Graph Neural Networks: Taxonomy, Advances, and Trends

YU ZHOU and HAIXIA ZHENG, College of Data Science/Shanxi Spatial Information Network Engineering Technology Research Center, Taiyuan University of Technology, China
XIN HUANG and SHUFENG HAO, College of Data Science, Taiyuan University of Technology, China
DENGAO LI, College of Data Science/Shanxi Spatial Information Network Engineering Technology Research Center, Taiyuan University of Technology, China
JUMIN ZHAO, College of Information and Computer/Shanxi Intelligent Perception Engineering Research Center, Taiyuan University of Technology, China

Graph neural networks provide a powerful toolkit for embedding real-world graphs into low-dimensional spaces according to specific tasks. Up to now, there have been several surveys on this topic. However, they usually lay emphasis on different angles so that the readers cannot see a panorama of the graph neural networks. This survey aims to overcome this limitation and provide a systematic and comprehensive review on the graph neural networks. First of all, we provide a novel taxonomy for the graph neural networks, and then refer to up to 327 relevant literatures to show the panorama of the graph neural networks. All of them are classified into the corresponding categories. In order to drive the graph neural networks into a new stage, we summarize four future research directions so as to overcome the challenges faced. It is expected that more and more scholars can understand and exploit the graph neural networks and use them in their research community.

CCS Concepts: • **Computing methodologies** → **Neural networks**; **Learning latent representations;**

Additional Key Words and Phrases: Graph convolutional neural network, graph recurrent neural network, graph pooling operator, graph attention mechanism, graph neural network

ACM Reference format:
Yu Zhou, Haixia Zheng, Xin Huang, Shufeng Hao, Dengao Li, and Jumin Zhao. 2022. Graph Neural Networks: Taxonomy, Advances, and Trends. *ACM Trans. Intell. Syst. Technol.* 13, 1, Article 15 (January 2022), 59 pages.
https://doi.org/10.1145/3495161

This work was jointly supported by the National Natural Science Foundation of China (Grant No. 62002255, 62076177, 61972273) and Shanxi Key Core Technology and Generic Technology Research and Development Special Project (Grant No. 2020XXX007).

Authors' addresses: Y. Zhou, H. Zheng (corresponding author), and D. Li, College of Data Science/Shanxi Spatial Information Network Engineering Technology Research Center, Taiyuan University of Technology, No. 79 Yingze West Street, WanBaiLin District, Taiyuan, Shanxi, China, 030024; emails: {zhouyu, zhenghaixia, lidengao}@tyut.edu.cn; X. Huang and S. Hao, College of Data Science, Taiyuan University of Technology, No. 79 Yingze West Street, WanBaiLin District, Taiyuan, Shanxi, China, 030024; emails: {huangxin, haoshufeng}@tyut.edu.cn; J. Zhao, College of Information and Computer/Shanxi Intelligent Perception Engineering Research Center, Taiyuan University of Technology, No. 79 Yingze West Street, WanBaiLin District, Taiyuan, Shanxi, China, 030024; email: zhaojumin@tyut.edu.cn.

2157-6904/2022/01-ART15 $15.00
https://doi.org/10.1145/3495161

1 INTRODUCTION

A graph, as a complex data structure, consists of vertices (or vertices) and edges (or links). It can be used to model lots of complex systems in real world, e.g., social networks, protein-protein interaction networks, brain networks, road networks, physical interaction networks, knowledge graphs, and so forth. Thus, analyzing complex networks becomes an intriguing research frontier. With the rapid development of deep learning techniques, many scholars would like to employ the deep learning architectures to tackle graph data. **Graph Neural Networks (GNNs)** emerge under these circumstances. Up to now, the GNNs have evolved into a prevalent and powerful computing framework for tackling irregular data such as graphs and manifolds.

GNNs can learn task-specific vertex/edge/graph representations via hierarchical iterative operators so that the traditional machine learning methods can be employed to perform graph-related learning tasks, e.g., vertex classification, graph classification, link prediction and clustering, and so forth. Although GNNs have attained substantial success over the graph-related learning tasks, they still face great challenges: (1) the structural complexity of graphs incurs expensive computational cost on gigantic graphs, (2) perturbing the graph structure and/or initial features incurs sharp performance decay, (3) the **Wesfeiler-Leman (WL)** graph isomorphism test impedes the performance improvement of the GNNs, and (4) the blackbox work mechanism of GNNs hinders safely deploying them to real-world applications.

The motivations of writing this article are twofold: (1) teasing out a complete and systematic taxonomic framework for GNNs so that each released article can be categorized into a specific topic and (2) providing an elegant tutorial for the people who want to quickly learn the principles and techniques of GNNs. As is well known, there are lots of articles, which are published in various conferences and journals, on GNNs every year. All of them aim to solve different problems on the graph data. On one hand, this reflects that the GNNs have become a quite hot research topic in the AI, machine learning, or data mining community. On the other hand, this results in huge burdens for researchers. As a result, they are overwhelmed by massive articles and so cannot select appropriate references in their research. This may be exacerbated especially for the ones who only want to leverage the GNNs in their research. The major reason for this lies in the lack of systematic and complete surveys on GNNs so that the researchers are unable to grasp the GNNS macroscopically.

Although there have been several surveys [52, 115, 166, 280, 325, 326] on GNNs, this article distinguishes from them in terms of its extensive coverage, up-to-date trends, and detailed discussions.

- **Extensive Coverage.** This article covers almost all branches of the GNNs, and therefore is a panorama of the GNNs.
- **Up-to-date Trends.** This article follows state-of-the-art literatures on the GNNs and provides up-to-date trends of the GNNs that may not be covered by the other surveys.
- **Detailed Discussions.** This article provides deep discussions for each topic of the GNNs. People wanting to enter into the GNN community can learn the principles and trends of the GNNs by reading this article.

Specifically, all of the off-the-shelf surveys more or less have different limitations. To the best of our knowledge, the first survey on the GNNs was conducted by Michael M. Bronstein et al. [166]. This article employs the GNNs to solve the geometric deep learning problems, but its emphasis is not on reviewing different GNN branches. Zhang et al. [325] reviewed different kinds of deep graph learning models from three aspects: semi-supervised graph learning, e.g., graph convolutional neural networks; unsupervised graph learning, e.g., graph auto-encoders; and recent advancements, e.g., graph recurrent neural networks and graph reinforcement learning. Zhou et al. [115] provided a detailed review of the spectral and spatial graph convolutional neural networks from three aspects: graph types, propagation step, and training method, and divided their applications into three scenarios: structural, non-structural, and others. Wu and Philip S. Yu et al. [326] conducted a comprehensive survey on the graph neural networks and investigated available datasets, open-source implementations, and practical applications. Davide Bacciu et al. [52] give a gentle introduction to the deep graph learning field. Their goal is to introduce the main concepts and building blocks to construct neural networks for graph data, and therefore they did not dive into the GNN field. These three surveys only provided a coarse granularity categorization of taxonomy about the GNNs and did not follow the state-of-the-art trends of the GNNs, e.g., capability and explainability of the GNNs, combinations of the probabilitic inference and GNNs, adversarial attacks for the GNNs, graph neural architecture search, self-supervised graph learning, graph pre-training models, and so forth. In addition, some other surveys investigated only a branch of the GNNs, e.g., the literature [280]. It focused on the self-supervised learning on graphs and provided a unified review of different ways of training GNNs using the self-supervised learning methods. Its purpose is different from ours. Moreover, the reason for writing this survey instead of a book lies in that their emphases are different. In general, a book on the GNNs should emphasize introducing the principles of classic GNNs and their applications yet pay less attention to the systematic and holistic taxonomy of the GNN models and their trends.

In this article, we provide a panorama of GNNs for readers from four perspectives: fundamental architectures, extended architectures and applications, implementations and evaluations, and future research directions, as shown in Figure 1. The fundamental architectures are responsible for introducing basic building blocks of GNNs and some canonical GNN architectures. Specifically, this section investigates the studies on **graph convolutional neural networks (GCNNs)**, graph pooling operators, graph attention mechanisms, and **graph recurrent neural networks (GRNNs)**. The extended architectures and applications are responsible for extending the fundamental GNNs to different types of graphs and learning tasks and different application scenarios. Specifically, this section focuses on the capabilities and explainability of GNNs, deep graph representation learning, deep graph generative models, combinations of **Probabilistic Inference (PI)** and GNNs, adversarial attacks for GNNs, graph neural architecture search, graph reinforcement learning, graph pre-training models, self-supervised graph learning, and applications of GNNs. Implementations and evaluations provide a guidance for researchers on how to implement a GNN, how to evaluate a GNN, and how to benchmark a GNN. Finally, the future research directions of GNNs are proposed in order to motivate future research on GNNs. In summary, our article provides a complete taxonomy for GNNs and comprehensively reviews the current advances and trends of the GNNs. These are our main differences from the aforementioned surveys. This article aims at telling readers the answers of four questions: (1) What building blocks do the GNNs consist of? (2) How are these building blocks organized? (3) How are the GNNs implemented and evaluated? (4) Which directions do the GNNs go to in future?

Contributions. Our main contributions boil down to the following threefold aspects:

(1) We propose a novel taxonomy for the GNNs. On the whole, the GNNs are categorized into four categories: fundamental architectures, extended architectures and applications, implementations and evaluations, and future research directions. Each category is again categorized into several sub-categories. This taxonomy basically covers all the research directions of the GNNs.
(2) We provide a comprehensive review of the GNNs. All the literatures fall into the corresponding categories. The readers can not only see the panorama of the GNNs but also comprehend the basic principles and various computation modules of the GNNs through reading this survey.

