Deep Learning on Graphs: A Survey

Ziwei Zhang, Peng Cui and Wenwu Zhu, *Fellow, IEEE*

Abstract—Deep learning has been shown successful in a number of domains, ranging from acoustics, images to natural language processing. However, applying deep learning to the ubiquitous graph data is non-trivial because of the unique characteristics of graphs. Recently, a significant amount of research efforts have been devoted to this area, greatly advancing graph analyzing techniques. In this survey, we comprehensively review different kinds of deep learning methods applied to graphs. We divide existing methods into five categories based on their model architectures: Graph Recurrent Neural Networks, Graph Convolutional Networks, Graph Autoencoders, Graph Reinforcement Learning, and Graph Adversarial Methods. We then provide a comprehensive overview of these methods in a systematic manner mainly following their history of developments. We also analyze the differences and compositionality of different architectures. Finally, we briefly outline their applications and discuss potential future directions.

Index Terms—Graph Data, Deep Learning, Graph Neural Network, Graph Convolutional Network, Graph Autoencoder.

1 Introduction

In the last decade, deep learning has been a "crown jewel" in artificial intelligence and machine learning [1], showing superior performance in acoustics [2], images [3], and natural language processing [4], etc. The expressive power of deep learning to extract complex patterns underlying data has been well recognized. On the other hand, graphs[1] are ubiquitous in the real world, representing objects and their relationships such as social networks, e-commerce networks, biology networks, and traffic networks, etc. Graphs are also known to have complicated structures that contain rich underlying values [5]. As a result, how to utilize deep learning methods for graph data analysis has attracted considerable research attention in the past few years. This problem is non-trivial because several challenges exist for applying traditional deep learning architectures to graphs:

- **Irregular structures of graphs**. Unlike images, audio, and texts which have a clear grid structure, graphs have irregular structures, making it hard to generalize some basic mathematical operations to graphs [6]. For example, it is not straightforward to define convolution and pooling operation for graph data, which are the fundamental operations in Convolutional Neural Networks (CNNs). This is often referred to as the geometric deep learning problem [7].
- **Heterogeneity of graphs**. The graph itself can be complicated with diverse types and properties. For example, graphs can be heterogeneous or homogenous, weighted or unweighted, and signed or unsigned. In addition, the tasks for graphs also vary greatly, ranging from node-focused problems such as node classification and link prediction to graph-focused problems such as graph classification and graph generation. The diverse types, properties, and tasks require different model architectures to tackle specific problems.
- **Large-scale graphs**. In the big-data era, real graphs can easily have millions or billions of nodes and edges, such as social networks or e-commerce networks [8]. As a result, how to design scalable models, preferably having a linear time complexity with respect to the graph size, becomes a key problem.
- **Incorporating interdisciplinary knowledge**. Graphs are often connected with other disciplines, such as biology, chemistry or social sciences. The interdisciplinary provides both opportunities and challenges: domain knowledge can be leveraged to solve specific problems, but integrating domain knowledge could complicate the model design. For example, in generating molecular graphs, the objective function and chemical constraints are often non-differentiable, so gradient-based training methods cannot be easily applied.

To tackle these challenges, tremendous effort has been made towards this area, resulting in a rich literature of related papers and methods. The architecture adopted also varies greatly, ranging from supervised to unsupervised, convolutional to recursive. However, to the best of our knowledge, little attention has been paid to systematically summarize the differences and connections between these diverse methods.

In this paper, we try to fill this gap by comprehensive reviewing deep learning methods on graphs. Specifically, as shown in Figure 1, we divide the existing methods into five categories based on their model architectures: Graph Recurrent Neural Networks (Graph RNNs), Graph Convolutional Networks (GCNs), Graph Autoencoders (GAEs), Graph Reinforcement Learning (Graph RL), and Graph Adversarial Methods. We summarize some main characteristics of these categories in Table 1 based on the following high-level distinctions. Graph RNNs capture recursive and sequential patterns of graphs by modeling and states in either the node-level or the graph-level. GCNs define convolution and readout operations on irregular graph structures to capture common local and global structural patterns. GAEs assume low-rank graph structures and adopt unsupervised methods for node representation learning. Graph RL defines graph-based actions and rewards to get feedbacks on graph tasks while following constraints. Graph Adversarial Methods adopt adversarial training to enhance the

• *Z. Zhang, P. Cui, and W. Zhu are with the Department of Computer Science and Technology, Tsinghua University, Beijing, China. E-mail: zw-zhang16@mails.tsinghua.edu.cn, cuip@tsinghua.edu.cn, wwzhu@tsinghua.edu.cn. P. Cui and W. Zhu are corresponding authors.*

1. Graphs are also called networks such as in social networks. In this paper, we use two terms interchangeably.

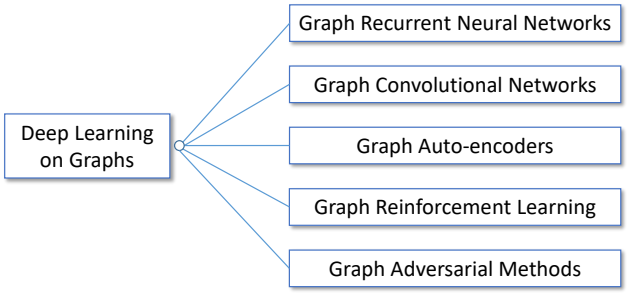

Fig. 1. The categorization of deep learning methods on graphs. We divide existing methods into five categories: Graph Recurrent Neural Networks, Graph Convolutional Networks, Graph Autoencoders, Graph Reinforcement Learning, and Graph Adversarial Methods.

TABLE 1
Some Main Distinctions of Deep Learning Methods on Graphs

Category	Basic Assumptions/Aims	Main Functions
Graph Recurrent Neural Networks	Recursive and sequential patterns of graphs	Definition of states for nodes or graphs
Graph Convolutional Networks	Common local and global structural patterns of graphs	Graph convolution and readout operations
Graph Autoencoders	Low-rank structures of graphs	Unsupervised node representation learning
Graph Reinforcement Learning	Feedbacks and constraints of graph tasks	Graph-based actions and rewards
Graph Adversarial Methods	The generalization ability and robustness of graph models	Graph adversarial training and attacks

generalization ability of graph models and test their robustness by adversarial attacks.

In the following sections, we provide a comprehensive overview of these methods in detail, mainly following their history of developments and how these methods solve the challenges of graphs. We also analyze the differences between these models and how to composite different architectures. Finally, we briefly outline their applications, introduce several open libraries, and discuss potential future directions. We also provide a collection of source codes of various methods discussed in the paper and summarize some common applications in the appendix.

Related works. There are several surveys that are related to our paper. Bronstein et al. [7] summarize some early GCN methods as well as CNNs on manifolds and study them comprehensively through geometric deep learning. Battaglia et al. [9] summarize how to use GNNs and GCNs for relational reasoning using a unified framework called graph networks, Lee et al. [10] review the attention models for graphs, Zhang et al. [11] summarize some GCNs, and Sun et al. [12] briefly survey adversarial attacks on graphs. We differ from these works in that we systematically and comprehensively review different deep learning architectures on graphs rather than focusing on one specific branch. Concurrent to our work, Zhou et al. [13] and Wu et al. [14] also survey this field with different focuses and categorizations. Specifically, both their works focus on GCNs without considering graph reinforcement learning and graph adversarial methods, which are covered in our paper.

Another closely related topic is network embedding, trying to embed nodes into a low-dimensional vector space [15]–[17]. The main distinction between network embedding and our paper is that we focus on how different deep learning models can be applied to graphs, and network embedding can be recognized as a concrete example using some of these models (and they use non-deep-learning methods as well).

The rest of this paper is organized as follows. In Section 2, we introduce notations and preliminaries. Then, we review Graph RNNs, GCNs, GAEs, Graph RL, and Graph Adversarial Methods in Section 3 to Section 7, respectively. We conclude with a discussion in Section 8.

TABLE 2
Table of Commonly Used Notations

$G = (V, E)$	A graph
N, M	The number of nodes and edges
$V = \{v_1, ..., v_N\}$	The set of nodes
$\mathbf{F}^V, \mathbf{F}^E$	Attributes/features for nodes and edges
$\mathbf{A}$	The adjacency matrix
$\mathbf{D}(i,i) = \sum_j \mathbf{A}(i,j)$	The diagonal degree matrix
$\mathbf{L} = \mathbf{D} - \mathbf{A}$	The Laplacian matrix
$\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T = \mathbf{L}$	The eigen-decomposition of $\mathbf{L}$
$\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$	The transition matrix
$\mathcal{N}_k(i), \mathcal{N}(i)$	The k-step and 1-step neighbors of v_i
$\mathbf{H}^l$	The hidden representation in the l^{th} layer
f_l	The number of dimensions of $\mathbf{H}^l$
$\rho(\cdot)$	Some non-linear activation
$\mathbf{X}_1 \odot \mathbf{X}_2$	Element-wise multiplication
$\mathbf{\Theta}$	Learnable parameters

2 Notations and Preliminaries

Notations. In this paper, a graph[2] is represented as $G = (V, E)$ where $V = \{v_1, ..., v_N\}$ is a set of $N = |V|$ nodes and $E \subseteq V \times V$ is a set of $M = |E|$ edges between nodes. We use $\mathbf{A} \in \mathbb{R}^{N \times N}$ to denote the adjacency matrix, where its i^{th} row, j^{th} column, and an element denoted as $\mathbf{A}(i,:), \mathbf{A}(:,j), \mathbf{A}(i,j)$, respectively. The graph can be directed/undirected and weighted/unweighted. We mainly consider unsigned graphs, so $\mathbf{A}(i,j) \geq 0$. Signed graphs will be discussed in future works. We use $\mathbf{F}^V$ and $\mathbf{F}^E$ to denote features for nodes and edges, respectively. For other variables, we use bold uppercase characters to denote matrices and bold lowercase characters to denote vectors, e.g. $\mathbf{X}$ and $\mathbf{x}$. The transpose of a matrix is denoted as $\mathbf{X}^T$ and the element-wise multiplication is denoted as $\mathbf{X}_1 \odot \mathbf{X}_2$. Functions are marked by curlicue, e.g. $\mathcal{F}(\cdot)$.

Preliminaries. For an undirected graph, its Laplacian matrix is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D} \in \mathbb{R}^{N \times N}$ is a diagonal degree

2. We only consider graphs without self-loops or multiple edges.

matrix with $\mathbf{D}(i,i) = \sum_j \mathbf{A}(i,j)$. Its eigen-decomposition is denoted as $\mathbf{L} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\mathbf{T}$, where $\mathbf{\Lambda} \in \mathbb{R}^{N\times N}$ is a diagonal matrix of eigenvalues sorted in the ascending order and $\mathbf{Q} \in \mathbb{R}^{N\times N}$ are the corresponding eigenvectors. The transition matrix is defined as $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$, where $\mathbf{P}(i,j)$ represents the probability of a random walk starting from node v_i lands at node v_j. The k-step neighbors of node v_i are defined as $\mathcal{N}_k(i) = \{j|\mathcal{D}(i,j) \leq k\}$, where $\mathcal{D}(i,j)$ is the shortest distance from node v_i to v_j, i.e. $\mathcal{N}_k(i)$ is a set of nodes reachable from node v_i within k-steps. To simplify notations, we drop the subscript for the immediate neighborhood, i.e. $\mathcal{N}(i) = \mathcal{N}_1(i)$.

For a deep learning model, we use superscripts to denote layers, e.g. $\mathbf{H}^l$. We use f_l to denote the number of hidden dimensions in layer l. The sigmoid activation function is defined as $\sigma(x) = 1/\left(1+e^{-x}\right)$ and rectifier linear unit (ReLU) is defined as $\text{ReLU}(x) = max(0,x)$. A general element-wise non-linear activation function is denoted as $\rho(\cdot)$. In this paper, unless stated otherwise, we assume all functions are differentiable so that we can learn model parameters $\mathbf{\Theta}$ through back-propagation [18] using commonly adopted optimizers, such as Adam [19], and training techniques, such as dropout [20]. We summarize the notations in Table 2.

The tasks for learning a deep model on graphs can be broadly categorized into two domains:

- **Node-focused tasks**: the tasks are associated with individual nodes in the graph. Examples include node classification, link prediction, and node recommendation.
- **Graph-focused tasks**: the tasks are associated with the whole graph. Examples include graph classification, estimating certain properties of the graph or generating graphs.

Note that such distinctions are more conceptually than mathematically rigorous. On the one hand, there exist tasks associated with mesoscopic structures such as community detection [21]. In addition, node-focused problems can sometimes be studied as graph-focused problems by transforming the former into ego-centric networks [22]. Nevertheless, we will explain the differences in algorithm designs for these two categories when necessary.

3 Graph Recurrent Neural Networks

Recurrent Neural Networks (RNNs) such as Gated Recurrent Units (GRU) [31] or LSTM [32] are de facto standards in modeling sequential data. In this section, we first review Graph Recurrent Neural Networks which can capture recursive and sequential patterns of graphs. Graph RNNs can be broadly divided into two categories: node-level RNNs and graph-level RNNs. The main distinction lies in whether the pattern is in the node-level and modeled by states of nodes, or in the graph-level and modeled by a common state of the graph. The main characteristics of the methods surveyed are summarized in Table 3.

3.1 Node-level RNNs

Node-level RNNs for graphs, which are also referred to as Graph Neural Networks (GNNs)[3], can be dated back to the "pre-deep-learning" era [23], [33]. The idea of GNN is simple: to encode structural information of the graph, each node v_i can be represented by a low-dimensional state vector $\mathbf{s}_i, 1 \leq i \leq N$. Motivated by recursive neural networks [34], a recursive definition of states is adopted [23]:

$$\mathbf{s}_i = \sum\nolimits_{j\in\mathcal{N}(i)} \mathcal{F}\left(\mathbf{s}_i, \mathbf{s}_j, \mathbf{F}_i^V, \mathbf{F}_j^V, \mathbf{F}_{i,j}^E\right), \tag{1}$$

where $\mathcal{F}(\cdot)$ is a parametric function to be learned. After obtaining $\mathbf{s}_i$, another parametric function $\mathcal{O}(\cdot)$ is applied to get the final outputs:

$$\hat{y}_i = \mathcal{O}\left(\mathbf{s}_i, \mathbf{F}_i^V\right). \tag{2}$$

For graph-focused tasks, the authors suggest adding a special node with unique attributes corresponding to the whole graph. To learn model parameters, the following semi-supervised[4] method is adopted: after iteratively solving Eq. (1) to a stable point using Jacobi method [35], one step of gradient descent is performed using the Almeida-Pineda algorithm [36], [37] to minimize a task-specific objective function, for example, the square loss between predicted values and the ground-truth for regression tasks; then, this process is repeated until convergence.

NN4G [24] takes a similar approach but differs in the definition of states. By removing $\mathbf{s}_i$ in the right-hand side of Eq (1), the states in NN4G are not recursively defined, which are easier to compute.

With two simple equations in Eqs. (1)(2), GNN plays two important roles. In retrospect, GNN unifies some early methods in processing graph data, such as recursive neural networks and Markov chains [23]. Looking to the future, the general idea in GNN has profound inspirations: as will be shown later, many state-of-the-art GCNs actually have a similar formulation as Eq. (1), following the framework of exchanging information with immediate node neighborhoods. In fact, GNNs and GCNs can be unified into some common frameworks and GNN is equivalent to GCN using identical layers to reach a stable state. More discussion will be given in Section 4.

Though conceptually important, GNN has several drawbacks. First, to ensure that Eq. (1) has a unique solution, $\mathcal{F}(\cdot)$ has to be a "contraction map" [38], which severely limits the modeling ability. Second, since many iterations are needed to reach stable states between gradient descend steps, GNN is computationally expensive. Because of these drawbacks and perhaps the lack of computational power (e.g. Graphics Processing Unit, GPU, is not widely used for deep learning those days) and lack of research interests, GNN was not a general research focus in the first place.

A notable improvement to GNN is Gated Graph Sequence Neural Networks (GGS-NNs) [25] with the following modifications. Most importantly, the authors replace the recursive definition of Eq. (1) with GRU, thus removing the requirement of "contraction map" and supporting modern optimization techniques. Specifically, Eq. (1) is adapted as:

$$\mathbf{s}_i^{(t)} = \left(1 - \mathbf{z}_i^{(t)}\right) \odot \mathbf{s}_i^{(t-1)} + \mathbf{z}_i^{(t)} \odot \widetilde{\mathbf{s}}_i^{(t)}, \tag{3}$$

where $\mathbf{z}$ is calculated by update gates, $\widetilde{\mathbf{s}}$ are candidates for updating and t is the pseudo time. Secondly, the authors propose using several such networks operating in sequence to produce a sequence output, which can be applied to applications such as program verification [39].

