A Comprehensive Survey on Graph Neural Networks

Zonghan Wu, Shirui Pan, *Member, IEEE*, Fengwen Chen, Guodong Long, Chengqi Zhang, *Senior Member, IEEE*, and Philip S. Yu, *Life 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 the existing machine learning algorithms. Recently, many studies on extending deep learning approaches for graph data have emerged. In this article, 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 GNNs into four categories, namely, recurrent GNNs, convolutional GNNs, graph autoencoders, and spatial–temporal GNNs. We further discuss the applications of GNNs across various domains and summarize the open-source codes, benchmark data sets, and model evaluation of GNNs. Finally, we propose potential research directions in this rapidly growing field.**

***Index Terms*— Deep learning, graph autoencoder (GAE), graph convolutional networks (GCNs), graph neural networks (GNNs), graph representation learning, network embedding.**

I. 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, have recently been revolutionized by various end-to-end deep learning paradigms, e.g., convolutional neural networks (CNNs) [6], recurrent neural networks (RNNs) [7], and autoencoders [8]. The success of deep learning in many domains is partially attributed to the rapidly developing computational resources (e.g., GPU), the availability of big training data, and the effectiveness of deep learning to extract latent representations from the Euclidean data (e.g., images, text, and videos). Taking image data as an example, we can represent an image as a regular grid in the Euclidean space. CNN is able to exploit the shift-invariance, local connectivity, and compositionality of image data [9]. As a result, CNNs can extract local meaningful features that are shared with the entire data sets for various image analyses.

While deep learning effectively captures hidden patterns of Euclidean data, there are an increasing number of applications, where data are represented in the form of graphs. For example, in e-commerce, a graph-based learning system can exploit the interactions between users and products to make highly accurate recommendations. In chemistry, molecules are modeled as graphs, and their bioactivity needs to be identified for drug discovery. In a citation network, articles are linked to each other via citationships, and they need to be categorized into different groups. The complexity of graph data has imposed significant challenges on the existing machine learning algorithms. As graphs can be irregular, a graph may have a variable size of unordered nodes, and nodes from a graph may have a different number of neighbors, resulting in some important operations (e.g., convolutions) being easy to compute in the image domain but difficult to apply to the graph domain. Furthermore, a core assumption of existing machine learning algorithms is that instances are independent of each other. This assumption no longer holds for graph data because each instance (node) is related to others by links of various types, such as citations, friendships, and interactions.

Recently, there is increasing interest in extending deep learning approaches for graph data. Motivated by CNNs, RNNs, and autoencoders from deep learning, new generalizations and definitions of important operations have been rapidly developed over the past few years to handle the complexity of graph data. For example, a graph convolution can be generalized from a 2-D convolution. As illustrated in Fig. 1, an image can be considered as a special case of graphs, where pixels are connected by adjacent pixels. Similar to 2-D convolution, one may perform graph convolutions by taking the weighted average of a node's neighborhood information.

There are a limited number of existing reviews on the topic of graph neural networks (GNNs). Using the term geometric deep learning, Bronstein *et al.* [9] give an overview

Manuscript received January 2, 2019; revised August 7, 2019 and December 3, 2019; accepted March 2, 2020. Date of publication March 24, 2020; date of current version January 5, 2021. This work was supported in part by the Australian Government through the Australian Research Council (ARC) under Grant LP160100630 and Grant LP180100654 and in part by the NSF under Grant III-1526499, Grant III-1763325, Grant III-1909323, and Grant CNS-1930941. *(Corresponding author: Shirui Pan.)*

Zonghan Wu, Fengwen Chen, Guodong Long, and Chengqi Zhang are with the Centre for Artificial Intelligence, Faculty of Engineering and Information Technology, University of Technology Sydney, Ultimo, 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).

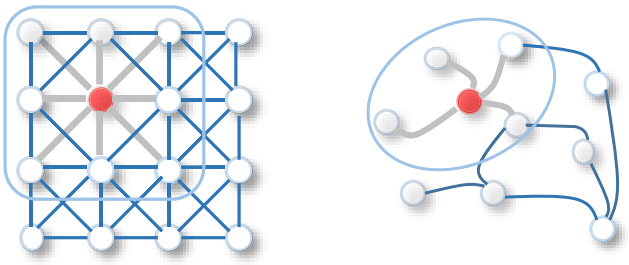
Shirui Pan is with the Faculty of Information Technology, Monash University, Clayton, VIC 3800, Australia (e-mail: shirui.pan@monash.edu).

Philip S. Yu is with the Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7053 USA (e-mail: psyu@cs.uic.edu).

This article has supplementary downloadable material available at https://ieeexplore.ieee.org, provided by the authors.

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2020.2978386



(a) (b)

Fig. 1. 2-D convolution versus graph convolution. (a) 2-D convolution: analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2-D convolution takes the weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size. (b) Graph convolution: to get a hidden representation of the red node, one simple solution of the graph convolutional operation is to take the average value of the node features of the red node along with its neighbors. Different from the image data, the neighbors of a node are unordered and variable in size.

of deep learning methods in the non-Euclidean domain, including graphs and manifolds. Although it is the first review on GNNs, this article mainly reviews convolutional GNNs. Hamilton *et al.* [10] cover a limited number of GNNs with a focus on addressing the problem of network embedding. Battaglia *et al.* [11] position graph networks as the building blocks for learning from relational data, reviewing part of GNNs under a unified framework. Lee *et al.* [12] conduct a partial survey of GNNs that apply different attention mechanisms. In summary, existing surveys only include some of the GNNs and examine a limited number of works, thereby missing the most recent development of GNNs. This article provides a comprehensive overview of GNNs, for both interested researchers who want to enter this rapidly developing field and experts who would like to compare GNN models. To cover a broader range of methods, this article considers GNNs as all deep learning approaches for graph data.

A. Our Contributions

This article makes notable contributions summarized as follows.

1) *New Taxonomy:* We propose a new taxonomy of GNNs. GNNs are categorized into four groups: recurrent GNNs (RecGNN), convolutional GNNs (ConvGNNs), graph autoencoders (GAEs), and spatial–temporal GNNs (STGNNs).
2) *Comprehensive Review:* We provide the most comprehensive overview of modern deep learning techniques for graph data. For each type of GNNs, we provide detailed descriptions of representative models, make the necessary comparison, and summarize the corresponding algorithms.
3) *Abundant Resources:* We collect abundant resources on GNNs, including state-of-the-art models, benchmark data sets, open-source codes, and practical applications. This article can be used as a hands-on guide for understanding, using, and developing different deep learning approaches for various real-life applications.
4) *Future Directions:* We discuss theoretical aspects of GNNs, analyze the limitations of existing methods, and suggest four possible future research directions in terms of model depth, scalability tradeoff, heterogeneity, and dynamicity.

B. Organization of This Article

The rest of this article is organized as follows. Section II outlines the background of GNNs, lists commonly used notations, and defines graph-related concepts. Section III clarifies the categorization of GNNs. Sections IV–VII provides an overview of GNN models. Section VIII presents a collection of applications across various domains. Section IX discusses the current challenges and suggests future directions. Section X summarizes this article.

II. Background and Definition

In this section, we outline the background of GNNs, list commonly used notations, and define graph-related concepts.

A. Background

1) Brief History of Graph Neural Networks: Sperduti and Starita [13] first applied neural networks to directed acyclic graphs, which motivated early studies on GNNs. The notion of GNNs was initially outlined in [14] and further elaborated in [15] and [16]. These early studies fall into the category of RecGNNs. They learn a target node's representation by propagating neighbor information in an iterative manner until a stable fixed point is reached. This process is computationally expensive, and recently, there have been increasing efforts to overcome these challenges [17], [18].

Encouraged by the success of CNNs in the computer vision domain, a large number of methods that redefine the notion of convolution for graph data are developed in parallel. These approaches are under the umbrella of ConvGNNs. ConvGNNs are divided into two main streams: the spectral-based approaches and the spatial-based approaches. The first prominent research on spectral-based ConvGNNs was presented by Bruna *et al.* [19], which developed a graph convolution based on the spectral graph theory. Since then, there have been increasing improvements, extensions, and approximations on spectral-based ConvGNNs [20]–[23]. The research about spatial-based ConvGNNs started much earlier than spectral-based ConvGNNs. In 2009, Micheli [24] first addressed graph mutual dependence by architecturally composite nonrecursive layers while inheriting ideas of message passing from RecGNNs. However, the importance of this article was overlooked. Until recently, many spatial-based ConvGNNs (e.g., [25]–[27]) emerged. Apart from RecGNNs and ConvGNNs, many alternative GNNs have been developed in the past few years, including GAEs and STGNNs. These learning frameworks can be built on RecGNNs, ConvGNNs, or other neural architectures for graph modeling. Details on the categorization of these methods are given in Section III.

2) Graph Neural Networks Versus Network Embedding: The research on GNNs is closely related to graph embedding or network embedding, another topic which attracts increasing attention from both the data mining and machine learning communities [10], [28]–[32]. Network embedding aims at representing network nodes as low-dimensional vector representations, preserving both network topology structure and node

content information, so that any subsequent graph analytics task, such as classification, clustering, and recommendation, can be easily performed using simple off-the-shelf machine learning algorithms (e.g., support vector machines for classification). Meanwhile, GNNs are deep learning models aiming at addressing graph-related tasks in an end-to-end manner. Many GNNs explicitly extract high-level representations. The main distinction between GNNs and network embedding is that GNNs are a group of neural network models that are designed for various tasks, while network embedding covers various kinds of methods targeting the same task. Therefore, GNNs can address the network embedding problem through a GAE framework. On the other hand, network embedding contains other nondeep learning methods, such as matrix factorization [33], [34] and random walks [35].

3) Graph Neural Networks Versus Graph Kernel Methods: Graph kernels are historically dominant techniques to solve the problem of graph classification [36]–[38]. These methods employ a kernel function to measure the similarity between pairs of graphs so that kernel-based algorithms, such as support vector machines, can be used for supervised learning on graphs. Similar to GNNs, graph kernels can embed graphs or nodes into vector spaces by a mapping function. The difference is that this mapping function is deterministic rather than learnable. Due to a pairwise similarity calculation, graph kernel methods suffer significantly from computational bottlenecks. GNNs, on the one hand, directly perform graph classification based on the extracted graph representations and, therefore, are much more efficient than graph kernel methods. For a further review of graph kernel methods, we refer the readers to [39].

B. Definition

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

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

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

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

TABLE I
COMMONLY USED NOTATIONS

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

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

III. CATEGORIZATION AND FRAMEWORKS

In this section, we present our taxonomy of GNNs, as shown in Table II. We categorize GNNs into RecGNNs, ConvGNNs, GAEs, and STGNNs. Fig. 2 shows the examples of various model architectures. In the following, we give a brief introduction to each category.

A. Taxonomy of Graph Neural Networks

1) Recurrent Graph Neural Networks: These are mostly pioneer works of GNNs. RecGNNs aim to learn node representations with recurrent neural architectures. They assume a node in a graph constantly exchanges information/message with its neighbors until a stable equilibrium is reached. RecGNNs are conceptually important and inspired later research on ConvGNNs. In particular, the idea of message passing is inherited by spatial-based ConvGNNs.

2) Convolutional Graph Neural Networks: These generalize the operation of convolution from grid data to graph data. The main idea is to generate a node v's representation by aggregating its own features $\mathbf{x}_v$ and neighbors' features $\mathbf{x}_u$, where $u \in N(v)$. Different from RecGNNs, ConvGNNs stack multiple graph convolutional layers to extract high-level node representations. ConvGNNs play a central role in building up many other complex GNN models. Fig. 2(a) shows a

TABLE II
TAXONOMY AND REPRESENTATIVE PUBLICATIONS OF GNNs

Category		Publications
Recurrent Graph Neural Networks (RecGNNs)		[15], [16], [17], [18]
Convolutional Graph Neural Networks (ConvGNNs)	Spectral methods	[19], [20], [21], [22], [23], [40], [41]
	Spatial methods	[24], [25], [26], [27], [42], [43], [44] [45], [46], [47], [48], [49], [50], [51] [52], [53], [54], [55], [56], [57], [58]
Graph Autoencoders (GAEs)	Network Embedding	[59], [60], [61], [62], [63], [64]
	Graph Generation	[65], [66], [67], [68], [69], [70]
Spatial-temporal Graph Neural Networks (STGNNs)		[71], [72], [73], [74], [75], [76], [77]

ConvGNN for node classification. Fig. 2(b) demonstrates a ConvGNN for graph classification.

3) Graph Autoencoders: These are unsupervised learning frameworks that encode nodes/graphs into a latent vector space and reconstruct graph data from the encoded information. GAEs are used to learn network embeddings and graph generative distributions. For network embedding, GAEs learn latent node representations through reconstructing graph structural information, such as the graph adjacency matrix. For graph generation, some methods generate nodes and edges of a graph step by step, while other methods output a graph all at once. Fig. 2(c) presents a GAE for network embedding.

4) Spatial–Temporal Graph Neural Networks: These aim to learn hidden patterns from spatial–temporal graphs, which becomes increasingly important in a variety of applications, such as traffic speed forecasting [72], driver maneuver anticipation [73], and human action recognition [75]. The key idea of STGNNs is to consider spatial dependence and temporal dependence at the same time. Many current approaches integrate graph convolutions to capture spatial dependence with RNNs or CNNs to model temporal dependence. Fig. 2(d) illustrates an STGNN for spatial–temporal graph forecasting.

