A Comprehensive Survey on Graph Neural Networks

Zonghan Wu, Shirui Pan, *Member, IEEE*, Fengwen Chen, Guodong Long, Chengqi Zhang, *Senior Member, IEEE*, Philip S. Yu, *Fellow, IEEE*

Abstract—Deep learning has revolutionized many machine learning tasks in recent years, ranging from image classification and video processing to speech recognition and natural language understanding. The data in these tasks are typically represented in the Euclidean space. However, there is an increasing number of applications where data are generated from non-Euclidean domains and are represented as graphs with complex relationships and interdependency between objects. The complexity of graph data has imposed significant challenges on existing machine learning algorithms. Recently, many studies on extending deep learning approaches for graph data have emerged. In this survey, we provide a comprehensive overview of graph neural networks (GNNs) in data mining and machine learning fields. We propose a new taxonomy to divide the state-of-the-art graph neural networks into different categories. With a focus on graph convolutional networks, we review alternative architectures that have recently been developed; these learning paradigms include graph attention networks, graph autoencoders, graph generative networks, and graph spatial-temporal networks. We further discuss the applications of graph neural networks across various domains and summarize the open source codes and benchmarks of the existing algorithms on different learning tasks. Finally, we propose potential research directions in this fast-growing field.

Index Terms—Deep Learning, graph neural networks, graph convolutional networks, graph representation learning, graph autoencoder, network embedding

1 Introduction

The recent success of neural networks has boosted research on pattern recognition and data mining. Many machine learning tasks such as object detection [1], [2], machine translation [3], [4], and speech recognition [5], 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, i.e., convolutional neural networks (CNNs) [6], long short-term memory (LSTM) [7], and autoencoders. The success of deep learning in many domains is partially attributed to the rapidly developing computational resources (e.g., GPU) and the availability of large training data, and is partially due to the effectiveness of deep learning to extract latent representation from Euclidean data (e.g., images, text, and video). Taking image analysis as an example, an image can be represented 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 [8], and as a result, CNN can extract local meaningful features that are shared with the entire datasets for various image analysis tasks.

While deep learning has achieved great success on Euclidean data, there is an increasing number of applications where data are generated from the non-Euclidean domain and need to be effectively analyzed. For instance, in e-commerce, a graph-based learning system is able to exploit the interactions between users and products [9], [10], [11] to make highly accurate recommendations. In chemistry, molecules are modeled as graphs and their bioactivity needs to be identified for drug discovery [12], [13]. In a citation network, papers are linked to each other via citations and they need to be categorized into different groups [14], [15]. The complexity of graph data has imposed significant challenges on existing machine learning algorithms. This is because graph data are irregular. Each graph has a variable size of unordered nodes and each node in a graph has a different number of neighbors, causing some important operations (e.g., convolutions), which are easy to compute in the image domain but are not directly applicable to the graph domain anymore. Furthermore, a core assumption of existing machine learning algorithms is that instances are independent of each other. However, this is not the case for graph data where each instance (node) is related to others (neighbors) via some complex linkage information, which is used to capture the interdependence among data, including citations, friendship, and interactions.

Recently, there is increasing interest in extending deep learning approaches for graph data. Driven by the success of deep learning, researchers have borrowed ideas from convolution networks, recurrent networks, and deep autoencoders to design the architecture of graph neural net-

- *Z. Wu, F. Chen, G. Long, C. Zhang are with Centre for Artificial Intelligence, FEIT, University of Technology Sydney, NSW 2007, Australia (E-mail: zonghan.wu-3@student.uts.edu.au; fengwen.chen@student.uts.edu.au; guodong.long@uts.edu.au; chengqi.zhang@uts.edu.au).*
- *S. Pan is with Faculty of Information Technology, Monash University, Clayton, VIC 3800, Australia (Email: shirui.pan@monash.edu).*
- *P. S. Yu is with Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7053, USA (Email: psyu@uic.edu)*
- *Corresponding author: Shirui Pan.*

Manuscript received Dec xx, 2018; revised Dec xx, 201x.

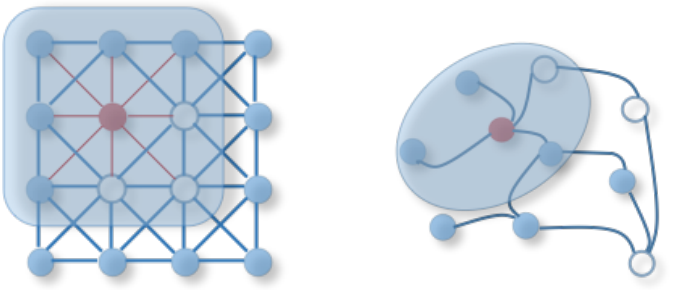
works. To handle the complexity of graph data, new generalizations and definitions for important operations have been rapidly developed over the past few years. For instance, Figure 1 illustrates how a kind of graph convolution is inspired by a standard 2D convolution. This survey aims to provide a comprehensive overview of these methods, for both interested researchers who want to enter this rapidly developing field and experts who would like to compare graph neural network algorithms.

A Brief History of Graph Neural Networks The notation of graph neural networks was firstly outlined in Gori et al. (2005) [16], and further elaborated in Micheli (2009) [17] and Scarselli et al. (2009) [18]. These early studies learn a target node's representation by propagating neighbor information via recurrent neural architectures 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 [19], [20]. In our survey, we generalize the term *graph neural networks* to represent all deep learning approaches for graph data.

Inspired by the huge success of convolutional networks in the computer vision domain, a large number of methods that re-define the notation of *convolution* for graph data have emerged recently. These approaches are under the umbrella of graph convolutional networks (GCNs). The first prominent research on GCNs is presented in Bruna et al. (2013), which develops a variant of graph convolution based on spectral graph theory [21]. Since that time, there have been increasing improvements, extensions, and approximations on spectral-based graph convolutional networks [12], [14], [22], [23], [24]. As spectral methods usually handle the whole graph simultaneously and are difficult to parallel or scale to large graphs, spatial-based graph convolutional networks have rapidly developed recently [25], [26], [27], [28]. These methods directly perform the convolution in the graph domain by aggregating the neighbor nodes' information. Together with sampling strategies, the computation can be performed in a batch of nodes instead of the whole graph [25], [28], which has the potential to improve efficiency.

In addition to graph convolutional networks, many alternative graph neural networks have been developed in the past few years. These approaches include graph attention networks, graph autoencoders, graph generative networks, and graph spatial-temporal networks. Details on the categorization of these methods are given in Section 3.

Related surveys on graph neural networks. There are a limited number of existing reviews on the topic of graph neural networks. Using the notation *geometric deep learning*, Bronstein et al. [8] give an overview of deep learning methods in the non-Euclidean domain, including graphs and manifolds. While being the first review on graph convolution networks, this survey misses several important spatial-based approaches, including [15], [20], [25], [27], [28], [29], which update state-of-the-art benchmarks. Furthermore, this survey does not cover many newly developed architectures which are equally important to graph convolutional networks. These learning paradigms, including graph attention networks, graph autoencoders, graph generative networks, and graph spatial-temporal networks, are comprehensively reviewed in this article. Battaglia et al. [30] position *graph networks* as the building blocks for learning from relational data, reviewing part of graph neural networks under a unified framework. However, their generalized framework is highly abstract, losing insights on each method from its original paper. Lee et al. [31] conduct a partial survey on the graph attention model, which is one type of graph neural network. Most recently, Zhang et al. [32] present a most up-to-date survey on deep learning for graphs, missing those studies on graph generative and spatial-temporal networks. In summary, none of the existing surveys provide a comprehensive overview of graph neural networks, only covering some of the graph convolution neural networks and examining a limited number of works, thereby missing the most recent development of alternative graph neural networks, such as graph generative networks and graph spatial-temporal networks.



(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 a 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 graph convolution operation takes the average value of 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.

Fig. 1: 2D Convolution vs. Graph Convolution.

Graph neural networks vs. network embedding The research on graph neural networks is closely related to graph embedding or network embedding, another topic which attracts increasing attention from both the data mining and machine learning communities [33] [34] [35] [36], [37], [38]. Network embedding aims to represent network vertices into a low-dimensional vector space, by preserving both network topology structure and node content information, so that any subsequent graph analytics tasks such as classification, clustering, and recommendation can be easily performed by using simple off-the-shelf machine learning algorithm (e.g., support vector machines for classification). Many network embedding algorithms are typically unsupervised algorithms and they can be broadly classified into three groups [33], i.e., matrix factorization [39], [40], random walks [41], and deep learning approaches. The deep learning approaches for network embedding at the same time belong to graph neural networks, which include graph autoencoder-based algorithms (e.g., DNGR [42] and SDNE [43]) and graph convolution neural networks with unsupervised training(e.g., GraphSage [25]). Figure 2 describes the differences between network embedding and graph neural

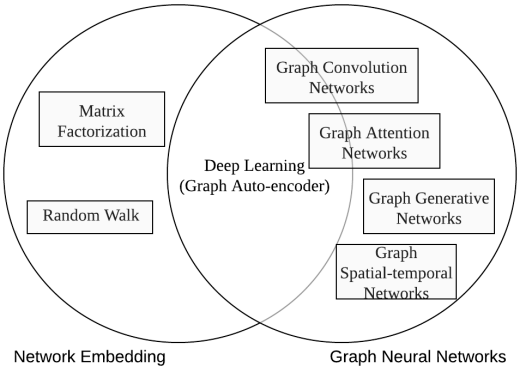


Fig. 2: Network Embedding v.s. Graph Neural Networks.

networks in this paper.

Our Contributions Our paper makes notable contributions summarized as follows:

- **New taxonomy** In light of the increasing number of studies on deep learning for graph data, we propose a new taxonomy of graph neural networks (GNNs). In this taxonomy, GNNs are categorized into five groups: graph convolution networks, graph attention networks, graph auto-encoders, graph generative networks, and graph spatial-temporal networks. We pinpoint the differences between graph neural networks and network embedding and draw the connections between different graph neural network architectures.
- **Comprehensive review** This survey provides the most comprehensive overview of modern deep learning techniques for graph data. For each type of graph neural network, we provide detailed descriptions on representative algorithms, and make a necessary comparison and summarise the corresponding algorithms.
- **Abundant resources** This survey provides abundant resources on graph neural networks, which include state-of-the-art algorithms, benchmark datasets, open-source codes, and practical applications. This survey can be used as a hands-on guide for understanding, using, and developing different deep learning approaches for various real-life applications.
- **Future directions** This survey also highlights the current limitations of the existing algorithms, and points out possible directions in this rapidly developing field.

Organization of Our Survey The rest of this survey is organized as follows. Section 2 defines a list of graph-related concepts. Section 3 clarifies the categorization of graph neural networks. Section 4 and Section 5 provides an overview of graph neural network models. Section 6 presents a gallery of applications across various domains. Section 7 discusses the current challenges and suggests future directions. Section 8 summarizes the paper.

TABLE 1: Commonly used notations.

Notations	Descriptions
$\|\cdot\|$	The length of a set
$\odot$	Element-wise product.
$\mathbf{A}^T$	Transpose of vector/matrix $\mathbf{A}$.
$[\mathbf{A}, \mathbf{B}]$	Concatenation of $\mathbf{A}$ and $\mathbf{B}$.
$\mathcal{G}$	A graph
$\mathbf{V}$	The set of nodes in a graph
v_i	A node $v_i \in \mathbf{V}$
$N(v)$	The neighbors of node v
$\mathbf{E}$	The set of edges in a graph
e_{ij}	An edge $e_{ij} \in \mathbf{E}$
$\mathbf{X} \in \mathbf{R}^{N \times D}$	The feature matrix of a graph.
$\mathbf{x} \in \mathbf{R}^N$	The feature vector of a graph in the case of $D = 1$.
$\mathbf{X}_i \in \mathbf{R}^D$	The feature vector of the node v_i.
N	The number of nodes, $N = \|\mathbf{V}\|$.
M	The number of edges, $M = \|\mathbf{E}\|$.
D	The dimension of a node vector.
T	The total number of time steps in time series.

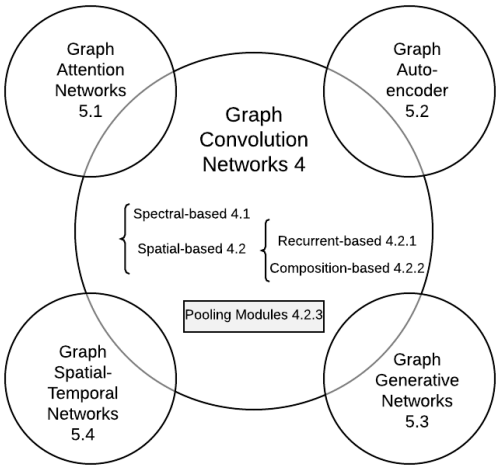


Fig. 3: Categorization of Graph Neural Networks.

2 Definition

In this section, we provide definitions of basic graph concepts. For easy retrieval, we summarize the commonly used notations in Table 1.