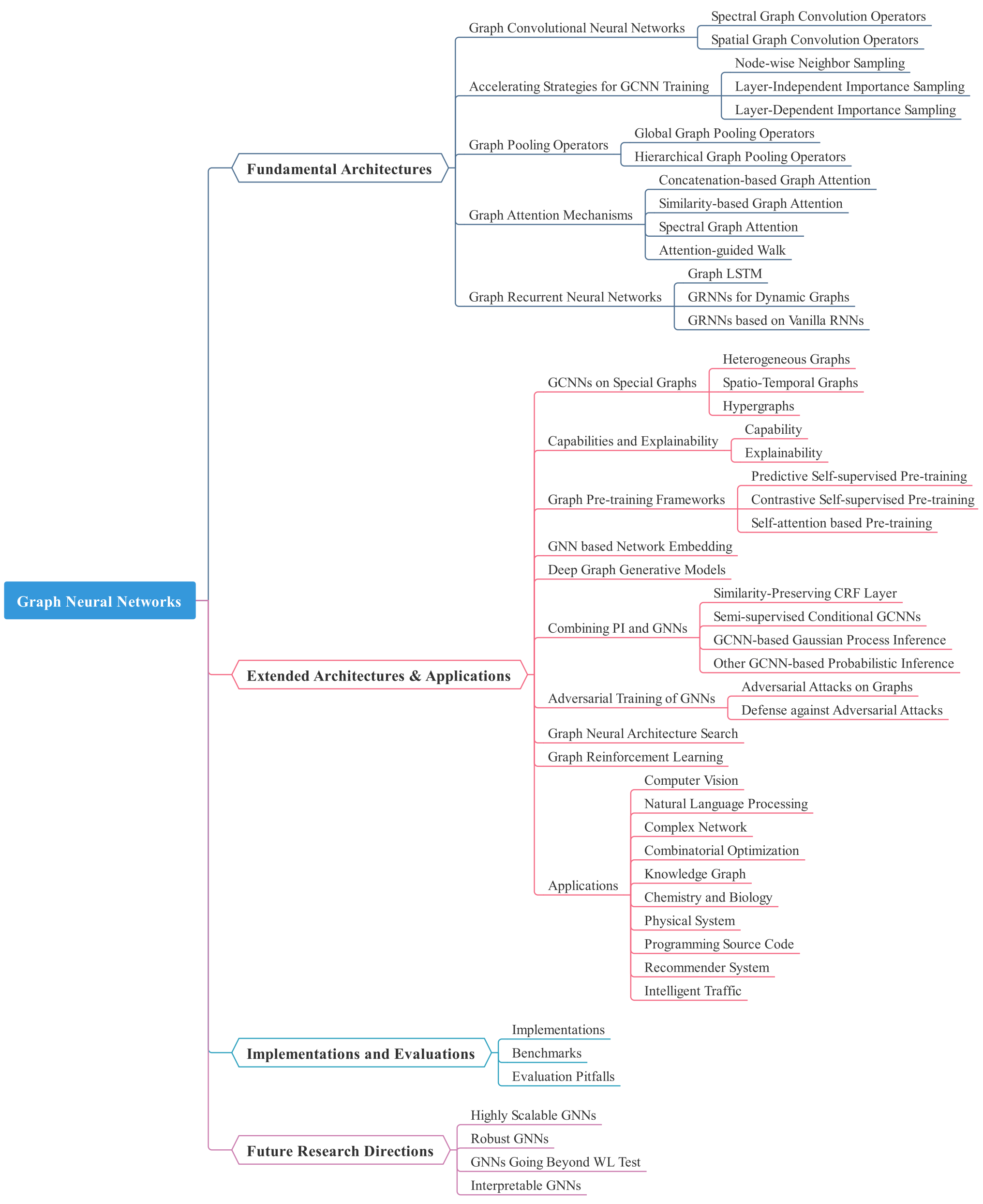
Graph Neural Networks
Fundamental Architectures
Graph Convolutional Neural Networks
Spectral Graph Convolution Operators
Spatial Graph Convolution Operators
Accelerating Strategies for GCNN Training
Node-wise Neighbor Sampling
Layer-Independent Importance Sampling
Layer-Dependent Importance Sampling
Graph Pooling Operators
Global Graph Pooling Operators
Hierarchical Graph Pooling Operators
Graph Attention Mechanisms
Concatenation-based Graph Attention
Similarity-based Graph Attention
Spectral Graph Attention
Attention-guided Walk
Graph Recurrent Neural Networks
Graph LSTM
GRNNs for Dynamic Graphs
GRNNs based on Vanilla RNNs
Extended Architectures & Applications
GCNNs on Special Graphs
Heterogeneous Graphs
Spatio-Temporal Graphs
Hypergraphs
Capabilities and Explainability
Capability
Explainability
Graph Pre-training Frameworks
Predictive Self-supervised Pre-training
Contrastive Self-supervised Pre-training
Self-attention based Pre-training
GNN based Network Embedding
Deep Graph Generative Models
Combining PI and GNNs
Similarity-Preserving CRF Layer
Semi-supervised Conditional GCNNs
GCNN-based Gaussian Process Inference
Other GCNN-based Probabilistic Inference
Adversarial Training of GNNs
Adversarial Attacks on Graphs
Defense against Adversarial Attacks
Graph Neural Architecture Search
Graph Reinforcement Learning
Applications
Computer Vision
Natural Language Processing
Complex Network
Combinatorial Optimization
Knowledge Graph
Chemistry and Biology
Physical System
Programming Source Code
Recommender System
Intelligent Traffic
Implementations and Evaluations
Implementations
Benchmarks
Evaluation Pitfalls
Future Research Directions
Highly Scalable GNNs
Robust GNNs
GNNs Going Beyond WL Test
Interpretable GNNs

Fig. 1. The overview of this article.

(3) We summarize four future research directions for the GNNs according to the current facing challenges, most of which are not mentioned in the other surveys. It is expected that the research on the GNNs can progress into a new stage by overcoming these challenges.

Roadmap. The remainder of this article is organized as follows. First of all, we provide some basic notations and definitions that will be often used in the following sections. Then, we start reviewing the GNNs from four aspects: fundamental architectures in Section 3, extended architectures and applications in Section 4, implementations and evaluations in Section 5, and future research directions in Section 6. Finally, we conclude our article.

2 PRELIMINARIES

In this section, we introduce relevant notations so as to conveniently describe the graph neural network models. A simple graph can be denoted by $G = (V, E)$, where V and E respectively denote the set of N vertices (or nodes) and M edges. Without loss of generality, let $V = \{v_1, \ldots, v_N\}$ and $E = \{e_1, \ldots, e_M\}$. Each edge $e_j \in E$ can be denoted by $e_j = (v_{s_j}, v_{r_j})$, where v_{s_j} and v_{r_j} respectively denote the two endpoints of e_j. Let $\mathbf{A}_G$ denote the adjacency matrix of G, where $\mathbf{A}_G(s, r) = 1$ iff there is an edge between v_s and v_r. If G is edge-weighted, $\mathbf{A}_G(s, r)$ equals the weight value of the edge (v_s, v_r). If G is directed, $(v_{s_j}, v_{r_j}) \neq (v_{r_j}, v_{s_j})$ and therefore $\mathbf{A}_G$ is asymmetric. A directed edge $e_j = (v_{s_j}, v_{r_j})$ is also called an arch, i.e., $e_j = \langle v_{s_j}, v_{s_j} \rangle$. Otherwise, $(v_{s_j}, v_{r_j}) = (v_{r_j}, v_{s_j})$ and $\mathbf{A}_G$ is symmetric. For a vertex $v_s \in V$, let $N_G(v_s)$ denote the set of neighbors of v_s and $d_G(v_s)$ denote the degree of v_s. If G is directed, let $N_G^+(v_s)$ and $N_G^-(v_s)$ respectively denote the incoming and outgoing neighbors of v_s and $d_G^+(v_s)$ and $d_G^-(v_s)$ respectively denote the incoming and outgoing degree of v_s. Given a vector $a = (a_1, \ldots, a_N) \in \mathbb{R}^N$, $\mathrm{diag}(a)$ (or $\mathrm{diag}(a_1, \ldots, a_N)$) denotes a diagonal matrix with $a_n, n = 1, \ldots, N$ as diagonal entries.

A vector $\mathbf{x} \in \mathbb{R}^N$ is called a 1-dimensional graph signal on G, and $\mathbf{X} \in \mathbb{R}^{N \times d}$ is called a d-dimensiaonl graph signal on G. Throughout this article, $\mathbf{X}_V \in \mathbb{R}^{N \times d_v}$ (i.e., a d-dimensional graph signal) and $\mathbf{X}_E \in \mathbb{R}^{M \times d_e}$ respectively denote an initial vertex and edge feature matrix on G. Similarly, $\mathbf{X}_V^{(l)}$ and $\mathbf{X}_E^{(l)}$ respectively denote the vertex and edge hidden feature matrix on the lth layer. Obviously, $\mathbf{X}_V^{(0)} = \mathbf{X}_V$ and $\mathbf{X}_E^{(0)} = \mathbf{X}_E$. It is noted that sometimes, the edge (hidden) feature can also be incorporated into a 3-order tensor $\underline{\mathbf{X}}_E \in \mathbb{R}^{N \times N \times d_e}$ (or $\underline{\mathbf{X}}_E^{(l)} \in \mathbb{R}^{N \times N \times d_e}$). In general, let $\mathbf{x}_{j,k}^{(v)} = \mathbf{X}_V[j,k]$, $\mathbf{x}_{j,k}^{(e)} = \mathbf{X}_E[j,k]$, $\mathbf{x}_{j,k}^{(v,l)} = \mathbf{X}_V^{(l)}[j,k]$, and $\mathbf{x}_{j,k}^{(e,l)} = \mathbf{X}_E^{(l)}[j,k]$ respectively for the (j,k)-th entry of $\mathbf{X}_V$, $\mathbf{X}_E$, $\mathbf{X}_V^{(l)}$, and $\mathbf{X}_E^{(l)}$. Similarly, let $\mathbf{x}_{j,:}^{(v)} = \mathbf{X}_V[j,:]$, $\mathbf{x}_{j,:}^{(e)} = \mathbf{X}_E[j,:]$, $\mathbf{x}_{j,:}^{(v,l)} = \mathbf{X}_V^{(l)}[j,:]$, and $\mathbf{x}_{j,:}^{(e,l)} = \mathbf{X}_E^{(l)}[j,:]$ respectively for the jth row of $\mathbf{X}_V$, $\mathbf{X}_E$, $\mathbf{X}_V^{(l)}$, and $\mathbf{X}_E^{(l)}$, and $\mathbf{x}_{:,k}^{(v)} = \mathbf{X}_V[:,k]$, $\mathbf{x}_{:,k}^{(e)} = \mathbf{X}_E[:,k]$, $\mathbf{x}_{:,k}^{(v,l)} = \mathbf{X}_V^{(l)}[:,k]$, and $\mathbf{x}_{:,k}^{(e,l)} = \mathbf{X}_E^{(l)}[:,k]$ respectively for the kth column (i.e., a graph signal) of $\mathbf{X}_V$, $\mathbf{X}_E$, $\mathbf{X}_V^{(l)}$, and $\mathbf{X}_E^{(l)}$. Unless stated otherwise, $\mathbf{X} \in \mathbb{R}^{N \times d}$ (or $\mathbf{X}^{(l)} \in \mathbb{R}^{N \times d}$) denotes a d-dimensional graph signal (i.e., a vertex feature matrix) on G (at the lth layer). Let $\mathbf{I}_N$ denote a $N \times N$ identity matrix. For undirected graphs, $\mathbf{L}_G = \mathbf{D}_G - \mathbf{A}_G$ is called the Laplacian matrix of G, where $\mathbf{D}_G[r,r] = \sum_{c=1}^{N} \mathbf{A}_G[r,c]$. For a 1-dimensional graph signal $\mathbf{x}$, its smoothness $s(\mathbf{x})$ is defined as

$$s(\mathbf{x}) = \mathbf{x}^T \mathbf{L}_G \mathbf{x} = \frac{1}{2} \sum_{s,t=1}^{N} \mathbf{A}_G(s,t) \left(\mathbf{x}[s] - \mathbf{x}[t]\right)^2 . \tag{1}$$