SSE [26] takes a similar approach as Eq. (3). Instead of using GRU in the calculation, SSE adopts stochastic fixed-point gradient descent to accelerate training, which basically alternates between

3. Recently, GNNs are also used to refer to general neural networks for graph data. We follow the traditional name convention and use GNNs to refer to this specific type of Graph RNNs.

4. It is called semi-supervised because all the graph structures and a part of node or graph labels are used during training.

TABLE 3
The Main Characteristics of Graph RNNs

Category	Method	Recursive/sequential patterns of graphs	Scalability	Other Improvements
Node-level	GNN [23]	Recursive definition of node states	No	-
	NN4G [24]		No	-
	GGS-NNs [25]		Yes	Sequence output
	SSE [26]		Yes	-
Graph-level	You et al. [27]	Generate nodes and edges in an autoregressive manner	No	-
	DGNN [28]	Capture the time dynamics of the formation of nodes and edges	Yes	-
	RMGCNN [29]	Recursively reconstruct the graph	Yes	Convolutional layers
	Dynamic GCN [30]	Gather node representations in different time slices	Yes	Convolutional layers

calculating steady states of nodes using local neighborhoods and optimizing model parameters, both in stochastic mini-batches.

GNN and its extensions have many applications. For example, CommNet [40] applies GNN to multi-agent AI systems by regarding each agent as a node and updating the states of agents by communicating with others for several time steps before taking an action. Interaction Network (IN) [41] uses GNN for physical reasoning by representing objects as nodes, relations as edges, and using pseudo-time as a simulation system. VAIN [42] improves CommNet and IN by introducing attentions to weigh different interactions. Relation Networks (RNs) [43] propose using GNN as a relational reasoning module to augment other neural networks and show promising results in visual question answering problems. [44] applied GNN to model the hidden states of the graph in graph generation.

3.2 Graph-level RNNs

In this subsection, we review how to apply RNNs to capture graph-level patterns, e.g. temporal patterns of dynamic graph or sequential patterns in the graph's different levels of granularities. In graph-level RNNs, instead of learning states for each node, the states are encoded in a single RNN applied to the whole graph.

You et al. [27] apply Graph RNN to the graph generation problem. Specifically, they adopt two RNNs, one for generating new nodes while the other generates edges for the newly added node in an autoregressive manner. They show that such hierarchical RNN architecture can effectively learn from input graphs compared to the traditional rule-based graph generative models while having a reasonable time complexity.

Dynamic Graph Neural Network (DGNN) [28] proposes using time-aware LSTM [45] to learn node representations, aiming to capture the temporal information of dynamic graphs. After a new edge is established, DGNN uses LSTM to update the representation of the two interacting nodes as well as their immediate neighbors, i.e. considering the one-step propagation effect. The authors show that time-aware LSTM can well model the establishing orders and time intervals of edge formations, which in turn benefits a range of graph applications.

It is also possible to use Graph RNN in conjunction with other architectures, such as GCNs or GAEs. For example, RMGCNN [29] applies LSTM to the results of GCNs to progressively reconstruct the graph as illustrated in Figure 2, aiming to tackle the graph sparsity problem. By using LSTM, the information from different parts of the graph can diffuse across long ranges without needing too many GCN layers. Dynamic GCN [30] applies LSTM to gather results of GCNs of different time slices in dynamic networks, aiming to capture both spatial and temporal graph information.

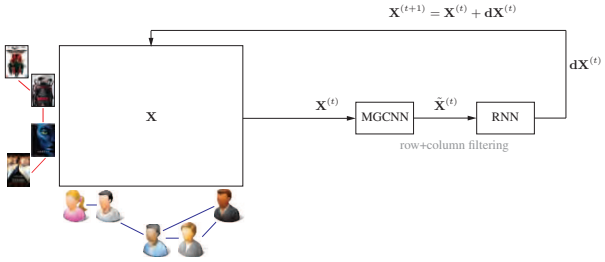

Fig. 2. The framework of RMGCNN reprinted from [29] with permission. RMGCNN adds LSTM into GCN to progressively reconstruct the graph. $\mathbf{X}^t$, $\tilde{\mathbf{X}}^t$, and $d\mathbf{X}^t$ represents the estimated matrix (inputs to GCNs), the outputs of GCNs, and the incremental updates produced by RNN at iteration t, respectively. MGCNN refers to Multi-Graph CNN.

4 Graph Convolutional Networks (GCNs)

Graph Convolutional Networks (GCNs) are inarguably the hottest topic of graph-based deep learning. Mimicking CNNs, modern GCNs learn common local and global structural patterns on graphs through designing convolution and readout functions. Since most GCNs can be trained with task-specific loss via back-propagation (with few exceptions such as unsupervised training in [80]), we focus on the architectures adopted. We first discuss the convolution operations, then move to the readout operations and some other improvements. We summarize the main characteristics of GCNs surveyed in this paper in Table 4.

4.1 Convolution Operations

4.1.1 Spectral Methods

For CNNs, convolution is the most fundamental operation. However, standard convolution for image or text can not be directly applied to graphs because of lacking a grid structure [6]. Bruna et al. [46] first introduce convolution for graph data from the spectral domain using the graph Laplacian matrix $\mathbf{L}$ [81], which plays a similar role as the Fourier basis for signal processing [6]. Specifically, the convolution operation on the graph $*_G$ is defined as:

$$\mathbf{u}_1 *_G \mathbf{u}_2 = \mathbf{Q}\left(\left(\mathbf{Q}^T\mathbf{u}_1\right) \odot \left(\mathbf{Q}^T\mathbf{u}_2\right)\right), \tag{4}$$

where $\mathbf{u}_1, \mathbf{u}_2 \in \mathbb{R}^N$ are two signals defined on nodes and $\mathbf{Q}$ are eigenvectors of $\mathbf{L}$. Briefly speaking, $\mathbf{Q}^T$ transforms graph signals $\mathbf{u}_1, \mathbf{u}_2$ into the spectral domain, and $\mathbf{Q}$ does the inverse transform after an element-wise product. The validity of this definition is based on the convolution theorem, i.e. the Fourier transform of a convolution operation is the element-wise product of their Fourier transforms. Then, filtering a signal $\mathbf{u}$ can be obtained as

$$\mathbf{u}' = \mathbf{Q}\mathbf{\Theta}\mathbf{Q}^T\mathbf{u}, \tag{5}$$

TABLE 4
A Comparison of Different Graph Convolutional Networks (GCNs). M.G. = Multiple Graphs

Method	Type	Convolution	Readout	Scalability	M.G.	Other Characteristics
Bruna et al. [46]	Spectral	Interpolation Kernel	Hierarchical Clustering + FC	No	No	-
Henaff et al. [47]	Spectral	Interpolation Kernel	Hierarchical Clustering + FC	No	No	Constructing the Graph
ChebNet [48]	Spectral/Spatial	Polynomial	Hierarchical Clustering	Yes	Yes	-
Kipf&Welling [49]	Spectral/Spatial	First-order	-	Yes	-	-
CayletNet [50]	Spectral	Polynomial	-	Yes	No	-
GWNN [51]	Spectral	Wavelet Transform	-	Yes	No	-
Neural FPs [52]	Spatial	First-order	Sum	No	Yes	-
PATCHY-SAN [53]	Spatial	Polynomial + Order	Order + Pooling	Yes	Yes	An Order for Neighbors
LGCN [54]	Spatial	First-order + Order	-	Yes	Yes	An Order for Neighbors
SortPooling [55]	Spatial	First-order	Order + Pooling	Yes	Yes	An Order for Nodes
DCNN [56]	Spatial	Polynomial Diffusion	Mean	No	Yes	Edge Features
DGCN [57]	Spatial	First-order + Diffusion	-	No	-	-
MPNNs [58]	Spatial	First-order	Set2set	Yes	Yes	General Framework
GraphSAGE [59]	Spatial	First-order + Sampling	-	Yes	-	General Framework
MoNet [60]	Spatial	First-order	Hierarchical Clustering	Yes	Yes	General Framework
GNs [9]	Spatial	First-order	Whole Graph Representation	Yes	Yes	General Framework
Kearnes et al. [61]	Spatial	Weave module	Fuzzy Histogram	Yes	Yes	Edge Features
DiffPool [62]	Spatial	Various	Hierarchical Clustering	No	Yes	Differentiable Pooling
GAT [63]	Spatial	First-order	-	Yes	Yes	Attention
GaAN [64]	Spatial	First-order	-	Yes	Yes	Attention
HAN [65]	Spatial	Meta-path Neighbors	-	Yes	Yes	Attention
CLN [66]	Spatial	First-order	-	Yes	-	-
PPNP [67]	Spatial	First-order	-	Yes	-	Teleportation Connection
JK-Nets [68]	Spatial	Various	-	Yes	Yes	Jumping Connection
ECC [69]	Spatial	First-order	Hierarchical Clustering	Yes	Yes	Edge Features
R-GCNs [70]	Spatial	First-order	-	Yes	-	Edge Features
LGNN [71]	Spatial	First-order + LINE graph	-	Yes	-	Edge Features
PinSage [72]	Spatial	Random Walk	-	Yes	-	Neighborhood Sampling
StochasticGCN [73]	Spatial	First-order + Sampling	-	Yes	-	Neighborhood Sampling
FastGCN [74]	Spatial	First-order + Sampling	-	Yes	Yes	Layer-wise Sampling
Adapt [75]	Spatial	First-order + Sampling	-	Yes	Yes	Layer-wise Sampling
Li et al. [76]	Spatial	First-order	-	Yes	-	Theoretical analysis
SGC [77]	Spatial	Polynomial	-	Yes	Yes	Theoretical analysis
GFNN [78]	Spatial	Polynomial	-	Yes	Yes	Theoretical analysis
GIN [79]	Spatial	First-order	Sum + MLP	Yes	Yes	Theoretical analysis
DGI [80]	Spatial	First-order	-	Yes	Yes	Unsupervised training

where $\mathbf{u}'$ is the output signal, $\mathbf{\Theta} = \mathbf{\Theta}(\mathbf{\Lambda}) \in \mathbb{R}^{N \times N}$ is a diagonal matrix of learnable filters and $\mathbf{\Lambda}$ are eigenvalues of $\mathbf{L}$. Then, a convolutional layer is defined by applying different filters to different input and output signals as follows:

$$\mathbf{u}_j^{l+1} = \rho \left(\sum\nolimits_{i=1}^{f_l} \mathbf{Q} \mathbf{\Theta}_{i,j}^l \mathbf{Q}^T \mathbf{u}_i^l \right) \; j = 1, ..., f_{l+1}, \tag{6}$$

where l is the layer, $\mathbf{u}_j^l \in \mathbb{R}^N$ is the j^{th} hidden representation (i.e. signals) for nodes in the l^{th} layer, $\mathbf{\Theta}_{i,j}^l$ are learnable filters. The idea of Eq. (6) is similar to conventional convolutions: passing the input signals through a set of learnable filters to aggregate the information, followed by some non-linear transformation. By using nodes features $\mathbf{F}^V$ as the input layer and stacking multiple convolutional layers, the overall architecture is similar to CNNs. Theoretical analysis shows that such a definition of convolution operation on graphs can mimic certain geometric properties of CNNs, which we refer readers to [7] for a comprehensive survey.

However, directly using Eq. (6) requires $O(N)$ parameters to be learned, which may not be feasible in practice. In addition, the filters in the spectral domain may not be localized in the spatial domain, i.e. each node can be affected by all other nodes rather than only nodes in a small region. To alleviate these problems, Bruna et al. [46] suggest using the following smooth filters:

$$diag \left(\mathbf{\Theta}_{i,j}^l \right) = \mathcal{K} \; \alpha_{l,i,j}, \tag{7}$$

where $\mathcal{K}$ is a fixed interpolation kernel and $\alpha_{l,i,j}$ are learnable interpolation coefficients. The authors also generalize this idea to the setting where the graph is not given but constructed from raw features using either a supervised or an unsupervised method [47].

However, two fundamental limitations still remain unsolved. First, since the full eigenvectors of the Laplacian matrix are needed during each calculation, the time complexity is at least $O(N^2)$ per forward and backward pass, not to mention the $O(N^3)$ complexity in calculating the eigen-decomposition, which is not scalable to large-scale graphs. Second, since the filters depend on the eigenbasis $\mathbf{Q}$ of the graph, parameters can not be shared across multiple graphs with different sizes and structures.

Next, we review two lines of works trying to solve these limitations and then unify them using some common frameworks.

4.1.2 Efficiency Aspect

To solve the efficiency problem, ChebNet [48] proposes using a polynomial filter as follows:

$$\mathbf{\Theta}(\mathbf{\Lambda}) = \sum\nolimits_{k=0}^{K} \theta_k \mathbf{\Lambda}^k, \tag{8}$$

where $\theta_0, ..., \theta_K$ are learnable parameters and K is the polynomial order. Then, instead of performing the eigen-decomposition, the authors rewrite Eq. (8) using the Chebyshev expansion [82]:

$$\mathbf{\Theta}(\mathbf{\Lambda}) = \sum\nolimits_{k=0}^{K} \theta_k \mathcal{T}_k(\tilde{\mathbf{\Lambda}}), \tag{9}$$

where $\tilde{\mathbf{\Lambda}} = 2\mathbf{\Lambda}/\lambda_{max} - \mathbf{I}$ are the rescaled eigenvalues, λ_{max} is the maximum eigenvalue, $\mathbf{I} \in \mathbb{R}^{N\times N}$ is the identity matrix and $\mathcal{T}_k(x)$ is the Chebyshev polynomial of order k. The rescaling is necessary because of the orthonormal basis of Chebyshev polynomials. Using the fact that polynomial of the Laplacian matrix acts as a polynomial of its eigenvalues, i.e. $\mathbf{L}^k = \mathbf{Q}\mathbf{\Lambda}^k\mathbf{Q}^T$, the filter operation in Eq. (5) can be rewritten as:

$$\begin{aligned}\mathbf{u}' = \mathbf{Q}\mathbf{\Theta}(\mathbf{\Lambda})\mathbf{Q}^T\mathbf{u} &= \sum_{k=0}^{K}\theta_k\mathbf{Q}\mathcal{T}_k(\tilde{\mathbf{\Lambda}})\mathbf{Q}^T\mathbf{u} \\ &= \sum_{k=0}^{K}\theta_k\mathcal{T}_k(\tilde{\mathbf{L}})\mathbf{u} = \sum_{k=0}^{K}\theta_k\bar{\mathbf{u}}_k,\end{aligned} \tag{10}$$

where $\bar{\mathbf{u}}_k = \mathcal{T}_k(\tilde{\mathbf{L}})\mathbf{u}$ and $\tilde{\mathbf{L}} = 2\mathbf{L}/\lambda_{max} - \mathbf{I}$. Using the recurrence relation of Chebyshev polynomial $\mathcal{T}_k(x) = 2x\mathcal{T}_{k-1}(x) - \mathcal{T}_{k-2}(x)$ and $\mathcal{T}_0(x) = 1, \mathcal{T}_1(x) = x$, $\bar{\mathbf{u}}_k$ can also be calculated recursively:

$$\bar{\mathbf{u}}_k = 2\tilde{\mathbf{L}}\bar{\mathbf{u}}_{k-1} - \bar{\mathbf{u}}_{k-2} \tag{11}$$

with $\bar{\mathbf{u}}_0 = \mathbf{u}, \bar{\mathbf{u}}_1 = \tilde{\mathbf{L}}\mathbf{u}$. Now, since only the matrix multiplication of a sparse matrix $\tilde{\mathbf{L}}$ and some vectors needs to be calculated, the time complexity is $O(KM)$ by using sparse matrix multiplications, where M is the number of edges and K is the polynomial order, i.e. linear with respect to the graph size. It is also easy to see that such a polynomial filter is strictly K-localized: after one convolution, the representations of node v_i will only be affected by its K-step neighborhood $\mathcal{N}_K(i)$. Interestingly, this idea is independently used in network embedding to preserve the high-order proximity [83], of which we omit the details for brevity.