B. Frameworks

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

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

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

1) *Semisupervised Learning for Node-Level Classification:* Given a single network with partial nodes being labeled and others remaining unlabeled, ConvGNNs can learn a robust model that effectively identifies the class labels for the unlabeled nodes [22]. 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 multiclass classification.
2) *Supervised Learning for Graph-Level Classification:* Graph-level classification aims to predict the class label(s) for an entire graph [52], [54], [78], [79]. The end-to-end learning for this task can be realized with a combination of graph convolutional layers, graph pooling layers, and/or readout layers. While graph convolutional layers are responsible for exacting high-level node representations, graph pooling layers play the role of downsampling, which coarsens each graph into a substructure each time. A readout layer collapses node representations of each graph into a graph representation. By applying a multilayer perceptron and a softmax layer to graph representations, we can build an end-to-end framework for graph classification. An example is given in Fig. 2(b).
3) *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 [61], [62]. Another popular way is to utilize the negative sampling approach that samples a portion of node pairs as negative pairs, while existing node pairs with links in the graphs are positive pairs. Then, a logistic regression layer is applied to distinguish between the positive and negative pairs [42].

In Table III, we summarize the main characteristics of representative RecGNNs and ConvGNNs. Input sources, pooling layers, readout layers, and time complexity are compared among various models. In more detail, we only compare the time complexity of the message-passing/graph convolutional

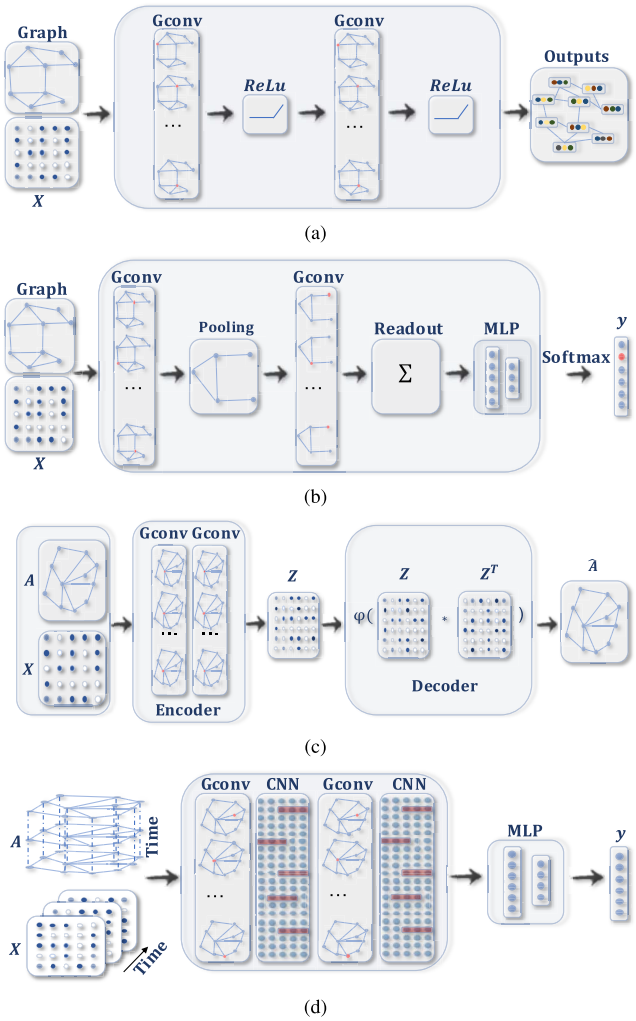


Fig. 2. Different GNN models built with graph convolutional layers. The term Gconv denotes a graph convolutional layer. The term MLP denotes a multilayer perceptron. The term CNN denotes a standard convolutional layer. (a) ConvGNN with multiple graph convolutional layers. A graph convolutional layer encapsulates each node's hidden representation by aggregating feature information from its neighbors. After feature aggregation, a nonlinear transformation is applied to the resulted outputs. By stacking multiple layers, the final hidden representation of each node receives messages from a further neighborhood. (b) ConvGNN with pooling and readout layers for graph classification [21]. A graph convolutional layer is followed by a pooling layer to coarsen a graph into subgraphs so that node representations on coarsened graphs represent higher graph-level representations. A readout layer summarizes the final graph representation by taking the sum/mean of hidden representations of subgraphs. (c) GAE for network embedding [61]. The encoder uses graph convolutional layers to get a network embedding for each node. The decoder computes the pairwise distance given network embeddings. After applying a nonlinear activation function, the decoder reconstructs the graph adjacency matrix. The network is trained by minimizing the discrepancy between the real adjacency matrix and the reconstructed adjacency matrix. (d) STGNN for spatial–temporal graph forecasting [74]. A graph convolutional layer is followed by a 1-D-CNN layer. The graph convolutional layer operates on A and $X^{(t)}$ to capture the spatial dependence, while the 1-D-CNN layer slides over X along the time axis to capture the temporal dependence. The output layer is a linear transformation, generating a prediction for each node, such as its future value at the next time step.

operation in each model. As methods in [19] and [20] require eigenvalue decomposition, the time complexity is $O(n^3)$. The time complexity of [46] is also $O(n^3)$ due to the node pairwise shortest-path computation. Other methods incur equivalent time complexity, which is $O(m)$ if the graph adjacency matrix is sparse and is $O(n^2)$ otherwise. This is because, in these methods, the computation of each node v_i's representation involves its d_i neighbors, and the sum of d_i over all nodes exactly equals the number of edges. The time complexity of several methods is missing in Table III. These methods either lack a time complexity analysis in their articles or report the time complexity of their overall models or algorithms.

IV. Recurrent Graph Neural Networks

RecGNNs are mostly pioneer works of GNNs. They apply the same set of parameters recurrently over nodes in a graph to extract high-level node representations. Constrained by computational power, earlier research is mainly focused on directed acyclic graphs [13], [80].

GNN*[2] proposed by Scarselli *et al.* extends prior recurrent models to handle general types of graphs, e.g., acyclic, cyclic, directed, and undirected graphs [15]. Based on an information diffusion mechanism, GNN* updates nodes' states by exchanging neighborhood information recurrently until a stable equilibrium is reached. A node's hidden state is recurrently updated by

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

where $f(\cdot)$ is a parametric function and $\mathbf{h}_v^{(0)}$ is initialized randomly. The sum operation enables GNN* to be applicable to all nodes, even if the number of neighbors differs and no neighborhood ordering is known. To ensure convergence, the recurrent function $f(\cdot)$ must be a contraction mapping, which shrinks the distance between two points after projecting them into a latent space. In the case of $f(\cdot)$ being a neural network, a penalty term has to be imposed on the Jacobian matrix of parameters. When a convergence criterion is satisfied, the last step node hidden states are forwarded to a readout layer. GNN* alternates the stage of node state propagation and the stage of parameter gradient computation to minimize a training objective. This strategy enables GNN* to handle cyclic graphs. In the follow-up works, the graph echo state network (GraphESN) [16] extends echo state networks to improve the training efficiency of GNN*. GraphESN consists of an encoder and an output layer. The encoder is randomly initialized and requires no training. It implements a contractive state transition function to recurrently update node states until the global graph state reaches convergence. Afterward, the output layer is trained by taking the fixed node states as inputs.

Gated GNN (GGNN) [17] employs a gated recurrent unit (GRU) [81] as a recurrent function, reducing the recurrence to a fixed number of steps. The advantage is that it no longer needs to constrain parameters to ensure convergence. A node hidden state is updated by its previous hidden states and its neighboring hidden states, defined as

$$\mathbf{h}_v^{(t)} = \mathrm{GRU}\left(\mathbf{h}_v^{(t-1)}, \sum_{u \in N(v)} \mathbf{W}\mathbf{h}_u^{(t-1)}\right) \tag{2}$$

where $\mathbf{h}_v^{(0)} = \mathbf{x}_v$. Different from GNN* and GraphESN, GGNN uses the backpropagation through time (BPTT) algorithm to learn the model parameters. This can be problematic

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

TABLE III
SUMMARY OF RecGNNs AND ConvGNNs. MISSING VALUES ("-") IN POOLING AND READOUT LAYERS INDICATE THAT THE METHOD ONLY EXPERIMENTS ON NODE-/EDGE-LEVEL TASKS

Approach	Category	Inputs	Pooling	Readout	Time Complexity
GNN* (2009) [15]	RecGNN	A, X, X^e	-	a dummy super node	$O(m)$
GraphESN (2010) [16]	RecGNN	A, X	-	mean	$O(m)$
GGNN (2015) [17]	RecGNN	A, X	-	attention sum	$O(m)$
SSE (2018) [18]	RecGNN	A, X	-	-	-
Spectral CNN (2014) [19]	Spectral-based ConvGNN	A, X	spectral clustering+max pooling	max	$O(n^3)$
Henaff et al. (2015) [20]	Spectral-based ConvGNN	A, X	spectral clustering+max pooling		$O(n^3)$
ChebNet (2016) [21]	Spectral-based ConvGNN	A, X	efficient pooling	sum	$O(m)$
GCN (2017) [22]	Spectral-based ConvGNN	A, X	-	-	$O(m)$
CayleyNet (2017) [23]	Spectral-based ConvGNN	A, X	mean/graclus pooling	-	$O(m)$
AGCN (2018) [40]	Spectral-based ConvGNN	A, X	max pooling	sum	$O(n^2)$
DualGCN (2018) [41]	Spectral-based ConvGNN	A, X	-	-	$O(m)$
NN4G (2009) [24]	Spatial-based ConvGNN	A, X	-	sum/mean	$O(m)$
DCNN (2016) [25]	Spatial-based ConvGNN	A, X	-	mean	$O(n^2)$
PATCHY-SAN (2016) [26]	Spatial-based ConvGNN	A, X, X^e	-	sum	-
MPNN (2017) [27]	Spatial-based ConvGNN	A, X, X^e	-	attention sum/set2set	$O(m)$
GraphSage (2017) [42]	Spatial-based ConvGNN	A, X	-	-	-
GAT (2017) [43]	Spatial-based ConvGNN	A, X	-	-	$O(m)$
MoNet (2017) [44]	Spatial-based ConvGNN	A, X	-	-	$O(m)$
LGCN (2018) [45]	Spatial-based ConvGNN	A, X	-	-	-
PGC-DGCNN (2018) [46]	Spatial-based ConvGNN	A, X	sort pooling	attention sum	$O(n^3)$
CGMM (2018) [47]	Spatial-based ConvGNN	A, X, X^e	-	sum	-
GAAN (2018) [48]	Spatial-based ConvGNN	A, X	-	-	$O(m)$
FastGCN (2018) [49]	Spatial-based ConvGNN	A, X	-	-	-
StoGCN (2018) [50]	Spatial-based ConvGNN	A, X	-	-	-
Huang et al. (2018) [51]	Spatial-based ConvGNN	A, X	-	-	-
DGCNN (2018) [52]	Spatial-based ConvGNN	A, X	sort pooling	-	$O(m)$
DiffPool (2018) [54]	Spatial-based ConvGNN	A, X	differential pooling	mean	$O(n^2)$
GeniePath (2019) [55]	Spatial-based ConvGNN	A, X	-	-	$O(m)$
DGI (2019) [56]	Spatial-based ConvGNN	A, X	-	-	$O(m)$
GIN (2019) [57]	Spatial-based ConvGNN	A, X	-	sum	$O(m)$
ClusterGCN (2019) [58]	Spatial-based ConvGNN	A, X	-	-	-

for large graphs, as GGNN needs to run the recurrent function multiple times over all nodes, requiring the intermediate states of all nodes to be stored in memory.

Stochastic steady-state embedding (SSE) proposes a learning algorithm that is more scalable to large graphs [18]. SSE updates node hidden states recurrently in a stochastic and asynchronous fashion. It alternatively samples a batch of nodes for state update and a batch of nodes for gradient computation. To maintain stability, the recurrent function of SSE is defined as a weighted average of the historical states and new states, which takes the form

$$\mathbf{h}_v^{(t)} = (1-\alpha)\mathbf{h}_v^{(t-1)} + \alpha \mathbf{W_1}\sigma\left(\mathbf{W_2}\left[\mathbf{x}_v, \sum_{u\in N(v)}[\mathbf{h}_u^{(t-1)}, \mathbf{x}_u]\right]\right) \tag{3}$$

where α is a hyperparameter and $\mathbf{h}_v^{(0)}$ is initialized randomly. While conceptually important, SSE does not theoretically prove that the node states will gradually converge to fixed points by applying (3) repeatedly.

V. CONVOLUTIONAL GRAPH NEURAL NETWORKS

ConvGNNs are closely related to recurrent graph neural networks. Instead of iterating node states with contractive constraints, ConvGNNs address the cyclic mutual dependencies architecturally using a fixed number of layers with different weights in each layer. This key distinction is illustrated in Fig. 3. As graph convolutions are more efficient and convenient to composite with other neural networks, the popularity of ConvGNNs has been rapidly growing in recent years. ConvGNNs fall into two categories: spectral-based and spatial-based. Spectral-based approaches define graph convolutions

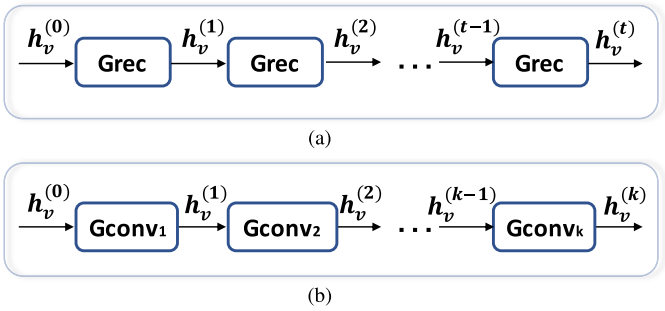


Fig. 3. RecGNNs versus ConvGNNs (a) RecGNNs use the same graph recurrent layer (Grec) in updating node representations. (b) ConvGNNs use a different graph convolutional layer (Gconv) in updating node representations.

by introducing filters from the perspective of graph signal processing [82], where the graph convolutional operation is interpreted as removing noises from graph signals. Spatial-based approaches inherit ideas from RecGNNs to define graph convolutions by information propagation. Since GCN [22] bridged the gap between spectral-based approaches and spatial-based approaches, spatial-based methods have developed rapidly recently due to its attractive efficiency, flexibility, and generality.

