mathbf{A}, \mathbf{X}) \rightarrow \mathbf{Y} \in \mathbf{R}^{1 \times c}$. For graph regression problems, $c = 1$. For graph classification problems, c equals the number of classes.

3.4 Spectral-Rooted Spatial Graph Convolution

Graph convolutional networks generally fall into spectral-based approaches and spatial-based approaches. Several representative methods such as ChebNet [32] and GCN [1] take roots in the spectral domain while ending up with spatial forms. As a key benefit, they avoid the computation of eigen-decomposition of the graph Laplacian matrix in order to perform graph convolution. Most recently, Balcilar et al. [194] connect spectral-based approaches and spatial-based approaches by a uniform formula, which is defined as

$$\mathbf{H}^{(l+1)} = \sigma(\sum_{k=1}^{K} \mathbf{C}^{(k)}\mathbf{H}^{(l)}\mathbf{W}^{(l,k)}) \tag{3.1}$$

with the graph convolutional kernel set to

$$\mathbf{C}^{(k)} = \mathbf{U}diag(F_k(\boldsymbol{\lambda}))\mathbf{U}^T \tag{3.2}$$

where $F_k(\cdot)$ is the filter function, $\mathbf{U}$ and $\boldsymbol{\lambda}$ denote the eigenvectors and the eigenvalues of the normalized graph Laplacian matrix $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ respectively, $\mathbf{D}$ is a diagnal matrix with $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$, $\mathbf{W}^{(l,k)}$ is a learnable parameter matrix, $\mathbf{H}^{(l)}$ is the node representation matrix at layer l, $\mathbf{H}^{(1)} = \mathbf{X}$, K is the number of filter functions, and $\sigma(\cdot)$ represents the activation function. With Equation 3.1 and 3.2, spectral-based apporaches can be designed by defining new filter functions, and spatial-based approaches can be analyzed from the spectral domain by computing the frequency profile (the filter function) of the convolutional kernel $\mathbf{C}^{(k)}$, using

$$F_k(\boldsymbol{\lambda}) = diag(\mathbf{U}^T\mathbf{C}^{(k)}\mathbf{U}). \tag{3.3}$$

Under this framework, ChebNet takes the filter function $F_1(\boldsymbol{\lambda}) = \mathbf{1}$, $F_2(\boldsymbol{\lambda}) = 2\boldsymbol{\lambda}/\lambda_{max} - \mathbf{1}$, and $F_k(\boldsymbol{\lambda}) = 2F_2(\boldsymbol{\lambda})F_{k-1}(\boldsymbol{\lambda}) - F_{k-2}(\boldsymbol{\lambda})$. As $\mathbf{L} = \mathbf{U}diag(\boldsymbol{\lambda})\mathbf{U}^T$, the convolutional kernel of ChebNet, i.e. $\mathbf{C}^{(k)}$, results in a polynomial function of $\mathbf{L}$ with order $k-1$. Taking higher orders of $\mathbf{L}$ is computationally expensive. GCN simplifies ChebNet with first order approximation by setting $K = 2$, $\lambda_{max} = 2$, and $\mathbf{W}^{(l,1)} = -\mathbf{W}^{(l,2)}$. The form of GCN is then derived as,

$$\mathbf{H}^{(l+1)} = \sigma(\mathbf{I}\mathbf{H}^{(l)}\mathbf{W}^{(l,1)} - (\mathbf{L} - \mathbf{I})\mathbf{H}^{(l)}\mathbf{W}^{(l,1)}) \tag{3.4}$$

$$= \sigma((\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{H}^{(l)}\mathbf{W}^{(l,1)}) \tag{3.5}$$

With a renormalization trick to avoid numerical instabilities, GCN replaces $\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ by $\tilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I})\tilde{\mathbf{D}}^{-\frac{1}{2}}$, where $\tilde{\mathbf{D}}_{ii} = \sum_j \mathbf{A}_{ij} + 1$. Proved by Balcilar et al. [194], the frequency

profile of GCN can be approximated by $F(\boldsymbol{\lambda}) = \mathbf{1} - \boldsymbol{\lambda}\bar{d}/(1+\bar{d})$. This corresponds to a low-pass filter with the cut-off frequency $(1+\bar{d})/\bar{d}$. Although GCN is simple and robust, it only contains a single low-pass filter. It is incapable of capturing potential patterns in middle and high frequency bands. DSGCN [194] incorporates low-pass, middle-pass and high-pass filters by proposing a set of customized filter functions $F(\boldsymbol{\lambda})$ for a particular dataset. For example, DSGCN proposes four filter functions for the CORA dataset. They are $F_1(\boldsymbol{\lambda}) = (1 - \boldsymbol{\lambda}/\lambda_{max})^5$, $F_2(\boldsymbol{\lambda}) = exp(-0.25(0.25\lambda_{max} - \boldsymbol{\lambda})^2)$, $F_3(\boldsymbol{\lambda}) = exp(-0.25(0.5\lambda_{max} - \boldsymbol{\lambda})^2)$, and $F_4(\boldsymbol{\lambda}) = exp(-0.25(0.75\lambda_{max} - \boldsymbol{\lambda})^2)$. While capturing customized frequency band of graph signals, DSGCN needs to calculate eigenvalues and eigenvectors, causing scalability issues. Furthermore, the filter functions of DSGCN are manually designed. It lacks an automatic mechanism to learn the optimal frequency profile of filters based on data.

3.5 Automatic Graph Convolution

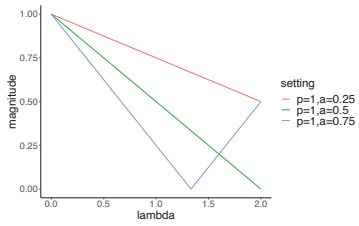
Based on spectral-rooted spatial graph convolution, I propose Automatic Graph Convolution that automatically selects low-pass, middle-pass, and high-pass filters with optimal frequency profiles. To avoid heavy computations of eigenvalues or higher orders of the graph Laplacian matrix $\mathbf{L}$, I restrict the search of filter functions within linear and quadratic forms. In the following, I first introduce the proposed low-pass, high-pass, and middle-pass filters, and then elaborate on the design of my proposed graph convolution.

My low-pass and high-pass filters are designed to be a linear function of $\boldsymbol{\lambda}$. I introduce two adjustable parameters to control the magnitude and the cut-off frequency of a filter function. I propose the **low-pass linear filter** function as

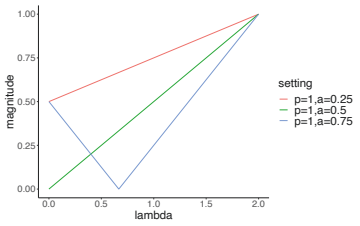
$$F_{low}(\boldsymbol{\lambda}) = p(1 - a\boldsymbol{\lambda}), \tag{3.6}$$

where the parameter $p > 0$ controls the magnitude of the frequency profile and the parameter $a \in (0,1)$ determines the cut-off frequency. The reason I constrain the scope of the parameter a within $a \in (0,1)$ is based on the assumption that $\lambda_{max} = 2$, , as the upper bound of eigenvalues is 2. When a approaches to 1, at least half of the lower spectrum can be retained. If a approaches to 0, the slope of the low-pass filter function will become flatter in a decreasing manner. In Figure 3.2a, I give examples of the frequency profiles of low-pass linear filters with three different settings. Inserting Equation 3.6 into Equation 3.2, the graph convolutional kernel of the low-pass linear filter is derived as

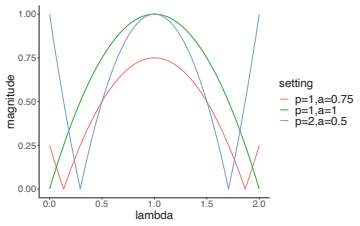
$$\mathbf{C}_{low}(p,a) = p(a\tilde{\mathbf{A}} + (1-a)\mathbf{I}) \tag{3.7}$$



(a) The frequency profiles of the low-pass filters.



(b) The frequency profiles of the high-pass filters.



(c) The frequency profiles of the middle-pass filters.

Figure 3.2: Frequency profiles of low-pass, high-pass, and middle-pass filters with three different settings. The horizontal axis is λ, ranging from 0 to 2. The vertical axis is the magnitude, which is the absolute value of the filter function.

where $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$. The derivation process can be found in Appendix B. According to Equation 3.7, the low-pass linear filter essentially aggregates a node's self-information with its neighborhood information. It is reasonable for graph data which follows the homophily assumption that connected nodes share similar features. From a spatial perspective, the parameter p controls the contribution weight of a low-pass filter, and a adjusts the confidence level of the homophily assumption.

Different from a low-pass filter, the spectrum of a high-pass filter should be retained in a increasing manner. I propose the **high-pass linear filter** function as

$$F_{high}(\boldsymbol{\lambda}) = p(a\boldsymbol{\lambda} + 1 - 2a), \tag{3.8}$$

where $p > 0$ and $a \in (0,1)$. We can easily derive that when $\lambda = 2$, the filter function achieves the highest magnitude p and when λ decreases to $2 - 1/a$, it reaches 0. In Figure 3.2b, I give examples of the frequency profiles of high-pass linear filters with three different settings. Placing Equation 3.8 into Equation 3.2, the graph convolutional kernel of the proposed high-pass linear filter is derived as

$$\mathbf{C}_{high}(p,a) = p(-a\tilde{\mathbf{A}} + (1-a)\mathbf{I}). \tag{3.9}$$

I provide the details of derivation in Appendix B. From Equation 3.9, the high-pass filter computes the difference between the self-information and neighborhood information. It highlights the features of a node that are distinct from its neighbors.

The middle-pass filter cuts off frequency values at low and high end. Due to linear functions are incapable of capturing this property, I assume the middle-pass filter function has a quadratic form. I propose the **middle-pass quadratic** filter function as

$$F_{mid}(\boldsymbol{\lambda}) = p((\boldsymbol{\lambda} - 1)^2 - a), \tag{3.10}$$

where $p > 0$ and $a \in (0, 1]$. The proposed middle-pass filter cuts off frequency at $\lambda = 1 \pm \sqrt{a}$, and reaches its maximum at $\lambda = 1$. Figure 3.2c shows examples of the frequency profiles of the middle-pass filters with three different settings. Taking Equation 3.10 into Equation 3.2, the graph convolutional kernel of the proposed middle-pass filter is derived as,

$$\mathbf{C}_{mid}(p, a) = p(\tilde{\mathbf{A}}^2 - a\mathbf{I}). \tag{3.11}$$

Viewing from a spatial perspective, my middle-pass filter can be interpreted as differentiating a node's self-information from its two-hop neighborhood information.

Theorem 1. *Assume a set of base low-pass linear filter functions with* $F_i(\boldsymbol{\lambda}) = 1 - a_i\boldsymbol{\lambda}$ $(i = 1, 2, \cdots, K)$, $a_i \in (0, 1)$*, the linear combination of the set of base low-pass linear filter functions,* $F(\boldsymbol{\lambda}) = \sum p_i F_i(\boldsymbol{\lambda})$*, with* $p_i > 0$*, is a low-pass linear filter with* $\tilde{p} = \sum p_i$ *and* $\tilde{a} = \sum p_i a_i / \sum p_i$.

Proof.

$$F(\boldsymbol{\lambda}) = \sum p_i F_i(\boldsymbol{\lambda}) \tag{3.12}$$

$$= \sum p_i (1 - a_i \boldsymbol{\lambda}) \tag{3.13}$$

$$= \sum p_i \times (1 - \frac{\sum p_i a_i}{\sum p_i} \boldsymbol{\lambda}) \tag{3.14}$$

Let $\tilde{p} = \sum p_i$, and $\tilde{a} = \sum p_i a_i / \sum p_i$,

$$F(\boldsymbol{\lambda}) = \tilde{p}(1 - \tilde{a}\boldsymbol{\lambda}) \tag{3.15}$$

I now prove $\tilde{p} > 0$, and $\tilde{a} \in (0, 1)$.

$$\tilde{p} = \sum p_i > \min(p_i) > 0 \tag{3.16}$$

As $p_i > 0$,

$$\tilde{a} = \sum p_i a_i / \sum p_i \geq \sum p_i \min(a_i) / \sum p_i = min(a_i) > 0 \tag{3.17}$$

$$\tilde{a} = \sum p_i a_i / \sum p_i \leq \sum p_i \max(a_i) / \sum p_i = max(a_i) < 1 \tag{3.18}$$

Therefore, $0 < \tilde{a} < 1$.

With $p_i > 0$ and $0 < \tilde{a} < 1$, Equation 3.15 fulfills the defintion of my proposed low-pass linear filter.

■

Theorem 2. *Assume a set of base high-pass linear filter functions with* $F_i(\boldsymbol{\lambda}) = a_i\boldsymbol{\lambda} + 1 - 2a_i$ $(i = 1, 2, \cdots, K)$, $a_i \in (0, 1)$*, the linear combination of the set of base high-pass linear filter functions,* $F(\boldsymbol{\lambda}) = \sum_i p_i F_i(\boldsymbol{\lambda})$*, with* $p_i > 0$*, is a high-pass linear filter with* $\tilde{p} = \sum p_i$ *and* $\tilde{a} = \sum p_i a_i / \sum p_i$.

Proof.

$$F(\boldsymbol{\lambda}) = \sum p_i F_i(\boldsymbol{\lambda}) \tag{3.19}$$

$$= \sum p_i (a_i \boldsymbol{\lambda} + 1 - 2a_i) \tag{3.20}$$

$$= (\sum p_i) \times (\frac{\sum p_i a_i}{\sum p_i} \boldsymbol{\lambda} + 1 - 2\frac{\sum p_i a_i}{\sum p_i}) \tag{3.21}$$

Let $\tilde{p} = \sum p_i$, and $\tilde{a} = \sum p_i a_i / \sum p_i$,

$$F(\boldsymbol{\lambda}) = \tilde{p}(\tilde{a}\boldsymbol{\lambda} + 1 - 2\tilde{a}) \tag{3.22}$$

As $\tilde{p} > 0$ and $\tilde{a} \in (0, 1)$, Equation 3.22 fulfills the defintion of my proposed high-pass filter. ■

Theorem 3. *Assume a set of base middle-pass quadratic filter functions with* $F_i(\boldsymbol{\lambda}) = (\boldsymbol{\lambda} - 1)^2 - a_i$ $(i = 1, 2, \cdots, K)$, $a_i \in (0, 1]$*, the linear combination of the set of base middle-pass quadratic filter functions,* $F(\boldsymbol{\lambda}) = \sum_i p_i F_i(\boldsymbol{\lambda})$*, with* $p_i > 0$*, is a middle-pass quadratic filter with* $\tilde{p} = \sum p_i$ *and* $\tilde{a} = \sum p_i a_i / \sum p_i$.

Proof.

$$F(\boldsymbol{\lambda}) = \sum p_i F_i(\boldsymbol{\lambda}) \tag{3.23}$$

$$= \sum p_i ((\boldsymbol{\lambda} - 1)^2 - a_i) \tag{3.24}$$

$$= (\sum p_i) \times ((\boldsymbol{\lambda} - 1)^2 - \frac{\sum p_i a_i}{\sum p_i}) \tag{3.25}$$

(3.26)

Let $\tilde{p} = \sum p_i$, and $\tilde{a} = \sum p_i a_i / \sum p_i$,

$$F(\boldsymbol{\lambda}) = \tilde{p}((\boldsymbol{\lambda} - 1)^2 - \tilde{a}) \tag{3.27}$$

As $\tilde{p} > 0$ and $\tilde{a} \in (0, 1]$, Equation 3.27 fulfills the defintion of my proposed middle-pass filter. ■

Over-parameterization of low-pass, high-pass, and middle-pass filter functions. For a single low-pass, high-pass, or a middle-pass filter, to learn the scalar parameter p and a by gradient descent is not robust. As p and a are both one-dimensional, the distance between two local optimums for these two parameters is quite small compared to points in higher-dimensional space. Therefore p and a can be easily shifted from one local optimum point to another in each learning iteration. To address this issue, I over-parameterize the proposed filter functions by a linear combination of base filter functions. According to Theorem 1, 2, and 3, the linear combination of base filter functions still lies in the definition of my proposed filter functions. Concretely, I set the over-parameterized low-pass, high-pass, and middle-pass filter functions as $F_{low}(\boldsymbol{\lambda}) = \sum_{i=1}^{K} p_i(1-a_i\boldsymbol{\lambda})$, $F_{high}(\boldsymbol{\lambda}) = \sum_{i=1}^{K} p_i(a_i\boldsymbol{\lambda}+1-2a_i)$, and $F_{mid}(\boldsymbol{\lambda}) = \sum_{i=1}^{K} p_i((\boldsymbol{\lambda}-1)^2-a_i)$, where K denotes the number of base filter functions, p_i is a learnable parameter constrained by $p_i > 0$, $\{a_i, i = 1,2,\cdots,K\}$ are set to a fixed value, which equally spaced across $(0,1)$ for low-pass and high-pass filter functions, and across $(0,1]$ for middle-pass filter functions. For example, if $K = 3$ and $a_i \in (0,1)$, then $\{a_1,a_2,a_3\}$ is set to $\{0+\epsilon, 0.5-\epsilon, 1-\epsilon\}$ with ϵ set to an infinitely small value. By setting a large K, I essentially over-parameterize and transform the proposed filter functions from learning two single parameters (p and a) to learning a set of parameters ($\{p_i, i = 1,2,\cdots,K\}$). In this way, the model generalization power is enhanced.

Complementary gating. Gating mechanisms are widely used in neural networks to control information flow and increase non-linearity. Without additional parameters, I introduce a complementary gating mechanism. The idea is to weigh the importance of one graph convolutional filter given the other twos. I assume that if one of the three graph convolutional filters contributes to the learning objective, the role of the other twos will be less important. Combining all components, I reach to my proposed graph convolution named **Automatic Graph Convolution** in its spatial form:

$$\begin{aligned}\mathbf{H}^{(l+1)} = \sigma(\mathbf{H}^{(l)}_{low} \odot \sigma(\mathbf{H}^{(l)}_{high} + \mathbf{H}^{(l)}_{mid}) + \\ \mathbf{H}^{(l)}_{high} \odot \sigma(\mathbf{H}^{(l)}_{low} + \mathbf{H}^{(l)}_{mid}) + \mathbf{H}^{(l)}_{mid} \odot \sigma(\mathbf{H}^{(l)}_{low} + \mathbf{H}^{(l)}_{high}))\end{aligned}$$

where $\odot$ denotes the elementwise product, $\mathbf{H}^{(l)}_f = \mathbf{C}^{(l)}_f \mathbf{H}^{(l)} \mathbf{W}^{(l)}_f$, $\mathbf{C}^{(l)}_f = \mathbf{U} diag(F_f(\boldsymbol{\lambda}))\mathbf{U}^T$, and $f \in \{low, mid, high\}$. To form a graph convolutional network, the automatic graph convolution can be performed multiple times attached with an output layer in the end. For node prediction tasks, the output layer can be an MLP (multi-layer perceptron). For graph prediction tasks, the output layer can be a sum/mean operation to read out graph representations, followed by an MLP layer.

Scalability analysis. As AutoGCN realizes a simple form in spatial domain without eigendecomposition, the computation complexity of AutoGCN is with the same magnitude as GCN [1], i.e. $O(M)$, where M is the number of edges. Concretely, the computation complexity of the low-pass, middle-pass, high-pass filters of AutoGCN is $O(M)$,$O(2M)$,$O(M)$ respectively. Therefore, the overall computation complexity of AutoGCN is $O(4M)$.

Overall, the benefit of AutoGCN can be understood in three aspects. First, AutoGCN captures the full spectrum of graph signals with a minimal set of graph convolutional filters, namely a low-pass filter, a high-pass filter, and a middle-pass filter. This enhances model expressivity compared to GCNs which only contain a low-pass filter. Second, AutoGCN is able to adjust the bandwidth and magnitude of its filter functions adaptively based on data. It could be extremely helpful when the distribution of data lies far from my prior knowledge. Third, AutoGCN achieves a simple form in the spatial domain. The low-pass filter can be considered as smoothing a node's self-information with its neighborhood information, the high-pass filter can be regarded as obtaining the difference between a node's self-information and its neighborhood information, and the middle-pass filter can be taken as distinguishing a node's self-information from its 2nd-hop neighborhood information. All three filters enrich the feature representations of nodes in a different way.

3.6 Experiments

I implement AutoGCN using PyTorch within a graph neural network benchmarking framework [206] based on DGL [137]. As pointed out by Dwivedi et al. [206], previous popular graph datasets such as Cora and Tu datasets are small and more likely to be overfitted, making it hard to identify the contribution of new methods. In this chapter, I choose medium-scale graph datasets. To validate the effectiveness of my AutoGCN, I test its performance on node and graph prediction tasks.

3.6.1 Node Classification

Datasets. I use six datasets: PUBMED, SBM-PATTERN, SBM-CLUSTER, Arxiv-year, YelpChi and Squirrel. PUBMED is a citation network consisting of 19,717 documents with 7 class labels. The feature of each document is a bag-of-words representation of dimension 500. I adopt the preprocessed version of PUBMED from DGL [137]. SBM-PATTERN and SBM-CLUSTER are synthesized graph datasets [206]. They are gener-

Table 3.1: Summary of node classification datasets. The number of training graphs, validiation graphs, and test graphs for PUBMED, Arxiv-year, YelphChi, and Squirrel are missing because they only contain a single graph.

	# Nodes	# Edges	# Features	# Class	# Train Graphs	# Val Graphs	# Test Graphs
PUBMED	19717	44338	500	7	-	-	-
SBM-PATTERN	50-180	4749 on average	3	2	10000	2000	2000
SBM-CLUSTER	40-190	4302 on average	7	6	10000	1000	1000
Arxiv-year	169343	1166243	128	5	-	-	-
YelpChi	45954	3846979	32	2	-	-	-
Squirrel	5201	216933	2089	5	-	-	-

ated by the stochastic block model with a probability p that two nodes are connected if they belong to the same community and a probability q that two nodes are linked if they fall into different communities. The node features of SBM-PATTERN and SBM-CLUSTER are uniformly generated from a vocabulary of $\{1,2,3\}$. The task of SBM-PATTERN is to predict whether a node belongs to a graph pattern. The task of SBM-CLUSTER is to classify nodes to their belonged clusters. For Arxiv-year, YelpChi and Squirrel, they are three non-homophilous graph datasets adopted from [207]. In a non-homophilous graph, connected nodes are not evidently more likely to share the same label. The summary statistics for these three datasets are provided in Table 3.1.

Experimental Settings. The training loss for all three datasets is the cross-entropy loss. I report the averaged accuracy on test data over 5 runs with 5 different seeds. For PUBMED, in each run, the dataset is randomly split into train, valid, and test data by 60%, 20%, 20%. For SBM-PATTERN and SBM-CLUSTER, I follow the same data split as [206]. For Arxiv-year, YelpChi, and Squirrel, I follow [207] to split the datasets into train, valid, and test data with a ratio of 2:1:1. I utilize the validation set to select the best model among all epochs in each run. For PUBMED, the default hyperparameter setting is used for each baseline model. For SBM-PATTERN, I choose 4 layers for all methods with a fixed budget of around 100k parameters. For SBM-CLUSTER, I set 8 layers for all methods with a fixed budget of around 200k parameters. For Arxiv-year, YelpChi, and Squirrel, I set 2 layers for all methods with a fixed budget of around 100k parameters. Residual connections, batch normalizations, and graph size normalizations [206] are employed for all methods on SBM-PATTERN SBM-CLUSTER, Arxiv-year, Yelp-Chi, and Squirrel. Other hyperparameter settings are provided in Table 3.5. All experiments are conducted using a single Titan XP GPU card.

Baselines. I compare AutoGCN against nine methods: MLP, ChebNet [32], GCN [1], SGC [195], DSGCN [194], APGCN [208], GIN [199], GraphSage [52], GAT [209], and MoNet [53]. I use the DGL built-in implementations of graph convolutional layers for

GCN, GraphSage, GAT and MoNet. For MLP and GIN, I adopt the implementations provided by Dwivedi et al. [206]. For ChebNet, I implement it with a second-order approximation and set $\lambda_{max} = 2$. For DSGCN, I use its original TensorFlow implementation. Due to DSGCN does not provide a general rule to design customized graph convolutional filters for new datasets, I only test DSGCN on the PUBMED dataset with its default setting.

Results & Discussion. Table 3.2 shows the experimental results for node classification tasks. On all datasets except SBM-PATTERN, AutoGCN significantly outperforms baseline methods that only work as a low-pass filter including GCN, SGC, GraphSage, and GAT. I observe that the experimental result of APGCN on SBM-PATTERN outperforms other baseline method. However, the performance of APGCN is not robust across all datasets. Besides, I observe that DSGCN does not perform well as expected on the PUBMED dataset. The same phenomenon also applies to GAT. In its original chapter, GAT evidently outperforms GCN on PUBMED. However, my experiment shows a different conclusion. This is partly due to the original data split of PUBMED used by GAT and DSGCN is not big enough to evaluate graph neural network models. The original data split consists of 140 nodes for training, 500 nodes for validation, and 1000 nodes for testing [1]. Therefore, it may easily drive the design of graph neural networks to overfit the small dataset.

3.6.2 Graph Prediction

Datasets. I use three medium-scale datasets for graph prediction [206]: ZINC, MNIST, and CIFAR10. ZINC is a molecular graph dataset. The task is to regress a molecular property called the constrained solubility [210]. For each molecular graph, node features are the type of atoms. MNIST and CIFAR10 are image classification datasets from computer vision. They are converted from images to graphs by representing nodes using super-pixels and constructing edges using k-nearest neighbors. The summary of statistics for these three datasets is provided in Table 3.3.

Experimental Settings. The training loss and the evaluation metric for ZINC data is the mean absolute error (MAE). For MNIST and CIFAR10, the training objective is the cross-entropy loss, and the evaluation metric is the accuracy score. I report the averaged evaluation metrics over test data over 5 runs with 5 different seeds. For all three datasets, I follow the same data split as [206]. I utilize the validation set to select the best model among all epochs in each run. The same output layer is employed for all baseline methods. It reads out the graph representation by taking the average over

Table 3.2: Performance on node classification tasks. Results on test sets are averaged over 5 runs with 5 different seeds. OOM stands for out of memory.

Model	**PUBMED**			**SBM-PATTERN**			**SBM-CLUSTER**		
	ACC	**#Param**	**s/epoch**	**ACC**	**#Param**	**s/epoch**	**ACC**	**#Param**	**s/epoch**
MLP	0.493 ± 0.164	9019	0.0074	0.505 ± 0.000	108112	7.6688	0.223 ± 0.003	201900	10.0579
GIN [199]	0.871 ± 0.004	16161	0.0097	0.863 ± 0.000	101764	17.7936	0.663 ± 0.010	207412	16.7895
GraphSage [52]	0.868 ± 0.006	16134	0.0126	0.663 ± 0.002	106563	12.6709	0.666 ± 0.018	205675	15.1923
GAT [209]	0.856 ± 0.005	32460	0.0285	0.780 ± 0.004	109936	29.5291	0.675 ± 0.006	205548	39.0724
MoNet [53]	0.847 ± 0.006	9903	3.6768	0.864 ± 0.000	104135	856.5839	0.663 ± 0.009	203299	803.4355
ChebNet [32]	0.886 ± 0.006	24201	0.0226	0.857 ± 0.000	103703	15.7381	0.734 ± 0.004	202435	15.4712
GCN [1]	0.866 ± 0.005	8105	0.0112	0.855 ± 0.000	106463	12.4331	0.645 ± 0.007	199015	14.7601
SGC [195]	0.866 ± 0.004	8067	0.0049	0.835 ± 0.001	108112	7.2495	0.475 ± 0.001	201900	5.7526
APGCN [208]	0.851 ± 0.005	32263	0.0447	**1.000 ± 0.000**	106124	7.7181	0.345 ± 0.002	202868	13.0703
DSGCN [194]	0.853 ± 0.007	9080	19.7565	-	-	-	-	-	-
AutoGCN	**0.893 ± 0.005**	24297	0.0372	0.859 ± 0.000	103895	20.7730	**0.741 ± 0.002**	202819	20.3951
Model	**Arxiv-year**			**YelpChi**			**Squirrel**		
	ACC	**#Param**	**s/epoch**	**ACC**	**#Param**	**s/epoch**	**ACC**	**#Param**	**s/epoch**
MLP	0.346 ± 0.001	107695	0.0525	0.854 ± 0.006	102062	0.0128	0.195 ± 0.006	104213	0.0041
GIN [199]	0.440 ± 0.003	104578	0.1200	0.855 ± 0.005	105013	0.0553	0.265 ± 0.024	101908	0.0111
GraphSage [52]	0.446 ± 0.002	103710	0.0984	0.854 ± 0.006	103324	0.0757	0.330 ± 0.018	101770	0.0118
GAT [209]	0.297 ± 0.017	101652	0.2460	OOM	OOM	OOM	0.272 ± 0.008	103220	0.0159
MoNet [53]	0.408 ± 0.016	103522	28.8757	OOM	OOM	OOM	0.277 ± 0.053	107921	5.2633
ChebNet [32]	0.483 ± 0.001	101007	0.1198	0.880 ± 0.004	105066	0.1189	0.297 ± 0.006	101391	0.0251
GCN [1]	0.452 ± 0.001	101135	0.0784	0.854 ± 0.005	102006	0.0569	0.273 ± 0.013	103119	0.0069
SGC [195]	0.399 ± 0.001	109355	0.1060	0.854 ± 0.005	100802	0.0826	0.253 ± 0.009	102917	0.0047
APGCN [208]	0.325 ± 0.006	100107	0.1137	0.854 ± 0.006	100805	0.1515	0.195 ± 0.013	102923	0.0260
DSGCN [194]	-	-	-	-	-	-	-	-	-
AutoGCN	**0.485 ± 0.001**	101151	0.1761	**0.888 ± 0.006**	105210	0.1504	**0.337 ± 0.026**	101607	0.0364

Table 3.3: Summary of graph prediction datasets.