An improvement to ChebNet introduced by Kipf and Welling [49] further simplifies the filtering by only using the first-order neighbors as follows:

$$\mathbf{h}_i^{l+1} = \rho\left(\sum_{j\in\tilde{\mathcal{N}}(i)}\frac{1}{\sqrt{\tilde{\mathbf{D}}(i,i)\tilde{\mathbf{D}}(j,j)}}\mathbf{h}_j^l\mathbf{\Theta}^l\right), \tag{12}$$

where $\mathbf{h}_j^l \in \mathbb{R}^{f_l}$ is the hidden representation of node v_i in the l^{th} layer[5], $\tilde{\mathbf{D}} = \mathbf{D} + \mathbf{I}$ and $\tilde{\mathcal{N}}(i) = \mathcal{N}(i) \cup \{i\}$. This can be written equivalently in the matrix form:

$$\mathbf{H}^{l+1} = \rho\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^l\mathbf{\Theta}^l\right), \tag{13}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, i.e. adding a self-connection. The authors show that Eq. (13) is a special case of Eq. (8) by setting $K = 1$ with a few minor changes. Then, the authors argue that stacking an adequate number of layers as illustrated in Figure 3 has a similar modeling capacity as ChebNet and leads to better results.

An important insight of ChebNet and its extension is that they connect the spectral graph convolution with the spatial architecture as in GNNs, i.e. defining graph convolutions by considering the neighborhoods of nodes. Specifically, they show that when the spectral convolution function is polynomial or first-order, spectral graph convolution is equivalent to the spatial convolution. In addition, the convolution in Eq. (12) is very similar to the definition of states in GNN in Eq. (1), except that the convolution definition replaces the recursive definition. In this aspect, GNN can be regarded as GCN using a large number of identical layers to reach stable states [7], i.e. GNN uses a fixed function with fixed parameters to iteratively update node hidden states until equilibrium, while GCN has a preset number of layers with each layer containing different parameters.

5. We use a different letter because $\mathbf{h}^l \in \mathbb{R}^{f_l}$ is the hidden representation of one node, while $\mathbf{u}^l \in \mathbb{R}^N$ represents a dimension for all nodes.

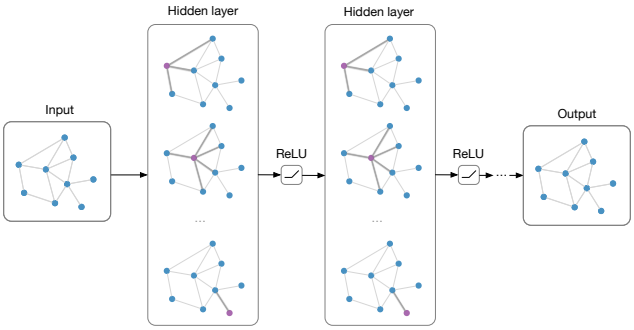

Fig. 3. An illustrating example of the spatial convolution operation proposed by Kipf and Welling [49]. Nodes are only affected by their immediate neighbors in each convolutional layer. Reprinted with permission.

It is worth mentioning that some pure spectral methods have also been proposed to solve the efficiency problem. For example, instead of using Chebyshev expansion as in Eq. (9), CayleyNet [50] adopts the Cayley polynomials in defining convolutions:

$$\mathbf{\Theta}(\mathbf{\Lambda}) = \theta_0 + 2Re\left\{\sum_{k=1}^{K}\theta_k\left(\theta_h\mathbf{\Lambda} - i\mathbf{I}\right)^j\left(\theta_h\mathbf{\Lambda} + i\mathbf{I}\right)^j\right\}, \tag{14}$$

where $i = \sqrt{-1}$ denotes the imaginary unit and θ_h is another spectral zoom parameter. Besides being efficient as ChebNet, the authors demonstrate that the Cayley polynomials can detect "narrow frequency bands of importance" to achieve better results. Graph Wavelet Neural Network (GWNN) [51] further proposes to replace the Fourier transform in spectral filters by the graph wavelet transform, i.e. rewrite Eq. (4) as:

$$\mathbf{u}_1 *_G \mathbf{u}_2 = \psi\left(\left(\psi^{-1}\mathbf{u}_1\right) \odot \left(\psi^{-1}\mathbf{u}_2\right)\right), \tag{15}$$

where ψ is the graph wavelet bases. By using fast approximating algorithms to calculate ψ and ψ^{-1}, the computational complexity is also $O(KM)$, i.e. linear with respect to the graph size.

4.1.3 Multiple Graphs Aspect

In the meantime, a parallel of works focuses on generalizing convolution operation to multiple graphs of arbitrary sizes. Neural FPs [52] propose a spatial method also using the first-order neighbors:

$$\mathbf{h}_i^{l+1} = \sigma\left(\sum_{j\in\tilde{\mathcal{N}}(i)}\mathbf{h}_j^l\mathbf{\Theta}^l\right). \tag{16}$$

Since the parameters $\mathbf{\Theta}$ can be shared across different graphs and are independent of graph sizes, Neural FPs can handle multiple graphs of arbitrary sizes. Note that Eq. (16) is very similar to Eq. (12). However, instead of considering the influence of node degrees by adding a normalization term, Neural FPs propose learning different parameters $\mathbf{\Theta}$ for nodes with different degrees. This strategy performs well for small graphs such as the molecular graphs, i.e. atoms as nodes and bonds as edges, but may not be scalable to large-scale graphs.

PATCHY-SAN [53] adopts a different idea to assign a unique order of neighbors using the graph labeling procedure such as the Weisfeiler-Lehman kernel [84] and arrange nodes in a line using this pre-defined order. In addition, PATCHY-SAN defines a "receptive field" for each node v_i by selecting a fixed number of nodes from its k-step neighborhoods $\mathcal{N}_k(i)$. Then standard 1-D CNN with proper normalization is adopted. Since now nodes in different graphs all have a "receptive field" with a fixed size and

order, PATCHY-SAN can learn from multiple graphs like normal CNNs. The drawbacks are that the convolution depends heavily on the graph labeling procedure which is a preprocessing step that is not learned. LGCN [54] further proposes to simplify the sorting process by using the lexicographical order, i.e. sort neighbors based on their hidden representations in the last layer $\mathbf{H}^L$. Instead of using a single order, the authors sort different channels of $\mathbf{H}^L$ separately. SortPooling [55] takes a similar idea, but rather than sorting neighbors for each node, the authors propose to sort all nodes directly, i.e. a single order for all neighborhoods. Despite the differences between these methods, enforcing a 1-D order of nodes may not be a natural choice for graphs.

DCNN [56] adopts another approach to replace the eigenbasis of the convolution by a diffusion-basis, i.e. the "receptive field" of nodes is determined by the diffusion transition probability between nodes. Specifically, the convolution is defined as:

$$\mathbf{H}^{l+1} = \rho\left(\mathbf{P}^K \mathbf{H}^l \mathbf{\Theta}^l\right), \tag{17}$$

where $\mathbf{P}^K = (\mathbf{P})^K$ is the transition probability of a length K diffusion process (i.e. random walk), K is a preset diffusion length and $\mathbf{\Theta}^l \in \mathbb{R}^{f_l \times f_l}$ is a diagonal matrix of learnable parameters. Since only $\mathbf{P}^K$ depend on the graph structure, the parameters $\mathbf{\Theta}^l$ can be shared across graphs of arbitrary sizes. However, calculating $\mathbf{P}^K$ induces the time complexity $O\left(N^2K\right)$, thus making the method not scalable to large-scale graphs.

DGCN [57] further proposes to jointly adopt diffusion and adjacency basis using a dual graph convolutional network. Specifically, DGCN uses two convolutions: one as Eq. (13), and the other replaces the adjacency matrix with the positive pointwise mutual information (PPMI) matrix [85] of the transition probability, i.e.

$$\mathbf{Z}^{l+1} = \rho\left(\mathbf{D}_P^{-\frac{1}{2}} \mathbf{X}_P \mathbf{D}_P^{-\frac{1}{2}} \mathbf{Z}^l \mathbf{\Theta}^l\right), \tag{18}$$

where $\mathbf{X}_P$ is the PPMI matrix and $\mathbf{D}_P(i,i) = \sum_j \mathbf{X}_P(i,j)$ is the diagonal degree matrix of $\mathbf{X}_P$. Then, two convolutions are ensembled by minimizing the mean square differences between $\mathbf{H}$ and $\mathbf{Z}$. A random walk sampling procedure is also proposed to accelerate the calculation of the transition probability. Experiments demonstrate that such dual convolutions are effective even for single-graph problems.

4.1.4 Frameworks

Based on the above two lines of works, MPNNs [58] propose a unified framework for the graph convolution operation in the spatial domain using the message-passing functions:

$$\begin{aligned} \mathbf{m}_i^{l+1} &= \sum\nolimits_{j \in \mathcal{N}(i)} \mathcal{F}^l\left(\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{F}_{i,j}^E\right) \\ \mathbf{h}_i^{l+1} &= \mathcal{G}^l\left(\mathbf{h}_i^l, \mathbf{m}_i^{l+1}\right), \end{aligned} \tag{19}$$

where $\mathcal{F}^l(\cdot)$ and $\mathcal{G}^l(\cdot)$ are message functions and vertex update functions that need to be learned, respectively, and $\mathbf{m}^l$ are the "messages" passed between nodes. Conceptually, MPNNs propose a framework that each node sends messages based on its states and updates its states based on messages received from immediate neighbors. The authors show that the above framework includes many existing methods such as [25], [46], [47], [49], [52], [61] as special cases. Besides, the authors propose adding a "master" node that is connected to all nodes to accelerate the passing of messages across long distances and split the hidden representations into different "towers" to improve the generalization ability. The authors show that a specific variant of the MPNNs can achieve state-of-the-art performance in predicting molecular properties.

Concurrently, GraphSAGE [59] takes a similar idea as Eq. (19) with multiple aggregating functions as follows:

$$\begin{aligned} \mathbf{m}_i^{l+1} &= \text{AGGREGATE}^l(\{\mathbf{h}_j^l, \forall j \in \mathcal{N}(i)\}) \\ \mathbf{h}_i^{l+1} &= \rho\left(\mathbf{\Theta}^l\left[\mathbf{h}_i^l, \mathbf{m}_i^{l+1}\right]\right), \end{aligned} \tag{20}$$

where $[\cdot,\cdot]$ is concatenation and $\text{AGGREGATE}(\cdot)$ is the aggregating function. The authors suggest three aggregating functions: element-wise mean, long short-term memory (LSTM) [32] and max-pooling as follows:

$$\text{AGGREGATE}^l = \max\{\rho(\mathbf{\Theta}_{\text{pool}}\mathbf{h}_j^l + \mathbf{b}_{\text{pool}}), \forall j \in \mathcal{N}(i)\}, \tag{21}$$

where $\mathbf{\Theta}_{\text{pool}}$ and $\mathbf{b}_{\text{pool}}$ are parameters to be learned and $\max\{\cdot\}$ is element-wise maximum. For the LSTM aggregating function, since an order of neighbors is needed, the authors adopt the simple random order.

Mixture model network (MoNet) [60] also tries to unify the existing works of GCNs as well as CNN for manifolds into a common framework using "template matching":

$$h_{ik}^{l+1} = \sum\nolimits_{j \in \mathcal{N}(i)} \mathcal{F}_k^l(\mathbf{u}(i,j))\mathbf{h}_j^l, k = 1, ..., f_{l+1}, \tag{22}$$

where $\mathbf{u}(i,j)$ are the pseudo-coordinates of node pair v_i and v_j, $\mathcal{F}_k^l(\mathbf{u})$ is a parametric function to be learned, h_{ik}^l is the k^{th} dimension of $\mathbf{h}_i^l$. In other words, $\mathcal{F}_k^l(\mathbf{u})$ serve as the weighting kernel for combining neighborhoods. Then, MoNet suggests using the Gaussian kernel:

$$\mathcal{F}_k^l(\mathbf{u}) = \exp\left(-\frac{1}{2}(\mathbf{u} - \boldsymbol{\mu}_k^l)^T (\mathbf{\Sigma}_k^l)^{-1} (\mathbf{u} - \boldsymbol{\mu}_k^l)\right), \tag{23}$$

where $\boldsymbol{\mu}_k^l$ are mean vectors and $\mathbf{\Sigma}_k^l$ are diagonal covariance matrices to be learned. The pseudo-coordinates are set to be degrees as in [49], i.e.

$$\mathbf{u}(i,j) = \left(\frac{1}{\sqrt{\mathbf{D}(i,i)}}, \frac{1}{\sqrt{\mathbf{D}(j,j)}}\right). \tag{24}$$

Graph Networks (GNs) [9] propose a more general framework for both GCNs and GNNs to learn three sets of representations: $\mathbf{h}_i^l, \mathbf{e}_{ij}^l, \mathbf{z}^l$ as the representation for nodes, edges and the whole graph respectively. The representations are learned using three aggregation functions and three update functions:

$$\begin{aligned} \mathbf{m}_i^l &= \mathcal{G}^{E\to V}(\{\mathbf{h}_j^l, \forall j \in \mathcal{N}(i)\}) \\ \mathbf{m}_V^l &= \mathcal{G}^{V\to G}(\{\mathbf{h}_i^l, \forall v_i \in V\}) \\ \mathbf{m}_E^l &= \mathcal{G}^{E\to G}(\{\mathbf{h}_{ij}^l, \forall (v_i, v_j) \in E\}) \\ \mathbf{h}_i^{l+1} &= \mathcal{F}^V(\mathbf{m}_i^l, \mathbf{h}_i^l, \mathbf{z}^l) \\ \mathbf{e}_{ij}^{l+1} &= \mathcal{F}^E(\mathbf{e}_{ij}^l, \mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{z}^l) \\ \mathbf{z}^{l+1} &= \mathcal{F}^G(\mathbf{m}_E^l, \mathbf{m}_V^l, \mathbf{z}^l), \end{aligned} \tag{25}$$

where $\mathcal{F}^V(\cdot), \mathcal{F}^E(\cdot), \mathcal{F}^G(\cdot)$ are corresponding updating functions for nodes, edges and the whole graph respectively and $\mathcal{G}(\cdot)$ are message-passing functions with superscripts denoting message-passing directions. Note that the message-passing functions all take sets as inputs, thus they should take variable numbers of arguments and invariant to input permutations, e.g. element-wise summation, mean or maximum. Compared with MPNNs, GNs introduce edge representations and the whole graph representation, thus making the framework more general.

In summary, the convolution operations have evolved from the spectral domain to the spatial domain and from multi-step neighbors to the immediate neighbors. Currently, gathering information from immediate neighbors like Eq. (13) and following the framework of Eqs. (19) (20) (25) are the most common choices for the graph convolution operation.

4.2 Readout Operations

Using convolution operations, useful features for nodes can be learned to solve many node-focused tasks. However, to tackle graph-focused tasks, information of nodes needs to be aggregated to form a graph-level representation. In the literature, this is usually called the readout operation[6]. This problem is non-trivial because stride convolutions or pooling in standard CNNs cannot be directly used due to the lack of a grid structure.

Order invariance. A critical requirement for the graph readout operations is that the operation should be invariant to the order of nodes, i.e. if we change the indices of nodes and edges using a bijective function between two vertex sets, representation of the whole graph should not change. For example, whether a drug can treat certain diseases should be independent of how the drug is represented as a graph. Note that since this problem is related to the graph isomorphism problem which is known to be NP [86], we can only find a function that is order-invariant but not vice versa in polynomial time, i.e. even two graphs are not isomorphic, they may have the same representation.

4.2.1 Statistics

The most basic order-invariant operations are simple statistics like taking the sum, average or max-pooling [52], [56], i.e.

$$\mathbf{h}_G = \sum_{i=1}^{N} \mathbf{h}_i^L \text{ or } \mathbf{h}_G = \frac{1}{N}\sum_{i=1}^{N} \mathbf{h}_i^L \text{ or } \mathbf{h}_G = \max\left\{\mathbf{h}_i^L, \forall i\right\}, \tag{26}$$

where $\mathbf{h}_G$ is the representation for graph G and $\mathbf{h}_i^L$ is the representation of node v_i in the final layer L. However, such first-moment statistics may not be representative enough to distinguish different graphs.

In [61], the authors suggest considering the distribution of node representations by using fuzzy histograms [87]. The basic idea of fuzzy histograms is to construct several "histogram bins" and then calculate the memberships of $\mathbf{h}_i^L$ to these bins, i.e. regarding representations of nodes as samples and match them to some pre-defined templates, and return the concatenation of the final histograms. In this way, nodes with the same sum/average/maximum but with different distributions can be distinguished.