A. Spectral-Based ConvGNNs

Background: Spectral-based methods have a solid mathematical foundation in graph signal processing [82]–[84]. They assume graphs to be undirected. The normalized graph Laplacian matrix is a mathematical representation of an undirected graph, defined as $\mathbf{L} = \mathbf{I_n} - \mathbf{D}^{-(1/2)}\mathbf{A}\mathbf{D}^{-(1/2)}$, 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 $\mathbf{U} = [\mathbf{u_0}, \mathbf{u_1}, \ldots, \mathbf{u_{n-1}}] \in \mathbf{R}^{n\times n}$ is the matrix of eigenvectors ordered by eigenvalues and $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues (spectrum), $\mathbf{\Lambda}_{ii} = \lambda_i$. 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 all nodes of a graph, where x_i is the value of the ith node. The graph Fourier transform to a signal $\mathbf{x}$ is defined as $\mathscr{F}(\mathbf{x}) = \mathbf{U}^T\mathbf{x}$, and the inverse graph Fourier transform is defined as $\mathscr{F}^{-1}(\hat{\mathbf{x}}) = \mathbf{U}\hat{\mathbf{x}}$, where $\hat{\mathbf{x}}$ represents the resulted signal from the graph Fourier transform. The graph Fourier transform projects the input graph signal to the orthonormal space, where the basis is formed by eigenvectors of the normalized graph Laplacian. Elements of the transformed signal $\hat{\mathbf{x}}$ are the coordinates of the graph signal in the new space so that the input signal can be represented as $\mathbf{x} = \sum_i \hat{x}_i\mathbf{u}_i$, which is exactly the inverse graph Fourier transform. Now, the graph convolution of the input signal $\mathbf{x}$ with a filter $\mathbf{g} \in \mathbf{R}^n$ is defined as

$$\begin{aligned}\mathbf{x} *_G \mathbf{g} &= \mathscr{F}^{-1}(\mathscr{F}(\mathbf{x}) \odot \mathscr{F}(\mathbf{g})) \\ &= \mathbf{U}(\mathbf{U}^T\mathbf{x} \odot \mathbf{U}^T\mathbf{g})\end{aligned} \tag{4}$$

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

$$\mathbf{x} *_G \mathbf{g}_\theta = \mathbf{U}\mathbf{g}_\theta\mathbf{U}^T\mathbf{x}. \tag{5}$$

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

Spectral CNN [19] assumes that the filter $\mathbf{g}_\theta = \mathbf{\Theta}_{i,j}^{(k)}$ is a set of learnable parameters and considers graph signals with multiple channels. The graph convolutional layer of Spectral CNN is defined as

$$\mathbf{H}_{:,j}^{(k)} = \sigma\left(\sum_{i=1}^{f_{k-1}} \mathbf{U}\mathbf{\Theta}_{i,j}^{(k)}\mathbf{U}^T\mathbf{H}_{:,i}^{(k-1)}\right) \quad (j = 1, 2, \ldots, f_k) \tag{6}$$

where k is the layer index, $\mathbf{H}^{(k-1)} \in \mathbf{R}^{n\times f_{k-1}}$ is the input graph signal, $\mathbf{H}^{(0)} = \mathbf{X}$, f_{k-1} is the number of input channels, f_k is the number of output channels, and $\mathbf{\Theta}_{i,j}^{(k)}$ is a diagonal matrix filled with learnable parameters. Due to the eigen-decomposition of the Laplacian matrix, spectral CNN faces three limitations. First, any perturbation to a graph results in a change of eigenbasis. Second, the learned filters are domain dependent, which means that they cannot be applied to a graph with a different structure. Third, eigendecomposition requires $O(n^3)$ computational complexity. In the follow-up works, ChebNet [21] and GCN [22] reduce the computational complexity to $O(m)$ by making several approximations and simplifications.

Chebyshev spectral CNN (ChebNet) [21] approximates the filter $\mathbf{g}_\theta$ by the Chebyshev polynomials of the diagonal matrix of eigenvalues, i.e., $\mathbf{g}_\theta = \sum_{i=0}^{K}\theta_i T_i(\tilde{\mathbf{\Lambda}})$, where $\tilde{\mathbf{\Lambda}} = 2\mathbf{\Lambda}/\lambda_{\max} - \mathbf{I_n}$, and the values of $\tilde{\mathbf{\Lambda}}$ lie in $[-1, 1]$. The Chebyshev polynomials are defined recursively by $T_i(\mathbf{x}) = 2\mathbf{x}T_{i-1}(\mathbf{x}) - T_{i-2}(\mathbf{x})$ with $T_0(\mathbf{x}) = 1$ and $T_1(\mathbf{x}) = \mathbf{x}$. As a result, the convolution of a graph signal $\mathbf{x}$ with the defined filter $\mathbf{g}_\theta$ is

$$\mathbf{x} *_G \mathbf{g}_\theta = \mathbf{U}\left(\sum_{i=0}^{K}\theta_i T_i(\tilde{\mathbf{\Lambda}})\right)\mathbf{U}^T\mathbf{x} \tag{7}$$

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

$$\mathbf{x} *_G \mathbf{g}_\theta = \sum_{i=0}^{K}\theta_i T_i(\tilde{\mathbf{L}})\mathbf{x}. \tag{8}$$

As an improvement over Spectral CNN, the filters defined by ChebNet are localized in space, which means that filters can extract local features independently of the graph size. The spectrum of ChebNet is mapped to $[-1, 1]$ linearly. CayleyNet [23] further applies the Cayley polynomials that are parametric rational complex functions to capture narrow frequency bands. The spectral graph convolution of CayleyNet is defined as

$$\mathbf{x} *_G \mathbf{g}_\theta = c_0\mathbf{x} + 2\mathrm{Re}\left\{\sum_{j=1}^{r} c_j(h\mathbf{L} - i\mathbf{I})^j(h\mathbf{L} + i\mathbf{I})^{-j}\mathbf{x}\right\} \tag{9}$$

where $\mathrm{Re}(\cdot)$ returns the real part of a complex number, c_0 is a real coefficent, c_j is a complex coefficent, i is the imaginary number, and h is a parameter that controls the spectrum of a Cayley filter. While preserving spatial locality, CayleyNet

shows that ChebNet can be considered as a special case of CayleyNet.

Graph convolutional network (GCN) [22] introduces a first-order approximation of ChebNet. Assuming that $K = 1$ and $\lambda_{\max} = 2$, (8) is simplified as

$$\mathbf{x} *_G \mathbf{g}_\theta = \theta_0 \mathbf{x} - \theta_1 \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \mathbf{x}. \tag{10}$$

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

$$\mathbf{x} *_G \mathbf{g}_\theta = \theta \left(\mathbf{I_n} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \right) \mathbf{x}. \tag{11}$$

To allow multichannels of inputs and outputs, GCN modifies (11) into a compositional layer, defined as

$$\mathbf{H} = \mathbf{X} *_G \mathbf{g}_\Theta = f(\bar{\mathbf{A}} \mathbf{X} \mathbf{\Theta}) \tag{12}$$

where $\bar{\mathbf{A}} = \mathbf{I_n} + \mathbf{D}^{-(1/2)} \mathbf{A} \mathbf{D}^{-(1/2)}$ and $f(\cdot)$ is an activation function. Using $\mathbf{I_n} + \mathbf{D}^{-(1/2)} \mathbf{A} \mathbf{D}^{-(1/2)}$ empirically causes numerical instability to GCN. To address this problem, GCN applies a normalization trick to replace $\bar{\mathbf{A}} = \mathbf{I_n} + \mathbf{D}^{-(1/2)} \mathbf{A} \mathbf{D}^{-(1/2)}$ by $\bar{\mathbf{A}} = \tilde{\mathbf{D}}^{-(1/2)} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-(1/2)}$ with $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I_n}$ and $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$. Being a spectral-based method, GCN can be also interpreted as a spatial-based method. From a spatial-based perspective, GCN can be considered as aggregating feature information from a node's neighborhood. Equation (12) can be expressed as

$$\mathbf{h}_v = f\left(\mathbf{\Theta}^T \left(\sum_{u \in \{N(v) \cup v\}} \bar{A}_{v,u} \mathbf{x}_u \right) \right) \quad \forall v \in V. \tag{13}$$

Several recent works made incremental improvements over GCN [22] by exploring alternative symmetric matrices. Adaptive GCN (AGCN) [40] learns hidden structural relations unspecified by the graph adjacency matrix. It constructs a so-called residual graph adjacency matrix through a learnable distance function that takes two nodes' features as inputs. Dual GCN (DGCN) [41] introduces a dual-graph convolutional architecture with two graph convolutional layers in parallel. While these two layers share parameters, they use the normalized adjacency matrix $\bar{\mathbf{A}}$ and the positive pointwise mutual information (PPMI) matrix that captures nodes co-occurrence information through random walks sampled from a graph. The PPMI matrix is defined as

$$\mathbf{PPMI}_{v_1, v_2} = \max\left(\log\left(\frac{\text{count}(v_1, v_2) \cdot |D|}{\text{count}(v_1)\text{count}(v_2)} \right), 0 \right) \tag{14}$$

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

B. Spatial-Based ConvGNNs

Analogous to the convolutional operation of a conventional CNN on an image, spatial-based methods define graph convolutions based on a node's spatial relations. Images can be considered as a special form of a graph with each pixel representing a node. Each pixel is directly connected to its nearby pixels, as shown in Fig. 1(a). A filter is applied to a 3×3 patch by taking the weighted average of pixel values of the central node and its neighbors across each channel. Similarly, the spatial-based graph convolutions convolve the central node's representation with its neighbors' representations to derive the updated representation for the central node, as shown in Fig. 1(b). From another perspective, spatial-based ConvGNNs share the same idea of information propagation/message passing with RecGNNs. The spatial graph convolutional operation essentially propagates node information along edges.

The neural network for graphs (NN4G) [24], proposed in parallel with GNN*, is the first work toward spatial-based ConvGNNs. Distinctively different from RecGNNs, NN4G learns graph mutual dependence through a compositional neural architecture with independent parameters at each layer. The neighborhood of a node can be extended through the incremental construction of the architecture. NN4G performs graph convolutions by summing up a node's neighborhood information directly. It also applies residual connections and skip connections to memorize information over each layer. As a result, NN4G derives its next-layer node states by

$$\mathbf{h}_v^{(k)} = f\left(\mathbf{W}^{(k)^T} \mathbf{x}_v + \sum_{i=1}^{k-1} \sum_{u \in N(v)} \mathbf{\Theta}^{(k)^T} \mathbf{h}_u^{(k-1)} \right) \tag{15}$$

where $f(\cdot)$ is an activation function and $\mathbf{h}_v^{(0)} = \mathbf{0}$. Equation (15) can also be written in a matrix form

$$\mathbf{H}^{(k)} = f\left(\mathbf{X} \mathbf{W}^{(k)} + \sum_{i=1}^{k-1} \mathbf{A} \mathbf{H}^{(k-1)} \mathbf{\Theta}^{(k)} \right) \tag{16}$$

which resembles the form of GCN [22]. One difference is that NN4G uses the unnormalized adjacency matrix, which may potentially cause hidden node states to have extremely different scales. Contextual graph Markov model (CGMM) [47] proposes a probabilistic model inspired by NN4G. While maintaining spatial locality, CGMM has the benefit of probabilistic interpretability.

Diffusion CNN (DCNN) [25] regards graph convolutions as a diffusion process. It assumes that information is transferred from one node to one of its neighboring nodes with a certain transition probability so that information distribution can reach equilibrium after several rounds. DCNN defines the diffusion graph convolution (DGC) as

$$\mathbf{H}^{(k)} = f(\mathbf{W}^{(k)} \odot \mathbf{P}^k \mathbf{X}) \tag{17}$$

where $f(\cdot)$ is an activation function and the probability transition matrix $\mathbf{P} \in \mathbf{R}^{n \times n}$ is computed by $\mathbf{P} = \mathbf{D}^{-1} \mathbf{A}$. Note that in DCNN, the hidden representation matrix $\mathbf{H}^{(k)}$ remains the same dimension as the input feature matrix $\mathbf{X}$ and is not a function of its previous hidden representation matrix $\mathbf{H}^{(k-1)}$. DCNN concatenates $\mathbf{H}^{(1)}, \mathbf{H}^{(2)}, \ldots, \mathbf{H}^{(K)}$ together as the final model outputs. As the stationary distribution of a diffusion process is a summation of power series of probability transition matrices, DGC [72] sums up outputs at each diffusion

step instead of concatenation. It defines the DGC by

$$\mathbf{H} = \sum_{k=0}^{K} f(\mathbf{P}^k \mathbf{X} \mathbf{W}^{(k)}) \tag{18}$$

where $\mathbf{W}^{(k)} \in \mathbf{R}^{D \times F}$ and $f(\cdot)$ is an activation function. Using the power of a transition probability matrix implies that distant neighbors contribute very little information to a central node. PGC-DGCNN [46] increases the contributions of distant neighbors based on the shortest paths. It defines a shortest-path adjacency matrix $\mathbf{S}^{(j)}$. If the shortest path from a node v to a node u is of length j, then $\mathbf{S}^{(j)}_{v,u} = 1$, otherwise 0. With a hyperparameter r to control the receptive field size, PGC-DGCNN introduces a graph convolutional operation as follows:

$$\mathbf{H}^{(k)} = \|_{j=0}^{r} f((\tilde{\mathbf{D}}^{(j)})^{-1} \mathbf{S}^{(j)} \mathbf{H}^{(k-1)} \mathbf{W}^{(j,k)}) \tag{19}$$

where $\tilde{D}^{(j)}_{ii} = \sum_l S^{(j)}_{i,l}$, $\mathbf{H}^{(0)} = \mathbf{X}$, and $\|$ represents the concatenation of vectors. The calculation of the shortest-path adjacency matrix can be expensive with $O(n^3)$ at maximum. Partition graph convolution (PGC) [75] partitions a node's neighbors into Q groups based on certain criteria not limited to shortest paths. PGC constructs Q adjacency matrices according to the defined neighborhood by each group. Then, PGC applies GCN [22] with a different parameter matrix to each neighbor group and sums the results

$$\mathbf{H}^{(k)} = \sum_{j=1}^{Q} \bar{\mathbf{A}}^{(j)} \mathbf{H}^{(k-1)} \mathbf{W}^{(j,k)} \tag{20}$$