	# Train Graphs	# Val Graphs	# Test Graphs	# Nodes	# Features	# Class
Zinc	10000	1000	100	9-37	28	-
MNIST	55000	5000	10000	40-75	3	10
CIFAR10	45000	5000	10000	85-150	5	10

Table 3.4: Performance on graph prediction tasks. Results on test sets are averaged over 5 runs with 5 different seeds.

Model	ZINC			MINST			CIFAR10		
	MAE	**#Param**	**s/epoch**	**ACC**	**#Param**	**s/epoch**	**ACC**	**#Param**	**s/epoch**
MLP	0.713 ± 0.001	204897	1.6209	0.951 ± 0.003	202383	36.6067	0.536 ± 0.007	202707	48.5513
GIN [199]	0.249 ± 0.008	204727	5.3311	0.974 ± 0.001	211078	59.0530	0.651 ± 0.005	211298	74.2930
GraphSage [52]	0.378 ± 0.015	207845	3.5253	0.977 ± 0.001	205457	45.2319	0.669 ± 0.002	205677	57.7408
GAT [209]	0.424 ± 0.006	208545	5.6672	0.966 ± 0.001	205248	70.3261	0.632 ± 0.008	205552	92.0455
MoNet [53]	0.340 ± 0.008	205074	8.9661	0.942 ± 0.004	203121	655.3304	0.623 ± 0.003	203301	926.6127
ChebNet [32]	0.254 ± 0.008	204210	4.4090	0.974 ± 0.002	202257	42.5235	0.677 ± 0.004	202437	63.5918
GCN [1]	0.314 ± 0.009	201975	3.2912	0.911 ± 0.002	198717	47.9303	0.551 ± 0.004	199017	57.0702
SGC [195]	0.725 ± 0.005	199575	1.0961	0.956 ± 0.002	196317	21.4136	0.542 ± 0.003	196617	21.8775
APGCN [208]	0.819 ± 0.004	199726	2.2744	0.958 ± 0.002	204766	39.6259	0.523 ± 0.009	205056	67.6515
AutoGCN	**0.225 ± 0.006**	204594	4.8771	**0.978 ± 0.001**	202641	59.3858	**0.678 ± 0.002**	202821	77.5203

all node features from the last layer and passes it to an MLP. For all three datasets, I set 8 layers for all methods with a fixed budget of around 200k parameters. Residual connections, bach normalizations, and graph size normalizations are employed for all methods on all three datasets. Other hyperparameter settings can be found in Table 3.5. All experiments are conducted using a single Titan XP GPU card.

Results & Discussion. Table 3.4 shows the experimental results of graph classification tasks. According to Table 3.4, AutoGCN outperforms baseline methods on all three datasets. The effectiveness of the introduction of automatic graph convolutional filter functions can be inferred from the comparison between ChebNet and AutoGCN. In fact, the graph convolutional kernels of both methods contain a zeroth-order, first-order, and second-order of the graph Laplacian matrix. Correspondingly, ChebNet consists of an all-pass filter, a low-high-pass filter, and a middle-pass filter. The main difference lies in that AutoGCN adjusts frequency profiles of graph convolutional filters adaptively according to data inputs. Though the performance difference between AutoGCN and ChebNet is smaller than the gap between AutoGCN and other baseline methods, according to Table 3.2 and Table 3.4, AutoGCN consistently outperforms ChebNet over all datasets, showing the effectiveness of my proposed method.

Baselines. The choice of baseline methods is the same as that in the node classification task.

Table 3.5: Hyperparameter settings for all experiments. L is the number of layers; *hidden* is the dimension of hidden features; *init lr* is the initial learning rate, *patience* is the decay patience, *min lr* is the stopping lr, *weight decay* is the weight decay rate, all experiments have *lr reduce factor* 0.5.

Dataset	Model	Hyperparameters				Learning Setup			
		L	hidden	dropout	Other	init lr	patience	min lr	weight decay
PUBMED	MLP	4	16	0.5	-	1e-2	-	-	5e-4
	GIN	1	16	0.5	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum	1e-2	-	-	5e-4
	GraphSage	1	16	0.5	sage_aggregator:mean	1e-2	-	-	5e-4
	GAT	1	8	0.6	n_heads:8;self_loop:true	1e-2	-	-	5e-4
	MoNet	1	16	0.5	kernel:3; pseudo_dim_MoNet:2	1e-2	-	-	5e-4
	ChebNet	1	16	0.5	K:2	1e-2	-	-	5e-4
	GCN	1	16	0.5	-	1e-2	-	-	5e-4
	DSGCN	1	16	0.25	-	1e-2	-	-	5e-4
	SGC	1	16	0.5	-	1e-2	-	-	5e-4
	APGCN	1	64	0.5	n_iter:10; prop_penalty: 0.05;	1e-2	-	-	5e-4
	AutoGCN	1	16	0.5	K:16	1e-2	-	-	5e-4
SBM-PATTERN	MLP	4	152	0	-	1e-3	5	1e-5	0
	GIN	4	110	0	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum	1e-3	5	1e-5	0
	GraphSage	4	110	0	sage_aggregator:mean	1e-3	5	1e-5	0
	GAT	4	19	0	n_heads:8;self_loop:true	1e-3	5	1e-5	0
	MoNet	4	90	0	kernel:3; pseudo_dim_MoNet:2	1e-3	5	1e-5	0
	ChebNet	4	90	0	K:2	1e-3	5	1e-5	0
	GCN	4	150	0	-	1e-3	5	1e-5	0
	SGC	4	152	0	-	1e-2	5	1e-5	0
	APGCN	4	170	0	n_iter:10; prop_penalty: 0.05;	1e-2	5	1e-5	0
	AutoGCN	4	90	0	K:16	1e-3	5	1e-5	0
SBM-CLUSTER	MLP	8	152	0	-	1e-3	5	1e-5	0
	GIN	8	110	0	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum	1e-3	5	1e-5	0
	GraphSage	8	110	0	sage_aggregator:mean	1e-3	5	1e-5	0
	GAT	8	19	0	n_heads:8;self_loop:true	1e-3	5	1e-5	0
	MoNet	8	90	0	kernel:3; pseudo_dim_MoNet:2	1e-3	5	1e-5	0
	ChebNet	8	90	0	K:2	1e-3	5	1e-5	0
	GCN	8	150	0	-	1e-3	5	1e-5	0
	SGC	8	152	0	-	1e-2	5	1e-5	0
	APGCN	8	162	0	n_iter:10; prop_penalty: 0.05;	1e-2	5	1e-5	0
	AutoGCN	8	90	0	K:16	1e-3	5	1e-5	0
Arxiv-year	MLP	2	220	0.5	-	1e-2	-	-	5e-4
	GIN	2	170	0.5	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum	1e-2	-	-	5e-4
	GraphSage	2	170	0.5	sage_aggregator:mean	1e-2	-	-	5e-4
	GAT	2	32	0.5	n_heads:8;self_loop:true	1e-2	-	-	5e-4
	MoNet	2	115	0.5	kernel:3; pseudo_dim_MoNet:2	1e-2	-	-	5e-4
	ChebNet	2	128	0.5	K:2	1e-2	-	-	5e-4
	GCN	2	256	0.5	-	1e-2	-	-	5e-4
	SGC	2	270	0.5	-	1e-2	-	-	5e-4
	APGCN	2	256	0.5	n_iter:10; prop_penalty: 0.05;	1e-2	-	-	5e-4
	AutoGCN	2	128	0.5	K:16	1e-2	-	-	5e-4
Yelp-chi	MLP	2	240	0.5	-	1e-2	-	-	5e-4
	GIN	2	210	0.5	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum	1e-2	-	-	5e-4
	GraphSage	2	210	0.5	sage_aggregator:mean	1e-2	-	-	5e-4
	GAT	2	32	0.5	n_heads:8;self_loop:true	1e-2	-	-	5e-4
	MoNet	2	115	0.5	kernel:3; pseudo_dim_MoNet:2	1e-2	-	-	5e-4
	ChebNet	2	170	0.5	K:2	1e-2	-	-	5e-4
	GCN	2	300	0.5	-	1e-2	-	-	5e-4
	SGC	2	300	0.5	-	1e-2	-	-	5e-4
	APGCN	2	300	0.5	n_iter:10; prop_penalty: 0.05;	1e-2	-	-	5e-4
	AutoGCN	2	165	0.5	K:16	1e-2	-	-	5e-4
Squirrel	MLP	2	48	0.5	-	1e-2	-	-	5e-4
	GIN	2	24	0.5	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum	1e-2	-	-	5e-4
	GraphSage	2	24	0.5	sage_aggregator:mean	1e-2	-	-	5e-4
	GAT	2	6	0.5	n_heads:8;self_loop:true	1e-2	-	-	5e-4
	MoNet	2	45	0.5	kernel:3; pseudo_dim_MoNet:2	1e-2	-	-	5e-4
	ChebNet	2	16	0.5	K:2	1e-2	-	-	5e-4
	GCN	2	48	0.5	-	1e-2	-	-	5e-4
	SGC	2	48	0.5	-	1e-2	-	-	5e-4
	APGCN	2	48	0.5	n_iter:10; prop_penalty: 0.05;	1e-2	-	-	5e-4
	AutoGCN	2	13	0.5	K:16	1e-2	-	-	5e-4
ZINC	MLP	8	152	0	-	1e-3	10	1e-5	0
	GIN	8	110	0	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum	1e-3	10	1e-5	0
	GraphSage	8	110	0	sage_aggregator:mean	1e-3	10	1e-5	0
	GAT	8	19	0	n_heads:8;self_loop:true	1e-3	10	1e-5	0
	MoNet	8	90	0	kernel:3; pseudo_dim_MoNet:2	1e-3	10	1e-5	0
	ChebNet	8	90	0	K:2	1e-3	10	1e-5	0
	GCN	8	150	0	-	1e-3	10	1e-5	0
	SGC	8	150	0	-	1e-2	10	1e-5	0
	APGCN	8	150	0	n_iter:10; prop_penalty: 0.05;	1e-2	10	1e-5	0
	AutoGCN	8	90	0	K:16	1e-3	10	1e-5	0
MNIST	MLP	8	162	0.1	-	1e-3	5	1e-5	0
	GIN	8	110	0.1	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum	1e-3	5	1e-5	0
	GraphSage	8	110	0.1	sage_aggregator:mean	1e-3	5	1e-5	0
	GAT	8	19	0.1	n_heads:8;self_loop:true	1e-3	5	1e-5	0
	MoNet	8	90	0.1	kernel:3; pseudo_dim_MoNet:2	1e-3	5	1e-5	0
	ChebNet	8	90	0.1	K:2	1e-3	5	1e-5	0
	GCN	8	150	0.1	-	1e-3	5	1e-5	0
	SGC	8	150	0.1	-	1e-2	5	1e-5	0
	APGCN	8	256	0.1	n_iter:10; prop_penalty: 0.05;	1e-2	5	1e-5	0
	AutoGCN	8	90	0.1	K:16	1e-3	5	1e-5	0
CIFAR10	MLP	8	162	0.1	-	1e-3	5	1e-5	0
	GIN	8	110	0.1	n_mlp_GIN:2; learn_eps_GIN:True; neighbor_aggr_GIN:sum	1e-3	5	1e-5	0
	GraphSage	8	110	0.1	sage_aggregator:mean	1e-3	5	1e-5	0
	GAT	8	19	0.1	n_heads:8;self_loop:true	1e-3	5	1e-5	0
	MoNet	8	90	0.1	kernel:3; pseudo_dim_MoNet:2	1e-3	5	1e-5	0
	ChebNet	8	90	0.1	K:2	1e-3	5	1e-5	0
	GCN	8	150	0.1	-	1e-3	5	1e-5	0
	SGC	8	150	0.1	-	1e-2	5	1e-5	0
	APGCN	8	256	0.1	n_iter:10; prop_penalty: 0.05;	1e-2	5	1e-5	0
	AutoGCN	8	90	0.1	K:16	1e-3	5	1e-5	0

Table 3.6: Ablation study.

Data	AutoGCN	w/o low	w/o high	w/o middle	w/o over	w/o par	w/o gate
SBM-CLUSTER	**74.104 ± 0.151**	72.168 ± 0.232	73.748 ± 0.181	71.924 ± 0.329	73.559 ± 0.283	73.450 ± 0.438	73.991 ± 0.206
ZINC	**0.225 ± 0.009**	0.253 ± 0.011	0.253 ± 0.010	0.242 ± 0.015	0.233 ± 0.003	0.231 ± 0.007	0.248 ± 0.006

3.6.3 Ablation Study

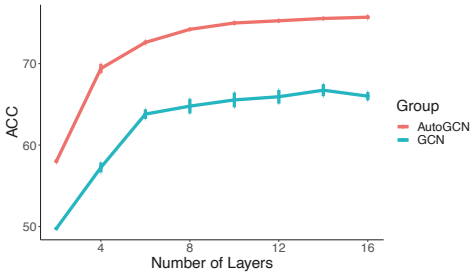
I conduct an ablation study on SBM-CLUSTER and ZINC to validate the effectiveness of key components that contribute to the improvement of my proposed model. I name AutoGCN without different components as follows - **w/o low**: AutoGCN without the low-pass linear filter; **w/o high**: AutoGCN without the high-pass linear filter; **w/o middle**: AutoGCN without the middle-pass quadratic filter; **w/o over**: AutoGCN without over-parameterizing filter functions; **w/o par**: AutoGCN without parameterizing filter functions. I set $p = 1, a = 0.5$ as fixed values for the low-pass filter, the high-pass filter, and the middle-pass filter; **w/o gate**: AutoGCN without the complementary gate. I simply sum up the outputs of three graph convolutional filters in each layer. For **w/o low**, **w/o high**, and **w/o middle**, I increase the hidden feature dimension to keep approximately the same number of parameters as AutoGCN. I run each experiment five times and report the averaged evaluation metrics with standard deviation on test data. Table 3.6 reports the experimental results of the ablation study. It shows that any absence of a key component reduces the performance of AutoGCN.

3.6.4 Limitation

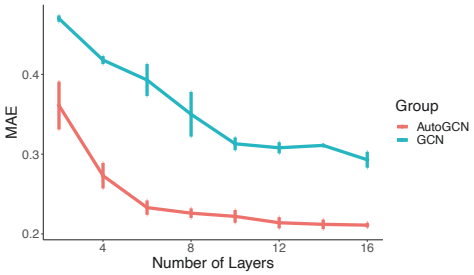
While being adaptive to frequency bandwidth, the flexibility of AutoGCN in adjusting filtering functions is limited. The linear and quadratic functions of AutoGCN are only approximating low-pass, mid-pass and high-pass filters. On the one side, each type of filter functions still contain other frequency bands which are not desired. For example, the designed low-pass frequency functions also has a small portion of mid-frequency bands and high-frequency bands. On the other side, if we choose to cut-off the frequency for the corresponding filters, the model cannot be simplified to first-order and second-order of the adjacency matrix, and thus such alternative model is no longer scalable.

3.6.5 Analysis of Model Depth.

Graph convolutional networks often face the over-smoothing problem. In theory, as the number of hidden layers goes to infinity, node features of a graph will converge to a



(a) The trend of ACC v.s model depth for the SBM-CLUSTER dataset.



(b) The trend of MAE v.s. model depth for the ZINC dataset.

Figure 3.3: Comparison analysis of model depth between AutoGCN and GCN.

fixed point [62]. A solution to mitigate the over-smoothing problem is to add residual connections to each layer. Figure 3.3 plots the performance curve of GCN and AutoGCN on SBM-CLUSTER and ZINC as the number of layers is increased from 2 to 16 every two steps. I run each experiment five times and report the averaged evaluation metrics with standard deviation on test data. With the increase in the number of layers, the performance of GCN and AutoGCN is also enhanced. It demonstrates the usefulness of residual connections in designing deep graph convolutional networks. Note that it may be argued that the significant improvement of AutoGCN over GCN is mainly attributed to the fact that each layer of AutoGCN receives a broader range of neighborhood information because of the second-order of the graph Laplacian matrix. It is suspected that comparing AutoGCN with GCN or other methods under the same number of layers may not be a fair option. However, if we look at Figure 3.3, a deeper GCN still cannot compete with a shallower AutoGCN. For example, a four-layer of GCN performs much worse than a two-layer of AutoGCN , even the depth of node information aggregation for both models is 4. This may suggest that the second-order of graph Laplacian matrix not only broadens a node's neighborhood, but also works as a middle-pass filter to filter low and high frequency of graph signals.

3.7 Summary

This chapter proposes a graph convolutional network that captures the full spectrum of graph signals with automatic filtering. My method AutoGCN consists of three different forms of graph convolutional filters: the low-pass filter, the high-pass filter, and the middle-pass filter. All three filters enrich node feature representations in a different way. In each filter, I introduce two parameters to control the magnitude and the curvature of

its frequency profile. It enables AutoGCN to update its parameterized filter functions based on data. My experiments show that AutoGCN achieves significant improvement over baseline methods that only work as low-pass filters.

CHAPTER 4

GRAPH WAVENET FOR DEEP SPATIAL-TEMPORAL GRAPH MODELING

4.1 Motivation

Spatial-temporal graph modeling has received increasing attention with the advance of graph neural networks. It aims to model the dynamic node-level inputs by assuming inter-dependency between connected nodes, as demonstrated by Figure 4.1. Spatial-temporal graph modeling has wide applications in solving complex system problems such as traffic speed forecasting [13], taxi demand prediction [14], human action recognition [15], and driver maneuver anticipation [18]. For a concrete example, in traffic speed forecasting, speed sensors on roads of a city form a graph where the edge weights are judged by two nodes' Euclidean distance. As the traffic congestion on one road could cause lower traffic speed on its incoming roads, it is natural to consider the underlying graph structure of the traffic system as the prior knowledge of inter-dependency relationships among nodes when modeling time series data of the traffic speed on each road.

A basic assumption behind spatial-temporal graph modeling is that a node's future information is conditioned on its historical information as well as its neighbors' historical information. Therefore how to capture spatial and temporal dependencies simultaneously becomes a primary challenge. Recent studies on spatial-temporal graph modeling mainly follow two directions. They either integrate graph convolution networks (GCN) into recurrent neural networks (RNN) [13, 19] or into convolution neural networks (CNN)

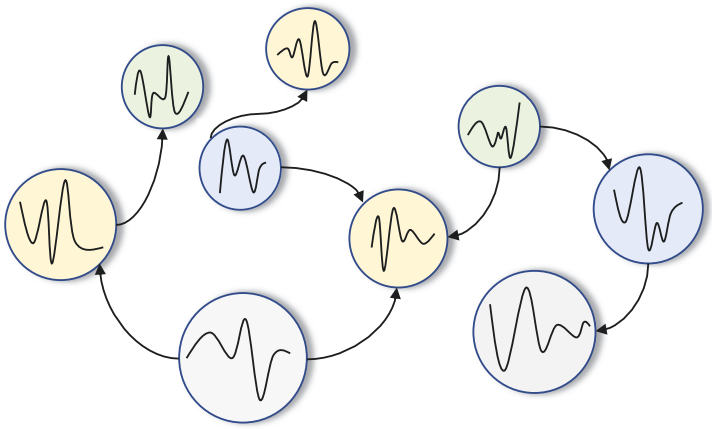


Figure 4.1: Spatial-temporal graph modeling. Each circle represents a node. Each node has dynamic input features. The aim is to model each node's dynamic features given the graph structure.

[15, 20]. While having shown the effectiveness of introducing the graph structure of data into a model, these approaches face two major shortcomings.

First, these studies assume the graph structure of data reflects the genuine dependency relationships among nodes. However, there are circumstances when a connection does not entail the inter-dependency relationship between two nodes and when the inter-dependency relationship between two nodes exists but a connection is missing. To give each circumstance an example, let us consider a recommendation system. In the first case, two users are connected, but they may have distinct preferences over products. In the second case, two users may share a similar preference, but they are not linked together. Zhang et al [57] used attention mechanisms to address the first circumstance by adjusting the dependency weight between two connected nodes, but they failed to consider the second circumstance.

Second, current studies for spatial-temporal graph modeling are ineffective to learn temporal dependencies. RNN-based approaches suffer from time-consuming iterative propagation and gradient explosion/vanishing for capturing long-range sequences [13, 19, 57]. On the contrary, CNN-based approaches enjoy the advantages of parallel computing, stable gradients and low memory requirement [15, 20]. However, these works need to use many layers in order to capture very long sequences because they adopt standard 1D convolution whose receptive field size grows linearly with an increase in the number of hidden layers.

In this work, I present a CNN-based method named Graph WaveNet, which addresses the two shortcomings I have aforementioned. I propose a graph convolution layer in

which a self-adaptive adjacency matrix can be learned from the data through end-to-end supervised training. In this way, the self-adaptive adjacency matrix preserves hidden spatial dependencies. Motivated by WaveNet [211], I adopt stacked dilated casual convolutions to capture temporal dependencies. The receptive field size of stacked dilated casual convolution networks grows exponentially with an increase in the number of hidden layers. With the support of stacked dilated casual convolutions, Graph WaveNet is able to handle spatial-temporal graph data with long-range temporal sequences efficiently and effectively. The main contributions of this work are as follows:

- I construct a self-adaptive adjacency matrix which preserves hidden spatial dependencies. My proposed self-adaptive adjacency matrix is able to uncover unseen graph structures automatically from the data without any guidance of prior knowledge. Experiments validate that my method improves the results when spatial dependencies are known to exist but are not provided.

- I present an effective and efficient framework to capture spatial-temporal dependencies simultaneously. The core idea is to assemble my proposed graph convolution with dilated casual convolution in a way that each graph convolution layer tackles spatial dependencies of nodes' information extracted by dilated casual convolution layers at different granular levels.

- I evaluate my proposed model on traffic datasets and achieve state-of-the-art results with low computation costs. The source codes of Graph WaveNet are publicly available from `https://github.com/nnzhan/Graph-WaveNet`.

4.2 Related Works to Graph WaveNet

The majority of Spatial-temporal Graph Networks follow two directions, namely, RNN-based and CNN-based approaches. One of the early RNN-based methods captured spatial-temporal dependencies by filtering inputs and hidden states passed to a recurrent unit using graph convolution [19]. Later works adopted different strategies such as diffusion convolution [13] and attention mechanisms [57] to improve model performance. Another parallel work used node-level RNNs and edge-level RNNs to handle different aspects of temporal information [18]. The main drawbacks of RNN-based approaches are that it becomes inefficient for long sequences and its gradients are more likely to explode when they are combined with graph convolution networks. CNN-based approaches combine a

graph convolution with a standard 1D convolution [15, 20]. While being computationally efficient, these two approaches have to stack many layers or use global pooling to expand the receptive field of a neural network model.

4.3 Methodology

In this section, I first give the mathematical definition of the problem I am addressing in this chapter. Next, I describe two building blocks of my framework, the graph convolution layer (GCN) and the temporal convolution layer (TCN). They work together to capture the spatial-temporal dependencies. Finally, I outline the architecture of my framework.

4.3.1 Problem Definition

A graph is represented by $G = (V, E)$ where V is the set of nodes and E is the set of edges. The adjacency matrix derived from a graph is denoted by $\mathbf{A} \in \mathbf{R}^{N \times N}$. If $v_i, v_j \in V$ and $(v_i, v_j) \in E$, then $\mathbf{A}_{ij}$ is one otherwise it is zero. At each time step t, the graph G has a dynamic feature matrix $\mathbf{X}^{(t)} \in \mathbf{R}^{N \times D}$. In this chapter, the feature matrix is used interchangeably with graph signals. Given a graph G and its historical S step graph signals, my problem is to learn a function f which is able to forecast its next T step graph signals. The mapping relation is represented as follows

$$[\mathbf{X}^{(t-S):t}, G] \xrightarrow{f} \mathbf{X}^{(t+1):(t+T)}, \tag{4.1}$$

where $\mathbf{X}^{(t-S):t} \in \mathbf{R}^{N \times D \times S}$ and $\mathbf{X}^{(t+1):(t+T)} \in \mathbf{R}^{N \times D \times T}$.

4.3.2 Graph Convolution Layer

Graph convolution is an essential operation to extract a node's features given its structural information. Kipf et al [212] proposed a first approximation of Chebyshev spectral filter [32]. From a spatial-based perspective, it smoothed a node's signal by aggregating and transforming its neighborhood information. The advantages of their method are that it is a compositional layer, its filter is localized in space, and it supports multi-dimensional inputs. Let $\tilde{\mathbf{A}} \in \mathbf{R}^{N \times N}$ denote the normalized adjacency matrix with self-loops, $\mathbf{X} \in \mathbf{R}^{N \times D}$ denote the input signals, $\mathbf{Z} \in \mathbf{R}^{N \times M}$ denote the output, and $\mathbf{W} \in \mathbf{R}^{D \times M}$ denote the model parameter matrix, in [212] the graph convolution layer is defined as

$$\mathbf{Z} = \tilde{\mathbf{A}}\mathbf{X}\mathbf{W}. \tag{4.2}$$

Li et al [13] proposed a diffusion convolution layer which proves to be effective in spatial-temporal modeling. They modeled the diffusion process of graph signals with K finite steps. I generalize its diffusion convolution layer into the form of Equation 4.2, which results in,

$$\mathbf{Z} = \sum_{k=0}^{K} \mathbf{P}^k \mathbf{X}\mathbf{W}_\mathbf{k}, \tag{4.3}$$

where $\mathbf{P}^k$ represents the power series of the transition matrix. In the case of an undirected graph, $\mathbf{P} = \mathbf{A}/rowsum(\mathbf{A})$. In the case of a directed graph, the diffusion process have two directions, the forward and backward directions, where the forward transition matrix $\mathbf{P}_f = \mathbf{A}/rowsum(\mathbf{A})$ and the backward transition matrix $\mathbf{P}_b = \mathbf{A}^\mathbf{T}/rowsum(\mathbf{A}^\mathbf{T})$. With the forward and the backward transition matrix, the diffusion graph convolution layer is written as

$$\mathbf{Z} = \sum_{k=0}^{K} \mathbf{P}_f^k \mathbf{X}\mathbf{W}_{k1} + \mathbf{P}_b^k \mathbf{X}\mathbf{W}_{k2}. \tag{4.4}$$

4.3.2.1 Self-adaptive Adjacency Matrix

In my work, I propose a self-adaptive adjacency matrix $\tilde{\mathbf{A}}_{adp}$. This self-adaptive adjacency matrix does not require any prior knowledge and is learned end-to-end through stochastic gradient descent. In doing so, I let the model discover hidden spatial dependencies by itself. I achieve this by randomly initializing two node embedding dictionaries with learnable parameters $\mathbf{E}_1, \mathbf{E}_2 \in \mathbf{R}^{N \times c}$. I propose the self-adaptive adjacency matrix as

$$\tilde{\mathbf{A}}_{adp} = SoftMax(ReLU(\mathbf{E}_1 \mathbf{E}_2^T)). \tag{4.5}$$

I name **E1** as the source node embedding and **E2** as the target node embedding. By multiplying **E1** and **E2**, I derive the spatial dependency weights between the source nodes and the target nodes. I use the ReLU activation function to eliminate weak connections. The SoftMax function is applied to normalize the self-adaptive adjacency matrix. The normalized self-adaptive adjacency matrix, therefore, can be considered as the transition matrix of a hidden diffusion process. By combining pre-defined spatial dependencies and

self-learned hidden graph dependencies, I propose the following graph convolution layer

$$\mathbf{Z} = \sum_{k=0}^{K} \mathbf{P}_f^k \mathbf{X}\mathbf{W}_{k1} + \mathbf{P}_b^k \mathbf{X}\mathbf{W}_{k2} + \tilde{\mathbf{A}}_{apt}^k \mathbf{X}\mathbf{W}_{k3}. \tag{4.6}$$

When the graph structure is unavailable, I propose to use the self-adaptive adjacency matrix alone to capture hidden spatial dependencies, i.e.,

$$\mathbf{Z} = \sum_{k=0}^{K} \tilde{\mathbf{A}}_{apt}^k \mathbf{X}\mathbf{W}_k. \tag{4.7}$$

It is worth noting that my graph convolution falls into spatial-based approaches. Although I use graph signals interchangeably with node feature matrix for consistency, my graph convolution in Equation 4.7 indeed is interpreted as aggregating transformed feature information from different orders of neighborhoods.

