

# Deep Learning on Graphs: A Survey

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

**Abstract**—Deep learning has been shown to be 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, substantial research efforts have been devoted to applying deep learning methods to graphs, resulting in beneficial advances in graph analysis techniques. In this survey, we comprehensively review the different types of deep learning methods on graphs. We divide the existing methods into five categories based on their model architectures and training strategies: 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 by following their development history. We also analyze the differences and compositions of different methods. Finally, we briefly outline the applications in which they have been used and discuss potential future research directions.

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

## 1 INTRODUCTION

Over the past decade, deep learning has become the “crown jewel” of 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 from underlying data is well recognized. On the other hand, graphs<sup>1</sup> are ubiquitous in the real world, representing objects and their relationships in varied domains, including social networks, e-commerce networks, biology networks, traffic networks, and so on. Graphs are also known to have complicated structures that can contain rich underlying values [5]. As a result, how to utilize deep learning methods to analyze graph data has attracted considerable research attention over the past few years. This problem is non-trivial because several challenges exist in applying traditional deep learning architectures to graphs:

- **Irregular structures of graphs.** Unlike images, audio, and text, which have a clear grid structure, graphs have irregular structures, making it hard to generalize some of the basic mathematical operations to graphs [6]. For example, defining convolution and pooling operations, which are the fundamental operations in convolutional neural networks (CNNs), for graph data is not straightforward. This problem is often referred to as the **geometric deep learning problem** [7].
- **Heterogeneity and diversity of graphs.** A graph itself can be complicated, containing diverse types and properties. For example, graphs can be heterogeneous or homogenous, weighted or unweighted, and signed or unsigned. In addition, the tasks of graphs also vary widely, ranging from node-focused problems such as node classification and link prediction to graph-focused problems such as graph classification and graph generation. These 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; some well-known examples are social networks and e-commerce networks [8]. Therefore, how to design scalable models, preferably models that have a linear time complexity with respect to the graph size, is a key problem.
- **Incorporating interdisciplinary knowledge.** Graphs are often connected to other disciplines, such as biology, chemistry, and social sciences. This interdisciplinary nature provides both opportunities and challenges: domain knowledge can be leveraged to solve specific problems but integrating domain knowledge can complicate model designs. For example, when generating molecular graphs, the objective function and chemical constraints are often non-differentiable; therefore gradient-based training methods cannot easily be applied.

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

In this paper, we try to fill this knowledge gap by comprehensively 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 and training strategies: 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 of the 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 states at 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 obtain feedbacks on graph tasks while following constraints. **Graph adversarial methods** adopt adversarial

• 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.

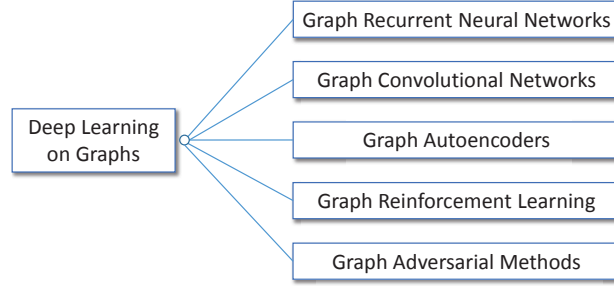


Fig. 1. A categorization of deep learning methods on graphs. We divide the existing methods into five categories: graph recurrent neural networks, graph convolutional networks, graph autoencoders, graph reinforcement learning, and graph adversarial methods.

TABLE 1  
Main Distinctions among Deep Learning Methods on Graphs

Category	Basic Assumptions/Aims	Main Functions
Graph recurrent neural networks	Recursive and sequential patterns of graphs	Definitions 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-based models	Graph adversarial trainings and attacks

training techniques to enhance the generalization ability of graph-based models and test their robustness by adversarial attacks.

In the following sections, we provide a comprehensive and detailed overview of these methods, mainly by following their development history and the various ways these methods solve the challenges posed by graphs. We also analyze the differences between these models and delve into how to composite different architectures. Finally, we briefly outline the applications of these models, introduce several open libraries, and discuss potential future research directions. In the appendix, we provide a source code repository, analyze the time complexity of various methods discussed in the paper, and summarize some common applications.

**Related works.** Several previous surveys are related to our paper. Bronstein *et al.* [7] summarized some early GCN methods as well as CNNs on manifolds and studied them comprehensively through geometric deep learning. Battaglia *et al.* [9] summarized how to use GNNs and GCNs for relational reasoning using a unified framework called graph networks, Lee *et al.* [10] reviewed the attention models for graphs, Zhang *et al.* [11] summarized some GCNs, and Sun *et al.* [12] briefly surveyed adversarial attacks on graphs. Our work differs from these previous 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] surveyed this field from different viewpoints and categorizations. Specifically, neither of their works consider graph reinforcement learning or graph adversarial methods, which are covered in this paper.

Another closely related topic is network embedding, aiming 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 are applied to graphs, and network embedding can be recognized as a concrete application example that uses some of these models (and it uses non-deep-learning methods as well).

The rest of this paper is organized as follows. In Section 2, we introduce the notations used in this paper and provide preliminaries. Then, we review Graph RNNs, GCNs, GAEs, Graph RL,

and graph adversarial methods in Section 3 to 7, respectively. We conclude with a discussion in Section 8.

## 2 NOTATIONS AND PRELIMINARIES

**Notations.** In this paper, a graph<sup>2</sup> is represented as  $G = (V, E)$  where  $V = \{v_1, \dots, v_N\}$  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, whose  $i^{th}$  row,  $j^{th}$  column, and an element are denoted as  $\mathbf{A}(i, :)$ ,  $\mathbf{A}(:, j)$ ,  $\mathbf{A}(i, j)$ , respectively. The graph can be either directed or undirected and weighted or unweighted. In this paper, we mainly consider unsigned graphs; therefore,  $\mathbf{A}(i, j) \geq 0$ . Signed graphs will be discussed in future research directions. We use  $\mathbf{F}^V$  and  $\mathbf{F}^E$  to denote features of nodes and edges, respectively. For other variables, we use bold uppercase characters to denote matrices and bold lowercase characters to denote vectors, e.g., a matrix  $\mathbf{X}$  and a vector  $\mathbf{x}$ . 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 with curlicues, e.g.,  $\mathcal{F}(\cdot)$ .

To better illustrate the notations, we take social networks as an example. Each node  $v_i \in V$  corresponds to a user, and the edges  $E$  correspond to relations between users. The profiles of users (e.g., age, gender, and location) can be represented as node features  $\mathbf{F}^V$  and interaction data (e.g., sending messages and comments) can be represented as edge features  $\mathbf{F}^E$ .

**Preliminaries.** The Laplacian matrix of an undirected graph is defined as  $\mathbf{L} = \mathbf{D} - \mathbf{A}$ , where  $\mathbf{D} \in \mathbb{R}^{N \times N}$  is a diagonal degree matrix with  $\mathbf{D}(i, i) = \sum_j \mathbf{A}(i, j)$ . Its eigendecomposition is denoted as  $\mathbf{L} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$ , where  $\mathbf{\Lambda} \in \mathbb{R}^{N \times N}$  is a diagonal matrix of eigenvalues sorted in 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  $v_i$  landing 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.

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

TABLE 2  
A Table for Commonly Used Notations

$G = (V, E)$	A graph
$N, M$	The number of nodes and edges
$V = \{v_1, \dots, v_N\}$	The set of nodes
$\mathbf{F}^V, \mathbf{F}^E$	The attributes/features of 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 eigendecomposition 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 dimensionality of $\mathbf{H}^l$
$\rho(\cdot)$	Some non-linear activation function
$\mathbf{X}_1 \odot \mathbf{X}_2$	The element-wise multiplication
$\Theta$	Learnable parameters
$s$	The sample size

To simplify the notation, we omit 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 dimensionality of the layer  $l$  (i.e.,  $\mathbf{H}^l \in \mathbb{R}^{N \times f_l}$ ). The sigmoid activation function is defined as  $\sigma(x) = 1/(1 + e^{-x})$  and the rectified linear unit (ReLU) is defined as  $\text{ReLU}(x) = \max(0, x)$ . A general element-wise nonlinear activation function is denoted as  $\rho(\cdot)$ . In this paper, unless stated otherwise, we assume all functions are differentiable, allowing the model parameters  $\Theta$  to be learned through back-propagation [18] using commonly adopted optimizers such as Adam [19] and training techniques such as dropout [20]. We denote the sample size as  $s$  if a sampling technique is adopted. We summarize the notations in Table 2.

The tasks for learning a deep model on graphs can be broadly divided into two categories:

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

Note that such distinctions are more conceptually than mathematically rigorous. Some existing tasks are 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 egocentric 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) [30] or long short-term memory (LSTM) [31] are de facto standards in modeling sequential data. In this section, we review Graph RNNs 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 patterns lie at the node-level and are modeled by node states, or at the graph-level and are modeled by a common graph state. 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)<sup>3</sup>, can be dated back to the "pre-deep-learning" era [23], [32]. The idea behind a GNN is simple: to encode graph structural information, each node  $v_i$  is represented by a low-dimensional state vector  $\mathbf{s}_i$ . Motivated by recursive neural networks [33], a recursive definition of states is adopted [23]:

$$\mathbf{s}_i = \sum_{j \in \mathcal{N}(i)} \mathcal{F}(\mathbf{s}_i, \mathbf{s}_j, \mathbf{F}_i^V, \mathbf{F}_j^V, \mathbf{F}_{i,j}^E), \quad (1)$$

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

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

For graph-focused tasks, the authors of [23] suggested adding a special node with unique attributes to represent the entire graph. To learn the model parameters, the following semi-supervised<sup>4</sup> method is adopted: after iteratively solving Eq. (1) to a stable point using the **Jacobi method** [34], one gradient descent step is performed using the **Almeida-Pineda algorithm** [35], [36] to minimize a task-specific objective function, for example, the squared loss between the predicted values and the ground-truth for regression tasks; then, this process is repeated until convergence.

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

Although they are conceptually important, GNNs have several drawbacks. First, to ensure that Eq. (1) has a unique solution,  $\mathcal{F}(\cdot)$  must be a "contraction map" [37], i.e.,  $\exists \mu, 0 < \mu < 1$  so that

$$\|\mathcal{F}(x) - \mathcal{F}(y)\| \leq \mu \|x - y\|, \forall x, y. \quad (3)$$

Intuitively, a "contraction map" requires that the distance between any two points can only "contract" after the  $\mathcal{F}(\cdot)$  operation, which severely limits the modeling ability. Second, because many iterations are needed to reach a stable state between gradient descend steps, GNNs are computationally expensive. Because of these drawbacks and perhaps a lack of computational power (e.g., the graphics processing unit, GPU, was not widely used for deep learning in those days) and lack of research interests, GNNs did not become a focus of general research.

A notable improvement to GNNs is gated graph sequence neural networks (GGS-NNs) [24] with the following modifications. Most importantly, the authors **replaced the recursive definition** in Eq. (1) with a **GRU**, thus removing the "contraction map" requirement and supporting modern optimization techniques. Specifically, Eq. (1) is adapted as follows:

$$\mathbf{s}_i^{(t)} = (1 - \mathbf{z}_i^{(t)}) \odot \mathbf{s}_i^{(t-1)} + \mathbf{z}_i^{(t)} \odot \tilde{\mathbf{s}}_i^{(t)}, \quad (4)$$

3. Recently, GNNs have also been used to refer to general neural networks for graph data. We follow the traditional naming 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 some subset of the node or graph labels is used during training.

TABLE 3  
The Main Characteristics of Graph Recurrent Neural Network (Graph RNNs)

Category	Method	Recursive/sequential patterns of graphs	Time Complexity	Other Improvements
Node-level	GNN [23]	A recursive definition of node states	$O(MI_f)$	-
	GGs-NNs [24]		$O(MT)$	Sequence outputs
	SSE [25]		$O(d_{avg}S)$	-
Graph-level	You <i>et al.</i> [26]	Generate nodes and edges in an autoregressive manner	$O(N^2)$	-
	DGNN [27]	Capture the time dynamics of the formation of nodes and edges	$O(Md_{avg})$	-
	RMGCNN [28]	Recursively reconstruct the graph	$O(M)$ or $O(MN)$	GCN layers
	Dynamic GCN [29]	Gather node representations in different time slices	$O(Mt)$	GCN layers

where  $\mathbf{z}$  is calculated by the update gate,  $\tilde{\mathbf{s}}$  is the candidate for updating, and  $t$  is the pseudo time. Second, the authors proposed using several such networks operating in sequence to produce sequence outputs and showed that their method could be applied to sequence-based tasks such as program verification [38].

SSE [25] took a similar approach as Eq. (4). However, instead of using a GRU in the calculation, SSE adopted stochastic fixed-point gradient descent to accelerate the training process. This scheme basically alternates between calculating steady node states using local neighborhoods and optimizing the model parameters, with both calculations in stochastic mini-batches.

### 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 graphs or sequential patterns at different levels of graph granularities. In graph-level RNNs, instead of applying one RNN to each node to learn the node states, a single RNN is applied to the entire graph to encode the graph states.

You *et al.* [26] applied Graph RNNs to the graph generation problem. Specifically, they adopted two RNNs: one to generate new nodes and the other to generate edges for the newly added node in an autoregressive manner. They showed that such hierarchical RNN architectures learn more effectively from input graphs than do the traditional rule-based graph generative models while having a reasonable time complexity.

To capture the temporal information of dynamic graphs, dynamic graph neural network (DGNN) [27] was proposed that used a time-aware LSTM [39] to learn node representations. When a new edge is established, DGNN used the 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 showed that the time-aware LSTM could model the establishing orders and time intervals of edge formations well, which in turn benefited a range of graph applications.

Graph RNN can also be combined with other architectures, such as GCNs or GAEs. For example, aiming to tackle the graph sparsity problem, RMGCNN [28] applied an LSTM to the results of GCNs to progressively reconstruct a graph as illustrated in Figure 2. By using an LSTM, the information from different parts of the graph can diffuse across long ranges without requiring as many GCN layers. Dynamic GCN [29] applied an LSTM to gather the results of GCNs from different time slices in dynamic networks to capture both the spatial and temporal graph information.

## 4 GRAPH CONVOLUTIONAL NETWORKS

Graph convolutional networks (GCNs) are inarguably the hottest topic in graph-based deep learning. Mimicking CNNs, modern GCNs learn the common local and global structural patterns

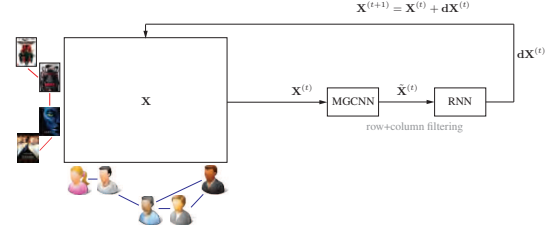


Fig. 2. The framework of RMGCNN (reprinted from [28] with permission). RMGCNN includes an LSTM in the GCN to progressively reconstruct the graph.  $\mathbf{x}^t$ ,  $\tilde{\mathbf{x}}^t$ , and  $d\mathbf{x}^t$  represent the estimated matrix, the outputs of GCNs, and the incremental updates produced by the RNN at iteration  $t$ , respectively. MGCNN refers to a multigraph CNN.

of graphs through designed convolution and readout functions. Because most GCNs can be trained with task-specific loss via backpropagation (with a few exceptions such as the unsupervised training method in [74]), we focus on the adopted architectures. 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

Graph convolutions can be divided into two groups: *spectral convolutions*, which perform convolution by transforming node representations into the spectral domain using the graph Fourier transform or its extensions, and *spatial convolutions*, which perform convolution by considering node neighborhoods. Note that these two groups can overlap, for example, when using a polynomial spectral kernel (please refer to Section 4.1.2 for details).