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

The message-passing neural network (MPNN) [27] outlines a general framework of spatial-based ConvGNNs. It treats graph convolutions as a message-passing process in which information can be passed from one node to another along edges directly. MPNN runs K-step message-passing iterations to let information propagate further. The message-passing function (namely, the spatial graph convolution) is defined as

$$\mathbf{h}_v^{(k)} = U_k\left(\mathbf{h}_v^{(k-1)}, \sum_{u \in N(v)} M_k\left(\mathbf{h}_v^{(k-1)}, \mathbf{h}_u^{(k-1)}, \mathbf{x}^e_{vu}\right)\right) \tag{21}$$

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

$$\mathbf{h}_G = R\left(\mathbf{h}_v^{(K)} | v \in G\right) \tag{22}$$

where $R(\cdot)$ represents the readout function with learnable parameters. MPNN can cover many existing GNNs by assuming different forms of $U_k(\cdot)$, $M_k(\cdot)$, and $R(\cdot)$, such as [22] and [85]–[87]. However, graph isomorphism network (GIN) [57] finds that previous MPNN-based methods are incapable of distinguishing different graph structures based on the graph embedding they produced. To amend this drawback, GIN adjusts the weight of the central node by a learnable parameter $\epsilon^{(k)}$. It performs graph convolutions by

$$\mathbf{h}_v^{(k)} = \text{MLP}\left((1 + \epsilon^{(k)})\mathbf{h}_v^{(k-1)} + \sum_{u \in N(v)} \mathbf{h}_u^{(k-1)}\right) \tag{23}$$

where MLP$(\cdot)$ represents a multilayer perceptron.

As the number of neighbors of a node can vary from one to a thousand or even more, it is inefficient to take the full size of a node's neighborhood. GraphSage [42] adopts sampling to obtain a fixed number of neighbors for each node. It performs graph convolutions by

$$\mathbf{h}_v^{(k)} = \sigma\left(\mathbf{W}^{(k)} \cdot f_k\left(\mathbf{h}_v^{(k-1)}, \left\{\mathbf{h}_u^{(k-1)} \forall u \in S_{\mathcal{N}(v)}\right\}\right)\right) \tag{24}$$

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

Graph attention network (GAT) [43] assumes that contributions of neighboring nodes to the central node are neither identical like GraphSage [42], nor predetermined like GCN [22] (this difference is illustrated in Fig. 4). GAT adopts attention mechanisms to learn the relative weights between two connected nodes. The graph convolutional operation according to GAT is defined as

$$\mathbf{h}_v^{(k)} = \sigma\left(\sum_{u \in \mathcal{N}(v) \cup v} \alpha^{(k)}_{vu} \mathbf{W}^{(k)} \mathbf{h}_u^{(k-1)}\right) \tag{25}$$

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

$$\alpha^{(k)}_{vu} = \text{softmax}\left(g(\mathbf{a}^T\left[\mathbf{W}^{(k)}\mathbf{h}_v^{(k-1)} \| \mathbf{W}^{(k)}\mathbf{h}_u^{(k-1)}\right)\right) \tag{26}$$

where $g(\cdot)$ is a LeakyReLU activation function and $\mathbf{a}$ is a vector of learnable parameters. The softmax function ensures that the attention weights sum up to one over all neighbors of the node v. GAT further performs the multihead attention to increase the model's expressive capability. This shows an impressive improvement over GraphSage on node

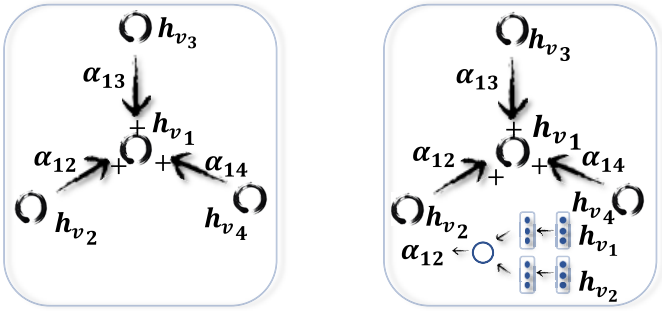


Fig. 4. Differences between GCN [22] and GAT [43]. (a) GCN [22] explicitly assigns a nonparametric weight $a_{ij} = (1/(\deg(v_i)\deg(v_j))^{1/2})$ to the neighbor v_j of v_i during the aggregation process. (b) GAT [43] implicitly captures the weight a_{ij} via an end-to-end neural network architecture so that more important nodes receive larger weights.

classification tasks. While GAT assumes the contributions of attention heads are equal, gated attention network (GAAN) [48] introduces a self-attention mechanism that computes an additional attention score for each attention head. Apart from applying graph attention spatially, GeniePath [55] further proposes an LSTM-like gating mechanism to control information flow across graph convolutional layers. There are other graph attention models that might be of interest [88], [89]. However, they do not belong to the ConvGNN framework.

Mixture model network (MoNet) [44] adopts a different approach to assign different weights to a node's neighbors. It introduces node pseudocoordinates to determine the relative position between a node and its neighbor. Once the relative position between two nodes is known, a weight function maps the relative position to the relative weight between these two nodes. In such a way, the parameters of a graph filter can be shared across different locations. Under the MoNet framework, several existing approaches for manifolds, such as geodesic CNN (GCNN) [90], anisotropic CNN (ACNN) [91], and spline CNN [92], and for graphs, such as GCN [22] and DCNN [25], can be generalized as special instances of MoNet by constructing nonparametric weight functions. MoNet additionally proposes a Gaussian kernel with learnable parameters to learn the weight function adaptively.

Another distinct line of works achieves weight sharing across different locations by ranking a node's neighbors based on certain criteria and associating each ranking with a learnable weight. PATCHY-SAN [26] orders neighbors of each node according to their graph labelings and selects the top q neighbors. Graph labelings are essentially node scores, which can be derived by node degree, centrality, and the Weisfeiler–Lehman (WL) color [93], [94]. As each node now has a fixed number of ordered neighbors, graph-structured data can be converted into the grid-structured data. PATCHY-SAN applies a standard 1-D convolutional filter to aggregate neighborhood feature information, where the order of the filter's weights corresponds to the order of a node's neighbors. The ranking criterion of PATCHY-SAN only considers graph structures, which requires heavy computation for data processing. Large-scale GCN (LGCN) [45] ranks a node's neighbors based on node feature information. For each node, LGCN assembles a feature matrix that consists of its neighborhood and sorts this feature matrix along each column. The first q rows of the sorted feature matrix are taken as the input data for the central node.

1) Improvement in Terms of Training Efficiency: Training ConvGNNs, such as GCN [22], is usually required to save the whole graph data and intermediate states of all nodes into memory. The full-batch training algorithm for ConvGNNs suffers significantly from the memory overflow problem, especially when a graph contains millions of nodes. To save memory, GraphSage [42] proposes a batch-training algorithm for ConvGNNs. It samples a tree rooted at each node by recursively expanding the root node's neighborhood by K steps with fixed sample size. For each sampled tree, GraphSage computes the root node's hidden representation by hierarchically aggregating hidden node representations from bottom to top.

Fast learning with GCN (FastGCN) [49] samples a fixed number of nodes for each graph convolutional layer instead of sampling a fixed number of neighbors for each node like GraphSage [42]. It interprets graph convolutions as integral transforms of embedding functions of nodes under probability measures. The Monte Carlo approximation and variance reduction techniques are employed to facilitate the training process. As FastGCN samples nodes independently for each layer, between-layer connections are potentially sparse. Huang *et al.* [51] propose an adaptive layerwise sampling approach, where node sampling for the lower layer is conditioned on the top one. This method achieves higher accuracy compared with FastGCN at the cost of employing a much more complicated sampling scheme.

In another work, stochastic training of GCNs (StoGCN) [50] reduces the receptive field size of a graph convolution to an arbitrarily small scale using historical node representations as a control variate. StoGCN achieves comparable performance even with two neighbors per node. However, StoGCN still has to save intermediate states of all nodes, which is memory consuming for large graphs.

Cluster-GCN [58] samples a subgraph using a graph clustering algorithm and performs graph convolutions to nodes within the sampled subgraph. As the neighborhood search is also restricted within the sampled subgraph, Cluster-GCN is capable of handling larger graphs and using deeper architectures at the same time, in less time and with less memory. Cluster-GCN notably provides a straightforward comparison of time complexity and memory complexity for existing ConvGNN training algorithms. We analyze its results based on Table IV.

In Table IV, GCN [22] is the baseline method that conducts the full-batch training. GraphSage saves memory at the cost of sacrificing time efficiency. Meanwhile, the time and memory complexities of GraphSage grow exponentially with an increase of K and r. The time complexity of Sto-GCN is the highest, and the bottleneck of the memory remains unsolved. However, Sto-GCN can achieve satisfactory performance with a very small r. The time complexity of Cluster-GCN remains the same as the baseline method since it does not introduce redundant computations. Of all the methods, Cluster-GCN realizes the lowest memory complexity.

2) Comparison Between Spectral and Spatial Models: Spectral models have a theoretical foundation in graph signal processing. By designing new graph signal filters (e.g., Cayleynets [23]), one can build new ConvGNNs. However, spatial models are preferred over spectral models due to efficiency, generality, and flexibility issues. First, spectral models are less efficient than spatial models. Spectral models either need to perform eigenvector computation or handle the whole graph at the same time. Spatial models are more scalable to large graphs as they directly perform convolutions in the graph domain via information propagation. The computation can be performed in a batch of nodes instead of the whole graph. Second, spectral models that rely on a graph Fourier basis generalize poorly to new graphs. They assume a fixed graph. Any perturbations to a graph would result in a change of eigenbasis. Spatial-based models, on the other hand, perform graph convolutions locally on each node, where weights can be easily shared across

TABLE IV

TIME AND MEMORY COMPLEXITY COMPARISON FOR CONVGNN TRAINING ALGORITHMS (SUMMARIZED BY [58]). n IS THE TOTAL NUMBER OF NODES. m IS THE TOTAL NUMBER OF EDGES. K IS THE NUMBER OF LAYERS. s IS THE BATCH SIZE. r IS THE NUMBER OF NEIGHBORS BEING SAMPLED FOR EACH NODE. FOR SIMPLICITY, THE DIMENSIONS OF THE NODE HIDDEN FEATURES REMAIN CONSTANT, DENOTED BY d

Complexity	GCN [22]	GraphSage [42]	FastGCN [49]	StoGCN [50]	Cluster-GCN [58]
Time	$O(Kmd + Knd^2)$	$O(r^K nd^2)$	$O(Krnd^2)$	$O(Kmd + Knd^2 + r^K nd^2)$	$O(Kmd + Knd^2)$
Memory	$O(Knd + Kd^2)$	$O(sr^K d + Kd^2)$	$O(Ksrd + Kd^2)$	$O(Knd + Kd^2)$	$O(Ksd + Kd^2)$

different locations and structures. Third, spectral-based models are limited to operate on undirected graphs. Spatial-based models are more flexible to handle multisource graph inputs, such as edge inputs [15], [27], [86], [95], [96], directed graphs [25], [72], signed graphs [97], and heterogeneous graphs [98], [99], because these graph inputs can be incorporated into the aggregation function easily.

C. Graph Pooling Modules

After a GNN generates node features, we can use them for the final task. However, using all these features directly can be computationally challenging; thus, a downsampling strategy is needed. Depending on the objective and the role it plays in the network, different names are given to this strategy.

1) The pooling operation aims to reduce the size of parameters by downsampling the nodes to generate smaller representations and, thus, avoid overfitting, permutation invariance, and computational complexity issues.
2) The readout operation is mainly used to generate graph-level representation based on node representations. Their mechanism is very similar.

In this section, we use pooling to refer to all kinds of downsampling strategies applied to GNNs.

In some earlier works, the graph coarsening algorithms use eigendecomposition to coarsen graphs based on their topological structure. However, these methods suffer from the time complexity issue. The Graclus algorithm [100] is an alternative of eigendecomposition to calculate a clustering version of the original graph. Some recent works [23] employed it as a pooling operation to coarsen graphs.

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

$$\mathbf{h}_G = \text{mean}/\max/\text{sum}\left(\mathbf{h}_1^{(K)}, \mathbf{h}_2^{(K)}, \ldots, \mathbf{h}_n^{(K)}\right) \tag{27}$$

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

Henaff *et al.* [20] show that performing a simple max/mean pooling at the beginning of the network is especially important to reduce the dimensionality in the graph domain and mitigate the cost of the expensive graph Fourier transform operation. Furthermore, some works [17], [27], [46] also use attention mechanisms to enhance the mean/sum pooling.

Even with attention mechanisms, the reduction operation (such as sum pooling) is not satisfactory since it makes the embedding inefficient; a fixed-size embedding is generated regardless of the graph size. Vinyals *et al.* [101] propose the Set2Set method to generate a memory that increases with the size of the input. It then implements an LSTM that intends to integrate order-dependent information into the memory embedding before a reduction is applied that would, otherwise, destroy that information.

Defferrard *et al.* [21] address this issue in another way by rearranging nodes of a graph in a meaningful way. They devise an efficient pooling strategy in their approach ChebNet. Input graphs are first coarsened into multiple levels by the Graclus algorithm [100]. After coarsening, the nodes of the input graph and its coarsened version are rearranged into a balanced binary tree. Arbitrarily aggregating the balanced binary tree from bottom to top will arrange similar nodes together. Pooling such a rearranged signal is much more efficient than pooling the original.