4.3.3 Temporal Convolution Layer

I adopt the dilated causal convolution [213] as my temporal convolution layer (TCN) to capture a node's temporal trends. Dilated causal convolution networks allow an exponentially large receptive field by increasing the layer depth. As opposed to RNN-based approaches, dilated casual convolution networks are able to handle long-range sequences properly in a non-recursive manner, which facilitates parallel computation and alleviates the gradient explosion problem. The dilated causal convolution preserves the temporal causal order by padding zeros to the inputs so that predictions made on the current time step only involve historical information. As a special case of standard 1D-convolution, the dilated causal convolution operation slides over inputs by skipping values with a certain step, as illustrated by Figure 4.2. Mathematically, given a 1D sequence input $\mathbf{x} \in \mathbf{R}^T$ and a filter $\mathbf{f} \in \mathbf{R}^K$, the dilated causal convolution operation of $\mathbf{x}$ with $\mathbf{f}$ at step t is represented as

$$\mathbf{x} \star \mathbf{f}(t) = \sum_{s=0}^{K-1} \mathbf{f}(s)\mathbf{x}(t - d \times s), \tag{4.8}$$

where d is the dilation factor which controls the skipping distance. By stacking dilated causal convolution layers with dilation factors in increasing order, the receptive field of a model grows exponentially. It enables dilated causal convolution networks to capture longer sequences with less layers, which saves computation resources.

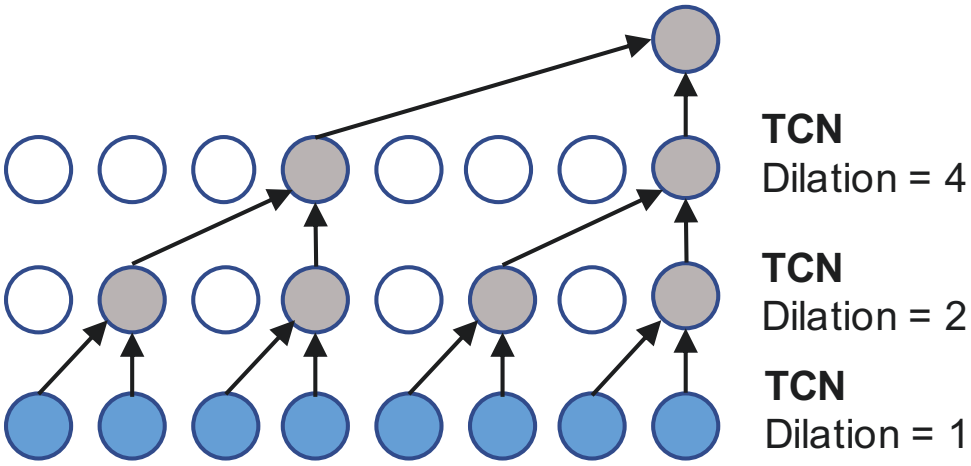


Figure 4.2: Dilated casual convolution with kernel size 2. With a dilation factor k, it picks inputs every k step and applies the standard 1D convolution to the selected inputs.

Gated TCN. Gating mechanisms are critical in recurrent neural networks. They have been shown to be powerful to control information flow through layers for temporal convolution networks as well [214]. A simple Gated TCN only contains an output gate. Given the input $\mathcal{X} \in R^{N \times D \times S}$, it takes the form

$$\mathbf{h} = g(\Theta_1 \star \mathcal{X} + \mathbf{b}) \odot \sigma(\Theta_2 \star \mathcal{X} + \mathbf{c}), \tag{4.9}$$

where Θ_1, Θ_2, $\mathbf{b}$ and $\mathbf{c}$ are model parameters, $\odot$ is the element-wise product, $g(\cdot)$ is an activation function of the outputs, and $\sigma(\cdot)$ is the sigmoid function which determines the ratio of information passed to the next layer. I adopt Gated TCN in my model to learn complex temporal dependencies. Although I empirically set the tangent hyperbolic function as the activation function $g(\cdot)$, other forms of Gated TCN can be easily fitted into my framework, such as an LSTM-like Gated TCN [215].

4.3.4 Framework of Graph WaveNet

I present the framework of Graph WaveNet in Figure 4.3. It consists of stacked spatial-temporal layers and an output layer. A spatial-temporal layer is constructed by a graph convolution layer (GCN) and a gated temporal convolution layer (Gated TCN) which consists of two parallel temporal convolution layers (TCN-a and TCN-b). By stacking multiple spatial-temporal layers, Graph WaveNet is able to handle spatial dependencies at different temporal levels. For example, at the bottom layer, GCN receives short-term

Data	#Nodes	#Edges	#Time Steps
METR-LA	207	1515	34272
PEMS-BAY	325	2369	52116

Table 4.1: Summary statistics of METR-LA and PEMS-BAY.

temporal information while at the top layer GCN tackles long-term temporal information. The inputs $\mathbf{h}$ to a graph convolution layer in practice are three-dimension tensors with size [N,C,L] where N is the number of nodes, and C is the hidden dimension, L is the sequence length. I apply the graph convolution layer to each of $\mathbf{h}[:,:,i] \in \mathbf{R}^{N \times C}$.

I choose to use mean absolute error (MAE) as the training objective of Graph WaveNet, which is defined by

$$L(\hat{\mathbf{X}}^{(t+1):(t+T)}; \Theta) = \frac{1}{TND} \sum_{i=1}^{i=T} \sum_{j=1}^{j=N} \sum_{k=1}^{k=D} |\hat{\mathbf{X}}_{jk}^{(t+i)} - \mathbf{X}_{jk}^{(t+i)}| \tag{4.10}$$

Unlike previous works such as [13, 20], my Graph WaveNet outputs $\hat{\mathbf{X}}^{(t+1):(t+T)}$ as a whole rather than generating $\hat{\mathbf{X}}^{(t)}$ recursively through T steps. It addresses the problem of inconsistency between training and testing due to the fact that a model learns to make predictions for one step during training and is expected to produce predictions for multiple steps during inference. To achieve this, I artificially design the receptive field size of Graph WaveNet equals the sequence length of the inputs so that in the last spatial-temporal layer the temporal dimension of the outputs exactly equals to one. After that I set the number of output channels of the last layer as a factor of step length T to get my desired output dimension.

4.4 Experiments

I verify Graph WaveNet on two public traffic network datasets, METR-LA and PEMS-BAY released by Li et al [13]. METR-LA records four months of statistics on traffic speed on 207 sensors on the highways of Los Angeles County. PEMS-BAY contains six months of traffic speed information on 325 sensors in the Bay area. I adopt the same data pre-processing procedures as in [13]. The readings of the sensors are aggregated into 5-minutes windows. The adjacency matrix of the nodes is constructed by road network distance with a thresholded Gaussian kernel [216]. Z-score normalization is applied to inputs. The datasets are split in chronological order with 70% for training, 10% for validation and 20% for testing. Detailed dataset statistics are provided in Table 4.1.

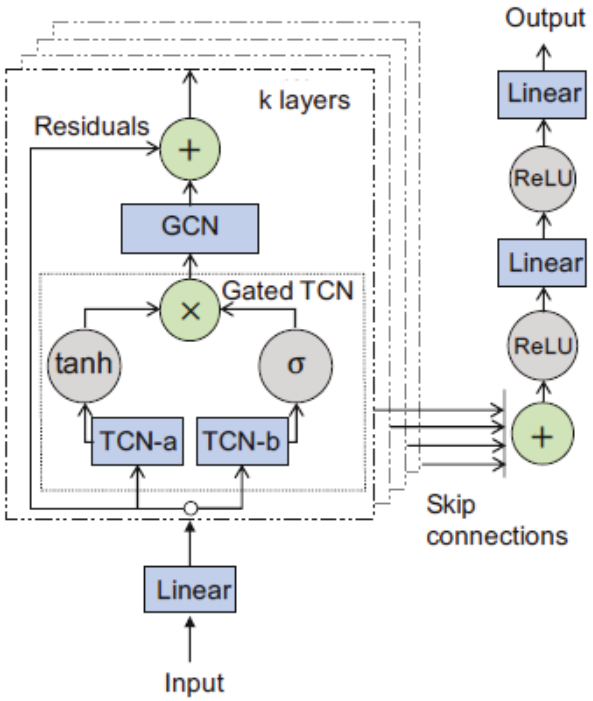


Figure 4.3: The framework of Graph WaveNet. It consists of K spatial-temporal layers on the left and an output layer on the right. The inputs are first transformed by a linear layer and then passed to the gated temporal convolution module (Gated TCN) followed by the graph convolution layer (GCN). Each spatial-temporal layer has residual connections and is skip-connected to the output layer.

4.4.1 Baselines

I compare Graph WaveNet with the following models.

- ARIMA. Auto-Regressive Integrated Moving Average model with Kalman filter [13].
- FC-LSTM Recurrent neural network with fully connected LSTM hidden units [13].
- WaveNet. A convolution network architecture for sequence data [211].
- DCRNN. Diffusion convolution recurrent neural network [13], which combines graph convolution networks with recurrent neural networks in an encoder-decoder manner.
- GGRU. Graph gated recurrent unit network [57]. Recurrent-based approaches. GGRU uses attention mechanisms in graph convolution.

Data	Models	15 min			30 min			60 min		
		MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
METR-LA	ARIMA [13]	3.99	8.21	9.60%	5.15	10.45	12.70%	6.90	13.23	17.40%
	FC-LSTM [13]	3.44	6.30	9.60%	3.77	7.23	10.90%	4.37	8.69	13.20%
	WaveNet [211]	2.99	5.89	8.04%	3.59	7.28	10.25%	4.45	8.93	13.62%
	DCRNN [13]	2.77	5.38	7.30%	3.15	6.45	8.80%	3.60	7.60	10.50%
	GGRU [57]	2.71	5.24	6.99%	3.12	6.36	8.56%	3.64	7.65	10.62%
	STGCN [20]	2.88	5.74	7.62%	3.47	7.24	9.57%	4.59	9.40	12.70%
	Graph WaveNet	**2.69**	**5.15**	**6.90**%	**3.07**	**6.22**	**8.37**%	**3.53**	**7.37**	**10.01**%
PEMS-BAY	ARIMA [13]	1.62	3.30	3.50%	2.33	4.76	5.40%	3.38	6.50	8.30%
	FC-LSTM [13]	2.05	4.19	4.80%	2.20	4.55	5.20%	2.37	4.96	5.70%
	WaveNet [211]	1.39	3.01	2.91%	1.83	4.21	4.16%	2.35	5.43	5.87%
	DCRNN [13]	1.38	2.95	2.90%	1.74	3.97	3.90%	2.07	4.74	4.90%
	GGRU [57]	-	-	-	-	-	-	-	-	-
	STGCN [20]	1.36	2.96	2.90%	1.81	4.27	4.17%	2.49	5.69	5.79%
	Graph WaveNet	**1.30**	**2.74**	**2.73**%	**1.63**	**3.70**	**3.67**%	**1.95**	**4.52**	**4.63**%

Table 4.2: Performance comparison of Graph WaveNet and other baseline models. Graph WaveNet achieves the best results on both datasets.

- STGCN. Spatial-temporal graph convolution network [20], which combines graph convolution with 1D convolution.

4.4.2 Experimental Setups

My experiments are conducted under a computer environment with one Intel(R) Core(TM) i9-7900X CPU @ 3.30GHz and one NVIDIA Titan Xp GPU card. To cover the input sequence length, I use eight layers of Graph WaveNet with a sequence of dilation factors $1, 2, 1, 2, 1, 2, 1, 2$. I use Equation 4.4 as my graph convolution layer with a diffusion step $K = 2$. I randomly initialize node embeddings by a uniform distribution with a size of 10. I train my model using Adam optimizer with an initial learning rate of 0.001. Dropout with p=0.3 is applied to the outputs of the graph convolution layer. The evaluation metrics I choose include mean absolute error (MAE), root mean squared error (RMSE), and mean absolute percentage error (MAPE). Missing values are excluded both from training and testing.

4.4.3 Experimental Results

Table 4.2 compares the performance of Graph WaveNet and baseline models for 15 minutes, 30 minutes and 60 minutes ahead prediction on METR-LA and PEMS-BAY datasets. Graph WaveNet obtains superior results on both datasets. It outperforms tem-

Dataset	Model Name	Adjacency Matrix Configuration	Mean MAE	Mean RMSE	Mean MAPE
METR-LR	Identity	$[\mathbf{I}]$	3.58	7.18	10.21%
	Forward-only	$[\mathbf{P}]$	3.13	6.26	8.65%
	Adaptive-only	$[\tilde{\mathbf{A}}_{adp}]$	3.10	6.21	8.68%
	Forward-backward	$[\mathbf{P}_f, \mathbf{P}_b]$	3.08	6.13	8.25%
	Forward-backward-adaptive	$[\mathbf{P}_f, \mathbf{P}_b, \tilde{\mathbf{A}}_{adp}]$	**3.04**	**6.09**	**8.23%**
PEMS-BAY	Identity	$[\mathbf{I}]$	1.80	4.05	4.18%
	Forward-only	$[\mathbf{P}_f]$	1.62	3.61	3.72%
	Adaptive-only	$[\tilde{\mathbf{A}}_{adp}]$	1.61	3.63	3.59%
	Forward-backward	$[\mathbf{P}_f, \mathbf{P}_b]$	1.59	3.55	3.57%
	Forward-backward-adaptive	$[\mathbf{P}_f, \mathbf{P}_b, \tilde{\mathbf{A}}_{adp}]$	**1.58**	**3.52**	**3.55%**

Table 4.3: Experimental results of different adjacency matrix configurations. The forward-backward-adaptive model achieves the best results on both datasets. The adaptive-only model achieves nearly the same performance with the forward-only model.

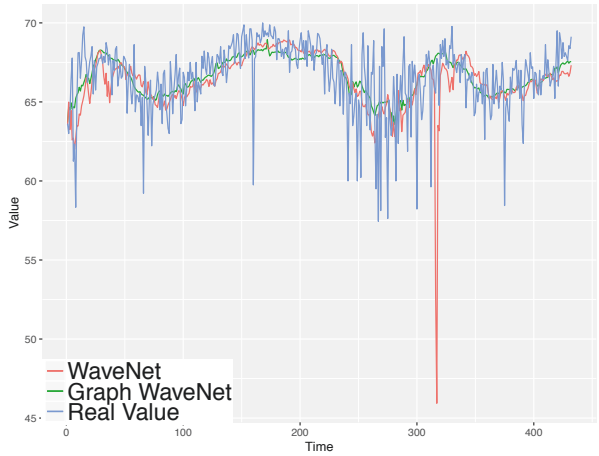
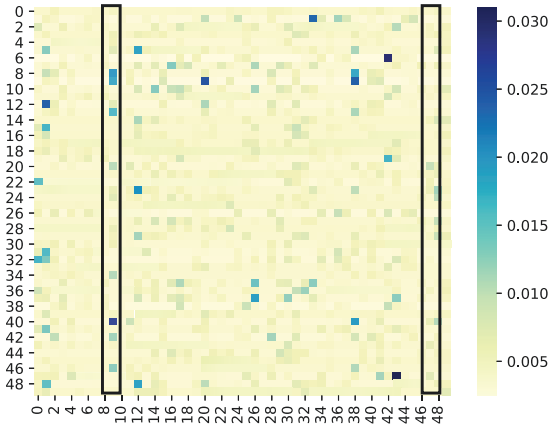


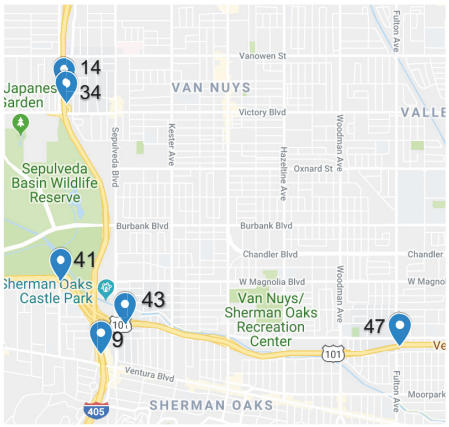
Figure 4.4: Comparison of prediction curves between WaveNet and Graph WaveNet for 60 minutes ahead prediction on a snapshot of the test data of METR-LA.

poral models including ARIMA, FC-LSTM, and WaveNet by a large margin. Compared to other spatial-temporal models, Graph WaveNet surpasses the previous convolution-based approach STGCN significantly and excels recurrent-based approaches DCRNN and GGRU at the same time. In respect of the second-best model GGRU, Graph WaveNet achieves small improvement over GGRU on the 15-minute horizons; however, realizes a bigger enhancement on the 60-minute horizons. My architecture is more capable of detecting spatial dependencies at each temporal stage. GGRU uses recurrent architectures in which parameters of the GCN layer are shared across all recurrent units. In contrast, Graph WaveNet employs stacked spatial-temporal layers which contain separate GCN layers with different parameters. Therefore each GCN layer in Graph WaveNet is able to focus on its own range of temporal inputs.

I plot 60-minutes-ahead predicted values v.s real values of Graph WaveNet and



(a) The heatmap of the learned self-adaptive adjacency matrix for the first 50 nodes.



(b) The geographical location of a part of nodes marked on Google Maps.

Figure 4.5: The learned self-adaptive adjacency matrix.

WaveNet on a snapshot of the test data in Figure 4.4. It shows that Graph WaveNet generates more stable predictions than WaveNet. In particular, there is a red sharp spike produced by WaveNet, which deviates far from real values. On the contrary, the curve of Graph WaveNet goes in the middle of real values all the time.

4.4.3.1 Effect of the Self-Adaptive Adjacency Matrix

To verify the effectiveness of my proposed adaptive adjacency matrix, I conduct experiments with Graph WaveNet using five different adjacency matrix configurations. Table 4.3 shows the average score of MAE, RMSE, and MAPE over 12 prediction horizons. I find that the adaptive-only model works even better than the forward-only model with mean MAE. When the graph structure is unavailable, Graph WaveNet would still be able to realize a good performance. The forward-backward-adaptive model achieves the lowest scores on all three evaluation metrics. It indicates that if graph structural information is given, adding the self-adaptive adjacency matrix could introduce new and useful information to the model. In Figure 4.5, I further investigate the learned self-adaptive adjacency matrix under the configuration of the forward-backward-adaptive model trained on the METR-LA dataset. According to Figure 4.5a, some columns have more high-value points than others such as column 9 in the left box compared to column 47 in the right box. It suggests that some nodes are influential to most nodes in a graph while other nodes have weaker impacts. Figure 4.5b confirms my observation. It can be seen that node 9 locates nearby the intersection of several main roads while node 47 lies in a single road.

Model	Computation Time	
	Training(s/epoch)	Inference(s)
DCRNN	249.31	18.73
STGCN	19.10	11.37
Graph WaveNet	53.68	2.27

Table 4.4: The computation cost on the METR-LA dataset.

4.4.3.2 Computation Time

I compare the computation cost of Graph WaveNet with DCRNN and STGCN on the METR-LA dataset in Table 4.4. Graph WaveNet runs five times faster than DCRNN but two times slower than STGCN in training. For inference, I measure the total time cost of each model on the validation data. Graph WaveNet is the most efficient of all at the inference stage. This is because that Graph WaveNet generates 12 predictions in one run while DCRNN and STGCN have to produce the results conditioned on previous predictions.

4.5 Summary

In this chapter, I present a novel model for spatial-temporal graph modeling. My model captures spatial-temporal dependencies efficiently and effectively by combining graph convolution with dilated casual convolution. I propose an effective method to learn hidden spatial dependencies automatically from the data. This opens up a new direction in spatial-temporal graph modeling where the dependency structure of a system is unknown but needs to be discovered. On two public traffic network datasets, Graph WaveNet achieves state-of-the-art results. In future work, I will study scalable methods to apply Graph WaveNet on large-scale datasets and explore approaches to learn dynamic spatial dependencies.

CHAPTER 5

CONNECTING THE DOTS: MULTIVARIATE TIME SERIES FORECASTING WITH GRAPH NEURAL NETWORKS

5.1 Motivation

Modern societies have benefited from a wide range of sensors to record changes in temperature, price, traffic speed, electricity usage, and many other forms of data. Recorded time series from different sensors can form multivariate time series data and can be interlinked. For example, the rise in daily temperature may cause an increase in electricity usage. To capture systematic trends over a group of dynamically changing variables, the problem of multivariate time series forecasting has been studied for at least sixty years. It has seen tremendous applications in the domains of economics, finance, bioinformatics, and traffic.

Multivariate time series forecasting methods inherently assume interdependencies among variables. In other words, each variable depends not only on its historical values but also on other variables. However, existing methods do not exploit latent interdependencies among variables efficiently and effectively. Statistical methods, such as vector auto-regressive model (VAR) and Gaussian process model (GP), assume a linear dependency among variables. The model complexity of statistical methods grows quadratically with the number of variables. They face the problem of overfitting with a large number of variables. Recently developed deep-learning-based methods, including LSTNet [217] and TPA-LSTM [218], are powerful to capture non-linear patterns. LSTNet encodes

short-term local information into low dimensional vectors using 1D convolutional neural networks and decodes the vectors through a recurrent neural network. TPA-LSTM processes the inputs by a recurrent neural network and employs a convolutional neural network to calculate the attention score across multiple steps. LSTNet and TPA-LSTM do not model the pair-wise dependencies among variables explicitly, which weakens model interpretability.

Graphs are a special form of data which describes the relationships between different entities. Recently, graph neural networks have achieved great success in handling graph data due to their permutation-invariance, local connectivity, and compositionality. By propagating information through structures, graph neural networks allow each node in a graph to be aware of its neighborhood context. Multivariate time series forecasting can be viewed naturally from a graph perspective. Variables from multivariate time series can be considered as nodes in a graph, and they are interlinked through their hidden dependency relationships. It follows that modeling multivariate time series data using graph neural networks can be a promising way to preserve their temporal trajectory while exploiting the interdependency among time series.

The most suitable type of graph neural networks for multivariate time series is spatial-temporal graph neural networks. Spatial-temporal graph neural networks take multivariate time series and an external graph structure as inputs, and they aim to predict future values or labels of multivariate time series. Spatial-temporal graph neural networks have achieved significant improvements compared to methods that do not utilize structural information. However, these approaches still fall short for modeling multivariate time series due to the following challenges:

- *Challenge 1: Unknown Graph Structure.* Existing GNN approaches rely heavily on a pre-defined graph structure in order to perform time series forecasting. In most cases, multivariate time series does not have an explicit graph structure. The relationships among variables have to be discovered from data rather than being provided as ground truth knowledge.

- *Challenge 2: Graph Learning & GNN Learning.* Even though a graph structure is available, most GNN approaches focus only on message passing (GNN Learning) and overlook the fact that the graph structure is not optimal and should be updated during training. The question then is how to simultaneously learn the graph structure and the GNN for time series in an end-to-end framework.

In this chapter, I propose a novel approach to overcome these challenges. As demonstrated by Figure 5.1, my framework consists of three core components - the graph learning layer, the graph convolution module, and the temporal convolution module. For *Challenge 1*, I propose a novel graph learning layer, which extracts a sparse graph adjacency matrix adaptively based on data. Furthermore, I develop a graph convolution module to address the spatial dependencies among variables, given the adjacency matrix computed by the graph learning layer. This is designed specifically for directed graphs and avoids the over-smoothing problem that frequently occurs in graph convolutional networks. Finally, I propose a temporal convolution module to capture temporal patterns by modified 1D convolutions. It can both discover temporal patterns with multiple frequencies and process very long sequences.

As all parameters are learnable through gradient descent, the proposed framework is able to model multivariate time series data and learn the internal graph structure simultaneously in an end-to-end manner (for *Challenge 2*). To reduce the difficulty of solving a highly non-convex optimization problem and to reduce memory occupation in processing large graphs, I propose a learning algorithm that uses a curriculum learning strategy to find a better local optimum and splits multivariate time series into subgroups during training. The advantages here are that my proposed framework is generally applicable to both **small and large graphs, short and long time series, with and without externally defined graph structures**. In summary, my main contributions are as follows:

- To the best of my knowledge, this is the first study on multivariate time series data generally from a graph-based perspective with graph neural networks.
- I propose a novel graph learning module to learn hidden spatial dependencies among variables. My method opens a new door for GNN models to handle data without explicit graph structure.
- I present a joint framework for modeling multivariate time series data and learning graph structures. My framework is more generic than any existing spatial-temporal graph neural network as it can handle multivariate time series with or without a pre-defined graph structure.
- Experimental results show that my method outperforms the state-of-the-art methods on 3 of 4 benchmark datasets and achieves on-par performance with other GNNs on two traffic datasets which provide extra structural information.

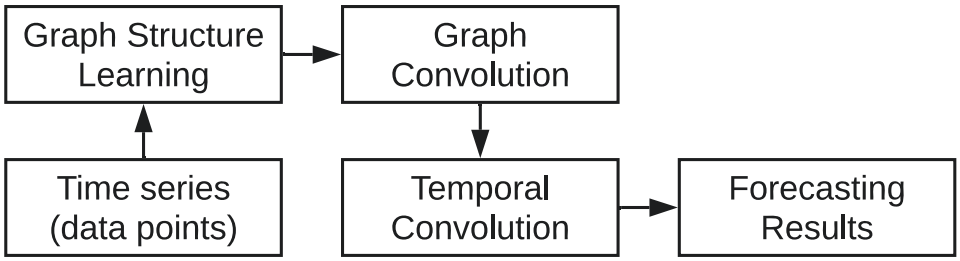


Figure 5.1: A concept map of my proposed framework.

5.2 Backgrounds

5.2.1 Multivariate Time Series Forecasting

Time series forecasting has been studied for a long time. The majority of existing methods follow a statistical approach. The auto-regressive integrated moving average (ARIMA) [219] generalizes a family of a linear model, including auto-regressive (AR), moving average (MA), and auto-regressive moving average (ARMA). The vector auto-regressive model (VAR) extends the AR model to capture the linear interdependencies among multiple time series. Similarly, the vector auto-regressive moving average model (VARMA) is proposed as a multivariate version of the ARMA model. Gaussian process (GP), as a Bayesian approach, models the distribution of a multivariate variable over functions. GP can be applied naturally to model multivariate time series data [220]. Although statistical models are widely used in time series forecasting due to their simplicity and interpretability, they make strong assumptions with respect to a stationary process and they do not scale well to multivariate time series data. Deep-learning-based approaches are free from stationary assumptions and they are effective methods to capture nonlinearity. Lai et al. [217] and Shih et al. [218] are the first two deep-learning-based models designed for multivariate time series forecasting. They employ convolutional neural networks to capture local dependencies among variables and recurrent neural networks to preserve long-term temporal dependencies. Convolutional neural networks encapsulate interactions among variables into a global hidden state. Therefore, they cannot fully exploit latent dependencies between pairs of variables.

5.2.2 Graph Neural Networks

Graph neural networks have enjoyed great success in handling spatial dependencies among entities in a network. Graph neural networks assume that the state of a node depends on the states of its neighbors. To capture this type of spatial dependency, various kinds of graph neural networks have been developed through message passing [37], information propagation [221], and graph convolution [1]. Sharing similar roles, they essentially capture a node's high-level representation by passing information from a node's neighbors to the node itself. Most recently, we have seen the emergence of a type of graph neural networks known as spatial-temporal graph neural networks. This form of neural networks is proposed initially to solve the problem of traffic prediction [13, 20, 80, 222, 223] and skeleton-based action recognition [15, 224]. The inputs to spatial-temporal graph neural networks are multivariate time series with an external graph structure which describes the relationships among variables in multivariate time series. For spatial-temporal graph neural networks, spatial dependencies among nodes are captured by graph convolutions, while temporal dependencies among historical states are preserved by recurrent neural networks [13, 19] or 1D convolutions [15, 20]. Although existing spatial-temporal graph neural networks have achieved significant improvements compared to methods without using a graph structure, they are incapable of handling pure multivariate time series data effectively due to the absence of a pre-defined graph and lack of a general framework.

5.3 Problem Formulation

In this chapter, I focus on the task of multivariate time series forecasting. Let $\mathbf{z}_t \in \mathbf{R}^N$ denote the value of a multivariate variable of dimension N at time step t, where $z_t[i] \in R$ denote the value of the i^{th} variable at time step t. Given a sequence of historical P time steps of observations on a multivariate variable, $\mathbf{X} = \{\mathbf{z}_{t_1}, \mathbf{z}_{t_2}, \cdots, \mathbf{z}_{t_P}\}$, my goal is to predict the Q-step-away value of $\mathbf{Y} = \{\mathbf{z}_{t_{P+Q}}\}$, or a sequence of future values $\mathbf{Y} = \{\mathbf{z}_{t_{P+1}}, \mathbf{z}_{t_{P+2}}, \cdots, \mathbf{z}_{t_{P+Q}}\}$. More generally, the input signals can be coupled with other auxiliary features such as time of the day, day of the week, and day of the season. Concatenating the input signals with auxiliary features, I assume the inputs instead are $\mathscr{X} = \{\mathbf{S}_{t_1}, \mathbf{S}_{t_2}, \cdots, \mathbf{S}_{t_P}\}$ where $\mathbf{S}_{t_i} \in \mathbf{R}^{N \times D}$, D is the feature dimension, the first column of $\mathbf{S}_{t_i}$ equals to $\mathbf{z}_{t_i}$, and the rest are auxiliary features. I aim to build a mapping $f(\cdot)$ from $\mathscr{X}$ to $\mathbf{Y}$ by minimizing the absolute loss with $l2$ regularization.