#### 4.1.1 Spectral Methods

Convolution is the most fundamental operation in CNNs. However, the standard convolution operation used for images or text cannot be directly applied to graphs because graphs lack a grid structure [6]. Bruna *et al.* [40] first introduced convolution for graph data from the spectral domain using the graph Laplacian matrix  $\mathbf{L}$  [75], which plays a similar role as the Fourier basis in signal processing [6]. The graph convolution operation,  $*_G$ , is defined as follows:

$$\mathbf{u}_1 *_G \mathbf{u}_2 = \mathbf{Q} \left( \left( \mathbf{Q}^T \mathbf{u}_1 \right) \odot \left( \mathbf{Q}^T \mathbf{u}_2 \right) \right), \quad (5)$$

where  $\mathbf{u}_1, \mathbf{u}_2 \in \mathbb{R}^N$  are two signals<sup>5</sup> defined on nodes and  $\mathbf{Q}$  are the eigenvectors of  $\mathbf{L}$ . Briefly, multiplying  $\mathbf{Q}^T$  transforms the graph signals  $\mathbf{u}_1, \mathbf{u}_2$  into the spectral domain (i.e., the graph Fourier transform), while multiplying  $\mathbf{Q}$  performs the inverse transform. The validity of this definition is based on the convolution theorem, i.e., the Fourier transform of a convolution operation

5. We give an example of graph signals in Appendix D.



TABLE 4  
A Comparison among Different Graph Convolutional Networks (GCNs). T.C. = Time Complexity, M.G. = Multiple Graphs

Method	Type	Convolution	Readout	T.C.	M.G.	Other Characteristics
Bruna <i>et al.</i> [40]	Spectral	Interpolation kernel	Hierarchical clustering + FC	$O(N^3)$	No	-
Henaff <i>et al.</i> [41]	Spectral	Interpolation kernel	Hierarchical clustering + FC	$O(N^3)$	No	Constructing the graph
ChebNet [42]	Spectral/Spatial	Polynomial	Hierarchical clustering	$O(M)$	Yes	-
Kipf&Welling [43]	Spectral/Spatial	First-order	-	$O(M)$	-	-
CayletNet [44]	Spectral	Polynomial	Hierarchical clustering + FC	$O(M)$	No	-
GWNN [45]	Spectral	Wavelet transform	-	$O(M)$	No	-
Neural FPs [46]	Spatial	First-order	Sum	$O(M)$	Yes	-
PATCHY-SAN [47]	Spatial	Polynomial + an order	An order + pooling	$O(M \log N)$	Yes	A neighbor order
LGCN [48]	Spatial	First-order + an order	-	$O(M)$	Yes	A neighbor order
SortPooling [49]	Spatial	First-order	An order + pooling	$O(M)$	Yes	A node order
DCNN [50]	Spatial	Polynomial diffusion	Mean	$O(N^2)$	Yes	Edge features
DGCN [51]	Spatial	First-order + diffusion	-	$O(N^2)$	-	-
MPNNs [52]	Spatial	First-order	Set2set	$O(M)$	Yes	A general framework
GraphSAGE [53]	Spatial	First-order + sampling	-	$O(Ns^L)$	Yes	A general framework
MoNet [54]	Spatial	First-order	Hierarchical clustering	$O(M)$	Yes	A general framework
GNs [9]	Spatial	First-order	A graph representation	$O(M)$	Yes	A general framework
Kearnes <i>et al.</i> [55]	Spatial	Weave module	Fuzzy histogram	$O(M)$	Yes	Edge features
DiffPool [56]	Spatial	Various	Hierarchical clustering	$O(N^2)$	Yes	Differentiable pooling
GAT [57]	Spatial	First-order	-	$O(M)$	Yes	Attention
GaAN [58]	Spatial	First-order	-	$O(Ns^L)$	Yes	Attention
HAN [59]	Spatial	Meta-path neighbors	-	$O(M_\phi)$	Yes	Attention
CLN [60]	Spatial	First-order	-	$O(M)$	-	-
PPNP [61]	Spatial	First-order	-	$O(M)$	-	Teleportation connections
JK-Nets [62]	Spatial	Various	-	$O(M)$	Yes	Jumping connections
ECC [63]	Spatial	First-order	Hierarchical clustering	$O(M)$	Yes	Edge features
R-GCNs [64]	Spatial	First-order	-	$O(M)$	-	Edge features
LGNN [65]	Spatial	First-order + LINE graph	-	$O(M)$	-	Edge features
PinSage [66]	Spatial	Random walk	-	$O(Ns^L)$	-	Neighborhood sampling
StochasticGCN [67]	Spatial	First-order + sampling	-	$O(Ns^L)$	-	Neighborhood sampling
FastGCN [68]	Spatial	First-order + sampling	-	$O(NsL)$	Yes	Layer-wise sampling
Adapt [69]	Spatial	First-order + sampling	-	$O(NsL)$	Yes	Layer-wise sampling
Li <i>et al.</i> [70]	Spatial	First-order	-	$O(M)$	-	Theoretical analysis
SGC [71]	Spatial	Polynomial	-	$O(M)$	Yes	Theoretical analysis
GFNN [72]	Spatial	Polynomial	-	$O(M)$	Yes	Theoretical analysis
GIN [73]	Spatial	First-order	Sum + MLP	$O(M)$	Yes	Theoretical analysis
DGI [74]	Spatial	First-order	-	$O(M)$	Yes	Unsupervised training

is the element-wise product of their Fourier transforms. Then, a signal  $\mathbf{u}$  can be filtered by

$$\mathbf{u}' = \mathbf{Q}\Theta\mathbf{Q}^T\mathbf{u}, \quad (6)$$

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

$$\mathbf{u}_j^{l+1} = \rho \left( \sum_{i=1}^{f_l} \mathbf{Q}\Theta_{i,j}^l \mathbf{Q}^T \mathbf{u}_i^l \right) \quad j = 1, \dots, f_{l+1}, \quad (7)$$

where  $l$  is the layer,  $\mathbf{u}_j^l \in \mathbb{R}^N$  is the  $j^{th}$  hidden representation (i.e., the signal) for the nodes in the  $l^{th}$  layer, and  $\Theta_{i,j}^l$  are learnable filters. The idea behind Eq. (7) is similar to a conventional convolution: it passes the input signals through a set of learnable filters to aggregate the information, followed by some nonlinear transformation. By using the node features  $\mathbf{F}^V$  as the input layer and stacking multiple convolutional layers, the overall architecture is similar to that of a CNN. Theoretical analysis has shown that such a definition of the graph convolution operation can mimic certain geometric properties of CNNs and we refer readers to [7] for a comprehensive survey.

However, directly using Eq. (7) requires learning  $O(N)$  parameters, which may not be feasible in practice. Besides, the filters in the spectral domain may not be localized in the spatial domain,

i.e., each node may be affected by all the other nodes rather than only the nodes in a small region. To alleviate these problems, Bruna *et al.* [40] suggested using the following smoothing filters:

$$\text{diag}(\Theta_{i,j}^l) = \mathcal{K} \alpha_{l,i,j}, \quad (8)$$

where  $\mathcal{K}$  is a fixed interpolation kernel and  $\alpha_{l,i,j}$  are learnable interpolation coefficients. The authors also generalized this idea to the setting where the graph is not given but constructed from raw features using either a supervised or an unsupervised method [41].

However, two fundamental problems remain unsolved. **First**, because the full eigenvectors of the Laplacian matrix are needed during each calculation, the time complexity is at least  $O(N^2)$  for each forward and backward pass, not to mention the  $O(N^3)$  complexity required to calculate the eigendecomposition, meaning that this approach is not scalable to large-scale graphs. **Second**, because the filters depend on the eigenbasis  $\mathbf{Q}$  of the graph, the parameters cannot 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 The Efficiency Aspect

To solve the efficiency problem, ChebNet [42] was proposed to use a polynomial filter as follows:

$$\Theta(\Lambda) = \sum_{k=0}^K \theta_k \Lambda^k, \quad (9)$$

where  $\theta_0, \dots, \theta_K$  are the learnable parameters and  $K$  is the polynomial order. Then, instead of performing the eigendecomposition, the authors rewrote Eq. (9) using the Chebyshev expansion [76]:

$$\Theta(\Lambda) = \sum_{k=0}^K \theta_k \mathcal{T}_k(\tilde{\Lambda}), \quad (10)$$

where  $\tilde{\Lambda} = 2\Lambda/\lambda_{max} - \mathbf{I}$  are the rescaled eigenvalues,  $\lambda_{max}$  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 a polynomial of the Laplacian matrix acts as a polynomial of its eigenvalues, i.e.,  $\mathbf{L}^k = \mathbf{Q}\Lambda^k\mathbf{Q}^T$ , the filter operation in Eq. (6) can be rewritten as follows:

$$\begin{aligned} \mathbf{u}' &= \mathbf{Q}\Theta(\Lambda)\mathbf{Q}^T\mathbf{u} = \sum_{k=0}^K \theta_k \mathbf{Q}\mathcal{T}_k(\tilde{\Lambda})\mathbf{Q}^T\mathbf{u} \\ &= \sum_{k=0}^K \theta_k \mathcal{T}_k(\tilde{\mathbf{L}})\mathbf{u} = \sum_{k=0}^K \theta_k \tilde{\mathbf{u}}_k, \end{aligned} \quad (11)$$

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

$$\tilde{\mathbf{u}}_k = 2\tilde{\mathbf{L}}\tilde{\mathbf{u}}_{k-1} - \tilde{\mathbf{u}}_{k-2} \quad (12)$$

with  $\tilde{\mathbf{u}}_0 = \mathbf{u}$  and  $\tilde{\mathbf{u}}_1 = \tilde{\mathbf{L}}\mathbf{u}$ . Now, because only the matrix multiplication of a sparse matrix  $\tilde{\mathbf{L}}$  and some vectors need to be calculated, the time complexity becomes  $O(KM)$  when using sparse matrix multiplication, where  $M$  is the number of edges and  $K$  is the polynomial order, i.e., the time complexity is linear with respect to the number of edges. It is also easy to see that such a polynomial filter is strictly  $K$ -localized: after one convolution, the representation of node  $v_i$  will be affected only by its  $K$ -step neighborhoods  $\mathcal{N}_K(i)$ . Interestingly, this idea is used independently in network embedding to preserve the high-order proximity [77], of which we omit the details for brevity.

Kipf and Welling [43] further simplified the filtering by using only the first-order neighbors:

$$\mathbf{h}_i^{l+1} = \rho \left( \sum_{j \in \mathcal{N}(i)} \frac{1}{\sqrt{\tilde{\mathbf{D}}(i,i)\tilde{\mathbf{D}}(j,j)}} \mathbf{h}_j^l \Theta^l \right), \quad (13)$$

where  $\mathbf{h}_i^l \in \mathbb{R}^{f_l}$  is the hidden representation of node  $v_i$  in the  $l^{th}$  layer<sup>6</sup>,  $\tilde{\mathbf{D}} = \mathbf{D} + \mathbf{I}$ , and  $\tilde{\mathcal{N}}(i) = \mathcal{N}(i) \cup \{i\}$ . This can be written equivalently in an matrix form as follows:

$$\mathbf{H}^{l+1} = \rho \left( \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^l \Theta^l \right), \quad (14)$$

where  $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ , i.e., adding a self-connection. The authors showed that Eq. (14) is a special case of Eq. (9) by setting  $K = 1$  with a few minor changes. Then, the authors argued that stacking an adequate number of layers as illustrated in Figure 3 has a modeling capacity similar to ChebNet but leads to better results.

An important insight of ChebNet and its extension is that they connect the spectral graph convolution with the spatial architecture. Specifically, they show that when the spectral convolution function is polynomial or first-order, the spectral graph convolution is equivalent to a spatial convolution. In addition, the convolution in Eq. (13) is highly similar to the state definition in a GNN in Eq. (1), except that the convolution definition replaces

6. 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.

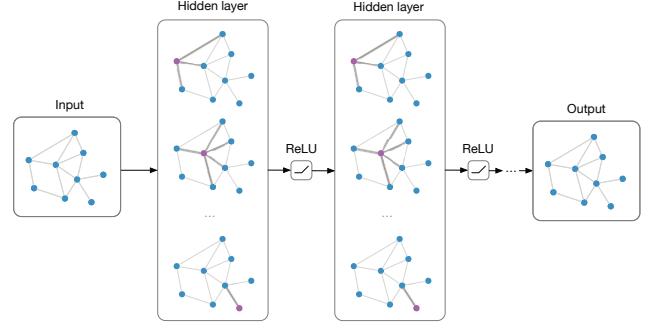


Fig. 3. An illustrative example of the spatial convolution operation proposed by Kipf and Welling [43] (reprinted with permission). Nodes are affected only by their immediate neighbors in each convolutional layer.

the recursive definition. From this aspect, a GNN can be regarded as a GCN with a large number of identical layers to reach stable states [7], i.e., a GNN uses a fixed function with fixed parameters to iteratively update the node hidden states until reaching an equilibrium, while a GCN has a preset number of layers and each layer contains different parameters.

Some spectral methods have also been proposed to solve the efficiency problem. For example, instead of using the Chebyshev expansion as in Eq. (10), CayleyNet [44] adopted Cayley polynomials to define graph convolutions:

$$\Theta(\Lambda) = \theta_0 + 2\text{Re} \left\{ \sum_{k=1}^K \theta_k (\theta_h \Lambda - i\mathbf{I})^k (\theta_h \Lambda + i\mathbf{I})^k \right\}, \quad (15)$$

where  $i = \sqrt{-1}$  denotes the imaginary unit and  $\theta_h$  is another spectral zoom parameter. In addition to showing that CayleyNet is as efficient as ChebNet, the authors demonstrated that the Cayley polynomials can detect “narrow frequency bands of importance” to achieve better results. Graph wavelet neural network (GWNN) [45] was further proposed to replace the Fourier transform in spectral filters by the graph wavelet transform by rewriting Eq. (5) as follows:

$$\mathbf{u}_1 *_G \mathbf{u}_2 = \psi \left( (\psi^{-1} \mathbf{u}_1) \odot (\psi^{-1} \mathbf{u}_2) \right), \quad (16)$$

where  $\psi$  denotes the graph wavelet bases. By using fast approximating algorithms to calculate  $\psi$  and  $\psi^{-1}$ , GWNN’s computational complexity is also  $O(KM)$ , i.e., linear with respect to the number of edges.

#### 4.1.3 The Aspect of Multiple Graphs

A parallel series of works has focuses on generalizing graph convolutions to multiple graphs of arbitrary sizes. Neural FPs [46] proposed a spatial method that also used the first-order neighbors:

$$\mathbf{h}_i^{l+1} = \sigma \left( \sum_{j \in \mathcal{N}(i)} \mathbf{h}_j^l \Theta^l \right). \quad (17)$$

Because the parameters  $\Theta$  can be shared across different graphs and are independent of the graph size, Neural FPs can handle multiple graphs of arbitrary sizes. Note that Eq. (17) is very similar to Eq. (13). However, instead of considering the influence of node degree by adding a normalization term, Neural FPs proposed learning different parameters  $\Theta$  for nodes with different degrees. This strategy performed well for small graphs such as molecular graphs (i.e., atoms as nodes and bonds as edges), but may not be scalable to larger graphs.

PATCHY-SAN [47] adopted a different idea. It assigned a unique node order using a graph labeling procedure such as the Weisfeiler-Lehman kernel [78] and then arranged node neighbors in a line using this pre-defined order. In addition, PATCHY-SAN defined 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 a standard 1-D CNN with proper normalization was adopted. Using this approach, nodes in different graphs all have a “receptive field” with a fixed size and order; thus, PATCHY-SAN can learn from multiple graphs like normal CNNs learn from multiple images. The drawbacks are that the convolution depends heavily on the graph labeling procedure which is a preprocessing step that is not learned. LGCN [48] further proposed to simplify the sorting process by using a lexicographical order (i.e., sorting neighbors based on their hidden representation in the final layer  $\mathbf{H}^L$ ). Instead of using a single order, the authors sorted different channels of  $\mathbf{H}^L$  separately. SortPooling [49] took a similar approach, but rather than sorting the neighbors of each node, the authors proposed to sort all the nodes (i.e., using a single order for all the neighborhoods). Despite the differences among these methods, enforcing a 1-D node order may not be a natural choice for graphs.

DCNN [50] adopted another approach by replacing the eigen-basis of the graph convolution with a diffusion-basis, i.e., the neighborhoods of nodes were determined by the diffusion transition probability between nodes. Specifically, the convolution was defined as follows:

$$\mathbf{H}^{l+1} = \rho \left( \mathbf{P}^K \mathbf{H}^l \Theta^l \right), \quad (18)$$

where  $\mathbf{P}^K = (\mathbf{P})^K$  is the transition probability of a length- $K$  diffusion process (i.e., random walks),  $K$  is a preset diffusion length, and  $\Theta^l$  are learnable parameters. Because only  $\mathbf{P}^K$  depends on the graph structure, the parameters  $\Theta^l$  can be shared across graphs of arbitrary sizes. However, calculating  $\mathbf{P}^K$  has a time complexity of  $O(N^2K)$ ; thus, this method is not scalable to large graphs.