Definition 1 (Graph). A Graph is $\mathcal{G} = (\mathbf{V}, \mathbf{E}, \mathbf{A})$ where $\mathbf{V}$ is the set of nodes, $\mathbf{E}$ is the set of edges, and $\mathbf{A}$ is the adjacency matrix. In a graph, let $v_i \in \mathbf{V}$ to denote a node and $e_{ij} = (v_i, v_j) \in \mathbf{E}$ to denote an edge. The adjacency matrix $\mathbf{A}$ is a $N \times N$ matrix with $\mathbf{A}_{ij} = w_{ij} > 0$ if $e_{ij} \in \mathbf{E}$ and $\mathbf{A}_{ij} = 0$ if $e_{ij} \notin \mathbf{E}$. The degree of a node is the number of edges connected to it.

A graph can be associated with node attributes $\mathbf{X}$ [1], where $\mathbf{X} \in \mathbf{R}^{N \times D}$ is a feature matrix with $\mathbf{X}_i \in \mathbf{R}^D$ representing the feature vector of node v_i. In the case of $D = 1$, we replace $\mathbf{x} \in \mathbf{R}^N$ with $\mathbf{X}$ to denote the feature vector of the graph.

Definition 2 (Directed Graph). A directed graph is a graph with all edges pointing from one node to another. For a directed graph, $\mathbf{A}_{ij} \neq \mathbf{A}_{ji}$. An undirected graph is

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

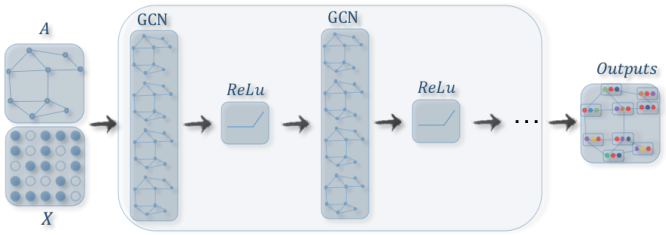


Fig. 4: A Variant of Graph Convolution Networks with Multiple GCN layers [14]. A GCN layer encapsulates each node's hidden representation by aggregating feature information from its neighbors. After feature aggregation, a non-linear transformation is applied to the resultant outputs. By stacking multiple layers, the final hidden representation of each node receives messages from a further neighborhood.

a graph with all edges undirected. For an undirected graph, $\mathbf{A}_{ij} = \mathbf{A}_{ji}$.

Definition 3 (Spatial-Temporal Graph). A spatial-temporal graph is an attributed graph where the feature matrix $\mathbf{X}$ evolves over time. It is defined as $\mathcal{G} = (\mathbf{V}, \mathbf{E}, \mathbf{A}, \mathbf{X})$ with $\mathbf{X} \in \mathbf{R}^{T \times N \times D}$ where T is the length of time steps.

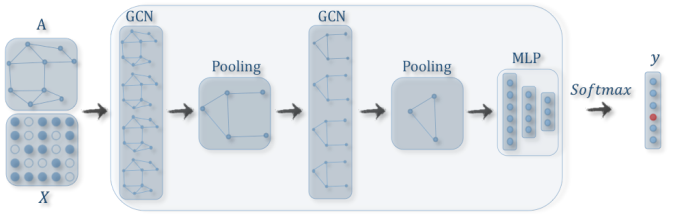
3 Categorization and Frameworks

In this section, we present our taxonomy of graph neural networks. We consider any differentiable graph models which incorporate neural architectures as graph neural networks. We categorize graph neural networks into graph convolution networks, graph attention networks, graph auto-encoders, graph generative networks and graph spatial-temporal networks. Of these, graph convolution networks play a central role in capturing structural dependencies. As illustrated in Figure 3, methods under other categories partially utilize graph convolution networks as building blocks. We summarize the representative methods in each category in Table 2, and we give a brief introduction of each category in the following.

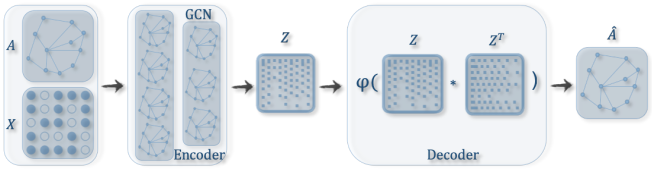
3.1 Taxonomy of GNNs

Graph Convolution Networks (GCNs) generalize the operation of *convolution* from traditional data (images or grids) to graph data. The key is to learn a function f to generate a node v_i's representation by aggregating its own features $\mathbf{X}_i$ and neighbors' features $\mathbf{X}_j$, where $j \in N(v_i)$. Figure 4 shows the process of GCNs for node representation learning. Graph convolutional networks play a central role in building up many other complex graph neural network models, including auto-encoder-based models, generative models, and spatial-temporal networks, etc. Figure 5 illustrates several graph neural network models building on GCNs.

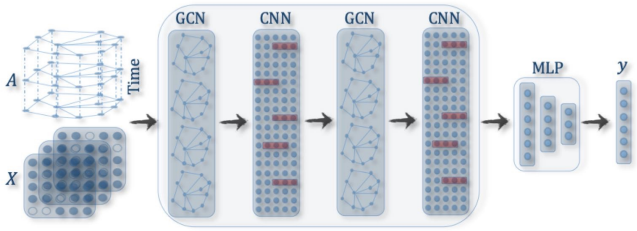
Graph Attention Networks are similar to GCNs and seek an aggregation function to fuse the neighboring nodes, random walks, and candidate models in graphs to learn a new representation. The key difference is that graph attention networks employ *attention* mechanisms which assign larger weights to the more important nodes, walks, or models. The attention weight is learned together with neural network parameters within an end-to-end framework. Figure 6 illustrates the difference between graph convolutional networks and graph attention networks in aggregating the neighbor node information.



(a) Graph Convolution Networks with Pooling Modules for Graph Classification [12]. A GCN layer [14] 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. To calculate the probability for each graph label, the output layer is a linear layer with the SoftMax function.



(b) Graph Auto-encoder with GCN [62]. The encoder uses GCN layers to get latent rerpesentations for each node. The decoder computes the pair-wise distance between node latent representations produced by the encoder. After applying a non-linear activation function, the decoder reconstructs the graph adjacency matrix.



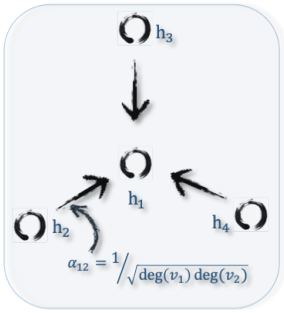
(c) Graph Spatial-Temporal Networks with GCN [74]. A GCN layer is followed by a 1D-CNN layer. The GCN layer operates on A_t and X_t to capture spatial dependency, while the 1D-CNN layer slides over X along the time axis to capture the temporal dependency. The output layer is a linear transformation, generating a prediction for each node.

Fig. 5: Different Graph Neural Network Models built with GCNs.

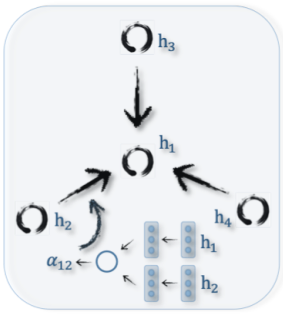
Graph Auto-encoders are unsupervised learning frameworks which aim to learn low dimensional node vectors via an encoder, and then reconstruct the graph data via a decoder. Graph autoencoders are a popular approach to learn the graph embedding, for both plain graphs without attributed information [42], [43] as well as attributed graphs [64], [65]. For plain graphs, many algorithms directly prepossess the adjacency matrix, by either constructing a new matrix (i.e., pointwise mutual information matrix) with rich information [42] or feeding the adjacency matrix to an autoencoder model and capturing both first order and second order information [43]. For attributed graphs, graph autoencoder models tend to employ GCN [14] as a building block for the encoder and reconstruct the structure information via a link prediction decoder [62], [64].

TABLE 2: Representative Publications of Graph Neural Networks

Category		Publications
Graph Convolution Networks	Spectral-based	[12], [14], [21], [22], [23], [24], [44], [45], [46]
	Spatial-based	[13], [16], [17], [18], [19], [20], [25], [26], [27], [28], [47] [48], [49], [50], [51], [52], [53], [54], [55], [56], [57]
	Pooling modules	[12], [22], [58], [59]
Graph Attention Networks		[15], [29], [60], [61]
Graph Auto-encoder		[42], [43], [62], [63], [64], [65], [66]
Graph Generative Networks		[67], [68], [69], [70], [71]
Graph Spatial-Temporal Networks		[72], [73], [74], [75], [76]



(a) Graph Convolution Networks [14] explicitly assign 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) Graph Attention Networks [15] implicitly capture the weight a_{ij} via an end-to-end neural network architecture, so that more important nodes receive larger weights.

Fig. 6: Differences between graph convolutional networks and graph attention networks.

Graph Generative Networks aim to generate plausible structures from data. Generating graphs given a graph empirical distribution is fundamentally challenging, mainly because graphs are complex data structures. To address this problem, researchers have explored to factor the generation process as forming nodes and edges alternatively [67], [68], to employ generative adversarial training [69], [70]. One promising application domain of graph generative networks is chemical compound synthesis. In a chemical graph, atoms are treated as nodes and chemical bonds are treated as edges. The task is to discover new synthesizable molecules which possess certain chemical and physical properties.

Graph Spatial-temporal Networks aim to learn unseen patterns from spatial-temporal graphs, which are increasingly important in many applications such as traffic forecasting and human activity prediction. For instance, the underlying road traffic network is a natural graph where each key location is a node whose traffic data is continuously monitored. By developing effective graph spatial-temporal network models, we can accurately predict the traffic status over the whole traffic system [73], [74]. The key idea of graph spatial-temporal networks is to consider spatial dependency and temporal dependency at the same time. Many current approaches apply GCNs to capture the dependency together with some RNN [73] or CNN [74] to model the temporal dependency.

3.2 Frameworks

Graph neural networks, graph convolution networks (GCNs) in particular, try to replicate the success of CNN in graph data by defining graph convolutions via graph spectral theory or spatial locality. With the graph structure and node content information as inputs, the outputs of GCN can focus on different graph analytics task with one of the following mechanisms:

- **Node-level** outputs relate to node regression and classification tasks. As a graph convolution module directly gives nodes' latent representations, a multi-perceptron layer or softmax layer is used as the final layer of GCN. We review graph convolution modules in Section 4.1 and Section 4.2.
- **Edge-level** outputs relate to the edge classification and link prediction tasks. To predict the label/connection strength of an edge, an additional function will take two nodes' latent representations from the graph convolution module as inputs.
- **Graph-level** outputs relate to the graph classification task. To obtain a compact representation on graph level, a pooling module is used to coarse a graph into sub-graphs or to sum/average over the node representations. We review the graph pooling module in Section 4.3.

In Table 3, we list the details of the inputs and outputs of the main GCNs methods. In particular, we summarize output mechanisms in between each GCN layer and in the final layer of each method. The output mechanisms may involve several pooling operations, which are discussed in Section 4.3.

End-to-end Training Frameworks. Graph convolutional networks 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, graph convolutional networks can learn a robust model that effectively identify the class labels for the unlabeled nodes [14]. 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.** Given a graph dataset, graph-level classification aims to predict the class label(s) for an entire graph [58],

[59], [77], [78]. The end-to-end learning for this task can be done with a framework which combines both graph convolutional layers and the pooling procedure [58], [59]. Specifically, by applying graph convolutional layers, we obtain representation with a fixed number of dimensions for each node in every single graph. Then, we can get the representation of an entire graph through pooling which summarizes the representation vectors of all nodes in a graph. Finally, by applying linear layers and a softmax layer, we can build an end-to-end framework for graph classification. An example is given in Fig 5a.

- **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 [62], [64]. Another 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 being positive pairs. Then a logistic regression layer is applied after the convolutional layers for end-to-end learning [25].

4 Graph Convolution Networks

In this section, we review graph convolution networks (GCNs), the fundamental of many complex graph neural network models. GCNs approaches 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 [79] where the graph convolution operation is interpreted as removing noise from graph signals. Spatial-based approaches formulate graph convolutions as aggregating feature information from neighbors. While GCNs operate on the node level, graph pooling modules can be interleaved with the GCN layer, to coarsen graphs into high-level sub-structures. As shown in Fig 5a, such an architecture design can be used to extract graph-level representations and to perform graph classification tasks. In the following, we introduce spectral-based GCNs, spatial-based GCNs, and graph pooling modules separately.

4.1 Spectral-based Graph Convolutional Networks

Spectral-based methods have a solid foundation in graph signal processing [79]. We first give some basic knowledge background of graph signal processing, after which we review the representative research on the spectral-based GCNs.