The normalization of $\mathbf{L}_G$ is defined by $\overline{\mathbf{L}}_G = \mathbf{I}_N - \mathbf{D}_G^{-\frac{1}{2}} \mathbf{A}_G \mathbf{D}_G^{-\frac{1}{2}}$. $\overline{\mathbf{L}}_G$ is a real symmetric semi-positive definite matrix. So, it has N ordered real non-negative eigenvalues $\{\lambda_n : n = 1, \ldots, N\}$ and corresponding orthonormal eigenvectors $\{\mathbf{u}_n : n = 1, \ldots, N\}$, namely $\overline{\mathbf{L}}_G = \mathbf{U} \Lambda \mathbf{U}^T$, where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_N)$ and $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_N)$ denote an orthonormomal matrix. Without loss of generality, $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N = \lambda_{\max}$. The eigenvectors $\mathbf{u}_n, n = 1, \ldots, N$ are also called the graph Fourier bases of G. Obviously, the graph Fourier basis is also the 1-dimensional graph signal on G. The graph Fourier transform [49] for a given graph signal x can be denoted by

$$\hat{\mathbf{x}} \triangleq \mathcal{F}(\mathbf{x}) = \mathbf{U}^T \mathbf{x}. \tag{2}$$

The inverse graph Fourier transform can be correspondingly denoted by

$$\mathbf{x} \triangleq \mathcal{F}^{-1}(\hat{\mathbf{x}}) = \mathbf{U}\hat{\mathbf{x}}. \tag{3}$$

Note that the eigenvalue λ_n actually measures the smoothness of the graph Fourier mode u_n. Throughout this article, let $\rho(\cdot)$ denote an activation function, $< \cdot, \cdot >$ the inner product of two vectors/matrices, and $\bowtie$ the concatenation of at least two vectors.

3 FUNDAMENTAL ARCHITECTURES

This section introduces the fundamental architectures of GNNs in order to tell readers what building blocks the GNNs consist of.

3.1 Graph Convolutional Neural Networks (GCNNs)

The GCNNs play pivotal roles in tackling the irregular data (e.g., graph and manifold). They are motivated by the **Convolutional Neural Networks (CNNs)** to learn hierarchical representations of irregular data. There have been many efforts to generalize the CNN to graphs [158]. However, they are usually computationally expensive and cannot capture spectral or spatial features. In general, the GCNNs can be categorized into two classes: spectral GCNNs and spatial GCNNs. Table 1 shows an overview of the main spectral and spatial GCNNs.

3.1.1 Spectral Graph Convolution Operators. The spectral graph convolution operator is defined via the graph Fourier transform. For two graph signals $\mathbf{x}$ and $\mathbf{y}$ on G, their spectral graph convolution $\mathbf{x} *_G \mathbf{y}$ is defined by

$$\begin{aligned} \mathbf{x} *_G \mathbf{y} &= \mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}) \circledast \mathcal{F}(\mathbf{y})) \\ &= \mathbf{U}(\mathbf{U}^T\mathbf{x} \circledast \mathbf{U}^T\mathbf{y}) \\ &= \mathbf{U}\mathrm{diag}(\mathbf{U}^T\mathbf{y})\mathbf{U}^T\mathbf{x}, \end{aligned} \tag{4}$$

where $\circledast$ denotes the element-wise Hadamard product [65, 120, 167]. The spectral graph convolution can be rewritten as

$$\mathbf{x} *_G f_\theta = \mathbf{U}\mathbf{F}_\theta\mathbf{U}^T\mathbf{x},$$

where $\mathbf{F}_\theta$ is a diagonal matrix consisting of the learnable parameters. That is, the graph signal $\mathbf{x}$ is filtered by the spectral graph filter (or graph convolution kernel) $\mathbf{F}_\theta$. For a $d^{(l)}$-dimensional graph signal $\mathbf{X}^{(l)}$ on G, the output $\mathbf{X}^{(l+1)}$ yielded by a graph convolution layer, namely $d^{(l+1)}$-dimensional graph signal on G, can be written as

$$\mathbf{X}_V^{(l+1)}[:, k] = \rho\left(\sum_{j=1}^{d^{(l)}} \mathbf{U}\mathbf{F}_{\theta,j,k}^{(l)}\mathbf{U}^T\mathbf{X}_V^{(l)}[:, j]\right), \tag{5}$$

where $\mathbf{F}_{\theta,j,k}^{(l)}$ is a spectral graph filter, i.e., an $N \times N$ diagonal matrix consisting of learnable parameters corresponding to the jth graph signal at the lth layer and the kth graph signal at the $(l+1)$-th layer. The computational framework of the spectral GCNN in Equation (5) is demonstrated in Figure 2. It is worth noting that the calculation of the above graph convolution layer takes $O(N^3)$ time and $O(N^2)$ space to perform the eigendecomposition of $\overline{L}_G$ especially for large graphs. The article [246] proposes a regularization technique, namely GraphMix, to augment the vanilla GCNN with a parameter-sharing **Fully Connected Network (FCN)**.

Spectral Graph Filter. Many studies [167] focus on designing different spectral graph filters. In order to circumvent the eigendecomposition, the spectral graph filter $\mathbf{F}_\theta$ can be formulated as

Table 1. Overview of Main Spectral and Spatial GCNNs

Category	Approach	Task	Architecture	Improvement
Spectral GCNNs	ChebyNet [165]	graph classification	Chebyshev graph convolution, fully connected layers	strictly localized filters, low computational complexity
	GCN [237]	node classification	first-order spectral graph convolution, renormalization trick	a simple, fast, scalable, and well-behaved layer-wise propagation rule
	GWNN [16]	node classification	graph wavelet transform	high efficiency, high sparseness, localized convolution
	CayleyNet [206]	community detection, node classification, graph classification, link prediction	Cayley graph filter, graph coarsening, fully connected layer	localization, linear complexity, detecting narrow-frequency bands
	DAGNN [162]	node classification	MLP transformation, large receptive field, adaptive adjustment	Overcoming the oversmoothing issue
Spatial GCNNs	GN [195]	relational reasoning, combinatorial generalization	configurable within-block structure, composable multi-block architecture	a generic message passing framework
	MPNN [128]	graph classification	message function, update function, readout	a generic message passing framework of the day
	NLNN [268]	video classification, image recognition	Non-local block	a generic family of building blocks capturing long-range dependencies
	GraphSage [257]	node classification	mean aggregator, LSTM aggregator, pooling aggregator, neighbor sampling	a general inductive message passing framework
	HGCN [96]	link prediction, node classification	hyperbolic feature transform + attention-based neighborhood aggregation + non-linear activation with different curvatures	the first inductive hyperbolic GCN
	k-GNN [30]	graph classification and regression	aggregating messages from local and/or global neighborhood	taking higher-order graph structures at multiple scales into account
	DeepGCN [79]	point cloud semantic segmentation	residual/dense connections, dilated convolution	new ways to train very deep GCNs
	JK-Net [136]	node classification, graph classification	neighborhood aggregation, concatenation/max-pooling/LSTM-attention	skip connections

a K-localized polynomial of the eigenvalues of the normalized graph Laplacian $\overline{\mathbf{L}}_G$ [165, 208, 237], i.e.,

$$\mathbf{F}_\theta = \mathbf{F}_\theta(\Lambda) \triangleq \sum_{k=0}^{K-1} \theta_k \Lambda^k. \tag{6}$$

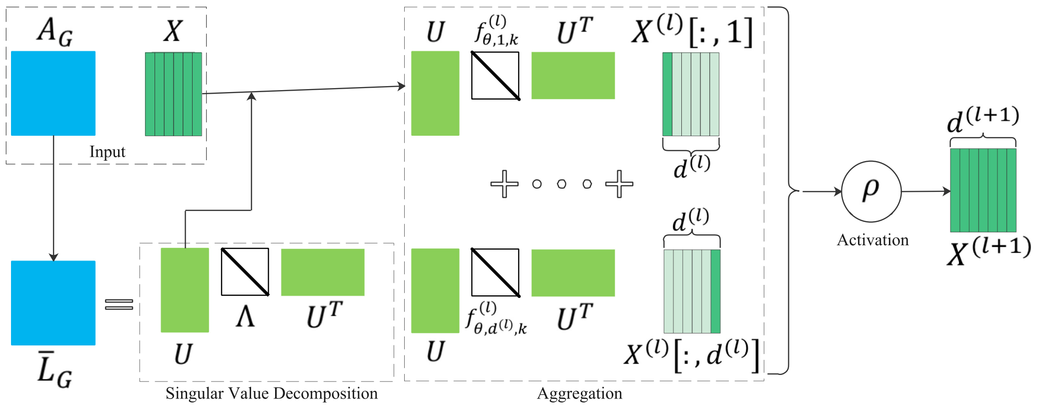

Fig. 2. Computational framework of the spectral GCNN.