Zhang *et al.* [52] propose the DGCNN with a similar pooling strategy named SortPooling that performs pooling by rearranging nodes to a meaningful order. Different from ChebNet [21], DGCNN sorts nodes according to their structural roles within the graph. The graph's unordered node features from spatial graph convolutions are treated as continuous WL colors [93], and they are then used to sort nodes. In addition to sorting the node features, it unifies the graph size to q by truncating/extending the node feature matrix. The last $n - q$ rows are deleted if $n > q$; otherwise, $q - n$ zero rows are added.

The aforementioned pooling methods mainly consider graph features and ignore the structural information of graphs. Recently, a differentiable pooling (DiffPool) [54] is proposed, which can generate hierarchical representations of graphs. Compared with all previous coarsening methods, DiffPool does not simply cluster the nodes in a graph but learns a cluster assignment matrix $\mathbf{S}$ at layer k referred to as $\mathbf{S}^{(k)} \in \mathbf{R}^{n_k \times n_{k+1}}$, where n_k is the number of nodes at the kth layer. The probability values in matrix $\mathbf{S}^{(k)}$ are being generated based on node features and topological structure using

$$\mathbf{S}^{(k)} = \text{softmax}(\text{ConvGNN}_k(\mathbf{A}^{(k)}, \mathbf{H}^{(k)})). \tag{28}$$

The core idea of this is to learn comprehensive node assignments that consider both topological and feature information of a graph, so (28) can be implemented with any standard ConvGNNs. However, the drawback of DiffPool is that it generates dense graphs after pooling, and thereafter, the computational complexity becomes $O(n^2)$.

Most recently, the SAGPool [102] approach is proposed, which considers both node features and graph topology and learns the pooling in a self-attention manner.

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

D. Discussion of Theoretical Aspects

We discuss the theoretical foundation of GNNs from different perspectives.

1) Shape of Receptive Field: The receptive field of a node is the set of nodes that contribute to the determination of its final node representation. When compositing multiple spatial graph convolutional layers, the receptive field of a node grows one step ahead toward its distant neighbors each time. Micheli [24] prove that a finite number of spatial graph convolutional layers exists such that for each node $v \in V$, the receptive field of node v covers all nodes in the graph. As a result, a ConvGNN is able to extract global information by stacking local graph convolutional layers.

2) VC Dimension: The VC dimension is a measure of model complexity defined as the largest number of points that can be shattered by a model. There are few works on analyzing the VC dimension of GNNs. Given the number of model parameter p and the number of nodes n, Scarselli *et al.* [103] derive that the VC dimension of a GNN* [15] is $O(p^4n^2)$ if it uses the sigmoid or tangent hyperbolic activation and is $O(p^2n)$ if it uses the piecewise polynomial activation function. This result suggests that the model complexity of a GNN* [15] increases rapidly with p and n if the sigmoid or tangent hyperbolic activation is used.

3) Graph Isomorphism: Two graphs are isomorphic if they are topologically identical. Given two nonisomorphic graphs G_1 and G_2, Xu *et al.* [57] prove that if a GNN maps G_1 and G_2 to different embeddings, these two graphs can be identified as nonisomorphic by the WL test of isomorphism [93]. They show that common GNNs, such as GCN [22] and GraphSage [42], are incapable of distinguishing different graph structures. Xu *et al.* [57] further prove if the aggregation functions and the readout functions of a GNN are injective, the GNN is at most as powerful as the WL test in distinguishing different graphs.

4) Equivariance and Invariance: A GNN must be an equivariant function when performing node-level tasks and must be an invariant function when performing graph-level tasks. For node-level tasks, let $f(\mathbf{A}, \mathbf{X}) \in R^{n\times d}$ be a GNN and $\mathbf{Q}$ be any permutation matrix that changes the order of nodes. A GNN is equivariant if it satisfies $f(\mathbf{QAQ}^T, \mathbf{QX}) = \mathbf{Q}f(\mathbf{A}, \mathbf{X})$. For graph-level tasks, let $f(\mathbf{A}, \mathbf{X}) \in R^d$. A GNN is invariant if it satisfies $f(\mathbf{QAQ}^T, \mathbf{QX}) = f(\mathbf{A}, \mathbf{X})$. In order to achieve equivariance or invariance, components of a GNN must be invariant to node orderings. Maron *et al.* [104] theoretically study the characteristics of permutation invariant and equivariant linear layers for graph data.

5) Universal Approximation: It is well known that multiperceptron feedforward neural networks with one hidden layer can approximate any Borel measurable function [105]. The universal approximation capability of GNNs has seldom been studied. Hammer *et al.* [106] prove that cascade correlation can approximate functions with structured outputs. Scarselli *et al.* [107] prove that a RecGNN [15] can approximate any function that preserves unfolding equivalence up to any degree of precision. Two nodes are unfolding equivalent if their unfolding trees are identical, where the unfolding tree of a node is constructed by iteratively extending a node's neighborhood at a certain depth. Xu *et al.* [57] show that ConvGNNs under the framework of message passing [27] are not universal approximators of continuous functions defined on multisets. Maron *et al.* [104] prove that an invariant graph network can approximate an arbitrary invariant function defined on graphs.

VI. Graph Autoencoders

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

A. Network Embedding

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

Earlier approaches mainly employ multilayer perceptrons to build GAEs for network embedding learning. Deep neural networks for graph representations (DNGRs) [59] use a stacked denoising autoencoder [108] to encode and decode the PPMI matrix via multilayer perceptrons. Concurrently, structural deep network embedding (SDNE) [60] uses a stacked autoencoder to preserve the node first-order proximity and second-order proximity jointly. SDNE proposes two loss functions on the outputs of the encoder and the outputs of the decoder separately. The first loss function enables the learned network embeddings to preserve the node first-order proximity by minimizing the distance between a node's network embedding and its neighbors' network embeddings. The first loss function $L_{1\text{st}}$ is defined as

$$L_{1\text{st}} = \sum_{(v,u)\in E} A_{v,u}||\text{enc}(\mathbf{x}_v) - \text{enc}(\mathbf{x}_u)||^2 \tag{29}$$

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

$$L_{2\text{nd}} = \sum_{v\in V} ||(\text{dec}(\text{enc}(\mathbf{x}_v)) - \mathbf{x}_v) \odot \mathbf{b}_v||^2 \tag{30}$$

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

DNGR [59] and SDNE [60] only consider node structural information that is about the connectivity between pairs of nodes. They ignore the nodes that may contain feature information that depicts the attributes of nodes themselves. GAE*[3] [61] leverages GCN [22] to encode node structural information

[3]We name it GAE* to avoid ambiguity in this article.

TABLE V
MAIN CHARACTERISTICS OF THE SELECTED GAEs

Approaches	Inputs	Encoder	Decoder	Objective
DNGR (2016) [59]	A	a multi-layer perceptron	a multi-layer perceptron	reconstruct the PPMI matrix
SDNE (2016) [60]	A	a multi-layer perceptron	a multi-layer perceptron	preserve node 1st-order and 2nd-order proximity
GAE* (2016) [61]	A, X	a ConvGNN	a similarity measure	reconstruct the adjacency matrix
VGAE (2016) [61]	A, X	a ConvGNN	a similarity measure	learn the generative distribution of data
ARVGA (2018) [62]	A, X	a ConvGNN	a similarity measure	learn the generative distribution of data adversarially
DNRE (2018) [63]	A	an LSTM network	an identity function	recover network embedding
NetRA (2018) [64]	A	an LSTM network	an LSTM network	recover network embedding with adversarial training
DeepGMG (2018) [65]	A, X, X^e	a RecGNN	a decision process	maximize the expected joint log-likelihood
GraphRNN (2018) [66]	A	a RNN	a decision process	maximize the likelihood of permutations
GraphVAE (2018) [67]	A, X, X^e	a ConvGNN	a multi-layer perceptron	optimize the reconstruction loss
RGVAE (2018) [68]	A, X, X^e	a CNN	a deconvolutional net	optimize the reconstruction loss with validity constraints
MolGAN (2018) [69]	A, X, X^e	a ConvGNN	a multi-layer perceptron	optimize the generative adversarial loss and the RL loss
NetGAN (2018) [70]	A	an LSTM network	an LSTM network	optimize the generative adversarial loss

and node feature information at the same time. The encoder of GAE* consists of two graph convolutional layers, which takes the form

$$\mathbf{Z} = \text{enc}(\mathbf{X}, \mathbf{A}) = \text{Gconv}(f(\text{Gconv}(\mathbf{A}, \mathbf{X}; \mathbf{\Theta_1})); \mathbf{\Theta_2}) \quad (31)$$

where $\mathbf{Z}$ denotes the network embedding matrix of a graph, $f(\cdot)$ is a ReLU activation function, and the Gconv$(\cdot)$ function is a graph convolutional layer defined by (12). The decoder of GAE* aims to decode node relational information from their embeddings by reconstructing the graph adjacency matrix, which is defined as

$$\hat{\mathbf{A}}_{v,u} = \text{dec}(\mathbf{z}_v, \mathbf{z}_u) = \sigma\left(\mathbf{z}_v^T \mathbf{z}_u\right) \quad (32)$$

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

Simply reconstructing the graph adjacency matrix may lead to overfitting due to the capacity of the autoencoders. Variational GAE (VGAE) [61] is a variational version of GAE to learn the distribution of data. VGAE optimizes the variational lower bound L

$$L = E_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{A}|\mathbf{Z})] - KL[q(\mathbf{Z}|\mathbf{X}, \mathbf{A})||p(\mathbf{Z})] \quad (33)$$

where $KL(\cdot)$ is the Kullback–Leibler divergence function that measures the distance between two distributions, $p(\mathbf{Z})$ is a Gaussian prior $p(\mathbf{Z}) = \prod_{i=1}^{n} p(\mathbf{z}_i) = \prod_{i=1}^{n} N(\mathbf{z}_i|0, \mathbf{I})$, $p(A_{ij} = 1|\mathbf{z}_i, \mathbf{z}_j) = \text{dec}(\mathbf{z}_i, \mathbf{z}_j) = \sigma(\mathbf{z}_i^T \mathbf{z}_j)$, and $q(\mathbf{Z}|\mathbf{X}, \mathbf{A}) = \prod_{i=1}^{n} q(\mathbf{z}_i|\mathbf{X}, \mathbf{A})$ with $q(\mathbf{z}_i|\mathbf{X}, \mathbf{A}) = N(\mathbf{z}_i|\mu_i, \text{diag}(\sigma_i^2))$. The mean vector μ_i is the ith row of an encoder's outputs defined by (31), and $\log \sigma_i$ is derived similarly as μ_i with another encoder. According to (33), VGAE assumes that the empirical distribution $q(\mathbf{Z}|\mathbf{X}, \mathbf{A})$ should be as close as possible to the prior distribution $p(\mathbf{Z})$. To further enforce that the empirical distribution $q(\mathbf{Z}|\mathbf{X}, \mathbf{A})$ approximates the prior distribution $p(\mathbf{Z})$, adversarially regularized VGAE (ARVGA) [62], [109] employs the training scheme of the generative adversarial networks (GANs) [110]. A GAN plays a competition game between a generator and a discriminator in training generative models. A generator tries to generate "fake samples" to be as real as possible, while a discriminator attempts to distinguish the "fake samples" from real ones. Inspired by GANs, ARVGA endeavors to learn an encoder that produces an empirical distribution $q(\mathbf{Z}|\mathbf{X}, \mathbf{A})$, which is indistinguishable from the prior distribution $p(\mathbf{Z})$.

Similar to GAE*, GraphSage [42] encodes node features with two graph convolutional layers. Instead of optimizing the reconstruction error, GraphSage shows that the relational information between two nodes can be preserved by negative sampling with the loss

$$L(\mathbf{z}_v) = -\log(\text{dec}(\mathbf{z}_v, \mathbf{z}_u)) - QE_{v_n \sim P_n(v)} \log(-\text{dec}(\mathbf{z}_v, \mathbf{z}_{v_n})) \quad (34)$$

where node u is a neighbor of node v, node v_n is a distant node to node v and is sampled from a negative sampling distribution $P_n(v)$, and Q is the number of negative samples. This loss function essentially enforces close nodes to have similar representations and distant nodes to have dissimilar representations. DGI [56] alternatively drives local network embeddings to capture global structural information by maximizing local mutual information. It shows a distinct improvement over GraphSage [42] experimentally.

For the aforementioned methods, they essentially learn network embeddings by solving a link prediction problem. However, the sparsity of a graph causes the number of positive node pairs to be far less than the number of negative node pairs. To alleviate the data sparsity problem in learning network embedding, another line of works convert a graph into sequences by random permutations or random walks. In such a way, those deep learning approaches that are applicable to sequences can be directly used to process graphs. Deep recursive network embedding (DRNE) [63] assumes that a node's network embedding should approximate the aggregation of its neighborhood network embeddings. It adopts a long short-term

memory (LSTM) network [7] to aggregate a node's neighbors. The reconstruction error of DRNE is defined as

$$L = \sum_{v \in V} ||\mathbf{z}_v - \text{LSTM}(\{\mathbf{z}_u | u \in N(v)\})||^2 \tag{35}$$

where $\mathbf{z}_v$ is the network embedding of node v obtained by a dictionary lookup, and the LSTM network takes a random sequence of node v's neighbors ordered by their node degree as inputs. As suggested by (35), DRNE implicitly learns network embeddings via an LSTM network rather than using the LSTM network to generate network embeddings. It avoids the problem that the LSTM network is not invariant to the permutation of node sequences. Network representations with adversarially regularized autoencoders (NetRAs) [64] propose a graph encoder–decoder framework with a general loss function, defined as

$$L = -E_{\mathbf{z} \sim p_{\text{data}}(\mathbf{z})}(\text{dist}(\mathbf{z}, \text{dec}(\text{enc}(\mathbf{z})))) \tag{36}$$

where $\text{dist}(\cdot)$ is the distance measure between the node embedding $\mathbf{z}$ and the reconstructed $\mathbf{z}$. The encoder and decoder of NetRA are LSTM networks with random walks rooted on each node $v \in V$ as inputs. Similar to ARVGA [62], NetRA regularizes the learned network embeddings within a prior distribution via adversarial training. Although NetRA ignores the node permutation variant problem of LSTM networks, the experimental results validate the effectiveness of NetRA.