4.1.1 Backgrounds

In spectral-based models, graphs are assumed to be undirected. A robust mathematical representation of an undirected graph is the normalized graph Laplacian matrix, 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 $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$, 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 $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues, $\mathbf{\Lambda}_{ii} = \lambda_i$. 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 the nodes of a graph where $\mathbf{x}_i$ 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 resulting signal from graph Fourier transform. To understand graph Fourier transform, from its definition we see that it indeed 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 $\mathbf{x} = \sum_i \hat{\mathbf{x}}_i\mathbf{u}_i$, 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

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

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

$$\mathbf{x} *_G \mathbf{g}_\theta = \mathbf{U}\mathbf{g}_\theta\mathbf{U}^T\mathbf{x} \tag{2}$$

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

4.1.2 Methods of Spectral-based GCNs

Spectral CNN. Bruna et al. [21] propose the first spectral convolution neural network (Spectral CNN). Assuming the filter $\mathbf{g}_\theta = \mathbf{\Theta}^k_{i,j}$ is a set of learnable parameters and considering graph signals of multi-dimension, they define a graph convolution layer as

$$\mathbf{X}^{k+1}_{:,j} = \sigma(\sum_{i=1}^{f_{k-1}} \mathbf{U}\mathbf{\Theta}^k_{i,j}\mathbf{U}^T\mathbf{X}^k_{:,i}) \quad (j = 1, 2, \cdots, f_k) \tag{3}$$

where $\mathbf{X}^k \in \mathbf{R}^{N\times f_{k-1}}$ is the input graph signal, N is the number of nodes, f_{k-1} is the number of input channels and f_k is the number of output channels, $\mathbf{\Theta}^k_{i,j}$ is a diagonal matrix filled with learnable parameters, and σ is a nonlinear transformation.

Chebyshev Spectral CNN (ChebNet). Defferrard et al. [12] propose ChebNet which defines a filter as Chebyshev polynomials of the diagonal matrix of eigenvalues, i.e, $\mathbf{g}_\theta = \sum_{i=0}^{K-1}\theta_i T_k(\tilde{\mathbf{\Lambda}})$, where $\tilde{\mathbf{\Lambda}} = 2\mathbf{\Lambda}/\lambda_{max} - \mathbf{I_N}$. The Chebyshev polynomials are defined recursively by $T_k(\mathbf{x}) = 2\mathbf{x}T_{k-1}(\mathbf{x}) - T_{k-2}(\mathbf{x})$ with $T_0(\mathbf{x}) = 1$ and $T_1(\mathbf{x}) = \mathbf{x}$. As a

TABLE 3: Summary of Graph Convolution Networks

Category	Approach	Inputs (allow edge features?)	Outputs	Output Mechanisms	
				Intermediate	Final
Spectral Based	Spectral CNN (2014) [21]	✗	Graph-level	cluster+max pooling	softmax function
	ChebNet (2016) [12]	✗	Graph-level	efficient pooling	mlp layer+softmax function
	1stChebNet (2017) [14]	✗	Node-level	activation function	softmax function
	AGCN (2018) [23]	✗	Graph-level	max pooling	sum pooling
Spatial Based	GNN (2009) [18]	✓	Node-level	-	mlp layer+softmax function
			Graph-level	-	add a dummy super node
	GGNNs (2015) [19]	✗	Node-level	-	mlp layer/softmax function
			Graph-level	-	sum pooling
	SSE (2018) [20]	✗	Node-level	-	softmax function
	MPNN (2017) [13]	✓	Node-level		softmax function
			Graph-level	-	sum pooling
	GraphSage (2017) [25]	✗	Node-level	activation function	softmax function
	DCNN (2016) [47]	✓	Node-level	activation function	softmax function
			Graph-level	-	mean pooling
	PATCHY-SAN (2016) [27]	✓	Graph-level	-	mlp layer+softmax function
	LGCN (2018) [28]	✗	Node-level	skip connections	mlp layer+softmax function

result, the convolution of a graph signal $\mathbf{x}$ with the defined filter $\mathbf{g}_\theta$ is

$$\begin{aligned}\mathbf{x} *_G \mathbf{g}_\theta &= \mathbf{U}(\sum_{i=0}^{K-1}\theta_i T_k(\tilde{\mathbf{\Lambda}}))\mathbf{U}^T\mathbf{x} \\ &= \sum_{i=0}^{K-1}\theta_i T_i(\tilde{\mathbf{L}})\mathbf{x}\end{aligned} \tag{4}$$

where $\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{max} - \mathbf{I_N}$.

From Equation 4, ChebNet implicitly avoids the computation of the graph Fourier basis, reducing the computation complexity from $O(N^3)$ to $O(KM)$. Since $T_i(\tilde{\mathbf{L}})$ is a polynomial of $\tilde{\mathbf{L}}$ of i^{th} order, $T_i(\tilde{\mathbf{L}})\mathbf{x}$ operates locally on each node. Therefore, the filters of ChebNet are localized in space.

First order of ChebNet (1stChebNet [2]) Kipf et al. [14] introduce a first-order approximation of ChebNet. Assuming $K = 1$ and $\lambda_{max} = 2$, Equation 4 is simplified as

$$\mathbf{x} *_G \mathbf{g}_\theta = \theta_0\mathbf{x} - \theta_1\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{x} \tag{5}$$

To restrain the number of parameters and avoid overfitting, 1stChebNet further assumes $\theta = \theta_0 = -\theta_1$, leading to the following definition of graph convolution,

$$\mathbf{x} *_G \mathbf{g}_\theta = \theta(\mathbf{I_n} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{x} \tag{6}$$

In order to incorporate multi-dimensional graph input signals, 1stChebNet proposes a graph convolution layer which modifies Equation 6,

$$\mathbf{X^{k+1}} = \tilde{\mathbf{A}}\mathbf{X^k}\mathbf{\Theta} \tag{7}$$

where $\tilde{\mathbf{A}} = \mathbf{I_N} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$.

The graph convolution defined by 1stChebNet is localized in space. It bridges the gap between spectral-based methods and spatial-based methods. Each row of the output represents the latent representation of each node obtained by a linear transformation of aggregated information from the node itself and its neighboring nodes with weights specified by the row of $\tilde{\mathbf{A}}$. From the perspective of spatial-based methods, the adjacency matrix $\tilde{\mathbf{A}}$ not necessarily need to be symmetric. For example, it can be the form of $\mathbf{D}^{-1}\mathbf{A}$. The main drawback of 1stChebNet is that the computation cost increases exponentially with the increase of the number of 1stChebNet layers during batch training. Each node in the last layer has to expand its neighborhood recursively across previous layers. Chen et al. [48] assume the rescaled adjacency matrix $\tilde{\mathbf{A}}$ in Equation 7 comes from a sampling distribution. Under this assumption, the technique of Monte Carlo and variance reduction techniques are used to facilitate the training process. Chen et al. [49] reduce the receptive field size of the graph convolution to an arbitrary small scale by sampling neighborhoods and using historical hidden representations. Huang et al. [57] propose an adaptive layer-wise sampling approach to accelerate the training of 1stChebNet, where sampling for the lower layer is conditioned on the top one. This method is also applicable for explicit variance reduction.

Adaptive Graph Convolution Network (AGCN). To explore hidden structural relations unspecified by the graph Laplacian matrix, Li et al. [23] propose the adaptive graph convolution network (AGCN). AGCN augments a graph with a so-called residual graph, which is constructed by computing a pairwise distance of nodes. Despite being able to capture complement relational information, AGCN incurs expensive $O(N^2)$ computation.

4.1.3 Summary

Spectral CNN [21] relies on the eigen-decomposition of the Laplacian matrix. It has three effects. 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 $O(N^3)$ computation and $O(N^2)$ memory. Filters defined by ChebNet [12] and 1stChebNet [14] are localized in space. The learned weights can be shared across different locations in a graph. However, a common drawback of spectral methods is they need to load the whole graph into the memory to perform graph convolution, which is not efficient in handling big graphs.

2. Due to its impressive performance in many node classification tasks, 1stChebNet is simply termed as **GCN** and is considered as a strong baseline in the research community.

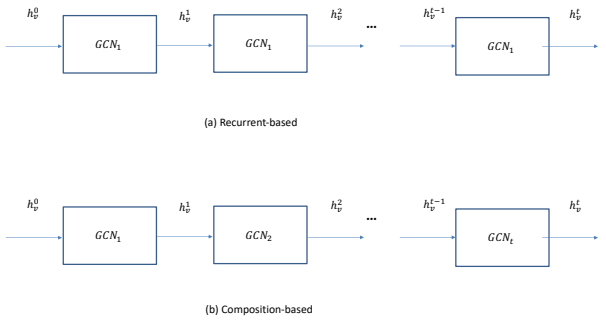


Fig. 7: Recurrent-based v.s. Composition-based Spatial GCNs.

4.2 Spatial-based Graph Convolutional Networks

Imitating the convolution operation of a conventional convolution neural network on an image, spatial-based methods define graph convolution based on a node's spatial relations. To relate images with graphs, images can be considered as a special form of a graph with each pixel representing a node. As illustrated in Figure 1a, each pixel is directly connected to its nearby pixels. With a 3×3 window, the neighborhood of each node is its surrounding eight pixels. The positions of these eight pixels indicate an ordering of a node's neighbors. A filter is then applied to this 3×3 patch by taking the weighted average of pixel values of the central node and its neighbors across each channel. Due to the specific ordering of neighboring nodes, the trainable weights are able to be shared across different locations. Similarly, for a general graph, the spatial-based graph convolution takes the aggregation of the central node representation and its neighbors representation to get a new representation for this node, as depicted by Figure 1b. To explore the depth and breadth of a node's receptive field, a common practice is to stack multiple graph convolution layer together. According to the different approaches of stacking convolution layers, spatial-based GCNs can be further divided into two categories, *recurrent-based* and *composition-based* spatial GCNs. Recurrent-based methods apply the same graph convolution layer to update hidden representations, while composition-based methods apply a different graph convolution layer to update hidden representations. Figure 7 illustrates this difference. In the following, we give an overview of these two branches.

4.2.1 Recurrent-based Spatial GCNs

The main idea of recurrent-based methods is to update a node's latent representation recursively until a stable fixed point is reached. This is done by imposing constraints on recurrent functions [18], employing gate recurrent unit architectures [19], updating node latent representations asynchronously and stochastically [20]. In the following, we will introduce these three methods.

Graph Neural Networks(GNNs) Being one of the earliest works on graph neural networks, GNNs recursively update node latent representations until convergence [18]. In other words, from the perspective of the diffusion process, each node exchanges information with its neighbors until equilibrium is reached. To handle heterogeneous graphs, the spatial graph convolution of GNNs is defined as

$$\mathbf{h}_v^t = f(\mathbf{l}_\mathbf{v}, \mathbf{l}_{co}[v], \mathbf{h}_{ne}^{t-1}[v], \mathbf{l}_{ne}[v]) \tag{8}$$

where $\mathbf{l}_v$ denotes the label attributes of node v, $\mathbf{l}_{co}[v]$ denotes the label attributes of corresponding edges of node v, $\mathbf{h}_{ne}^t[v]$ denotes the hidden representations of node v's neighbors at time step t, and $\mathbf{l}_{ne}[v]$ denotes the label attributes of node v's neighbors.

To ensure convergence, the recurrent function $f(\cdot)$ must be a contraction mapping, which shrinks the distance between two points after mapping. In the case of $f(\cdot)$ is a neural network, a penalty term has to be imposed on the Jacobian matrix of parameters. GNNs uses the Almeida-Pineda algorithm [80], [81] to train its model. The core idea is to run the propagation process to reach fixed points and then perform the backward procedure given the converged solution.

Gated Graph Neural Networks (GGNNs) employs gated recurrent units(GRU) [82] as the recurrent function, reducing the recurrence to a fixed number of steps. The spatial graph convolution of GGNNs is defined as

$$\mathbf{h}_v^t = GRU(\mathbf{h}_v^{t-1}, \sum_{u \in N(v)} \mathbf{W}\mathbf{h}_u^t) \tag{9}$$

Different from GNNs, GGNNs use back-propagation through time (BPTT) to learn the parameters. The advantage is that it no longer needs to constrain parameters to ensure convergence. However, the downside of training by BPTT is that it sacrifices efficiency both in time and memory. This is especially problematic for large graphs, as GGNNs need to run the recurrent function multiple times over all nodes, requiring intermediate states of all nodes to be stored in memory.

Stochastic Steady-state Embedding (SSE). To improve the learning efficiency, the SSE algorithm [20] updates the node latent representations stochastically in an asynchronous fashion. As shown in Algorithm 1, SSE recursively estimates node latent representations and updates the parameters with sampled batch data. To ensure convergence to steady states, the recurrent function of SSE is defined as a weighted average of the historical states and new states,

$$\mathbf{h_v}^t = (1-\alpha)\mathbf{h_v}^{t-1} + \alpha \mathbf{W_1} \sigma(\mathbf{W_2}[\mathbf{x_v}, \sum_{u \in N(v)} [\mathbf{h_u}^{t-1}, \mathbf{x_u}]]) \tag{10}$$

Though summing neighborhood information implicitly considers node degree, it remains questionable whether the scale of this summation affects the stability of this algorithm.

4.2.2 Composition Based Spatial GCNs

Composition-based methods update the nodes' representations by stacking multiple graph convolution layers.