DGCN [51] was further proposed to jointly adopt the diffusion and the adjacency bases using a dual graph convolutional network. Specifically, DGCN used two convolutions: one was Eq. (14), and the other replaced the adjacency matrix with the positive pointwise mutual information (PPMI) matrix [79] of the transition probability as follows:

$$\mathbf{Z}^{l+1} = \rho \left( \mathbf{D}_P^{-\frac{1}{2}} \mathbf{X}_P \mathbf{D}_P^{-\frac{1}{2}} \mathbf{Z}^l \Theta^l \right), \quad (19)$$

where  $\mathbf{X}_P$  is the PPMI matrix calculated as:

$$\mathbf{X}_P(i, j) = \max \left( \log \left( \frac{\mathbf{P}(i, j) \sum_{i,j} \mathbf{P}(i, j)}{\sum_i \mathbf{P}(i, j) \sum_j \mathbf{P}(i, j)} \right), 0 \right), \quad (20)$$

and  $\mathbf{D}_P(i, i) = \sum_j \mathbf{X}_P(i, j)$  is the diagonal degree matrix of  $\mathbf{X}_P$ . Then, these two convolutions were ensembled by minimizing the mean square differences between  $\mathbf{H}$  and  $\mathbf{Z}$ . DGCN adopted a random walk sampling technique to accelerate the transition probability calculation. The experiments demonstrated that such dual convolutions were effective even for single-graph problems.

#### 4.1.4 Frameworks

Based on the above two lines of works, MPNNs [52] were proposed as a unified framework for the graph convolution operation in the spatial domain using message-passing functions:

$$\begin{aligned} \mathbf{m}_i^{l+1} &= \sum_{j \in \mathcal{N}(i)} \mathcal{F}^l(\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{F}_{i,j}^E) \\ \mathbf{h}_i^{l+1} &= \mathcal{G}^l(\mathbf{h}_i^l, \mathbf{m}_i^{l+1}), \end{aligned} \quad (21)$$

where  $\mathcal{F}^l(\cdot)$  and  $\mathcal{G}^l(\cdot)$  are the message functions and vertex update functions to be learned, respectively, and  $\mathbf{m}^l$  denotes the “messages” passed between nodes. Conceptually, MPNNs are a framework in which each node sends messages based on its states and updates its states based on messages received from the immediate neighbors. The authors showed that the above framework had included many existing methods such as GGSNNs [24], Bruna *et al.* [40], Henaff *et al.* [41], Neural FPs [46], Kipf and Welling [43] and Kearnes *et al.* [55] as special cases. In addition, the authors proposed adding a “master” node that was connected to all the nodes to accelerate the message-passing across long distances, and they split the hidden representations into different “towers” to improve the generalization ability. The authors showed that a specific variant of MPNNs could achieve state-of-the-art performance in predicting molecular properties.

Concurrently, GraphSAGE [53] took a similar idea as Eq. (21) using multiple aggregating functions as follows:

$$\begin{aligned} \mathbf{m}_i^{l+1} &= \text{AGGREGATE}^l(\{\mathbf{h}_j^l, \forall j \in \mathcal{N}(i)\}) \\ \mathbf{h}_i^{l+1} &= \rho \left( \Theta^l \left[ \mathbf{h}_i^l, \mathbf{m}_i^{l+1} \right] \right), \end{aligned} \quad (22)$$

where  $[\cdot, \cdot]$  is the concatenation operation and  $\text{AGGREGATE}(\cdot)$  represents the aggregating function. The authors suggested three aggregating functions: the element-wise mean, an LSTM, and max-pooling as follows:

$$\text{AGGREGATE}^l = \max\{\rho(\Theta_{\text{pool}} \mathbf{h}_j^l + \mathbf{b}_{\text{pool}}), \forall j \in \mathcal{N}(i)\}, \quad (23)$$

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

Mixture model network (MoNet) [54] also tried to unify the existing GCN models as well as CNNs for manifolds into a common framework using “template matching”:

$$h_{ik}^{l+1} = \sum_{j \in \mathcal{N}(i)} \mathcal{F}_k^l(\mathbf{u}(i, j)) \mathbf{h}_j^l, k = 1, \dots, f_{l+1}, \quad (24)$$

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

$$\mathcal{F}_k^l(\mathbf{u}) = \exp \left( -\frac{1}{2} (\mathbf{u} - \boldsymbol{\mu}_k^l)^T (\boldsymbol{\Sigma}_k^l)^{-1} (\mathbf{u} - \boldsymbol{\mu}_k^l) \right), \quad (25)$$

where  $\boldsymbol{\mu}_k^l$  and  $\boldsymbol{\Sigma}_k^l$  are the mean vectors and diagonal covariance matrices to be learned, respectively. The pseudo-coordinates were degrees as in Kipf and Welling [43], i.e.,

$$\mathbf{u}(i, j) = \left( \frac{1}{\sqrt{\mathbf{D}(i, i)}}, \frac{1}{\sqrt{\mathbf{D}(j, j)}} \right). \quad (26)$$

Graph networks (GNs) [9] proposed a more general framework for both GCNs and GNNs that learned three sets of representations:  $\mathbf{h}_i^l$ ,  $\mathbf{e}_{ij}^l$ , and  $\mathbf{z}^l$  as the representation for nodes, edges, and the entire graph, respectively. These representations were learned using three aggregation and three updating functions:

$$\begin{aligned} \mathbf{m}_i^l &= \mathcal{G}^{E \rightarrow V}(\{\mathbf{e}_{ij}^l, \forall j \in \mathcal{N}(i)\}), \mathbf{m}_V^l = \mathcal{G}^{V \rightarrow G}(\{\mathbf{h}_i^l, \forall v_i \in V\}) \\ \mathbf{m}_E^l &= \mathcal{G}^{E \rightarrow G}(\{\mathbf{e}_{ij}^l, \forall (v_i, v_j) \in E\}), \mathbf{h}_i^{l+1} = \mathcal{F}^V(\mathbf{m}_i^l, \mathbf{h}_i^l, \mathbf{z}^l) \\ \mathbf{e}_{ij}^{l+1} &= \mathcal{F}^E(\mathbf{e}_{ij}^l, \mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{z}^l), \mathbf{z}^{l+1} = \mathcal{F}^G(\mathbf{m}_E^l, \mathbf{m}_V^l, \mathbf{z}^l), \end{aligned} \quad (27)$$



where  $\mathcal{F}^V(\cdot)$ ,  $\mathcal{F}^E(\cdot)$ , and  $\mathcal{F}^G(\cdot)$  are the corresponding updating functions for nodes, edges, and the entire graph, respectively, and  $\mathcal{G}(\cdot)$  represents message-passing functions whose superscripts denote message-passing directions. Note that the message-passing functions all take a set as the input, thus their arguments are variable in length and these functions should be invariant to input permutations; some examples include the element-wise summation, mean, and maximum. Compared with MPNNs, GNs introduced the edge representations and the representation of the entire graph, thus making the framework more general.

In summary, the convolution operations have evolved from the spectral domain to the spatial domain and from multistep neighbors to the immediate neighbors. Currently, gathering information from the immediate neighbors (as in Eq. (14)) and following the framework of Eqs. (21)(22)(27) are the most common choices for graph convolution operations.

## 4.2 Readout Operations

Using graph convolution operations, useful node features can be learned to solve many node-focused tasks. However, to tackle graph-focused tasks, node information needs to be aggregated to form a graph-level representation. In the literature, such procedures are usually called the readout operations<sup>7</sup>. Based on a regular and local neighborhood, standard CNNs conduct multiple stride convolutions or poolings to gradually reduce the resolution. Since graphs lack a grid structure, these existing methods cannot be used directly.

**Order invariance.** A critical requirement for the graph readout operations is that the operation should be invariant to the node order, i.e., if we change the indices of nodes and edges using a bijective function between two node sets, the representation of the entire graph should not change. For example, whether a drug can treat certain diseases depends on its inherent structure; thus, we should get identical results if we represent the drug using different node indices. Note that because this problem is related to the **graph isomorphism problem**, of which the best-known algorithm is quasipolynomial [80], we only can find a function that is order-invariant but not vice versa in a polynomial time, i.e., even two structurally different graphs may have the same representation.

### 4.2.1 Statistics

The most basic order-invariant operations involve simple statistics such as summation, averaging or max-pooling [46], [50], i.e.,

$$\mathbf{h}_G = \sum_{i=1}^N \mathbf{h}_i^L \text{ or } \mathbf{h}_G = \frac{1}{N} \sum_{i=1}^N \mathbf{h}_i^L \text{ or } \mathbf{h}_G = \max \{ \mathbf{h}_i^L, \forall i \}, \quad (28)$$

where  $\mathbf{h}_G$  is the representation of the 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 sufficiently representative to distinguish different graphs.

Kearnes *et al.* [55] suggested **considering the distribution of node representations by using fuzzy histograms** [81]. The basic idea behind fuzzy histograms is to construct several “histogram bins” and then calculate the memberships of  $\mathbf{h}_i^L$  to these bins, i.e., by regarding node representations as samples and matching

7. Readout operations are also related to graph coarsening, i.e., reducing a large graph to a smaller graph, because a graph-level representation can be obtained by coarsening the graph to a single node. Some papers use these two terms interchangeably.

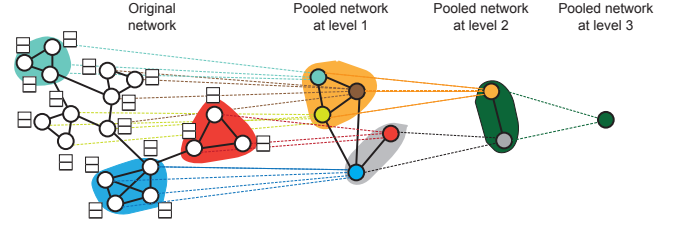


Fig. 4. An example of performing a hierarchical clustering algorithm. Reprinted from [56] with permission.

them to some pre-defined templates, and finally 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 **aggregating node representation** is to add a fully connected (FC) layer as the final layer [40], i.e.,

$$\mathbf{h}_G = \rho \left( [\mathbf{H}^L] \Theta_{FC} \right), \quad (29)$$

where  $[\mathbf{H}^L] \in \mathbb{R}^{N f_L}$  is the concatenation of the final node representation  $\mathbf{H}^L$ ,  $\Theta_{FC} \in \mathbb{R}^{N f_L \times f_{\text{output}}}$  are parameters, and  $f_{\text{output}}$  is the dimensionality of the output. Eq. (29) can be regarded as a weighted sum of node-level features. One advantage is that the model can learn different weights for different nodes; however, this ability comes at the cost of being unable to guarantee order invariance.

### 4.2.2 Hierarchical Clustering

Rather than a dichotomy between node and graph level structures, graphs are known to exhibit rich hierarchical structures [82], which can be explored by hierarchical clustering methods as shown in Figure 4. For example, a **density-based agglomerative clustering** [83] was used in Bruna *et al.* [40] and **multi-resolution spectral clustering** [84] was used in Henaff *et al.* [41]. ChebNet [42] and MoNet [54] adopted another **greedy hierarchical clustering algorithm**, Graclus [85], to merge two nodes at a time, along with a fast pooling method to **rearrange the nodes into a balanced binary tree**. ECC [63] adopted another **hierarchical clustering method by performing eigendecomposition** [86]. However, these hierarchical clustering methods are all independent of the graph convolutions (i.e., they can be performed as a preprocessing step and are not trained in an end-to-end fashion).

To solve that problem, DiffPool [56] proposed a differentiable hierarchical clustering algorithm jointly trained with the graph convolutions. Specifically, the authors **proposed learning a soft cluster assignment matrix in each layer using the hidden representations** as follows:

$$\mathbf{S}^l = \mathcal{F} \left( \mathbf{A}^l, \mathbf{H}^l \right), \quad (30)$$

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 the layer  $l$  and  $\mathcal{F}(\cdot)$  is a function to be learned. Then, **the node representations and the new adjacency matrix for this “coarsened” graph** can be obtained by taking the average according to  $\mathbf{S}^l$  as follows:

$$\mathbf{H}^{l+1} = (\mathbf{S}^l)^T \hat{\mathbf{H}}^{l+1}, \mathbf{A}^{l+1} = (\mathbf{S}^l)^T \mathbf{A}^l \mathbf{S}^l, \quad (31)$$

where  $\hat{\mathbf{H}}^{l+1}$  is obtained by applying a graph 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. The initial number of nodes is  $N_0 = N$  and the last layer is  $N_L = 1$ , i.e., a single node that represents the entire graph. Because the cluster assignment operation is soft, the connections between clusters are not sparse; thus the time complexity of the method is  $O(N^2)$  in principle.

#### 4.2.3 Imposing Orders and Others

As mentioned in Section 4.1.3, PATCHY-SAN [47] and SortPooling [49] took the idea of imposing a node order and then resorted to standard 1-D pooling as in CNNs. Whether these methods can preserve order invariance depends on how the order is imposed, which is another research field that we refer readers to [87] for a survey. However, whether imposing a node order is a natural choice for graphs and if so, what the best node orders are constitute still on-going research topics.

In addition to the aforementioned methods, there are some heuristics. In GNNs [23], the authors suggested adding a special node connected to all nodes to represent the entire graph. Similarly, GNs [9] proposed to directly learn the representation of the entire graph by receiving messages from all nodes and edges.

MPNNs adopted set2set [88], a modification of the seq2seq model. Specifically, set2set uses a “Read-Process-and-Write” model that receives all inputs simultaneously, computes internal memories using an attention mechanism and an LSTM, and then writes the outputs. Unlike seq2seq which is order-sensitive, set2set is invariant to the input order.

#### 4.2.4 Summary

In short, statistics such as averaging or summation are the most simple readout operations, while hierarchical clustering algorithms jointly trained with graph convolutions are more advanced but are also more sophisticated. Other methods such as adding a pseudo node or imposing a node order have also been investigated.

### 4.3 Improvements and Discussions

Many techniques have been introduced to further improve GCNs. Note that some of these methods are general and could be applied to other deep learning models on graphs as well.

#### 4.3.1 Attention Mechanism

In the aforementioned GCNs, the node neighborhoods are aggregated with equal or pre-defined weights. However, the influences of neighbors can vary greatly; thus, they should be learned during training rather than being predetermined. Inspired by the attention mechanism [89], graph attention network (GAT) [57] introduces the attention mechanism into GCNs by modifying the convolution operation in Eq. (13) as follows:

$$\mathbf{h}_i^{l+1} = \rho \left( \sum_{j \in \tilde{\mathcal{N}}(i)} \alpha_{ij}^l \mathbf{h}_j^l \boldsymbol{\Theta}^l \right), \quad (32)$$

where  $\alpha_{ij}^l$  is node  $v_i$ 's attention to node  $v_j$  in the  $l^{th}$  layer:

$$\alpha_{ij}^l = \frac{\exp \left( \text{LeakyReLU} \left( \mathcal{F} \left( \mathbf{h}_i^l \boldsymbol{\Theta}^l, \mathbf{h}_j^l \boldsymbol{\Theta}^l \right) \right) \right)}{\sum_{k \in \tilde{\mathcal{N}}(i)} \exp \left( \text{LeakyReLU} \left( \mathcal{F} \left( \mathbf{h}_i^l \boldsymbol{\Theta}^l, \mathbf{h}_k^l \boldsymbol{\Theta}^l \right) \right) \right)}, \quad (33)$$

where  $\mathcal{F}(\cdot, \cdot)$  is another function to be learned such as a multi-layer perceptron (MLP). To improve model capacity and stability, the authors also suggested using multiple independent attentions and concatenating the results, i.e., the multi-head attention mechanism [89] as illustrated in Figure 5. GaAN [58] further proposed

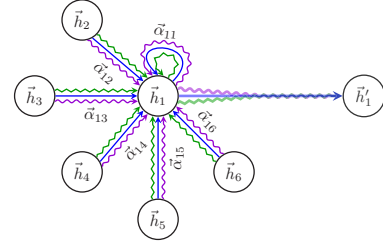


Fig. 5. An illustration of the multi-head attention mechanism proposed in GAT [57] (reprinted with permission). Each color denotes an independent attention vector.

learning different weights for different heads and applied such a method to the traffic forecasting problem.

HAN [59] proposed a two-level attention mechanism, i.e., a node-level and a semantic-level attention mechanism, for heterogeneous graphs. Specifically, the node-level attention mechanism was similar to a GAT, but also considered node types; therefore, it could assign different weights to aggregating meta-path-based neighbors. The semantic-level attention then learned the importance of different meta-paths and outputted the final results.