B. Graph Generation

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

Sequential approaches generate a graph by proposing nodes and edges step by step. Gómez-Bombarelli *et al.* [111], Kusner *et al.* [112], and Dai *et al.* [113] model the generation process of a string representation of molecular graphs named SMILES with deep CNNs and RNNs as the encoder and the decoder, respectively. While these methods are domain-specific, alternative solutions are applicable to general graphs by means of iteratively adding nodes and edges to a growing graph until a certain criterion is satisfied. Deep generative model of graphs (DeepGMG) [65] assumes that the probability of a graph is the sum over all possible node permutations

$$p(G) = \sum_{\pi} p(G, \pi) \tag{37}$$

where π denotes a node ordering. It captures the complex joint probability of all nodes and edges in the graph. DeepGMG generates graphs by making a sequence of decisions, namely, whether to add a node, which node to add, whether to add an edge, and which node to connect to the new node. The decision process of generating nodes and edges is conditioned on the node states and the graph state of a growing graph updated by a RecGNN. In another work, GraphRNN [66] proposes a graph-level RNN and an edge-level RNN to model the generation process of nodes and edges. The graph-level RNN adds a new node to a node sequence each time, while the edge-level RNN produces a binary sequence indicating connections between the new node and the nodes previously generated in the sequence.

Global approaches output a graph all at once. Graph variational autoencoder (GraphVAE) [67] models the existence of nodes and edges as independent random variables. By assuming the posterior distribution $q_\phi(\mathbf{z}|G)$ defined by an encoder and the generative distribution $p_\theta(G|\mathbf{z})$ defined by a decoder, GraphVAE optimizes the variational lower bound

$$L(\phi, \theta; G) = E_{q_\phi(z|G)}[-\log p_\theta(G|\mathbf{z})] + KL[q_\phi(\mathbf{z}|G)||p(\mathbf{z})] \tag{38}$$

where $p(\mathbf{z})$ follows a Gaussian prior and ϕ and θ are learnable parameters. With a ConvGNN as the encoder and a simple multilayer perception as the decoder, GraphVAE outputs a generated graph with its adjacency matrix, node attributes and edge attributes. It is challenging to control the global properties of generated graphs, such as graph connectivity, validity, and node compatibility. Regularized GraphVAE (RGVAE) [68] further imposes validity constraints on a GraphVAE to regularize the output distribution of the decoder. Molecular GAN (MolGAN) [69] integrates convGNNs [114], GANs [115], and reinforcement learning objectives to generate graphs with the desired properties. MolGAN consists of a generator and a discriminator, competing with each other to improve the authenticity of the generator. In MolGAN, the generator tries to propose a fake graph along with its feature matrix, while the discriminator aims to distinguish the fake sample from the empirical data. In addition, a reward network is introduced in parallel with the discriminator to encourage the generated graphs to possess certain properties according to an external evaluator. NetGAN [70] combines LSTMs [7] with the Wasserstein GANs [116] to generate graphs from a random-walk-based approach. NetGAN trains a generator to produce plausible random walks through an LSTM network and enforces a discriminator to identify fake random walks from the real ones. After training, a new graph is derived by normalizing a co-occurrence matrix of nodes computed based on random walks produced by the generator.

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

VII. Spatial–Temporal Graph Neural Networks

Graphs in many real-world applications are dynamic both in terms of graph structures and graph inputs. STGNNs occupy important positions in capturing the dynamicity of graphs. Methods under this category aim to model the dynamic node inputs while assuming interdependency between connected nodes. For example, a traffic network consists of speed sensors placed on roads, where edge weights are determined by the distance between pairs of sensors. As the traffic condition of

one road may depend on its adjacent roads' conditions, it is necessary to consider spatial dependence when performing traffic speed forecasting. As a solution, STGNNs capture spatial and temporal dependencies of a graph simultaneously. The task of STGNNs can be forecasting future node values or labels or predicting spatial–temporal graph labels. STGNNs follow two directions: RNN-based methods and CNN-based methods.

Most RNN-based approaches capture spatial–temporal dependencies by filtering inputs and hidden states passed to a recurrent unit using graph convolutions [48], [71], [72]. To illustrate this, suppose that a simple RNN takes the form

$$\mathbf{H}^{(t)} = \sigma\left(\mathbf{W}\mathbf{X}^{(t)} + \mathbf{U}\mathbf{H}^{(t-1)} + \mathbf{b}\right) \tag{39}$$

where $\mathbf{X}^{(t)} \in \mathbf{R}^{n\times d}$ is the node feature matrix at time step t. After inserting graph convolution, (39) becomes

$$\mathbf{H}^{(t)} = \sigma\left(\text{Gconv}(\mathbf{X}^{(t)}, \mathbf{A}; \mathbf{W}) + \text{Gconv}(\mathbf{H}^{(t-1)}, \mathbf{A}; \mathbf{U}) + \mathbf{b}\right) \tag{40}$$

where Gconv$(\cdot)$ is a graph convolutional layer. Graph convolutional recurrent network (GCRN) [71] combines an LSTM network with ChebNet [21]. Diffusion convolutional RNN (DCRNN) [72] incorporates a proposed diffusion graph convolutional layer [see (18)] into a GRU network. In addition, DCRNN adopts an encoder–decoder framework to predict the future K steps of node values.

Another parallel work uses node-level RNNs and edge-level RNNs to handle different aspects of temporal information. Structural-RNN [73] proposes a recurrent framework to predict node labels at each time step. It comprises two kinds of RNNs, namely, a node-RNN and an edge-RNN. The temporal information of each node and each edge is passed through a node-RNN and an edge-RNN, respectively. To incorporate the spatial information, a node-RNN takes the outputs of edge-RNNs as inputs. Since assuming different RNNs for different nodes and edges significantly increases model complexity, it instead splits nodes and edges into semantic groups. Nodes or edges in the same semantic group share the same RNN model, which saves the computational cost.

RNN-based approaches suffer from time-consuming iterative propagation and gradient explosion/vanishing issues. As alternative solutions, CNN-based approaches tackle spatial–temporal graphs in a nonrecursive manner with the advantages of parallel computing, stable gradients, and low-memory requirements. As illustrated in Fig. 2(d), CNN-based approaches interleave 1-D-CNN layers with graph convolutional layers to learn temporal and spatial dependencies, respectively. Assume that the inputs to an STGNN is a tensor $\mathcal{X} \in R^{T\times n\times d}$, and the 1-D-CNN layer slides over $\mathcal{X}_{[:,i,:]}$ along the time axis to aggregate temporal information for each node, while the graph convolutional layer operates on $\mathcal{X}_{[i,:,:]}$ to aggregate spatial information at each time step. CGCN [74] integrates 1-D convolutional layers with ChebNet [21] or GCN [22] layers. It constructs a spatial–temporal block by stacking a gated 1-D convolutional layer, a graph convolutional layer, and another gated 1-D convolutional layer in sequential order. ST-GCN [75] composes a spatial–temporal block using a 1-D convolutional layer and a PGC layer [see (20)].

Previous methods all use a predefined graph structure. They assume that the predefined graph structure reflects the genuine dependence relationships among nodes. However, with many snapshots of graph data in a spatial–temporal setting, it is possible to learn latent static graph structures automatically from data. To realize this, Graph WaveNet [76] proposes a self-adaptive adjacency matrix to perform graph convolutions. The self-adaptive adjacency matrix is defined as

$$\mathbf{A}_{\text{adp}} = \text{SoftMax}\left(\text{ReLU}\left(\mathbf{E}_1\mathbf{E}_2^T\right)\right) \tag{41}$$

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

Learning latent static spatial dependencies can help researchers discover interpretable and stable correlations among different entities in a network. However, in some circumstances, learning latent dynamic spatial dependencies may further improve model precision. For example, in a traffic network, the travel time between two roads may depend on their current traffic conditions. GaAN [48] employs attention mechanisms to learn dynamic spatial dependencies through an RNN-based approach. An attention function is used to update the edge weight between two connected nodes given their current node inputs. ASTGCN [77] further includes a spatial attention function and a temporal attention function to learn latent dynamic spatial dependencies and temporal dependencies through a CNN-based approach. The common drawback of learning latent spatial dependencies is that it needs to calculate the spatial dependence weight between each pair of nodes, which costs $O(n^2)$.

VIII. Applications

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

A. Data Sets

We mainly sort data sets into four groups, namely, citation networks, biochemical graphs, social networks, and others. In Table VI, we summarize selected benchmark data sets. More details are given in the Supplementary Material A.

B. Evaluation and Open-Source Implementations

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

1) Node Classification: In node classification, most methods follow a standard split of train/valid/test on benchmark data sets, including Cora, Citeseer, Pubmed, PPI, and Reddit. They reported the average accuracy or F1 score on the test data set over multiple runs. A summarization of experimental results of methods can be found in the Supplementary

TABLE VI
SUMMARY OF THE SELECTED BENCHMARK DATA SETS

Category	Data set	Source	# Graphs	# Nodes(Avg.)	# Edges (Avg.)	#Features	# Classes	Citation
Citation Networks	Cora	[117]	1	2708	5429	1433	7	[22], [23], [25], [41], [43], [44], [45] [49], [50], [51], [53], [56], [61], [62]
	Citeseer	[117]	1	3327	4732	3703	6	[22], [41], [43], [45], [50], [51], [53] [56], [61], [62]
	Pubmed	[117]	1	19717	44338	500	3	[18], [22], [25], [41], [43], [44], [45] [49], [51], [53], [55], [56], [61], [62] [70], [95]
	DBLP (v11)	[118]	1	4107340	36624464	-	-	[64], [70], [99]
Bio-chemical Graphs	PPI	[119]	24	56944	818716	50	121	[18], [42], [43], [48], [45], [50], [55] [56], [58], [64]
	NCI-1	[120]	4110	29.87	32.30	37	2	[25], [26], [46], [52], [57], [96], [98]
	MUTAG	[121]	188	17.93	19.79	7	2	[25], [26], [46], [52], [57], [96]
	D&D	[122]	1178	284.31	715.65	82	2	[26], [46], [52], [54], [96], [98]
	PROTEIN	[123]	1113	39.06	72.81	4	2	[26], [46], [52], [54], [57]
	PTC	[124]	344	25.5	-	19	2	[25], [26], [46], [52], [57]
	QM9	[125]	133885	-	-	-	-	[27], [69]
	Alchemy	[126]	119487	-	-	-	-	-
Social Networks	Reddit	[42]	1	232965	11606919	602	41	[42], [48], [49], [50], [51], [56]
	BlogCatalog	[127]	1	10312	333983	-	39	[18], [55], [60], [64]
Others	MNIST	[128]	70000	784	-	1	10	[19], [23], [21], [44], [96]
	METR-LA	[129]	1	207	1515	2	-	[48], [72], [76]
	Nell	[130]	1	65755	266144	61278	210	[22], [41], [50]

Material B. It should be noted that these results do not necessarily represent a rigorous comparison. Shchur *et al.* [131] identified two pitfalls in evaluating the performance GNNs on node classification. First, using the same train/valid/test split throughout all experiments underestimates the generalization error. Second, different methods employed different training techniques, such as hyperparameter tuning, parameter initialization, learning rate decay, and early stopping. For a relatively fair comparison, we refer the readers to [131].

2) Graph Classification: In graph classification, researchers often adopt tenfold cross validation (cv) for model evaluation. However, as pointed out in [132], the experimental settings are ambiguous and not unified across different works. In particular, [132] raises the concern of the correct usage of data splits for model selection versus model assessment. An often encountered problem is that the external test set of each fold is used both for model selection and risk assessment. Reference [132] compares GNNs in a standardized and uniform evaluation framework. They apply an external tenfold CV to get an estimate of the generalization performance of a model and an inner holdout technique with a 90%/10% training/validation split for model selection. An alternative procedure would be a double-cv method, which uses an external k-fold cv for model assessment and an inner k-fold cv for model selection. We refer the readers to [132] for a detailed and rigorous comparison of GNN methods for graph classification.

3) Open-Source Implementations: These facilitate the work of baseline experiments in deep learning research. In the Supplementary Material C, we provide the hyperlinks of the open-source implementations of the GNN models reviewed in this article. Noticeably, Fey *et al.* [92] published a geometric learning library in PyTorch named PyTorch Geometric,[4] which implements many GNNs. Most recently, the deep graph library (DGL)[5] [133] is released, which provides a fast implementation of many GNNs on top of popular deep learning platforms, such as PyTorch and MXNet.

C. Practical Applications

GNNs have many applications across different tasks and domains. Despite general tasks that can be handled by each category of GNNs directly, including node classification, graph classification, network embedding, graph generation, and spatial–temporal graph forecasting, other general graph-related tasks, such as node clustering [134], link prediction [135], and graph partitioning [136], can also be addressed by GNNs. We detail some applications based on the following research domains.

1) Computer Vision: Applications of GNNs in computer vision include scene graph generation, point clouds' classification, and action recognition.

Recognizing semantic relationships between objects facilitates the understanding of the meaning behind a visual scene. Scene graph generation models aim to parse an image into a semantic graph that consists of objects and their semantic relationships [137]–[139]. Another application inverses the process by generating realistic images given scene graphs [140]. As natural language can be parsed as semantic graphs, where each word represents an object, it is a promising solution to synthesize images given textual descriptions.