In practice, the K-localized Chebyshev polynomial [165] is a favorable choice of formulating the spectral graph filter, i.e.,

$$\mathbf{F}_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\widetilde{\Lambda}),$$

where the Chebyshev polynomial is defined as

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \tag{7}$$

and $\widetilde{\Lambda} = \frac{2}{\lambda_{\max}}\Lambda - \mathbf{I}_N$. The reason that $\widetilde{\Lambda} = \frac{2}{\lambda_{\max}}\Lambda - \mathbf{I}_N$ is because it can map eigenvalues $\lambda \in [0, \lambda_{\max}]$ into $[-1, 1]$. This filter is K-localized in the sense that it leverages information from vertices that are at most K-hops away. In order to further decrease the computational cost, the first-order Chebyshev polynomial is used to define the spectral graph filter. Specifically, it lets $\lambda_{\max} \approx 2$ (because the largest eigenvalue of $\bar{\mathbf{L}}_G$ is less than or equal to 2 [33]) and $\theta = \theta_0 = -\theta_1$. Moreover, the renormalization trick is used here to mitigate the limitations of the vanishing/exploding gradient, namely substituting $\widetilde{\mathbf{D}}_G^{-\frac{1}{2}}\widetilde{\mathbf{A}}_G\widetilde{\mathbf{D}}_G^{-\frac{1}{2}}$ for $\mathbf{I}_N + \mathbf{D}_G^{-\frac{1}{2}}\mathbf{A}_G\mathbf{D}_G^{-\frac{1}{2}}$, where $\widetilde{\mathbf{A}}_G = \mathbf{A}_G + \mathbf{I}_N$ and $\widetilde{\mathbf{D}}_G = \mathrm{diag}(\sum_{k=1}^N \widetilde{\mathbf{A}}[1,k], \ldots, \sum_{k=1}^N \widetilde{\mathbf{A}}[N,k])$. As a result, the **Graph Convolutional Network (GCN)** [212, 237] can be defined as

$$\mathbf{X}_V^{(l+1)} = \rho\left(\widetilde{\mathbf{D}}_G^{-\frac{1}{2}}\widetilde{\mathbf{A}}_G\widetilde{\mathbf{D}}_G^{-\frac{1}{2}}\mathbf{X}_V^{(l)}\mathbf{W}^{(l)}\right). \tag{8}$$

The Chebyshev spectral graph filter suffers from a drawback that the spectrum of $\bar{\mathbf{L}}_G$ is linearly mapped into $[-1, 1]$. This drawback makes it hard to specialize in the low-frequency bands. In order to mitigate this problem, Levie and Bronstein et al. [206] proposes the Cayley spectral graph filter via the order-r Cayley polynomial $f_{c,h}(\lambda) = c_0 + 2\mathrm{Re}(\sum_{j=0}^r c_h C(\lambda)^j)$ with the Cayley transform $C(\lambda) = \frac{\lambda - i}{\lambda + i}$. The literature [327] employed the relationship between GCNNs and PageRank to devise an improved propagation scheme based on personalized PageRank, which incorporated a restarting factor into the message propagation operation. Moreover, there are many other spectral graph filters, e.g., [26, 67, 101, 201, 208, 213, 309], as well as capsule-based GCNNs [215, 272], which are inspired by the capsule network [214].

Graph Wavelet Neural Networks. As stated previously, the spectral and spatial GCNNs are respectively inspired by the graph Fourier transform and message-passing mechanism. Here, we introduce a new GCNN architecture from the perspective of the **Spectral Graph Wavelet Transform (SGWT)** [50]. First of all, the SGWT is determined by a graph wavelet generating kernel $g : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ with the property $g(0) = 0$, $g(+\infty) = \lim_{x\rightarrow\infty} g(x) = 0$. A feasible instance of $g(\cdot)$ is parameterized by two integers α and β and two positive real numbers x_1 and x_2 determining the transition regions, i.e.,

$$g(x;\alpha,\beta,x_1,x_2) = \begin{cases} x_1^{-\alpha}x^{\alpha} & x < x_1 \\ s(x) & x_1 \leq x \leq x_2 \\ x^{-\beta}x_2^{\beta} & x > x_2, \end{cases}$$

where $s(x)$ is a cubic polynomial whose coefficients can be determined by the continuity constraints $s(x_1) = s(x_2) = 1$, $s'(x_1) = \frac{\alpha}{x_1}$, and $s'(x_2) = -\frac{\beta}{x_2}$. Given the graph wavelet generating kernel $g(\cdot)$ and a scaling parameter $s \in \mathbb{R}^+$, the spectral graph wavelet operator Ψ_g^s is defined to be $\Psi_g^s = \mathbf{U}g(s\Lambda)\mathbf{U}^T$, where $g(s\Lambda) = g(\text{diag}(s\lambda_1, \ldots, s\lambda_N))$. A graph signal $\mathbf{x} \in \mathbb{R}^N$ on G can thereby be filtered by the spectral graph wavelet operator, i.e., $\mathcal{W}_{g,s}(\mathbf{x}) = \Psi_g^s\mathbf{x} \in \mathbb{R}^N$. The literature [59] utilizes a special instance of the graph wavelet operator Ψ_g^s to construct a graph scattering network and proves its covariance and approximate invariance to permutations and stability to graph operations. The literature [16] shows the operator Ψ_g^s to construct a **Graph Wavelet Neural Network (GWNN)**. Specifically, let $\Psi_g^{-s} \triangleq (\Psi_g^s)^{-1}$. The graph-wavelet-based convolution is defined to be

$$\mathbf{x} *_G \mathbf{y} = \Psi_g^{-s}\left(\Psi_g^s\mathbf{x} \circledast \Psi_g^s\mathbf{y}\right).$$

The GWNN is composed of multiple layers of the graph-wavelet-based convolution. The structure of the lth layer is defined as

$$\mathbf{X}_V^{(l+1)}[:,j] = \rho\left(\sum_{k=1}^{d^{(l)}} \Psi_g^{-s}\Theta_{j,k}^{(l)}\Psi_g^s\mathbf{X}_V^{(l)}[:,k]\right), \tag{9}$$

where $\Theta_{j,k}^{(l)}$ is a diagonal filter matrix learned in the spectral domain. Equation (9) can be rewritten as a matrix form, i.e., $\mathbf{X}_V^{(l+1)} = \rho(\Psi_g^{-s}\Theta^{(l)}\Psi_g^s\mathbf{X}_V^{(l)})$. The learnable filter matrix $\Theta^{(l)}$ can be replaced with the K-localized Chebyshev Polynomial so as to eschew the time-consuming eigendecomposition of $\overline{L}_G$.

Depth Trap of Spectral GCNNs. A bottleneck of GCNNs is that their performance may decrease with ever-increasing numbers of layers. This decay is often attributed to three factors: (1) overfitting resulting from the ever-increasing number of parameters, (2) gradient vanishing/explosion during training, and (3) oversmoothing, making vertices from different clusters more and more indistinguishable. The reason for oversmoothing is that performing the Laplacian smoothing many times forces the features of vertices within the same connected component to get stuck in stationary points [197]. There are some available approaches, e.g., [3, 149, 281, 297], to circumvent the depth trap of the spectral GCNNs. The literature [162] proposed new insights toward mitigating the oversmoothing problem and argued that the key factor compromising the performance significantly lies in the entanglement of representation transformation and propagation in current graph convolution operations. The oversmoothing problem can be mitigated by decoupling these two operations. Based on the above analysis, this article proposed **Deep Adaptive Graph Neural Network (DAGNN)** to adaptively incorporate information from large receptive fields.

3.1.2 *Spatial Graph Convolution Operators.*

Original spatial GCNNs [71, 72, 77, 247] constitute a transition function, which must be a contraction map in order to ensure the uniqueness of states,

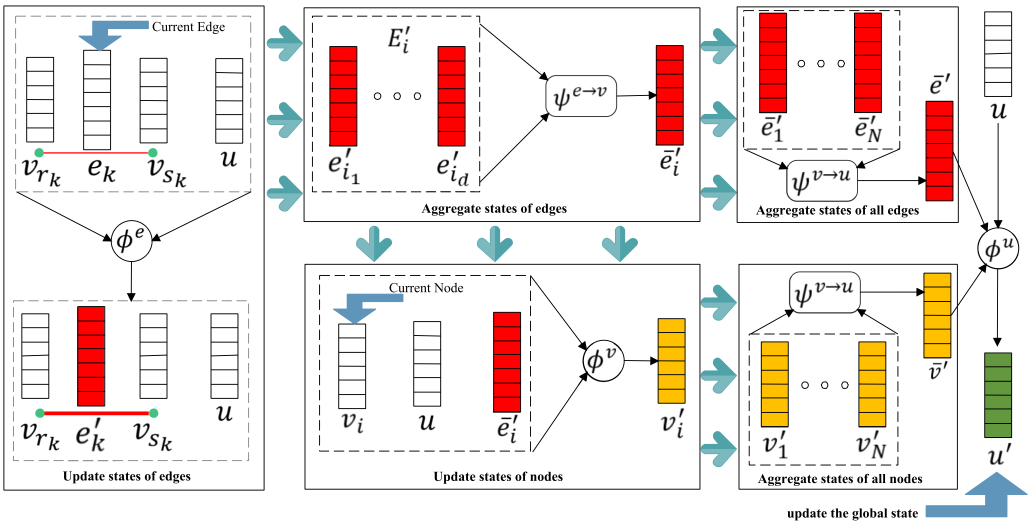

Fig. 3. Computational framework of the spatial GCNN.

and an update function. In the following, we first introduce a generic framework of the spatial GCNN and then investigate its variants.

Graph networks (GNs) as generic architectures with relational inductive bias [195] provide an elegant interface for learning entities, relations, and structured knowledge. Specifically, GNs are composed of GN blocks in a sequential, encode-process-decode or recurrent manner. GN blocks contain a message function $\phi^e(\cdot)$; two kinds of update functions, namely $\phi^v(\cdot), \phi^u(\cdot)$; and three kinds of aggregation functions, namely $\psi^{e\to v}(\cdot), \psi^{e\to u}(\cdot), \psi^{v\to u}(\cdot)$. The interactions between them (see Figure 3) are described as follows:

$$\begin{aligned}
\mathbf{x}_{k,:}^{(e,l+1)} &= \phi^e\left(\mathbf{x}_{k,:}^{(e,l)}, \mathbf{x}_{s_k,:}^{(v,l)}, \mathbf{v}_{t_k,:}^{(v,l)}, \mathbf{x}^{(u,l)}\right), & \hat{\mathbf{x}}_{i,:}^{(e,l+1)} &= \psi^{e\to v}\left(\hat{E}_i^{(l+1)}\right),\\
\mathbf{x}_{i,:}^{v,l+1} &= \phi^v\left(\mathbf{x}_{i,:}^{(v,l)}, \hat{\mathbf{x}}_{i,:}^{(e,l+1)}, \mathbf{x}^{(u,l)}\right), & \hat{\mathbf{x}}^{(e,l+1)} &= \psi^{e\to u}\left(\hat{E}^{(l+1)}\right),\\
\hat{\mathbf{x}}^{(v,l+1)} &= \psi^{v\to u}\left(\hat{V}^{(l+1)}\right), & \mathbf{x}^{(u,l+1)} &= \phi^u\left(\mathbf{x}^{(u,l)}, \hat{\mathbf{x}}^{(e,l+1)}, \hat{\mathbf{x}}^{(v,l+1)}\right),
\end{aligned} \tag{10}$$

where e_k is an arch from v_{s_k} to v_{t_k}, $\hat{E}_i^{(l+1)} = \{(\mathbf{x}_{k,:}^{(e,l+1)}, s_k, t_k) : t_k = i, k = 1, \ldots, M\}$, and $\hat{V}^{(l+1)} = \{\mathbf{x}_{i,:}^{(v,l+1)} : i = 1, \ldots, N\}$ and $\hat{E}^{(l+1)} = \{(\mathbf{x}_{k,:}^{(e,l+1)}, s_k, t_k) : k = 1, \ldots, M\}$. It is noted that the aggregation functions should be invariant to any permutations of vertices or edges. In practice, the GN framework can be used to implement a wide variety of architectures in accordance with three key design principles, namely flexible representations, configuable within-block structure, and flexible multi-block architectures. It is worth noting that the literature [288] found that the conventional neighborhood aggregation mechanism is not always necessary and helpful especially under two circumstances: (1) when a node's neighbors are highly dissimilar and (2) when a node's embedding is already similar to that of its neighbors. To resolve these two issues, the authors proposed two novel metrics, namely neighborhood entropy and center-neighbor similarity, to overcome these two shortcomings, and incorporated them into an adaptive-layer module. Below, we introduce three prevalent variants of the GNs, namely **Message Passing Neural Networks (MPNNs)** [128], **Non-local Neural Networks (NLNNs)** [268], and GraphSAGE [257].