Message Passing Neural Networks (MPNNs). Gilmer et al. [13] generalize several existing graph convolution networks including [12], [14], [19], [21], [56], [83], [84] into a unified framework named Message Passing Neural Networks (MPNNs). The MPNNs consists of two phases, the message passing phase and the readout phase. The message passing

ALGORITHM 1: Learning with Stochastic Fixed Point Iteration [20]

Initialize parameters,$\{\mathbf{h}_v^0\}_{v\in\mathbf{V}}$
for $k = 1$ ***to*** K **do**
  for $t = 1$ ***to*** T **do**
    Sample n nodes from the whole node set $\mathbf{V}$
    Use Equation 10 to update hidden representations of sampled n nodes
  end
  for $p = 1$ ***to*** P **do**
    Sample m nodes from the labeled node set $\mathbf{V}$
    Forward model according to Equation 10
    Back-propagate gradients
  end
end

phase actually runs T-step spatial-based graph convolutions. The graph convolution operation is defined through a message function $M_t(\cdot)$ and an updating function $U_t(\cdot)$ according to

$$\mathbf{h}_v^t = \mathbf{U}_t(\mathbf{h}_v^{t-1}, \sum_{w\in N(v)} M_t(\mathbf{h}_v^{t-1}, \mathbf{h}_w^{t-1}, \mathbf{e}_{vw})) \tag{11}$$

The readout phase is actually a pooling operation which produces a representation of the entire graph based on hidden representations of each individual node. It is defined as

$$\hat{\mathbf{y}} = R(\mathbf{h}_v^T | v \in G) \tag{12}$$

Through the output function $R(\cdot)$, the final representation $\hat{y}$ is used to perform graph-level prediction tasks. The authors present that several other graph convolution networks fall into their framework by assuming different forms of $U_t(\cdot)$ and $M_t(\cdot)$.

GraphSage [25] introduces the notion of the aggregation function to define graph convolution. The aggregation function essentially assembles a node's neighborhood information. It must be invariant to permutations of node orderings such as mean, sum and max function. The graph convolution operation is defined as,

$$\mathbf{h}_v^t = \sigma(\mathbf{W}^t \cdot aggregate_t(\mathbf{h}_v^{t-1}, \{\mathbf{h}_u^{t-1}, \forall u \in \mathcal{N}(v)\}) \tag{13}$$

Instead of updating states over all nodes, GraphSage proposes a batch-training algorithm, which improves scalability for large graphs. The learning process of GraphSage consists of three steps. First, it samples a node's local k-hop neighborhood with fixed-size. Second, it derives the central node's final state by aggregating its neighbors feature information. Finally, it uses the central node's final state to make predictions and backpropagate errors. This process is illustrated in Figure 8.

Assuming the number of neighbors to be sampled at t^{th} hop is s_t, the time complexity of GraphSage in one batch is $O(\prod_{t=1}^{T} s_t)$. Therefore the computation cost increases exponentially with the increase of t. This prevents GraphSage from having a deep architecture. However, in practice, the authors find that with $t = 2$ GraphSage already achieves high performance.

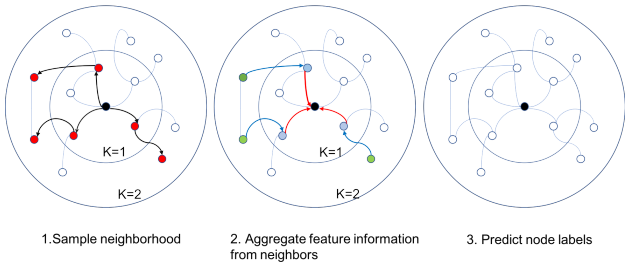


Fig. 8: Learning Process of GraphSage [25]

4.2.3 Miscellaneous Variants of Spatial GCNs

Diffusion Convolution Neural Networks (DCNN) [47] proposed a graph convolution network which encapsulates the graph diffusion process. A hidden node representation is obtained by independently convolving inputs with power series of transition probability matrix. The diffusion convolution operation of DCNN is formulated as

$$\mathbf{Z}_{i,j,:}^m = f(\mathbf{W}_{j,:} \odot \mathbf{P}_{i,j,:}^m \mathbf{X}_{i,:}^m) \tag{14}$$

In Equation 14, $\mathbf{z}_{i,j,:}^m$ denotes the hidden representation of node i for hop j in graph m, $\mathbf{P}_{:,j,:}^m$ denotes the probability transition matrix of hop j in graph m, and $\mathbf{X}_{i,:}^m$ denote the input features of node i in graph m, where $\mathbf{z}^m \in \mathbf{R}^{N_m \times H \times F}$, $\mathbf{W} \in \mathbf{R}^{H \times F}$, $\mathbf{P}^m \in \mathbf{R}^{N_m \times H \times N_m}$ and $\mathbf{X}^m \in \mathbf{R}^{N_m \times F}$.

Though covering a larger receptive field through higher orders of the transition matrix, the DCNN model needs $O(N_m^2 H)$ memory, causing severe problems when applying it to large graphs.

PATCHY-SAN [27] uses a standard convolution neural network (CNN) to solve graph classification tasks. To do this, it converts graph-structured data into grid-structured data. First, it selects a fixed number of nodes for each graph using a graph labeling procedure. A graph labeling procedure essentially assigns a ranking to each node in the graph, which can be based on node-degree, centrality, Weisfeiler-Lehman color [85] [86] and etc. Second, as each node in a graph can have a different number of neighbors, PATCHY-SAN selects and orders a fixed number of neighbors for each node according to their graph labelings. Finally, after the grid-structured data with fixed-size is formed, PATCHY-SAN employed standard CNN to learn the graph hidden representations. Utilizing standard CNN in GCNs has the advantage of keeping shift-invariance, which relies on the sorting function. As a result, the ranking criteria in the node selection and ordering process is of paramount importance. In PATCHY-SAN, the ranking is based on graph labelings. However, graph labelings only take graph structures into consideration, ignoring node feature information.

Large-scale Graph Convolution Networks (LGCN). In follow-up work, large-scale graph convolution networks (LGCN) [28] proposes a ranking method based on node feature information. Unlike PATCHY-SAN, LGCN uses standard CNN to generate node-level outputs. For each node, LGCN assembles a feature matrix of its neighborhood and sorts this feature matrix along each column. The first k rows of the sorted feature matrix are taken as the input

grid-data for the target node. In the end, LGCN applies 1D CNN on the resultant inputs to get the target node's hidden representation. While deriving graph labelings in PATCHY-SAN requires complex pre-processing, sorting feature values in LGCN does not need a pre-processing step, making it more efficient. To suit the scenario of large-scale graphs, LGCN proposes a subgraph training strategy, which puts the sampled subgraphs into a mini-batch.

Mixture Model Network (MoNet) [26] unifies standard CNN with convolutional architectures on non-Euclidean domains. While several spatial-based approaches ignore the relative positions between a node and its neighbors when aggregating neighborhood feature information, MoNet introduce pseudo-coordinates and weight functions to let the weight of a node's neighbor be determined by the relative position (pseudo-coordinates) between the node and its neighbor. Under such a framework, several approaches on manifolds such as Geodesic CNN (GCNN) [87], Anisotropic CNN(ACNN) [88], Spline CNN [89], and on graphs such as GCN [14], DCNN [47] can be generalized as special instances of MoNet. These approaches under the framework of MoNet have fixed weight functions. MoNet instead proposes a Gaussian kernel with learnable parameters to freely adjust the weight function.

4.2.4 Summary

Spatial-based methods define graph convolutions via aggregating feature information from neighbors. According to different ways of stacking graph convolution layers, spatial-based methods are split into two groups, recurrent-based and composition-based. While recurrent-based approaches try to obtain nodes' steady states, composition-based approaches try to incorporate higher orders of neighborhood information. In each layer, both two groups have to update hidden states over all nodes during training. However, it is inefficient to store all the intermediate states into memory. To address this issue, several training strategies have been proposed, including sub-graph training for composition-based approaches such as GraphSage [25] and stochastically asynchronous training for recurrent-based approaches such as SSE [20]. In addition, recent advances in spatial-based approaches tend to construct more complex network architectures. For examples, GeniePath [51] leverages gating mechanisms to control the depth and breadth of a node's neighborhood. DualGCN [52] devises two graph convolution networks to jointly embed the local consistency and the global consistency knowledge of a graph. Tran et al. [90] introduce a hyper-parameter to influence the receptive field size of a node.

4.3 Graph Pooling Modules

When generalizing convolutional neural networks to graph-structured data, another key component, graph pooling module, is also of vital importance, particularly for graph-level classification tasks [58], [59], [91]. According to Xu et al. [92], pooling-assisted GCNs are as powerful as the Weisfeiler-Lehman test [85] in distinguishing graph structures. Similar to the original pooling layer which comes with CNNs, graph pooling module could easily reduce the variance and computation complexity by down-sampling from original feature data. Mean/max/sum pooling is the most primitive and most effective way of implementing this since calculating the mean/max/sum value in the pooling window is rapid.

$$\mathbf{h}_G = mean/max/sum(\mathbf{h}_1^T, \mathbf{h}_2^T, ..., \mathbf{h}_n^T) \tag{15}$$

Henaff et al. [22] prove 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.

Defferrard et al. optimize max/min pooling and devise an efficient pooling strategy in their approach ChebNet [12]. Input graphs are first processed by the coarsening process described in Fig 5a . After coarsening, the vertices of the input graph and its coarsened versions are reformed in a balanced binary tree. Arbitrarily ordering the nodes at the coarsest level then propagating this ordering to the lower level in the balanced binary tree would finally produce a regular ordering in the finest level. Pooling such a rearranged 1D signal is much more efficient than the original.

Zhang et al. also propose a framework DGCNN [58] with a similar pooling strategy named SortPooling which performs pooling by rearranging vertices to a meaningful order. Different from ChebNet [12], DGCNN sorts vertices according to their structural roles within the graph. The graph's unordered vertex features from spatial graph convolutions are treated as continuous WL colors [85], and they are then used to sort vertices. In addition to sorting the vertex features, it unifies the graph size to k by truncating/extending the graph's feature tensor. The last $n-k$ rows are deleted if $n > k$, otherwise $k - n$ zero rows are added. This method enhances the pooling network to improve the performance of GCNs by solving one challenge underlying graph-structured tasks which is referred to as permutation invariant. Verma and Zhang propose graph capsule networks [93] which further explore the permutation invariant problem for graph data.

Recently a pooling module, DIFFPOOL [59], is proposed which can generate hierarchical representations of graphs and can be combined with not only CNNs, but also various graph neural network architectures in an end-to-end fashion. Compared to all previous coarsening methods, DIFFPOOL does not simply cluster the nodes in one graph but provide a general solution to hierarchically pool nodes across a broad set of input graphs. This is done by learning a cluster assignment matrix $\mathbf{S}$ at layer l referred to as $\mathbf{S}^{(l)} \in \mathbf{R}^{n_l \times n_l + 1}$. Two separate GNNs with both input cluster node features $\mathbf{X}^{(l)}$ and coarsened adjacency matrix $\mathbf{A}^{(l)}$ are being used to generate the assignment matrix $\mathbf{S}^{(l)}$ and embedding matrices $\mathbf{Z}^{(l)}$ as follows:

$$\mathbf{Z}^{(l)} = GNN_{l,embed}(\mathbf{A}^{(l)}, \mathbf{X}^{(l)}) \tag{16}$$

$$\mathbf{S}^{(l)} = softmax(GNN_{l,pool}(\mathbf{A}^{(l)}, \mathbf{X}^{(l)})) \tag{17}$$

Equation 16 and 17 can be implemented with any standard GNN module, which processes the same input data but has distinct parametrizations since the roles they play in the framework are different. The $GNN_{l,embed}$ will produce new embeddings while the $GNN_{l,pool}$ generates a

probabilistic assignment of the input nodes to n_{l+1} clusters. The Softmax function is applied in a row-wise fashion in Equation 17. As a result, each row of $\mathbf{S}^{(l)}$ corresponds to one of the n_l nodes(or clusters) at layer l, and each column of $\mathbf{S}^{(l)}$ corresponds to one of the n_l at the next layer. Once we have $\mathbf{Z}^{(l)}$ and $\mathbf{S}^{(l)}$, the pooling operation comes as follows:

$$\mathbf{X}^{(l+1)} = {\mathbf{S}^{(l)}}^T \mathbf{Z}^{(l)} \in \mathbf{R}^{n_{l+1} \times d} \tag{18}$$

$$\mathbf{A}^{(l+1)} = {\mathbf{S}^{(l)}}^T \mathbf{A}^{(l)} \mathbf{S}^{(l)} \in \mathbf{R}^{n_{l+1} \times n_{l+1}} \tag{19}$$

Equation 18 takes the cluster embeddings $\mathbf{Z}^{(l)}$ then aggregates these embeddings according to the cluster assignments $\mathbf{S}^{(l)}$ to calculate embedding for each of the n_{l+1} clusters. Initial cluster embedding would be node representation. Similarly, Equation 19 takes the adjacency matrix $\mathbf{A}^{(l)}$ as inputs and generates a coarsened adjacency matrix denoting the connectivity strength between each pair of the clusters.

Overall, DIFFPOOL [59] redefines the graph pooling module by using two GNNs to cluster the nodes. Any standard GCN module is able to combine with DIFFPOOL, not only to achieve enhanced performance but also to speed up the convolution operation.

4.4 Comparison Between Spectral and Spatial Models

As the earliest convolutional networks for graph data, spectral-based models have achieved impressive results in many graph related analytics tasks. These models are appealing in that they have a theoretical foundation in graph signal processing. By designing new graph signal filters [24], we can theoretically design new graph convolution networks. However, there are several drawbacks to spectral-based models. We illustrate this in the following from three aspects, efficiency, generality and flexibility.