Graphs describe the relationships among entities in a network. I give a formal definition of graph-related concepts below.

Definition 4 (Graph)**.** *A graph is formulated as* $G = (V, E)$ *where* V *is the set of nodes, and* E *is the set of edges. I use* N *to denote the number of nodes in a graph.*

Definition 5 (Node Neighborhood)**.** *Let* $v \in V$ *to denote a node and* $e = (v, u) \in E$ *to denote an edge pointing from* u *to* v*. The neighborhood of a node* v *is defined as* $N(v) = \{u \in V | (v, u) \in E\}$.

Definition 6 (Adjacency Matrix)**.** *The adjacency matrix is a mathematical representation of a graph, denoted as* $\mathbf{A} \in R^{N \times N}$ *with* $A_{ij} = c > 0$ *if* $(v_i, v_j) \in E$ *and* $A_{ij} = 0$ *if* $(v_i, v_j) \notin E$.

From a graph-based perspective, I consider variables in multivariate time series as nodes in graphs. I describe the relationships among nodes using the graph adjacency matrix. The graph adjacency matrix is not given by the multivariate time series data in most cases and will be learned by my model.

5.4 Framework of MTGNN

5.4.1 Model Architecture

I first elaborate on the general framework of my model. As illustrated in Figure 5.2, MTGNN on the highest level consists of a *graph learning layer*, *m graph convolution modules*, *m temporal convolution modules*, and an output module. To discover hidden associations among nodes, a graph learning layer computes a graph adjacency matrix, which is later used as an input to all graph convolution modules. Graph convolution modules are interleaved with temporal convolution modules to capture spatial and temporal dependencies respectively. Figure 5.3 gives a demonstration of how a temporal convolution module and a graph convolution module collaborate with each other. To avoid the problem of gradient vanishing, residual connections are added from the inputs of a temporal convolution module to the outputs of a graph convolution module. Skip connections are added after each temporal convolution module. To get the final outputs, the output module projects the hidden features to the desired output dimension. In more detail, the core components of my model are illustrated in the following:

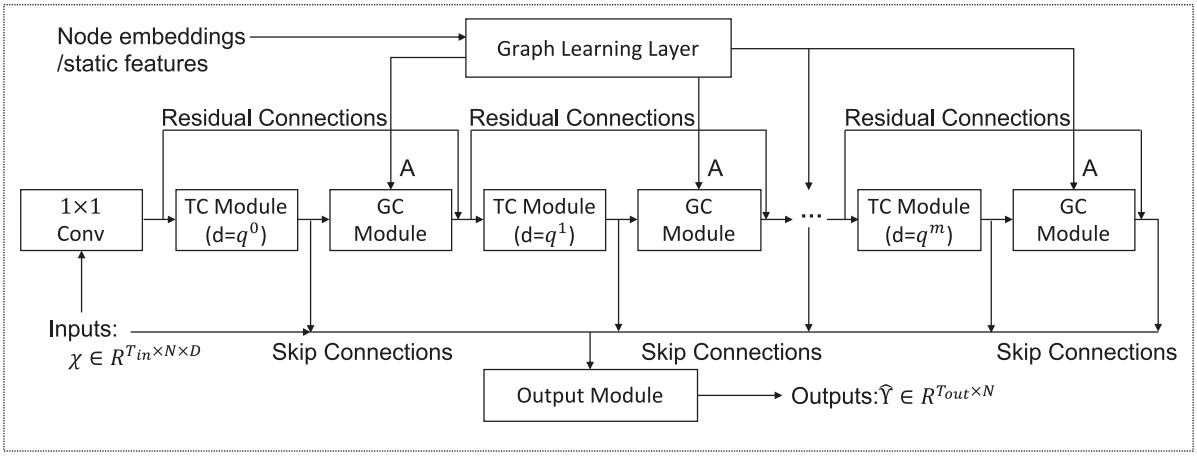


Figure 5.2: The framework of MTGNN. A 1×1 standard convolution first projects the inputs into a latent space. Afterward, temporal convolution modules and graph convolution modules are interleaved with each other to capture temporal and spatial dependencies respectively. The hyper-parameter, dilation factor d, which controls the receptive field size of a temporal convolution module, is increased at an exponential rate of q. The graph learning layer learns the hidden graph adjacency matrix, which is used by graph convolution modules. Residual connections and skip connections are added to the model to avoid the problem of gradient vanishing. The output module projects hidden features to the desired dimension to get the final results.

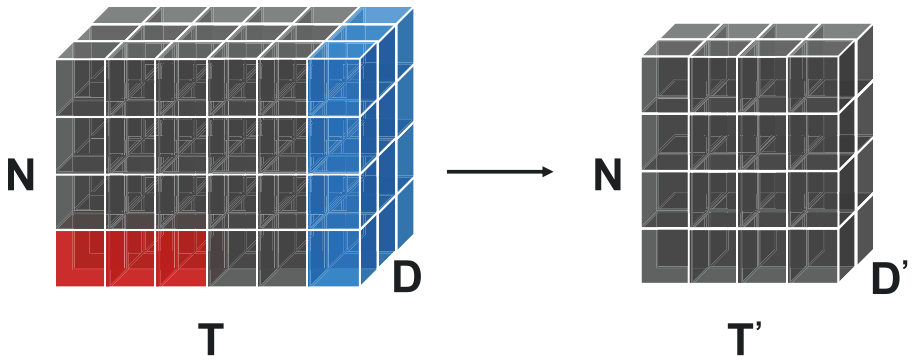


Figure 5.3: A demonstration of how a temporal convolution module and a graph convolution module collaborate with each other. A temporal convolution module filters the inputs by sliding a 1D window over the time and node axes, as denoted by the red. A graph convolution module filters the inputs at each step, denoted by the blue.

5.4.2 Graph Learning Layer

The graph learning layer learns a graph adjacency matrix adaptively to capture the hidden relationships among time series data. To construct a graph, existing studies measure the similarity between pairs of nodes by a distance metric, such as dot product and Euclidean distance [13]. This leads inevitably to the problem of high time and space complexity with $O(N^2)$. It means the computation and memory cost grows quadratically with the increase of graph size. This restricts the model's capability of handling larger graphs. To address this limitation, I adopt a sampling approach, which only calculates pair-wise relationships among a subset of nodes. This cuts off the bottleneck of computation and memory in each minibatch. More details will be provided in Section 5.4.6.

Another problem is that existing distance metrics are often symmetric or bi-directional. In multivariate time series forecasting, I expect that the change of a node's condition causes the change of another node's condition such as traffic flow. Therefore the learned relation is supposed to be uni-directional. My proposed graph learning layer is specifically designed to extract uni-directional relationships, illustrated as follows:

$$\mathbf{M}_1 = tanh(\alpha \mathbf{E}_1 \Theta_1) \tag{5.1}$$

$$\mathbf{M}_2 = tanh(\alpha \mathbf{E}_2 \Theta_2) \tag{5.2}$$

$$\mathbf{A} = ReLU(tanh(\alpha(\mathbf{M}_1 \mathbf{M}_2^T - \mathbf{M}_2 \mathbf{M}_1^T))) \tag{5.3}$$

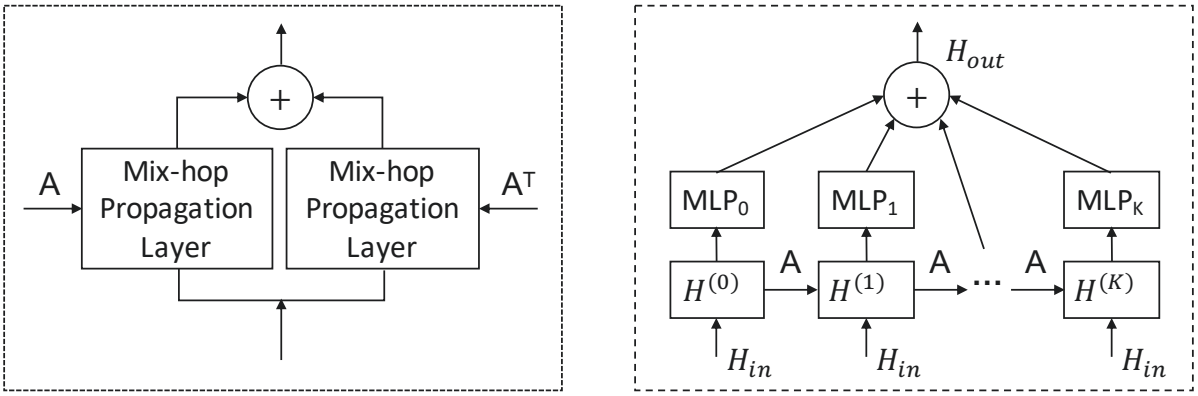
$$for\ i = 1, 2, \cdots, N \tag{5.4}$$

$$\mathbf{idx} = argtopk(\mathbf{A}[i, :]) \tag{5.5}$$

$$\mathbf{A}[i, -\mathbf{idx}] = 0, \tag{5.6}$$

where $\mathbf{E}_1, \mathbf{E}_2$ represents randomly initialized node embeddings, which are learnable during training, Θ_1, Θ_2 are model parameters, α is a hyper-parameter for controlling the saturation rate of the activation function, and $argtopk(\cdot)$ returns the index of the top-k largest values of a vector. The asymmetric property of my proposed graph adjacency matrix is achieved by Equation 5.3. The subtraction term and the ReLU activation function regularize the adjacency matrix so that if A_{vu} is positive, its diagonal counterpart A_{uv} will be zero. Equation 5.5-5.6 is a strategy to make the adjacency matrix sparse while reducing the computation cost of the following graph convolution. For each node, I select its top-k closest nodes as its neighbors. While retaining the weights for connected nodes, I set the weights of non-connected nodes as zero.

Incorporate External Data. The inputs to the graph learning layer are not limited to node embeddings. In case that external knowledge about the attributes of each node is



(a) GC module (b) Mix-hop propagation layer

Figure 5.4: Graph convolution and mix-hop propagation layer.

given, I can also set $\mathbf{E}_1 = \mathbf{E}_2 = \mathbf{Z}$, where $\mathbf{Z}$ is a static node feature matrix. Some works have considered capturing dynamic spatial dependencies [224, 225]. In other words, they dynamically adjust the weight of two connected nodes based on temporal inputs. However, assuming dynamic spatial dependencies makes the model extremely hard to converge when we need to learn the graph structure at the same time. The advantage of my approach is that we can learn stable and interpretable node relationships over the period of the training dataset. Once the model is trained in an online learning version, my graph adjacency matrix is also adaptable to change as new training data updates the model parameters.

5.4.3 Graph Convolution Module

The graph convolution module aims to fuse a node's information with its neighbors' information to handle spatial dependencies in a graph. The graph convolution module consists of two mix-hop propagation layers to process inflow and outflow information passed through each node separately. The net inflow information is obtained by adding the outputs of the two mix-hop propagation layers. Figure 5.4 shows the architecture of the graph convolution module and the mix-hop propagation layer.

Mix-hop Propagation Layer. Given a graph adjacency matrix, I propose the mix-hop propagation layer to handle information flow over spatially dependent nodes. The proposed mix-hop propagation layer consists of two steps - the information propagation step and the information selection step. I first give the mathematical form of these two steps and then illustrate my motivations. The information propagation step is defined as

follows:

$$\mathbf{H}^{(k)} = \beta \mathbf{H}_{in} + (1-\beta)\tilde{\mathbf{A}}\mathbf{H}^{(k-1)}, \tag{5.7}$$

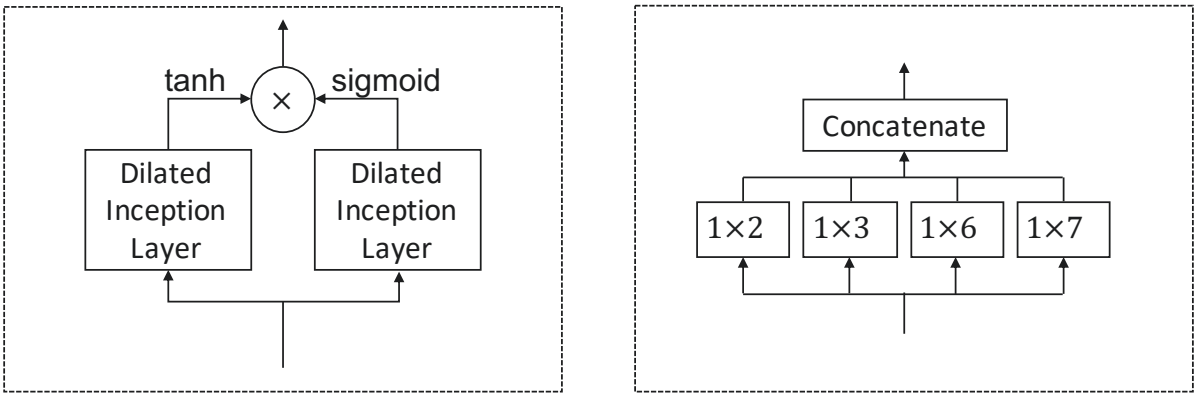
where β is a hyper parameter, which controls the ratio of retaining the root node's original states. The information selection step is defined as follows

$$\mathbf{H}_{out} = \sum_{i=0}^{K} \mathbf{H}^{(k)}\mathbf{W}^{(k)}, \tag{5.8}$$

where K is the depth of propagation, $\mathbf{H}_{in}$ represents the input hidden states outputted by the previous layer, $\mathbf{H}_{out}$ represents the output hidden states of the current layer, $\mathbf{H}^{(0)} = \mathbf{H}_{in}$, $\tilde{\mathbf{A}} = \tilde{\mathbf{D}}^{-1}(\mathbf{A}+\mathbf{I})$, and $\tilde{\mathbf{D}}_{ii} = 1 + \sum_j \mathbf{A}_{ij}$. In Figure 5.4b, I demonstrate the information propagation step and information selection step in the proposed mix-hop propagation layer. It first propagates information horizontally and selects information vertically.

The information propagation step propagates node information along with the given graph structure recursively. A severe limitation of graph convolutional networks is that node hidden states converge to a single point as the number of graph convolution layers goes to infinity. This is because the graph convolutional network with many layers reaches the random walk's limit distribution regardless of the initial node states. To address this problem, motivated by Klicpera et al. [221], I retain a proportion of nodes' original states during the propagation process so that the propagated node states can both preserve locality and explore a deep neighborhood. However, if I only apply Equation 5.7, some node information will be lost. Under the extreme circumstance that no spatial dependencies exist, aggregating neighborhood information simply adds useless noises to each node. Therefore, the information selection step is introduced to filter out important information produced at each hop. According to Equation 5.8, the parameter matrix $\mathbf{W}^{(k)}$ functions as a feature selector. When the given graph structure does not entail spatial dependencies, Equation 5.8 is still able to preserve the original node-self information by adjusting $\mathbf{W}^{(k)}$ to 0 for all $k > 0$.

Connection to existing works. The idea of mix-hop has been explored by [226] and [227]. Kapoor et al. [226] concatenate information from different hops. Chen et al. [227] propose an attention mechanism to weight information among different hops. They both apply GCN for information propagation. However, as GCN faces the over-smoothing problem, information from higher hops may not or negatively contribute to the overall performance. To avoid this, my approach keeps a balance between local and neighborhood information. Furthermore, Kapoor et al. [226] show that their proposed model with two mix-hop layers has the capability to represent the delta difference between two consecutive hops. My



(a) TC module (b) Dilated inception layer

Figure 5.5: The temporal convolution and dilated inception layer.

approach can achieve the same effect with only one mix-hop propagation layer. Suppose $K = 2$, $\mathbf{W}^{(0)} = \mathbf{0}$, $\mathbf{W}^{(1)} = -\mathbf{1}$, and $\mathbf{W}^{(2)} = \mathbf{1}$, then

$$\mathbf{H}_{out} = \Delta(\mathbf{H}^{(2)}, \mathbf{H}^{(1)}) = \mathbf{H}^2 - \mathbf{H}^1. \tag{5.9}$$

From this perspective, using summation is more efficient to represent all linear interactions of different hops compared with the concatenation method.

5.4.4 Temporal Convolution Module

The temporal convolution module applies a set of standard dilated 1D convolution filters to extract high-level temporal features. This module consists of two dilated inception layers. One dilated inception layer is followed by a tangent hyperbolic activation function and works as a filter. The other layer is followed by a sigmoid activation function and functions as a gate to control the amount of information that the filter can pass to the next module. Figure 5.5 shows the architecture of the temporal convolution module and the dilated inception layer.

Dilated Inception Layer The temporal convolution module captures sequential patterns of time series data through 1D convolutional filters. To come up with a temporal convolution module that is able to both discover temporal patterns with various ranges and handle very long sequences, I propose the dilated inception layer which combines two widely applied strategies from convolutional neural networks, i.e., using filters with multiple sizes [228] and applying dilated convolution [213].

First, choosing the right kernel size is a challenging problem for convolutional networks. The filter size can be too large to represent short-term signal patterns subtly, or

too small to discover long-term signal patterns sufficiently. In image processing, a widely employed strategy is called inception, which concatenates the outputs of 2D convolution filters with three different kernel sizes, 1×1, 3×3, and 5×5. Moving from 2D images to 1D time series, the set of 1×1, 1×3, and 1×5 filter sizes do not suit the nature of temporal signals. As temporal signals tend to have several inherent periods such as 7, 12, 24, 28, and 60, a stack of inception layers with filter size 1×1, 1×3, and 1×5 cannot well encompass those periods. Alternatively, I propose a temporal inception layer consisting of four filter sizes, viz. 1×2, 1×3, 1×6, and 1×7. The aforementioned periods can all be covered by the combination of these filter sizes. For example, to represent the period 12, a model can pass the inputs through a 1×7 filter from the first temporal inception layer followed by a 1×6 filter from the second temporal inception layer.

Second, the receptive field size of a convolutional network grows in a linear progression with the depth of the network and the kernel size of the filter. Consider a convolutional network with m $1D$ convolution layers of kernel size c, the receptive field size of the convolutional network is,

$$R = m(c-1)+1. \tag{5.10}$$

To process very long sequences, it requires either a very deep network or very large filters. I adopt dilated convolution to reduce model complexity. Dilated convolution operates a standard convolution filter on down-sampled inputs with a certain frequency. For example, where the dilation factor is 2, it applies standard convolution on inputs sampled every two steps. Following [211], I let the dilation factor for each layer increase exponentially at a rate of q $(q > 1)$. Suppose the initial dilation factor is 1, the receptive field size of a m layer dilated convolutional network with kernel size c is

$$R = 1 + (c-1)(q^m - 1)/(q-1). \tag{5.11}$$

This indicates that the receptive field size of the network also grows exponentially with an increase in the number of hidden layers at the rate of q. Therefore, using this dilation strategy can capture much longer sequences than proceeding without it.

Formally, combining inception and dilation, I propose the dilated inception layer, demonstrated by Figure 5.5b. Given a 1D sequence input $\mathbf{z} \in \mathbf{R}^T$ and filters consisting of $\mathbf{f}_{1\times 2} \in \mathbf{R}^2$, $\mathbf{f}_{1\times 3} \in \mathbf{R}^3$, $\mathbf{f}_{1\times 6} \in \mathbf{R}^6$, and $\mathbf{f}_{1\times 7} \in \mathbf{R}^7$, my dilated inception layer takes the form,

$$\mathbf{z} = concat(\mathbf{z} \star \mathbf{f}_{1\times 2}, \mathbf{z} \star \mathbf{f}_{1\times 3}, \mathbf{z} \star \mathbf{f}_{1\times 6}, \mathbf{z} \star \mathbf{f}_{1\times 7}), \tag{5.12}$$

where the outputs of the four filters are truncated to the same length according to the largest filter and concatenated across the channel dimension, and the dilated convolution denoted by $\mathbf{z} \star \mathbf{f}_{1\times k}$ is defined as

$$\mathbf{z} \star \mathbf{f}_{1\times k}(t) = \sum_{s=0}^{k-1} \mathbf{f}_{1\times k}(s)\mathbf{z}(t - d \times s), \tag{5.13}$$

where d is the dilation factor.

5.4.5 Skip Connection Layer & Output Module

Skip connection layers are essentially $1 \times L_i$ standard convolutions where L_i is the sequence length of the inputs to the i^{th} skip connection layer. It standardizes information that jumps to the output module to have the same sequence length 1. The output module consists of two 1×1 standard convolution layers, transforming the channel dimension of the inputs to the desired output dimension. In case we want to predict a certain future step only, the desired output dimension is 1. When we want to predict Q consecutive steps, the desired output dimension is Q.

5.4.6 Proposed Learning Algorithm

I propose a learning algorithm to enhance my model's capability of handling large graphs and stabilizing in a better local optimum. Training on a graph often requires storing all node intermediate states into memory. If a graph is large, it will face the problem of memory overflow. Most relevant to us, Chiang et al. [3] propose a sub-graph training algorithm to tackle the memory bottleneck. They apply a graph clustering algorithm to partition a graph into sub-graphs and train a graph convolutional network on the partitioned sub-graphs. In my problem, it is not practical to cluster nodes based on their topological information because my model learns the latent graph structure at the same time. Alternatively, in each iteration, I randomly split the nodes into several groups and let the algorithm learn a sub-graph structure based on the sampled nodes. This gives each node the full possibility of being assigned with another node in one group so that the similarity score between these two nodes can be computed and updated. As a side benefit, if I split the nodes into s groups, I can reduce the time and space complexity of my graph learning layer from $O(N^2)$ to $(N/s)^2$ in each iteration. After training, as all node embeddings are well-trained, a global graph can be constructed to fully utilize spatial relationships. Although it is computationally expensive, the adjacency matrix can be pre-computed in parallel before making predictions.

Algorithm 1 The learning algorithm of MTGNN.

Input: The dataset O, node set V, the initialized MTGNN model $f(\cdot)$ with Θ, learning rate γ, batch size b, step size s, split size m (default=1).
set $iter = 1, r = 1$
repeat
  sample a batch ($\mathcal{X} \in R^{b \times T \times N \times D}, \mathcal{Y} \in R^{b \times T' \times N}$) from O.
  random split the node set V into m groups, $\cup_{i=1}^{m} V_i = V$.
  if $iter\%s == 0$ and $r <= T'$ **then**
    $r = r + 1$
  end if
  for i in 1:m **do**
    compute $\hat{\mathcal{Y}} = f(\mathcal{X}[:,:,id(V_i),:];\Theta)$
    compute $L = loss(\hat{\mathcal{Y}}[:,:r,:], \mathcal{Y}[:,:r,id(V_i)])$
    compute the stochastic gradient of Θ according to L.
    update model parameters Θ according to their gradients and the learning rate γ.
  end for
  $iter = iter + 1$.
until convergence

The second consideration of my proposed algorithm is to facilitate my model to stabilize in a better local optimum. In the task of multi-step forecasting, I observe that long-term predictions often achieve greater improvements than those in the short-term in terms of model performance. I believe the reason is that my model predicts multi-steps altogether, and long-term predictions produce a much higher loss than short-term predictions. As a result, to minimize the overall loss, the model focuses more on improving the accuracy of long-term predictions. To address this issue I propose a *curriculum learning* strategy for the multi-step forecasting task. The algorithm starts with solving the easiest problem, predicting the next one-step only. It is very advantageous for the model to find a good starting point. With the increase in iteration numbers, I increase the prediction length of the model gradually so that the model can learn the hard task step by step. Covering all this, my algorithm is given in Algorithm 1. Further complexity analysis of my model can be found in Appendix **C**.

5.5 Experimental Studies

I validate MTGNN on two tasks - both single-step and multi-step forecasting. First, I compare the performance of MTGNN with other multivariate time series models on four benchmark datasets for multivariate time series forecasting, where the aim is to predict a single future step. Furthermore, to show how well MTGNN performs, compared

Table 5.1: Dataset statistics.

Datasets	# Samples	# Nodes	Sample Rate	Input Length	Output Length
traffic	17,544	862	1 hour	168	1
solar-energy	52,560	137	10 minutes	168	1
electricity	26,304	321	1 hour	168	1
exchange-rate	7,588	8	1 day	168	1
metr-la	34272	207	5 minutes	12	12
pems-bay	52116	325	5 minutes	12	12

with other spatial-temporal graph neural networks which, in contrast, use pre-defined graph structural information, I evaluate MTGNN on two benchmark datasets for spatial-temporal graph neural networks, where the aim is to predict multiple future steps. Further results on parameter study can be found in Appendix **C**.

5.5.1 Experimental Setting

5.5.1.1 Data

In Table 5.1, I summarize statistics of benchmark datasets. Details of these datasets are introduced below.

Single-step forecasting

- Traffic: the traffic dataset from the California Department of Transportation contains road occupancy rates measured by 862 sensors in San Francisco Bay area freeways during 2015 and 2016.
- Solar-Energy: the solar-energy dataset from the National Renewable Energy Laboratory contains the solar power output collected from 137 PV plants in Alabama State in 2007.
- Electricity: the electricity dataset from the UCI Machine Learning Repository contains electricity consumption for 321 clients from 2012 to 2014.
- Exchange-Rate: the exchange-rate dataset contains the daily exchange rates of eight foreign countries including Australia, British, Canada, Switzerland, China, Japan, New Zealand, and Singapore ranging from 1990 to 2016.

Following [217], I split these four datasets into a training set (60%), validation set (20%), and test set (20%) in chronological order. The input sequence length is 168 and the output sequence length is 1. Models are trained independently to predict the target future step (horizon) 3, 6, 12, and 24.

Multi-step forecasting

- METR-LA: the METR-LA dataset from the Los Angeles Metropolitan Transportation Authority contains average traffic speed measured by 207 loop detectors on the highways of Los Angeles County ranging from Mar 2012 to Jun 2012.
- PEMS-BAY: the PEMS-BAY dataset from California Transportation Agencies (CalTrans) contains average traffic speed measured by 325 sensors in the Bay Area ranging from Jan 2017 to May 2017.

Following [13], I split these two datasets into a training set (70%), validation set (20%), and test set (10%) in chronological order. The input sequence length is 12, and the target sequence contains the next 12 future steps. The time of the day is used as an auxiliary feature for the inputs. For the selected baseline methods, the pairwise road network distances are used as the pre-defined graph structure.

5.5.1.2 Experimental Setup

I use five evaluation metrics, including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), Root Relative Squared Error (RRSE), and Empirical Correlation Coefficient (CORR). For RMSE, MAE, MAPE, and RRSE, lower values are better. For CORR, higher values are better. I repeat the experiment 10 times and report the average value of evaluation metrics. The model is trained by the Adam optimizer with gradient clip 5. The learning rate is 0.001. The l2 regularization penalty is 0.0001. Dropout with 0.3 is applied after each temporal convolution module. Layernorm is applied after each graph convolution module. The depth of the mix-hop propagation layer is set to 2. The retain ratio from the mix-hop propagation layer is set to 0.05. The saturation rate of the activation function from the graph learning layer is set to 3. The dimension of node embeddings is 40. Other hyper-parameters are reported according to different tasks.

Single-step forecasting I use 5 graph convolution modules and 5 temporal convolution modules with the dilation exponential factor 2. The starting 1×1 convolution has 1

input channel and 16 output channels. The graph convolution module and the temporal convolution modules both have 16 output channels. The skip connection layers all have 32 output channels. The first layer of the output module has 64 output channels and the second layer of the output module has 1 output channel. The number of training epochs is 30. For Traffic, Solar-Energy, and Electricity, the number of neighbors for each node is 20. For Exchange-Rate, the number of neighbors for each node is 8. The batch size is set to 4. For the MTGNN+sampling model, I split the nodes of a graph into three partitions randomly with a batch size of 16. Following [217], I use RSE and CORR as evaluation metrics.

Multi-step forecasting I use 3 graph convolution modules and 3 temporal convolution modules with the dilation exponential factor 1. The starting 1×1 convolution has 2 input channels and 32 output channels. The graph convolution module and the temporal convolution module both have 32 output channels. The skip connection layers all have 64 output channels. The first layer of the output module has 128 output channels and its second layer has 12 output channels. The number of neighbors for each node is 20. The number of training epochs is 100. The batch size is set to 64. Following [13], I use MAE, RMSE , and MAPE as evaluation metrics.

5.5.2 Baseline Methods for Comparision

MTGNN and MTGNN+sampling are my models to be evaluated. MTGNN is my proposed model. MTGNN+sampling is my proposed model trained on a sampled subset of a graph in each iteration. Baseline methods are summarized in the following:

5.5.2.1 Single-step forecasting

- AR: An auto-regressive model.
- VAR-MLP: A hybrid model of the multilayer perception (MLP) and auto-regressive model (VAR) [229].
- GP: A Gaussian Process time series model [230, 231].
- RNN-GRU: A recurrent neural network with fully connected GRU hidden units.
- LSTNet: A deep neural network, which combines convolutional neural networks and recurrent neural networks [217].