Another commonly used approach for gathering information is to add a fully connected (FC) layer as the final layer [46], i.e.

$$\mathbf{h}_G = \rho\left(\left[\mathbf{H}^L\right]\mathbf{\Theta}_{FC}\right), \tag{27}$$

where $\left[\mathbf{H}^L\right] \in \mathbb{R}^{Nf_L}$ is the concatenation of the final node representation $\mathbf{H}^L$, $\mathbf{\Theta}_{FC} \in \mathbb{R}^{Nf_L \times f_{output}}$ are parameters, and f_{output} is the dimensionality of outputs. Eq. (27) can be regarded as a weighted sum of combing node-level features. One advantage is that the model can learn different weights for different nodes, at the cost of being unable to guarantee order invariance.

6. This is also related to graph coarsening, i.e. reducing a large graph to a smaller graph, since the graph-level representation can be obtained by coarsening the graph to a single node. Some papers use two terms exchangeably.

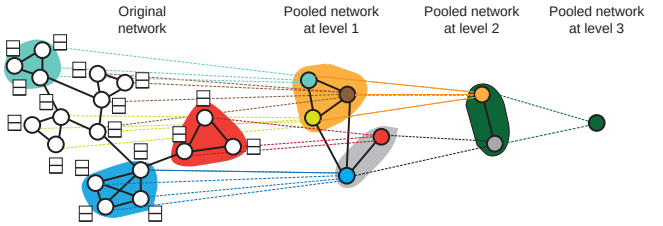

Fig. 4. An example of hierarchical clustering results of the nodes in the graph. Reprinted from [62] with permission.

4.2.2 Hierarchical clustering

Rather than a dichotomy between node or graph level structures, graphs are known to exhibit rich hierarchical structures [88], which can be explored by hierarchical clustering methods as shown in Figure 4. For example, a density-based agglomerative clustering [89] is used in Bruna et al. [46] and multi-resolution spectral clustering [90] is used in Henaff et al. [47]. ChebNet [48] and MoNet [60] adopt Graclus [91], another greedy hierarchical clustering algorithm to merge two nodes at a time, together with a fast pooling method by rearranging the nodes into a balanced binary tree. ECC [69] adopts another hierarchical clustering method by eigen-decomposition [92]. However, these hierarchical clustering methods are all independent of the convolution operation, i.e. can be done as a pre-processing step and not trained end-to-end.

To solve that problem, DiffPool [62] proposes a differentiable hierarchical clustering algorithm jointly trained with graph convolutions. Specifically, the authors propose learning a soft cluster assignment matrix in each layer using the hidden representations:

$$\mathbf{S}^l = \mathcal{F}\left(\mathbf{A}^l, \mathbf{H}^l\right), \tag{28}$$

where $\mathbf{S}^l \in \mathbb{R}^{N_l \times N_{l+1}}$ is the cluster assignment matrix, N_l is the number of clusters in layer l and $\mathcal{F}(\cdot)$ is a function to be learned. Then, the node representations and new adjacency matrix for this "coarsened" graph can be obtained by taking the average according to $\mathbf{S}^l$ as follows:

$$\mathbf{H}^{l+1} = (\mathbf{S}^l)^T\hat{\mathbf{H}}^{l+1}, \mathbf{A}^{l+1} = (\mathbf{S}^l)^T\mathbf{A}^l\mathbf{S}^l, \tag{29}$$

where $\hat{\mathbf{H}}^{l+1}$ is obtained by applying a convolution layer to $\mathbf{H}^l$, i.e. coarsening the graph from N_l nodes to N_{l+1} nodes in each layer after the convolution operation. However, since the cluster assignment is soft, the connections between clusters are not sparse and the time complexity of the method is $O(N^2)$ in principle.

4.2.3 Others

Besides the aforementioned methods, there are other readout operations worthy of discussion.

In GNNs [23], the authors suggest adding a special node that is connected to all nodes to represent the whole graph. Similarly, GNs [9] take the idea of directly learning the representation of the whole graph by receiving messages from all nodes and edges.

MPNNs adopt set2set [93], a modification of seq2seq model that is invariant to the order of inputs. Specifically, set2set uses a Read-Process-and-Write model that receives all inputs at once, computes internal memories using an attention mechanism and LSTM, and then writes the outputs.

As mentioned in Section 4.1.3, PATCHY-SAN [53] takes the idea of imposing an order for nodes using a graph labeling procedure and then resorts to standard 1-D pooling as in CNNs.

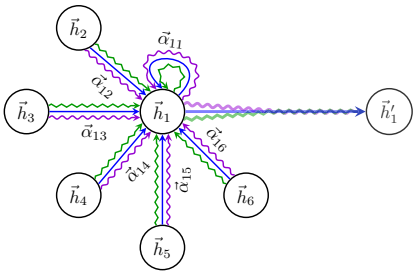

Fig. 5. An illustration of the multi-head attentions proposed in GAT [63] where each color denotes an independent attention. Reprinted with permission.

Whether this method can preserve order invariance depends on the graph labeling procedure, which is another research field that we refer readers to [94] for a survey. However, imposing an order for nodes may not be a natural choice for graphs and could hinder the performance of downstream tasks.

4.2.4 Summary

In short, statistics like taking the average or sum are most simple readout operations, while hierarchical clustering algorithms jointly trained with graph convolutions are more advanced but are more sophisticated solutions. Other methods like adding a pseudo node or imposing an order exist as well.

4.3 Improvements and Discussions

4.3.1 Attention Mechanism

In aforementioned GCNs, the neighborhoods of nodes are aggregated with equal or pre-defined weights. However, the influence of neighbors can vary greatly, which should be learned during training than pre-determined. Inspired by the attention mechanism [95], Graph Attention Network (GAT) [63] introduces attentions into GCNs by modifying the convolution in Eq (12) as follows:

$$\mathbf{h}_i^{l+1} = \rho\left(\sum\nolimits_{j\in\hat{\mathcal{N}}(i)} \alpha_{ij}^l \mathbf{h}_j^l \mathbf{\Theta}^l\right), \tag{30}$$

where α_{ij}^l is node v_i's attention to node v_j in layer l defined as:

$$\alpha_{ij}^l = \frac{\exp\left(\text{LeakyReLU}\left(\mathcal{F}\left(\mathbf{h}_i^l\mathbf{\Theta}^l, \mathbf{h}_j^l\mathbf{\Theta}^l\right)\right)\right)}{\sum_{k\in\hat{\mathcal{N}}(i)}\exp\left(\text{LeakyReLU}\left(\mathcal{F}\left(\mathbf{h}_i^l\mathbf{\Theta}^l, \mathbf{h}_k^l\mathbf{\Theta}^l\right)\right)\right)}, \tag{31}$$

where $\mathcal{F}(\cdot,\cdot)$ is another function to be learned such as a small fully connected network. The authors also suggest using multiple independent attentions and concatenating the results to improve model capacity and stability, i.e. the multi-head attention in [95], as illustrated in Figure 5. GaAN [64] further proposes to learn different weights for different heads, and applies their method to the traffic forecasting problem using RNN to generate the outputs.

HAN [65] proposes a two-level attention mechanism for heterogeneous graphs. Specifically, the node-level attention is similar to GAT, but considers the types of nodes. As a result, it can assign weights in aggregating meta-path-based neighbors for nodes. The semantic-level attention then learns the importance of different meta-paths and outputs the final results.

4.3.2 Residual and Jumping Connections

Many works observe that the most suitable depth for existing GCNs is often very limited, e.g. two or three layers, potentially due to the practical difficulty in training GCNs or the over-smoothing problem, i.e. all nodes have the same representation in deeper layers [68], [76]. To remedy this problem, residual connections similar to ResNet [96] can be added to skip layers. For example, [49] add residual connections into Eq. (13) as follows:

$$\mathbf{H}^{l+1} = \rho\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^l\mathbf{\Theta}^l\right) + \mathbf{H}^l. \tag{32}$$

They show experimentally that adding such residual connections can increase the depth of the network, which is similar to the results of ResNet.

Column Network (CLN) [66] takes a similar idea using the following residual connections with learnable weights:

$$\mathbf{h}_i^{l+1} = \boldsymbol{\alpha}_i^l \odot \widetilde{\mathbf{h}}_i^{l+1} + (1-\boldsymbol{\alpha}_i^l)\odot \mathbf{h}_i^l, \tag{33}$$

where $\widetilde{\mathbf{h}}_i^{l+1}$ is calculated similar to Eq. (13) and $\boldsymbol{\alpha}_i^l$ are weights calculated as follows:

$$\boldsymbol{\alpha}_i^l = \rho\left(\mathbf{b}_\alpha^l + \mathbf{\Theta}_\alpha^l\mathbf{h}_i^l + \mathbf{\Theta}_\alpha'^l\sum\nolimits_{j\in\mathcal{N}(i)}\mathbf{h}_j^l\right), \tag{34}$$

where $\mathbf{b}_\alpha^l, \mathbf{\Theta}_\alpha^l, \mathbf{\Theta}_\alpha'^l$ are parameters. Note that Eq. (33) is very similar to the GRU as in GGS-NNs [25], but the overall architecture is still as GCN instead of pseudo time.

Inspired by personalized PageRank, PPNP [67] defines graph convolution with teleportation to the initial layer:

$$\mathbf{H}^{l+1} = (1-\alpha)\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^l + \alpha\mathbf{H}^0, \tag{35}$$

where $\mathbf{H}_0 = \mathcal{F}_\theta(\mathbf{F}^V)$ and α is a hyper-parameter. Note that all parameters are in $\mathcal{F}_\theta(\mathbf{F}^V)$ rather than in graph convolutions.

Jumping Knowledge Networks (JK-Nets) [68] propose another architecture to connect the last layer of the network with all lower hidden layers, i.e. "jumping" all representations to the final output as illustrated in Figure 6. In this way, the model can learn to selectively exploit information from different layers. Formally, JK-Nets can be formulated as:

$$\mathbf{h}_i^{\text{final}} = \text{AGGREGATE}(\mathbf{h}_i^0, \mathbf{h}_i^1, ..., \mathbf{h}_i^L), \tag{36}$$

where $\mathbf{h}_i^{\text{final}}$ are the final representation for node v_i, $\text{AGGREGATE}(\cdot)$ is the aggregating function and L is the number of hidden layers. JK-Nets use three aggregating functions similar to GraphSAGE [59]: concatenation, max-pooling, and LSTM attention. Experimental results show that adding jumping connections can improve the performance of multiple GCN architectures.

4.3.3 Edge Features

The aforementioned GCNs mostly focus on utilizing node features and graph structures. In this section, we briefly discuss how to use another important source of information, the edge features.

For simple edge features with discrete values such as edge types, a straight-forward method is to train different parameters for different edge types and aggregate the results. For example, Neural FPs [52] train different parameters for nodes with different degrees, which corresponds to the hidden edge feature of bond types in a molecular graph, and sum over the results. CLN [66] trains different parameters for different types of edges in a heterogeneous graph and averages the results. Edge-Conditioned Convolution (ECC) [69] also trains different parameters based on edge types and applies them to graph classification. Relational

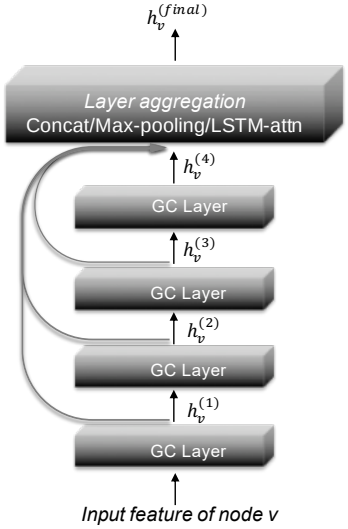

Fig. 6. Jumping Knowledge Networks proposed in [68] where the last layer is connected to all layers to selectively exploit different information. GC stands for graph convolutions. Reprinted with permission.

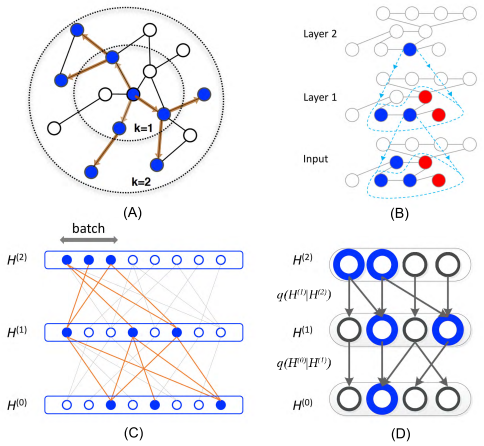

Fig. 7. Node sampling method of (A) GraphSAGE [59], (B) StochasticGCN [73], (C) FastGCN [74], (D) Adapt [75], where blue nodes indicate samples from one batch and arrows indicate sampling directions. Red nodes in figure (B) represent historical samples. Reprinted with permission.

GCNs (R-GCNs) [70] take a similar idea in knowledge graphs by training different weights for different relation types and add regularizations to reduce the number of parameters. However, these methods can only handle limited discrete edge features.

DCNN [56] proposes another method to convert each edge into a node connected to the head and tail node of the edge. Then, edge features can be treated as node features.

Similarly, LGCN [71] constructs a line graph $\mathbf{B} \in \mathbb{R}^{2M \times 2M}$ to incorporate edge features as follows:

$$\mathbf{B}_{i \to j, i' \to j'} = \begin{cases} 1 & \text{if } j = i' \text{ and } j' \neq i, \\ 0 & \text{otherwise,} \end{cases} \tag{37}$$

i.e. nodes in the line graph are directed edges in the original graph and two nodes in the line graph are connected if the information can flow through their corresponding edges in the original graph. Then, LGCN adopts two GCNs on the original graph and line graph, respectively.

Kearnes et al. [61] propose another architecture using the "weave module". Specifically, they learn representations for both nodes and edges and exchange information between them in each weave module with four different functions: Node-to-Node (NN), Node-to-Edge (NE), Edge-to-Edge (EE) and Edge-to-Node (EN):

$$\begin{aligned} \mathbf{h}_i^{l'} &= \mathcal{F}_{NN}(\mathbf{h}_i^0, \mathbf{h}_i^1, ..., \mathbf{h}_i^l) \\ \mathbf{h}_i^{l''} &= \mathcal{F}_{EN}(\{\mathbf{e}_{ij}^l | j \in \mathcal{N}(i)\}) \\ \mathbf{h}_i^{l+1} &= \mathcal{F}_{NN}(\mathbf{h}_i^{l'}, \mathbf{h}_i^{l''}) \\ \mathbf{e}_{ij}^{l'} &= \mathcal{F}_{EE}(\mathbf{e}_{ij}^0, \mathbf{e}_{ij}^1, ..., \mathbf{e}_{ij}^l) \\ \mathbf{e}_{ij}^{l''} &= \mathcal{F}_{NE}(\mathbf{h}_i^l, \mathbf{h}_j^l) \\ \mathbf{e}_{ij}^{l+1} &= \mathcal{F}_{EE}(\mathbf{e}_{ij}^{l'}, \mathbf{e}_{ij}^{l''}), \end{aligned} \tag{38}$$

where $\mathbf{e}_{ij}^l$ are representations for edge (v_i, v_j) in the l^{th} layer and $\mathcal{F}(\cdot)$ are learnable functions with subscripts representing message-passing directions. By stacking multiple such modules, information can propagate through alternative passing between nodes and edges representations. Note that in Node-to-Node and Edge-to-Edge functions, jumping connections similar to JK-Nets [68] are implicitly added. Graph Networks [9] also propose learning edge representation and update both nodes and edges representations using message-passing functions as discussed in Section 4.1.4, which contain the "weave module" as a special case by not learning the whole graph representation.

4.3.4 Sampling Methods

One critical bottleneck of training GCNs for large-scale graphs is efficiency. As shown in Section 4.1.4, many GCNs follow the framework of aggregating information from neighborhoods. However, since many real graphs follow the power-law distribution [97], i.e. few nodes have very large degrees, the expansion of neighbors can grow extremely fast. To deal with this problem, two kinds of sampling methods have been proposed: neighborhood sampling and layer-wise sampling, as illustrated in Figure 7.

In neighborhood samplings, the sampling is performed for each node during the calculation. GraphSAGE [59] uniformly samples a fixed number of neighbors for each node during training. PinSage [72] proposes sampling neighbors using random walks on graphs together with several implementation improvements, e.g. coordination between CPU and GPU, a map-reduce inference pipeline, etc. PinSage is shown to be capable of handling a real billion-scale graph. StochasticGCN [73] further proposes to reduce the sampling variances by using historical activations in the last batches as a control variate, allowing for arbitrarily small sample sizes with a theoretical guarantee.