#### 4.3.2 Residual and Jumping Connections

Many researches have observed that the most suitable depth for the existing GCNs is often very limited, e.g., 2 or 3 layers. This problem is potentially due to the practical difficulties involved in training deep GCNs or the over-smoothing problem, i.e., all nodes in deeper layers have the same representation [62], [70]. To remedy this problem, residual connections similar to ResNet [90] can be added to GCNs. For example, Kipf and Welling [43] added residual connections into Eq. (14) as follows:

$$\mathbf{H}^{l+1} = \rho \left( \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^l \boldsymbol{\Theta}^l \right) + \mathbf{H}^l. \quad (34)$$

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

Column network (CLN) [60] adopted a similar idea by using the following residual connections with learnable weights:

$$\mathbf{h}_i^{l+1} = \alpha_i^l \odot \tilde{\mathbf{h}}_i^{l+1} + (1 - \alpha_i^l) \odot \mathbf{h}_i^l, \quad (35)$$

where  $\tilde{\mathbf{h}}_i^{l+1}$  is calculated similar to Eq. (14) and  $\alpha_i^l$  is a set of weights calculated as follows:

$$\alpha_i^l = \rho \left( \mathbf{b}_\alpha^l + \boldsymbol{\Theta}_\alpha^l \mathbf{h}_i^l + \boldsymbol{\Theta}_\alpha^l \sum_{j \in \mathcal{N}(i)} \mathbf{h}_j^l \right), \quad (36)$$

where  $\mathbf{b}_\alpha^l, \boldsymbol{\Theta}_\alpha^l, \boldsymbol{\Theta}_\alpha^l$  are parameters. Note that Eq. (35) is very similar to the GRU as in GGS-NNs [24]. The differences are that in a CLN, the superscripts denote the number of layers, and different layers contain different parameters, while in GGS-NNs, the superscript denotes the pseudo time and a single set of parameters is used across time steps.

Inspired by personalized PageRank, PPNP [61] defined graph convolutions with teleportation to the initial layer:

$$\mathbf{H}^{l+1} = (1 - \alpha) \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{H}^l + \alpha \mathbf{H}^0, \quad (37)$$

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

Jumping knowledge networks (JK-Nets) [62] proposed another architecture to connect the last layer of the network with all the

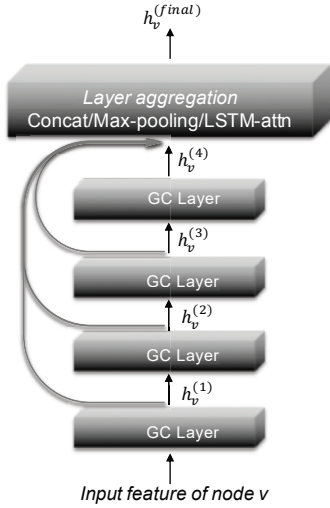


Fig. 6. Jumping knowledge networks proposed in [62] in which the last layer is connected to all the other layers to selectively exploit different information from different layers. GC denotes graph convolutions. Reprinted with permission.

lower hidden layers, i.e., by “jumping” all the 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 was formulated as follows:

$$\mathbf{h}_i^{\text{final}} = \text{AGGREGATE}(\mathbf{h}_i^0, \mathbf{h}_i^1, \dots, \mathbf{h}_i^L), \quad (38)$$

where  $\mathbf{h}_i^{\text{final}}$  is the final representation for node  $v_i$ ,  $\text{AGGREGATE}(\cdot)$  is the aggregating function, and  $L$  is the number of hidden layers. JK-Nets used three aggregating functions similar to GraphSAGE [53]: concatenation, max-pooling, and the LSTM attention. The experimental results showed that adding jump connections could improve the performance of multiple GCNs.

### 4.3.3 Edge Features

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

For simple edge features with discrete values such as the edge type, a straightforward method is to train different parameters for different edge types and aggregate the results. For example, Neural FPs [46] trained different parameters for nodes with different degrees, which corresponds to the implicit edge feature of bond types in a molecular graph, and then summed over the results. CLN [60] trained different parameters for different edge types in a heterogeneous graph and averaged the results. Edge-conditioned convolution (ECC) [63] also trained different parameters based on edge types and applied them to graph classification. Relational GCNs (R-GCNs) [64] adopted a similar idea for knowledge graphs by training different weights for different relation types. However, these methods are suitable only for a limited number of discrete edge features.

DCNN [50] proposed another method to convert each edge into a node connected to the head and tail node of that edge. After this conversion, edge features can be treated as node features.

LGCN [65] constructed a line graph  $\mathbf{B} \in \mathbb{R}^{2M \times 2M}$  to incorporate edge features as follows:

$$\mathbf{B}_{i \rightarrow j, i' \rightarrow j'} = \begin{cases} 1 & \text{if } j = i' \text{ and } j' \neq i, \\ 0 & \text{otherwise.} \end{cases} \quad (39)$$

In other words, nodes in the line graph are directed edges in the original graph, and two nodes in the line graph are connected if information can flow through their corresponding edges in the original graph. Then, LGCN adopted two GCNs: one on the original graph and one on the line graph.

Kearnes *et al.* [55] proposed an architecture using a “weave module”. Specifically, they learned representations for both nodes and edges and exchanged information between them in each weave module using four different functions: node-to-node (NN), node-to-edge (NE), edge-to-edge (EE) and edge-to-node (EN):

$$\begin{aligned} \mathbf{h}_i' &= \mathcal{F}_{NN}(\mathbf{h}_i^0, \mathbf{h}_i^1, \dots, \mathbf{h}_i^L), \mathbf{h}_i'' = \mathcal{F}_{EN}(\{\mathbf{e}_{ij}^l | j \in \mathcal{N}(i)\}) \\ \mathbf{e}_{ij}' &= \mathcal{F}_{EE}(\mathbf{e}_{ij}^0, \mathbf{e}_{ij}^1, \dots, \mathbf{e}_{ij}^L), \mathbf{e}_{ij}'' = \mathcal{F}_{NE}(\mathbf{h}_i^l, \mathbf{h}_j^l) \\ \mathbf{h}_i^{l+1} &= \mathcal{F}_{NN}(\mathbf{h}_i^l, \mathbf{h}_i''), \mathbf{e}_{ij}^{l+1} = \mathcal{F}_{EE}(\mathbf{e}_{ij}^l, \mathbf{e}_{ij}''), \end{aligned} \quad (40)$$

where  $\mathbf{e}_{ij}^l$  is the representation of edge  $(v_i, v_j)$  in the  $l^{\text{th}}$  layer and  $\mathcal{F}(\cdot)$  are learnable functions whose subscripts represent message-passing directions. By stacking multiple such modules, information can propagate by alternately passing between node and edge representations. Note that in the node-to-node and edge-to-edge functions, jump connections similar to those in JK-Nets [62] are implicitly added. GNs [9] also proposed learning an edge representation and updating both node and edge representations using message-passing functions as shown in Eq. (27) in Section 4.1.4. In this aspect, the “weave module” is a special case of GNs that does not a representation of the entire graph.

### 4.3.4 Sampling Methods

One critical bottleneck when training GCNs for large-scale graphs is efficiency. As shown in Section 4.1.4, many GCNs follow a neighborhood aggregation scheme. However, because many real graphs follow a power-law distribution [91] (i.e., a few nodes have very large degrees), the number of neighbors can expand extremely quickly. To deal with this problem, two types of sampling methods have been proposed: neighborhood samplings and layer-wise samplings, as illustrated in Figure 7.

In neighborhood samplings, the sampling is performed for each node during the calculations. GraphSAGE [53] uniformly sampled a fixed number of neighbors for each node during training. PinSage [66] proposed sampling neighbors using random walks on graphs along with several implementation improvements including coordination between the CPU and GPU, a map-reduce inference pipeline, and so on. PinSage was shown to be capable of handling a real billion-scale graph. StochasticGCN [67] further proposed reducing the sampling variances by using the historical activations of the last batches as a control variate, allowing for arbitrarily small sample sizes with a theoretical guarantee.

Instead of sampling neighbors of nodes, FastGCN [68] adopted a different strategy: it sampled nodes in each convolutional layer (i.e., a layer-wise sampling) by interpreting the nodes as i.i.d. samples and the graph convolutions as integral transforms under probability measures. FastGCN also showed that sampling nodes via their normalized degrees could reduce variances and lead to better performance. Adapt [69] further proposed sampling nodes in the lower layers conditioned on their top layer; this approach was more adaptive and applicable to explicitly reduce variances.

### 4.3.5 Inductive Setting

Another important aspect of GCNs is that whether they can be applied to an inductive setting, i.e., training on a set of nodes or

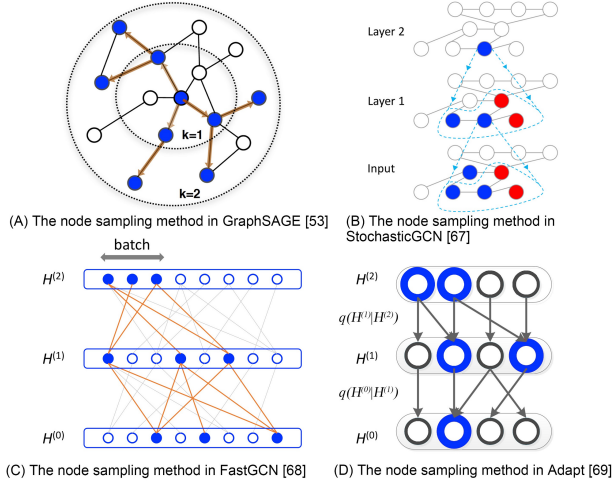


Fig. 7. Different node sampling methods, in which the blue nodes indicate samples from one batch and the arrows indicate the sampling directions. The red nodes in (B) represent historical samples.

graphs and testing on another unseen set of nodes or graphs. In principle, this goal is achieved by learning a mapping function on the given features that are not dependent on the graph basis and can be transferred across nodes or graphs. The inductive setting was verified in GraphSAGE [53], GAT [57], GaAN [58], and FastGCN [68]. However, the existing inductive GCNs are suitable only for graphs with explicit features. **How to conduct inductive learnings for graphs without explicit features**, usually called the out-of-sample problem [92], remains largely open in the literature.

#### 4.3.6 Theoretical Analysis

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

For **node-focused tasks**, Li *et al.* [70] first analyzed the performance of GCNs by using a special form of Laplacian smoothing, which makes the features of nodes in the same cluster similar. The **original Laplacian smoothing operation** is formulated as follows:

$$\mathbf{h}'_i = (1 - \gamma)\mathbf{h}_i + \gamma \sum_{j \in \mathcal{N}(i)} \frac{1}{d_i} \mathbf{h}_j, \quad (41)$$

where  $\mathbf{h}_i$  and  $\mathbf{h}'_i$  are the original and smoothed features of node  $v_i$ , respectively. We can see that Eq. (41) is very similar to the graph convolution in Eq. (13). Based on this insight, Li *et al.* also proposed a **co-training and a self-training method** for GCNs.

Recently, Wu *et al.* [71] analyzed GCNs from a signal processing perspective. By regarding node features as graph signals, they showed that Eq. (13) is basically a fixed low-pass filter. Using this insight, they **proposed an extremely simplified graph convolution (SGC) architecture by removing all the nonlinearities and collapsing the learning parameters into one matrix**:

$$\mathbf{H}^L = \left( \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \right)^L \mathbf{F}_V \Theta. \quad (42)$$

The authors showed that such a “non-deep-learning” GCN variant achieved comparable performance to existing GCNs in many tasks. Maehara [72] enhanced this result by showing that **the low-pass filtering operation did not equip GCNs with a nonlinear manifold learning ability**, and further proposed GFNN model to remedy this problem by adding a MLP after the graph convolution layers.

For **graph-focused tasks**, Kipf and Welling [43] and the authors of SortPooling [49] both **considered the relationship between GCNs and graph kernels** such as the Weisfeiler-Lehman (WL) kernel [78], which is widely used in graph isomorphism tests. They showed that GCNs are conceptually a generalization of the WL kernel because both methods iteratively aggregate information from node neighbors. Xu *et al.* [73] formalized this idea by **proving that the WL kernel provides an upper bound for GCNs in terms of distinguishing graph structures**. Based on this analysis, they proposed graph isomorphism network (GIN) and showed that a readout operation using summation and a MLP can achieve provably **maximum discriminative power**, i.e., the highest training accuracy in graph classification tasks.

For general analysis, Scarselli *et al.* [93] showed that the **Vapnik-Chervonenkis dimension (VC-dim) of GCNs with different activation functions** has the same scale as the existing RNNs. Chen *et al.* [65] analyzed the optimization landscape of linear GCNs and showed that any local minimum is relatively close to the global minimum under certain simplifications. Verma and Zhang [94] analyzed the algorithmic stability and generalization bound of GCNs. They showed that single-layer GCNs satisfy the strong notion of uniform stability if the largest absolute eigenvalue of the graph convolution filters is independent of the graph size.

## 5 GRAPH AUTOENCODERS

The autoencoder (AE) and its variations have been widely applied in unsupervised learning tasks [95] and are suitable for learning node representations for graphs. The implicit assumption is that **graphs have an inherent, potentially nonlinear low-rank structure**. In this section, we 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 originated from sparse autoencoder (SAE) [96]. 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 dimensionality reduction technique to learn low-dimensional node representations**. Specifically, SAE adopted the following L2-reconstruction loss:

$$\min_{\Theta} \mathcal{L}_2 = \sum_{i=1}^N \left\| \mathbf{P}(i, :) - \hat{\mathbf{P}}(i, :) \right\|_2^2 \quad (43)$$

$$\hat{\mathbf{P}}(i, :) = \mathcal{G}(\mathbf{h}_i), \mathbf{h}_i = \mathcal{F}(\mathbf{P}(i, :)),$$

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  $\Theta$  are parameters. Both the encoder and decoder are an MLP with many hidden layers. In other words, a SAE compresses the information of  $\mathbf{P}(i, :)$  into a low-dimensional vector  $\mathbf{h}_i$  and then reconstructs the original feature from that vector. Another sparsity regularization term was also added. After obtaining the low-dimensional representation  $\mathbf{h}_i$ , k-means [106] was applied for the node clustering task. The experiments prove that SAEs outperform non-deep learning baselines. However, SAE was based on an incorrect theoretical analysis.<sup>8</sup> The mechanism underlying its effectiveness remained unexplained.

8. SAE [96] motivated the problem by analyzing the connection between spectral clustering and singular value decomposition, which is mathematically incorrect as pointed out in [107].



TABLE 5  
A Comparison among Different Graph Autoencoders (GAEs). T.C. = Time Complexity

Method	Type	Objective Function	T.C.	Node Features	Other Characteristics
SAE [96]	AE	L2-reconstruction	$O(M)$	No	-
SDNE [97]	AE	L2-reconstruction + Laplacian eigenmaps	$O(M)$	No	-
DNGR [98]	AE	L2-reconstruction	$O(N^2)$	No	-
GC-MC [99]	AE	L2-reconstruction	$O(M)$	Yes	GCN encoder
DRNE [100]	AE	Recursive reconstruction	$O(Ns)$	No	LSTM aggregator
G2G [101]	AE	KL + ranking	$O(M)$	Yes	Nodes as distributions
VGAE [102]	VAE	Pairwise reconstruction	$O(N^2)$	Yes	GCN encoder
DVNE [103]	VAE	Wasserstein + ranking	$O(M)$	No	Nodes as distributions
ARGA/ARVGA [104]	AE/VAE	L2-reconstruction + GAN	$O(N^2)$	Yes	GCN encoder
NetRA [105]	AE	Recursive reconstruction + Laplacian eigenmaps + GAN	$O(M)$	No	LSTM encoder

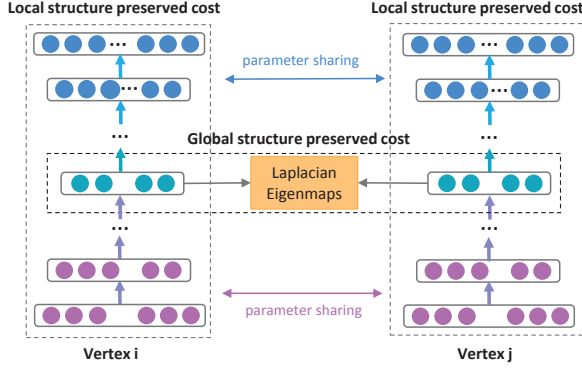


Fig. 8. The framework of SDNE [97]. Both the first and second-order proximities of nodes are preserved using deep autoencoders.

Structure deep network embedding (SDNE) [97] filled in the puzzle by showing that the L2-reconstruction loss in Eq. (43) actually corresponds to the second-order proximity between nodes, i.e., two nodes share similar latent representations if they have similar neighborhoods, which is a well-studied concept in network science known as collaborative filtering or triangle closure [5]. Motivated by network embedding methods showing that the first-order proximity is also important [108], SDNE modified the objective function by adding another Laplacian eigenmaps term [75]:

$$\min_{\Theta} \mathcal{L}_2 + \alpha \sum_{i,j=1}^N \mathbf{A}(i,j) \|\mathbf{h}_i - \mathbf{h}_j\|_2, \quad (44)$$

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

$$\mathcal{L}_2 = \sum_{i=1}^N \|(\mathbf{A}(i,:) - \mathcal{G}(\mathbf{h}_i)) \odot \mathbf{b}_i\|_2, \quad (45)$$

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

Motivated by another line of studies, a contemporary work DNGR [98] replaced the transition matrix  $\mathbf{P}$  in Eq. (43) with the positive pointwise mutual information (PPMI) [79] matrix defined in Eq. (20). In this way, the raw features can be associated with some random walk probability of the graph [109]. However, constructing the input matrix has a time complexity of  $O(N^2)$ , which is not scalable to large-scale graphs.