- TPA-LSTM: An attention-recurrent neural network [218].

5.5.2.2 Multi-step forecasting

- DCRNN: A diffusion convolutional recurrent neural network, which combines diffusion graph convolutions with recurrent neural networks [13].
- STGCN: A spatial-temporal graph convolutional network, which incorporates graph convolutions with 1D convolutions [20].
- Graph WaveNet: A spatial-temporal graph convolutional network, which integrates diffusion graph convolutions with 1D dilated convolutions [80].
- ST-MetaNet: A sequence-to-sequence architecture, which employs meta networks to generate parameters [232].
- GMAN: A graph multi-attention network with spatial and temporal attentions [222].
- MRA-BGCN: A multi-range attentive bicomponent GCN [223].

5.5.3 Main Results

Table 5.2 and Table 5.3 provide the main experimental results of MTGNN and MTGNN+sampling. I observe that MTGNN achieves state-of-the-art results on most of the tasks, and the performance of MTGNN only degrades marginal when it samples sub-graphs for training. In the following, I discuss experimental results of single-step and multi-step forecasting respectively.

5.5.3.1 Single-step forecasting

In this experiment, I compare MTGNN with other multivariate time series models. Table 5.2 shows the experimental results for the single-step forecasting task. In general, my MTGNN achieves state-of-the-art results over almost all horizons on Solar-Energy, Traffic, and Electricity data. In particular, on Traffic data, the improvement of MTGNN in terms of RSE is significant. MTGNN lowers down RSE by 7.24%, 3.88%, 4.83% over the horizons of 3, 12, 24 on the traffic data. The main reason why MTGNN improves the results of traffic data evidently is that the nature of traffic data is better suited for my model assumption about the spatial-temporal dependencies. Obviously, the future

Table 5.2: Baseline comparison under single-step forecasting for multivariate time series methods.

Dataset		Solar-Energy				Traffic				Electricity				Exchange-Rate			
		Horizon				Horizon				Horizon				Horizon			
Methods	Metrics	3	6	12	24	3	6	12	24	3	6	12	24	3	6	12	24
AR	RSE	0.2435	0.3790	0.5911	0.8699	0.5991	0.6218	0.6252	0.63	0.0995	0.1035	0.1050	0.1054	0.0228	0.0279	0.0353	0.0445
	CORR	0.9710	0.9263	0.8107	0.5314	0.7752	0.7568	0.7544	0.7519	0.8845	0.8632	0.8591	0.8595	0.9734	0.9656	0.9526	0.9357
VARMLP	RSE	0.1922	0.2679	0.4244	0.6841	0.5582	0.6579	0.6023	0.6146	0.1393	0.1620	0.1557	0.1274	0.0265	0.0394	0.0407	0.0578
	CORR	0.9829	0.9655	0.9058	0.7149	0.8245	0.7695	0.7929	0.7891	0.8708	0.8389	0.8192	0.8679	0.8609	0.8725	0.8280	0.7675
GP	RSE	0.2259	0.3286	0.5200	0.7973	0.6082	0.6772	0.6406	0.5995	0.1500	0.1907	0.1621	0.1273	0.0239	0.0272	0.0394	0.0580
	CORR	0.9751	0.9448	0.8518	0.5971	0.7831	0.7406	0.7671	0.7909	0.8670	0.8334	0.8394	0.8818	0.8713	0.8193	0.8484	0.8278
RNN-GRU	RSE	0.1932	0.2628	0.4163	0.4852	0.5358	0.5522	0.5562	0.5633	0.1102	0.1144	0.1183	0.1295	0.0192	0.0264	0.0408	0.0626
	CORR	0.9823	0.9675	0.9150	0.8823	0.8511	0.8405	0.8345	0.8300	0.8597	0.8623	0.8472	0.8651	0.9786	**0.9712**	0.9531	0.9223
LSTNet-skip	RSE	0.1843	0.2559	0.3254	0.4643	0.4777	0.4893	0.4950	0.4973	0.0864	0.0931	0.1007	0.1007	0.0226	0.0280	0.0356	0.0449
	CORR	0.9843	0.9690	0.9467	0.8870	0.8721	0.8690	0.8614	0.8588	0.9283	0.9135	0.9077	0.9119	0.9735	0.9658	0.9511	0.9354
TPA-LSTM	RSE	0.1803	**0.2347**	0.3234	0.4389	0.4487	0.4658	0.4641	0.4765	0.0823	0.0916	0.0964	0.1006	**0.0174**	**0.0241**	**0.0341**	**0.0444**
	CORR	0.9850	**0.9742**	0.9487	0.9081	0.8812	0.8717	0.8717	0.8629	0.9439	0.9337	0.9250	0.9133	**0.9790**	0.9709	0.9564	0.9381
MTGNN	RSE	**0.1778**	0.2348	**0.3109**	**0.4270**	**0.4162**	0.4754	**0.4461**	**0.4535**	**0.0745**	0.0878	**0.0916**	**0.0953**	0.0194	0.0259	0.0349	0.0456
	CORR	**0.9852**	0.9726	**0.9509**	0.9031	**0.8963**	0.8667	**0.8794**	**0.8810**	**0.9474**	0.9316	**0.9278**	**0.9234**	0.9786	0.9708	0.9551	0.9372
MTGNN+sampling	RSE	0.1875	0.2521	0.3347	0.4386	0.4170	**0.4435**	0.4469	0.4537	0.0762	**0.0862**	0.0938	0.0976	0.0212	0.0271	0.0350	0.0454
	CORR	0.9834	0.9687	0.9440	0.8990	0.8960	**0.8815**	0.8793	0.8758	0.9467	**0.9354**	0.9261	0.9219	0.9788	0.9704	**0.9574**	**0.9382**

traffic occupancy rate of a road not only depends on its past but also on its connected roads' occupancy rates. MTGNN fails to make improvements on the exchange-rate data, possibly due to the smaller graph size and fewer training examples of exchange-rate data.

5.5.3.2 Multi-step forecasting

In this experiment, I compare MTGNN with other spatial-temporal graph neural network models. Table 5.3 shows the experimental results for the task of multi-step forecasting. The significance of MTGNN lies in that it achieves on-par performance with state-of-the-art spatial-temporal graph neural networks without using a pre-defined graph, while DCRNN, STGCN, and MRA-BGCN fully rely on pre-defined graphs. Graph Wavenet proposes a self-adaptive adjacency matrix, but it needs to combine with a pre-defined graph in order to achieve optimal performance. ST-MetaNet employs attention mechanisms to adjust the edge weights of a pre-defined graph. GMAN leverages node2vec algorithm to preserve node structural information while performing attention mechanisms. When a graph is not defined, these methods cannot model multivariate times series data efficiently.

Table 5.3: Baseline comparison under multi-step forecasting for spatial-temporal graph neural networks.

	Horizon 3			Horizon 6			Horizon 12		
	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
METR-LA									
DCRNN	2.77	5.38	7.30%	3.15	6.45	8.80%	3.60	7.60	10.50%
STGCN	2.88	5.74	7.62%	3.47	7.24	9.57%	4.59	9.40	12.70%
Graph WaveNet	2.69	5.15	6.90%	3.07	6.22	8.37%	3.53	7.37	10.01%
ST-MetaNet	2.69	5.17	6.91%	3.10	6.28	8.57%	3.59	7.52	10.63%
MRA-BGCN	**2.67**	**5.12**	6.80%	3.06	6.17	8.30%	3.49	7.30	10.00%
GMAN	2.77	5.48	7.25%	3.07	6.34	8.35%	**3.40**	**7.21**	**9.72%**
MTGNN	2.69	5.18	6.86%	**3.05**	**6.17**	**8.19%**	3.49	7.23	9.87%
MTGNN+sampling	2.76	5.34	**5.18%**	3.11	6.32	8.47%	3.54	7.38	10.05%
PEMS-BAY									
DCRNN	1.38	2.95	2.90%	1.74	3.97	3.90%	2.07	4.74	4.90%
STGCN	1.36	2.96	2.90%	1.81	4.27	4.17%	2.49	5.69	5.79%
Graph WaveNet	1.30	2.74	**2.73%**	1.63	3.70	3.67%	1.95	4.52	4.63%
ST-MetaNet	1.36	2.90	2.82%	1.76	4.02	4.00%	2.20	5.06	5.45%
MRA-BGCN	**1.29**	**2.72**	2.90%	**1.61**	**3.67**	3.80%	1.91	4.46	4.60%
GMAN	1.34	2.82	2.81%	1.62	3.72	**3.63%**	**1.86**	**4.32**	**4.31%**
MTGNN	1.32	2.79	2.77%	1.65	3.74	3.69%	1.94	4.49	4.53%
MTGNN+sampling	1.34	2.83	2.83%	1.67	3.79	3.78%	1.95	4.49	4.62%

5.5.4 Parameter Study

I conduct a parameter study on eight core hyper-parameters which influence the model complexity of MTGNN. I list these hyper-parameters as follows: Number of layers, the number of temporal convolution modules, ranges from 1 to 6; number of filters, the number of output channels for temporal convolution modules and graph convolution modules, ranges from 4 to 128; number of neighbors, the parameter k in Equation 5.5, ranges from 10 to 60; saturation rate, the parameter α in Equation 5.1, 5.2, and 5.3, ranges from 0.5 to 5; retain ratio of mix-hop propagation layer, the parameter β in Equation 5.7, ranges from 0 to 0.8; depth of mix-hop propagation layer, the parameter K in Equation 5.8, ranges from 1 to 6.

I repeat each experiment 10 times with 50 epochs each time and report the average of MAE with a standard deviation over 10 runs on the validation set. I change the parameter under investigation and fix other parameters in each experiment. Figure 5.6 shows the experimental results of my parameter study. As shown in Figure 5.6a and Figure 5.6b, increasing the number of layers and filters enhances my model's expressive capacity,

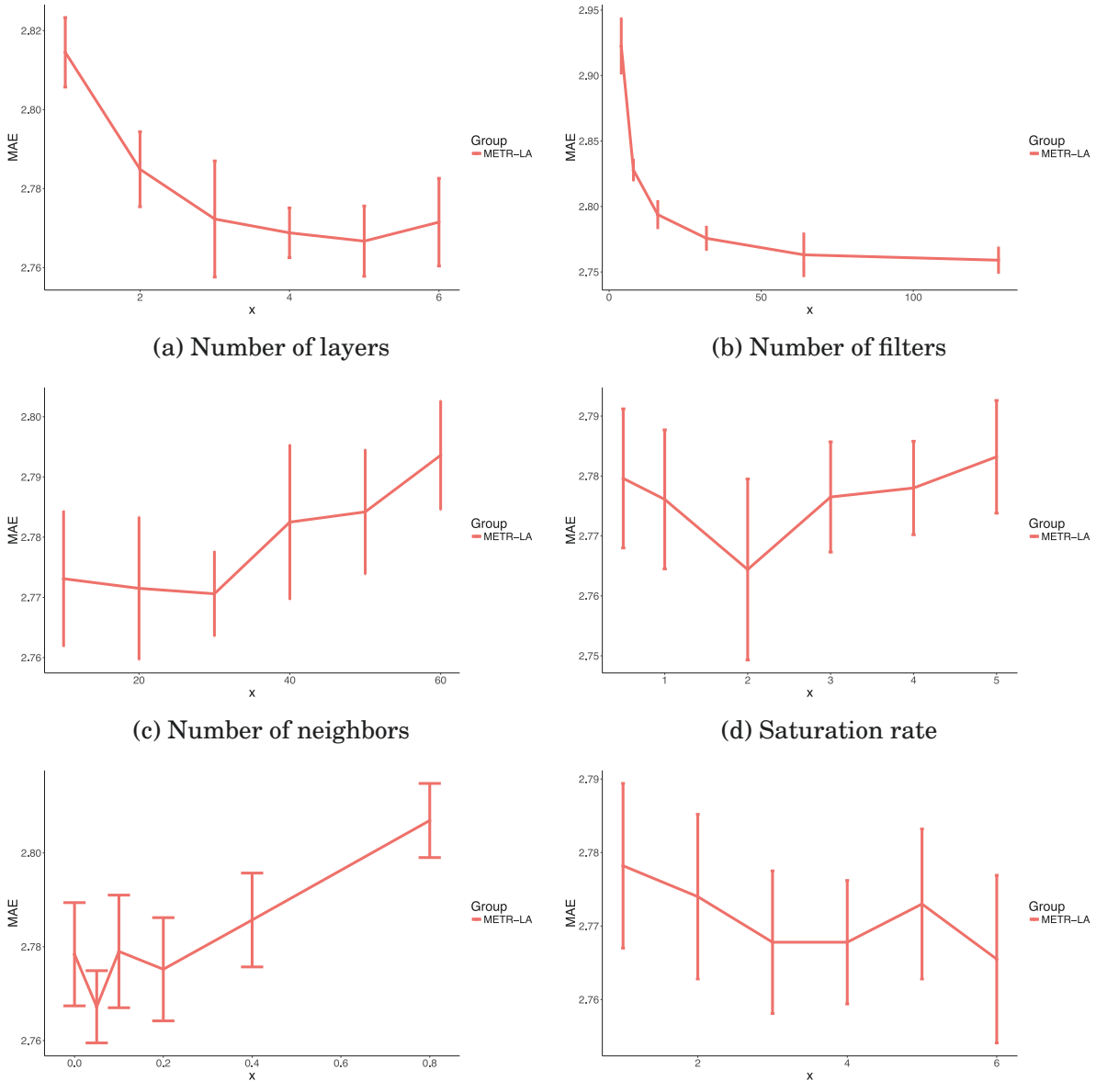
while reducing the MAE loss. Figure 5.6c shows that a small number of neighbors gives better results. It is possibly because a node may only depend on a limited number of other nodes, and increasing its neighborhood merely introduces noises to the model. The model performance is not sensitive to the saturation rate, as shown in Figure 5.6d. However, a large saturation rate can impose values of the adjacency matrix produced by the graph learning layer approach to 0 or 1. As shown in Figure 5.6e, a high retain ratio degrades the model performance significantly. I think it is because by default the propagation depth of the mix-hop propagation layer is set to 2, and as a result, keeping a high proportion of root information constrains a node from exploring its neighborhood. Figure 5.6f shows that it is enough to propagate node information with 2 or 3 steps. With the increase of the depth of propagation, the proposed mix-hop propagation layer does not suffer from the over-smoothing problem incurred by information aggregation. With the depth of propagation equal to 6, it has the lowest mean MAE with a larger variation.

5.5.5 Ablation Study

I conduct an ablation study on the METR-LA data to validate the effectiveness of key components that contribute to the improved outcomes of my proposed model. I name MTGNN without different components as follows:

- **w/o GC**: MTGNN without the graph convolution module. I replace the graph convolution module with a linear layer.
- **w/o Mix-hop**: MTGNN without the information selection step in the mix-hop propagation layer. I pass the outputs of the information propagation step to the next module directly.
- **w/o Inception**: MTGNN without inception in the dilated inception layer. While keeping the same number of output channels, I use a single 1×7 filter only.
- **w/o CL**: MTGNN without curriculum learning. I train MTGNN without gradually increasing the prediction length.

I repeat each experiment 10 times with 50 epochs per repetition and report the average of MAE, RMSE, MAPE with a standard deviation over 10 runs on the validation set in Table 5.4. The introduction of graph convolution modules significantly improves the results as it enables information flow among isolated but interdependent nodes. The effect of mix-hop is evident as well: it validates that the use of mix-hop is helpful



(a) Number of layers

(b) Number of filters

(c) Number of neighbors

(d) Saturation rate

(e) Retain ratio

(f) Depth of propagation

Figure 5.6: Parameter Study. X-axis is the parameter to be studied. Y-axis is the MAE score.

Table 5.4: Ablation study.

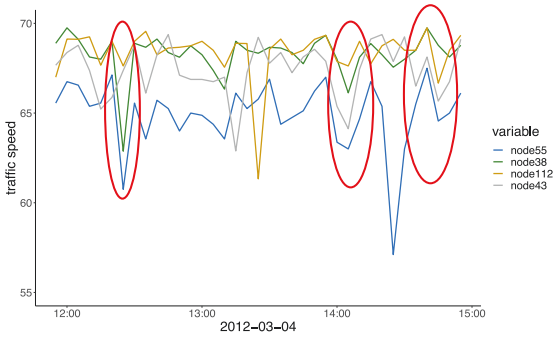
Methods	MTGNN	w/o GC	w/o Mix-hop	w/o Inception	w/o CL
MAE	**2.7715±0.0119**	2.8953±0.0054	2.7975±0.0089	2.7772±0.0100	2.7828±0.0105
RMSE	**5.8070±0.0512**	6.1276±0.0339	5.8549±0.0474	5.8251±0.0429	5.8248±0.0366
MAPE	**0.0778±0.0009**	0.0831±0.0009	0.0779±0.0009	0.0778±0.0010	0.0784±0.0009

for selecting useful information at each information propagation step in the mix-hop propagation layer. The effect of inception is significant in terms of RMSE, but marginal in terms of MAE. This is because using a single 1×7 filter has half more parameters than using a combination of $1 \times 2, 1 \times 3, 1 \times 5, 1 \times 7$ filters under the condition that the number of output channels for the dilated inception layer remains the same. Lastly, my curriculum learning strategy proves to be effective. It enables my model to converge quickly to an optimum that fits for the easiest task, and fine-tune parameters step by step as the level of learning difficulty increases.

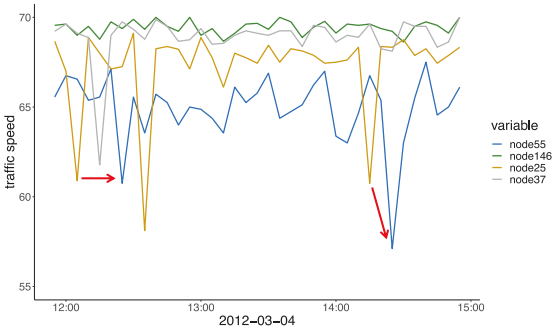
5.5.6 Study of the Graph Learning Layer

To validate the effectiveness of my proposed graph learning layer, I conduct a study which experiments with different ways of constructing a graph adjacency matrix. Table 5.5 shows different forms of $\mathbf{A}$ with experimental results tested on the validation set of the METR-LA data averaged on 10 runs. Predefined-A is constructed by road network distance [13]. Global-A assumes the adjacency matrix is a parameter matrix, which contains N^2 parameters. Motivated by [80], Undirected-A and Directed-A are computed by the similarity scores of node embeddings. Motivated by [224, 225], Dynamic-A assumes the spatial dependency at each time step is dependent on its node inputs. Uni-directed-A is my proposed method. According to Table 5.5, my proposed uni-directed-A achieves the lowest mean MAE, RMSE, and MAPE. It improves over predefined-A, undirected-A, and dynamic-A significantly. My uni-directed-A improves over undirected-A and directed-A marginally in terms of MAE and MAPE but proves to be more robust due to a lower RMSE.

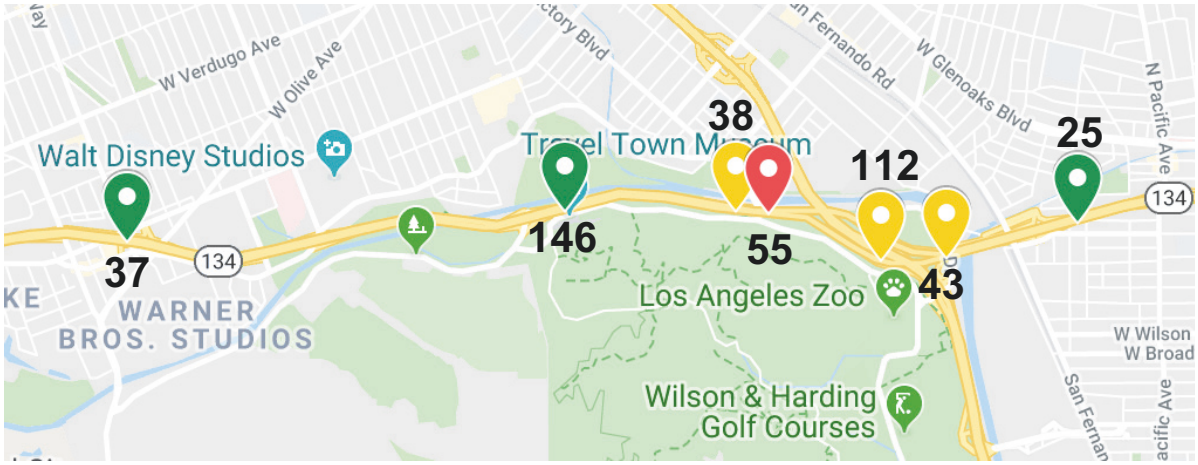
I further investigate the learned graph adjacency matrix via a case study. In Figure 5.7a, I plot the raw time series of node 55 and its pre-defined top-3 neighbors. In Figure 5.7b, I chart the raw time series of node 55 and its learned top-3 neighbors. Figure 5.7c shows the geo-location of these nodes, with green nodes representing the central node's learned top-3 neighbors and yellow nodes representing the central node's pre-defined



(a) Time series of node 55 and its top-3 neighbors given by the pre-defined **A**. The blue line represents node 55.



(b) Time series of node 55 and its top-3 neighbors given by the learned **A**. The blue line represents node 55.



(c) Node locations of node 55 and its neighbors marked on Google Maps. Yellow nodes represent node 55's top-3 neighbors given by the pre-defined **A**. Green nodes represent node 55's top-3 neighbors given by the learned **A**.

Figure 5.7: Case study

top-3 neighbors. I observe that the central node's pre-defined top-3 neighbors are much closer to the node itself on the map. As a result, their time series are more correlated simultaneously, as shown by the red circles in Figure 5.7a. On the contrary, the central node's learned top-3 neighbors distribute further away from it but still lie on the same road it follows. According to Figure 5.7b, time series of the learned top-3 neighbors are more capable of indicating extreme traffic conditions of the central node in advance.

5.5.7 Complexity Analysis

I analyze the time complexity of the main components of the proposed model MTGNN, which is summarized in Table 5.6. The time complexity of the graph learning layer is

Table 5.5: Comparison of different graph learning methods.

Methods	Equation	MAE	RMSE	MAPE
Pre-defined-A	-	2.9017±0.0078	6.1288±0.0345	0.0836±0.0009
Global-A	$\mathbf{A} = ReLU(\mathbf{W})$	2.8457±0.0107	5.9900±0.0390	0.0805±0.0009
Undirected-A	$\mathbf{A} = ReLU(tanh(\alpha(\mathbf{M}_1\mathbf{M}_1^T)))$	2.7736±0.0185	5.8411±0.0523	0.0783±0.0012
Directed-A	$\mathbf{A} = ReLU(tanh(\alpha(\mathbf{M}_1\mathbf{M}_2^T)))$	2.7758±0.0088	5.8217±0.0451	0.0783±0.0006
Dynamic-A	$\mathbf{A}_t = SoftMax(tanh(\mathbf{X}_t\mathbf{W}_1)tanh(\mathbf{W}_2^T\mathbf{X}_t^T))$	2.8124±0.0102	5.9189±0.0281	0.0794±0.0008
Uni-directed-A (mine)	$\mathbf{A} = ReLU(tanh(\alpha(\mathbf{M}_1\mathbf{M}_2^T - \mathbf{M}_2\mathbf{M}_1^T)))$	**2.7715±0.0119**	**5.8070±0.0512**	**0.0778±0.0009**

Components	Time Complexity
Graph Learning Layer	$O(Ns_1s_2 + N^2s_2)$
Graph Convolution Module	$O(K(Md_1 + Nd_1d_2)$
Temporal Convolution Module	$O(Nlc_ic_o/d)$

Table 5.6: Time Complexity Analysis

($O(Ns_1s_2 + N^2s_2)$) where N denotes the number of nodes, s_1 represents the dimension of node input feature vectors, and s_2 represents the dimension of node hidden feature vectors. Treating s_1 and s_2 as constants, the time complexity of the graph learning layer becomes $O(N^2)$. It is attributed to the pairwise computation of node hidden feature vectors. The graph convolution module incurs $O(K(Md_1+Nd_1d_2)$ time complexity, where K is the propagation depth, N is the number of nodes, d_1 denotes the input dimension of node states, d_2 denotes the output dimension of node states. Regarding K, d_1 and d_2 as constants, the time complexity of the graph convolution module turns to $O(M)$. This result comes from the fact that in the information propagation step, each node receives information from its neighbors and the sum of the number of neighbors of each node exactly equals the number of edges. The time complexity of the temporal convolution module equals to $O(Nlc_ic_o/d)$, where l is the input sequence length, c_i is the number of input channels, c_o is the number of output channels, and d is the dilation factor. The time complexity of the temporal convolution module mainly depends on $N \times l$, which is the size of the input feature map.

5.6 Summary

In this chapter, I introduce a novel framework for multivariate time series forecasting. To the best of my knowledge, I am the first to address the multivariate time series forecasting problem via a graph-based deep learning approach. I propose an effective method to exploit the inherent dependency relationships among multiple time series. My method demonstrates superb performance in a variety of multivariate time series forecasting tasks and opens a new door to use GNNs to handle diverse non-structural data.

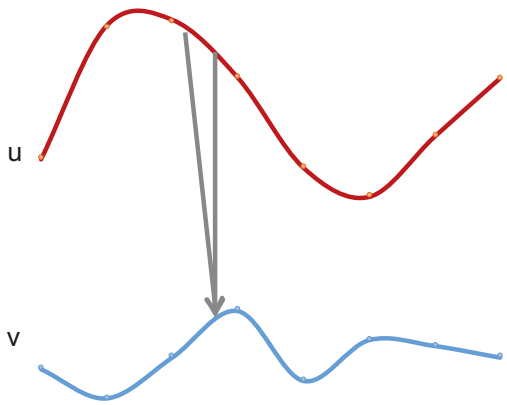
CHAPTER 6

TraverseNet: Unifying Space and Time in Message Passing

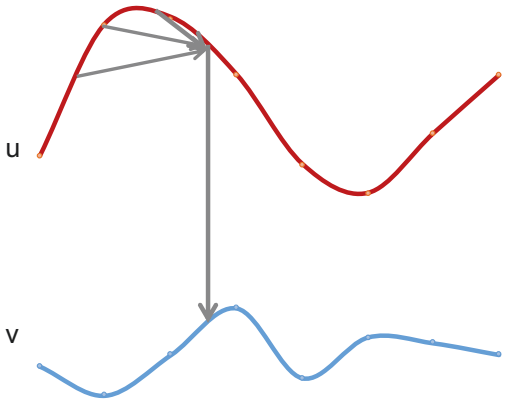
6.1 Motivation

Spatial-temporal graphs are ubiquitous in the present world (e.g., EEG signals recognition [233], skeleton-based action recognition [15], and traffic networks [232]). Spatial-temporal graph modeling assumes that the nodes in a topological structure contain spatial-temporal attributes and a node's future pattern is subject to its neighbors' historical results as well as its own past records. The traffic network is a quintessential example. The traffic conditions on a particular road at a given time depend not only on that road's previous traffic conditions but also on the traffic conditions of its adjacent roads several minutes ago, as it takes time for vehicles to flow from one point to the next. Hence, how to effectively exploit and preserve both spatial and temporal inner-dependency turns into an essential challenge to answer.

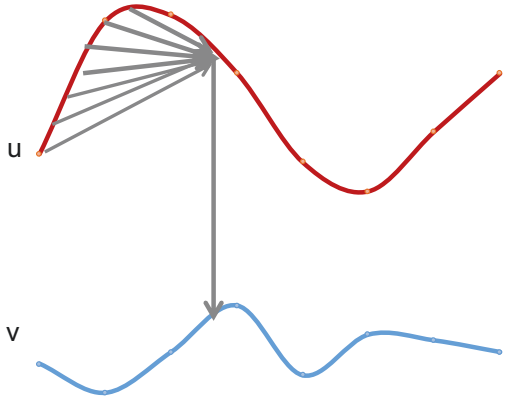
A natural approach is to stack graph convolutional networks (GCNs) and recurrent neural networks (RNNs) to jointly capture spatial topology and temporal sequence in a spatial-temporal graph (ST-Graph) [13, 19, 57]. While effective in fusing topological information into temporal sequence learning, RNN-based frameworks are inefficient in capturing long-range spatial-temporal dependencies. As illustrated by Figure 6.1a, during each recurrent step, they only allow a node to be aware of its neighbors' current inputs and its neighbors' previous hidden states. Information loss is inevitable when



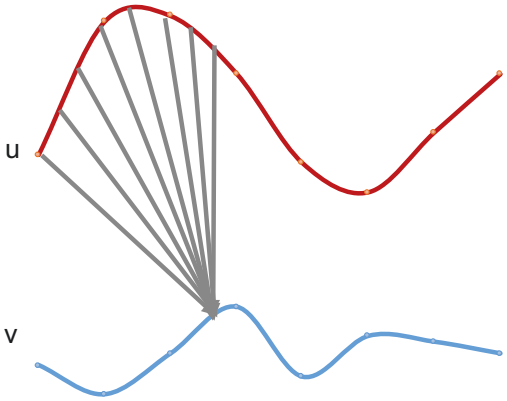
(a) RNN-based Methods. The node v receives raw information of its neighbor u at the current step and latent information from the last time step.