Instead of sampling neighbors of nodes, FastGCN [74] adopts a different strategy to sample nodes in each convolutional layer, i.e. layer-wise sampling, by interpreting nodes as i.i.d. samples and graph convolutions as integral transforms under probability measures. FastGCN also shows that sampling nodes via their normalized degrees can reduce variances and lead to better performance. Adapt [75] further proposes to sample nodes in the lower layers conditioned on their top one, which is more adaptive and applicable for explicit variance reduction.

4.3.5 Inductive Setting

Another important aspect of GCNs is applying to the inductive setting, i.e. training on a set of nodes/graphs and testing on another set of nodes/graphs unseen during training. In principle, this is

achieved by learning a mapping function on the given features, which are not dependent on the graph basis and can be transferred across nodes/graphs. The inductive setting is verified in GraphSAGE [59], GAT [63], GaAN [64], and FastGCN [74]. However, existing inductive GCNs are only suitable for graphs with features. How to conduct inductive learning for graphs without features, usually called the out-of-sample problem [98], largely remains open in the literature.

4.3.6 Theoretical Analysis

To understand the effectiveness of GCNs, some theoretical analysis has been proposed, which can be divided into three categories: node-focused tasks, graph-focused tasks, and general analysis.

For node-focused tasks, Li et al. [76] first analyze the performance of GCNs as a special form of Laplacian smoothing, which makes the features of vertices in the same cluster similar. The original Laplacian smoothing can be formulated as:

$$\mathbf{h}_i' = (1-\gamma)\mathbf{h}_i + \gamma \sum\nolimits_{j\in\mathcal{N}(i)} \frac{1}{d_i}\mathbf{h}_j, \tag{39}$$

where $\mathbf{h}_i$ and $\mathbf{h}_i'$ are the original and smoothed features of node v_i, respectively. We can see that Eq. (39) is very similar to the graph convolution proposed by Kipf and Welling in Eq. (12). Based on this insight, Li et al. also propose a co-training and a self-training method for GCN.

Recently, Wu et al. [77] analyze GCNs from the signal processing perspective. By regarding node features as graph signals, they show that Eq. (12) is basically a fixed low-pass filter. Using this insight, an extremely simplified graph convolution (SGC) is proposed by removing all non-linearities and collapsing learning parameters into one matrix:

$$\mathbf{H}^L = \left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\right)^L \mathbf{F}_V\mathbf{\Theta}. \tag{40}$$

The authors show that such a "non-deep-learning" GCN variant achieves comparable performance in many tasks. [78] enhances this result by showing that the low-pass filtering nature does not equip GCNs with non-linear manifold learning property, and further proposes GFNN to remedy this problem by adding an MLP after the graph convolution layer.

For graph-focused tasks, [49] and [55] both consider the relationship between GCNs and graph kernels such as the Weisfeiler-Lehman (WL) kernel [84], which is widely used in graph isomorphism tests. They show that GCNs are conceptually generalization of the WL kernel since both methods iteratively aggregate information from node neighbors. Xu et al. [79] formalize this idea by proving that WL kernel provides an upper bound for GCNs in terms of distinguishing graph structures. Based on the analysis, Xu et al. propose Graph Isomorphism Network (GIN) and show that a simple readout operation using sum and MLP can achieve provably maximum discriminative power, i.e. training accuracy in graph classification tasks.

For general analysis, Scarselli et al. [99] show the Vapnik Chervonenkis dimension (VC-dim) of GCNs with different activation functions, which are comparable to standard RNNs. Chen et al. [71] analyze the optimization landscape of linear GCNs and show that any local minimum is relatively close to the global minimum under certain simplifications. Verma and Zhang [100] analyze the algorithmic stability and generalization bound of GCNs. They show that single-layer GCNs satisfy the strong notion of uniform stability if the largest absolute eigenvalue of graph convolution filters is independent of the graph size.

5 Graph Autoencoders (GAEs)

Autoencoder (AE) and its variations are widely used for unsupervised learning [101], which are suitable to learn node representations for graphs. The implicit assumption is that graphs have an inherent, potentially non-linear low-rank structure. In this section, we will first elaborate graph autoencoders and then introduce graph variational autoencoders and other improvements. The main characteristics of GAEs are summarized in Table 5.

5.1 Autoencoders

The use of AEs for graphs is originated from Sparse Autoencoder (SAE) [102]. The basic idea is that, by regarding the adjacency matrix or its variations as the raw features of nodes, AEs can be leveraged as a dimension reduction technique to learn low-dimensional node representations. Specifically, SAE adopts the following L2-reconstruction loss:

$$\begin{aligned} &\min_{\mathbf{\Theta}} \mathcal{L}_2 = \sum\nolimits_{i=1}^{N} \left\| \mathbf{P}\left(i,:\right) - \hat{\mathbf{P}}\left(i,:\right) \right\|_2 \\ &\hat{\mathbf{P}}\left(i,:\right) = \mathcal{G}\left(\mathbf{h}_i\right), \mathbf{h}_i = \mathcal{F}\left(\mathbf{P}\left(i,:\right)\right), \end{aligned} \tag{41}$$

where $\mathbf{P}$ is the transition matrix, $\hat{\mathbf{P}}$ is the reconstructed matrix, $\mathbf{h}_i \in \mathbb{R}^d$ is the low-dimensional representation of node v_i, $\mathcal{F}(\cdot)$ is the encoder, $\mathcal{G}(\cdot)$ is the decoder, $d \ll N$ is the dimensionality and $\mathbf{\Theta}$ are parameters. Both the encoder and decoder are multi-layer perceptrons with many hidden layers. In other words, SAE tries to compress the information of $\mathbf{P}(i,:)$ into a low-dimensional vector $\mathbf{h}_i$ and reconstruct the original feature. Another sparsity regularization term is also added. After getting the low-dimensional representation $\mathbf{h}_i$, k-means [112] is applied for the node clustering task, which proves empirically to outperform non-deep learning baselines. However, SAE is based on an incorrect theoretical analysis[7] and the mechanism underlying such effectiveness remains unexplained.

Structure Deep Network Embedding (SDNE) [103] fills in the puzzle by showing that the L2-reconstruction loss in Eq. (41) actually corresponds to the second-order proximity between nodes, i.e. two nodes share similar latten representations if they have similar neighborhoods, which is well studied in network science such as in collaborative filtering or triangle closure [5]. Motivated by network embedding methods, which show that the first-order proximity is also important [114], SDNE modifies the objective function by adding another term similar to the Laplacian Eigenmaps [81]:

$$\min_{\mathbf{\Theta}} \mathcal{L}_2 + \alpha \sum\nolimits_{i,j=1}^{N} \mathbf{A}(i,j) \left\|\mathbf{h}_i - \mathbf{h}_j\right\|_2, \tag{42}$$

i.e. two nodes also need to share similar latten representations if they are directly connected. The authors also modify the L2-reconstruction loss by using the adjacency matrix and assigning different weights to zero and non-zero elements:

$$\mathcal{L}_2 = \sum\nolimits_{i=1}^{N} \left\|\left(\mathbf{A}\left(i,:\right) - \mathcal{G}\left(\mathbf{h}_i\right)\right) \odot \mathbf{b}_i\right\|_2, \tag{43}$$

where $\mathbf{h}_i = \mathcal{F}\left(\mathbf{A}\left(i,:\right)\right)$, $b_{ij} = 1$ if $\mathbf{A}(i,j) = 0$, $b_{ij} = \beta > 1$ else, and β is another hyper-parameter. The overall architecture of SDNE is shown in Figure 8.

Motivated by another line of works, a contemporary work DNGR [104] replaces the transition matrix $\mathbf{P}$ in Eq. (41) with

7. The original paper [102] motivates the problem by analyzing the connection between spectral clustering and Singular Value Decomposition, which is mathematically incorrect as pointed out in [113].

TABLE 5
A Comparison of Different Graph Autoencoders (GAEs)

Method	Type	Objective	Scalability	Node Features	Other Characteristics
SAE [102]	AE	L2-Reconstruction	Yes	No	-
SDNE [103]	AE	L2-Reconstruction + Laplacian Eigenmaps	Yes	No	-
DNGR [104]	AE	L2-Reconstruction	No	No	-
GC-MC [105]	AE	L2-Reconstruction	Yes	Yes	GCN Encoder
DRNE [106]	AE	Recursive Reconstruction	Yes	No	LSTM Encoder
G2G [107]	AE	KL + Ranking	Yes	Yes	Nodes as distributions
VGAE [108]	VAE	Pairwise Probability of Reconstruction	No	Yes	GCN Encoder
DVNE [109]	VAE	Wasserstein + Ranking	Yes	No	Nodes as distributions
ARGA/ARVGA [110]	AE/VAE	L2-Reconstruction + GAN	Yes	Yes	GCN Encoder
NetRA [111]	AE	Recursive Reconstruction + Laplacian Eigenmaps + GAN	Yes	No	LSTM Encoder

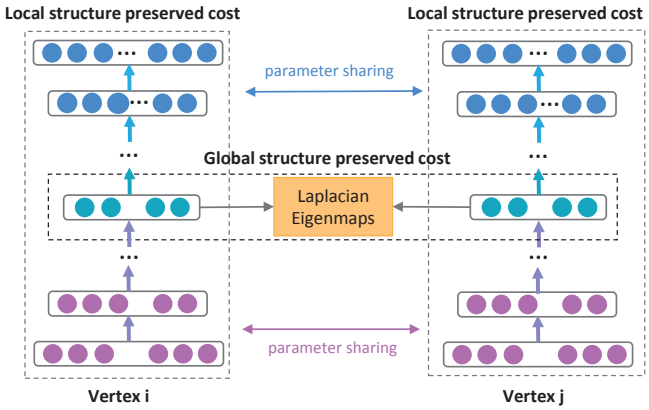

Fig. 8. The framework of SDNE reprinted from [103] with permission. Both the first and second-order proximity of nodes are preserved using deep autoencoders.

the positive pointwise mutual information (PPMI) [85] matrix of a random surfing probability. In this way, the raw features can be associated with some random walk probability of the graph [115]. However, constructing the input matrix takes $O(N^2)$ time complexity, which is not scalable to large-scale graphs.

GC-MC [105] further takes a different approach by using GCN in [49] as the encoder:

$$\mathbf{H} = GCN\left(\mathbf{F}^V, \mathbf{A}\right), \tag{44}$$

and the decoder is a simple bilinear function:

$$\hat{\mathbf{A}}(i,j) = \mathbf{H}(i,:)\mathbf{\Theta}_{de}\mathbf{H}(j,:)^T, \tag{45}$$

where $\mathbf{\Theta}_{de}$ are parameters for the encoder. In this way, node features can be naturally incorporated. For graphs without node features, a one-hot encoding of node IDs can be utilized. The authors demonstrate the effectiveness of GC-MC on the recommendation problem of bipartite graphs.

Instead of reconstructing the adjacency matrix or its variations, DRNE [106] proposes another modification to directly reconstruct the low-dimensional vectors of nodes by aggregating neighborhood information using LSTM. Specifically, DRNE minimizes the following objective function:

$$\mathcal{L} = \sum_{i=1}^{N} \left\| \mathbf{h}_i - \text{LSTM}\left(\{\mathbf{h}_j | j \in \mathcal{N}(i)\}\right) \right\|. \tag{46}$$

Since LSTM requires the inputs to be a sequence, the authors suggest ordering the neighborhoods of nodes according to their degrees. A sampling of neighbors is also adopted for nodes with large degrees to prevent the memory from being too long. The authors prove that such a method can preserve regular equivalence and many centrality measures of nodes such as PageRank [116].

Unlike the existing works that map nodes into low-dimensional vectors, Graph2Gauss (G2G) [107] proposes encoding each node as a Gaussian distribution $\mathbf{h}_i = \mathcal{N}\left(\mathbf{M}(i,:), diag\left(\mathbf{\Sigma}(i,:)\right)\right)$ to capture the uncertainties of nodes. Specifically, the authors use a deep mapping from node attributes to the means and variances of the Gaussian distribution as the encoder:

$$\mathbf{M}(i,:) = \mathcal{F}_{\mathbf{M}}(\mathbf{F}^V(i,:)), \mathbf{\Sigma}(i,:) = \mathcal{F}_{\mathbf{\Sigma}}(\mathbf{F}^V(i,:)), \tag{47}$$

where $\mathcal{F}_{\mathbf{M}}(\cdot)$ and $\mathcal{F}_{\mathbf{\Sigma}}(\cdot)$ are parametric functions need to be learned. Then, instead of using an explicit decoder function, they use pairwise constraints to learn the model:

$$\begin{aligned} &\text{KL}\left(\mathbf{h}_j || \mathbf{h}_i\right) < \text{KL}\left(\mathbf{h}_{j'} || \mathbf{h}_i\right) \\ &\forall i, \forall j, \forall j' \; s.t. \; d(i,j) < d(i,j'), \end{aligned} \tag{48}$$

where $d(i,j)$ is the shortest distance from node v_i to v_j and $\text{KL}[q(\cdot)||p(\cdot)]$ is the Kullback-Leibler (KL) divergence between $q(\cdot)$ and $p(\cdot)$ [117]. In other words, the constraints ensure that KL-divergence between node pairs has the same relative order as the graph distance. However, since Eq. (48) is hard to optimize, an energy-based loss [118] is resorted to as relaxation:

$$\mathcal{L} = \sum\nolimits_{(i,j,j') \in \mathcal{D}} \left(E_{ij}^2 + \exp^{-E_{ij'}}\right), \tag{49}$$

where $\mathcal{D} = \{(i,j,j') | d(i,j) < d(i,j')\}$ and $E_{ij} = \text{KL}(\mathbf{h}_j || \mathbf{h}_i)$. An unbiased sampling strategy is further proposed to accelerate the training process.

5.2 Variational Autoencoders

Different from the aforementioned autoencoders, Variational Autoencoder (VAE) is another type of deep learning method that combines dimension reduction with generative models, potential benefits including tolerating noise and learning smooth representations [119]. VAE was first introduced into modeling graph data in [108], where the decoder is a simple linear product:

$$p\left(\mathbf{A} | \mathbf{H}\right) = \prod\nolimits_{i,j=1}^{N} \sigma\left(\mathbf{h}_i \mathbf{h}_j^T\right), \tag{50}$$

where $\mathbf{h}_i$ are assumed to follow a Gaussian posterior distribution $q\left(\mathbf{h}_i | \mathbf{M}, \mathbf{\Sigma}\right) = \mathcal{N}\left(\mathbf{h}_i | \mathbf{M}(i,:), diag\left(\mathbf{\Sigma}(i,:)\right)\right)$. For the encoder of mean and variance matrices, the authors adopt GCN in [49]:

$$\mathbf{M} = GCN_{\mathbf{M}}\left(\mathbf{F}^V, \mathbf{A}\right), \log \mathbf{\Sigma} = GCN_{\mathbf{\Sigma}}\left(\mathbf{F}^V, \mathbf{A}\right). \tag{51}$$

Then, the model parameters can be learned by minimizing the variational lower bound [119]:

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{H}|\mathbf{F}^V, \mathbf{A})}\left[\log p\left(\mathbf{A} | \mathbf{H}\right)\right] - \text{KL}\left[q\left(\mathbf{H} | \mathbf{F}^V, \mathbf{A}\right) || p(\mathbf{H})\right]. \tag{52}$$

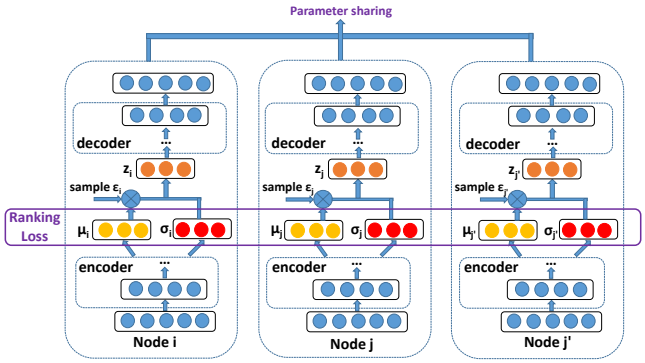

Fig. 9. The framework of DVNE reprinted from [109] with permission. DVNE represents nodes as distributions using VAE and adopts Wasserstein distance to preserve the transitivity of nodes similarity.

However, since the full graph needs to be reconstructed, the time complexity is $O(N^2)$.