GC-MC [99] took a different approach by using the GCN proposed by Kipf and Welling [43] as the encoder:

$$\mathbf{H} = \text{GCN}(\mathbf{F}^V, \mathbf{A}), \quad (46)$$

and using a simple bilinear function as the decoder:

$$\hat{\mathbf{A}}(i,j) = \mathbf{H}(i,:) \Theta_{de} \mathbf{H}(j,:)^T, \quad (47)$$

where  $\Theta_{de}$  are the decoder parameters. Using this approach, node features were naturally incorporated. For graphs without node features, a one-hot encoding of node IDs was utilized. The authors demonstrated the effectiveness of GC-MC on the recommendation problem on bipartite graphs.

Instead of reconstructing the adjacency matrix or its variations, DRNE [100] proposed another modification that directly reconstructed the low-dimensional node vectors by aggregating neighborhood information using an LSTM. Specifically, DRNE adopted the following objective function:

$$\mathcal{L} = \sum_{i=1}^N \|\mathbf{h}_i - \text{LSTM}(\{\mathbf{h}_j | j \in \mathcal{N}(i)\})\|. \quad (48)$$

Because an LSTM requires its inputs to be a sequence, the authors suggested ordering the node neighborhoods based on their degrees. They also adopted a neighborhood sampling technique for nodes with large degrees to prevent an overlong memory. The authors proved that such a method can preserve regular equivalence as well as many centrality measures of nodes, such as PageRank [110].

Unlike the above works that map nodes into a low-dimensional vector, Graph2Gauss (G2G) [101] proposed encoding each node as a Gaussian distribution  $\mathbf{h}_i = \mathcal{N}(\mathbf{M}(i,:), \text{diag}(\Sigma(i,:)))$  to capture the uncertainties of nodes. Specifically, the authors used a deep mapping from the node attributes to the means and variances of the Gaussian distribution as the encoder:

$$\mathbf{M}(i,:) = \mathcal{F}_{\mathbf{M}}(\mathbf{F}^V(i,:)), \Sigma(i,:) = \mathcal{F}_{\Sigma}(\mathbf{F}^V(i,:)), \quad (49)$$

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

$$\text{KL}(\mathbf{h}_j || \mathbf{h}_i) < \text{KL}(\mathbf{h}_{j'} || \mathbf{h}_i) \quad (50)$$

$$\forall i, \forall j, \forall j' \text{ s.t. } d(i,j) < d(i,j'),$$

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  $p(\cdot)$  [111]. In other words, the constraints ensure that the KL-divergence between node representations has the same relative order as the graph distance. However, because Eq. (50) is hard to optimize, an energy-based loss [112] was adopted as a relaxation:

$$\mathcal{L} = \sum_{(i,j,j') \in \mathcal{D}} (E_{ij}^2 + \exp^{-E_{ij'}}), \quad (51)$$

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

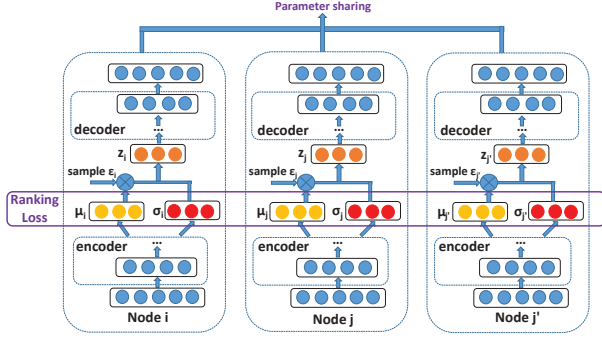


Fig. 9. The framework of DVNE [103]. DVNE represents nodes as distributions using a VAE and adopts the Wasserstein distance to preserve the transitivity of the nodes similarities.

## 5.2 Variational Autoencoders

Different from the aforementioned autoencoders, variational autoencoders (VAEs) are another type of deep learning method that combines dimensionality reduction with generative models. Its potential benefits include tolerating noise and learning smooth representations [113]. VAEs were first introduced to graph data in VGAE [102], where the decoder was a simple linear product:

$$p(\mathbf{A}|\mathbf{H}) = \prod_{i,j=1}^N \sigma(\mathbf{h}_i \mathbf{h}_j^T), \quad (52)$$

in which the node representation was assumed to follow a Gaussian distribution  $q(\mathbf{h}_i|\mathbf{M}, \Sigma) = \mathcal{N}(\mathbf{h}_i|\mathbf{M}(i, :), \text{diag}(\Sigma(i, :)))$ . For the encoder of the mean and variance matrices, the authors also adopted the GCN proposed by Kipf and Welling [43]:

$$\mathbf{M} = \text{GCN}_{\mathbf{M}}(\mathbf{F}^V, \mathbf{A}), \log \Sigma = \text{GCN}_{\Sigma}(\mathbf{F}^V, \mathbf{A}). \quad (53)$$

Then, the model parameters were learned by minimizing the variational lower bound [113]:

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{H}|\mathbf{F}^V, \mathbf{A})} [\log p(\mathbf{A}|\mathbf{H})] - \text{KL}(q(\mathbf{H}|\mathbf{F}^V, \mathbf{A}) \| p(\mathbf{H})). \quad (54)$$

However, because this approach required reconstructing the full graph, its time complexity is  $O(N^2)$ .

Motivated by SDNE and G2G, DVNE [103] proposed another VAE for graph data that also represented each node as a Gaussian distribution. Unlike the existing works that had adopted KL-divergence as the measurement, DVNE used the Wasserstein distance [114] to preserve the transitivity of the nodes similarities. Similar to SDNE and G2G, DVNE also preserved both the first and second-order proximity in its objective function:

$$\min_{\Theta} \sum_{(i,j,j') \in \mathcal{D}} (E_{ij}^2 + \exp^{-E_{ij'}}) + \alpha \mathcal{L}_2, \quad (55)$$

where  $E_{ij} = W_2(\mathbf{h}_j || \mathbf{h}_i)$  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 triples corresponding to the ranking loss of the first-order proximity. The reconstruction loss was defined as follows:

$$\mathcal{L}_2 = \inf_{q(\mathbf{Z}|\mathbf{P})} \mathbb{E}_{p(\mathbf{P})} \mathbb{E}_{q(\mathbf{Z}|\mathbf{P})} \|\mathbf{P} \odot (\mathbf{P} - \mathcal{G}(\mathbf{Z}))\|_2^2, \quad (56)$$

where  $\mathbf{P}$  is the transition matrix and  $\mathbf{Z}$  represents samples drawn from  $\mathbf{H}$ . The framework is shown in Figure 9. Using this approach, the objective function can be minimized as in conventional VAEs using the reparameterization trick [113].

## 5.3 Improvements and Discussions

Several improvements have also been proposed for GAEs.

### 5.3.1 Adversarial Training

An adversarial training scheme<sup>9</sup> was incorporated into GAEs as an additional regularization term in ARGAE [104]. The overall architecture is shown in Figure 10. Specifically, the encoder of GAEs was used as the generator while the discriminator aimed to distinguish whether a latent representation came from the generator or from a prior distribution. In this way, the autoencoder was forced to match the prior distribution as a regularization. The objective function was:

$$\min_{\Theta} \mathcal{L}_2 + \alpha \mathcal{L}_{GAN}, \quad (57)$$

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

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \mathbb{E}_{\mathbf{h} \sim p_{\mathbf{h}}} [\log \mathcal{D}(\mathbf{h})] + \mathbb{E}_{\mathbf{z} \sim \mathcal{G}(\mathbf{F}^V, \mathbf{A})} [\log (1 - \mathcal{D}(\mathbf{z}))], \quad (58)$$

where  $\mathcal{G}(\mathbf{F}^V, \mathbf{A})$  is a generator that uses the graph convolutional encoder from Eq. (53),  $\mathcal{D}(\cdot)$  is a discriminator based on the cross-entropy loss, and  $p_{\mathbf{h}}$  is the prior distribution. The study adopted a simple Gaussian prior, and the experimental results demonstrated the effectiveness of the adversarial training scheme.

Concurrently, NetRA [105] also proposed using a generative adversarial network (GAN) [115] to enhance the generalization ability of graph autoencoders. Specifically, the authors used the following objective function:

$$\min_{\Theta} \mathcal{L}_2 + \alpha_1 \mathcal{L}_{LE} + \alpha_2 \mathcal{L}_{GAN}, \quad (59)$$

where  $\mathcal{L}_{LE}$  is the Laplacian eigenmaps objective function shown in Eq. (44). In addition, the authors adopted an LSTM as the encoder to aggregate information from neighborhoods similar to Eq. (48). Instead of sampling only immediate neighbors and ordering the nodes using degrees as in DRNE [100], the authors used random walks to generate the input sequences. In contrast to ARGAE, NetRA considered the representations in GAEs as the ground-truth and adopted random Gaussian noises followed by an MLP as the generator.

### 5.3.2 Inductive Learning

Similar to GCNs, GAEs can be applied to the inductive learning setting if node attributes are incorporated in the encoder. This can be achieved by using a GCN as the encoder, such as in GC-MC [99], VGAE [102], and VGAE [104], or by directly learning a mapping function from node features as in G2G [101]. Because the edge information is utilized only when learning the parameters, the model can also 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 have been adopted, for example, L2-reconstruction loss, Laplacian eigenmaps, and 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 is needed to understand the underlying differences between these metrics.

9. We will discuss more adversarial methods for graphs in Section 7.

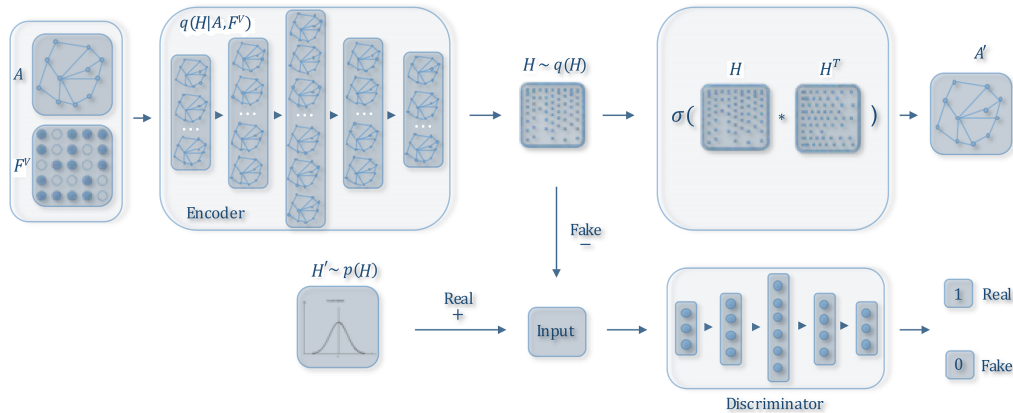


Fig. 10. The framework of ARGAN/ARVGA reprinted from [104] with permission. This model incorporates the adversarial training scheme into GAEs.

TABLE 6  
The Main Characteristics of Graph Reinforcement Learning

Method	Task	Actions	Rewards	Time Complexity
GCPN [116]	Graph generation	Link prediction	GAN + domain knowledge	$O(MN)$
MolGAN [117]	Graph generation	Generate the entire graph	GAN + domain knowledge	$O(N^2)$
GTPN [118]	Chemical reaction prediction	Predict node pairs and new bonding types	Prediction results	$O(N^2)$
GAM [119]	Graph classification	Predict graph labels and select the next node	Classification results	$O(d_{\text{avg}} sT)$
DeepPath [120]	Knowledge graph reasoning	Predict the next node of the reasoning path	Reasoning results + diversity	$O(d_{\text{avg}} sT + s^2T)$
MINERVA [121]	Knowledge graph reasoning	Predict the next node of the reasoning path	Reasoning results	$O(d_{\text{avg}} sT)$

## 6 GRAPH REINFORCEMENT LEARNING

One aspect of deep learning not yet discussed is reinforcement learning (RL), which has been shown to be effective in AI tasks such as playing games [122]. RL is known to be good at learning from feedbacks, especially when dealing with non-differentiable objectives and constraints. In this section, we review Graph RL methods. Their main characteristics are summarized in Table 6.

GCPN [116] utilized RL to generate goal-directed molecular graphs while considering 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 treating agent actions as link predictions, using domain-specific as well as adversarial rewards, and using GCNs to learn the node representations, GCPN can be trained in an end-to-end manner using a policy gradient [123].

A concurrent work, MolGAN [117], adopted a similar idea of using RL for generating molecular graphs. However, rather than generating the graph through a sequence of actions, MolGAN proposed directly generating the full graph; this approach worked particularly well for small molecules.

GTPN [118] adopted RL to predict chemical reaction products. Specifically, the agent acted to select node pairs in the molecule graph and predicted their new bonding types, and rewards were given both immediately and at the end based on whether the predictions were correct. GTPN used a GCN to learn the node representations and an RNN to memorize the prediction sequence.

GAM [119] applied RL to graph classification by using random walks. The authors modeled the generation of random walks as a partially observable Markov decision process (POMDP). The agent performed two actions: first, it predicted the label of the graph; then, it selected the next node in the random walk. The reward was determined simply by whether the agent correctly

classified the graph, i.e.,

$$\mathcal{J}(\theta) = \mathbb{E}_{P(S_{1:T}; \theta)} \sum_{t=1}^T r_t, \quad (60)$$

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

DeepPath [120] and MINERVA [121] both adopted RL for knowledge graph (KG) reasoning. Specifically, DeepPath targeted at pathfinding, i.e., find the most informative path between two target nodes, while MINERVA tackled question-answering tasks, i.e., find the correct answer node given a question node and a relation. In both methods, the RL agents need to predict the next node in the path at each step and output a reasoning path in the KG. Agents receive rewards if the paths reach the correct destinations. DeepPath also added a regularization term to encourage the path diversity.

## 7 GRAPH ADVERSARIAL METHODS

Adversarial methods such as GANs [115] and adversarial attacks have drawn increasing attention in the machine learning community in recent years. In this section, we review how to apply adversarial methods to graphs. The main characteristics of graph adversarial methods are summarized in Table 7.

### 7.1 Adversarial Training

The basic idea behind a 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. Subsequently, both models benefit from each other by joint training using a minimax game. Adversarial training has been shown to be effective in generative models and enhancing the generalization ability of



TABLE 7  
The Main Characteristics of Graph Adversarial Methods

Category	Method	Adversarial Methods	Time Complexity	Node Features
Adversarial Training	ARGA/ARVGA [104]	Regularization for GAEs	$O(N^2)$	Yes
	NetRA [105]	Regularization for GAEs	$O(M)$	No
	GCPN [116]	Rewards for Graph RL	$O(MN)$	Yes
	MolGAN [117]	Rewards for Graph RL	$O(N^2)$	Yes
	GraphGAN [124]	Generation of negative samples (i.e., node pairs)	$O(MN)$	No
	ANE [125]	Regularization for network embedding	$O(N)$	No
	GraphSGAN [126]	Enhancing semi-supervised learning on graphs	$O(N^2)$	Yes
	NetGAN [127]	Generation of graphs via random walks	$O(M)$	No
Adversarial Attack	Nettack [128]	Targeted attacks of graph structures and node attributes	$O(Nd_0^2)$	Yes
	Dai <i>et al.</i> [129]	Targeted attacks of graph structures	$O(M)$	No
	Zugner and Gunnemann [130]	Non-targeted attacks of graph structures	$O(N^2)$	No

discriminative models. In Section 5.3.1 and Section 6, we reviewed how adversarial training schemes are used in GAEs and Graph RL, respectively. Here, we review several other adversarial training methods on graphs in detail.