Variants of GNs—MPNNs. MPNNs [128] have two phases, a message passing phase and a readout phase. The message passing phase is essentially composed of an edge-to-vertex aggregation function $\psi^{e\to v}(\cdot)$ (summation in MPNNs), a message function $\phi^e(\cdot)$, and a vertex update function

$\phi^v(\cdot)$. The interactions between them are described as follows:

$$\hat{\mathbf{x}}_{t,:}^{(v,l+1)} = \psi^{e\to v}\left(\left\{\phi^e\left(\mathbf{x}_{s,:}^{(v,l)}, \mathbf{x}_{t,:}^{(v,l)}, \mathbf{x}_{k,:}^{(e,l)}\right) : e_k = (s,t), s \in N_G(t)\right\}\right),$$
$$\mathbf{x}_{t,:}^{(v,l+1)} = \phi^v\left(\mathbf{x}_{t,:}^{(v,l)}, \hat{\mathbf{x}}_{t,:}^{(v,l+1)}\right).$$

The readout phase computes a universal feature vector for the whole graph using a readout function $R(\cdot)$, i.e., $\mathbf{x}^{(u,L)} = R(\{\mathbf{x}_{s,:}^{(v,L)} : s \in V\})$. It is noted that the readout function $R(\cdot)$ should be invariant to permutations of vertices. The GCNNs proposed in the literatures [48, 193, 229, 303] can be regarded as special forms of the MPNN.

Variants of GNs—NLNNs. NLNNs [268] give a general definition of non-local operations that are a flexible building block and can be easily integrated into convolutional/recurrent layers. Specifically, a generic non-local operation is defined as

$$\begin{aligned}\hat{\mathbf{x}}_{s,:}^{(v,l)} &= \frac{1}{C(\mathbf{x}_{s,:}^{(v,l)})}\sum_t f\left(\mathbf{x}_{s,:}^{(v,l)}, \mathbf{x}_{t,:}^{(v,l)}\right) g\left(\mathbf{x}_{t,:}^{(v,l)}\right),\\ \mathbf{x}_{s,:}^{(v,l+1)} &= \mathbf{W}_o\hat{\mathbf{x}}_{s,:}^{(v,l)} + \mathbf{W}_s\mathbf{x}_{s,:}^{(v,l)}\end{aligned} \tag{11}$$

where $f(\cdot,\cdot)$ denotes the affinity between $\mathbf{x}_{s,:}^{(v,l)}$ and $\mathbf{x}_{t,:}^{(v,l)}$, and $C(\mathbf{x}_{s,:}^{(v,l)}) = \sum_t f(\mathbf{x}_{s,:}^{(v,l)}, \mathbf{x}_{t,:}^{(v,l)})$ is a normalization factor. The affinity function $f(\cdot,\cdot)$ is of the following form:

(1) Gaussian: $f(\mathbf{x}_{s,:}^{(v,l)}, \mathbf{x}_{t,:}^{(v,l)}) = \exp\{\langle \mathbf{x}_{s,:}^{(v,l)}, \mathbf{x}_{t,:}^{(v,l)}\rangle\}$;
(2) Embedded Gaussian: $f(\mathbf{x}_{s,:}^{(v,l)}, \mathbf{x}_{t,:}^{(v,l)}) = \exp\{\langle \theta(\mathbf{x}_{s,:}^{(v,l)}), \eta(\mathbf{x}_{t,:}^{(v,l)})\rangle\}$, where $\theta(\mathbf{x}_{s,:}^{(v,l)}) = \mathbf{W}_\theta\mathbf{x}_{s,:}^{(v,l)}$ and $\eta(\mathbf{x}_{t,:}^{(v,l)}) = \mathbf{W}_\eta\mathbf{x}_{t,:}^{(v,l)}$;
(3) Dot Product: $f(\mathbf{x}_{s,:}^{(v,l)}, \mathbf{x}_{t,:}^{(v,l)}) = \theta(\mathbf{x}_{s,:}^{(v,l)})^T\eta(\mathbf{x}_{t,:}^{(v,l)})$;
(4) Concatenation: $f(\mathbf{x}_{s,:}^{(v,l)}, \mathbf{x}_{t,:}^{(v,l)}) = \mathrm{ReLU}(\mathbf{w}^T[\theta(\mathbf{x}_{s,:}^{(v,l)}), \eta(\mathbf{x}_{t,:}^{(v,l)})])$.

The first equation in Equation (11) denotes the composition of a message $\phi^e(\cdot)$ and an aggregation function $\psi^{e\to v}(\cdot)$, and the second one denotes an update function $\phi^v(\cdot)$.

Variants of GNs—GraphSAGE. GraphSAGE (SAmple and aggreGatE) [257] is a general inductive framework capitalizing on vertex feature information to efficiently generate vertex embedding vectors for previously unseen vertices. Specifically, GraphSAGE is composed of an aggregation function $\psi^{e\to v}(\cdot)$ and an update function $\phi^v(\cdot)$, i.e.,

$$\hat{\mathbf{x}}_{t,:}^{(v,l+1)} = \psi^{e\to v}\left(\left\{\mathbf{x}_{s,:}^{(v,l)} : s \in \ddot{N}_G(t)\right\} \cup \left\{\mathbf{x}_{t,:}^{(v,l)}\right\}\right)$$
$$\mathbf{x}_{t,:}^{(v,l)} = \phi^v\left(\left\{\mathbf{x}_{t,:}^{(v,l)}, \hat{\mathbf{x}}_{t,:}^{(v,l+1)}\right\}\right),$$

where $\ddot{N}_G(v)$ denotes a fixed-size set of neighbors of v uniformly sampling from its whole neighbors. The aggregation function is of the following form:

(1) Mean Aggregator: $\hat{\mathbf{x}}_{t,:}^{(v,l)} = \rho(\mathbf{W}\cdot\textsc{Mean}(\{\mathbf{x}_{t,:}^{(v,l)}\} \cup \{\mathbf{x}_{s,:}^{(v,l)} : s \in \ddot{N}_G(t)\}))$;
(2) LSTM Aggregator: applying the LSTM [217] to aggregate the selected neighbors of v;
(3) MaxPooling Aggregator: $\hat{\mathbf{x}}_{t,:}^{(v,l)} = \max(\{\rho(\mathbf{W}_s\mathbf{x}_{s,:}^{(v,l)} + \mathbf{b}_s) : s \in \ddot{N}_G(t)\} \cup \{\mathbf{W}_t\mathbf{x}_{t,:}^{(v,l)} + \mathbf{b}_t\})$.

Variants of GNs—Hyperbolic GCNNs. The Euclidean GCNNs aim to embed vertices in a graph into a Euclidean space. This will incur a large distortion especially when embedding real-world graphs with a scale-free and hierarchical structure. Hyperbolic GCNNs pave an alternative way of embedding with little distortion. The n-dimensional hyperbolic space [20, 203], denoted as $\mathbb{H}_K^n$, is a unique, complete, simply connected d-dimensional Riemannian manifold with constant negative sectional curvature $-\frac{1}{K}$, i.e.,

$$\mathbb{H}_K^n = \left\{\mathbf{x} \in \mathbb{R}^{n+1} : \langle \mathbf{x}, \mathbf{x}\rangle_{\mathcal{M}} = -K, x_0 = \mathbf{x}[0] > 0\right\},$$

where the Minkowski inner produces $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{M}} = -x_0 y_0 + \sum_{j=1}^{d} x_j y_j, \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^{n+1}$. Its tangent space centered at point x is denoted as $\mathcal{T}_{\mathbf{x}}\mathbb{H}_K^n = \{\mathbf{v} \in \mathbb{R}^{n+1} : \langle \mathbf{x}, \mathbf{v} \rangle_{\mathcal{M}} = 0\}$. Given $\mathbf{x} \in \mathbb{H}_K^n$, let $\mathbf{u} \in \mathcal{T}_{\mathbf{x}}\mathbb{H}_K^n$ be unit-speed. The unique unit-speed geodesic $\gamma_{\mathbf{x}\to\mathbf{u}}(\cdot)$ such that $\gamma_{\mathbf{x}\to\mathbf{u}}(0) = x$ and $\dot{\gamma}_{\mathbf{x}\to\mathbf{u}}(0) = u$ is denoted as $\gamma_{\mathbf{x}\to\mathbf{u}}(t) = \cosh(\frac{t}{\sqrt{K}})\mathbf{x} + \sqrt{K}\sinh(\frac{t}{\sqrt{K}}), \mathbf{u}, t > 0$. The intrinsic distance between two points $\mathbf{x}, \mathbf{y} \in \mathbb{H}_K^n$ is then equal to

$$d_{\mathcal{M}}^K(\mathbf{x}, \mathbf{y}) = \sqrt{K}\mathrm{arcosh}\left(-\frac{\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{M}}}{K}\right).$$