(b) CNN-based Methods. The node v receives information from its neighbor u at the current time step and a short window of recent time steps.



(c) Attention-based Methods. The node v receives information from its neighbor u at the current time step and a conditional weighted sum of information from recent time steps based on the states between two sides.



(d) My method. The node v selectively receives information of its neighbor u within a period of time directly. The importance weight of each time step is condition on the corresponding states between two sides.

Figure 6.1: Message Passing Diagrams of Different STGNNs. The blue line denotes the time series of a node v. The red line is the time series of its neighbor u. The arrows are used to illustrate the message passing paradigms in different spatial-temporal approaches.

processing long time series.

Meanwhile, capitalizing on parallel computing and stable convolutional propagation, CNN-based ST-Graph frameworks have received considerable attention [15, 20]. They stack temporal convolution layers and graph convolution layers to capture local spatial-temporal dependencies. As illustrated in Figure 6.1b, they first use a small 1D convolutional kernel to propagate node temporal information to the current time step, then use the graph convolution to pass the aggregated temporal information of a node's neighbors to the node itself. To capture the long-range spatial-temporal dependencies, they face the trade-off between kernel size and the number of layers. If a large 1D kernel is used to retain long-range spatial-temporal relations, the model has to be shallow because of a small number of layers. Alternatively, if a small kernel is used, a large number of 1D CNN layers and graph convolutional layers are requested, resulting in efficiency issues.

The attention mechanism is known for its efficiency of delivering important information [234, 235]. A number of studies integrate spatial attentions with temporal attentions for spatial-temporal data modeling [225, 236, 237]. The information flow diagram of attention-based methods is similar to that of CNN-based methods. Instead of using convolutional kernels, they apply attention mechanisms for information aggregation. As illustrated by Figure 6.1c, they first use temporal attentions to aggregate important historical information of a node u into its current step. Then they select important information of a node v's neighborhood by spatial attention and pass it to the node v itself. The temporal attention weights of a node only depend on the node itself. When passing the temporal information to its neighbors, its neighbors will receive the same temporal information and cannot determine which period of information is more important to themselves based on their own conditions.

To summarize, existing approaches either model spatial-temporal dependencies locally or model spatial correlations and temporal correlations separately. They prevent a node from being directly aware of its neighborhood long-range historical information. In fact, a node,Äôs current state may depend on its neighbors' previous states within a certain period of time. As illustrated by Figure 6.1d, the rise of a node's curve may exert influence on its neighbors several time steps later because of physical distance. For example, traffic congestion of a road will cause another congestion of its nearby roads 15 minutes later. It suggests that treating spatial correlations and temporal correlations locally or separately is inappropriate. Additionally, existing ST-GNNs tend to simply stack different layers (e.g., inception layer, dilated convolutional layer and

RNN/CNN layer) together, resulting in overly-complex and cumbersome architecture. Such construction confuses the significance and contribution of each kind of layers, and further damages the space-time continuum as well as tears the space-time integrity.

To overcome the above challenges, I present TraverseNet, a novel spatial-temporal neural network for structured data. TraverseNet processes a spatial-temporal graph as an inseparable entity. My specially designed message traverse layer enables a node to be wise to a period of information from its neighborhood explicitly. I leverage attention mechanisms to select influential neighboring conditions of a node. Instead of attending the central node's current state with its neighbors,Äô concurrent states, I attend a node,Äôs current state over each of its neighbors,Äô historical states within a certain period of time. By building connections from each neighbor,Äôs past to the central node,Äôs present, a node,Äôs neighborhood information no matter in the past or in the present is traversed efficiently and effectively.

The main contributions of this chapter are as follows:

- I propose TraverseNet, a simple and powerful framework that captures the inner spatial-temporal dependencies without compromising space-time integrity.
- I propose a message traverse layer, effectively unifying space and time in message passing by traversing information of a node's neighbors' past to the node's present.
- I construct TraverseNet with message traverse layers and validate the significance of message traverse mechanism with an experimental study.

6.2 Background and Related Work to TraverseNet

6.2.1 Definitions and Notations

A graph $G = (V, E)$, where V is the set of nodes and E is the set of edges, is a mathematical description of structured data consisting of entities and relationships. I let $v \in V$ to represent a node and $e = (v, u) \in E$ to denote an directed edge from u to v. The neighborhood of a node v is the set of nodes $N(v) = \{u \in V | (v, u) \in E\}$ that points to node v. The adjacency matrix $\mathbf{A}$ is an another way that defines a graph with a dense matrix. It is an N by N matrix with $A_{ij} = 1$ if $(v_i, v_j) \in E$ and $A_{ij} = 0$ if $(v_i, v_j) \notin E$, where N is the number of nodes. I let $\mathbf{X}_t \in \mathbf{R}^{N \times D}$ to denote the node feature matrix at time step t.

The goal of spatial-temporal graph forecasting is to predict each node's future values given its historical sequential data and the topological graph structure. Formally, the

spatial-temporal graph forecasting problem is defined as finding a mapping from the input values to the target values:

$$[\mathbf{X}_{t_0}, \mathbf{X}_{t_1}, \cdots, \mathbf{X}_{t_p}; G] \xrightarrow{f} [\mathbf{X}_{t_{p+1}}, \mathbf{X}_{t_{p+2}}, \cdots \mathbf{X}_{t_{p+q}}]. \tag{6.1}$$

6.2.2 Related work

Standard graph neural networks assume that the input node features are static and only consider spatial information flow. When the node features dynamically change over time, a group of methods under the name of *spatial-temporal graph neural networks* can handle the data more effectively. I divide existing spatial-temporal graph neural networks into three categories: recurrent-based methods, convolution-based methods, and attention-based methods.

6.2.2.1 Recurrent-based STGNNs

Recurrent-based STGNNs simply assume a node,Äôs current hidden state depends on its own current inputs, its neighbors,Äô current inputs, its own previous hidden states and its neighbors,Äô previous hidden states [13, 19, 57]. The form of recurrent-based approaches can be conceptualized as

$$\mathbf{H}_t = RNN(GCN([\mathbf{X}_t, \mathbf{H}_{t-1}], \mathbf{A}; \Theta); \mathbf{U}) \tag{6.2}$$

where Θ and $\mathbf{U}$ are model parameters, and $\mathbf{H}_{t-1}$ represents the nodes' previous hidden state matrix. The previous hidden state of a node is essentially a memory vector of its historical information. As the number of recurrent steps increases, the memory vector will gradually forget information many steps before.

Another drawback of recurrent-based STGNNs is the high computation cost induced by recurrent propagation accompanied by graph convolution. Many recent studies follow a convolution-based approach.

6.2.2.2 Convolution-based STGNNs

Convolution-based STGNNs take advantage of the efficiency and shift-invariance property of convolutional neural networks [80, 193, 238]. They interleave temporal convolutions with graph convolutions to handle temporal correlations and spatial dependencies respectively. The core difference to recurrent-based methods is that they replace recurrent neural networks with temporal convolution networks (TCN) for capturing temporal

patterns, illustrated as the following,

$$\mathbf{Z} = GCN(TCN(\|_{t=1}^{T} \mathbf{X}_t; \Theta), \mathbf{A}; \mathbf{U}) \tag{6.3}$$

where $\|$ represents concatenation, $\mathbf{Z} \in R^{N \times D \times (T-c+1)}$, T is the sequence length, and c is the kernel size of the temporal convolution. The information flow of an STGNN layer occurs first temporally then spatially. Such models are against nature, where an object moves in space and time simultaneously.

6.2.2.3 Attention-based STGNNs

Similar to convolution-based approaches, attention-based STGNNs treat spatial dependencies and temporal correlations in separate steps [225, 236, 237],

$$\mathbf{Z} = SA(\|_{t=1}^{T} TA(\mathbf{X}_t; \Theta), \mathbf{A}); \mathbf{U}) \tag{6.4}$$

where $SA(\cdot)$ is a spatial attention layer and $TA(\cdot)$ is an temporal attention layer. The motivation of attention-based methods is that node spatial dependency and temporal dependency could be dynamically changing over time. The spatial attention layer first updates the graph adjacency matrix by computing the distance between a query node's input and a key node's input, then performs message passing. The temporal convolution layer computes a weighted sum of a node's historical state based on attention scores.

6.2.2.4 Other relevant works.

Song et al. [239] propose a localized spatial-temporal graph convolution network (STSGCN) that synchronously capture consecutive local spatial-temporal correlations. Li et al. [240] propose the Spatial-Temporal Fusion Graph Neural Networks (STFGNN) that considers pre-defined spatial dependencies, pre-computed time series similarities, and local temporal dependencies. Both STSGCN and STFGNN are not efficient to transfer the historical information of a node's neighbor to the node itself due to local connections and fixed dependency weights.

6.3 TraverseNet

To enable the direct flow of information both in space and time, I propose a novel spatial-temporal graph neural network named TraverseNet. The proposed design of

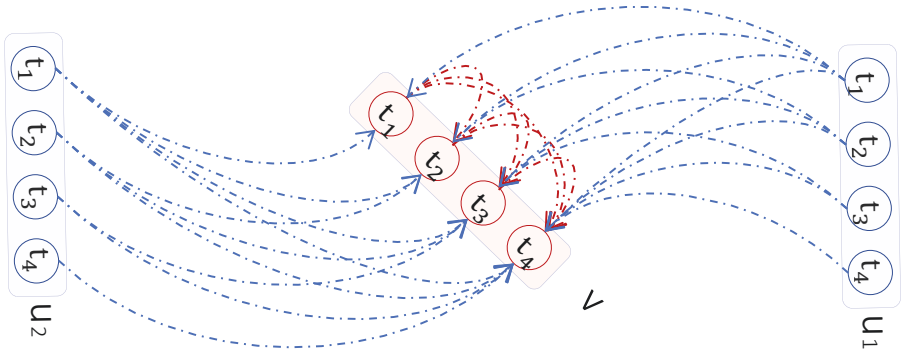


Figure 6.2: A demonstration of message traverse layer. The node v has two neighbors u_1 and u_2. Each node has four consecutive states at t_1, t_2, t_3, t_4. The message traverse layer allows the node v at a certain time step to receive information from its own previous time steps as well as its neighbors' previous time steps.

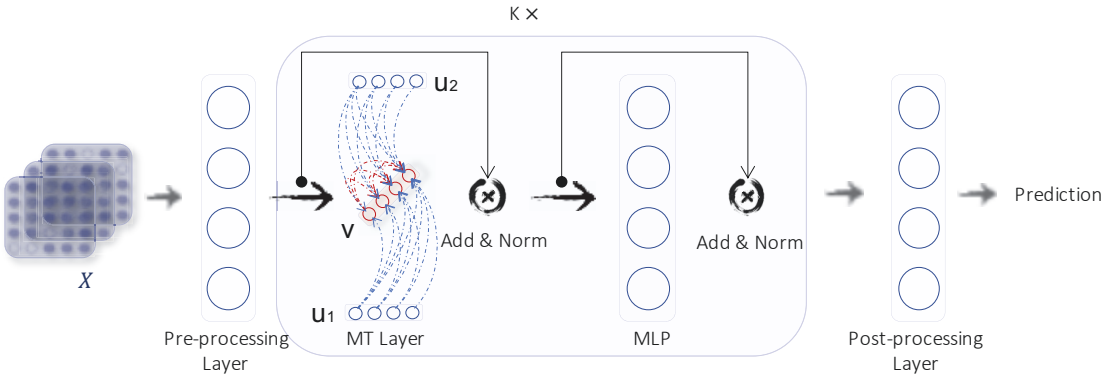


Figure 6.3: The model framework of TraverseNet. The TraverseNet mainly consists of three parts, the pre-processing layer, the message traverse layer, and the post-processing layer. The inputs is a sequence of node feature matrix $\mathbf{X}_{t_1}, \mathbf{X}_{t_2}, \cdots, \mathbf{X}_{t_p}$. The pre-processing MLP layer projects the input feature matrices to a latent feature space. The message traverse layer propagates information across space and time. The post-processing layer projects the nodes hidden states to the output space.

TraverseNet is simple. Apart from multi-layer perceptrons (MLPs), TraverseNet only contains my newly proposed message traverse layers. In the following, I introduce the message traverse layer and present the model framework of TraverseNet.

6.3.1 Message Traverse Layer

Conventional graph convolution layers or message passing layers only pass nodes' neighborhood information across space, ignoring temporal dynamics. I propose the message

traverse layer, a message passing layer that allows node neighborhood information to be simultaneously delivered across space and time. As demonstrated by Figure 6.2, node v has two neighbors u_1 and u_2. States of u_1 prior to t_4 can traverse to the state of node v at t_4 directly. This is in fundamental contrast to existing spatial-temporal works, where node v at t_4 can only receive information of node u_1 and u_2 from their concurrent time step t_4. The spatial-temporal dependency between two nodes may change from time to time depending on their states. I further leverage the attention mechanism to select important neighborhood information and to handle dynamic spatial-temporal dependencies. Formally, the message traverse layer updates the state of node v for each time step from $t = 0$ to $t = p - 1$ by

$$\mathbf{h}_{v_t}^{(k)} = \sum_{u \in N(v) \cup v} \alpha_r^{(k)}(\mathbf{c}_{(v \to v)_t}^{(k)}, \mathbf{c}_{(u \to v)_t}^{(k)}) \mathbf{W}_s^{(k)} \mathbf{c}_{(u \to v)_t}^{(k)}, \tag{6.5}$$

where $\mathbf{h}_{v_t}^0 = \mathbf{x}_{v_t}$, k denote the index of layers, and $\sigma(\cdot)$ denote an activation function. The function $\alpha_r^{(k)}(\cdot, \cdot)$ is an attention function of the form

$$\alpha_r^{(k)}(\mathbf{z}_q, \mathbf{z}_o) = \frac{exp(\sigma(\gamma^{(k)^T}[\Theta_{iq}^{(k)} \mathbf{z}_q || \Theta_{io}^{(k)} \mathbf{z}_o]))}{\sum_{o \in S} exp(\sigma(\gamma^{(k)^T}[\Theta_{iq}^{(k)} \mathbf{z}_q || \Theta_{io}^{(k)} \mathbf{z}_o]))}, \tag{6.6}$$

where Θ, $\mathbf{W}$, and γ represent model parameters. The term $\mathbf{c}_{(v \to v)_t}^{(k)}$ represents the latent information received by node v from its own time steps prior to time t. It is calculated as a weighted sum of its own historical states

$$\mathbf{c}_{(v \to v)_t}^{(k)} = \sum_{m=0}^{Q} \alpha_c^{(k)}(\mathbf{h}_{v_t}^{(k-1)}, \mathbf{h}_{v_{t-m}}^{(k-1)}) \mathbf{W}_c^{(k)} \mathbf{h}_{v_{t-m}}^{(k-1)}, \tag{6.7}$$

The term $\mathbf{c}_{(u \to v)_t}^{(k)}$ denotes the latent information received by node v from its neighbor u's previous time steps prior to time t. To assess the importance of each state of neighbor u, I involve the state of node v at the current time step as a query in the attention function

$$\mathbf{c}_{(u \to v)_t}^{(k)} = \sum_{m=0}^{Q} \alpha_e^{(k)}(\mathbf{h}_{v_t}^{(k-1)}, \mathbf{h}_{u_{t-m}}^{(k-1)}) \mathbf{W}_e^{(k)} \mathbf{h}_{u_{t-m}}^{(k-1)}. \tag{6.8}$$

The attention functions $\alpha_c^{(k)}(\cdot, \cdot)$ and $\alpha_e^{(k)}(\cdot, \cdot)$ have the same form as $\alpha_r^{(k)}(\cdot, \cdot)$. The hyper-parameter Q controls the time-window size within which a node receives information from its neighbor's past states. I differentiate the node itself from its neighborhood set because the node's own information has a decisive influence on its predictions for sequence forecasting. The computation complexity of the proposed message traverse

layer is $O(M \times p \times Q)$, where M denotes the number of edges including self-loops and p is the input sequence length.

The message traverse layer is a generalization of both spatial attention layers and temporal attention layers. Specifically, if the neighborhood set of the node v is empty, then Equation 6.5 reduces to a **temporal attention layer**

$$\mathbf{h}_{v_t}^{(k)} = \mathbf{W}_s^{(k)} \sum_{m=0}^{Q} \alpha_c^{(k)}(\mathbf{h}_{v_t}^{(k-1)}, \mathbf{h}_{v_{t-m}}^{(k-1)}) \mathbf{W}_c^{(k)} \mathbf{h}_{v_{t-m}}^{(k-1)}. \tag{6.9}$$

Alternatively, if the window size Q is set to 0, then Equation 6.5 becomes a **spatial attention layer**

$$\mathbf{h}_{v_t}^{(k)} = \sum_{u \in N(v) \cup v} \alpha_r^{(k)}(\mathbf{c}_{(v \to v)_t}^{(k)}, \mathbf{c}_{(u \to v)_t}^{(k)}) \mathbf{W}_s^{(k)} \mathbf{c}_{(u \to v)_t}^{(k)}, \tag{6.10}$$

where

$$\mathbf{c}_{(v \to v)_t}^{(k)} = \mathbf{W}_c^{(k)} \mathbf{h}_{v_t}^{(k-1)} \tag{6.11}$$

$$\mathbf{c}_{(u \to v)_t}^{(k)} = \mathbf{W}_e^{(k)} \mathbf{h}_{u_t}^{(k-1)}. \tag{6.12}$$

In contrary to existing works that interleave spatial computations with temporal computations, the proposed message traverse layer handles spatial-temporal dependency as a whole. The message traverse layer can not be separated into a spatial attention layer and a temporal attention layer. It is mainly because I assume the spatial-temporal dependency between a node's state at time step t and its neighbor's state at time step $t-m$ is dynamic. I use the state of the central node v at time step t as a query to assess the importance of its neighboring node u's historical states at each time step. Overall this design shortens the path length of message passing and enables a node to be aware of its neighborhood variation at firsthand.

6.3.2 Model Framework

As the message traverse layer is sufficient to capture spatial-temporal dependencies, I design a framework named TraverseNet that is simple and powerful to accomplish the spatial-temporal graph forecasting task. In Figure 6.3, I present the framework of TraverseNet. The TraverseNet consists of three parts, the pre-processing layer, a stack of message traverse layers, and the post-processing layer. The pre-processing layer is a feedforward layer which projects node feature matrix at each time step to a latent space. Next I capture nodes' spatial-temporal dependencies by message traverse layers. Finally,

I use the post-processing layer to map node hidden states to the output space. The post-processing layer contains a $1 \times p$ standard convolutional layer followed by a feedforward layer. Suppose the inputs of node v to the post-processing layer is $\mathbf{z}_v = \|_{i=0}^{p-1} \mathbf{h}'^{(K-1)}_{v_i}$, where $\mathbf{z}_v \in \mathbf{R}^{d \times 1 \times p}$, and p is the input sequence length. The $1 \times p$ standard convolutional layer is used to squeeze the third dimension of the inputs $\mathbf{z}_v$ to 1. Afterward, the feedforward layer is applied to generate the prediction $\mathbf{z}'_v \in \mathbf{R}^q$ for node v, where q is the output sequence length. In addition, residual connections and batch normalization are applied to message traverse layers to improve model robustness. In particular, as the inputs of each node may have very different scales, I let the batch normalization scale the hidden features on the node dimension.

6.3.3 Optimization & Implementation

I optimize model parameters of TraverseNet end-to-end by minimizing the Mean Absolute Error (MAE) loss with gradient descent. The MAE is defined as

$$L = \text{Average}\left(\sum_{i=p+1}^{p+q} |\mathbf{X}_{t_i} - \hat{\mathbf{X}}_{t_i}| \right). \tag{6.13}$$

I implement TraverseNet with Pytorch and DGL [137]. In more detail, I first construct a heterogeneous graph by treating each node at each time step as a unique node and creating connections as illustrated by Figure 6.2. The state of each node at a certain time step is linked to its historical states as well its neighbors' historical states within a time window Q. The constructed graph is in a sparse form thus efficient for computation. I implement the message traverse layer by customizing the HeterGraphConv module of DGL. The code is publicly available at https://github.com/nnzhan/TraverseNet.

6.4 Experimental Studies

6.4.1 Dataset

I follow the experimental setup in [239]. I use three traffic datasets, PEMS-03, PEMS-04, and PEMS-08, in my experiments. These datasets contain traffic signals of road sensors aggregated every five minutes collected by the Caltrans Performance Measurement Systems in different districts of California. I provide summary statistics of each dataset in Table 6.1. Two tasks, i.e. traffic flow prediction and traffic speed prediction, are evaluated using these datasets. I predict the next twelve steps traffic speed/flow given

Table 6.1: Dataset statistics.

Datasets	# Sensors	Sampling rate	# Time steps	Signals
PEMS03	358	5 mins	26209	F
PEMS04	307	5 mins	16992	F,S,O
PEMS08	170	5 mins	17856	F,S,O

In column titled “Signals”, F represents traffic flow, S represents traffic speed, and O represents traffic occupancy rate.

the previous twelve steps of traffic signals and the traffic graph. I construct the traffic graph by regarding each sensor as a node and connecting two sensors if they are on the same road. As data pre-processing, I standardize the inputs to have zero mean and unit variance by

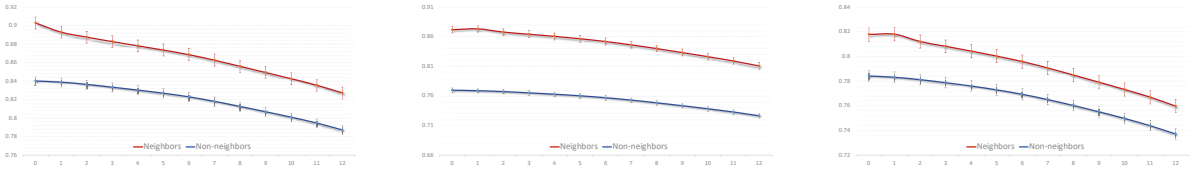
$$\tilde{\mathbf{X}} = \frac{X - \text{mean}(X)}{\text{std}(X)}. \tag{6.14}$$

To check if the datasets exhibit spatial-temporal dependencies, I plot time lags v.s. cross-correlations of pairs of connected nodes and pairs of far-away nodes (i.e. more than nine hops away) for each dataset respectively.

The cross-correlation between a sequence $\{x_1, x_2, \cdots, x_L\}$ and a sequence $\{y_1, y_2, \cdots, y_L\}$ at time lag k is essentially the correlation between the sequence $\mathbf{y}$ and the sequence $\mathbf{x}$ shifted k steps back:

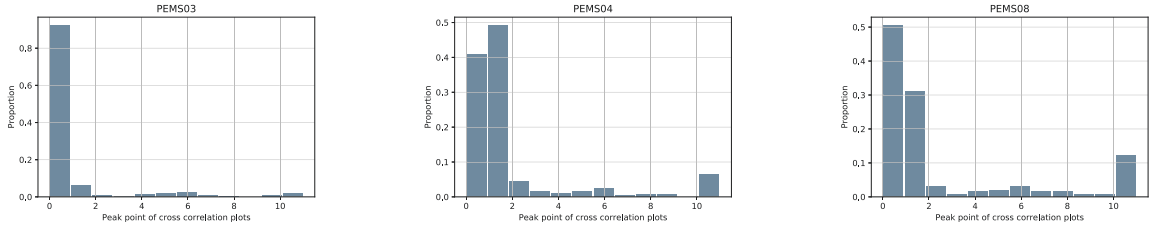
$$C = \frac{\frac{1}{L}\sum_{t=k}^{L} x_{t-k} y_t - \frac{1}{L^2}\sum_{t=k}^{L} x_{t-k} \sum_{t=k}^{L} y_t}{\sqrt{\frac{1}{L}\sum_{t=k}^{L} x_{t-k}^2 - (\frac{1}{L}\sum_{t=k}^{L} x_{t-k})^2}\sqrt{\frac{1}{L}\sum_{t=k}^{L} y_t^2 - (\frac{1}{L}\sum_{t=k}^{L} y_t)^2}} \tag{6.15}$$

Figure 6.4 and Figure 6.5 presents my analysis. In Figure 6.4a, 6.4b and 6.4c, it shows that the cross-correlations between pairs of connected nodes are always higher than the cross-correlations between pairs of far-away nodes across all time lags and datasets. Dig into detail, I plot the distribution of peak points of cross-correlation curves between pairs of connected nodes for each datasets, as shown by Figure 6.5a, 6.5b, 6.5c. The majority of two connected nodes,Äô cross-correlations peak at time lag 0 and 1. However, there is still a small a amount of nodes of which the cross-correlation values peak at higher time lags. In particular, for PEMS-04 and PEMS-08, there are 5% and 10% of connected nodes of which the cross-correlation peak at time 11. This is suggests that it is reasonable to consider spatial-temporal dependencies.



(a) PEMS-03 (b) PEMS-04 (c) PEMS-08

Figure 6.4: Cross-correlations between pairs of connected nodes and between pairs of far-away nodes. The x-axis is the time lag. The y-axis is the mean of correlation coefficients with standard deviation.



(a) PEMS-03 (b) PEMS-04 (c) PEMS-08

Figure 6.5: The distribution of peak points of cross-correlation plots between pairs of connected nodes. The x-axis is the time lag. The y-axis is the proportion.

Table 6.2: Performance comparison.

Task	Models	PEMS-03			PEMS-04			PEMS-08		
		MAE	MAPE	RMSE	MAE	MAPE	RMSE	MAE	MAPE	RMSE
Traffic Flow	GRU	20.01 ± 0.02	19.82 ± 0.06	32.52 ± 0.10	24.84 ± 0.04	16.98 ± 0.09	38.87 ± 0.08	18.86 ± 0.05	12.07 ± 0.05	30.25 ± 0.05
	DCRNN	16.37 ± 0.05	**15.96 ± 0.10**	28.37 ± 0.33	24.85 ± 0.12	16.94 ± 0.16	38.95 ± 0.14	17.82 ± 0.13	11.39 ± 0.10	27.69 ± 0.16
	STGCN	19.58 ± 0.15	20.17 ± 0.36	32.57 ± 0.85	23.96 ± 0.07	17.40 ± 0.58	36.94 ± 0.14	18.75 ± 0.15	13.00 ± 0.44	28.49 ± 0.10
	ASTGCN	18.19 ± 0.22	18.47 ± 0.54	30.58 ± 0.48	22.91 ± 0.44	16.96 ± 0.54	35.60 ± 0.75	18.74 ± 0.41	12.23 ± 0.30	28.80 ± 0.72
	Graph WaveNet	16.74 ± 0.05	18.56 ± 1.66	27.75 ± 0.13	20.95 ± 0.09	14.55 ± 0.17	32.64 ± 0.11	**15.66 ± 0.08**	**10.31 ± 0.11**	**24.59 ± 0.12**
	STSGCN	17.77 ± 0.20	17.28 ± 0.06	28.93 ± 0.34	22.61 ± 0.07	14.90 ± 0.05	35.15 ± 0.13	17.92 ± 0.14	11.60 ± 0.14	27.48 ± 0.21
	STFGNN	16.56 ± 0.32	16.09 ± 0.16	28.60 ± 0.23	21.47 ± 0.10	**14.10 ± 0.08**	33.57 ± 0.11	17.75 ± 0.15	11.23 ± 0.09	27.64 ± 0.23
	TraverseNet	**15.44 ± 0.10**	16.41 ± 0.87	**24.75 ± 0.32**	**19.86 ± 0.11**	14.38 ± 0.79	**31.54 ± 0.28**	15.68 ± 0.12	10.87 ± 0.05	24.62 ± 0.13
Traffic Speed	GRU	-	-	-	2.34 ± 0.07	5.03 ± 0.08	5.09 ± 0.02	1.80 ± 0.03	3.59 ± 0.06	3.99 ± 0.04
	DCRNN	-	-	-	2.24 ± 0.06	5.05 ± 0.33	4.89 ± 0.18	1.72 ± 0.04	3.75 ± 0.15	3.75 ± 0.09
	STGCN	-	-	-	1.80 ± 0.01	3.87 ± 0.03	4.09 ± 0.04	1.52 ± 0.01	3.28 ± 0.05	3.65 ± 0.05
	ASTGCN	-	-	-	1.78 ± 0.02	3.87 ± 0.12	3.97 ± 0.12	1.51 ± 0.04	3.41 ± 0.10	3.67 ± 0.12
	Graph WaveNet	-	-	-	1.61 ± 0.00	3.39 ± 0.02	3.71 ± 0.01	**1.34 ± 0.00**	**2.97 ± 0.05**	**3.36 ± 0.04**
	STSGCN	-	-	-	1.96 ± 0.06	4.28 ± 0.16	4.30 ± 0.09	1.73 ± 0.07	3.80 ± 0.21	3.87 ± 0.15
	STFGNN	-	-	-	1.79 ± 0.03	3.89 ± 0.08	4.03 ± 0.05	1.54 ± 0.01	3.36 ± 0.02	3.62 ± 0.02
	TraverseNet	-	-	-	**1.59 ± 0.00**	**3.37 ± 0.02**	**3.67 ± 0.01**	1.35 ± 0.01	3.02 ± 0.05	3.44 ± 0.04

Results of best-performing method is shown in bold font.