Motivated by SDNE and G2G, DVNE [109] proposes another VAE for graph data also by representing each node as a Gaussian distribution. Unlike the existing works that adopt KL-divergence as the measurement, DVNE uses Wasserstein distance [120] to preserve the transitivity of nodes' similarities. Similar to SDNE and G2G, DVNE also preserves both the first and second-order proximity in the objective function:

$$\min_{\mathbf{\Theta}} \sum\nolimits_{(i,j,j')\in\mathcal{D}} \left(E_{ij}^2 + \exp^{-E_{ij'}}\right) + \alpha\mathcal{L}_2, \tag{53}$$

where $E_{ij} = W_2\left(\mathbf{h}_j||\mathbf{h}_i\right)$ is the 2^{nd} Wasserstein distance between two Gaussian distributions $\mathbf{h}_j$ and $\mathbf{h}_i$ and $\mathcal{D} = \{(i,j,j')|j \in \mathcal{N}(i), j' \notin \mathcal{N}(i)\}$ is a set of all triples corresponding to the ranking loss of the first-order proximity. The reconstruction loss is defined as:

$$\mathcal{L}_2 = \inf_{q(\mathbf{Z}|\mathbf{P})} \mathbb{E}_{p(\mathbf{P})}\mathbb{E}_{q(\mathbf{Z}|\mathbf{P})} \left\|\mathbf{P} \odot (\mathbf{P} - \mathcal{G}(\mathbf{Z}))\right\|_2^2, \tag{54}$$

where $\mathbf{P}$ is the transition matrix and $\mathbf{Z}$ are samples drawn from $\mathbf{H}$. The framework is shown in Figure 9. Then, the objective function can be minimized as conventional VAEs using the reparameterization trick [119].

5.3 Improvements and Discussions

Besides these two main categories, there are also several improvements that are worthy of discussion.

5.3.1 Adversarial Training

The adversarial training scheme[8] is incorporated into GAEs as an additional regularization term in [110]. The overall architecture is shown in Figure 10. Specifically, the encoder of GAEs is used as the generator, and the discriminator aims to distinguish whether a latent representation comes from the generator or a prior distribution. In this way, the autoencoder is forced to match the prior distribution as regularization. The objective function is:

$$\min_{\mathbf{\Theta}} \mathcal{L}_2 + \alpha\mathcal{L}_{GAN}, \tag{55}$$

where $\mathcal{L}_2$ is similar to the reconstruction loss defined in GAEs, and $\mathcal{L}_{GAN}$ is

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \mathbb{E}_{\mathbf{h}\sim p_{\mathbf{h}}} \left[\log \mathcal{D}(\mathbf{h})\right] + \mathbb{E}_{\mathbf{z}\sim\mathcal{G}(\mathbf{F}^V,\mathbf{A})} \left[\log\left(1 - \mathcal{D}\left(\mathbf{z}\right)\right)\right], \tag{56}$$

8. We will discuss adversarial methods for graphs in general in Section 7.

where $\mathcal{G}\left(\mathbf{F}^V, \mathbf{A}\right)$ is a generator using the graph convolutional encoder in Eq. (51), $\mathcal{D}(\cdot)$ is a discriminator with the cross-entropy loss and $p_{\mathbf{h}}$ is the prior distribution. In the paper, a simple Gaussian prior is adopted and experimental results demonstrate the effectiveness of the adversarial training scheme.

Concurrently, NetRA [111] also proposes using GAN to enhance the generalization ability of graph autoencoders. Specifically, the authors use the following objective function:

$$\min_{\mathbf{\Theta}} \mathcal{L}_2 + \alpha_1\mathcal{L}_{LE} + \alpha_2\mathcal{L}_{GAN}, \tag{57}$$

where $\mathcal{L}_{LE}$ is the Laplacian Eigenmaps objective function as in Eq. (42). In addition, the authors adopt LSTM as the encoder to aggregate information from neighborhoods similar to Eq. (46). Instead of only sampling immediate neighbors and ordering nodes using degrees as DRNE [106], the authors use random walks to generate the input sequences for LSTM. NetRA considers GAEs as the ground-truth and adopts random Gaussian noise followed by a small fully connected network as the generator, just the opposite to ARGA.

5.3.2 Inductive Learning and GCN encoder

Similar to GCNs, GAEs can be applied to the inductive setting if node attributes are incorporated in the encoder. This can be achieved by using GCNs as the encoder such as in [105], [108], [110], or directly learning a mapping function from node features as in [107]. Since the edge information is only utilized in learning the parameters, the model can be applied to nodes unseen during training. These works also show that, although GCNs and GAEs are based on different architectures, it is possible to use them jointly, which we believe is a promising future direction.

5.3.3 Similarity Measures

In GAEs, many similarity measures are adopted, for example, L2-reconstruction loss, Laplacian Eigenmaps, the ranking loss for graph AEs, and KL divergence and Wasserstein distance for graph VAEs. Although these similarity measures are based on different motivations, how to choose an appropriate similarity measure for a given task and model architecture remains unstudied. More research to understand the underlying differences between these metrics is needed.

6 Graph Reinforcement Learning

One aspect of deep learning that has not been discussed so far is reinforcement learning (RL), which has been shown effective in AI tasks such as game playing [127]. RL is known to be good at learning from feedbacks, especially handling non-differentiable objectives and constraints. In this section, we review Graph RL methods, whose main characteristics are summarized in Table 6.

GCPN [121] utilizes RL for goal-directed molecular graph generations to deal with non-differential objectives and constraints. Specifically, the graph generation is modeled as a Markov decision process of adding nodes and edges, and the generative model is regarded as an RL agent operating in the graph generation environment. By resembling agent actions as link predictions, using domain-specific as well as adversarial rewards and using GCNs to learn node representations, GCPN can be trained end-to-end using policy gradient [128].

A concurrent work, MolGAN [122], takes a similar idea of using RL for generating molecular graphs. Instead of generating

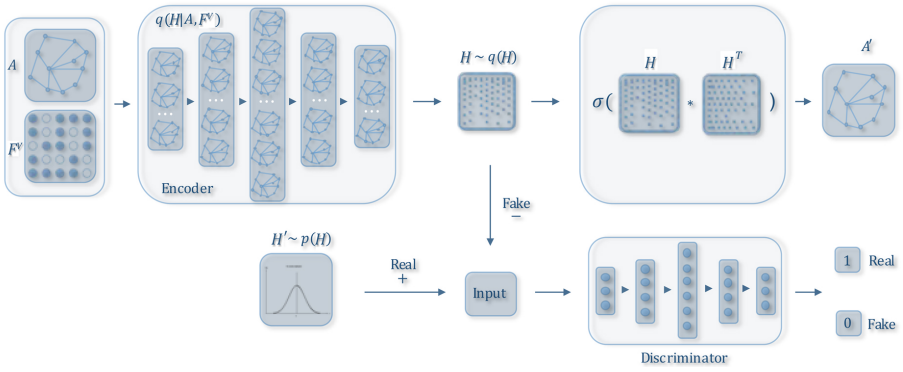

Fig. 10. The framework of ARGA/ARVGA reprinted from [110] with permission. The adversarial training scheme is incorporated into GAEs.

TABLE 6
The Main Characteristics of Graph Reinforcement Learning

Method	Task	Actions	Rewards	Scalability
GCPN [121]	Graph generation	Link prediction	GAN + domain knowledge	No
MolGAN [122]	Graph generation	Generate the whole graph	GAN + domain knowledge	No
GTPN [123]	Chemical reaction prediction	Predict node pairs and new bonds	Prediction results	No
GAM [124]	Graph classification	Predict graph labels and select the next node	Classification results	Yes
DeepPath [125]	Knowledge graph reasoning	Predict the next node of the reasoning path	Reasoning results + diversity	Yes
MINERVA [126]	Knowledge graph reasoning	Predict the next node of the reasoning path	Reasoning results	Yes

the graph by a sequence of actions, MolGAN proposes to directly generate the full graph, working well for small molecules.

GTPN [123] adopts RL for predicting chemical reaction products. Specifically, the agent acts to select node pairs in the molecule graph and predict their new bond types, while the reward is given both immediately and in the end based on whether the predictions are correct. GTPN also adopts a GCN to learn node representations and an RNN to memorize the prediction sequence.

GAM [124] applies RL into graph classification by using random walks to classify graphs and modeling the generation of random walks as a Partially Observable Markov Decision Process. The agent performs two actions: predicting the label of the graph and selecting the next node in the random walk. The reward simply is whether the agent correctly classifies the graph, i.e.

$$\mathcal{J}(\theta) = \mathbb{E}_{P(S_{1:T};\theta)}\left[\sum_{t=1}^{T} r_t\right], \tag{58}$$

where $r_t = 1$ stands for a correct prediction, $r_t = -1$ otherwise, T is the total time steps, and S_t is the environment.

DeepPath [125] and MINERVA [126] both adopt RL for knowledge graph (KG) reasoning. Specifically, DeepPath aims at pathfinding, i.e. find the most informative path between two target nodes, while MINERVA tackles question answering, i.e. find the correct answer node given a question node and a relation. RL agents in both methods need to output a reasoning path in the KG by predicting the next node in the path at each step and receive rewards if the paths reach the correct destinations. DeepPath also adds a regularization term to encourage the diversity of the paths.

7 Graph Adversarial Methods

Adversarial methods such as Generative Adversarial Networks (GANs) [136] or adversarial attacks have drawn increasing attention in the machine learning community in the past few years. In this section, we review how to apply adversarial methods to graphs, whose main characteristics are summarized in Table 7.

7.1 Adversarial training

The basic idea of GAN is to build two linked models: a discriminator and a generator. The goal of the generator is to "fool" the discriminator by generating fake data, while the discriminator aims to distinguish whether a sample comes from real data or is generated by the generator. Then, both models can benefit from each other by joint training using a minimax game. Adversarial training is shown effective in generative models and enhancing the generalization ability of discriminative models. In Section 5.3.1 and Section 6, we have reviewed how to use adversarial training schemes in GAEs and Graph RL, respectively. Here, we review several other adversarial training methods on graphs in detail.

GraphGAN [129] proposes using GAN to enhance graph embedding methods [17] with the following objective function:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \sum_{i=1}^{N} \left(\mathbb{E}_{v\sim p_{graph}(\cdot|v_i)}\left[\log \mathcal{D}(v, v_i)\right] + \mathbb{E}_{v\sim \mathcal{G}(\cdot|v_i)}\left[\log\left(1 - \mathcal{D}\left(v, v_i\right)\right)\right]\right). \tag{59}$$

The discriminator $\mathcal{D}(\cdot)$ and the generator $\mathcal{G}(\cdot)$ are:

$$\mathcal{D}(v, v_i) = \sigma(\mathbf{d}_v \mathbf{d}_{v_i}^T), \mathcal{G}(v|v_i) = \frac{\exp(\mathbf{g}_v \mathbf{g}_{v_i}^T)}{\sum_{v' \neq v_i} \exp(\mathbf{g}_{v'} \mathbf{g}_{v_i}^T)}, \tag{60}$$

where $\mathbf{d}_v$ and $\mathbf{g}_v$ are low-dimensional embedding vectors for node v in the discriminator and the generator, respectively. Combining the above equations, the discriminator actually has two objectives: node pairs in the original graph should have large similarities and node pairs generated by the generator should have small similarities. Such architecture is similar to network embedding methods such as LINE [114], except that we use the generator $\mathcal{G}(\cdot)$ to generate negative node pairs instead of random sampling. The authors show that adversarial training can enhance the inference ability of node embedding vectors.

Meanwhile, Adversarial Network Embedding (ANE) [130] also proposes using adversarial training to improve network embedding methods. Similar to ARGA [110], ANE uses GAN as an

TABLE 7
The Main Characteristics of Graph Adversarial Methods

Category	Method	Adversarial Methods	Scalability	Node Features
Adversarial Training	ARGA/ARVGA [110]	Regularization for GAEs	Yes	Yes
	NetRA [111]	Regularization for GAEs	Yes	No
	GCPN [121]	Rewards for Graph RL	No	Yes
	MolGAN [122]	Rewards for Graph RL	No	Yes
	GraphGAN [129]	Generate negative samples (node pairs)	Yes	No
	ANE [130]	Regularization for network embedding	No	No
	GraphSGAN [131]	Enhancing semi-supervised learning on graphs	Yes	Yes
	NetGAN [132]	Generate graphs via random walks	No	No
Adversarial Attack	Nettack [133]	Targeted attacks of graph structures and node attributes	Yes	Yes
	Dai et al. [134]	Targeted attacks of graph structures	Yes	No
	Zugner and Gunnemann [135]	Non-targeted attacks of graph structures	No	Yes

additional regularization to existing network embedding methods, such as DeepWalk [137] by imposing a prior distribution as real data and regarding embedding vectors as generated samples.

GraphSGAN [131] proposes using GAN to enhance semi-supervised learning on graphs. Specifically, GraphSGAN observes that fake nodes should be generated in the density gap between subgraphs so that the propagation effect of existing models across different clusters in the graph is weakened. To achieve that, the authors design a new optimization objective with several sophisticated loss terms to ensure that the generator generates samples in density gaps at equilibrium.

NetGAN [132] adopts GAN for generating graphs. Specifically, the authors regard graph generation as learning the distribution of biased random walks and propose a GAN framework for generating and discriminating random walks using LSTM. Experiments show that such a generative model through random walks can also learn global network patterns.

7.2 Adversarial attacks

Adversarial attacks are another class of adversarial methods to deliberately "fool" targeted methods by adding small perturbations to data. Studying adversarial attacks can subsequently deepen our understanding of existing models and inspire more robust architectures. Next, we review the graph-based adversarial attacks.

Nettack [133] first proposes attacking node classification models such as GCNs by modifying graph structures and node attributes. Specifically, denoting the targeted node as v_0, its true class as c_{true}, the targeted model as $\mathcal{F}(\mathbf{A}, \mathbf{F}^V)$, and its loss function as $\mathcal{L}_{\mathcal{F}}(\mathbf{A}, \mathbf{F}^V)$, the following objective function is adopted:

$$\begin{aligned} &\underset{(\mathbf{A}', \mathbf{F}^{V\prime}) \in \mathcal{P}}{\operatorname{argmax}} \max_{c \neq c_{true}} \log \mathbf{Z}^*_{v_0,c} - \log \mathbf{Z}^*_{v_0,c_{true}} \\ &s.t.\ \mathbf{Z}^* = \mathcal{F}_{\theta^*}(\mathbf{A}', \mathbf{F}^{V\prime}), \theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}_{\mathcal{F}}(\mathbf{A}', \mathbf{F}^{V\prime}), \end{aligned} \tag{61}$$

where $\mathbf{A}', \mathbf{F}^{V\prime}$ are the modified adjacency matrix and node feature matrix, $\mathbf{Z}$ are classification probabilities predicted by $\mathcal{F}(\cdot)$ and $\mathcal{P}$ is the space determined by the attack constrains. Simply speaking, the optimization aims to find the best legit changes in graph structures and node attributes so that v_0 is misclassified and θ^* indicates that the attack is causative, i.e. the attack is before training the targeted model. The authors propose several constraints for the attacks. The most important constraint is that the attack should be "unnoticeable", i.e. only small changes should be added. Specifically, the authors propose to preserve data characteristics including node degree distributions and feature co-occurrences. Two attacking scenarios, namely direct attack (directly attacking v_0) and influence attack (only attacking other nodes), and several relaxations to make the optimization tractable are also proposed.

Concurrently, Dai et al. [134] also study adversarial attacks for graphs with a similar objective function as Eq. (61), but focus on the case of only changing graph structures. Instead of assuming that the attacker has all the information, the authors consider several settings with different amounts of information available. The most effective strategy, RL-S2V, adopts structure2vec [138] to learn node and graph representations and uses reinforcement learning to solve the optimization. Experimental results show that the attack is effective for both node and graph classification tasks.

The aforementioned two attacks are targeted, i.e. to misclassify some targeted node v_0. Zugner and Gunnemann [135] first study non-targeted attacks, i.e. aiming to reduce the global performance of the model, by treating the graph structure as hyper-parameters to optimize and using meta-gradients in the optimization. Several techniques are utilized to approximate the meta-gradients.

8 Discussion and Conclusion

So far, we have reviewed the different architectures of graph-based deep learning methods as well as their differences and connections. Next, we briefly discuss their applications, implementations, and future directions before we summarize the paper.

8.1 Applications

Besides standard graph inference tasks such as node or graph classification[9], graph-based deep learning methods have also been applied to a wide range of disciplines, such as modeling social influence [139], recommendation [29], [72], [105], [140], chemistry and biology [52], [58], [61], [121], [122], physics [141], [142], disease or drug prediction [143]–[145], gene expression [146], natural language processing (NLP) [147], [148], computer vision [149]–[153], traffic forecasting [154], [155], program induction [156] and solving graph-based NP problems [157], [158].