In terms of efficiency, the computational cost of spectral-based models increases dramatically with the graph size because they either need to perform eigenvector computation [21] or handle the whole graph at the same time, which makes them difficult to parallel or scale to large graphs. Spatial based models have the potential to handle large graphs as they directly perform the convolution in the graph domain via aggregating the neighboring nodes. The computation can be performed in a batch of nodes instead of the whole graph. When the number of neighboring nodes increases, sampling techniques [25], [28] can be developed to improve efficiency.

In terms of generality, spectral-based models assumed a fixed graph, making them generalize poorly to new or different graphs. Spatial-based models, on the other hand, perform graph convolution locally on each node, where weights can be easily shared across different locations and structures.

In terms of flexibility, spectral-based models are limited to work on undirected graphs. There is no clear definition of the Laplacian matrix on directed graphs so that the only way to apply spectral-based models to directed graphs is to transfer directed graphs to undirected graphs. Spatial-based models are more flexible to deal with multi-source inputs such as edge features and edge directions because these inputs can be incorporated into the aggregation function (e.g. [13], [18], [54], [55], [56]).

As a result, spatial models have attracted increasing attention in recent years [26].

5 Beyond Graph Convolutional Networks

In this section, we review other graph neural networks including graph attention neural networks, graph autoencoder, graph generative networks, and graph spatial-temporal networks. In Table 4, we provide a summary of the main approaches under each category.

5.1 Graph Attention Networks

Attention mechanisms have almost become a standard in sequence-based tasks [94]. The virtue of attention mechanisms is their ability to focus on the most important parts of an object. This specialty has been proven to be useful for many tasks, such as machine translation and natural language understanding. Thanks to the increased model capacity of attention mechanisms, graph neural networks also benefit from this by using attention during aggregation, integrating outputs from multiple models, and generating importance-oriented random walks. In this section, we will discuss how attention mechanisms are being used in graph-structured data.

5.1.1 Methods of Graph Attention Networks

Graph Attention Network (GAT) [15] is a spatial-based graph convolution network where the attention mechanism is involved in determining the weights of a node's neighbors when aggregating feature information. The graph convolution operation of GAT is defined as,

$$\mathbf{h}_i^t = \sigma\Big(\sum_{j \in \mathcal{N}_i} \alpha(\mathbf{h}_i^{t-1}, \mathbf{h}_j^{t-1}) \mathbf{W}^{t-1} \mathbf{h}_j^{t-1}\Big) \tag{20}$$

where $\alpha(\cdot)$ is an attention function which adaptively controls the contribution of a neighbor j to the node i. In order to learn attention weights in different subspaces, GAT uses multi-head attentions.

$$\mathbf{h}_i^t = \|_{k=1}^K \sigma\Big(\sum_{j \in \mathcal{N}_i} \alpha_k(\mathbf{h}_i^{t-1}, \mathbf{h}_j^{t-1}) W_k^{t-1} \mathbf{h}_j^{t-1}\Big) \tag{21}$$

where $\|$ denotes concatenation.

Gated Attention Network (GAAN) [29] also employs the multi-head attention mechanism in updating a node's hidden state. However, rather than assigning equal weight to each head, GAAN introduces a self-attention mechanism which computes a different weight for each head. The updating rule is defined as,

$$\mathbf{h}_i^t = \phi_o(\mathbf{x}_i \oplus \|_{k=1}^K g_i^k \sum_{j \in \mathcal{N}_i} \alpha_k(\mathbf{h}_i^{t-1}, \mathbf{h}_j^{t-1}) \phi_v(\mathbf{h}_j^{t-1})) \tag{22}$$

where $\phi_o(\cdot)$ and $\phi_v(\cdot)$ denote feedforward neural networks and g_i^k is the attention weight of the k^{th} attention head.

Graph Attention Model (GAM) [60] proposes a recurrent neural network model to solve graph classification problems, which processes informative parts of a graph

TABLE 4: Summary of Alternative Graph Neural Networks (Graph Convolutional Networks Excluded). We summarize methods based on their inputs, outputs, targeted tasks, and whether a method is GCN-based. Inputs indicate whether a method suits attributed graphs (A), directed graphs (D), and spatial-temporal graphs (S).

Category	Approaches	Inputs			Outputs	Tasks	GCN Based
		A	D	S			
Graph Attention Networks	GAT (2017) [15]	✓	✓	✗	node labels	node classification	✓
	GAAN (2018) [29]	✓	✓	✗	node labels	node classification	✓
	GAM (2018) [60]	✓	✓	✗	graph labels	graph classification	✗
	Attention Walks (2018) [61]	✗	✗	✗	node embedding	network embedding	✗
Graph Auto-encoder	GAE (2016) [62]	✓	✗	✗	reconstructed adajacency matrix	network embedding	✓
	ARGA (2018) [64]	✓	✗	✗	reconstructed adajacency matrix	network embedding	✓
	NetRA (2018) [65]	✗	✗	✗	reconstructed sequences of random walks	network embedding	✗
	DNGR (2016) [42]	✗	✗	✗	reconstructed PPMI matrix	network embedding	✗
	SDNE (2016) [43]	✗	✓	✗	reconstructed adajacency matrix	network embedding	✗
	DNRE (2018) [66]	✓	✗	✗	reconstructed node embedding	network embedding	✗
Graph Generative Networks	MolGAN (2018) [69]	✓	✗	✗	new graphs	graph generation	✓
	DGMG (2018) [68]	✗	✗	✗	new graphs	graph generation	✓
	GraphRNN (2018) [67]	✗	✗	✗	new graphs	graph generation	✗
	NetGAN (2018) [70]	✗	✗	✗	new graphs	graph generation	✗
Graph Spatial-Temporal Networks	DCRNN (2018) [73]	✗	✗	✓	node value vectors	spatial-temporal forecasting	✓
	CNN-GCN (2017) [74]	✗	✗	✓	node value vectors	spatial-temporal forecasting	✓
	ST-GCN (2018) [75]	✗	✗	✓	graph labels	spatial-temporal classification	✓
	Structural RNN (2016) [76]	✗	✗	✓	node labels/value vectors	spatial-temporal forecasting	✗

by adaptively visiting a sequence of important nodes. The GAM model is defined as

$$\mathbf{h}_t = \mathbf{f}_h(\mathbf{f}_s(\mathbf{r}_{t-1}, \mathbf{v}_{t-1}, g; \theta_s), \mathbf{h}_{t-1}; \theta_h) \tag{23}$$

where $\mathbf{f}_h(\cdot)$ is an LSTM network, $\mathbf{f}_s$ is the step network which takes a step from the current node $\mathbf{v}_{t-1}$ to one of its neighbors $\mathbf{c}_t$, prioritizing those whose type have a higher rank in $\mathbf{v}_{t-1}$ which is generated by a policy network:

$$\mathbf{r}_t = f_r(\mathbf{h}_t; \theta_r) \tag{24}$$

where $\mathbf{r}_t$ is a stochastic rank vector which indicates which node is more important and thus should be further explored with high priority, $\mathbf{h}_t$ contains the historical information that the agent has aggregated from the exploration of a graph, and is used to make a prediction for the graph label.

Attention Walks [61] learns node embeddings through random walks. Unlike DeepWalk [41] using fixed apriori, Attention Walks factorizes the co-occurrence matrix with differentiable attention weights.

$$E[\mathbf{D}] = \tilde{\mathbf{P}}^{(0)} \sum_{k=1}^{C} a_k(\mathbf{P})^k \tag{25}$$

where $\mathbf{D}$ denotes the co-occurrence matrix, $\tilde{\mathbf{P}}^{(0)}$ denotes the initial position matrix, and $\mathbf{P}$ denotes the probability transition matrix.

5.1.2 Summary

Attention mechanisms contribute to graph neural networks in three different ways, namely assigning attention weights to different neighbors when aggregating feature information, ensembling multiple models according to attention weights, and using attention weights to guide random walks. Despite categorizing GAT [15] and GAAN [29] under the umbrella of graph attention networks, they can also be considered as spatial-based graph convolution networks at the same time. The advantage of GAT [15] and GAAN [29] is that they can adaptively learn the importance weights of neighbors as illustrated in Fig 6. However, the computation cost and memory consumption increase rapidly as the attention weights between each pair of neighbors must be computed.

5.2 Graph Auto-encoders

Graph auto-encoders are one class of network embedding approaches which aim at representing network vertices into a low-dimensional vector space by using neural network architectures. A typical solution is to leverage multi-layer perceptrons as the encoder to obtain node embeddings, where a decoder reconstructs a node's neighborhood statistics such as positive pointwise mutual information (PPMI) [42] or the first and second order of proximities [43]. Recently, researchers have explored the use of GCN [14] as an encoder, combining GCN [14] with GAN [95], or combining LSTM [7] with GAN [95] in designing a graph auto-encoder. We will first review GCN based autoencoder and then summarize other variants in this category.

5.2.1 GCN Based Auto-encoders

Graph Auto-encoder (GAE) [62] firstly integrates GCN [14] into a graph auto encoder framework. The encoder is defined as

$$\mathbf{Z} = GCN(\mathbf{X}, \mathbf{A}) \tag{26}$$

while the decoder is defined as

$$\hat{\mathbf{A}} = \sigma(\mathbf{Z}\mathbf{Z}^{\mathbf{T}}) \tag{27}$$

The framework of GAE is also depicted in Fig 5b. The GAE can be trained in a variational manner, i.e., to minimize the variational lower bound L:

$$L = E_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}[\log_p(\mathbf{A}|\mathbf{Z})] - KL[q(\mathbf{Z}|\mathbf{X},\mathbf{A})||p(\mathbf{Z})] \quad (28)$$

Adversarially Regularized Graph Autoencoder (ARGA) [64] employs the training scheme of generative adversarial networks (GANs) [95] to regularize a graph auto-encoder. In ARGA, an encoder encodes a node's structural information with its features into a hidden representation by GCN [14], and a decoder reconstructs the adjacency matrix from the outputs of the encoder. The GANs play a min-max game between a generator and a discriminator in training generative models. A generator generates "faked samples" as real as possible while a discriminator makes its best to distinguish the "faked samples" from the real ones. A GAN helps the ARGA to regularize the learned hidden representations of nodes to follow a prior distribution. In detail, the encoder, working as a generator, tries to make the learned node hidden representations indistinguishable from a real prior distribution. A discriminator, on the other side, tries to identify whether the learned node hidden representations are generated from the encoder or from a real prior distribution.

5.2.2 Miscellaneous Variants of Graph Auto-encoders

Network Representations with Adversarially Regularized Autoencoders (NetRA) [65] is a graph auto-encoder framework which shares a similar idea with ARGA. It also regularizes node hidden representations to comply with a prior distribution via adversarial training. Instead of reconstructing the adjacency matrix, they recover node sequences sampled from random walks by a sequence-to-sequence architecture [96].

Deep Neural Networks for Graph Representations (DNGR) [42] uses the stacked denoising autoencoder [97] to reconstruct the pointwise mutual information matrix(PPMI). The PPMI matrix intrinsically captures nodes co-occurrence information when a graph is serialized as sequences by random walks. Formally, the PPMI matrix is defined as

$$\mathbf{PPMI}_{v_1,v_2} = max(\log(\frac{count(v_1,v_2)\cdot|D|}{count(v_1)count(v_2)}), 0) \quad (29)$$

where $|D| = \sum_{v_1,v_2} count(v_1,v_2)$ and $v_1, v_2 \in V$. The stacked denoising autoencoder is able to learn highly non-linear regularity behind data. Different from conventional neural autoencoders, it adds noise to inputs by randomly switching entries of inputs to zero. The learned latent representation is more robust especially when there are missing values present.

Structural Deep Network Embedding (SDNE) [43] uses stacked auto encoder to preserve nodes first-order proximity and second-order proximity jointly. The first-order proximity is defined as the distance between a node's hidden representation and its neighbor's hidden representation. The goal for the first-order proximity is to drive representations of adjacent nodes close to each other as much as possible. Specifically, the loss function L_{1st} is defined as

$$L_{1st} = \sum_{i,j=1}^{n} \mathbf{A}_{i,j}||\mathbf{h}_i^{(k)} - \mathbf{h}_j^{(k)}||^2 \quad (30)$$

The second-order proximity is defined as the distance between a node's input and its reconstructed inputs where the input is the corresponding row of the node in the adjacency matrix. The goal for the second-order proximity is to preserve a node's neighborhood information. Concretely, the loss function L_{2nd} is defined as

$$L_{2nd} = \sum_{i=1}^{n} ||(\hat{\mathbf{x}}_i - \mathbf{x}_i) \odot \mathbf{b}_i||^2 \quad (31)$$

The role of vector b_i is to penalize non-zero elements more than zero elements since the inputs are highly sparse. In detail, $b_{i,j} = 1$ if $A_{i,j} = 0$ and $b_{i,j} = \beta > 1$ if $A_{i,j} = 1$. Overall, the objective function is defined as

$$L = L_{2nd} + \alpha L_{1st} + \lambda L_{reg} \quad (32)$$

where L_{reg} is the L_2 regularization term.