Therefore, the above n-dimensional hyperbolic space with constant negative sectional curvature $-\frac{1}{K}$ is usually denoted as $(\mathbb{H}_K^n, d_{\mathcal{M}}^K(\cdot,\cdot))$. In particular, $\mathbb{H}_1^n$, i.e., $K = 1$, is called the hyperboloid model of the hyperbolic space. **Hyperbolic Graph Convolutional Networks (HGCNs)** [96] benefit from the expressiveness of both GCNNs and hyperbolic embedding. It employs the exponential and logarithmic maps of the hyperboloid model, respectively denoted as $\exp_{\mathbf{x}}^K(\cdot)$ and $\log_{\mathbf{x}}^K(\cdot)$, to realize the mutual transformation between Euclidean features and hyperbolic ones. Let $\|\mathbf{v}\|_{\mathcal{M}} = \langle \mathbf{v}, \mathbf{v} \rangle_{\mathcal{M}}, \mathbf{v} \in \mathcal{T}_{\mathbf{x}}\mathbb{H}_K^n$. The $\exp_{\mathbf{x}}^K(\cdot)$ and $\log_{\mathbf{x}}^K(\cdot)$ are respectively defined to be

$$\exp_{\mathbf{x}}^K(\mathbf{v}) = \cosh\left(\frac{\|\mathbf{v}\|_{\mathcal{M}}}{\sqrt{K}}\right)\mathbf{x} + \sqrt{K}\sinh\left(\frac{\|\mathbf{v}\|_{\mathcal{M}}}{\sqrt{K}}\right)\frac{\mathbf{v}}{\|\mathbf{v}\|_{\mathcal{M}}}$$
$$\log_{\mathbf{x}}^K(\mathbf{y}) = d_{\mathcal{M}}^K(\mathbf{x}, \mathbf{y})\frac{\mathbf{y} + \frac{1}{K}\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{M}}\mathbf{x}}{\left\|\mathbf{y} + \frac{1}{K}\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{M}}\mathbf{x}\right\|_{\mathcal{M}}},$$

where $\mathbf{x} \in \mathbb{H}_K^n$, $\mathbf{v} \in \mathcal{T}_{\mathbf{x}}\mathbb{H}_K^n$, and $\mathbf{y} \in \mathbb{H}_K^n$ such that $\mathbf{y} \neq 0$ and $\mathbf{y} \neq \mathbf{x}$. The HGCN architecture is composed of three components: a **Hyperbolic Feature Transform (HFT)**, an **Attention-Based Aggregation (ABA),** and a **Non-Linear Activation with Different Curvatures (NLADC)**. They are respectively defined as

$$\mathbf{h}_{t,:}^{(H,l+1)} = \left(W^{(l+1)} \otimes^{K_l} \mathbf{x}_{t,:}^{(H,l)}\right) \oplus^{K_l} \mathbf{b}^{(l+1)} \qquad \text{(HFT)},$$
$$\mathbf{y}_{t,:}^{(H,l+1)} = \textsc{Aggregate}^{K_l}\left(\left\{\mathbf{h}_{s,:}^{(H,l+1)} : s \in N_G(t)\right\}\right) \qquad \text{(ABA)},$$
$$\mathbf{x}_{t,:}^{(H,l+1)} = \exp_{\mathbf{o}}^{K_{l+1}}\left(\rho\left(\log_{\mathbf{o}}^{K_l}\left(\mathbf{y}_{t,:}^{(H,l+1)}\right)\right)\right) \qquad \text{(NLADC)},$$

where $\mathbf{o} = (\sqrt{K}, 0, \ldots, 0) \in \mathbb{H}_K^n$ and $t \in V_G$. The linear transform in hyperboloid manifold is defined to be $\mathbf{W} \otimes^K \mathbf{x}_{t,:}^{(H)} = \exp_{\mathbf{o}}^K(\mathbf{W}\log_{\mathbf{o}}^K(\mathbf{x}_{t,:}^{(H)}))$ and $\mathbf{x}_{t,:}^{(H)} \oplus^K \mathbf{b} = \exp_{\mathbf{x}_{t,:}^{(H)}}^K(P_{\mathbf{o}\to\mathbf{x}_{t,:}^{(H)}}^K(\mathbf{b}))$, where $P_{\mathbf{o}\to\mathbf{x}^{(H)}}^K(\mathbf{b})$ is the parallel transport from $\mathcal{T}_{\mathbf{o}}\mathbb{H}_K^n$ to $\mathcal{T}_{\mathbf{x}^{(H)}}\mathbb{H}_K^n$. The attention-based aggregation is defined to be $\textsc{Aggregate}^K(\mathbf{x}_{t,:}^{(H)}) = \exp_{\mathbf{x}_{t,:}^{(H)}}^K(\sum_{s\in N_G(t)} \omega_{s,t} \log_{\mathbf{x}_{t,:}^{(H)}}^K(\mathbf{x}_{s,:}^{(H)}))$, where the attention weight $\omega_{s,t} = \mathrm{Softmax}_{s\in N_G(t)}(\mathrm{MLP}(\log_{\mathbf{o}}^K(\mathbf{x}_{t,:}^{(H)}) \bowtie \log_{\mathbf{o}}^K(\mathbf{x}_{s,:}^{(H)})))$.

Other Variants of GNs. In addition to the aforementioned GNs and its variants, there are still many other spatial GCNNs that are defined from other perspectives, e.g., **Diffusion-Convolutional Neural Network (DCNN)** [97], **Position-aware Graph Neural Network (P-GNN)** [112], **Memory-based Graph Neural Network (MemGNN), Graph Memory Network (GMN)** [7], **Graph Partition Neural Network (GPNN)** [200], **Edge-Conditioned Convolution (ECC)** [156], DEMO-Net [140], Column network [240], and Graph-CNN [66].

Higher-Order Spatial GCNNs. The aforementioned GCNN architectures are constructed from the microscopic perspective. They only consider vertices and edges yet overlook the higher-order substructures and their connections, i.e., subgraphs consisting of at least three vertices. Here, we introduce the studies on the k-dimensional GCNNs [30]. Specifically, they take higher-order graph structures at multiple scales into consideration by leveraging the **k-Weisfeiler-Leman (k-WL)**

graph isomorphism test so that the message passing is performed directly between subgraph structures rather than individual vertices. Let $\{\!\!\{\cdots\}\!\!\}$ denote a multiset, $\textsc{Hash}(\cdot)$ a hashing function, and $C_{l,k}^{(l)}(s)$ the vertex coloring (label) of $s = (s_1, \ldots, s_k) \in V^k$ at the lth time. Moreover, let $N_G^j(s) = \{(s_1, \ldots, s_{j-1}, r, s_{j+1}, \ldots, s_k) : r \in V\}$. The k-WL is computed by

$$C_{l,k}^{(l+1)}(s) = \textsc{Hash}\left(C_{l,k}^{(l)}(s), \left(c_1^{(l+1)}(s), \ldots, c_k^{(l+1)}(s)\right)\right),$$

where $c_j^{(l+1)}(s) = \textsc{Hash}(\{\!\!\{C_{l,k}^{(l)}(s') : s' \in N_G^j(s)\}\!\!\})$. The k-GCNN computes new features of $s \in V^k$ by multiple computational layers. Each layer is computed by

$$\mathbf{x}_{s,:}^{(k,l+1)} = \rho\left(\mathbf{x}_{s,:}^{(k,l)}\mathbf{W}_1^{(l)} + \sum_{t \in N_G(s)} \mathbf{x}_{s,:}^{(k,l)}\mathbf{W}_2^{(l)}\right).$$

In practice, the local k-GCNNs are often employed to learn the hierarchical representations of vertices in order to scale to larger graphs and mitigate the overfitting problem.

Invariance and Equivariance. Let $\mathbf{P}$ be a permutation matrix and $\mathbf{A}_G \in \mathbb{R}^{n^k}$ is a k-order tensor of edges or multi-edges in the (hyper)graph G. Permutation invariance refers to a function $f : \mathbb{R}^{n^k} \to \mathbb{R}$ (e.g., the aggregation function) being independent of any permutations of vertex/edge indices [80, 179], i.e., $f(\mathbf{P}^T\mathbf{A}_G\mathbf{P}) = f(\mathbf{A}_G)$. Permutation equivariance refers to a function $f : \mathbb{R}^{n^k} \to \mathbb{R}^{n^l}$ coinciding with permutations of vertex/edge indices [80, 179], i.e., $f(\mathbf{P}^T\mathbf{A}_G\mathbf{P}) = \mathbf{P}^T f(\mathbf{A}_G)\mathbf{P}$. For permutation-invariant aggregation functions, a straightforward choice is to take sum/ max /average/concatenation as heuristic aggregation schemes [179]. Nevertheless, these aggregation functions treat all the neighbors of a vertex equivalently so that they cannot precisely distinguish the structural effects of different neighbors to the target vertex. That is, the aggregation functions should extract and filter graph signals aggregated from neighbors of different hops away and different importance. GeniePath [324] proposes a scalable approach for learning adaptive receptive fields of GCNNs. It is composed of two complementary functions, namely adaptive breadth function and adaptive depth function. The former learns the importance of different-sized neighborhoods, whereas the latter extracts and filters graph signals aggregated from neighbors of different hops away. More specifically, the adaptive breadth function is defined as follows:

$$\mathbf{x}_{t,:}^{(v,l+1)} = \tanh\left(\left(\mathbf{W}^{(l+1)}\right)^T \sum_{s \in N_G(t) \cup \{t\}} \alpha\left(\mathbf{x}_{s,:}^{(l)}, \mathbf{x}_{t,:}^{(l)}\right) \cdot \mathbf{x}_{s,:}^{(l)}\right),$$

where $\alpha(\mathbf{x}, \mathbf{y}) = \text{Softmax}_{\mathbf{y}}(\alpha^T \tanh(\mathbf{W}_{\mathbf{x}}^T\mathbf{x} + \mathbf{W}_{\mathbf{y}}^T\mathbf{y}))$. The adaptive depth function is defined as a LSTM [217] consisting of an input gate $\mathbf{i}_{t,:}$, froget gate $\mathbf{f}_{t,:}$, and output gate $\mathbf{o}_{t,:}$, i.e.,

$$\begin{aligned}
\mathbf{i}_{t,:}^{(l+1)} &= \sigma\left(\left(\mathbf{W}_i^{(t)}\right)^T\left(\mathbf{x}_{t,:} \bowtie \mathbf{h}_{t,:}^{(l)}\right) + \mathbf{b}_i^{(l)}\right) & \mathbf{f}_{t,:}^{(l+1)} &= \sigma\left(\left(\mathbf{W}_f^{(l)}\right)^T\left(\mathbf{x}_{t,:} \bowtie \mathbf{h}_{t,:}^{(l)}\right) + \mathbf{b}_f^{(l)}\right) \\
\mathbf{o}_{t,:}^{(l+1)} &= \sigma\left(\left(\mathbf{W}_o^{(l)}\right)^T\left(\mathbf{x}_{t,:} \bowtie \mathbf{h}_{t,:}^{(l)}\right) + \mathbf{b}_o^{(l)}\right) & \widetilde{\mathbf{c}}_{t,:}^{(l+1)} &= \tanh\left(\left(\mathbf{W}_c^{(t)}\right)^T\left(\mathbf{x}_{t,:} \bowtie \mathbf{h}_{t,:}^{(l)}\right) + \mathbf{b}_c^{(l)}\right) \\
\mathbf{c}_{t,:}^{(l+1)} &= \mathbf{f}_{t,:}^{(l+1)} \circledast \mathbf{c}_{t,:}^{(l)} + \mathbf{i}_{t,:}^{(l+1)} \circledast \widetilde{\mathbf{c}}_{t,:}^{(l+1)} & \mathbf{h}_{t,:}^{(l+1)} &= \mathbf{o}_{t,:}^{((l+1))} \circledast \tanh\left(\mathbf{c}_{t,:}^{(l+1)}\right).
\end{aligned} \tag{12}$$

Geom-GCN [88] proposes a novel permutation-invariant geometric aggregation scheme consisting of three modules, namely vertex embedding, structural neighborhood, and bi-level aggregation. This aggregation scheme does not lose structural information of vertices and fails to capture long-range dependencies in disassortative graphs. For the permutation-invariant graph representations, PiNet [194] proposes an end-to-end spatial GCNN architecture that utilizes the permutation equivariance of graph convolutions. It is composed of a pair of double-stacked message passing layers, namely attention-oriented message passing layers and feature-oriented message passing layers.