Though a thorough review of these methods is beyond the scope of this paper due to the diversity of these applications, we list several key inspirations. First, it is important to incorporate domain knowledge into the model, e.g. in constructing the graph or choosing architectures. For example, building a graph based on relative distance may be suitable for traffic forecasting problems, but may not work well for a weather prediction problem because the geographical location is also important. Second, the graph-based model can usually be built on top of other architectures rather than working alone. For example, the computer vision

9. We have collected a list of methods for common tasks in the appendix.

community usually adopts CNNs to detect objects and then uses graph-based deep learning as a reasoning module [43]. On the other hand, GCNs can be adopted as syntactic constraints for NLP problems [147], [148]. As a result, how to integrate different models is usually the key challenge. These applications also show that graph-based deep learning not only empowers us to mine the rich value underlying existing graph data but also helps to advance other disciplines by naturally modeling relational data as graphs, greatly widening the applicability of graph-based deep learning.

8.2 Implementations

Recently, there are several open libraries for deep learning on graphs, which we list in Table 8. We also collect a list of the source codes for papers discussed in this paper, mostly by their original authors, in the appendix. These open implementations make it easy to learn, compare, and improve different methods. Some implementations also address the problem of distributed computing, which we do not discuss in this paper.

8.3 Future Directions

There are also several on-going or future directions which are worthy of discussions:

- **New models for unstudied graph structures**. Due to the extremely diverse structures of graph data, the existing methods cannot handle all of them. For example, most methods focus on homogeneous graphs, while heterogeneous graphs are seldom studied, especially those containing different modalities like in [162]. Signed networks, where negative edges represent conflicts between nodes, also have unique structures and pose additional challenges to the existing methods [163]. Hyper-graphs, representing complex relations between more than two objects [164], are also understudied. An important next step is to design specific deep learning models to handle these different types of graphs.
- **Compositionality of existing models**. As shown in many sections, many existing architectures can work together, for example using GCN as a layer in GAEs or Graph RL. Besides designing new building blocks, how to systematically composite these architectures is an interesting future direction. In this process, how to incorporate interdisciplinary knowledge in a principled way rather than case by case is also an open problem. A recent work, Graph Networks [9], takes the first step and focuses on using a general framework of GNNs and GCNs for relational reasoning problems. AutoML may also be helpful by lighting human burdens in assembling different components and choosing hyper-parameters [165].
- **Dynamic graphs**. Most existing methods focus on static graphs. However, many real graphs are dynamic in nature, where nodes, edges and their features can change over time. For example, in social networks, people may establish new social relations, remove old edges and their features like hobbies and occupations can change over time. New users may join the network while old users can leave. How to model the evolving characteristics of dynamic graphs and support incrementally updating model parameters largely remains open in the literature. Some preliminary works try to tackle this problem using Graph RNN architectures with encouraging results [28], [30].
- **Interpretability and Robustness**. Since graphs are often related to other risk-sensitive scenarios, interpreting deep learning models on graphs is critical towards decision-making problems. For example, in medicine or disease-related problems, interpretability is essential in transforming computer experiments into clinical usages. However, interpretability for graph-based deep learning is even more challenging than other black-box models since nodes and edges in the graph are heavily interconnected. In addition, since many existing deep learning models on graphs are sensitive to adversarial attacks as shown in Section 7.2, how to enhance the robustness of existing methods is another important issue. Some pioneering works for interpretability and robustness can be found in [166] and [167], [168], respectively.

8.4 Summary

Our above survey shows that deep learning on graphs is a promising and fast-developing research field, containing exciting opportunities as well as challenges. Studying deep learning on graphs provides a critical building block in modeling relational data, and is an important step towards better machine learning and artificial intelligence eras.

ACKNOWLEDGEMENT

We thank Jianfei Chen, Jie Chen, William L. Hamilton, Wenbing Huang, Thomas Kipf, Federico Monti, Shirui Pan, Petar Velickovic, Keyulu Xu, Rex Ying for providing their figures.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.
[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.
[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.
[4] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proceedings of the 4th International Conference on Learning Representations*, 2015.
[5] A.-L. Barabasi, *Network science*. Cambridge university press, 2016.
[6] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, 2013.
[7] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.
[8] C. Zang, P. Cui, and C. Faloutsos, "Beyond sigmoids: The nettide model for social network growth, and its applications," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 2015–2024.
[9] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.
[10] J. B. Lee, R. A. Rossi, S. Kim, N. K. Ahmed, and E. Koh, "Attention models in graphs: A survey," *arXiv preprint arXiv:1807.07984*, 2018.
[11] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: Algorithms, applications and open challenges," in *International Conference on Computational Social Networks*. Springer, 2018, pp. 79–91.

TABLE 8
Libraries of Deep Learning on Graphs

Name	URL	Language/Framework	Key Characteristics
PyTorch Geometric [159]	https://github.com/rusty1s/pytorch_geometric	PyTorch	Improved efficiency, unified operations, comprehensive existing methods
Deep Graph Library [160]	https://github.com/dmlc/dgl	PyTorch	Improved efficiency, unified operations, large-scale
AliGraph [161]	https://github.com/alibaba/aligraph	Unknown	Distributed, large-scale, in-house algorithms
Euler	https://github.com/alibaba/euler	C++/Tensorflow	Distributed, large-scale

[12] L. Sun, J. Wang, P. S. Yu, and B. Li, "Adversarial attack and defense on graph data: A survey," *arXiv preprint arXiv:1812.10528*, 2018.

[13] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[14] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.

[15] S. Yan, D. Xu, B. Zhang, H.-J. Zhang, Q. Yang, and S. Lin, "Graph embedding and extensions: A general framework for dimensionality reduction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 1, pp. 40–51, 2007.

[16] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[17] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, 1986.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations*, 2014.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2017.

[22] J. Leskovec and J. J. Mcauley, "Learning to discover social circles in ego networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 539–547.

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

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

[25] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," in *Proceedings of the 5th International Conference on Learning Representations*, 2016.

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

[27] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "Graphrnn: Generating realistic graphs with deep auto-regressive models," in *International Conference on Machine Learning*, 2018, pp. 5694–5703.

[28] Y. Ma, Z. Guo, Z. Ren, E. Zhao, J. Tang, and D. Yin, "Dynamic graph neural networks," *arXiv preprint arXiv:1810.10627*, 2018.

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

[30] F. Manessi, A. Rozza, and M. Manzo, "Dynamic graph convolutional networks," *arXiv preprint arXiv:1704.06199*, 2017.

[31] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 2014, pp. 1724–1734.

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

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

[34] P. Frasconi, M. Gori, and A. Sperduti, "A general framework for adaptive processing of data structures," *IEEE transactions on Neural Networks*, vol. 9, no. 5, pp. 768–786, 1998.

[35] M. J. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *The computer journal*, vol. 7, no. 2, pp. 155–162, 1964.

[36] L. B. Almeida, "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment." in *Proceedings, 1st First International Conference on Neural Networks*. IEEE, 1987.

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

[38] M. A. Khamsi and W. A. Kirk, *An introduction to metric spaces and fixed point theory*. John Wiley & Sons, 2011, vol. 53.

[39] M. Brockschmidt, Y. Chen, B. Cook, P. Kohli, and D. Tarlow, "Learning to decipher the heap for program verification," in *Workshop on Constructive Machine Learning at the International Conference on Machine Learning*, 2015.

[40] S. Sukhbaatar, R. Fergus *et al.*, "Learning multiagent communication with backpropagation," in *Advances in Neural Information Processing Systems*, 2016, pp. 2244–2252.

[41] P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," in *Advances in Neural Information Processing Systems*, 2016.

[42] Y. Hoshen, "Vain: Attentional multi-agent predictive modeling," in *Advances in Neural Information Processing Systems*, 2017.

[43] A. Santoro, D. Raposo, D. G. T. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap, "A simple neural network module for relational reasoning," in *Advances in Neural Information Processing Systems*, 2017.

[44] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv preprint arXiv:1803.03324*, 2018.

[45] I. M. Baytas, C. Xiao, X. Zhang, F. Wang, A. K. Jain, and J. Zhou, "Patient subtyping via time-aware lstm networks," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 65–74.

[46] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," in *Proceedings of the 3rd International Conference on Learning Representations*, 2014.

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

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

[49] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the 6th International Conference on Learning Representations*, 2017.

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

[51] B. Xu, H. Shen, Q. Cao, Y. Qiu, and X. Cheng, "Graph wavelet neural network," in *Proceedings of the 8th International Conference on Learning Representations*, 2019.

[52] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in Neural Information Processing Systems*, 2015, pp. 2224–2232.

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

[54] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 1416–1424.

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

[56] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2016.

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

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

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

[60] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *Proceedings of Computer Vision and Pattern Recognition*, vol. 1, no. 2, 2017, p. 3.

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

[62] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in Neural Information Processing Systems*, 2018.

[63] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proceedings of the 7th International Conference on Learning Representations*, 2018.

[64] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, "Gaan: Gated attention networks for learning on large and spatiotemporal graphs," in *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*, 2018.

[65] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *The World Wide Web Conference*. ACM, 2019, pp. 2022–2032.

[66] T. Pham, T. Tran, D. Q. Phung, and S. Venkatesh, "Column networks for collective classification." in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2017, pp. 2485–2491.

[67] J. Klicpera, A. Bojchevski, and S. Günnemann, "Predict then propagate: Graph neural networks meet personalized pagerank," in *Proceedings of the 8th International Conference on Learning Representations*, 2019.

[68] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *International Conference on Machine Learning*, 2018.

[69] M. Simonovsky and N. Komodakis, "Dynamic edgeconditioned filters in convolutional neural networks on graphs," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

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

[71] Z. Chen, L. Li, and J. Bruna, "Supervised community detection with line graph neural networks," in *Proceedings of the 8th International Conference on Learning Representations*, 2019.

[72] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018.

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

[74] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," in *Proceedings of the 7th International Conference on Learning Representations*, 2018.

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

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

[77] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International Conference on Machine Learning*, 2019, pp. 6861–6871.

[78] T. Maehara, "Revisiting graph neural networks: All we have is low-pass filters," *arXiv preprint arXiv:1905.09550*, 2019.

[79] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proceedings of the 8th International Conference on Learning Representations*, 2019.

[80] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax," in *Proceedings of the 8th International Conference on Learning Representations*, 2019.

[81] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Advances in neural information processing systems*, 2002, pp. 585–591.

[82] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[83] Z. Zhang, P. Cui, X. Wang, J. Pei, X. Yao, and W. Zhu, "Arbitrary-order proximity preserved network embedding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2778–2786.

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

[85] O. Levy and Y. Goldberg, "Neural word embedding as implicit matrix factorization," in *Advances in Neural Information Processing Systems*, 2014, pp. 2177–2185.

[86] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1367–1372, 2004.

[87] G. Klir and B. Yuan, *Fuzzy sets and fuzzy logic*. Prentice hall New Jersey, 1995, vol. 4.

[88] J. Ma, P. Cui, X. Wang, and W. Zhu, "Hierarchical taxonomy aware network embedding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 1920–1929.

[89] D. Ruppert, "The elements of statistical learning: Data mining, inference, and prediction," *Journal of the Royal Statistical Society*, vol. 99, no. 466, pp. 567–567, 2010.

[90] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[91] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 11, 2007.

[92] D. I. Shuman, M. J. Faraji, and P. Vandergheynst, "A multiscale pyramid transform for graph signals," *IEEE Transactions on Signal Processing*, vol. 64, no. 8, pp. 2119–2134, 2016.

[93] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," *Proceedings of the 5th International Conference on Learning Representations*, 2016.

[94] B. D. Mckay and A. Piperno, *Practical graph isomorphism, II*. Academic Press, Inc., 2014.

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

[96] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[97] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[98] J. Ma, P. Cui, and W. Zhu, "Depthlgp: Learning embeddings of out-of-sample nodes in dynamic networks," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018.

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

[100] S. Verma and Z.-L. Zhang, "Stability and generalization of graph convolutional neural networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019, pp. 1539–1548.

[101] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *International Conference on Machine Learning*, 2008, pp. 1096–1103.

[102] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu, "Learning deep representations for graph clustering." in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[103] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016, pp. 1225–1234.

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

[105] R. v. d. Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *arXiv preprint arXiv:1706.02263*, 2017.

[106] K. Tu, P. Cui, X. Wang, P. S. Yu, and W. Zhu, "Deep recursive network embedding with regular equivalence," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2357–2366.

[107] A. Bojchevski and S. Günnemann, "Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking," in *Proceedings of the 7th International Conference on Learning Representations*, 2018.

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

[109] D. Zhu, P. Cui, D. Wang, and W. Zhu, "Deep variational network embedding in wasserstein space," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2827–2836.

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

[111] W. Yu, C. Zheng, W. Cheng, C. C. Aggarwal, D. Song, B. Zong, H. Chen, and W. Wang, "Learning deep network representations with adversarially regularized autoencoders," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2663–2671.

[112] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA, 1967, pp. 281–297.

[113] Z. Zhang, "A note on spectral clustering and svd of graph data," *arXiv preprint arXiv:1809.11029*, 2018.

[114] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.

[115] L. Lovász *et al.*, "Random walks on graphs: A survey," *Combinatorics*, vol. 2, no. 1, pp. 1–46, 1993.

[116] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[117] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[118] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang, "A tutorial on energy-based learning," *Predicting structured data*, 2006.

[119] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proceedings of the 3rd International Conference on Learning Representations*, 2014.

[120] S. Vallender, "Calculation of the wasserstein distance between probability distributions on the line," *Theory of Probability & Its Applications*, vol. 18, no. 4, pp. 784–786, 1974.

[121] J. You, B. Liu, R. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *Advances in Neural Information Processing Systems*, 2018.

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

[123] K. Do, T. Tran, and S. Venkatesh, "Graph transformation policy network for chemical reaction prediction," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019, pp. 750–760.

[124] J. B. Lee, R. Rossi, and X. Kong, "Graph classification using structural attention," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 1666–1674.

[125] W. Xiong, T. Hoang, and W. Y. Wang, "Deeppath: A reinforcement learning method for knowledge graph reasoning," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. ACL, 2017.

[126] R. Das, S. Dhuliawala, M. Zaheer, L. Vilnis, I. Durugkar, A. Krishnamurthy, A. Smola, and A. McCallum, "Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning," in *Proceedings of the 7th International Conference on Learning Representations*, 2018.

[127] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[128] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing systems*, 2000.

[129] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "Graphgan: Graph representation learning with generative adversarial nets," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[130] Q. Dai, Q. Li, J. Tang, and D. Wang, "Adversarial network embedding," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[131] M. Ding, J. Tang, and J. Zhang, "Semi-supervised learning on graphs with generative adversarial nets," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, 2018, pp. 913–922.

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

[133] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2847–2856.

[134] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, "Adversarial attack on graph structured data," in *Proceedings of the 35th International Conference on Machine Learning*, 2018, pp. 1115–1124.

[135] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," in *Proceedings of the 8th International Conference on Learning Representations*, 2019.

[136] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014.

[137] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 701–710.

[138] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *International Conference on Machine Learning*, 2016, pp. 2702–2711.

[139] J. Qiu, J. Tang, H. Ma, Y. Dong, K. Wang, and J. Tang, "Deepinf: Modeling influence locality in large social networks," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018.

[140] J. Ma, C. Zhou, P. Cui, H. Yang, and W. Zhu, "Learning disentangled representations for recommendation," in *Advances in Neural Information Processing Systems*, 2019.

[141] C. W. Coley, R. Barzilay, W. H. Green, T. S. Jaakkola, and K. F. Jensen, "Convolutional embedding of attributed molecular graphs for physical property prediction," *Journal of chemical information and modeling*, vol. 57, no. 8, pp. 1757–1772, 2017.

[142] T. Xie and J. C. Grossman, "Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties," *Physical Review Letters*, vol. 120, no. 14, p. 145301, 2018.

[143] S. I. Ktena, S. Parisot, E. Ferrante, M. Rajchl, M. Lee, B. Glocker, and D. Rueckert, "Distance metric learning using graph convolutional networks: Application to functional brain networks," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2017, pp. 469–477.

[144] M. Zitnik, M. Agrawal, and J. Leskovec, "Modeling polypharmacy side effects with graph convolutional networks," *arXiv preprint arXiv:1802.00543*, 2018.

[145] S. Parisot, S. I. Ktena, E. Ferrante, M. Lee, R. G. Moreno, B. Glocker, and D. Rueckert, "Spectral graph convolutions for population-based disease prediction," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2017.