Deep Recursive Network Embedding (DRNE) [66] directedly reconstructs a node's hidden state instead of the whole graph statistics. Using an aggregation function as the encoder, DRNE designs the loss function as,

$$L = \sum_{v\in V} ||\mathbf{h}_v - aggregate(\mathbf{h}_u|u \in N(v))||^2 \quad (33)$$

One innovation of DRNE is that it chooses an LSTM as the aggregation function where the neighbors' sequence is ordered by their node degree.

5.2.3 Summary

DNGR and SDNE learn node embeddings only given the topological structures, while GAE, ARGA, NetRA, DRNE learn node embeddings when both topological information and node content features are available. One challenge of graph auto-encoders is the sparsity of the adjacency matrix A, causing the number of positive entries of the decoder to be far less than the negative ones. To tackle this issue, DNGR reconstructs a denser matrix namely the PPMI matrix, SDNE imposes a penalty to zero entries of the adjacency matrix, GAE reweights the terms in the adjacency matrix, and NetRA linearizes Graphs into sequences.

5.3 Graph Generative Networks

The goal of graph generative networks is to generate graphs given an observed set of graphs. Many approaches to graph generative networks are domain specific. For instance, in molecular graph generation, some works model a string representation of molecular graphs called SMILES [98], [99], [100], [101]. In natural language processing, generating a semantic or a knowledge graph is often conditioned on a given sentence [102], [103]. Recently, several general approaches have been proposed. Some work factor the generation process as forming nodes and edges alternatively [67], [68] while others employ generative adversarial training [69], [70]. The methods in this category either employ GCN as building blocks or use different architectures.

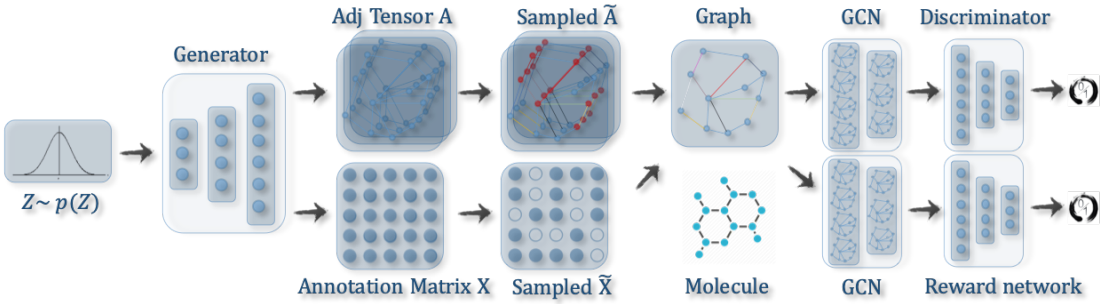


Fig. 9: Framework of MolGAN [70]. A generator first samples an initial vector from a standard normal distribution. Passing this initial vector through a neural network, the generator outputs a dense adjacency matrix A and a corresponding feature matrix X. Next, the generator produces a sampled discrete $\tilde{A}$ and $\tilde{X}$ from categorical distributions based on A and X. Finally, GCN is used to derive a vector representation of the sampled graph. Feeding this graph representation to two distinct neural networks, a discriminator and a reward network output a score between zero and one separately, which will be used as feedback to update the model parameters.

5.3.1 GCN Based Graph Generative Networks

Molecular Generative Adversarial Networks (MolGAN) [69] integrates relational GCN [104], improved GAN [105] and reinforcement learning (RL) objective to generate graphs with desired properties. The GAN 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 faked graph along with its feature matrix while the discriminator aims to distinguish the faked 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. The framework of MolGAN is described in Fig 9.

Deep Generative Models of Graphs (DGMG) [68] utilizes spatial-based graph convolution networks to obtain a hidden representation of an existing graph. The decision process of generating nodes and edges is conditioned on the resultant graph representation. Briefly, DGMG recursively proposes a node to a growing graph until a stopping criterion is evoked. In each step after adding a new node, DGMG repeatedly decides whether to add an edge to the added node until the decision turns to false. If the decision is true, it evaluates the probability distribution of connecting the newly added node to all existing nodes and samples one node from the probability distribution. After a new node and its connections are added to the existing graph, DGMG updates the graph representation again.

5.3.2 Miscellaneous Graph Generative Networks

GraphRNN [67] exploits deep graph generative models through two-level recurrent neural networks. The graph-level RNN adds a new node each time to a node sequence while the edge level RNN produces a binary sequence indicating connections between the newly added node and previously generated nodes in the sequence. To linearize a graph into a sequence of nodes for training the graph level RNN, GraphRNN adopts the breadth-first-search (BFS) strategy. To model the binary sequence for training the edge-level RNN, GraphRNN assumes multivariate Bernoulli or conditional Bernoulli distribution.

NetGAN [70] combines LSTM [7] with Wasserstein GAN [106] to generate graphs from a random-walk-based approach. The GAN framework consists of two modules, a generator and a discriminator. The generator makes its best effort to generate plausible random walks through an LSTM network while the discriminator tries to distinguish faked random walks from the real ones. After training, a new graph is obtained by normalizing a co-occurrence matrix of nodes which occur in a set of random walks.

5.3.3 Summary

Evaluating generated graphs remains a difficult problem. Unlike synthesized images or audios, which can be directly assessed by human experts, the quality of generated graphs is difficult to inspect visually. MolGAN and DGMG make use of external knowledge to evaluate the validity of generated molecule graphs. GraphRNN and NetGAN evaluate generated graphs by graph statistics (e.g. node degrees). Whereas DGMG and GraphRNN generate nodes and edges sequentially, MolGAN and NetGAN generate nodes and edges jointly. According to [71], the disadvantage of the former approaches is that when graphs become large, modeling a long sequence is not realistic. The challenge of the latter approaches is that the global properties of a graph are difficult to control. A recent approach [71] adopts variational auto-encoder to generate a graph by proposing the adjacency matrix, imposing penalty terms to address validity constraints. However, as the output space of a graph with n nodes is n^2, none of these methods is scalable to large graphs.

5.4 Graph Spatial-Temporal Networks

Graph spatial-temporal networks capture spatial and temporal dependencies of a spatial-temporal graph simultaneously. Spatial-temporal graphs have a global graph structure with inputs to each node which are changing across time. For instance, in traffic networks, each sensor taken as a node records the traffic speed of a certain road continuously where the edges of the traffic network are determined by the distance between pairs of sensors. The goal of graph spatial-temporal networks can be forecasting future node

values or labels, or predicting spatial-temporal graph labels. Recent studies have explored the use of GCNs [75] solely, a combination of GCNs with RNN [73] or CNN [74], and a recurrent architecture tailored to graph structures [76]. In the following, we introduce these methods.

5.4.1 GCN Based Graph Spatial-Temporal Networks

Diffusion Convolutional Recurrent Neural Network (DCRNN) [73] introduces diffusion convolution as graph convolution for capturing spatial dependency and uses sequence-to-sequence architecture [96] with gated recurrent units (GRU) [82] to capture temporal dependency.

Diffusion convolution models a truncated diffusion process with the forward and backward directions. Formally, the diffusion convolution is defined as

$$\mathbf{X}_{:,p} \star_G f(\theta) = \sum_{k=0}^{K-1} (\theta_{k1}(\mathbf{D}_O^{-1}\mathbf{A})^k + \theta_{k2}(\mathbf{D}_I^{-1}\mathbf{A}^T)^k)\mathbf{X}_{:,p} \tag{34}$$

where $\mathbf{D}_O$ is the out-degree matrix and $\mathbf{D}_I$ is the in-degree matrix. To allow multiple input and output channels, DCRNN proposes a diffusion convolution layer, defined as

$$\mathbf{Z}_{:,q} = \sigma(\sum_{p=1}^{P} \mathbf{X}_{:,p} \star_G f(\mathbf{\Theta}_{q,p,:,:})) \tag{35}$$

where $\mathbf{X} \in \mathbf{R}^{N\times P}$ and $\mathbf{Z} \in \mathbf{R}^{N\times Q}$, $\mathbf{\Theta} \in \mathbf{R}^{Q\times P\times K\times 2}$, Q is the number of output channels and P is the number of input channels.

To capture temporal dependency, DCRNN processes the inputs of GRU using a diffusion convolution layer so that the recurrent unit simultaneously receives history information from the last time step and neighborhood information from graph convolution. The modified GRU in DCRNN is named as the diffusion convolutional gated recurrent Unit (DCGRU),

$$\begin{aligned}
\mathbf{r}^{(t)} &= sigmoid(\mathbf{\Theta}_r \star_G [\mathbf{X}^{(t)}, \mathbf{H}^{(t-1)}] + \mathbf{b}_r) \\
\mathbf{u}^{(t)} &= sigmoid(\mathbf{\Theta}_u \star_G [\mathbf{X}^{(t)}, \mathbf{H}^{(t-1)}] + \mathbf{b_u}) \\
\mathbf{C}^{(t)} &= tanh(\mathbf{\Theta}_C \star_G [\mathbf{X}^{(t)}, (\mathbf{r}^{(t)} \odot \mathbf{H}^{(t-1)})] + \mathbf{b}_r) \\
\mathbf{H}^{(t)} &= \mathbf{u}^{(t)} \odot \mathbf{H}^{(t-1)} + (1 - \mathbf{u}^{(t)}) \odot \mathbf{C}^{(t)}
\end{aligned} \tag{36}$$

To meet the demands of multi-step forecasting, DCRNN adopts sequence-to-sequence architecture [96] where the recurrent unit is replaced by DCGRU.

CNN-GCN [74] interleaves 1D-CNN with GCN [14] to learn spatial-temporal graph data. For an input tensor $\mathbf{X} \in R^{T\times N\times D}$, the 1D-CNN layer slides over $\mathbf{X}_{[:,i,:]}$ along the time axis to aggregate temporal information for each node while the GCN layer operates on $\mathbf{X}_{[i,:,:]}$ to aggregate spatial information at each time step. The output layer is a linear transformation, generating a prediction for each node. The framework of CNN-GCN is depicted in Fig 5c.

Spatial Temporal GCN (ST-GCN) [75] adopts a different approach by extending the temporal flow as graph edges so that spatial and temporal information can be extracted using a unified GCN model at the same time. ST-GCN defines a labelling function to assign a label to each edge of the graph according to the distance of the two related nodes. In this way, the adjacency matrix can be represented as a summation of K adjacency matrices where K is the number of labels. Then ST-GCN applies GCN [14] with a different weight matrix to each of the K adjacency matrix and sums them.

$$\mathbf{f}_{out} = \sum_j \mathbf{\Lambda}_j^{-\frac{1}{2}} \mathbf{A_j} \mathbf{\Lambda}_j^{-\frac{1}{2}} \mathbf{f}_{in} \mathbf{W}_j \tag{37}$$

5.4.2 Miscellaneous Variants

Structural-RNN [76] proposes a recurrent structured framework named Structural-RNN. The aim of Structural-RNN is to predict node labels at each time step. In Structural-RNN, it comprises of two kinds of RNNs, namely nodeRNN and edgeRNN. The temporal information of each node and each edge is passed through a nodeRNN and an edgeRNN respectively. Since assuming different RNNs for different nodes and edges increases model complexity dramatically, they instead split nodes and edges into semantic groups. For example, a human-object interaction graph consists of two groups of nodes, human nodes and object nodes, and three groups of edges, human-human edges, object-object edges, and human-object edges. Nodes or edges in a same semantic group share the same RNN model. To incorporate the spatial information, a nodeRNN will take the outputs of edgeRNN as inputs.

5.4.3 Summary

The advantage of DCRNN is that it is able to handle long-term dependencies because of the recurrent network architectures. Though simpler than DCRNN, CNN-GCN processes spatial-temporal graphs more efficiently owing to the fast implementation of 1D CNN. ST-GCN considers temporal flow as graph edges, resulting in the size of the adjacency matrix growing quadratically. On the one hand, it increases the computation cost of the graph convolution layer. On the other hand, to capture the long-term dependency, the graph convolution layer has to be stacked many times. Structural-RNN improves model efficiency by sharing the same RNN within the same semantic group. However, Structural-RNN demands human prior knowledge to split the semantic groups.

6 Applications

Graph neural networks have a wide variety of applications. In this section, we first summarize the benchmark datasets frequently used in the literature. Then we report the benchmark performance on four commonly used datasets and list the available open source implementations of graph neural networks. Finally, we provide practical applications of graph neural networks in various domains.

6.1 Datasets

In our survey, we count the frequency of each dataset which occurs in the papers reviewed in this work, and report in Table 5 the datasets which occur at least twice.

Citation Networks consist of papers, authors and their relationship such as citation, authorship, 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 datasets for paper-citation networks, Cora, Citeseer and Pubmed. The Cora dataset contains 2708 machine learning publications grouped into seven classes. The Citeseer dataset 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 dataset 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 dataset with millions of papers and authors which have been collected from computer science bibliographies. The raw dataset 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.

Social Networks are formed by user interactions from online services such as BlogCatalog, Reddit, and Epinions. The BlogCatalog dataset is a social network which consists of bloggers and their social relationships. The labels of bloggers represent their personal interests. The Reddit dataset 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. The Epinions dataset is a multi-relation graph collected from an online product review website where commenters can have more than one type of relation, such as trust, distrust, co-review, and co-rating.