Classifying and segmenting points' clouds enable LiDAR devices to "see" the surrounding environment. A point cloud is a set of 3-D points recorded by LiDAR scans. References [141]–[143] convert point clouds into k-nearest neighbor graphs or superpoint graphs and use ConvGNNs to explore the topological structure.

Identifying human actions contained in videos facilitates a better understanding of video content from a machine aspect.

[4]https://github.com/rusty1s/pytorch_geometric

[5]https://www.dgl.ai/

Some solutions detect the locations of human joints in video clips. Human joints that are linked by skeletons naturally form a graph. Given the time series of human joint locations, [73] and [75] apply STGNNs to learn human action patterns.

Moreover, the number of applicable directions of GNNs in computer vision is still growing. It includes human–object interaction [144], few-shot image classification [145]–[147], semantic segmentation [148], [149], visual reasoning [150], and question answering [151].

2) Natural Language Processing: A common application of GNNs in natural language processing is the text classification. GNNs utilize the interrelations of documents or words to infer document labels [22], [42], [43].

Despite the fact that natural language data exhibit a sequential order, they may also contain an internal graph structure, such as a syntactic dependency tree. A syntactic dependency tree defines the syntactic relations among words in a sentence. Marcheggiani and Titov [152] propose the Syntactic GCN that runs on top of a CNN/RNN sentence encoder. The Syntactic GCN aggregates hidden word representations based on the syntactic dependency tree of a sentence. Bastings *et al.* [153] apply the Syntactic GCN to the task of neural machine translation. Marcheggiani *et al.* [154] further adopt the same model as in [153] to handle the semantic dependency graph of a sentence.

Graph-to-sequence learning learns to generate sentences with the same meaning given a semantic graph of abstract words (known as abstract meaning representation). Song *et al.* [155] propose a graph-LSTM to encode graph-level semantic information. Beck *et al.* [156] apply a GGNN [17] to graph-to-sequence learning and neural machine translation. The inverse task is sequence-to-graph learning. Generating a semantic or knowledge graph given a sentence is very useful in knowledge discovery [157], [158].

3) Traffic: Accurately forecasting traffic speed, volume, or the density of roads in traffic networks is fundamentally important in a smart transportation system. Zhang *et al.* [48], Li *et al.* [72], and Yu *et al.* [74] address the traffic prediction problem using STGNNs. They consider the traffic network as a spatial–temporal graph, where the nodes are sensors installed on roads, the edges are measured by the distance between pairs of nodes, and each node has the average traffic speed within a window as dynamic input features. Another industrial-level application is taxi-demand prediction. Given historical taxi demands, location information, weather data, and event features, Yao *et al.* [159] incorporate LSTM, CNN, and network embeddings trained by LINE [160] to form a joint representation for each location to predict the number of taxis demanded for a location within a time interval.

4) Recommender Systems: Graph-based recommender systems take items and users as nodes. By leveraging the relations between items and items, users and users, users and items, as well as content information, graph-based recommender systems are able to produce high-quality recommendations. The key to a recommender system is to score the importance of an item to a user. As a result, it can be cast as a link prediction problem. To predict the missing links between users and items, Berg *et al.* [161] and Ying *et al.* [162] propose a GAE that uses ConvGNNs as encoders. Monti *et al.* [163] combine RNNs with graph convolutions to learn the underlying process that generates the known ratings.

5) Chemistry: In the field of chemistry, researchers apply GNNs to study the graph structure of molecules/compounds. In a molecule/compound graph, atoms are considered as nodes, and chemical bonds are treated as edges. Node classification, graph classification, and graph generation are the three main tasks targeting molecular/compound graphs in order to learn molecular fingerprints [85], [86], predict molecular properties [27], infer protein interfaces [164], and synthesize chemical compounds [65], [69], [165].

6) Others: The application of GNNs is not limited to the aforementioned domains and tasks. There have been explorations of applying GNNs to a variety of problems, such as program verification [17], program reasoning [166], social influence prediction [167], adversarial attacks prevention [168], electrical health records modeling [169], [170], brain networks [171], event detection [172], and combinatorial optimization [173].

IX. Future Directions

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

A. Model depth

The success of deep learning lies in deep neural architectures [174]. However, Li *et al.* [53] show that the performance of a ConvGNN drops dramatically with an increase in the number of graph convolutional layers. As graph convolutions push representations of adjacent nodes closer to each other, in theory, with an infinite number of graph convolutional layers, all nodes' representations will converge to a single point [53]. This raises the question of whether going deep is still a good strategy for learning graph data.

B. Scalability Tradeoff

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

C. Heterogenity

The majority of current GNNs assume homogeneous graphs. It is difficult to directly apply current GNNs to heterogeneous graphs, which may contain different types of nodes and edges or different forms of node and edge inputs, such as images and text. Therefore, new methods should be developed to handle heterogeneous graphs.

D. Dynamicity

Graphs are, in nature, dynamic in a way that nodes or edges may appear or disappear and that node/edge inputs may change

time by time. New graph convolutions are needed to adapt to the dynamicity of graphs. Although the dynamicity of graphs can be partly addressed by STGNNs, few of them consider how to perform graph convolutions in the case of dynamic spatial relations.

X. Conclusion

In this article, we conduct a comprehensive overview of GNNs. We provide a taxonomy that groups GNNs into four categories: RecGNNs, ConvGNNs, GAEs, and STGNNs. We provide a thorough review, comparisons, and summarizations of the methods within or between categories. Then, we introduce a wide range of applications of GNNs. Data sets, open-source codes, and model assessment for GNNs are summarized. Finally, we suggest four future directions for GNNs.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 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 *Proc. NIPS*, 2015, pp. 91–99.

[3] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2015, pp. 1412–1421.

[4] Y. Wu *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016, *arXiv:1609.08144*. [Online]. Available: http://arxiv.org/abs/1609.08144

[5] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[6] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," in *The Handbook of Brain Theory and Neural Networks*, vol. 3361, no. 10. Cambridge, MA, USA: MIT Press, 1995.

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

[8] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, no. 12, pp. 3371–3408, Dec. 2010.

[9] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Van der Gheynst, "Geometric deep learning: Going beyond Euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, Jul. 2017.

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

[11] P. W. Battaglia *et al.*, "Relational inductive biases, deep learning, and graph networks," 2018, *arXiv:1806.01261*. [Online]. Available: http://arxiv.org/abs/1806.01261

[12] J. Boaz Lee, R. A. Rossi, S. Kim, N. K. Ahmed, and E. Koh, "Attention models in graphs: A survey," 2018, *arXiv:1807.07984*. [Online]. Available: http://arxiv.org/abs/1807.07984

[13] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 714–735, May 1997.

[14] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, vol. 2, Aug. 2005, pp. 729–734.

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

[16] C. Gallicchio and A. Micheli, "Graph echo state networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2010, pp. 1–8.

[17] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *Proc. ICLR*, 2015, pp. 1–20.

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

[19] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *Proc. ICLR*, 2014, pp. 1–14.

[20] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," 2015, *arXiv:1506.05163*. [Online]. Available: http://arxiv.org/abs/1506.05163

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

[22] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. ICLR*, 2017, pp. 1–14.

[23] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "CayleyNets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Trans. Signal Process.*, vol. 67, no. 1, pp. 97–109, Jan. 2019.

[24] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Trans. Neural Netw.*, vol. 20, no. 3, pp. 498–511, Mar. 2009.

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

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

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

[28] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 5, pp. 833–852, May 2019.

[29] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Network representation learning: A survey," *IEEE Trans. Big Data*, vol. 6, no. 1, pp. 3–28, Mar. 2020.

[30] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 9, pp. 1616–1637, Sep. 2018.

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

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

[33] X. Shen, S. Pan, W. Liu, Y.-S. Ong, and Q.-S. Sun, "Discrete network embedding," in *Proc. 27th Int. Joint Conf. Artif. Intell.*, Jul. 2018, pp. 3549–3555.

[34] H. Yang, S. Pan, P. Zhang, L. Chen, D. Lian, and C. Zhang, "Binarized attributed network embedding," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2018, pp. 1476–1481.

[35] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: Online learning of social representations," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2014, pp. 701–710.

[36] S. V. N. Vishwanathan, N. N. Schraudolph, I. R. Kondor, and K. M. Borgwardt, "Graph kernels," *J. Mach. Learn. Res.*, vol. 11, pp. 1201–1242, Mar. 2010.

[37] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler–Lehman graph kernels," *J. Mach. Learn. Res.*, vol. 12, pp. 2539–2561, Sep. 2011.

[38] N. Navarin and A. Sperduti, "Approximated neighbours MinHash graph node kernel," in *Proc. ESANN*, 2017, pp. 1–6.

[39] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," 2019, *arXiv:1903.11835*. [Online]. Available: http://arxiv.org/abs/1903.11835

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

[41] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *Proc. World Wide Web Conf. World Wide Web (WWW)*, 2018, pp. 499–508.

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

[43] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. ICLR*, 2017, pp. 1–12.

[44] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5115–5124.

[45] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2018, pp. 1416–1424.

[46] D. V. Tran, N. Navarin, and A. Sperduti, "On filter size in graph convolutional networks," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Nov. 2018, pp. 1534–1541.

[47] D. Bacciu, F. Errica, and A. Micheli, "Contextual graph Markov model: A deep and generative approach to graph processing," in *Proc. ICML*, 2018, pp. 1–10.

[48] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, "GaAN: Gated attention networks for learning on large and spatiotemporal graphs," in *Proc. UAI*, 2018, pp. 1–10.

[49] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," in *Proc. ICLR*, 2018, pp. 1–15.

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

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

[52] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proc. AAAI*, 2018, pp. 1–8.

[53] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *Proc. AAAI*, 2018, pp. 1–8.

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

[55] Z. Liu *et al.*, "GeniePath: Graph neural networks with adaptive receptive paths," in *Proc. AAAI Conf. Artif. Intell.*, Jul. 2019, pp. 4424–4431.

[56] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *Proc. ICLR*, 2019, pp. 1–17.

[57] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks," in *Proc. ICLR*, 2019, pp. 1–17.

[58] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2019, pp. 257–266.

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

[60] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2016, pp. 1225–1234.

[61] T. N. Kipf and M. Welling, "Variational graph auto-encoders," in *Proc. NIPS Workshop Bayesian Deep Learn.*, 2016, pp. 1–3.

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

[63] K. Tu, P. Cui, X. Wang, P. S. Yu, and W. Zhu, "Deep recursive network embedding with regular equivalence," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2018, pp. 2357–2366.

[64] W. Yu *et al.*, "Learning deep network representations with adversarially regularized autoencoders," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2018, pp. 2663–2671.

[65] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," in *Proc. ICML*, 2018, pp. 1–21.

[66] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "GraphRNN: A deep generative model for graphs," in *Proc. ICML*, 2018, pp. 1–12.

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

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

[69] N. De Cao and T. Kipf, "MolGAN: An implicit generative model for small molecular graphs," *ICML Workshop Theor. Found. Appl. Deep Generative Models*, 2018, pp. 1–11.

[70] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "NetGAN: Generating graphs via random walks," in *Proc. ICML*, 2018, pp. 1–16.

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

[72] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *Proc. ICLR*, 2018, pp. 1–16.

[73] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-RNN: Deep learning on spatio-temporal graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 5308–5317.

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

[75] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Proc. AAAI*, 2018, pp. 1–9.

[76] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph WaveNet for deep spatial-temporal graph modeling," in *Proc. IJCAI*, Aug. 2019, pp. 1–7.

[77] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *Proc. AAAI*, 2019, pp. 922–929.

[78] S. Pan, J. Wu, X. Zhu, C. Zhang, and P. S. Yu, "Joint structure feature exploration and regularization for multi-task graph classification," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 3, pp. 715–728, Mar. 2016.

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

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

[81] K. Cho *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1724–1734.

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

[83] A. Sandryhaila and J. M. F. Moura, "Discrete signal processing on graphs," *IEEE Trans. Signal Process.*, vol. 61, no. 7, pp. 1644–1656, Apr. 2013.

[84] S. Chen, R. Varma, A. Sandryhaila, and J. Kovacevic, "Discrete signal processing on graphs: Sampling theory," *IEEE Trans. Signal Process.*, vol. 63, no. 24, pp. 6510–6523, Dec. 2015.

[85] D. K. Duvenaud *et al.*, "Convolutional networks on graphs for learning molecular fingerprints," in *Proc. NIPS*, 2015, pp. 2224–2232.

[86] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, "Molecular graph convolutions: Moving beyond fingerprints," *J. Comput.-Aided Mol. Des.*, vol. 30, no. 8, pp. 595–608, Aug. 2016.

[87] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko, "Quantum-chemical insights from deep tensor neural networks," *Nature Commun.*, vol. 8, no. 1, p. 13890, Jan. 2017.

[88] J. B. Lee, R. Rossi, and X. Kong, "Graph classification using structural attention," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2018, pp. 1666–1674.

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

[90] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on Riemannian manifolds," in *Proc. IEEE Int. Conf. Comput. Vis. Workshop (ICCVW)*, Dec. 2015, pp. 37–45.

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

[92] M. Fey, J. E. Lenssen, F. Weichert, and H. Müller, "SplineCNN: Fast geometric deep learning with continuous B-spline kernels," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 869–877.

[93] 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.

[94] B. L. Douglas, "The Weisfeiler–Lehman method and graph isomorphism testing," 2011, *arXiv:1101.5211*. [Online]. Available: http://arxiv.org/abs/1101.5211

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

[96] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 3693–3702.

[97] T. Derr, Y. Ma, and J. Tang, "Signed graph convolutional networks," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2018, pp. 929–934.

[98] F. P. Such *et al.*, "Robust spatial filtering with graph convolutional neural networks," *IEEE J. Sel. Topics Signal Process.*, vol. 11, no. 6, pp. 884–896, Sep. 2017.

[99] X. Wang *et al.*, "Heterogeneous graph attention network," in *Proc. World Wide Web Conf. (WWW)*, 2019, pp. 2022–2032.