6.4.2 Baseline Methods

Seven baseline methods are selected in my experiments. Except STSGCN and STFGNN, I implement all baseline methods in a unified framework. As it is difficult to merge STSGCN and STFGNN into my framework, I directly use the codes of STSGCN and STFGNN in experiments. I give a short description of each baseline method in the following:

- GRU a sequence-to-sequence model [241] consists of GRU units [85], not considering spatial dependency.
- DCRNN [13] that adopts LSTMs and diffusion graph convolution in a seq-to-seq framework.
- STGCN [20] that combines gated temporal convolution with graph convolution to capture spatial dependencies and temporal dependencies respectively.
- ASTGCN [225] that interleaves spatial attentions with temporal attentions to capture dynamic spatial dependencies and temporal dependencies.
- Graph WaveNet [80] that integrates WaveNet with graph convolution.
- STSGCN [239] that considers spatial-temporal dependencies in local adjacent time steps.
- STFGNN [240] that considers pre-defined spatial dependencies, pre-computed time series similarities, and local temporal dependencies.

6.4.3 Experimental Setting

I conduct the experiments on the AWS cloud with the p3.8xlarge instance. I train the proposed TraverseNet with the Adam optimizer on a single 16GB Tesla V100 GPU. I split the datasets into train, validation, and test data with a ratio of 6:2:2. I set the number of training epochs to 50, the learning rate to 0.001, the weight decay rate to 0.00001, and the dropout rate to 0.1. I set the number of layers to 3, the hidden feature dimension to 64, and the window size Q to 12. For other baseline methods, I use the default parameters settings reported in their papers. Each experiment is repeated 5 times and the mean of evaluation metrics including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE) on test data are reported based on the best model on validation data.

6.4.4 Overall Results

Table 6.2 presents the main experimental results of my TraverseNet compared with baseline methods. There are missing values on PEMS-03 for the traffic speed prediction because PEMS-03 does not contain traffic speed information. Among all methods, TraverseNet achieves the lowest MAE and RMSE on PEMS-03 and PEMS-04 for traffic flow

Table 6.3: Ablation study.

	w/o spatial info	w/o st traversing	w/o attention	w/o residual	w/o norm	default
MAE	15.95 ± 0.12	15.84 ± 0.07	15.69 ± 0.11	17.01 ± 0.02	15.76 ± 0.10	15.68 ± 0.12
MAPE	10.83 ± 0.20	11.05 ± 0.27	10.56 ± 0.13	13.31 ± 0.90	10.86 ± 0.19	10.87 ± 0.05
RMSE	24.99 ± 0.12	24.80 ± 0.07	24.66 ± 0.15	26.24 ± 0.26	24.82 ± 0.10	24.62 ± 0.13

The results were obtained on the PEMS-08 dataset.

prediction and on PEMS-04 for traffic speed prediction. It achieves the second lowest MAE, MAPE, and RMSE on PEMS-08 for both traffic flow prediction and traffic speed prediction–though the performance gap between the top two methods is extremely small.

I believe that the reason that Graph WaveNet performs slightly better than TraverseNet on PEMS-08 is that the spatial signal on that dataset is weak. This is supported by the results of the ablation study (cf. Table 6.3), which shows that the performance of TraverseNet decreases only slightly when the spatial component of the model is removed. For temporal patterns, the WaveNet component in Graph WaveNet is a very powerful feature extractor for time series data and this is why its performance is somewhat better than TraverseNet. Besides, TraverseNet significantly outperforms methods that consider local message traversing between adjacent time steps across two connected nodes including DCRNN, STSGCN, and STFGNN.

6.4.5 Ablation Study

I perform an ablation study to validate the effectiveness of the message traverse layer in my model. I am mainly concerned about whether the spatial-temporal message traverse layer is effective and whether the attention mechanism in the message traverse layer is useful. To answer these questions, I compare my TraverseNet model with five different settings listed below:

- **w/o spatial information.** I only use temporal information, which means the neighborhood set of a node is empty.

- **w/o st traversing (without spatial-temporal traversing).** I handle spatial dependencies and temporal dependencies separately by interleaving spatial attentions with temporal attentions.

- **w/o attention.** I replace attention scores produced by the attention functions with identical weights.

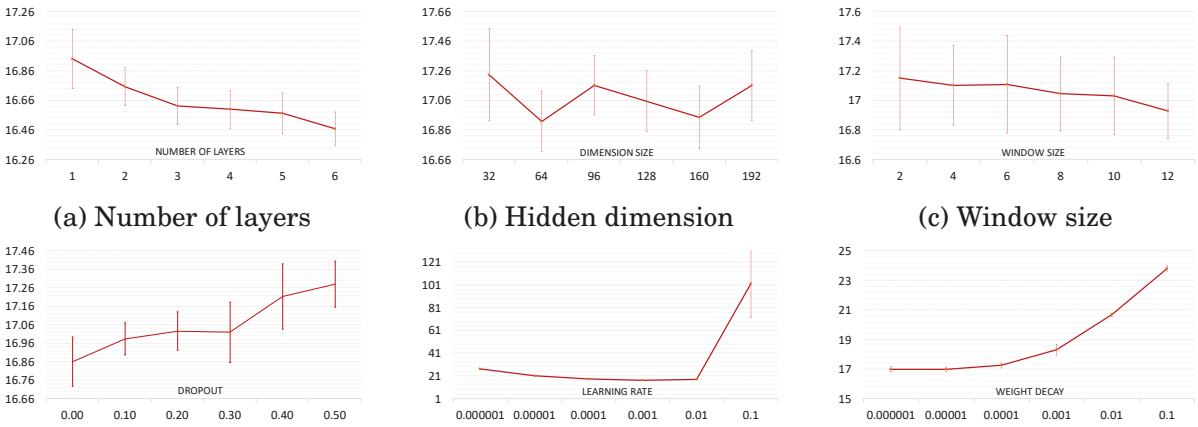
- **w/o residual.** I cancel residual connections for message traversing layers and MLP layers.

- **w/o norm.** I remove batch normalization after message traversing layers and MLP layers.

I repeat each experiment 5 times. Table 6.3 reports the mean MAE, MAPE, and RMSE with standard deviation on PEMS-08 test data. I observe that the involvement of spatial information incrementally contributes to model performance. More importantly, spatial-temporal traversing is superior to process spatial dependencies and temporal dependencies separately based on the fact the performance of **w/o st traversing** is lower than the performance of **default**. **W/o attention** nearly does not improve model performance, suggesting that the effect of attention mechanisms is limited in time series forecasting. Besides, according to Table 6.3, the effectiveness of residual connections and batch normalization is verified.

6.4.6 Hyperparameter Study

To get an understanding of the effect of key hyper-parameters in TraverseNet, I study the effect of varying one hyperparameter at a time whilst keeping others the same as Section 6.4.3 except that the default number of layers is set to 1 on validation set of PEMS-08. I vary the number of layers ranging from 1 to 6 by 1, the hidden feature dimension ranging from 32 to 192 by 32, the window size ranging from 2 to 12 by 2, the dropout rate ranging from 0 to 0.5 by 0.1, and both the learning rate and weight decay among $\{1 \times 10^{-6}, 1 \times 10^{-5}, 1 \times 10^{-4}, 1 \times 10^{-3}, 1 \times 10^{-2}, 1 \times 10^{-1}\}$. I run each experiment 5 times. The mean MAE with one standard deviation for each experiment is calculated. Figure 5.6 plots the trends of model performance for each hyper-parameter. As the number of layers or the window size increases, the model performance is gradually improved. Increasing the number of layers or the window size enlarges the receptive field of a node, thus a node can track longer and broader neighborhood history. The model performance is not sensitive to the change of hidden dimension. I think it may be due to the nature of time series data that the input dimension is thin so that a small hidden feature dimension is enough to capture original information. Dropout and weight decay are not necessary in my model since the model performance drops evidently when it increases. Besides, the optimal learning rate is 0.01.



(a) Number of layers (b) Hidden dimension (c) Window size

(d) Dropout rate (e) Learning rate (f) Weight decay

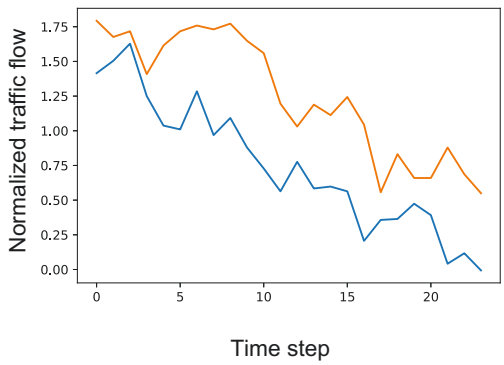
Figure 6.6: MAE plots for Parameter Study.

6.4.7 Case Study

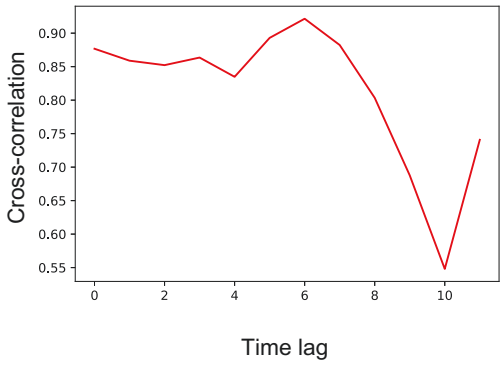
I perform a case study to understand the effect of TraverseNet in capturing inner spatial-temporal dependency. Figure 6.7a plots the time series of two neighboring nodes, node 15 and node 151 in PEMS-08. The blue line is the time series of the source node 15 and the yellow line is the time series of the target node 151. I observe that the trend of node 151 follows the trend of node 15 with some extent of latency. For example, the blue line of node 15 starts to drop sharply at step 2 while this phenomenon happens on the yellow line of node 151 6 steps later. Figure 6.7b shows that it is not always the case that the cross-correlation between time series of two neighboring nodes is the highest at time step 0. In fact, the time series of node 151 is mostly correlated with the time series of node 15 shifted 6 time steps. Figure 6.7c provides a heat-map which visualizes the attention scores produced by TraverseNet in the first message traverse layer between these two time series from time step 0 to time 12. It shows that the state of node 15 at time step 6, 7, and 8 is very important to the state of node 151 at time step 8, 9 and 10. This is consistent with the fact the trend of node 15 at time 6,7,8 is similar to the trend of node 151 at time 8,9,10 from Figure 6.7a.

6.4.8 Computation Time

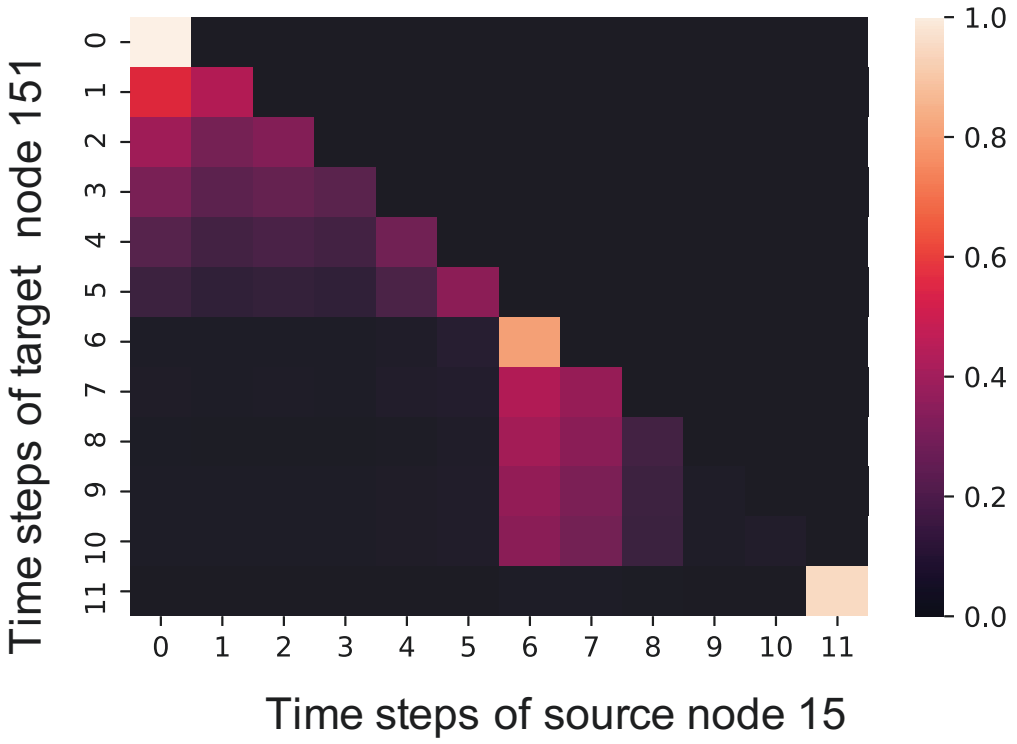
I compare the computation time of my method with baseline methods on PEMS04 and PEMS08 data in Table 6.4. The training speed of DCRNN is the slowest while the training speed of STGCN is the fastest. The running speed of my method stays in the middle. Although the time complexity of the message traversing layer is $O(M \times p \times Q)$,



(a) Time series of two connected nodes that contain inner spatial-temporal dependencies.



(b) Time lag v.s. cross-correlation of the two time series in (a).



(c) Heatmap of attention scores.

Figure 6.7: Case study.

Table 6.4: Comparison of running time.

Models	PEMS-04		PEMS-08	
	Training	Inference	Training	Inference
DCRNN	314.08 s/epoch	52.35 s	145.40 s/epoch	24.19 s
STGCN	10.89 s/epoch	0.73 s	6.96 s/epoch	0.45 s
ASTGCN	24.11 s/epoch	2.89 s	16.41 s/epoch	1.89 s
Graph WaveNet	26.39 s/epoch	1.59s	16.94 s/epoch	0.91 s
STSGCN	66.12 s/epoch	6.01 s	36.79 s/epoch	3.27 s
STFGNN	45.03 s/epoch	5.19 s	24.17 s/epoch	2.82 s
TraverseNet	56.51 s/epoch	5.93 s	39.57 s/epoch	4.49 s

the speed of my model is still affordable due to an efficient sparse implementation empowered by DGL.

6.5 Summary

In this chapter, I propose TraverseNet, a graph neural network that unifies space and time. The proposed TraverseNet processes a spatial-temporal graph as an inseparable whole. Through the proposed novel message traverse layers, information can be delivered from the neighbors,Äô past to the node,Äôs present directly. This design shortens the path length of message passing and enables a node to be aware of its neighborhood variation at firsthand. Experimental results validate the effectiveness of my framework. As the graph structure is determinant to a spatial-temporal forecasting model [193], I will consider improving the efficiency of the message traversing by refining the underlying explicit edge relationships in the future.

CHAPTER 7

FUTURE WORK

GNNs are proven to be powerful when there exist inter-dependencies in data. The study of GNNs is still in a preliminary stage due to the complexity, heterogeneity, and dynamicity. For future work, I think four directions are highly worth to investigating.

Model depth The success of deep learning lies in deep neural architectures [242]. However, Li et al. show that the performance of a GNN drops dramatically with an increase in the number of graph convolutional layers [62]. As graph convolutions push representations of adjacent nodes closer to each other, in theory, with an infinite number of graph convolutional layers, all nodes' representations will converge to a single point [62]. This also applies to STGNNs and raises the question of whether going deep is still a good strategy for learning graph data.

Scalability trade-off The scalability of GNNs is gained at the price of corrupting graph completeness. Whether using sampling or clustering, a model will lose part of the graph information. By sampling, a node may lose information about its influential neighbors. By clustering, a graph may be deprived of a distinct structural pattern. How to trade-off algorithm scalability and graph integrity could be a future research direction.

Heterogeneity The majority of current STGNNs assume homogeneous graphs. It is difficult to directly apply current STGNNs to heterogeneous graphs, which may contain different types of nodes and edges, or different forms of node and edge inputs. Therefore, new methods should be developed to handle heterogeneous graphs for spatial-temporal

modeling.

Dynamicity Graphs are in nature dynamic in a way that nodes or edges may appear or disappear, and that node/edge inputs may change time by time. New graph convolutions are needed to adapt to the dynamicity of graphs. Although the dynamicity of graphs can be partly addressed by STGNNs, few of them consider how to perform graph convolutions in the case of dynamic spatial relations and continuous dynamicity.

CHAPTER 8

CONCLUSION

The aim of this thesis is to study spatial-temporal data from the perspective of deep learning on graphs. I have studied the research objective in deep depth with four research questions: (1) How to coordinate the low, middle, and high frequency band of graph signals in graph convolution networks. (2) How to model spatial-temporal graph data effectively and efficiently; (3) How to handle spatial dependencies when a graph is totally missing, incomplete or inaccurate in spatial-temporal graph modeling; (4) In contrast to traditional spatial-temporal graph neural networks that handle spatial dependencies and temporal dependencies in separate, how to unify space and time as a whole in message passing.

To address the aforementioned four research problems, I proposed four algorithms or models that can achieve satisfactory results. Specifically, I proposed an Automatic Graph Convolutional Network for learning graph frequency bands for graph convolution filters automatically; I introduced an efficient and effective framework that integrates diffusion graph convolution and dilated temporal convolution to capture spatial-temporal dependencies simultaneously. I developed a novel joint-learning algorithm that can capture spatial-temporal dependencies and learn latent graph structures at the same time; I designed a unified graph neural network that captures the inner spatial-temporal dependencies without compromising space-time integrity. To validate the proposed methods, I have conducted experiments on real-world datasets with a range of tasks including node classification, graph classification, and spatial-temporal graph forecasting. Experimental results demonstrate the effectiveness of the proposed methods.

APPENDIX A

A.1 Data Set

Citation Networks consist of papers, authors, and their relationships such as citations, authorship, and co-authorship. Although citation networks are directed graphs, they are often treated as undirected graphs in evaluating model performance with respect to node classification, link prediction, and node clustering tasks. There are three popular data sets for paper-citation networks, Cora, Citeseer and Pubmed. The Cora data set contains 2708 machine learning publications grouped into seven classes. The Citeseer data set contains 3327 scientific papers grouped into six classes. Each paper in Cora and Citeseer is represented by a one-hot vector indicating the presence or absence of a word from a dictionary. The Pubmed data set contains 19717 diabetes-related publications. Each paper in Pubmed is represented by a term frequency-inverse document frequency (TF-IDF) vector. Furthermore, DBLP is a large citation data set with millions of papers and authors which are collected from computer science bibliographies. The raw data set of DBLP can be found on `https://dblp.uni-trier.de`. A processed version of the DBLP paper-citation network is updated continuously by `https://aminer.org/citation`.

Biochemical Graphs Chemical molecules and compounds can be represented by chemical graphs with atoms as nodes and chemical bonds as edges. This category of graphs is often used to evaluate graph classification performance. The NCI-1 and NCI-9 data set contain 4110 and 4127 chemical compounds respectively, labeled as to whether they are

active to hinder the growth of human cancer cell lines. The MUTAG data set contains 188 nitro compounds, labeled as to whether they are aromatic or heteroaromatic. The D&D and PROTEIN data set represent proteins as graphs, labeled as to whether they are enzymes or non-enzymes. The PTC data set consists of 344 chemical compounds, labeled as to whether they are carcinogenic for male and female rats. The QM9 data set records 13 physical properties of 133885 molecules with up to 9 heavy atoms. The Alchemy data set records 12 quantum mechanical properties of 119487 molecules comprising up to 14 heavy atoms. Another important data set is the Protein-Protein Interaction network (PPI). It contains 24 biological graphs with nodes represented by proteins and edges represented by the interactions between proteins. In PPI, each graph is associated with one human tissue. Each node is labeled with its biological states.

Social Networks are formed by user interactions from online services such as BlogCatalog and Reddit. The BlogCatalog data set is a social network which consists of bloggers and their social relationships. The classes of bloggers represent their personal interests. The Reddit data set is an undirected graph formed by posts collected from the Reddit discussion forum. Two posts are linked if they contain comments by the same user. Each post has a label indicating the community to which it belongs.

Others There are several other data sets worth mentioning. The MNIST data set contains 70000 images of size 28×28 labeled with ten digits. An MNINST image is converted to a graph by constructing an 8-nearest-neighbors graph based on its pixel locations. The METR-LA is a spatial-temporal graph data set. It contains four months of traffic data collected by 207 sensors on the highways of Los Angeles County. The adjacency matrix of the graph is computed by the sensor network distance with a Gaussian threshold. The NELL data set is a knowledge graph obtained from the Never-Ending Language Learning project. It consists of facts represented by a triplet which involves two entities and their relation.

A.2 Reported Experimental Results for Node Classification

A summarization of experimental results of methods which follow a standard train/-valid/test split is given in Table A.1.

Table A.1: Reported experimental results for node classification on five frequently used data sets. Cora, Citeseer, and Pubmed are evaluated by classification accuracy. PPI and Reddit are evaluated by micro-averaged F1 score.

Method	Cora	Citeseer	Pubmed	PPI	Reddit
SSE (2018)	-	-	-	83.60	-
GCN (2016)	81.50	70.30	79.00	-	-
Cayleynets (2017)	81.90	-	-	-	-
DualGCN (2018)	83.50	72.60	80.00	-	-
GraphSage (2017)	-	-	-	61.20	95.40
GAT (2017)	83.00	72.50	79.00	97.30	-
MoNet (2017)	81.69	-	78.81	-	-
LGCN (2018)	83.30	73.00	79.50	77.20	-
GAAN (2018)	-	-	-	98.71	96.83
FastGCN (2018)	-	-	-	-	93.70
StoGCN (2018)	82.00	70.90	78.70	97.80	96.30
Huang et al. (2018)	-	-	-	-	96.27
GeniePath (2019)	-	-	78.50	97.90	-
DGI (2018)	82.30	71.80	76.80	63.80	94.00
Cluster-GCN (2019)	-	-	-	99.36	96.60

A.3 Open-source Implementations

Here I summarize the open-source implementations of graph neural networks reviewed in the survey. I provide the hyperlinks of the source codes of the GNN models in table A.2.

Table A.2: A Summary of Open-source Implementations

Model	Framework	Github Link
GGNN (2015)	torch	https://github.com/yujiali/ggnn
SSE (2018)	c	https://github.com/Hanjun-Dai/steady_state_embedding
ChebNet (2016)	tensorflow	https://github.com/mdeff/cnn_graph
GCN (2017)	tensorflow	https://github.com/tkipf/gcn
CayleyNet (2017)	tensorflow	https://github.com/amoliu/CayleyNet.
DualGCN (2018)	theano	https://github.com/ZhuangCY/DGCN
GraphSage (2017)	tensorflow	https://github.com/williamleif/GraphSAGE
GAT (2017)	tensorflow	https://github.com/PetarV-/GAT
LGCN (2018)	tensorflow	https://github.com/divelab/lgcn/
PGC-DGCNN (2018)	pytorch	https://github.com/dinhinfotech/PGC-DGCNN
FastGCN (2018)	tensorflow	https://github.com/matenure/FastGCN
StoGCN (2018)	tensorflow	https://github.com/thu-ml/stochastic_gcn
DGCNN (2018)	torch	https://github.com/muhanzhang/DGCNN
DiffPool (2018)	pytorch	https://github.com/RexYing/diffpool
DGI (2019)	pytorch	https://github.com/PetarV-/DGI
GIN (2019)	pytorch	https://github.com/weihua916/powerful-gnns
Cluster-GCN (2019)	pytorch	https://github.com/benedekrozemberczki/ClusterGCN
DNGR (2016)	matlab	https://github.com/ShelsonCao/DNGR
SDNE (2016)	tensorflow	https://github.com/suanrong/SDNE
GAE (2016)	tensorflow	https://github.com/limaosen0/Variational-Graph-Auto-Encoders
ARVGA (2018)	tensorflow	https://github.com/Ruiqi-Hu/ARGA
DRNE (2016)	tensorflow	https://github.com/tadpole/DRNE
GraphRNN (2018)	tensorflow	https://github.com/snap-stanford/GraphRNN
MolGAN (2018)	tensorflow	https://github.com/nicola-decao/MolGAN
NetGAN (2018)	tensorflow	https://github.com/danielzuegner/netgan
GCRN (2016)	tensorflow	https://github.com/youngjoo-epfl/gconvRNN
DCRNN (2018)	tensorflow	https://github.com/liyaguang/DCRNN
Structural RNN (2016)	theano	https://github.com/asheshjain399/RNNexp
CGCN (2017)	tensorflow	https://github.com/VeritasYin/STGCN_IJCAI-18
ST-GCN (2018)	pytorch	https://github.com/yysijie/st-gcn
GraphWaveNet (2019)	pytorch	https://github.com/nnzhan/Graph-WaveNet
ASTGCN (2019)	mxnet	https://github.com/Davidham3/ASTGCN

APPENDIX B

APPENDIX B

B.1 Derivation of the graph convolutional kernels of low-pass, high-pass, and middle filter functions.

Definition 7 (Low-pass linear filter)**.** *The low-pass linear filter function is defined as*

$$F_{low}(\boldsymbol{\lambda}) = p(1 - a\boldsymbol{\lambda}), \tag{B.1}$$

with $p > 0$ *and* $a \in (0, 1)$*.*

The graph convolutional kernel of the low-pass filter is then derived as

$$\mathbf{C}_{low} = \mathbf{U} diag(F_{low}(\boldsymbol{\lambda}))\mathbf{U}^T \tag{B.2}$$

$$= \mathbf{U} diag(p(1 - a\boldsymbol{\lambda})\mathbf{U}^T \tag{B.3}$$

$$= p(\mathbf{I} - a\mathbf{L}) \tag{B.4}$$

$$= p(a\tilde{\mathbf{A}} + (1 - a)\mathbf{I}) \tag{B.5}$$

Definition 8 (High-pass linear filter)**.** *The high-pass linear filter function is defined as*

$$F_{high}(\boldsymbol{\lambda}) = p(a\boldsymbol{\lambda} + 1 - 2a). \tag{B.6}$$

with $p > 0$ *and* $a \in (0, 1)$*.*

The graph convolutional kernel of the high-pass filter is then derived as

$$\mathbf{C}_{high} = \mathbf{U} diag(F_{high}(\boldsymbol{\lambda}))\mathbf{U}^T \tag{B.7}$$

$$= \mathbf{U} diag(p(a\boldsymbol{\lambda} + 1 - 2a))\mathbf{U}^T \tag{B.8}$$

$$= p(a\mathbf{L} + \mathbf{I} - 2a\mathbf{I}) \tag{B.9}$$

$$= p(-a\tilde{\mathbf{A}} + (1-a)\mathbf{I}) \tag{B.10}$$

Definition 9 (Middle-pass quadractic filter)**.** *The middle-pass quadractic filter function is defined as*

$$F_{mid}(\boldsymbol{\lambda}) = p((\boldsymbol{\lambda} - 1)^2 - a). \tag{B.11}$$

with $p > 0$ *and* $a \in (0, 1]$.

The graph convolutional kernel of the high-pass filter is then derived as

$$\mathbf{C}_{mid} = \mathbf{U} diag(F_{mid}(\boldsymbol{\lambda}))\mathbf{U}^T \tag{B.12}$$

$$= \mathbf{U} diag(p((\boldsymbol{\lambda} - 1)^2 - a))\mathbf{U}^T \tag{B.13}$$

$$= p(\mathbf{U}(diag(\boldsymbol{\lambda} - 1))^2\mathbf{U}^T - a\mathbf{I}) \tag{B.14}$$

(B.15)

As $\mathbf{U}^T\mathbf{U} = \mathbf{I}$*,*

$$\mathbf{C}_{mid} = p(\mathbf{U} diag(\boldsymbol{\lambda} - 1)\mathbf{U}^T\mathbf{U} diag(\boldsymbol{\lambda} - 1)\mathbf{U}^T - a\mathbf{I}) \tag{B.16}$$

$$= p((\mathbf{L} - \mathbf{I})^2 - a\mathbf{I}) \tag{B.17}$$

$$= p(\tilde{\mathbf{A}}^2 - a\mathbf{I}) \tag{B.18}$$

BIBLIOGRAPHY

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. of ICLR*, 2017.

[2] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. of ICLR*, 2017.

[3] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proc. of KDD*. ACM, 2019.

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. of CVPR*, 2016, pp. 779–788.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Proc. of NIPS*, 2015, pp. 91–99.

[6] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proc. of EMNLP*, 2015, pp. 1412–1421.

[7] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[8] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[9] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

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

[11] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of machine learning research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[12] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[13] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *Proc. of ICLR*, 2018.