Depth Trap of Spatial GCNNs. Similar to the spectral GCNNs, the spatial GCNNs are also confronted with the depth trap. As stated previously, the depth trap results from oversmoothing, overfitting, and gradient vanishing/explosion. In order to escape from the depth trap, some studies propose some available strategies, e.g., DeepGCN [79] and Jumping Knowledge Network [136]. The jumping knowledge networks [136] adopt neighborhood aggregation with skip connections to integrate information from different layers. The DeepGCN [79] applies the residual/dense connections [75, 130] and dilated aggregation [69] in the CNNs to construct the spatial GCNN architecture. They have three instantiations, namely ResGCN, DenseGCN, and dilated graph convolution. ResGCN is inspired by the ResNet [130], which is defined to be

$$\begin{aligned}\mathbf{X}_V^{(l+1)} &\triangleq \mathcal{H}\left(\mathbf{X}_V^{(l)}, \Theta^{(l)}\right)\\ &= \mathcal{F}\left(\mathbf{X}_V^{(l)}, \Theta^{(l)}\right) + \mathbf{X}_V^{(l)},\end{aligned}$$

where $\mathcal{F}(\cdot,\cdot)$ can be computed by spectral or spatial GCNNs. DenseGCN collectively exploits information from different GCNN layers like the DenseNet [75], which is defined to be

$$\begin{aligned}\mathbf{X}_V^{(l+1)} &\triangleq \mathcal{H}\left(\mathbf{X}_V^{(l)}, \Theta^{(l)}\right)\\ &= \left(\bowtie_{i=0}^{l} \mathcal{F}(\mathbf{X}_V^{(i)}\right) \bowtie \mathbf{X}_V^{(0)}.\end{aligned}$$

The dilated aggregation [69] can magnify the receptive field of spatial GCNNs by a dilation rate d. More specifically, let $N_G^{(k,d)}(v)$ denote the set of k d-dilated neighbors of vertex v in G. If $(u_1, u_2, \ldots, u_{k\times d})$ are the first sorted $k \times d$ nearest neighbors, then $N_G^{(k,d)}(v) = \{u_1, u_{1+d}, \ldots, u_{1+(k-1)d}\}$. Thereby, we can construct a new graph $G^{(k,d)} = (V^{(k,d)}, E^{(k,d)})$, where $V^{(k,d)} = V$ and $E^{(k,d)} = \{\langle v, u\rangle : v \in V, u \in N_G^{(k,d)}\}$. The dilated graph convolution layer can be obtained by running the spatial GCNNs over $G^{(k,d)}$. The literature [78] studied reversible connections, group convolutions, weight tying, and equilibrium models to advance the memory and parameter efficiency of GNNs, and found that reversible connections in combination with deep network architectures can expedite the training of overparameterized GNNs that significantly outperform existing methods on multiple graph datasets.

3.1.3 Summary. The aforementioned GCNN architectures provide available ingredients of constructing the GNNs. In practice, we can construct our own GCNNs by assembling different modules introduced above. Additionally, some scholars also study the GCNNs from some novel perspectives, e.g., the parallel computing framework of the GCNNs [148], the hierarchical covariant compositional networks [205], the transfer active learning for GCNNs [220], and quantum-walk-based subgraph convolutional neural network [313]. They are closely related to the GCNNs yet fairly different from the ones introduced above.

3.2 Accelerating Strategies for GNN Training

Although GCNNs have achieved remarkable success on node-, edge-, and graph-level tasks, training GCNNs on large graphs is still a big challenge. Original full-batch GCNN training must collect messages from all the neighbors of each vertex in the input graph per each GCNN layer. This results in high computational and memory costs. A trivial approach is to partition a large graph into disjoint subgraphs and then incorporate embedding matrices over these small subgraphs into a complete embedding matrix over the large graph. The literatures [252, 271] adopted this strategy to circumvent the memory and time issues. It is noted that partitioning a large graph also requires high time cost. A more feasible approach to mitigating this issue is to sample neighbors

Table 2. Overview of Main Accelerating Strategies for GNN Training

Category	Approach	Task	Architecture	Improvement
Node-wise	CV-Net [103]	node classification	control-variate-based estimator	reducing the bias and variance of the neighbor sampling
Layer-independent	FastGCN [114]	node classification	layer-wise importance sampling	efficient training, good generalization, variance reduction
	LGCL [90]	node classification	k-largest node selection, subgraph selection, 1-D convolution	learnable graph convolution layer enabling the use of regular convolutional operations
Layer-dependent	Adapt-Net [253]	node classification	top-down importance sampling, skip connection	adaptive layer-wise sampling, variance reduction, preserving second-order proximity
	LADIES [327]	node classification	layer-dependent importance sampling, normalization	avoiding exponential expansion of receptive field, guaranteeing the connectivity of the sampled vertices

from neighborhoods of each vertex and then train GNNs via the sampled subgraphs for training. Table 2 presents an overview of the main accelerating strategies for GNN training.

3.2.1 Node-wise Neighbor Sampling. A straightforward sampling method, named the neighbor sampling method, is to randomly choose neighbors from each vertex so as to reduce the receptive field size. Specifically, the aggregation function $\psi^{e\to v}$ should only collect messages from the sampled neighbors instead of the full ones, i.e.,

$$\hat{\mathbf{x}}_{i,:}^{(e,l+1)} = \psi^{e\to v}\left(\textsc{Sample}\left(\hat{E}_i^{(l+1)}, S^{(l)}\right)\right),$$

where $\textsc{Sample}(\cdot,\cdot)$ denotes the sampling function and $S^{(l)}$ is a hyperparameter for the number of sampled neighbors. The literatures [23, 257] adopted this kind of sampling method to accelerate the training of their GNNs on large graphs. The literature [103] developed a control-variate-based algorithm, which allowed sampling an arbitrarily small neighbor size. Though the neighbor sampling method achieves a remarkable convergence rate and to some extent mitigates the memory bottleneck of full-batch GNNs, the time complexity is still comparatively higher due to redundant computations for each layer resulting from independent aggregation and update operations in the same layer.

3.2.2 Layer-Independent Importance Sampling. A more advanced sampling method, named layer-wise importance sampling, is to conduct sampling on each layer with a specified sampling probability P. FastGCN [114] adopted this strategy to approximately evaluate the aggregation function. Without loss of generality, assume we obtain $N^{(l)}$ i.i.d. samples $v_1^{(l)}, \ldots, v_{N^{(l)}}^{(l)} \sim P$ at the lth layer. Thereby, the $(l+1)$-th hidden feature can be computed by

$$\hat{\mathbf{x}}_{N^{(l+1)}}^{(l+1)}[v,:] = \frac{1}{N^{(l)}}\sum_{j=1}^{N^{(l)}} \mathbf{K}\left[v, v_j^{(l)}\right] \mathbf{x}_{N^{(l)}}^{(l)}\left[v_j^{(l)}\right] \mathbf{W}^{(l)}, \quad x_{N^{(l+1)}}^{(l+1)}[v,:] = \rho\left(\hat{\mathbf{x}}_{N^{(l+1)}}^{(l+1)}[v,:]\right),$$

where $\mathbf{K}$ denotes the similarity matrix yielded by a kernel function. The literature [90] proposed a **learnable graph convolutional layer (LGCL),** which automatically selected a fixed number of neighbors for each vertex via a ranking strategy in order to convert input graphs into grid-like structures in 1-D format and then use regular convolution operations on the grid-like structures. In order to overcome the memory and time bottleneck of conventional GNNs on large graphs, LGCL randomly extracted subgraphs expanding from some initial seed vertices and trained its deep models on the randomly cropped subgraphs. Though the layer-independent importance sampling notably reduces both the memory and time complexities, it cannot guarantee connectivity between sampled vertices at different layers incurring a large variance of the final embeddings.

3.2.3 Layer-Dependent Importance Sampling. In order to overcome the drawbacks of the above two sampling methods, there have been several studies on layer-dependent importance sampling [253, 327]. In general, the layer-dependent sampling method selects their neighboring vertices according to the sampled vertices at the upper layer. The literature [253] proposed an adaptive layer-dependent sampler, which aimed to reduce the resulting variance. The optimal sampler with the minimum variance was uncomputable due to the inconsistency between the top-down sampling and the bottom-up propagation in GNNs. To tackle this issue, the authors replaced the uncomputable item with a self-dependent function and then integrated the variance into the loss function. The literature [327] proposed a novel effective sampling method, named **Layer-Dependent ImportancE Sampling (LADIES)**. Specifically, LADIES was designed in a top-down manner. That is to say, the sampled vertices at the lth layer depended on the sampled ones that were generated at all the upper layers. It is noted that each vertex only aggregated embeddings from its neighboring vertices at the upper layer. Suppose $S^{(l)}$ denotes a set of sampled vertices at the lth layer. The sample space at the $(l-1)$-th layer is thus $V^{(l-1)} = \cup_{v \in S^{(l)}} N_G(v)$, and the sampled vertices at this layer is denoted as $S^{(l-1)}$. Each vertex in $V^{(l-1)}$ is associated with an importance probability

$$p_j^{(l-1)} = \frac{\|\mathbf{Q}^{(l)}\mathbf{P}[:, j]\|_2^2}{\|\mathbf{Q}^{(l)}\mathbf{P}\|_F^2},$$

where $\mathbf{Q}^{(l)} \in \mathbb{R}^{|S^{(l)}| \times |V|}$ is a row selection matrix

$$\mathbf{Q}^{(l)}[k, s] = \begin{cases} 1 & (k, s) = \left(k, v_k^{(l-1)}\right) \\ 0 & \text{otherwise} \end{cases},$$

and $\mathbf{P}$ serves as an aggregation matrix, e.g., the normalized Laplacian matrix.