GraphGAN [124] proposed using a GAN to enhance graph embedding methods [17] with the following objective function:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \sum_{i=1}^N (\mathbb{E}_{v \sim p_{\text{graph}}(\cdot|v_i)} [\log \mathcal{D}(v, v_i)] + \mathbb{E}_{v \sim \mathcal{G}(\cdot|v_i)} [\log (1 - \mathcal{D}(v, v_i))]) . \quad (61)$$

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

$$\mathcal{D}(v, v_i) = \sigma(\mathbf{d}_v \mathbf{d}_{v_i}^T), \mathcal{G}(v|v_i) = \frac{\exp(\mathbf{g}_v \mathbf{g}_{v_i}^T)}{\sum_{v' \neq v_i} \exp(\mathbf{g}_{v'} \mathbf{g}_{v_i}^T)}, \quad (62)$$

where  $\mathbf{d}_v$  and  $\mathbf{g}_v$  are the low-dimensional embedding vectors for node  $v$  in the discriminator and the generator, respectively. Combining the above equations, the discriminator actually has two objectives: the node pairs in the original graph should possess large similarities, while the node pairs generated by the generator should possess small similarities. This architecture is similar to network embedding methods such as LINE [108], except that negative node pairs are generated by the generator  $\mathcal{G}(\cdot)$  instead of by random samplings. The authors showed that this method enhanced the inference abilities of the node embedding vectors.

Adversarial network embedding (ANE) [125] also adopted an adversarial training scheme to improve network embedding methods. Similar to ARGA [104], ANE used a GAN as an additional regularization term to existing network embedding methods such as DeepWalk [131] by imposing a prior distribution as the real data and regarding the embedding vectors as generated samples.

GraphSGAN [126] used a GAN to enhance semi-supervised learning on graphs. Specifically, the authors observed that fake nodes should be generated in the density gaps between subgraphs to weaken the propagation effect across different clusters of the existing models. To achieve that goal, the authors designed a novel optimization objective with elaborate loss terms to ensure that the generator generated samples in the density gaps at equilibrium.

NetGAN [127] adopted a GAN for graph generation tasks. Specifically, the authors regarded graph generation as a task to learn the distribution of biased random walks and adopted a GAN framework to generate and discriminate among random walks using an LSTM. The experiments showed that using random walks could also learn global network patterns.

## 7.2 Adversarial Attacks

Adversarial attacks are another class of adversarial methods intended to deliberately “fool” the targeted methods by adding small

perturbations to data. Studying adversarial attacks can deepen our understanding of the existing models and inspire more robust architectures. We review the graph-based adversarial attacks below.

Nettack [128] first proposed attacking node classification models such as GCNs by modifying graph structures and node attributes. Denoting the targeted node as  $v_0$  and 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 model adopted the following objective function:

$$\begin{aligned} & \arg \max_{(\mathbf{A}', \mathbf{F}^{V'}) \in \mathcal{P}} \max_{c \neq c_{true}} \log \mathbf{Z}_{v_0, c}^* - \log \mathbf{Z}_{v_0, c_{true}}^* \\ & s.t. \mathbf{Z}^* = \mathcal{F}_{\theta^*}(\mathbf{A}', \mathbf{F}^{V'}), \theta^* = \arg \min_{\theta} \mathcal{L}_{\mathcal{F}}(\mathbf{A}', \mathbf{F}^{V'}), \end{aligned} \quad (63)$$

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

Concurrently, Dai *et al.* [129] studied adversarial attacks for graphs with an objective function similar to Eq. (63); however, they focused on the case in which only graph structures were changed. Instead of assuming that the attacker possessed all the information, the authors considered several settings in which different amounts of information were available. The most effective strategy, RL-S2V, adopted structure2vec [132] to learn the node and graph representations and used reinforcement learning to solve the optimization. The experimental results showed that the attacks were effective for both node and graph classification tasks.

The aforementioned two attacks are targeted, i.e., they are intended to cause misclassification of some targeted node  $v_0$ . Zugner and Gunnemann [130] were the first to study non-targeted attacks, which were intended to reduce the overall model performance. They treated the graph structure as hyper-parameters to be optimized and adopted meta-gradients in the optimization process, along with several techniques to approximate the meta-gradients.

## 8 DISCUSSIONS AND CONCLUSION

Thus far, we have reviewed the different graph-based deep learning architectures as well as their similarities and differences. Next, we briefly discuss their applications, implementations, and future directions before summarizing this paper.

### 8.1 Applications

In addition to standard graph inference tasks such as node or graph classification<sup>10</sup>, graph-based deep learning methods have also been applied to a wide range of disciplines, including modeling social influence [133], recommendation [28], [66], [99], [134], chemistry and biology [46], [52], [55], [116], [117], physics [135], [136], disease and drug prediction [137]–[139], gene expression [140], natural language processing (NLP) [141], [142], computer vision [143]–[147], traffic forecasting [148], [149], program induction [150], solving graph-based NP problems [151], [152], and multi-agent AI systems [153]–[155].

A thorough review of these methods is beyond the scope of this paper due to the sheer diversity of these applications; however, we list several key inspirations. First, it is important to incorporate domain knowledge into the model when constructing a graph or choosing architectures. For example, building a graph based on the relative distance may be suitable for traffic forecasting problems, but may not work well for a weather prediction problem where the geographical location is also important. Second, a graph-based model can usually be built on top of other architectures rather than as a stand-alone model. For example, the computer vision community usually adopts CNNs for detecting objects and then uses graph-based deep learning as a reasoning module [156]. For NLP problems, GCNs can be adopted as syntactic constraints [141]. As a result, key challenge is how to integrate different models. These applications also show that graph-based deep learning not only enables mining the rich value underlying the existing graph data but also helps to naturally model relational data as graphs, greatly widening the applicability of graph-based deep learning models.

### 8.2 Implementations

Recently, several open libraries have been made available for developing deep learning models on graphs. These libraries are listed in Table 8. We also collected a list of source code (mostly from their original authors) for the studies discussed in this paper. This repository is included in Appendix A. 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 several ongoing or future research directions which are also worthy of discussion:

- **New models for unstudied graph structures.** Due to the extremely diverse structures of graph data, the existing methods are not suitable for all of them. For example, most methods focus on homogeneous graphs, while heterogeneous graphs are seldom studied, especially those containing different modalities such as those in [157]. Signed networks, in which negative edges represent conflicts between nodes, also

have unique structures, and they pose additional challenges to the existing methods [158]. Hypergraphs, which represent complex relations between more than two objects [159], are also understudied. Thus, an important next step is to design specific deep learning models to handle these types of graphs.

- **Compositionality of existing models.** As shown multiple times in this paper, many of the existing architectures can be integrated: for example, using a GCN as a layer in GAEs or Graph RL. In addition to 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 on a case-by-case basis is also an open problem. One 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 reducing the human burden of assembling different components and choosing hyper-parameters [160].
- **Dynamic graphs.** Most of the existing methods focus on static graphs. However, many real graphs are dynamic in nature: their nodes, edges, and features can change over time. For example, in social networks, people may establish new social relations, remove old relations, and their features, such as hobbies and occupations, can change over time. New users may join the network and existing users may leave. How to model the evolving characteristics of dynamic graphs and support incremental updates to model parameters remain largely unaddressed. Some preliminary works have obtained encouraging results by using Graph RNNs [27], [29].
- **Interpretability and robustness.** Because graphs are often related to other risk-sensitive scenarios, the ability to interpret the results of deep learning models on graphs is critical in decision-making problems. For example, in medicine or disease-related problems, interpretability is essential in transforming computer experiments into applications for clinical use. However, interpretability for graph-based deep learning is even more challenging than are other black-box models because graph nodes and edges are often heavily interconnected. In addition, because many existing deep learning models on graphs are sensitive to adversarial attacks as shown in Section 7.2, enhancing the robustness of the existing methods is another important issue. Some pioneering works regarding interpretability and robustness can be found in [161] and [162], [163], respectively.

### 8.4 Summary

The above survey shows that deep learning on graphs is a promising and fast-developing research field that both offers exciting opportunities and presents many challenges. Studying deep learning on graphs constitutes a critical building block in modeling relational data, and it is an important step towards a future with better machine learning and artificial intelligence techniques.

## ACKNOWLEDGEMENT

The authors thank Jianfei Chen, Jie Chen, William L. Hamilton, Wenbing Huang, Thomas Kipf, Federico Monti, Shirui Pan, Petar Velickovic, Keyulu Xu, Rex Ying for allowing us to use their figures. This work was supported in part by National Program on Key Basic Research Project (No. 2015CB352300), National

10. A collection of methods for common tasks is listed in Appendix B.

TABLE 8  
Libraries of Deep Learning on Graphs

Name	URL	Language/Framework	Key Characteristics
PyTorch Geometric [164]	<a href="https://github.com/rusty1s/pytorch_geometric">https://github.com/rusty1s/pytorch_geometric</a>	PyTorch	Improved efficiency, unified operations, comprehensive existing methods
Deep Graph Library [165]	<a href="https://github.com/dmlc/dgl">https://github.com/dmlc/dgl</a>	PyTorch	Improved efficiency, unified operations, scalability
AliGraph [166]	<a href="https://github.com/alibaba/aligraph">https://github.com/alibaba/aligraph</a>	Unknown	Distributed environment, scalability, in-house algorithms
Euler	<a href="https://github.com/alibaba/euler">https://github.com/alibaba/euler</a>	C++/TensorFlow	Distributed environment, scalability

Key R&D Program of China under Grand 2018AAA0102004, National Natural Science Foundation of China (No. U1936219, No. U1611461, No. 61772304), and Beijing Academy of Artificial Intelligence (BAAI). All opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

## REFERENCES

- [1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, 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*, 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.
- [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*, 2017.
- [8] C. Zang, P. Cui, and C. Faloutsos, "Beyond sigmoids: The netlode model for social network growth, and its applications," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.
- [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," *ACM Transactions on Knowledge Discovery from Data*, 2019.
- [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*, 2019.
- [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*, 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*, 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 *NeurIPS*, 2012.
- [23] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, 2009.
- [24] 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.
- [25] 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.
- [26] 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.
- [27] Y. Ma, Z. Guo, Z. Ren, E. Zhao, J. Tang, and D. Yin, "Streaming graph neural networks," *arXiv preprint arXiv:1810.10627*, 2018.
- [28] F. Monti, M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," in *Advances in Neural Information Processing Systems*, 2017.
- [29] F. Manessi, A. Rozza, and M. Manzo, "Dynamic graph convolutional networks," *arXiv preprint arXiv:1704.06199*, 2017.
- [30] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 2014.
- [31] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, 1997.
- [32] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *IEEE International Joint Conference on Neural Networks Proceedings*, 2005.
- [33] P. Frasconi, M. Gori, and A. Sperduti, "A general framework for adaptive processing of data structures," *IEEE transactions on Neural Networks*, 1998.
- [34] M. J. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *The computer journal*, 1964.
- [35] L. B. Almeida, "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment," in *Proceedings, 1st First International Conference on Neural Networks*, 1987.
- [36] F. J. Pineda, "Generalization of back-propagation to recurrent neural networks," *Physical Review Letters*, 1987.
- [37] M. A. Khamsi and W. A. Kirk, *An introduction to metric spaces and fixed point theory*. John Wiley & Sons, 2011.
- [38] 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.
- [39] 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*, 2017.
- [40] 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.



- [41] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.
- [42] 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.
- [43] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the 6th International Conference on Learning Representations*, 2017.
- [44] 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*, 2017.
- [45] 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.
- [46] 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.
- [47] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International Conference on Machine Learning*, 2016.
- [48] 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*, 2018.
- [49] 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.
- [50] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2016.
- [51] 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.
- [52] 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.
- [53] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017.
- [54] 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 *CVPR*, 2017.
- [55] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, "Molecular graph convolutions: moving beyond fingerprints," *Journal of Computer-Aided Molecular Design*, 2016.
- [56] 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.
- [57] 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.
- [58] 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.
- [59] 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*, 2019.
- [60] 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.
- [61] 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.
- [62] 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.
- [63] M. Simonovsky and N. Komodakis, "Dynamic edgeconditioned filters in convolutional neural networks on graphs," in *2017 IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- [64] 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*, 2018.
- [65] 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.
- [66] 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.
- [67] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *International Conference on Machine Learning*, 2018.
- [68] 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.
- [69] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *Advances in Neural Information Processing Systems*, 2018.
- [70] 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.
- [71] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International Conference on Machine Learning*, 2019.
- [72] T. Maehara, "Revisiting graph neural networks: All we have is low-pass filters," *arXiv preprint arXiv:1905.09550*, 2019.
- [73] 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.
- [74] 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.
- [75] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Advances in neural information processing systems*, 2002.
- [76] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, 2011.
- [77] Z. Zhang, P. Cui, X. Wang, J. Pei, X. Yao, and W. Zhu, "Arbitrary-order proximity preserved network embedding," in *KDD*, 2018.
- [78] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, 2011.
- [79] O. Levy and Y. Goldberg, "Neural word embedding as implicit matrix factorization," in *Advances in Neural Information Processing Systems*, 2014.
- [80] L. Babai, "Graph isomorphism in quasipolynomial time," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, 2016.
- [81] G. Klir and B. Yuan, *Fuzzy sets and fuzzy logic*. Prentice hall New Jersey, 1995.
- [82] 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*, 2018.
- [83] D. Ruppert, "The elements of statistical learning: Data mining, inference, and prediction," *Journal of the Royal Statistical Society*, 2010.
- [84] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, 2007.
- [85] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007.
- [86] D. I. Shuman, M. J. Faraji, and P. Vandergheynst, "A multiscale pyramid transform for graph signals," *IEEE Transactions on Signal Processing*, 2016.
- [87] B. D. McKay and A. Piperno, *Practical graph isomorphism, II*. Academic Press, Inc., 2014.
- [88] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," *Proceedings of the 5th International Conference on Learning Representations*, 2016.
- [89] 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.
- [90] 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.
- [91] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, 1999.
- [92] J. Ma, P. Cui, and W. Zhu, "Depthlgn: Learning embeddings of out-of-sample nodes in dynamic networks," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 2018.
- [93] F. Scarselli, A. C. Tsoi, and M. Hagenbuchner, "The vapnik-chervonenkis dimension of graph and recursive neural networks," *Neural Networks*, 2018.

- [94] S. Verma and Z.-L. Zhang, “Stability and generalization of graph convolutional neural networks,” in *KDD*, 2019.
- [95] 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.
- [96] 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.
- [97] D. Wang, P. Cui, and W. Zhu, “Structural deep network embedding,” in *KDD*, 2016.
- [98] 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.
- [99] R. v. d. Berg, T. N. Kipf, and M. Welling, “Graph convolutional matrix completion,” *KDD’18 Deep Learning Day*, 2018.
- [100] K. Tu, P. Cui, X. Wang, P. S. Yu, and W. Zhu, “Deep recursive network embedding with regular equivalence,” in *KDD*, 2018.
- [101] 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.
- [102] T. N. Kipf and M. Welling, “Variational graph auto-encoders,” *NIPS Workshop on Bayesian Deep Learning*, 2016.
- [103] 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*, 2018.
- [104] 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.
- [105] 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*, 2018.
- [106] 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*, 1967.
- [107] Z. Zhang, “A note on spectral clustering and svd of graph data,” *arXiv preprint arXiv:1809.11029*, 2018.
- [108] 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*, 2015.
- [109] L. Lovász *et al.*, “Random walks on graphs: A survey,” *Combinatorics*, 1993.
- [110] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web.” Stanford InfoLab, Tech. Rep., 1999.
- [111] S. Kullback and R. A. Leibler, “On information and sufficiency,” *The annals of mathematical statistics*, 1951.
- [112] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang, “A tutorial on energy-based learning,” *Predicting structured data*, 2006.
- [113] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” in *Proceedings of the 3rd International Conference on Learning Representations*, 2014.
- [114] S. Vallender, “Calculation of the wasserstein distance between probability distributions on the line,” *Theory of Probability & Its Applications*, 1974.
- [115] 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.
- [116] 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.
- [117] N. De Cao and T. Kipf, “MolGAN: An implicit generative model for small molecular graphs,” *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.
- [118] 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*, 2019.
- [119] 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*, 2018.
- [120] 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*, 2017.
- [121] 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.
- [122] 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*, 2017.
- [123] 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.
- [124] 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.
- [125] Q. Dai, Q. Li, J. Tang, and D. Wang, “Adversarial network embedding,” in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [126] 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*, 2018.
- [127] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, “Netgan: Generating graphs via random walks,” in *International Conference on Machine Learning*, 2018.
- [128] 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*, 2018.
- [129] 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.
- [130] 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.
- [131] 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*, 2014.
- [132] H. Dai, B. Dai, and L. Song, “Discriminative embeddings of latent variable models for structured data,” in *International Conference on Machine Learning*, 2016.
- [133] 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.
- [134] J. Ma, C. Zhou, P. Cui, H. Yang, and W. Zhu, “Learning disentangled representations for recommendation,” in *Advances in Neural Information Processing Systems*, 2019.
- [135] 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*, 2017.
- [136] T. Xie and J. C. Grossman, “Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties,” *Physical Review Letters*, 2018.
- [137] 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*, 2017.
- [138] M. Zitnik, M. Agrawal, and J. Leskovec, “Modeling polypharmacy side effects with graph convolutional networks,” *Bioinformatics*, 2018.
- [139] 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*, 2017.
- [140] 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.
- [141] 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.