Chemical/Biological 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 dataset 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 dataset contains 188 nitro compounds, labeled as to whether they are aromatic or heteroaromatic. The D&D dataset contains 1178 protein structures, labeled as to whether they are enzymes or non-enzymes. The QM9 dataset contains 133885 molecules labeled with 13 chemical properties. The Tox21 dataset contains 12707 chemical compounds labeled with 12 types of toxicity. Another important dataset 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.

Unstructured Graphs To test the generalization of graph neural networks to unstructured data, the k nearest neighbor graph (k-NN graph) has been widely used. The MNIST dataset contains 70000 images of size 28×28 labeled with ten digits. A typical way to convert an MNIST image to a graph is to construct an 8-NN graph based on its pixel locations. The Wikipedia dataset is a word co-occurrence network extracted from the first million bytes of the Wikipedia dump. Labels of words represent part-of-speech (POS) tags. The 20-NewsGroup dataset consists of around 20,000 News Group (NG) text documents categorized by 20 news types. The graph of the 20-NewsGroup is constructed by representing each document as a node and using the similarities between nodes as edge weights.

Others There are several other datasets worth mentioning. The METR-LA is a traffic dataset collected from the highways of Los Angeles County. The MovieLens-1M dataset from the MovieLens website contains 1 million item ratings given by 6k users. It is a benchmark dataset for recommender systems. The NELL dataset 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.

6.2 Benchmarks & Open-source Implementations

Of the datasets listed in Table 5, Cora, Pubmed, Citeseer, and PPI are the most frequently used datasets. They are often tested to compare the performance of graph convolution networks in node classification tasks. In Table 6, we report the benchmark performance of these four datasets, all of which use standard data splits. Open-source implementations facilitate the work of baseline experiments in deep learning research. Due to the vast number of hyper-parameters, it is difficult to achieve the same results as reported in the literature without using published codes. In Table 7, we provide the hyperlinks of open-source implementations of the graph neural network models reviewed in Section 4-5. Noticeably, Fey et al. [89] published a geometric learning library in PyTorch named PyTorch Geometric [3], which implements several graph neural networks including ChebNet [12], 1stChebNet [14], GraphSage [25], MPNNs [13], GAT [15] and SplineCNN [89]. Most recently, the Deep Graph Library (DGL) [4] is released which provides a fast implementation of many graph neural networks with a set of functions on top of popular deep learning platforms such as PyTorch and MXNet.

6.3 Practical Applications

Graph neural networks have a wide range of applications across different tasks and domains. Despite general tasks at which each category of GNNs is specialized, including node classification, node representation learning, graph classification, graph generation, and spatial-temporal forecasting, GNNs can also be applied to node clustering, link prediction [124], and graph partition [125]. In this section, we mainly introduce practical applications according to general domains to which they belong.

Computer Vision One of the biggest application areas for graph neural networks is computer vision. Researchers have explored leveraging graph structures in scene graph generation, point clouds classification and segmentation, action recognition and many other directions.

In scene graph generation, semantic relationships between objects facilitate the understanding of the semantic meaning behind a visual scene. Given an image, scene

3. https://github.com/rusty1s/pytorch_geometric
4. https://www.dgl.ai/

TABLE 5: Summary of Commonly Used Datasets

Category	Dataset	Source	# Graphs	# Nodes	# Edges	#Features	# Labels	Citation
Citation Networks	Cora	[107]	1	2708	5429	1433	7	[14], [15], [24], [26], [28] [47], [48], [49], [52], [57] [61], [62], [63], [64], [108] [109], [110]
	Citeseer	[107]	1	3327	4732	3703	6	[14], [15], [28], [49], [52] [57], [61], [62], [63], [64] [109], [110]
	Pubmed	[107]	1	19717	44338	500	3	[14], [15], [26], [28], [47] [48], [51], [52], [54], [57] [62], [64], [70], [109], [110]
	DBLP	dblp.uni-trier.de [111](aminer.org /citation)	1	-	-	-	-	[65], [70], [108], [112]
Social Networks	BlogCatalog	[113]	1	10312	333983	-	39	[43], [51], [65], [114]
	Reddit	[25]	1	232965	11606919	602	41	[25], [29], [48], [49], [57] [109]
	Epinions	www.epinions.com	1	-	-	-	-	[53], [112]
Chemical/ Biological Graphs	PPI	[115]	24	56944	818716	50	121	[15], [20], [25], [28], [29] [49], [51], [65], [109]
	NCI-1	[116]	4110	-	-	37	2	[27], [47], [50], [55], [58] [60], [90], [92], [93]
	NCI-109	[116]	4127	-	-	38	2	[27], [47], [55], [93]
	MUTAG	[117]	188	-	-	7	2	[27], [47], [55], [58], [90] [92]
	D&D	[118]	1178	-	-	-	2	[27], [50], [55], [58], [59] [90], [93]
	QM9	[119]	133885	-	-	-	13	[13], [69]
	tox21	tripod.nih.gov/ tox21/challenge/	12707	-	-	-	12	[23], [56]
Unstruct-ured Graphs	MNIST	yann.lecun.com /exdb/mnist/	70000	-	-	-	10	[12], [21], [24], [26], [55]
	Wikipedia	www.mattmahoney .net/dc/textdata	1	4777	184812	-	40	[65], [114]
	20NEWS	[120]	1	18846	-	-	20	[12], [42]
Others	METR-LA	[121]	-	-	-	-	-	[29], [73]
	Movie-Lens1M	[122] grouplens.org/ datasets/ movielens/1m/	1	10000	1 Million	-	-	[24], [114]
	Nell	[123]	1	65755	266144	61278	210	[14], [49], [52]

TABLE 6: Benchmark performance of four most frequently used datasets. The listed methods use the same training, validation, and test data for evaluation.

Method	Cora	Citeseer	Pubmed	PPI
1stChebnet (2016) [14]	81.5	70.3	79.0	-
GraphSage (2017) [25]	-	-	-	61.2
GAT (2017) [15]	83.0±0.7	72.5±0.7	79.0±0.3	97.3±0.2
Cayleynets (2017) [24]	81.9±0.7	-	-	-
StoGCN (2018) [49]	82.0±0.8	70.9±0.2	78.7±0.3	97.8±0.05
DualGCN (2018) [52]	**83.5**	72.6	**80.0**	-
GAAN (2018) [29]	-	-	-	**98.71±0.02**
GraphInfoMax (2018) [109]	82.3±0.6	71.8±0.7	76.8±0.6	63.8±0.2
GeniePath (2018) [51]	-	-	78.5	97.9
LGCN (2018) [28]	83.3±0.5	**73.0±0.6**	79.5±0.2	77.2±0.2
SSE (2018) [20]	-	-	-	83.6

graph generation models detect and recognize objects and predict semantic relationships between pairs of objects [126], [127], [128]. Another application inverses the process by generating realistic images given scene graphs [129]. 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.

In point clouds classification and segmentation, a point cloud is a set of 3D points recorded by LiDAR scans. Solutions for this task enable LiDAR devices to see the surrounding environment, which is typically beneficial for unmanned vehicles. To identify objects depicted by point clouds, [130], [131], [132] convert point clouds into k-nearest neighbor graphs or superpoint graphs, and use graph convolution networks to explore the topological structure.

In action recognition, recognizing human actions contained in videos facilitates a better understanding of video content from a machine aspect. One group of solutions detects 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, [75], [76] applies spatial-temporal neural networks to learn human action patterns.

In addition, the number of possible directions in which to apply graph neural networks in computer vision is still growing. This includes human-object interaction [133], few-shot image classification [134], [135], semantic segmentation [136], [137], visual reasoning [138] and question answering [139].

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,

TABLE 7: A Summary of Open-source Implementations

Model	Framework	Github Link
ChebNet (2016) [12]	tensorflow	https://github.com/mdeff/cnn_graph
1stChebNet (2017) [14]	tensorflow	https://github.com/tkipf/gcn
GGNNs (2015) [19]	lua	https://github.com/yujiali/ggnn
SSE (2018) [20]	c	https://github.com/Hanjun-Dai/steady_state_embedding
GraphSage (2017) [25]	tensorflow	https://github.com/williamleif/GraphSAGE
LGCN (2018) [28]	tensorflow	https://github.com/divelab/lgcn/
SplineCNN (2018) [89]	pytorch	https://github.com/rusty1s/pytorch_geometric
GAT (2017) [15]	tensorflow	https://github.com/PetarV-/GAT
GAE (2016) [62]	tensorflow	https://github.com/limaosen0/Variational-Graph-Auto-Encoders
ARGA (2018) [64]	tensorflow	https://github.com/Ruiqi-Hu/ARGA
DNGR (2016) [42]	matlab	https://github.com/ShelsonCao/DNGR
SDNE (2016) [43]	python	https://github.com/suanrong/SDNE
DRNE (2016) [66]	tensorflow	https://github.com/tadpole/DRNE
GraphRNN (2018) [67]	tensorflow	https://github.com/snap-stanford/GraphRNN
DCRNN (2018) [73]	tensorflow	https://github.com/liyaguang/DCRNN
CNN-GCN (2017) [74]	tensorflow	https://github.com/VeritasYin/STGCN_IJCAI-18
ST-GCN (2018) [75]	pytorch	https://github.com/yysijie/st-gcn
Structural RNN (2016) [76]	theano	https://github.com/asheshjain399/RNNexp

it can be cast as a link prediction problem. The goal is to predict the missing links between users and items. To address this problem, Van et al. [9] and Ying et al. [11] et al. propose a GCN-based graph auto-encoder. Monti et al. [10] combine GCN and RNN to learn the underlying process that generates the known ratings.

Traffic Traffic congestion has become a hot social issue in modern cities. Accurately forecasting traffic speed, volume or the density of roads in traffic networks is fundamentally important in route planning and flow control. [29], [73], [74], [140] adopt a graph-based approach with spatial-temporal neural networks. The input to their models is a spatial-temporal graph. In this spatial-temporal graph, nodes are represented by sensors placed on roads, edges are represented by the distance of pair-wise nodes above a threshold and each node contains a time series as features. The goal is to forecast the average speed of a road within a time interval. Another interesting application is the taxi-demand prediction. This greatly helps intelligent transportation systems make use of resources and save energy effectively. Given historical taxi demands, location information, weather data, and event features, Yao et al. [141] incorporate LSTM, CNN and node embeddings trained by LINE [142] to form a joint representation for each location to predict the number of taxis demanded for a location within a time interval.

Chemistry In chemistry, researchers apply graph neural networks to study the graph structure of molecules. In a molecular graph, atoms function as nodes and chemical bonds function as edges. Node classification, graph classification and graph generation are three main tasks targeting at molecular graphs in order to learn molecular fingerprints [56], [83], to predict molecular properties [13], to infer protein interfaces [143], and to synthesize chemical compounds [68], [69], [144].

Others There have been initial explorations into applying GNNs to other problems such as program verification [19], program reasoning [145], social influence prediction [146], adversarial attacks prevention [147], electrical health records modeling [148], [149], brain networks [150], event detection [151] and combinatorial optimization [152].

7 Future Directions

Though graph neural networks have proven their power in learning graph data, challenges still exist due to the complexity of graphs. In this section, we provide four future directions of graph neural networks.

Go Deep The success of deep learning lies in deep neural architectures. In image classification, for example, an outstanding model named ResNet [153] has 152 layers. However, when it comes to graphs, experimental studies have shown that with the increase in the number of layers, the model performance drops dramatically [110]. According to [110], this is due to the effect of graph convolutions in that it essentially pushes representations of adjacent nodes closer to each other so that, in theory, with an infinite times of convolutions, all nodes' representations will converge to a single point. This raises the question of whether going deep is still a good strategy for learning graph-structured data.

Receptive Field The receptive field of a node refers to a set of nodes including the central node and its neighbors. The number of neighbors of a node follows a power law distribution. Some nodes may only have one neighbor, while other nodes may neighbors as many as thousands. Though sampling strategies have been adopted [25], [27], [28], how to select a representative receptive field of a node remains to be explored.

Scalability Most graph neural networks do not scale well for large graphs. The main reason for this is when stacking multiple layers of a graph convolution, a node's final state involves a large number of its neighbors' hidden states, leading to the high complexity of backpropagation. While several approaches try to improve their model efficiency by fast sampling [48], [49] and sub-graph training [25], [28], they are still not scalable enough to handle deep architectures with large graphs.

Dynamics and Heterogeneity The majority of current graph neural networks tackle with static homogeneous graphs. On the one hand, graph structures are assumed to be fixed. On the other hand, nodes and edges from a graph are assumed to come from a single source. However, these two assumptions are not realistic in many scenarios. In a social

network, a new person may enter into a network at any time and an existing person may quit the network as well. In a recommender system, products may have different types where their inputs may have different forms such as texts or images. Therefore, new methods should be developed to handle dynamic and heterogeneous graph structures.

8 Conclusion

In this survey, we conduct a comprehensive overview of graph neural networks. We provide a taxonomy which groups graph neural networks into five categories: graph convolutional networks, graph attention networks, graph autoencoders, graph generative networks and graph spatial-temporal networks. We provide a thorough review, comparisons, and summarizations of the methods within or between categories. Then we introduce a wide range of applications of graph neural networks. Datasets, open source codes, and benchmarks for graph neural networks are summarized. Finally, we suggest four future directions for graph neural networks.