[146] F. Dutil, J. P. Cohen, M. Weiss, G. Derevyanko, and Y. Bengio, "Towards gene expression convolutions using gene interaction graphs," in *International Conference on Machine Learning Workshop on Computational Biology*, 2018.

[147] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, and K. Simaan, "Graph convolutional encoders for syntax-aware neural machine translation," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 1957–1967.

[148] D. Marcheggiani and I. Titov, "Encoding sentences with graph convolutional networks for semantic role labeling," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 1506–1515.

[149] V. Garcia and J. Bruna, "Few-shot learning with graph neural networks," in *Proceedings of the 7th International Conference on Learning Representations*, 2018.
[150] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-rnn: Deep learning on spatio-temporal graphs," in *Computer Vision and Pattern Recognition*, 2016, pp. 5308–5317.
[151] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun, "3d graph neural networks for rgbd semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
[152] K. Marino, R. Salakhutdinov, and A. Gupta, "The more you know: Using knowledge graphs for image classification," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 20–28.
[153] S. Qi, W. Wang, B. Jia, J. Shen, and S.-C. Zhu, "Learning human-object interactions by graph parsing neural networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 401–417.
[154] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 2018.
[155] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *Proceedings of the 7th International Conference on Learning Representations*, 2018.
[156] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," in *Proceedings of the 7th International Conference on Learning Representations*, 2018.
[157] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *Advances in Neural Information Processing Systems*, 2018, pp. 536–545.
[158] M. O. Prates, P. H. Avelar, H. Lemos, L. Lamb, and M. Vardi, "Learning to solve np-complete problems-a graph neural network for the decision tsp," *arXiv preprint arXiv:1809.02721*, 2018.
[159] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
[160] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma, Z. Huang, Q. Guo, H. Zhang, H. Lin, J. Zhao, J. Li, A. J. Smola, and Z. Zhang, "Deep graph library: Towards efficient and scalable deep learning on graphs," *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
[161] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, "Aligraph: A comprehensive graph neural network platform," in *Proceedings of the 45th International Conference on Very Large Data Bases*, 2019.
[162] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang, "Heterogeneous network embedding via deep architectures," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 119–128.
[163] T. Derr, Y. Ma, and J. Tang, "Signed graph convolutional network," in *Data Mining (ICDM), 2018 IEEE International Conference on*. IEEE, 2018, pp. 559–568.
[164] K. Tu, P. Cui, X. Wang, F. Wang, and W. Zhu, "Structural deep embedding for hyper-networks," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
[165] K. Tu, J. Ma, P. Cui, J. Pei, and W. Zhu, "Autone: Hyperparameter optimization for massive network embedding," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: ACM, 2019, pp. 216–225. [Online]. Available: http://doi.acm.org/10.1145/3292500.3330848
[166] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnn explainer: A tool for post-hoc explanation of graph neural networks," in *Advances in Neural Information Processing Systems*, 2019.
[167] D. Zhu, Z. Zhang, P. Cui, and W. Zhu, "Robust graph convolutional networks against adversarial attacks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019, pp. 1399–1407.
[168] M. Jin, H. Chang, W. Zhu, and S. Sojoudi, "Power up! robust graph convolutional network against evasion attacks based on graph powering," *arXiv preprint arXiv:1905.10029*, 2019.
[169] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *Relational Representation Learning Workshop, NeurIPS 2018*, 2018.
[170] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

PLACE PHOTO HERE

Ziwei Zhang received his B.S. from the Department of Physics, Tsinghua University in 2016. He is currently pursuing the Ph.D. Degree in the Department of Computer Science and Technology at Tsinghua University. His research interests focus on network embedding and machine learning on graph data, especially developing scalable algorithms for large-scale networks. He has published several papers in prestigious conferences and journals, including KDD, AAAI, IJCAI, and TKDE.

PLACE PHOTO HERE

Peng Cui received the PhD degree from Tsinghua University, in 2010. He is an associate professor with tenure at Tsinghua University. His research interests include network representation learning, human behavioral modeling, and socialsensed multimedia computing. He has published more than 60 papers in prestigious conferences and journals in data mining and multimedia. His recent research received the SIGKDD 2016 Best Paper Finalist, ICDM 2015 Best Student Paper Award, SIGKDD 2014 Best Paper Finalist, IEEE ICME 2014 Best Paper Award, ACM MM12 Grand Challenge Multimodal Award, and MMM13 Best Paper Award. He is an associate editor of IEEE Transactions on Knowledge and Data Engineering, the IEEE Transactions on Big Data, the ACM Transactions on Multimedia Computing, Communications, and Applications, the Elsevier Journal on Neurocomputing, etc. He was the recipient of ACM China Rising Star Award in 2015.

PLACE PHOTO HERE

Wenwu Zhu received the PhD degree from New York University, New York, NY, in 1996. He is currently a professor and the vice chair of the Department of Computer Science, Tsinghua University, Beijing, China. Prior to his current position, he was a senior researcher and a research manager with Microsoft Research Asia, Beijing, China. His current research interests include the areas of multimedia computing, communications, and networking, as well as big data. He has been serving as the editor-in-chief for the IEEE Transactions on Multimedia (T-MM) since January 1, 2017. He was the recipient of five Best Paper Awards including T-CSVT in 2001 and ACM Multimedia 2012. He is a fellow of the IEEE, AAAS, SPIE, and an ACM distinguished scientist.

APPENDIX A
SOURCE CODES

We collect and summarize a list of the source codes for papers discussed in this paper, as shown in Table 9. Besides the method name and link, we also list the programming language and frameworks adopted as well as whether they are published by the original authors of the paper.

APPENDIX B
APPLICABILITY IN COMMON TASKS

We summarize the applicability of different models in six common tasks on graphs including node clustering, node classification, network reconstruction, link prediction, graph classification, and graph generation, as shown in Table 10. Note that our results are based on whether the experiments are conducted in the original papers.

APPENDIX C
NODE CLASSIFICATION RESULTS ON BENCHMARK DATASETS

As shown in Appendix Section B, node classification is the most common task for graph-based deep learning models. Here, we report the results of different methods on five node classification benchmark datasets[10]:

- Cora, Citeseer, PubMed [170]: These are citation graphs with nodes representing papers, edges representing citations between papers, and papers associated with bag-of-words features and ground-truth topics as labels.
- Reddit [59]: Reddit is an online discussion forum where nodes stand for posts, two nodes are connected if they are commented by the same user, and each post contains a low-dimensional word vector as features and a label indicating its community.
- PPI [59]: PPI is a collection of protein-protein interaction graphs of different human tissues with features representing biological signatures and labels representing the roles of proteins.

In Cora, Citeseer, Pubmed, and Reddit, there exists one graph and the same graph structure is used in both training and testing, thus the tasks are considered transductive. In PPI, since training and testing nodes are in different graphs, it is considered as an inductive node classification benchmark.

In Table 11, we report the results of different models on these benchmark datasets. The results are extracted from their original papers when a fixed dataset split is adopted. The table shows that many state-of-the-art methods achieve roughly comparable performance in these benchmarks, with differences smaller than one percent. Shchur et al. also find that a fixed dataset split can easily result in spurious comparisons [169]. As a result, though these benchmarks are widely adopted to compare different models, more comprehensive evaluation setups are critically needed.

10. Publicly available at https://github.com/tkipf/gcn or http://snap.stanford.edu/graphsage/.

TABLE 9
A collection of published source codes. O.A. = Original Authors

Category	Method	Urls	O.A.	Language/Framework
Graph RNNs	GGS-NNs [25]	https://github.com/yujiali/ggnn	Yes	Lua/Torch
	SSE [26]	https://github.com/Hanjun-Dai/steady_state_embedding	Yes	C
	CommNet [40]	https://github.com/facebookresearch/CommNet	Yes	Lua/Torch
	Interaction Network [41]	https://github.com/jaesik817/Interaction-networks_tensorflow	No	Python/Tensorflow
	Relation Networks [43]	https://github.com/kimhc6028/relational-networks	No	Python/Pytorch
	You et al. [27]	https://github.com/JiaxuanYou/graph-generation	Yes	Python/Pytorch
	RMGCNN [29]	https://github.com/fmonti/mgcnn	Yes	Python/Tensorflow
GCNs	ChebNet [48]	https://github.com/mdeff/cnn_graph	Yes	Python/Tensorflow
	Kipf&Welling [49]	https://github.com/tkipf/gcn	Yes	Python/Tensorflow
	GWNN [51]	https://github.com/Eilene/GWNN	Yes	Python/Tensorflow
	Neural FPs [52]	https://github.com/HIPS/neural-fingerprint	Yes	Python
	PATCHY-SAN [53]	https://github.com/seiya-kumada/patchy-san	No	Python
	SortPooling [55]	https://github.com/muhanzhang/DGCNN	Yes	Lua/Torch
	LGCN [54]	https://github.com/divelab/lgcn/	Yes	Python/Tensorflow
	DCNN [56]	https://github.com/jcatw/dcnn	Yes	Python/Theano
	DGCN [57]	https://github.com/ZhuangCY/Coding-NN	Yes	Python/Theano
	MPNNs [58]	https://github.com/brain-research/mpnn	Yes	Python/Tensorflow
	GraphSAGE [59]	https://github.com/williamleif/GraphSAGE	Yes	Python/Tensorflow
	GNs [9]	https://github.com/deepmind/graph_nets	Yes	Python/Tensorflow
	DiffPool [62]	https://github.com/RexYing/graph-pooling	Yes	Python/Pytorch
	GAT [63]	https://github.com/PetarV-/GAT	Yes	Python/Tensorflow
	HAN [65]	https://github.com/Jhy1993/HAN	Yes	Python/Tensorflow
	CLN [66]	https://github.com/trangptm/Column_networks	Yes	Python/Keras
	JK-Nets [68]	https://github.com/mori97/JKNet-dgl	No	Python/DGL
	PPNP [67]	https://github.com/klicperajo/ppnp	Yes	Python/Tensorflow
	ECC [69]	https://github.com/mys007/ecc	Yes	Python/Pytorch
	R-GCNs [70]	https://github.com/tkipf/relational-gcn	Yes	Python/Keras
	LGNN [71]	https://github.com/joanbruna/GNN_community	Yes	Lua/Torch
	StochasticGCN [73]	https://github.com/thu-ml/stochastic_gcn	Yes	Python/Tensorflow
	FastGCN [74]	https://github.com/matenure/FastGCN	Yes	Python/Tensorflow
	Adapt [75]	https://github.com/huangwb/AS-GCN	Yes	Python/Tensorflow
	Li et al. [76]	https://github.com/liqimai/gcn	Yes	Python/Tensorflow
	SGC [77]	https://github.com/Tiiiger/SGC	Yes	Python/Pytorch
	GFNN [78]	https://github.com/gear/gfnn	Yes	Python/Pytorch
	GIN [79]	https://github.com/weihua916/powerful-gnns	Yes	Python/Pytorch
	DGI [80]	https://github.com/PetarV-/DGI	Yes	Python/Pytorch
GAEs	SAE [102]	https://github.com/quinngroup/deep-representations-clustering	No	Python/Keras
	SDNE [103]	https://github.com/suanrong/SDNE	Yes	Python/Tensorflow
	DNGR [104]	https://github.com/ShelsonCao/DNGR	Yes	Matlab
	GC-MC [105]	https://github.com/riannevdberg/gc-mc	Yes	Python/Tensorflow
	DRNE [106]	https://github.com/tadpole/DRNE	Yes	Python/Tensorflow
	G2G [107]	https://github.com/abojchevski/graph2gauss	Yes	Python/Tensorflow
	VGAE [108]	https://github.com/tkipf/gae	Yes	Python/Tensorflow
	DVNE [109]	http://nrl.thumedialab.com	Yes	Python/Tensorflow
	ARGA/ARVGA [110]	https://github.com/Ruiqi-Hu/ARGA	Yes	Python/Tensorflow
	NetRA [111]	https://github.com/chengw07/NetRA	Yes	Python/Pytorch
Graph RLs	GCPN [121]	https://github.com/bowenliu16/rl_graph_generation	Yes	Python/Tensorflow
	MolGAN [122]	https://github.com/nicola-decao/MolGAN	Yes	Python/Tensorflow
	GAM [124]	https://github.com/benedekrozemberczki/GAM	Yes	Python/Pytorhc
	DeepPath [125]	https://github.com/xwhan/DeepPath	Yes	Python/Tensorflow
	MINERVA [126]	https://github.com/shehzaadzd/MINERVA	Yes	Python/Tensorflow
Graph Adversarial Methods	GraphGAN [129]	https://github.com/hwwang55/GraphGAN	Yes	Python/Tensorflow
	GraphSGAN [131]	https://github.com/dm-thu/GraphSGAN	Yes	Python/Pytorch
	NetGAN [132]	https://github.com/danielzuegner/netgan	Yes	Python/Tensorflow
	Nettack [133]	https://github.com/danielzuegner/nettack	Yes	Python/Tensorflow
	Dai et al. [134]	https://github.com/Hanjun-Dai/graph_adversarial_attack	Yes	Python/Pytorch
	Zugner&Gunnemann [135]	https://github.com/danielzuegner/gnn-meta-attack	Yes	Python/Tensorflow
Miscellaneous	structure2vec [138]	https://github.com/Hanjun-Dai/pytorch_structure2vec	Yes	Python/Pytorch
	SGCN [163]	http://www.cse.msu.edu/~derrtyle/	Yes	Python/Pytorch
	Dutil et al. [146]	https://github.com/mila-iqia/gene-graph-conv	Yes	Python/Pytorch
	RGCN [167]	https://github.com/thumanlab/nrlweb	Yes	Python/Tensorflow
	GNN-benchmark [169]	https://github.com/shchur/gnn-benchmark	Yes	Python/Tensorflow

TABLE 10
A Table for Methods of Six Common Tasks

Type	Task		Methods
Node-focused Tasks	Node Clustering		[50], [71], [102]–[104], [107], [109], [110], [129], [130], [162]
	Node Classification	Transductive	[9], [23], [26], [28], [48]–[51], [54], [56], [57], [59], [60] [63]–[68], [70], [71], [73]–[78] [80], [103], [106], [107], [109], [111], [129]–[131], [162], [167]
		Inductive	[26], [54], [59], [63], [64], [68], [73], [77], [78], [80]
	Network Reconstruction		[103], [109], [111], [162]
	Link Prediction		[28], [29], [50], [70], [72], [103], [105], [107]–[111], [129]
Graph-focused Tasks	Graph Classification		[9], [23], [48], [52], [53], [55], [56], [61], [62], [69], [77], [79], [124], [138]
	Graph Generation	Structure-only	[27], [132]
		Structure+features	[44], [121], [122]

TABLE 11
Statistics of benchmark datasets and node classification results of different methods when a fixed dataset split is adopted. - indicates unavailable.

	Cora	Citeseer	Pubmed	Reddit	PPI
Type	Citation	Citation	Citation	Social	Biology
Nodes	2,708	3,327	19,717	232,965	56,944 (24 graphs)
Edges	5,429	4,732	44,338	11,606,919	818,716
Classes	7	6	3	41	121
Features	1,433	3,703	500	602	50
Task	Transductive	Transductive	Transductive	Transductive	Inductive
GCN [49]	81.5	70.3	79.0	-	-
GAT [63]	83.0±0.7	72.5±0.7	79.0±0.3	-	97.3±0.2
GraphSAGE [59]	-	-	-	95.4	61.2
MoNet [60]	81.7±0.5	-	78.8±0.4	-	-
LGCN [54]	83.3±0.5	73.0±0.6	79.5±0.2	-	77.2±0.2
FastGCN [74]	-	-		93.7	-
StochasticGCN [73]	82.0±0.8	70.9±0.2	79.0±0.4	97.9±0.0	96.3±0.1
Adapt [75]	-	-	-	96.3±0.3	-
CayleyNets [50]	81.9±0.7	-	-	-	-
JK-Nets [68]	-	-	-	96.5	97.6±0.7
SSE [26]	-	-	-	-	83.6
GaAN [64]	-	-	-	96.4±0.0	98.7±0.0
DGCN [57]	83.5	72.6	80.0	-	-
GraphSGAN [131]	83.0±1.3	73.1±1.8	-	-	-
DGI [80]	82.3±0.6	71.8±0.7	76.8±0.6	94.0±0.1	63.8±0.2
RGCN [167]	82.8±0.6	71.2±0.5	79.1±0.3	-	-
GWNN [51]	82.8	71.7	79.1	-	-
SGC [77]	81.0±0.0	71.9±0.1	78.9±0.0	94.9	-