[14] H. Yao, F. Wu, J. Ke, X. Tang, Y. Jia, S. Lu, P. Gong, J. Ye, and Z. Li, "Deep multi-view spatial-temporal network for taxi demand prediction," in *Proc. of AAAI*, 2018, pp. 2588–2595.

[15] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Proc. of AAAI*, 2018.

[16] C. Yu, X. Ma, J. Ren, H. Zhao, and S. Yi, "Spatio-temporal graph transformer networks for pedestrian trajectory prediction," in *European Conference on Computer Vision*. Springer, 2020, pp. 507–523.

[17] J. Yang, W.-S. Zheng, Q. Yang, Y.-C. Chen, and Q. Tian, "Spatial-temporal graph convolutional network for video-based person re-identification," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 3289–3299.

[18] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-rnn: Deep learning on spatio-temporal graphs," in *Proc. of CVPR*, 2016, pp. 5308–5317.

[19] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *Proc. of NeurIPS*. Springer, 2018, pp. 362–373.

[20] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *Proc. of IJCAI*, 2018, pp. 3634–3640.

[21] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," in *Proc. of NIPS*, 2017, pp. 1024–1034.

[22] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[23] J. B. Lee, R. A. Rossi, S. Kim, N. K. Ahmed, and E. Koh, "Attention models in graphs: A survey," *arXiv preprint arXiv:1807.07984*, 2018.

[24] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 714–735, 1997.

[25] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proc. of IJCNN*, vol. 2. IEEE, 2005, pp. 729–734.

[26] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[27] C. Gallicchio and A. Micheli, "Graph echo state networks," in *IJCNN*. IEEE, 2010, pp. 1–8.

[28] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *Proc. of ICLR*, 2015.

[29] H. Dai, Z. Kozareva, B. Dai, A. Smola, and L. Song, "Learning steady-states of iterative algorithms over graphs," in *Proc. of ICML*, 2018, pp. 1114–1122.

[30] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *Proc. of ICLR*, 2014.

[31] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[32] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. of NIPS*, 2016, pp. 3844–3852.

[33] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "Cayleynets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Transactions on Signal Processing*, vol. 67, no. 1, pp. 97–109, 2017.

[34] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.

[35] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Proc. of NIPS*, 2016, pp. 1993–2001.

[36] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proc. of ICML*, 2016, pp. 2014–2023.

[37] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. of ICML*, 2017, pp. 1263–1272.

[38] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Transactions on Knowledge and Data Engineering*, 2017.

[39] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Network representation learning: A survey," *IEEE Transactions on Big Data*, 2018.

[40] H. Cai, V. W. Zheng, and K. Chang, "A comprehensive survey of graph embedding: problems, techniques and applications," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[41] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[42] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, "Tri-party deep network representation," in *Proc. of IJCAI*, 2016, pp. 1895–1901.

[43] X. Shen, S. Pan, W. Liu, Y.-S. Ong, and Q.-S. Sun, "Discrete network embedding," in *Proc. of IJCAI*, 2018, pp. 3549–3555.

[44] H. Yang, S. Pan, P. Zhang, L. Chen, D. Lian, and C. Zhang, "Binarized attributed network embedding," in *Proc. of ICDM*. IEEE, 2018.

[45] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proc. of KDD*. ACM, 2014, pp. 701–710.

[46] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *Journal of Machine Learning Research*, vol. 11, no. Apr, pp. 1201–1242, 2010.

[47] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.

[48] N. Navarin and A. Sperduti, "Approximated neighbours minhash graph node kernel." in *Proc. of ESANN*, 2017.

[49] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," *arXiv preprint arXiv:1903.11835*, 2019.

[50] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *Proc. of AAAI*, 2018, pp. 3546–3553.

[51] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *WWW*, 2018, pp. 499–508.

[52] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. of NIPS*, 2017, pp. 1024–1034.

[53] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *Proc. of CVPR*, 2017, pp. 5115–5124.

[54] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proc. of KDD*. ACM, 2018, pp. 1416–1424.

[55] D. V. Tran, A. Sperduti *et al.*, "On filter size in graph convolutional networks," in *SSCI*. IEEE, 2018, pp. 1534–1541.

[56] D. Bacciu, F. Errica, and A. Micheli, "Contextual graph markov model: A deep and generative approach to graph processing," in *Proc. of ICML*, 2018.

[57] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, "Gaan: Gated attention networks for learning on large and spatiotemporal graphs," in *Proc. of UAI*, 2018.

[58] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," in *Proc. of ICLR*, 2018.

[59] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *Proc. of ICML*, 2018, pp. 941–949.

[60] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *Proc. of NeurIPS*, 2018, pp. 4563–4572.

[61] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. of AAAI*, 2018.

[62] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proc. of AAAI*, 2018.

[63] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Proc. of NeurIPS*, 2018, pp. 4801–4811.

[64] Z. Liu, C. Chen, L. Li, J. Zhou, X. Li, and L. Song, "Geniepath: Graph neural networks with adaptive receptive paths," in *Proc. of AAAI*, 2019.

[65] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *Proc. of ICLR*, 2019.

[66] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks," in *Proc. of ICLR*, 2019.

[67] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *Proc. of AAAI*, 2016, pp. 1145–1152.

[68] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proc. of KDD*. ACM, 2016, pp. 1225–1234.

[69] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *NIPS Workshop on Bayesian Deep Learning*, 2016.

[70] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding." in *Proc. of IJCAI*, 2018, pp. 2609–2615.

[71] K. Tu, P. Cui, X. Wang, P. S. Yu, and W. Zhu, "Deep recursive network embedding with regular equivalence," in *Proc. of KDD*. ACM, 2018, pp. 2357–2366.

[72] W. Yu, C. Zheng, W. Cheng, C. C. Aggarwal, D. Song, B. Zong, H. Chen, and W. Wang, "Learning deep network representations with adversarially regularized autoencoders," in *Proc. of AAAI*. ACM, 2018, pp. 2663–2671.

[73] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," in *Proc. of ICML*, 2018.

[74] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "Graphrnn: A deep generative model for graphs," *Proc. of ICML*, 2018.

[75] M. Simonovsky and N. Komodakis, "Graphvae: Towards generation of small graphs using variational autoencoders," in *ICANN*. Springer, 2018, pp. 412–422.

[76] T. Ma, J. Chen, and C. Xiao, "Constrained generation of semantically valid graphs via regularizing variational autoencoders," in *Proc. of NeurIPS*, 2018, pp. 7110–7121.

[77] N. De Cao and T. Kipf, "MolGAN: An implicit generative model for small molecular graphs," *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

[78] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "Netgan: Generating graphs via random walks," in *Proc. of ICML*, 2018.

[79] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *Proc. of IJCAI*, 2018, pp. 3634–3640.

[80] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph wavenet for deep spatial-temporal graph modeling," in *Proc. of IJCAI*, 2019.

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

[82] S. Pan, J. Wu, X. Zhu, C. Zhang, and P. S. Yu, "Joint structure feature exploration and regularization for multi-task graph classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 3, pp. 715–728, 2016.

[83] S. Pan, J. Wu, X. Zhu, G. Long, and C. Zhang, "Task sensitive feature exploration and learning for multitask graph classification," *IEEE transactions on cybernetics*, vol. 47, no. 3, pp. 744–758, 2017.

[84] A. Micheli, D. Sona, and A. Sperduti, "Contextual processing of structured data by recursive cascade correlation," *IEEE Transactions on Neural Networks*, vol. 15, no. 6, pp. 1396–1410, 2004.

[85] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Proc. of EMNLP*, 2014, pp. 1724–1734.

[86] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[87] A. Sandryhaila and J. M. Moura, "Discrete signal processing on graphs," *IEEE transactions on signal processing*, vol. 61, no. 7, pp. 1644–1656, 2013.

[88] S. Chen, R. Varma, A. Sandryhaila, and J. Kovačević, "Discrete signal processing on graphs: Sampling theory," *IEEE Transactions on Signal Processing*, vol. 63, no. 24, pp. 6510–6523, 2015.

[89] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Proc. of NIPS*, 2015, pp. 2224–2232.

[90] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, "Molecular graph convolutions: moving beyond fingerprints," *Journal of computer-aided molecular design*, vol. 30, no. 8, pp. 595–608, 2016.

[91] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko, "Quantum-chemical insights from deep tensor neural networks," *Nature communications*, vol. 8, p. 13890, 2017.

[92] J. B. Lee, R. Rossi, and X. Kong, "Graph classification using structural attention," in *Proc. of KDD*. ACM, 2018, pp. 1666–1674.

[93] S. Abu-El-Haija, B. Perozzi, R. Al-Rfou, and A. A. Alemi, "Watch your step: Learning node embeddings via graph attention," in *Proc. of NeurIPS*, 2018, pp. 9197–9207.

[94] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on riemannian manifolds," in *Proc. of CVPR Workshops*, 2015, pp. 37–45.

[95] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein, "Learning shape correspondence with anisotropic convolutional neural networks," in *Proc. of NIPS*, 2016, pp. 3189–3197.

[96] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller, "Splinecnn: Fast geometric deep learning with continuous b-spline kernels," in *Proc. of CVPR*, 2018, pp. 869–877.

[97] B. Weisfeiler and A. Lehman, "A reduction of a graph to a canonical form and an algebra arising during this reduction," *Nauchno-Technicheskaya Informatsia*, vol. 2, no. 9, pp. 12–16, 1968.

[98] B. L. Douglas, "The weisfeiler-lehman method and graph isomorphism testing," *arXiv preprint arXiv:1101.5211*, 2011.

[99] T. Pham, T. Tran, D. Q. Phung, and S. Venkatesh, "Column networks for collective classification," in *Proc. of AAAI*, 2017, pp. 2485–2491.

[100] M. Simonovsky and N. Komodakis, "Dynamic edgeconditioned filters in convolutional neural networks on graphs," in *Proc. of CVPR*, 2017.

[101] T. Derr, Y. Ma, and J. Tang, "Signed graph convolutional network," in *Proc. of ICDM*, 2018.

[102] F. P. Such, S. Sah, M. A. Dominguez, S. Pillai, C. Zhang, A. Michael, N. D. Cahill, and R. Ptucha, "Robust spatial filtering with graph convolutional neural networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 11, no. 6, pp. 884–896, 2017.

[103] X. Wang, H. Ji, C. Shi, B. Wang, C. Peng, Y. P., and Y. Ye, "Heterogeneous graph attention network," in *WWW*, 2019.

[104] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 11, pp. 1944–1957, 2007.

[105] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," in *Proc. of ICLR*, 2016.

[106] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *Proc. of ICML*, 2019, pp. 3734–3743.

[107] F. Scarselli, A. C. Tsoi, and M. Hagenbuchner, "The vapnik–chervonenkis dimension of graph and recursive neural networks," *Neural Networks*, vol. 108, pp. 248–259, 2018.

[108] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman, "Invariant and equivariant graph networks," in *ICLR*, 2019.

[109] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[110] B. Hammer, A. Micheli, and A. Sperduti, "Universal approximation capability of cascade correlation for structures," *Neural Computation*, vol. 17, no. 5, pp. 1109–1159, 2005.

[111] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "Computational capabilities of graph neural networks," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 81–102, 2008.

[112] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proc. of ICML*. ACM, 2008, pp. 1096–1103.

[113] S. Pan, R. Hu, S.-f. Fung, G. Long, J. Jiang, and C. Zhang, "Learning graph embedding with adversarial training methods," *IEEE Transactions on Cybernetics*, 2019.

[114] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. of NIPS*, 2014, pp. 2672–2680.

[115] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS central science*, vol. 4, no. 2, pp. 268–276, 2018.

[116] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar variational autoencoder," in *Proc. of ICML*, 2017.

[117] H. Dai, Y. Tian, B. Dai, S. Skiena, and L. , "Syntax-directed variational autoencoder for molecule generation," in *Proc. of ICLR*, 2018.

[118] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *ESWC*. Springer, 2018, pp. 593–607.

[119] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Proc. of NIPS*, 2017, pp. 5767–5777.

[120] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[121] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, p. 93, 2008.

[122] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: extraction and mining of academic social networks," in *Proc. of KDD*. ACM, 2008, pp. 990–998.

[123] M. Zitnik and J. Leskovec, "Predicting multicellular function through multi-layer tissue networks," *Bioinformatics*, vol. 33, no. 14, pp. i190–i198, 2017.

[124] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *Knowledge and Information Systems*, vol. 14, no. 3, pp. 347–375, 2008.

[125] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of medicinal chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[126] P. D. Dobson and A. J. Doig, "Distinguishing enzyme structures from non-enzymes without alignments," *Journal of molecular biology*, vol. 330, no. 4, pp. 771–783, 2003.

[127] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.

[128] H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, and C. Helma, "Statistical evaluation of the predictive toxicology challenge 2000–2001," *Bioinformatics*, vol. 19, no. 10, pp. 1183–1193, 2003.

[129] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. Von Lilienfeld, "Quantum chemistry structures and properties of 134 kilo molecules," *Scientific data*, vol. 1, p. 140022, 2014.

[130] G. Chen, P. Chen, C.-Y. Hsieh, C.-K. Lee, B. Liao, R. Liao, W. Liu, J. Qiu, Q. Sun, J. Tang *et al.*, "Alchemy: A quantum chemistry dataset for benchmarking ai models," *arXiv preprint arXiv:1906.09427*, 2019.

[131] L. Tang and H. Liu, "Relational learning via latent social dimensions," in *Proc. of KDD*. ACM, 2009, pp. 817–826.

[132] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[133] H. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi, "Big data and its technical challenges," *Communications of the ACM*, vol. 57, no. 7, pp. 86–94, 2014.

[134] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell, "Toward an architecture for never-ending language learning." in *Proc. of AAAI*, 2010, pp. 1306–1313.

[135] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," in *NeurIPS workshop*, 2018.

[136] Anonymous, "A fair comparison of graph neural networks for graph classification," in *Submitted to ICLR*, 2020, under review. [Online]. Available: https://openreview.net/forum?id=HygDF6NFPB

[137] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma, Z. Huang, Q. Guo, H. Zhang, H. Lin, J. Zhao, J. Li, A. J. Smola, and Z. Zhang, "Deep graph library: Towards efficient and scalable deep learning on graphs," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[138] C. Wang, S. Pan, G. Long, X. Zhu, and J. Jiang, "Mgae: Marginalized graph autoencoder for graph clustering," in *Proc. of CIKM*. ACM, 2017, pp. 889–898.

[139] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Proc. of NeurIPS*, 2018.

[140] T. Kawamoto, M. Tsubaki, and T. Obuchi, "Mean-field theory of graph neural networks in graph partitioning," in *Proc. of NeurIPS*, 2018, pp. 4362–4372.

[141] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, "Scene graph generation by iterative message passing," in *Proc. of CVPR*, vol. 2, 2017.

[142] J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh, "Graph r-cnn for scene graph generation," in *Proc. of ECCV*. Springer, 2018, pp. 690–706.

[143] Y. Li, W. Ouyang, B. Zhou, J. Shi, C. Zhang, and X. Wang, "Factorizable net: an efficient subgraph-based framework for scene graph generation," in *Proc. of ECCV*. Springer, 2018, pp. 346–363.

[144] J. Johnson, A. Gupta, and L. Fei-Fei, "Image generation from scene graphs," in *Proc. of CVPR*, 2018.

[145] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *ACM Transactions on Graphics (TOG)*, 2019.

[146] L. Landrieu and M. Simonovsky, "Large-scale point cloud semantic segmentation with superpoint graphs," in *Proc. of CVPR*, 2018.

[147] G. Te, W. Hu, A. Zheng, and Z. Guo, "Rgcnn: Regularized graph cnn for point cloud segmentation," in *2018 ACM Multimedia Conference on Multimedia Conference*. ACM, 2018, pp. 746–754.

[148] S. Qi, W. Wang, B. Jia, J. Shen, and S.-C. Zhu, "Learning human-object interactions by graph parsing neural networks," in *Proc. of ECCV*. Springer, 2018, pp. 401–417.

[149] V. G. Satorras and J. B. Estrach, "Few-shot learning with graph neural networks," in *Proc. of ICLR*, 2018.

[150] M. Guo, E. Chou, D.-A. Huang, S. Song, S. Yeung, and L. Fei-Fei, "Neural graph matching networks for fewshot 3d action recognition," in *Proc. of ECCV*. Springer, 2018, pp. 673–689.

[151] L. Liu, T. Zhou, G. Long, J. Jiang, L. Yao, and C. Zhang, "Prototype propagation networks (ppn) for weakly-supervised few-shot learning on category graph," in *Proc. of IJCAI*, 2019.

[152] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun, "3d graph neural networks for rgbd semantic segmentation," in *Proc. of CVPR*, 2017, pp. 5199–5208.

[153] L. Yi, H. Su, X. Guo, and L. J. Guibas, "Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation." in *Proc. of CVPR*, 2017, pp. 6584–6592.

[154] X. Chen, L.-J. Li, L. Fei-Fei, and A. Gupta, "Iterative visual reasoning beyond convolutions," in *Proc. of CVPR*, 2018.

[155] M. Narasimhan, S. Lazebnik, and A. Schwing, "Out of the box: Reasoning with graph convolution nets for factual visual question answering," in *Proc. of NeurIPS*, 2018, pp. 2655–2666.

[156] D. Marcheggiani and I. Titov, "Encoding sentences with graph convolutional networks for semantic role labeling," in *Proc. of EMNLP*, 2017, pp. 1506–1515.

[157] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, and K. Sima'an, "Graph convolutional encoders for syntax-aware neural machine translation," in *Proc. of EMNLP*, 2017, pp. 1957–1967.

[158] D. Marcheggiani, J. Bastings, and I. Titov, "Exploiting semantics in neural machine translation with graph convolutional networks," in *Proc. of NAACL*, 2018.

[159] L. Song, Y. Zhang, Z. Wang, and D. Gildea, "A graph-to-sequence model for amr-to-text generation," in *Proc. of ACL*, 2018.

[160] D. Beck, G. Haffari, and T. Cohn, "Graph-to-sequence learning using gated graph neural networks," in *Proc. of ACL*, 2018.

[161] D. D. Johnson, "Learning graphical state transitions," in *Proc. of ICLR*, 2016.

[162] B. Chen, L. Sun, and X. Han, "Sequence-to-action: End-to-end semantic graph generation for semantic parsing," in *Proc. of ACL*, 2018, pp. 766–777.

[163] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.

[164] R. van den Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *stat*, vol. 1050, p. 7, 2017.

[165] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proc. of KDD*. ACM, 2018, pp. 974–983.

[166] F. Monti, M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," in *Proc. of NIPS*, 2017, pp. 3697–3707.

[167] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein interface prediction using graph convolutional networks," in *Proc. of NIPS*, 2017, pp. 6530–6539.

[168] J. You, B. Liu, R. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *Proc. of NeurIPS*, 2018.

[169] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[170] G. Long, Y. Tan, J. Jiang, and C. Zhang, "Federated learning for open banking." Springer Nature, 2020, vol. 12500, pp. 240–254.

[171] N. Rieke, J. Hancox, W. Li, F. Milletari, H. R. Roth, S. Albarqouni, S. Bakas, M. N. Galtier, B. A. Landman, K. Maier-Hein *et al.*, "The future of digital health with federated learning," *NPJ digital medicine*, vol. 3, no. 1, pp. 1–7, 2020.

[172] G. Long, T. Shen, Y. Tan, L. Gerrard, A. Clarke, and J. Jiang, "Federated learning for privacy-preserving open innovation future on digital health," in *Humanity Driven AI*. Springer, 2022, pp. 113–133.

[173] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[174] G. Cormode, S. Jha, T. Kulkarni, N. Li, D. Srivastava, and T. Wang, "Privacy at scale: Local differential privacy in practice," in *Proceedings of the 2018 International Conference on Management of Data*, 2018, pp. 1655–1658.

[175] J. Jiang, S. Ji, and G. Long, "Decentralized knowledge acquisition for mobile internet applications," *World Wide Web*, vol. 23, no. 5, pp. 2653–2669, 2020.

[176] Y. Tan, G. Long, L. Liu, T. Zhou, Q. Lu, J. Jiang, and C. Zhang, "Fedproto: Federated prototype learning across heterogeneous clients," in *AAAI Conference on Artificial Intelligence*, vol. 1, 2022.

[177] F. Chen, G. Longr, Z. Wu, T. Zhou, and J. Jiang, "Personalized federated learning with structure," *arXiv preprint arXiv:2203.00829*, 2022.

[178] M. Xie, G. Long, T. Shen, T. Zhou, X. Wang, J. Jiang, and C. Zhang, "Multi-center federated learning," *arXiv preprint arXiv:2108.08647*, 2021.

[179] J. Ma, G. Long, T. Zhou, J. Jiang, and C. Zhang, "On the convergence of clustered federated learning," *arXiv preprint arXiv:2202.06187*, 2022.

[180] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," in *Proc. of ICLR*, 2017.

[181] J. Qiu, J. Tang, H. Ma, Y. Dong, K. Wang, and J. Tang, "Deepinf: Social influence prediction with deep learning," in *Proc. of KDD*. ACM, 2018, pp. 2110–2119.

[182] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proc. of KDD*. ACM, 2018, pp. 2847–2856.

[183] E. Choi, M. T. Bahadori, L. Song, W. F. Stewart, and J. Sun, "Gram: graph-based attention model for healthcare representation learning," in *Proc. of KDD*. ACM, 2017, pp. 787–795.

[184] E. Choi, C. Xiao, W. Stewart, and J. Sun, "Mime: Multilevel medical embedding of electronic health records for predictive healthcare," in *Proc. of NeurIPS*, 2018, pp. 4548–4558.

[185] J. Kawahara, C. J. Brown, S. P. Miller, B. G. Booth, V. Chau, R. E. Grunau, J. G. Zwicker, and G. Hamarneh, "Brainnetcnn: convolutional neural networks for brain networks; towards predicting neurodevelopment," *NeuroImage*, vol. 146, pp. 1038–1049, 2017.

[186] T. H. Nguyen and R. Grishman, "Graph convolutional networks with argument-aware pooling for event detection," in *Proc. of AAAI*, 2018, pp. 5900–5907.

[187] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *Proc. of NeurIPS*, 2018, pp. 536–545.

[188] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[189] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[190] H. Wang, C. Zhou, X. Chen, J. Wu, S. Pan, and J. Wang, "Graph stochastic neural networks for semi-supervised learning," in *Proc. of NeurIPS*, vol. 33, 2020.

[191] M. Wu, S. Pan, and X. Zhu, "Openwgl: Open-world graph learning," in *Proc. of ICDM*, 2020.

[192] S. Zhu, S. Pan, C. Zhou, J. Wu, Y. Cao, and B. Wang, "Graph geometry interaction learning," in *Proc. of NeurIPS*, vol. 33, 2020.

[193] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang, "Connecting the dots: Multivariate time series forecasting with graph neural networks," in *Proc. of KDD*. ACM, 2018, pp. 753–763.

[194] M. Balcilar, G. Renton, P. Héroux, B. Gauzere, S. Adam, and P. Honeine, "Bridging the gap between spectral and spatial domains in graph neural networks," *arXiv preprint arXiv:2003.11702*, 2020.

[195] F. Wu, T. Zhang, A. H. d. Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *Proc. of ICML*, 2019.

[196] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "Cayleynets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Transactions on Signal Processing*, vol. 67, no. 1, pp. 97–109, 2018.

[197] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[198] B. Xu, H. Shen, Q. Cao, Y. Qiu, and X. Cheng, "Graph wavelet neural network," in *Proc. of ICLR*, 2019.

[199] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. of ICLR*, 2019.

[200] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *Proc. of ICLR*, 2019.

[201] H. Gao and S. Ji, "Graph u-net," in *Proc. of ICML*, 2019, pp. 2083–2092.

[202] S. Luan, M. Zhao, X.-W. Chang, and D. Precup, "Break the ceiling: Stronger multi-scale deep graph convolutional networks," in *Proc. of NeuIPS*, 2019, pp. 10 945–10 955.

[203] S. Abu-El-Haija, B. Perozzi, A. Kapoor, N. Alipourfard, K. Lerman, H. Harutyunyan, G. V. Steeg, and A. Galstyan, "Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," in *Proc. of ICML*, 2019.

[204] R. R. Coifman and M. Maggioni, "Diffusion wavelets," *Applied and Computational Harmonic Analysis*, vol. 21, no. 1, pp. 53–94, 2006.

[205] F. Gao, G. Wolf, and M. Hirn, "Geometric scattering for graph data analysis," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2122–2131.

[206] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *arXiv preprint arXiv:2003.00982*, 2020.

[207] D. Lim, X. Li, F. Hohne, and S.-N. Lim, "New benchmarks for learning on non-homophilous graphs," *arXiv preprint arXiv:2104.01404*, 2021.

[208] I. Spinelli, S. Scardapane, and A. Uncini, "Adaptive propagation graph convolutional network," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[209] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," in *Proc. of ICLR*, 2018.

[210] W. Jin, R. Barzilay, and T. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," in *ICML*, 2018.

[211] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[212] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. of ICLR*, 2017.

[213] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *Proc. of ICLR*, 2016.

[214] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *Proc. of ICML*, 2017, pp. 933–941.

[215] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. v. d. Oord, A. Graves, and K. Kavukcuoglu, "Neural machine translation in linear time," *arXiv preprint arXiv:1610.10099*, 2016.

[216] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *arXiv preprint arXiv:1211.0053*, 2012.

[217] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, "Modeling long-and short-term temporal patterns with deep neural networks," in *Proc. of SIGIR*. ACM, 2018, pp. 95–104.

[218] S.-Y. Shih, F.-K. Sun, and H.-y. Lee, "Temporal pattern attention for multivariate time series forecasting," *Machine Learning*, vol. 108, no. 8-9, pp. 1421–1441, 2019.

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

[220] R. Frigola, "Bayesian time series learning with gaussian processes," Ph.D. dissertation, University of Cambridge, 2015.

[221] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *Proc. of ICLR*, 2019.

[222] C. Zheng, X. Fan, C. Wang, and J. Qi, "Gman: A graph multi-attention network for traffic prediction," in *Proc. of AAAI*, 2020.

[223] W. Chen, L. Chen, Y. Xie, W. Cao, Y. Gao, and X. Feng, "Multi-range attentive bicomponent graph convolutional network for traffic forecasting," in *Proc. of AAAI*, 2019.

[224] L. Shi, Y. Zhang, J. Cheng, and H. Lu, "Two-stream adaptive graph convolutional networks for skeleton-based action recognition," in *Proc. of CVPR*, 2019, pp. 12 026–12 035.

[225] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *Proc. of AAAI*, vol. 33, 2019, pp. 922–929.

[226] A. Kapoor, A. Galstyan, B. Perozzi, G. Ver Steeg, H. Harutyunyan, K. Lerman, N. Alipourfard, and S. Abu-El-Haija, "Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," in *Proc. of ICML*, 2019.

[227] F. Chen, S. Pan, J. Jiang, H. Huo, and G. Long, "Dagcn: Dual attention graph convolutional networks," in *Proc. of IJCNN*, 2019.

[228] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. of CVPR*, 2015, pp. 1–9.

[229] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, 2003.

[230] S. Roberts, M. Osborne, M. Ebden, S. Reece, N. Gibson, and S. Aigrain, "Gaussian processes for time-series modelling," *Philos. Trans. R. Soc. A*, vol. 371, no. 1984, p. 20110550, 2013.

[231] R. Frigola-Alcalde, "Bayesian time series learning with gaussian processes," Ph.D. dissertation, University of Cambridge, 2016.

[232] Z. Pan, Y. Liang, W. Wang, Y. Yu, Y. Zheng, and J. Zhang, "Urban traffic prediction from spatio-temporal data using deep meta learning," in *Proc. of KDD*. ACM, 2019, pp. 1720–1730.

[233] D. Zhang, L. Yao, K. Chen, S. Wang, P. D. Haghighi, and C. Sullivan, "A graph-based hierarchical attention model for movement intention detection from eeg

signals," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 27, no. 11, pp. 2247–2253, 2019.

[234] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. of NIPS*, 2017, pp. 5998–6008.

[235] G. Tang, M. Müller, A. Rios, and R. Sennrich, "Why self-attention? a targeted evaluation of neural machine translation architectures," in *Proc. of ACL*, 2018, pp. 4263–4272.

[236] 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," in *Proc. of CIKM*, 2020.

[237] X. Wang, Y. Ma, Y. Wang, W. Jin, X. Wang, J. Tang, C. Jia, and J. Yu, "Traffic flow prediction via spatial temporal graph neural network," in *Proc. of WWW*, 2020, pp. 1082–1092.

[238] B. Li, X. Li, Z. Zhang, and F. Wu, "Spatio-temporal graph routing for skeleton-based action recognition," in *Proc. of AAAI*, 2019, pp. 8561–8568.

[239] 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. of AAAI*, vol. 34, no. 01, 2020, pp. 914–921.

[240] L. Mengzhang and Z. Zhanxing, "Spatial-temporal fusion graph neural networks for traffic flow forecasting," in *Proc. of AAAI*, 2021.

[241] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. of NIPS*, 2014, pp. 3104–3112.

[242] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of CVPR*, 2016, pp. 770–778.