Acknowledgment

This research was funded by the Australian Government through the Australian Research Council (ARC) under grants 1) LP160100630 partnership with Australia Government Department of Health and 2) LP150100671 partnership with Australia Research Alliance for Children and Youth (ARACY) and Global Business College Australia (GBCA). We acknowledge the support of NVIDIA Corporation and MakeMagic Australia with the donation of GPU used for this research.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[3] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1412–1421.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] R. van den Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *stat*, vol. 1050, p. 7, 2017.

[10] F. Monti, M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 3697–3707.

[11] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2018, pp. 974–983.

[12] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.

[13] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the International Conference on Machine Learning*, 2017, pp. 1263–1272.

[14] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the International Conference on Learning Representations*, 2017.

[15] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proceedings of the International Conference on Learning Representations*, 2017.

[16] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings of the International Joint Conference on Neural Networks*, vol. 2. IEEE, 2005, pp. 729–734.

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

[18] 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.

[19] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *Proceedings of the International Conference on Learning Representations*, 2015.

[20] H. Dai, Z. Kozareva, B. Dai, A. Smola, and L. Song, "Learning steady-states of iterative algorithms over graphs," in *Proceedings of the International Conference on Machine Learning*, 2018, pp. 1114–1122.

[21] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *Proceedings of International Conference on Learning Representations*, 2014.

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

[23] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, pp. 3546–3553.

[24] 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.

[25] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

[26] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5115–5124.

[27] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proceedings of the International Conference on Machine Learning*, 2016, pp. 2014–2023.

[28] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 1416–1424.

[29] 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 *Proceedings of the Uncertainty in Artificial Intelligence*, 2018.

[30] 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.

[31] 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.

[32] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *arXiv preprint arXiv:1812.04202*, 2018.

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

[34] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

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

[36] 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.

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

[38] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, "Tri-party deep network representation," in *Proceedings of the International Joint Conference on Artificial Intelligence*. AAAI Press, 2016, pp. 1895–1901.

[39] X. Shen, S. Pan, W. Liu, Y.-S. Ong, and Q.-S. Sun, "Discrete network embedding," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 7 2018, pp. 3549–3555.

[40] H. Yang, S. Pan, P. Zhang, L. Chen, D. Lian, and C. Zhang, "Binarized attributed network embedding," in *IEEE International Conference on Data Mining*. IEEE, 2018.

[41] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 701–710.

[42] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016, pp. 1145–1152.

[43] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 1225–1234.

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

[45] 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.

[46] A. Susnjara, N. Perraudin, D. Kressner, and P. Vandergheynst, "Accelerated filtering on graphs using lanczos method," *arXiv preprint arXiv:1509.04537*, 2015.

[47] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 1993–2001.

[48] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," in *Proceedings of the International Conference on Learning Representations*, 2018.

[49] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *Proceedings of the International Conference on Machine Learning*, 2018, pp. 941–949.

[50] 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.

[51] Z. Liu, C. Chen, L. Li, J. Zhou, X. Li, and L. Song, "Geniepath: Graph neural networks with adaptive receptive paths," *arXiv preprint arXiv:1802.00910*, 2018.

[52] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *Proceedings of the World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2018, pp. 499–508.

[53] T. Derr, Y. Ma, and J. Tang, "Signed graph convolutional network," *arXiv preprint arXiv:1808.06354*, 2018.

[54] T. Pham, T. Tran, D. Q. Phung, and S. Venkatesh, "Column networks for collective classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017, pp. 2485–2491.

[55] M. Simonovsky and N. Komodakis, "Dynamic edgeconditioned filters in convolutional neural networks on graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

[56] 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.

[57] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 4563–4572.

[58] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[59] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in Neural Information Processing Systems*, 2018, pp. 4801–4811.

[60] J. B. Lee, R. Rossi, and X. Kong, "Graph classification using structural attention," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 1666–1674.

[61] S. Abu-El-Haija, B. Perozzi, R. Al-Rfou, and A. A. Alemi, "Watch your step: Learning node embeddings via graph attention," in *Advances in Neural Information Processing Systems*, 2018, pp. 9197–9207.

[62] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[63] C. Wang, S. Pan, G. Long, X. Zhu, and J. Jiang, "Mgae: Marginalized graph autoencoder for graph clustering," in *Proceedings of the ACM on Conference on Information and Knowledge Management*. ACM, 2017, pp. 889–898.

[64] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding." in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2018, pp. 2609–2615.

[65] 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 *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2663–2671.

[66] K. Tu, P. Cui, X. Wang, P. S. Yu, and W. Zhu, "Deep recursive network embedding with regular equivalence," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2018, pp. 2357–2366.

[67] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "Graphrnn: A deep generative model for graphs," *Proceedings of International Conference on Machine Learning*, 2018.

[68] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," in *Proceedings of the International Conference on Machine Learning*, 2018.

[69] N. De Cao and T. Kipf, "Molgan: An implicit generative model for small molecular graphs," *arXiv preprint arXiv:1805.11973*, 2018.

[70] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "Netgan: Generating graphs via random walks," in *Proceedings of the International Conference on Machine Learning*, 2018.

[71] T. Ma, J. Chen, and C. Xiao, "Constrained generation of semantically valid graphs via regularizing variational autoencoders," in *Advances in Neural Information Processing Systems*, 2018, pp. 7110–7121.

[72] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," *arXiv preprint arXiv:1612.07659*, 2016.

[73] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *Proceedings of International Conference on Learning Representations*, 2018.

[74] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2018, pp. 3634–3640.

[75] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[76] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-rnn: Deep learning on spatio-temporal graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5308–5317.

[77] 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.

[78] 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.

[79] 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.

[80] L. B. Almeida, "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment." in *Proceedings of the International Conference on Neural Networks*, vol. 2. IEEE, 1987, pp. 609–618.

[81] F. J. Pineda, "Generalization of back-propagation to recurrent neural networks," *Physical review letters*, vol. 59, no. 19, p. 2229, 1987.

[82] 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 *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014, pp. 1724–1734.

[83] 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 *Advances in Neural Information Processing Systems*, 2015, pp. 2224–2232.

[84] 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.

[85] 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.

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

[87] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on riemannian manifolds," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2015, pp. 37–45.

[88] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein, "Learning shape correspondence with anisotropic convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 3189–3197.

[89] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller, "Splinecnn: Fast geometric deep learning with continuous b-spline kernels," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 869–877.

[90] D. V. Tran, A. Sperduti *et al.*, "On filter size in graph convolutional networks," in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2018, pp. 1534–1541.

[91] S. Pan, J. Wu, and X. Zhu, "Cogboost: Boosting for fast cost-sensitive graph classification," *IEEE Transactions on Knowledge & Data Engineering*, no. 1, pp. 1–1, 2015.

[92] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks," *arXiv preprint arXiv:1810.00826*, 2018.

[93] S. Verma and Z.-L. Zhang, "Graph capsule convolutional neural networks," *arXiv preprint arXiv:1805.08090*, 2018.

[94] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[95] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[96] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, 2014, pp. 3104–3112.

[97] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the international conference on Machine learning*. ACM, 2008, pp. 1096–1103.

[98] G. L. Guimaraes, B. Sanchez-Lengeling, C. Outeiral, P. L. C. Farias, and A. Aspuru-Guzik, "Objective-reinforced generative adversarial networks (organ) for sequence generation models," *arXiv preprint arXiv:1705.10843*, 2017.

[99] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar variational autoencoder," *arXiv preprint arXiv:1703.01925*, 2017.

[100] H. Dai, Y. Tian, B. Dai, S. Skiena, and L. Song, "Syntax-directed variational autoencoder for molecule generation," in *Proceedings of the International Conference on Learning Representations*, 2018.

[101] 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.

[102] B. Chen, L. Sun, and X. Han, "Sequence-to-action: End-to-end semantic graph generation for semantic parsing," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2018, pp. 766–777.

[103] D. D. Johnson, "Learning graphical state transitions," in *Proceedings of the International Conference on Learning Representations*, 2016.

[104] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*. Springer, 2018, pp. 593–607.

[105] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems*, 2017, pp. 5767–5777.

[106] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[107] 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.

[108] X. Zhang, Y. Li, D. Shen, and L. Carin, "Diffusion maps for textual network embedding," in *Advances in Neural Information Processing Systems*, 2018.

[109] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," *arXiv preprint arXiv:1809.10341*, 2018.

[110] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[111] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: extraction and mining of academic social networks," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2008, pp. 990–998.

[112] Y. Ma, S. Wang, C. C. Aggarwal, D. Yin, and J. Tang, "Multi-dimensional graph convolutional networks," *arXiv preprint arXiv:1808.06099*, 2018.

[113] L. Tang and H. Liu, "Relational learning via latent social dimensions," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Ciscovery and Data Mining*. ACM, 2009, pp. 817–826.

[114] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "Graphgan: Graph representation learning with generative adversarial nets," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[115] M. Zitnik and J. Leskovec, "Predicting multicellular function through multi-layer tissue networks," *Bioinformatics*, vol. 33, no. 14, pp. i190–i198, 2017.

[116] 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.

[117] 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.

[118] 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.

[119] 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.

[120] T. Joachims, "A probabilistic analysis of the rocchio algorithm with tfidf for text categorization." Carnegie-mellon univ pittsburgh pa dept of computer science, Tech. Rep., 1996.

[121] 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.

[122] B. N. Miller, I. Albert, S. K. Lam, J. A. Konstan, and J. Riedl, "Movielens unplugged: experiences with an occasionally connected recommender system," in *Proceedings of the international conference on Intelligent user interfaces*. ACM, 2003, pp. 263–266.

[123] 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 *Proceedings of the AAAI Conference on Artificial Intelligence*, 2010, pp. 1306–1313.
[124] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Advances in Neural Information Processing Systems*, 2018.
[125] T. Kawamoto, M. Tsubaki, and T. Obuchi, "Mean-field theory of graph neural networks in graph partitioning," in *Advances in Neural Information Processing Systems*, 2018, pp. 4362–4372.
[126] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, "Scene graph generation by iterative message passing," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2017.
[127] J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh, "Graph r-cnn for scene graph generation," in *European Conference on Computer Vision*. Springer, 2018, pp. 690–706.
[128] 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 *European Conference on Computer Vision*. Springer, 2018, pp. 346–363.
[129] J. Johnson, A. Gupta, and L. Fei-Fei, "Image generation from scene graphs," *arXiv preprint*, 2018.
[130] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *arXiv preprint arXiv:1801.07829*, 2018.
[131] L. Landrieu and M. Simonovsky, "Large-scale point cloud semantic segmentation with superpoint graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.
[132] G. Te, W. Hu, Z. Guo, and A. Zheng, "Rgcnn: Regularized graph cnn for point cloud segmentation," *arXiv preprint arXiv:1806.02952*, 2018.
[133] S. Qi, W. Wang, B. Jia, J. Shen, and S.-C. Zhu, "Learning human-object interactions by graph parsing neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 401–417.
[134] V. G. Satorras and J. B. Estrach, "Few-shot learning with graph neural networks," in *Proceedings of the International Conference on Learning Representations*, 2018.
[135] 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 *European Conference on Computer Vision*. Springer, 2018, pp. 673–689.
[136] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun, "3d graph neural networks for rgbd semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5199–5208.
[137] L. Yi, H. Su, X. Guo, and L. J. Guibas, "Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation." in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 6584–6592.
[138] X. Chen, L.-J. Li, L. Fei-Fei, and A. Gupta, "Iterative visual reasoning beyond convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.
[139] M. Narasimhan, S. Lazebnik, and A. Schwing, "Out of the box: Reasoning with graph convolution nets for factual visual question answering," in *Advances in Neural Information Processing Systems*, 2018, pp. 2655–2666.
[140] Z. Cui, K. Henrickson, R. Ke, and Y. Wang, "High-order graph convolutional recurrent neural network: a deep learning framework for network-scale traffic learning and forecasting," *arXiv preprint arXiv:1802.07007*, 2018.
[141] 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 *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, pp. 2588–2595.
[142] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.
[143] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein interface prediction using graph convolutional networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 6530–6539.
[144] J. You, B. Liu, R. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *Advances in Neural Information Processing Systems*, 2018.
[145] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," in *Proceedings of the International Conference on Learning Representations*, 2017.
[146] J. Qiu, J. Tang, H. Ma, Y. Dong, K. Wang, and J. Tang, "Deepinf: Social influence prediction with deep learning," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2110–2119.
[147] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2018, pp. 2847–2856.
[148] E. Choi, M. T. Bahadori, L. Song, W. F. Stewart, and J. Sun, "Gram: graph-based attention model for healthcare representation learning," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 787–795.
[149] E. Choi, C. Xiao, W. Stewart, and J. Sun, "Mime: Multilevel medical embedding of electronic health records for predictive healthcare," in *Advances in Neural Information Processing Systems*, 2018, pp. 4548–4558.
[150] 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.
[151] T. H. Nguyen and R. Grishman, "Graph convolutional networks with argument-aware pooling for event detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018, pp. 5900–5907.
[152] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *Advances in Neural Information Processing Systems*, 2018, pp. 536–545.
[153] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.