- [142] D. Marcheggiani and I. Titov, "Encoding sentences with graph convolutional networks for semantic role labeling," in *EMNLP*, 2017.
- [143] V. Garcia and J. Bruna, "Few-shot learning with graph neural networks," in *Proceedings of the 7th International Conference on Learning Representations*, 2018.
- [144] 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.
- [145] 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.
- [146] 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*, 2017.
- [147] 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*, 2018.
- [148] 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.
- [149] 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.
- [150] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," in *Proceedings of the 7th International Conference on Learning Representations*, 2018.
- [151] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *NeurIPS*, 2018.
- [152] M. Prates, P. H. Avelar, H. Lemos, L. C. Lamb, and M. Y. Vardi, "Learning to solve np-complete problems: A graph neural network for decision tsp," in *AAAI*, 2019.
- [153] S. Sukhbaatar, R. Fergus *et al.*, "Learning multiagent communication with backpropagation," in *Advances in Neural Information Processing Systems*, 2016.
- [154] 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.
- [155] Y. Hoshen, "Vain: Attentional multi-agent predictive modeling," in *Advances in Neural Information Processing Systems*, 2017.
- [156] 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 *NeurIPS*, 2017.
- [157] 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*, 2015.
- [158] T. Derr, Y. Ma, and J. Tang, "Signed graph convolutional network," in *Data Mining, 2018 IEEE International Conference on*, 2018.
- [159] K. Tu, P. Cui, X. Wang, F. Wang, and W. Zhu, "Structural deep embedding for hyper-networks," in *AAAI*, 2018.
- [160] 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*, 2019.
- [161] 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.
- [162] 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*, 2019.
- [163] 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.
- [164] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
- [165] 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.
- [166] 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.
- [167] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *Relational Representation Learning Workshop, NeurIPS 2018*, 2018.
- [168] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv preprint arXiv:1803.03324*, 2018.
- [169] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, 2008.
- [170] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, 2018.



**Ziwei Zhang** received his B.S. from the Department of Physics, Tsinghua University in 2016. He is currently pursuing a 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 in developing scalable algorithms for large-scale networks. He has published several papers in prestigious conferences and journals, including KDD, AAAI, IJCAI, and TKDE.



**Peng Cui** received the Ph.D. degree from Tsinghua University in 2010. He is currently an associate professor with tenure at Tsinghua University. His research interests include network representation learning, human behavioral modeling, and social-sensed multimedia computing. He has published more than 100 papers in prestigious conferences and journals in data mining and multimedia. His recent research efforts have received the SIGKDD 2016 Best Paper Finalist, the ICDM 2015 Best Student Paper Award, the

SIGKDD 2014 Best Paper Finalist, the IEEE ICME 2014 Best Paper Award, the ACM MM12 Grand Challenge Multimodal Award, and the 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 the ACM China Rising Star Award in 2015.



**Wenwu Zhu** is currently a Professor and Deputy Head of the Computer Science Department of Tsinghua University and Vice Dean of National Research Center on Information Science and Technology. Prior to his current post, he was a Senior Researcher and Research Manager at Microsoft Research Asia. He was the Chief Scientist and Director at Intel Research China from 2004 to 2008. He worked at Bell Labs New Jersey as a Member of Technical Staff during 1996-1999. He received his Ph.D. degree from New York University in 1996.

He served as the Editor-in-Chief for the IEEE Transactions on Multimedia (T-MM) from January 1, 2017, to December 31, 2019. He has been serving as Vice EiC for IEEE Transactions on Circuits and Systems for Video Technology (TCSVT) and the chair of the steering committee for IEEE T-MM since January 1, 2020. His current research interests are in the areas of multimedia computing and networking, and big data. He has published over 400 papers in the referred journals and received nine Best Paper Awards including IEEE TCSVT in 2001 and 2019, and ACM Multimedia 2012. He is an IEEE Fellow, AAAS Fellow, SPIE Fellow and a member of the European Academy of Sciences (Academia Europaea).



## APPENDIX A

### SOURCE CODES

Table 9 shows a collection and summary of the source code we collected for the papers discussed in this manuscript. In addition to method names and links, the table also lists the programming language used and the frameworks adopted as well as whether the code was published by the original authors of the paper.

## APPENDIX B

### APPLICABILITY FOR COMMON TASKS

Table 10 summarizes the applicability of different models for six common graph tasks, including node clustering, node classification, network reconstruction, link prediction, graph classification, and graph generation. Note that these results are based on whether the experiments were reported in the original papers.

## APPENDIX C

### NODE CLASSIFICATION RESULTS ON BENCHMARK DATASETS

As shown in Appendix 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<sup>11</sup>:

- Cora, Citeseer, PubMed [169]: 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 [53]: Reddit is an online discussion forum in which nodes represent posts and two nodes are connected when they are commented by the same user, and each post contains a low-dimensional word vector as features and a label indicating the Reddit community in which it was posted.
- PPI [53]: PPI is a collection of protein-protein interaction graphs for different human tissues. It includes features that represent biological signatures and labels that represent the roles of proteins.

Cora, Citeseer, and Pubmed each include one graph, and the same graph structure is used for both training and testing, thus the tasks are considered transductive. In Reddit and PPI, because the training and testing graphs are different, these two datasets are considered to be inductive node classification benchmarks.

In Table 11, we report the results of different models on these benchmark datasets. The results were extracted from their original papers when a fixed dataset split was adopted. The table shows that many state-of-the-art methods achieve roughly comparable performance on these benchmarks, with differences smaller than one percent. Shchur *et al.* [167] also found that a fixed dataset split can easily result in spurious comparisons. As a result, although these benchmarks have been widely adopted to compare different models, more comprehensive evaluation setups are critically needed.

11. These five benchmark datasets are publicly available at <https://github.com/tkipf/gcn> or <http://snap.stanford.edu/graphsage/>.

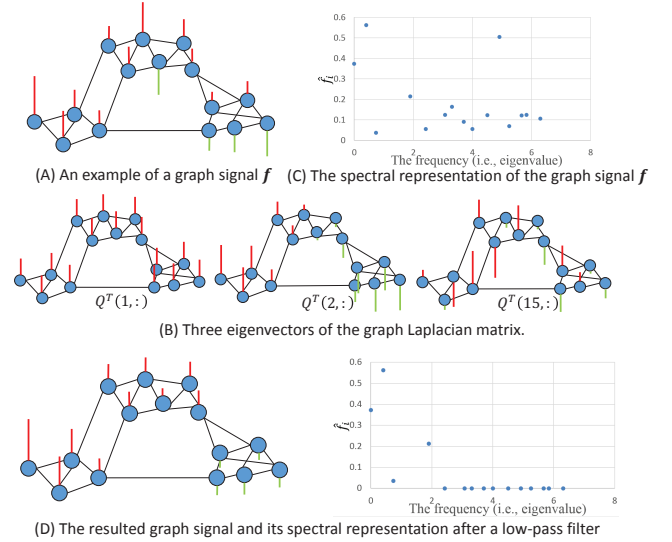


Fig. 11. An example of graph signals and its spectral representations transformed using the eigenvectors of the graph Laplacian matrix. The upward-pointing red lines represent positive values and the downward-pointing green lines represent negative values. These images were adapted from [6].

## APPENDIX D

### AN EXAMPLE OF GRAPH SIGNALS

To help understanding GCNs, we provide an example of graph signals and refer readers to [6], [7], [170] for more comprehensive surveys.

Given a graph  $G = (V, E)$ , a graph signal  $\mathbf{f}$  corresponds to a collection of numbers: one number for each node in the graph. For undirected graphs, we usually assume that the signal takes real values, i.e.,  $\mathbf{f} \in \mathbb{R}^N$ , where  $N$  is the number of nodes. Any node feature satisfying the above requirement can be regarded as a graph signal, with an example shown in Fig 11 (A). Both the signal values and the underlying graph structure are important in processing and analyzing graph signals. For example, we can transform a graph signal into the spectral domain using the eigenvectors of the graph Laplacian matrix:

$$\hat{\mathbf{f}} = \mathbf{Q}^T \mathbf{f} \quad (64)$$

or equivalently

$$\hat{\mathbf{f}}_i = \mathbf{Q}^T(i, :)\mathbf{f}. \quad (65)$$

Because the eigenvectors  $\mathbf{Q}^T$  are sorted in ascending order based on their corresponding eigenvalues, it has been shown [6] that they form a basis for graph signals based on different “smoothness”. Specifically, eigenvectors corresponding to small eigenvalues represent smooth signals and low frequencies, while eigenvectors corresponding to large eigenvalues represent non-smooth signals and high frequencies, as shown in Fig 11 (B). Note that the smoothness is measured with respect to the graph structure, i.e., whether the signals oscillate across edges in the graph. As a result,  $\hat{\mathbf{f}}$  provides a spectral representation of the signal  $\mathbf{f}$  as shown in Fig 11 (C). This is similar to the Fourier transform in Euclidean spaces. Using  $\hat{\mathbf{f}}$ , we can design various signal processing operations. For example, if we apply a low-pass filter, the resulted signal will be more smooth, as shown in Fig 11 (D) (in this example, we set the frequency threshold as 2, i.e., only keeping the lowest 4 frequencies).

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

Category	Method	URL	O.A.	Language/Framework
Graph RNNs	GGs-NNs [24]	<a href="https://github.com/yujiali/ggcn">https://github.com/yujiali/ggcn</a>	Yes	Lua/Torch
	SSE [25]	<a href="https://github.com/HanJun-Dai/steady_state_embedding">https://github.com/HanJun-Dai/steady_state_embedding</a>	Yes	C
	You <i>et al.</i> [26]	<a href="https://github.com/JiaxuanYou/graph-generation">https://github.com/JiaxuanYou/graph-generation</a>	Yes	Python/PyTorch
	RMGCNN [28]	<a href="https://github.com/fmonti/mgcn">https://github.com/fmonti/mgcn</a>	Yes	Python/TensorFlow
GCNs	ChebNet [42]	<a href="https://github.com/mdeff/cnn_graph">https://github.com/mdeff/cnn_graph</a>	Yes	Python/TensorFlow
	Kipf&Welling [43]	<a href="https://github.com/tkipf/gcn">https://github.com/tkipf/gcn</a>	Yes	Python/TensorFlow
	CayletNet [44]	<a href="https://github.com/amoliu/CayleyNet">https://github.com/amoliu/CayleyNet</a>	Yes	Python/TensorFlow
	GWNN [45]	<a href="https://github.com/Eilene/GWNN">https://github.com/Eilene/GWNN</a>	Yes	Python/TensorFlow
	Neural FPs [46]	<a href="https://github.com/HIPS/neural-fingerprint">https://github.com/HIPS/neural-fingerprint</a>	Yes	Python
	PATCHY-SAN [47]	<a href="https://github.com/seiya-kumada/patchy-san">https://github.com/seiya-kumada/patchy-san</a>	No	Python
	LGCN [48]	<a href="https://github.com/divelab/lgcnn">https://github.com/divelab/lgcnn</a>	Yes	Python/TensorFlow
	SortPooling [49]	<a href="https://github.com/muhanzhang/DGCNN">https://github.com/muhanzhang/DGCNN</a>	Yes	Lua/Torch
	DCNN [50]	<a href="https://github.com/jcatw/dcn">https://github.com/jcatw/dcn</a>	Yes	Python/Theano
	DGCN [51]	<a href="https://github.com/ZhuangCY/Coding-NN">https://github.com/ZhuangCY/Coding-NN</a>	Yes	Python/Theano
	MPNNs [52]	<a href="https://github.com/brain-research/mpnn">https://github.com/brain-research/mpnn</a>	Yes	Python/TensorFlow
	GraphSAGE [53]	<a href="https://github.com/williamleif/GraphSAGE">https://github.com/williamleif/GraphSAGE</a>	Yes	Python/TensorFlow
	GNs [9]	<a href="https://github.com/deepmind/graph_nets">https://github.com/deepmind/graph_nets</a>	Yes	Python/TensorFlow
	DiffPool [56]	<a href="https://github.com/RexYing/graph-pooling">https://github.com/RexYing/graph-pooling</a>	Yes	Python/PyTorch
	GAT [57]	<a href="https://github.com/PetarV-/GAT">https://github.com/PetarV-/GAT</a>	Yes	Python/TensorFlow
	GaAN [58]	<a href="https://github.com/jennyzhang0215/GaAN">https://github.com/jennyzhang0215/GaAN</a>	Yes	Python/MXNet
	HAN [59]	<a href="https://github.com/Jhy1993/HAN">https://github.com/Jhy1993/HAN</a>	Yes	Python/TensorFlow
	CLN [60]	<a href="https://github.com/trangptm/Column_networks">https://github.com/trangptm/Column_networks</a>	Yes	Python/Keras
	PPNP [61]	<a href="https://github.com/klicperajo/ppnp">https://github.com/klicperajo/ppnp</a>	Yes	Python/TensorFlow
	JK-Nets [62]	<a href="https://github.com/mori97/JKNet-dgl">https://github.com/mori97/JKNet-dgl</a>	No	Python/DGL
	ECC [63]	<a href="https://github.com/mys007/ecc">https://github.com/mys007/ecc</a>	Yes	Python/PyTorch
	R-GCNs [64]	<a href="https://github.com/tkipf/relational-gcn">https://github.com/tkipf/relational-gcn</a>	Yes	Python/Keras
	LGNN [65]	<a href="https://github.com/joanbruna/GNN_community">https://github.com/joanbruna/GNN_community</a>	Yes	Lua/Torch
	StochasticGCN [67]	<a href="https://github.com/thu-ml/stochastic_gcn">https://github.com/thu-ml/stochastic_gcn</a>	Yes	Python/TensorFlow
	FastGCN [68]	<a href="https://github.com/matennure/FastGCN">https://github.com/matennure/FastGCN</a>	Yes	Python/TensorFlow
	Adapt [69]	<a href="https://github.com/huangwb/AS-GCN">https://github.com/huangwb/AS-GCN</a>	Yes	Python/TensorFlow
	Li <i>et al.</i> [70]	<a href="https://github.com/liqimai/gcn">https://github.com/liqimai/gcn</a>	Yes	Python/TensorFlow
	SGC [71]	<a href="https://github.com/Tiiiger/SGC">https://github.com/Tiiiger/SGC</a>	Yes	Python/PyTorch
	GFNN [72]	<a href="https://github.com/gear/gfnn">https://github.com/gear/gfnn</a>	Yes	Python/PyTorch
	GIN [73]	<a href="https://github.com/weihua916/powerful-gnns">https://github.com/weihua916/powerful-gnns</a>	Yes	Python/PyTorch
	DGI [74]	<a href="https://github.com/PetarV-/DGI">https://github.com/PetarV-/DGI</a>	Yes	Python/PyTorch
GAEs	SAE [96]	<a href="https://github.com/quinnqgroup/deep-representations-clustering">https://github.com/quinnqgroup/deep-representations-clustering</a>	No	Python/Keras
	SDNE [97]	<a href="https://github.com/suanrong/SDNE">https://github.com/suanrong/SDNE</a>	Yes	Python/TensorFlow
	DNGR [98]	<a href="https://github.com/ShelsonCao/DNGR">https://github.com/ShelsonCao/DNGR</a>	Yes	Matlab
	GC-MC [99]	<a href="https://github.com/riannevdberg/gc-mc">https://github.com/riannevdberg/gc-mc</a>	Yes	Python/TensorFlow
	DRNE [100]	<a href="https://github.com/tadpole/DRNE">https://github.com/tadpole/DRNE</a>	Yes	Python/TensorFlow
	G2G [101]	<a href="https://github.com/abojchevski/graph2gauss">https://github.com/abojchevski/graph2gauss</a>	Yes	Python/TensorFlow
	VGA [102]	<a href="https://github.com/tkipf/gae">https://github.com/tkipf/gae</a>	Yes	Python/TensorFlow
	DVNE [103]	<a href="http://nrl.thumediab.com">http://nrl.thumediab.com</a>	Yes	Python/TensorFlow
	ARGA/ARVGA [104]	<a href="https://github.com/Ruiqi-Hu/ARGA">https://github.com/Ruiqi-Hu/ARGA</a>	Yes	Python/TensorFlow
	NetRA [105]	<a href="https://github.com/chengw07/NetRA">https://github.com/chengw07/NetRA</a>	Yes	Python/PyTorch
Graph RLs	GCPN [116]	<a href="https://github.com/bowenliu16/r1_graph_generation">https://github.com/bowenliu16/r1_graph_generation</a>	Yes	Python/TensorFlow
	MolGAN [117]	<a href="https://github.com/nicola-decao/MolGAN">https://github.com/nicola-decao/MolGAN</a>	Yes	Python/TensorFlow
	GAM [119]	<a href="https://github.com/benedekrozemberczki/GAM">https://github.com/benedekrozemberczki/GAM</a>	Yes	Python/PyTorch
	DeepPath [120]	<a href="https://github.com/xwhan/DeepPath">https://github.com/xwhan/DeepPath</a>	Yes	Python/TensorFlow
	MINERVA [121]	<a href="https://github.com/shehzaad/minerva">https://github.com/shehzaad/minerva</a>	Yes	Python/TensorFlow
Graph adversarial methods	GraphGAN [124]	<a href="https://github.com/hwwang55/GraphGAN">https://github.com/hwwang55/GraphGAN</a>	Yes	Python/TensorFlow
	GraphSGAN [126]	<a href="https://github.com/dm-thu/GraphSGAN">https://github.com/dm-thu/GraphSGAN</a>	Yes	Python/PyTorch
	NetGAN [127]	<a href="https://github.com/danielzuegner/netgan">https://github.com/danielzuegner/netgan</a>	Yes	Python/TensorFlow
	Nettack [128]	<a href="https://github.com/danielzuegner/nettack">https://github.com/danielzuegner/nettack</a>	Yes	Python/TensorFlow
	Dai <i>et al.</i> [129]	<a href="https://github.com/HanJun-Dai/graph_adversarial_attack">https://github.com/HanJun-Dai/graph_adversarial_attack</a>	Yes	Python/PyTorch
Applications	Zugner&Gunnemann [130]	<a href="https://github.com/danielzuegner/gnn-meta-attack">https://github.com/danielzuegner/gnn-meta-attack</a>	Yes	Python/TensorFlow
	DeepInf [133]	<a href="https://github.com/xptree/DeepInf">https://github.com/xptree/DeepInf</a>	Yes	Python/PyTorch
	Ma <i>et al.</i> [134]	<a href="https://github.com/xianxinma/assets/disentangle-recsys-v1.zip">https://github.com/xianxinma/assets/disentangle-recsys-v1.zip</a>	Yes	Python/TensorFlow
	CGCNN [136]	<a href="https://github.com/txie-93/cgcnn">https://github.com/txie-93/cgcnn</a>	Yes	Python/PyTorch
	Ktena <i>et al.</i> [137]	<a href="https://github.com/sk1712/gcn_metric_learning">https://github.com/sk1712/gcn_metric_learning</a>	Yes	Python
	Decagon [138]	<a href="https://github.com/mims-harvard/decagon">https://github.com/mims-harvard/decagon</a>	Yes	Python/PyTorch
	Pariset <i>et al.</i> [139]	<a href="https://github.com/parisots/population-gcn">https://github.com/parisots/population-gcn</a>	Yes	Python/TensorFlow
	Dutil <i>et al.</i> [140]	<a href="https://github.com/mila-iqua/gene-graph-conv">https://github.com/mila-iqua/gene-graph-conv</a>	Yes	Python/PyTorch
	Bastings <i>et al.</i> [141]	<a href="https://github.com/bastings/neuralmonkey/tree/emnlp_gcn">https://github.com/bastings/neuralmonkey/tree/emnlp_gcn</a>	Yes	Python/TensorFlow
	Neural-dep-srl [142]	<a href="https://github.com/diegma/neural-dep-srl">https://github.com/diegma/neural-dep-srl</a>	Yes	Python/Therano
	Garcia&Bruna [143]	<a href="https://github.com/vgsatorras/few-shot-gnn">https://github.com/vgsatorras/few-shot-gnn</a>	Yes	Python/PyTorch
	S-RNN [144]	<a href="https://github.com/asheshjain399/RNNexp">https://github.com/asheshjain399/RNNexp</a>	Yes	Python/Therano
	3DGNN [145]	<a href="https://github.com/xjqicuhk/3DGNN">https://github.com/xjqicuhk/3DGNN</a>	Yes	Matlab/Caffe
	GPNN [147]	<a href="https://github.com/SiyuanQi/gpnn">https://github.com/SiyuanQi/gpnn</a>	Yes	Python/PyTorch
	STGCN [148]	<a href="https://github.com/VeritasYin/STGCN_IJCAI-18">https://github.com/VeritasYin/STGCN_IJCAI-18</a>	Yes	Python/TensorFlow
	DCRNN [149]	<a href="https://github.com/liyaguang/DCRNN">https://github.com/liyaguang/DCRNN</a>	Yes	Python/TensorFlow
	Allamanis <i>et al.</i> [150]	<a href="https://github.com/microsoft/tf-gnn-samples">https://github.com/microsoft/tf-gnn-samples</a>	Yes	Python/TensorFlow
	Li <i>et al.</i> [151]	<a href="https://github.com/intel-isl/NPHard">https://github.com/intel-isl/NPHard</a>	Yes	Python/TensorFlow
	TSPGNN [152]	<a href="https://github.com/machine-reasoning-ufrgs/TSP-GNN">https://github.com/machine-reasoning-ufrgs/TSP-GNN</a>	Yes	Python/TensorFlow
	CommNet [153]	<a href="https://github.com/facebookresearch/CommNet">https://github.com/facebookresearch/CommNet</a>	Yes	Lua/Torch
	Interaction network [154]	<a href="https://github.com/jaesik817/Interaction-networks_tensorflow">https://github.com/jaesik817/Interaction-networks_tensorflow</a>	No	Python/TensorFlow
Miscellaneous	Relation networks [156]	<a href="https://github.com/kimhc6028/relational-networks">https://github.com/kimhc6028/relational-networks</a>	No	Python/PyTorch
	SGCN [158]	<a href="http://www.cse.msu.edu/~derrtyl/">http://www.cse.msu.edu/~derrtyl/</a>	Yes	Python/PyTorch
	DHNE [159]	<a href="https://github.com/tadpole/DHNE">https://github.com/tadpole/DHNE</a>	Yes	Python/TensorFlow
	AutoNE [160]	<a href="https://github.com/tadpole/AutoNE">https://github.com/tadpole/AutoNE</a>	Yes	Python
	Gnn-explainer [161]	<a href="https://github.com/RexYing/gnn-model-explainer">https://github.com/RexYing/gnn-model-explainer</a>	Yes	Python/PyTorch
	RGCN [162]	<a href="https://github.com/thumanlab/nrlweb">https://github.com/thumanlab/nrlweb</a>	Yes	Python/TensorFlow
	GNN-benchmark [167]	<a href="https://github.com/schur/gnn-benchmark">https://github.com/schur/gnn-benchmark</a>	Yes	Python/TensorFlow