[100] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 11, pp. 1944–1957, Nov. 2007.

[101] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," in *Proc. ICLR*, 2016, pp. 1–11.

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

[103] F. Scarselli, A. C. Tsoi, and M. Hagenbuchner, "The Vapnik–Chervonenkis dimension of graph and recursive neural networks," *Neural Netw.*, vol. 108, pp. 248–259, Dec. 2018.

[104] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman, "Invariant and equivariant graph networks," in *ICLR*, 2019, pp. 1–14.

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

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

[107] F. Scarselli, M. Gori, A. Chung Tsoi, M. Hagenbuchner, and G. Monfardini, "Computational capabilities of graph neural networks," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 81–102, Jan. 2009.

[108] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proc. 25th Int. Conf. Mach. Learn. (ICML)*, 2008, pp. 1096–1103.

[109] S. Pan, R. Hu, S.-F. Fung, G. Long, J. Jiang, and C. Zhang, "Learning graph embedding with adversarial training methods," *IEEE Trans. Cybern.*, early access, Sep. 2, 2019, doi: 10.1109/TCYB.2019.2932096.

[110] I. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. NIPS*, 2014, pp. 2672–2680.

[111] R. Gómez-Bombarelli *et al.*, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS Central Sci.*, vol. 4, no. 2, pp. 268–276, Jan. 2018.

[112] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar variational autoencoder," in *Proc. ICML*, 2017, pp. 1945–1954.

[113] H. Dai, Y. Tian, B. Dai, S. Skiena, and L. Song, "Syntax-directed variational autoencoder for structured data," in *Proc. ICLR*, 2018, pp. 1–17.

[114] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *ESWC*. Cham, Switzerland: Springer, 2018, pp. 593–607.

[115] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of Wasserstein GANs," in *Proc. NIPS*, 2017, pp. 5767–5777.

[116] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," 2017, *arXiv:1701.07875*. [Online]. Available: http://arxiv.org/abs/1701.07875

[117] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Mag.*, vol. 29, no. 3, p. 93, Sep. 2008.

[118] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer: Extraction and mining of academic social networks," in *Proc. KDD*, 2008, pp. 990–998.

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

[120] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *Knowl. Inf. Syst.*, vol. 14, no. 3, pp. 347–375, 2008.

[121] A. K. Debnath, R. L. L. 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," *J. Medicinal Chem.*, vol. 34, no. 2, pp. 786–797, Feb. 1991.

[122] P. D. Dobson and A. J. Doig, "Distinguishing enzyme structures from non-enzymes without alignments," *J. Mol. Biol.*, vol. 330, no. 4, pp. 771–783, Jul. 2003.

[123] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. 1, pp. i47–i56, Jun. 2005.

[124] H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, and C. Helma, "Statistical evaluation of the predictive toxicology challenge 2000–2001," *Bioinformatics*, vol. 19, no. 10, pp. 1183–1193, Jul. 2003.

[125] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld, "Quantum chemistry structures and properties of 134 kilo molecules," *Sci. Data*, vol. 1, no. 1, Aug. 2014, Art. no. 140022.

[126] G. Chen *et al.*, "Alchemy: A quantum chemistry dataset for benchmarking AI models," 2019, *arXiv:1906.09427*. [Online]. Available: http://arxiv.org/abs/1906.09427

[127] L. Tang and H. Liu, "Relational learning via latent social dimensions," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2009, pp. 817–826.

[128] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[129] H. V. Jagadish *et al.*, "Big data and its technical challenges," *Commun. ACM*, vol. 57, no. 7, pp. 86–94, Jul. 2014.

[130] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, Jr., and T. M. Mitchell, "Toward an architecture for never-ending language learning," in *Proc. AAAI*, 2010, pp. 1306–1313.

[131] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," in *Proc. NeurIPS workshop*, 2018, pp. 1–11.

[132] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," in *Proc. ICLR*, 2020, pp. 1–15. [Online]. Available: https://openreview.net/forum?id=HygDF6NFPB

[133] M. Wang *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs," in *Proc. ICLR Workshop Represent. Learn. Graphs Manifolds*, 2019, pp. 1–7.

[134] C. Wang, S. Pan, G. Long, X. Zhu, and J. Jiang, "MGAE: Marginalized graph autoencoder for graph clustering," in *Proc. CIKM*, 2017, pp. 889–898.

[135] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Proc. NeurIPS*, 2018, pp. 5165–5175.

[136] T. Kawamoto, M. Tsubaki, and T. Obuchi, "Mean-field theory of graph neural networks in graph partitioning," in *Proc. NeurIPS*, 2018, pp. 4362–4372.

[137] D. Xu, Y. Zhu, C. B. Choy, and L. Fei-Fei, "Scene graph generation by iterative message passing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5410–5419.

[138] J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh, "Graph R-CNN for scene graph generation," in *Proc. ECCV*. Springer, 2018, pp. 690–706.

[139] Y. Li, W. Ouyang, B. Zhou, J. Shi, C. Zhang, and X. Wang, "Factorizable net: An efficient subgraph-based framework for scene graph generation," in *Proc. ECCV*. Springer, 2018, pp. 346–363.

[140] J. Johnson, A. Gupta, and L. Fei-Fei, "Image generation from scene graphs," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 1219–1228.

[141] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM Trans. Graph.*, vol. 38, no. 5, pp. 1–12, Oct. 2019.

[142] L. Landrieu and M. Simonovsky, "Large-scale point cloud semantic segmentation with superpoint graphs," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4558–4567.

[143] G. Te, W. Hu, A. Zheng, and Z. Guo, "RGCNN: Regularized graph CNN for point cloud segmentation," in *Proc. ACM Multimedia Conf. Multimedia Conf. (MM)*, 2018, pp. 746–754.

[144] S. Qi, W. Wang, B. Jia, J. Shen, and S.-C. Zhu, "Learning human-object interactions by graph parsing neural networks," in *Proc. ECCV*. Springer, 2018, pp. 401–417.

[145] V. G. Satorras and J. B. Estrach, "Few-shot learning with graph neural networks," in *Proc. ICLR*, 2018, pp. 1–13.

[146] M. Guo, E. Chou, D.-A. Huang, S. Song, S. Yeung, and L. Fei-Fei, "Neural graph matching networks for fewshot 3D action recognition," in *Proc. ECCV*. Springer, 2018, pp. 673–689.

[147] L. Liu, T. Zhou, G. Long, J. Jiang, L. Yao, and C. Zhang, "Prototype propagation networks (PPN) for weakly-supervised few-shot learning on category graph," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 1–8.

[148] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun, "3D graph neural networks for RGBD semantic segmentation," in *Proc. CVPR*, Oct. 2017, pp. 5199–5208.

[149] L. Yi, H. Su, X. Guo, and L. Guibas, "SyncSpecCNN: Synchronized spectral CNN for 3D shape segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6584–6592.

[150] X. Chen, L.-J. Li, L. Fei-Fei, and A. Gupta, "Iterative visual reasoning beyond convolutions," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7239–7248.

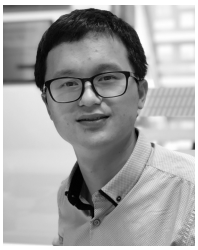
[151] M. Narasimhan, S. Lazebnik, and A. Schwing, "Out of the box: Reasoning with graph convolution nets for factual visual question answering," in *Proc. NeurIPS*, 2018, pp. 2655–2666.

[152] D. Marcheggiani and I. Titov, "Encoding sentences with graph convolutional networks for semantic role labeling," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2017, pp. 1506–1515.

[153] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, and K. Simaan, "Graph convolutional encoders for syntax-aware neural machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2017, pp. 1957–1967.
[154] D. Marcheggiani, J. Bastings, and I. Titov, "Exploiting semantics in neural machine translation with graph convolutional networks," in *Proc. NAACL*, 2018, pp. 1–8.
[155] L. Song, Y. Zhang, Z. Wang, and D. Gildea, "A graph-to-sequence model for AMR-to-text generation," in *Proc. ACL*, 2018, pp. 1–11.
[156] D. Beck, G. Haffari, and T. Cohn, "Graph-to-sequence learning using gated graph neural networks," in *Proc. ACL*, 2018, pp. 1–13.
[157] D. D. Johnson, "Learning graphical state transitions," in *Proc. ICLR*, Mar. 2016, pp. 1–19.
[158] B. Chen, L. Sun, and X. Han, "Sequence-to-action: End-to-end semantic graph generation for semantic parsing," in *Proc. ACL*, 2018, pp. 766–777.
[159] H. Yao *et al.*, "Deep multi-view spatial-temporal network for taxi demand prediction," in *Proc. AAAI*, 2018, pp. 2588–2595.
[160] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proc. WWW*, 2015, pp. 1067–1077.
[161] R. van den Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," 2017, *arXiv:1706.02263*. [Online]. Available: https://arxiv.org/abs/1706.02263
[162] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for Web-scale recommender systems," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2018, pp. 974–983.
[163] F. Monti, M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," in *Proc. NIPS*, 2017, pp. 3697–3707.
[164] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein interface prediction using graph convolutional networks," in *Proc. NIPS*, 2017, pp. 6530–6539.
[165] J. You, B. Liu, R. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *Proc. NeurIPS*, 2018, pp. 6410–6421.
[166] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," in *Proc. ICLR*, 2017, pp. 1–17.
[167] J. Qiu, J. Tang, H. Ma, Y. Dong, K. Wang, and J. Tang, "DeepInf: Social influence prediction with deep learning," in *Proc. KDD*, 2018, pp. 2110–2119.
[168] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, Aug. 2019, pp. 2847–2856.
[169] E. Choi, M. T. Bahadori, L. Song, W. F. Stewart, and J. Sun, "GRAM: Graph-based attention model for healthcare representation learning," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2017, pp. 787–795.
[170] E. Choi, C. Xiao, W. Stewart, and J. Sun, "Mime: Multilevel medical embedding of electronic health records for predictive healthcare," in *Proc. NeurIPS*, 2018, pp. 4548–4558.
[171] J. Kawahara *et al.*, "BrainNetCNN: Convolutional neural networks for brain networks; towards predicting neurodevelopment," *NeuroImage*, vol. 146, pp. 1038–1049, Feb. 2017.
[172] T. H. Nguyen and R. Grishman, "Graph convolutional networks with argument-aware pooling for event detection," in *Proc. AAAI*, 2018, pp. 5900–5907.
[173] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *Proc. NeurIPS*, 2018, pp. 536–545.
[174] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

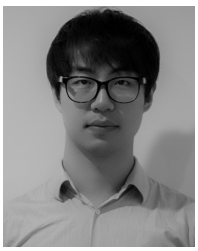
Zonghan Wu received the B.S. degree in systems science from the University of Shanghai for Science and Technology, Shanghai, China, in 2014, and the M.S. degree in statistics from Linköping University, Linköping, Sweden, in 2016. He is currently pursuing the Ph.D. degree in computer science with the University of Technology Sydney (UTS), Ultimo, NSW, Australia.

His research concentrates on data mining, machine learning, and deep learning on graphs.



Shirui Pan (Member, IEEE) received the Ph.D. degree in computer science from the University of Technology Sydney (UTS), Ultimo, NSW, Australia.

He was a Lecturer with the School of Software, UTS. He is currently a Lecturer with the Faculty of Information Technology, Monash University, Clayton, VIC, Australia. He has published over 60 research articles in top-tier journals and conferences, including the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS (TNNLS), the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING (TKDE), the IEEE TRANSACTIONS ON CYBERNETICS (TCYB), the IEEE International Conference on Data Engineering (ICDE), the AAAI Conference on Artificial Intelligence (AAAI), the International Joint Conferences on Artificial Intelligence (IJCAI), and the IEEE International Conference on Data Mining (ICDM). His research interests include data mining and machine learning.



Fengwen Chen received the B.S. degree in computer science (software engineering) from Arizona State University, Tempe, AZ, USA. He is currently pursuing the Ph.D. degree in computer science with the University of Technology Sydney (UTS), Ultimo, NSW, Australia.

His research concentrates on data mining and deep learning on graphs.

Guodong Long was born in China. He received the Ph.D. degree in computer science from the University of Technology Sydney, Ultimo, NSW, Australia, in 2014.

He is currently a Senior Lecturer and a Core Member with the Centre for Artificial Intelligence (CAI), Faculty of Engineering and Information Technology, University of Technology Sydney. His research focuses on machine learning, data mining, and cloud computing.

Chengqi Zhang (Senior Member, IEEE) received the Ph.D. degree from The University of Queensland, Brisbane, QLD, Australia, in 1991, and the D.Sc. (higher doctorate) degree from Deakin University, Geelong, VIC, Australia, in 2002.

Since December 2001, he has been a Professor of information technology with the University of Technology Sydney (UTS), Ultimo, NSW, Australia, where he was the Director of the UTS Priority Investment Research Centre for Quantum Computation and Intelligent Systems from 2008 to 2016. His research interests mainly focus on data mining and its applications.

Dr. Zhang is a fellow of the Australian Computer Society. He was the General Co-Chair of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) 2015 in Sydney and the Local Arrangements Chair of the International Joint Conferences on Artificial Intelligence (IJCAI)-2017 in Melbourne.

Philip S. Yu (Life Fellow, IEEE) received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, USA.

He is currently a Distinguished Professor of computer science with the University of Illinois at Chicago, Chicago, IL, USA, where he is also the Wexler Chair in Information Technology. He has published more than 830 articles in refereed journals and conferences. He holds or has applied for more than 300 U.S. patents. His research interests include big data, data mining, data streams, databases, and privacy.

Dr. Yu is a fellow of the ACM. He received the ACM SIGKDD 2016 Innovation Award, the Research Contributions Award from the IEEE International Conference on Data Mining in 2003, and the Technical Achievement Award from the IEEE Computer Society in 2013.