TABLE 10  
A Table of Methods for Six Common Tasks

Type	Task		Methods
Node-focused Tasks	Node Clustering		[44], [59], [96]–[98], [104], [124], [157]
	Node Classification	Transductive	[23], [25], [27], [29], [41]–[45], [48], [50], [51], [53], [54] [57]–[62], [64], [65], [67]–[72]
		Inductive	[74], [97], [100], [101], [103], [105], [124]–[126], [157], [162] [25], [48], [53], [57], [58], [62], [67]–[69], [71], [72], [74], [101]
	Network Reconstruction		[97], [103], [105], [157]
	Link Prediction		[27], [28], [44], [64], [66], [97], [99], [101]–[105], [124]
	Graph Classification		[23], [29], [40]–[42], [44], [47], [49], [50], [54]–[56], [63], [71], [73], [119], [132]
Graph-focused Tasks	Graph Generation	Structure-only	[26], [127]
		Structure+features	[116], [117], [168]

TABLE 11  
Statistics of the benchmark datasets and the node classification results of different methods when a fixed dataset split is adopted. A hyphen ('-') indicates that the result is unavailable in the paper.

	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	Inductive	Inductive
Bruna <i>et al.</i> [40] <sup>12</sup>	73.3	58.9	73.9	-	-
ChebNet [42] <sup>13</sup>	81.2	69.8	74.4	-	-
GCN [43]	81.5	70.3	79.0	-	-
CayleyNets [44]	81.9±0.7	-	-	-	-
GWNN [45]	82.8	71.7	79.1	-	-
LGCN [48]	83.3±0.5	73.0±0.6	79.5±0.2	-	77.2±0.2
DGCN [51]	83.5	72.6	80.0	-	-
GraphSAGE [53]	-	-	-	95.4	61.2
MoNet [54]	81.7±0.5	-	78.8±0.4	-	-
GAT [57]	83.0±0.7	72.5±0.7	79.0±0.3	-	97.3±0.2
GaAN [58]	-	-	-	96.4±0.0	98.7±0.0
JK-Nets [62]	-	-	-	96.5	97.6±0.7
StochasticGCN [67]	82.0±0.8	70.9±0.2	79.0±0.4	96.3±0.0	97.9±0.0
FastGCN [68]	72.3	-	72.1	93.7	-
Adapt [69]	-	-	-	96.3±0.3	-
SGC [71]	81.0±0.0	71.9±0.1	78.9±0.0	94.9	-
DGI [74]	82.3±0.6	71.8±0.7	76.8±0.6	94.0±0.1	63.8±0.2
SSE [25]	-	-	-	-	83.6
GraphSGAN [126]	83.0±1.3	73.1±1.8	-	-	-
RGCN [162]	82.8±0.6	71.2±0.5	79.1±0.3	-	-

## APPENDIX E TIME COMPLEXITY

In this section, we explain how we obtained the time complexity in all the tables. Specifically, we mainly focus on the time complexity with respect to the graph size, e.g., the number of nodes  $N$  and the number of edges  $M$ , and omit other factors, e.g., the number of hidden dimensions  $f_l$  or the number of iterations, since the latter terms are usually set as small constants and are less dominant. Note that we focus on the theoretical results, while the exact efficiency of one algorithm also depends heavily on its implementations and techniques to reduce the constants in the time complexity.

- GNN [23]:  $O(MI_f)$ , where  $I_f$  is the number of iterations for Eq. (1) to reach stable points, as shown in the paper.
- GGS-NNs [24]:  $O(MT)$ , where  $T$  is a preset maximum pseudo time since the method utilizes all the edges in each updating.
- SSE [25]:  $O(d_{\text{avg}}S)$ , where  $d_{\text{avg}}$  is the average degree and  $S$  is the total number of samples, as shown in the paper.

- You *et al.* [26]:  $O(N^2)$ , as shown in the paper.
- DGNN [27]:  $O(Md_{\text{avg}})$ , where  $d_{\text{avg}}$  is the average degree since the effect of the one-step propagation of each edge is considered.
- RMGCNN [28]:  $O(MN)$  or  $O(M)$ , depending on whether an approximation technique is adopted, as shown in the paper.
- Dynamic GCN [29]:  $O(Mt)$ , where  $t$  denotes the number of time slices since the model runs one GCN at each time slice.
- Brunna *et al.* [40] and Henaff *et al.* [41]:  $O(N^3)$ , due to the time complexity of the eigendecomposition.
- ChebNet [42], Kipf and Welling [43], CayletNet [44], GWNN [45], and Neural FPs [46]:  $O(M)$ , as shown in the corresponding papers.
- PATCHY-SAN [47]:  $O(M \log N)$ , assuming the method adopts WL to label nodes, as shown in the paper.
- LGCN [48]:  $O(M)$  since all the neighbors of each node are sorted in the method.
- SortPooling [49]:  $O(M)$ , due to the time complexity of adopted graph convolution layers.
- DCNN [50]:  $O(N^2)$ , as reported in [43].
- DGCN [51]:  $O(N^2)$  since the PPMI matrix is not sparse.

13. The results were reported in GWNN [45].

13. The results were reported in Kipf and Welling [102].

- MPNNs [52]:  $O(M)$ , as shown in the paper.
- GraphSAGE [53]:  $O(Ns^L)$ , where  $s$  is the size of the sampled neighborhoods and  $L$  is the number of layers, as shown in the paper.
- MoNet [54]:  $O(M)$  since only the existing node pairs are involved in the calculation.
- GNs [9]:  $O(M)$  since only the existing node pairs are involved in the calculation.
- Kearnes *et al.* [55]:  $O(M)$ , since only the existing node pairs are used in the calculation.
- DiffPool [56]:  $O(N^2)$  since the coarsened graph is not sparse.
- GAT [57]:  $O(M)$ , as shown in the paper.
- GaAN [58]:  $O(Ns^L)$ , where  $s$  is a preset maximum neighborhood length and  $L$  is the number of layers, as shown in the paper.
- HAN [59]:  $O(M_\phi)$ , the number of meta-path-based node pairs, as shown in the paper.
- CLN [60]:  $O(M)$  since only the existing node pairs are involved in the calculation.
- PPNP [61]:  $O(M)$ , as shown in the paper.
- JK-Nets [62]:  $O(M)$ , due to the time complexity in adopted graph convolutional layers.
- ECC [63]:  $O(M)$ , as shown in the paper.
- R-GCNs [64]:  $O(M)$  since the edges of different types sum up to the total number of edges of the graph.
- LGNN [65]:  $O(M)$ , as shown in the paper.
- PinSage [66]:  $O(Ns^L)$ , where  $s$  is the size of the sampled neighborhoods and  $L$  is the number of layers since a sampling strategy similar to that of GraphSAGE [53] is adopted.
- StochasticGCN [67]:  $O(Ns^L)$ , as shown in the paper.
- FastGCN [68] and Adapt [69]:  $O(NsL)$  since the samples are drawn in each layer instead of in the neighborhoods, as shown in the paper.
- Li *et al.* [70]:  $O(M)$ , due to the time complexity in adopted graph convolutional layers.
- SGC [71]:  $O(M)$  since the calculation is the same as Kipf and Welling [43] by not adopting nonlinear activations.
- GFNN [72]:  $O(M)$  since the calculation is the same as SGC [71] by adding an extra MLP layer.
- GIN [73]:  $O(M)$ , due to the time complexity in adopted graph convolutional layers.
- DGI [74]:  $O(M)$ , due to the time complexity in adopted graph convolutional layers.
- SAE [96] and SDNE [97]:  $O(M)$ , as shown in the corresponding papers.
- DNGR [98]:  $O(N^2)$ , due to the time complexity of calculating the PPMI matrix.
- GC-MC [99]:  $O(M)$  since the encoder adopts the GCN proposed by Kipf and Welling [43] and only the non-zero elements of the graph are considered in the decoder.
- DRNE [100]:  $O(Ns)$ , where  $s$  is a preset maximum neighborhood length, as shown in the paper.
- G2G [101]:  $O(M)$ , due to the definition of the ranking loss.
- VGAE [102]:  $O(N^2)$ , due to the reconstruction of all the node pairs.
- DVNE [103]: Though the original paper reported to have a time complexity of  $O(Md_{\text{avg}})$  where  $d_{\text{avg}}$  is the average degree, we have confirmed that it can be easily improved to  $O(M)$  through personal communications with the authors.
- ARG/ARVGA [104]:  $O(N^2)$ , due to the reconstruction of all the node pairs.
- NetRA [105]:  $O(M)$ , as shown in the paper.
- GCPN [116]:  $O(MN)$  since the embedding of all the nodes are used when generating each edge.
- MolGAN [117] and GTPN [118]:  $O(N^2)$  since the scores for all the node pairs have to be calculated.
- GAM [119]:  $O(d_{\text{avg}}sT)$ , where  $d_{\text{avg}}$  is the average degree,  $s$  is the number of sampled random walks, and  $T$  is the walk length, as shown in the paper.
- DeepPath [120]:  $O(d_{\text{avg}}sT + s^2T)$ , where  $d_{\text{avg}}$  is the average degree,  $s$  is the number of sampled paths, and  $T$  is the path length. The former term corresponds to finding paths and the latter term results from the diversity constraint.
- MINERVA [121]:  $O(d_{\text{avg}}sT)$ , where  $d_{\text{avg}}$  is the average degree,  $s$  is the number of sampled paths, and  $T$  is the path length, similar to the pathfinding method in DeepPath [120].
- GraphGAN [124]:  $O(MN)$ , as shown in the paper.
- ANE [125]:  $O(N)$ , which is the extra time complexity introduced by the model in the generator and the discriminator.
- GraphSGAN [126]:  $O(N^2)$ , due to the time complexity in the objective function.
- NetGAN [127]:  $O(M)$ , as shown in the paper.
- Nettack [128]:  $O(Nd_0^2)$ , where  $d_0$  is the degree of the targeted node, as shown in the paper.
- Dai *et al.* [129]:  $O(M)$ , which is the time complexity of the most effective strategy RL-S2V, as shown in the paper.
- Zugner and Gunnemann [130]:  $O(N^2)$ , as shown in the paper.