3.3 Graph Pooling Operators

Graph pooling operators are very important and useful modules of the GCNNs, especially for graph-level tasks such as the graph classification. The literature [164] evaluates the influence of the pooling operations on the success of GNNs and finds that, in contrast to the common belief, local pooling does not play a decisive role in the success of GNNs on relevant and widely used benchmarks. There are two kinds of graph pooling operators, namely global graph pooling operators and hierarchical graph pooling operators. The former aims to obtain the universal representations of input graphs, and the latter aims to capture adequate structural information for vertex representations. Table 3 presents an overview of the main graph pooling operators.

3.3.1 Global Graph Pooling Operators. Global graph pooling operators pool all representations of vertices into a universal graph representation. Many literatures [126, 194, 208] apply some simple global graph pooling operators, e.g., max/average/concatenate graph pooling, to performing graph-level classification tasks. Here, we introduce some more sophisticated global graph pooling operators in contrast to the simple ones. **Relational pooling (RP)** [209] provides a novel framework for graph representation with maximal representation power. Specifically, all vertex embeddings can be aggregated via a learnable function to form a global embedding of G. Let

Table 3. Overview of Main Graph Pooling Operators

Category	Approach	Task	Architecture	Improvement
global	RP [209]	graph classification	joint/separate relational pooling, GNN/MLP/RNN approximation	achieving maximal representation power for graphs and making existing graph representation models more powerful
	SortPooling [173]	graph classification	graph convolution layers, concatenation, sort pooling, 1-D convolution, fully connected layers	a novel spatial graph convolution layer to extract multi-scale vertex features, sorting the feature descriptors in a consistent order
hierarchical	EigenPooling [278]	graph classification	graph convolution, EigenPooling	naturally summarizing the subgraph information while utilizing the subgraph structure, theoretically understanding from local and global perspectives
	StructPool [84]	graph classification	graph pooling via node clustering, learning clustering assignments via CRFs, Gibbs energy with topology information	capturing higher-order structural relationships among the assignments of different nodes, incorporating the graph topological information
	DiffPool [202]	graph classification	pooling with an assignment matrix, learning the assignment matrix, auxiliary link prediction objective and entropy regularization	adapting to various GNNs in a hierarchical and end-to-end fashion, automatically learning a differentiable soft assignment for vertices
	SAGPool [127]	graph classification	graph convolution, top-rank selection, attention-based masking	learning hierarchical representations using relatively few parameters, considering both node features and graph topology
	gPool [73]	node classification	graph convolution, gPool for graph coarsening, gUnpool for graph restoring	proposing U-Net-like architectures for graph data

$\underline{\mathbf{X}}_E \in \mathbb{R}^{N\times N\times d_e}$ denote an edge feature tensor, which can be. The tensor $\underline{\mathbf{A}}_G \in \mathbb{R}^{N\times N\times(1+d_e)}$ concatenates the adjacency matrix $\mathbf{A}_G$ with its edge feature tensor $\underline{\mathbf{X}}_E$, i.e., $\underline{\mathbf{A}}_G[s,t,:] = \mathbb{I}((s,t) \in E_G) \bowtie \underline{\mathbf{X}}_E[s,t,:]$. After performing a permutation on V_G, let $\underline{\mathbf{A}}_G^{(\pi,\pi)}[\pi(s),\pi(t),:] = \underline{\mathbf{A}}_G[s,t,:]$ and the vertex feature matrix $\mathbf{X}_V^{(\pi)}[\pi(r),:] = \mathbf{X}_V[r,:]$. The joint RP permutation-invariant function for directed or undirected graphs is defined as

$$\bar{\bar{f}}(G) = \frac{1}{N!}\sum_{\pi\in\Pi_{|V|}} \vec{f}\left(\underline{\mathbf{A}}_G^{(\pi,\pi)}, \mathbf{X}_V^{(\pi)}\right),$$

where $\Pi_{|V|}$ is the set of all distinct permutations on V_G and $\vec{f}(\cdot,\cdot)$ is an arbitrary (possibly permutation-sensitive) vector-valued function. Specifically, $\vec{f}(\cdot,\cdot)$ can be denoted as **Multi-Layer Perceptrons (MLPs)**, **Recurrent Neural Networks (RNNs)**, CNNs, or GNNs. The literature [209] proves that $\bar{\bar{f}}(G)$ has the most expressive representation of G under some mild conditions and provides approximation approaches to making RP computationally tractable. In addition, there are some other available global graph pooling operators, e.g., SortPooling [173] and function space pooling [187].

3.3.2 Hierarchical Graph Pooling Operators. Hierarchical graph pooling operators group a set of proximal vertices into a super-vertex via graph clustering methods. As a result, the original graph is coarsened into a new graph with coarser granularity. Graph clustering is an indispersible operation for the hierarchical graph pooling operators. There are usually two available graph clustering methods, i.e., spectral clustering [120] and Graclus [95]. The former suffered from an expensive computational cost since eigenvector computation is prohibitive especially for large-scale matrices. The latter incorporated the weighted kernel k-means [94] and weighted graph association and graph cut clustering [13, 102, 188] into a unified matrix trace maximization framework and exploited it to develop a fast high-quality multilevel algorithm including three phases: coarsening phase, base clustering phase, and refining phase. The hierarchical graph pooling operators are usually interleaved with the vanilla GCNN modules for use in practice. In general, there are three kinds of approaches to performing the graph coarsening operations: (1) invoking the existing graph clustering algorithms, (2) learning a soft cluster assignment, and (3) selecting the first k top-rank vertices.

Invoking existing graph clustering algorithms. The graph clustering aims to assign proximal vertices to the same cluster and in-proximal vertices to different clusters. The coarsened graph regards the resulting clusters as super-vertices and connections between two clusters as super-edges. The hierarchical graph pooling operators aggregate the representations of vertices in super-vertices via aggregation functions such as max pooling and average pooling [230] to compute the representations of super-vertices. The literature [278] proposed the EigenPooling method and presented the relationship between the original and coarsened graph. In order to construct a coarsened graph of G, a graph clustering method is employed to partition G into K disjoint clusters, namely $\{G_k : k = 1, \ldots, K\}$. Suppose each cluster G_k has N_k vertices, namely $\{v_{k,1}, \ldots, v_{k,N_k}\}$, and its adjacency matrix is denoted as A_{G_k}. The coarsened graph G_{coar} of G can be constructed by regarding the clusters $G_k, k = 1, \ldots, K$ as super-vertices and connections between two super-vertices as edges. For G_k, its sampling matrix $\mathbf{C}_k$ of size $(N \times N_k)$ is defined by

$$\mathbf{C}_k(s,t) = \begin{cases} 1 & \text{if vertex } v_{k,s} \text{ in } G_k \text{ is identical to vertex } v_t \text{ in } G \\ 0 & \text{otherwise.} \end{cases}$$

On one hand, $\mathbf{C}_k$ can be used to down-sample a 1-dimensional graph signal x on G to obtain an induced graph signal $\mathbf{x}[G_k]$ on the induced subgraph G_k, i.e., $\mathbf{x}[G_k] = \mathbf{C}_k^T\mathbf{x}$. On the other hand, $\mathbf{C}_k$ can also be used to up-sample a graph signal $\mathbf{x}[G_k]$ on G_k to obtain a dilated graph G, i.e., $\mathbf{x} = \mathbf{C}_k\mathbf{x}[G_k]$. Furthermore, the adjacency matrix A_{G_k} of G_k can be computed by

$$\mathbf{A}_{G_k} = \mathbf{C}_k^T \mathbf{A}_G \mathbf{C}_k.$$

The intra-subgraph adjacency matrix of G is computed by $\mathbf{A}_{\text{intra}} = \sum_{k=1}^{K} \mathbf{C}_k \mathbf{A}_{G_k} \mathbf{C}_k^T$. Thereby, the inter-subgraph adjacency matrix of G can be computed by $\mathbf{A}_{\text{inter}} = \mathbf{A}_G - \mathbf{A}_{\text{intra}}$. Let $\mathbf{M}_{\text{coar}} \in \mathbb{R}^{N \times K}$ denote the assignment matrix from G to G_{coar}. Its (j,k)-th entry is defined as

$$\mathbf{M}_{\text{coar}}[j,k] = \begin{cases} 1 & \text{if } v_j \text{ in } G \text{ is grouped into } G_k \text{ in } G_{\text{coar}} \\ 0 & \text{otherwise.} \end{cases}$$

As a result, the adjacency matrix $\mathbf{A}_{\text{coar}}$ of the coarsened graph G_{coar} is computed by $\mathbf{A}_{\text{coar}} = \mathbf{M}_{\text{coar}}^T \mathbf{A}_{\text{inter}} \mathbf{M}_{\text{coar}}$. In fact, $\mathbf{A}_{\text{coar}}$ can be written as $\mathbf{A}_{\text{coar}} = f(\mathbf{M}_{\text{coar}}^T \mathbf{A}_G \mathbf{M}_{\text{coar}})$ as well, where $f(\widetilde{a}_{i,j}) = 1$ if $\widetilde{a}_{i,j} > 0$ and $f(\widetilde{a}_{i,j}) = 0$ otherwise. As stated previously, $\mathbf{X}_V$ is a d-dimensional graph signal on G.

Then, a d-dimensional graph signal $\mathbf{X}_V^{(\text{coar})}$ on G_{coar} can be computed by $\mathbf{X}_V^{(\text{coar})} = \mathbf{M}_{\text{coar}}^T \mathbf{X}_V$. EigenPooling [278] employs spectral clustering to obtain the coarsened graph and then up-sample the Fourier bases of subgraphs $G_k, k = 1, \ldots, K$. These Fourier bases are then organized into pooling operators with regard to ascending eigenvalues. Consequently, the pooled vertex feature matrix is obtained via concatenating the pooled results. The literature [68] proposes a novel **Hierarchical Graph Convolutional Network (H-GCN)** consisting of graph coarsening layers and graph refining layers. The former employs structural equivalence grouping and structural similarity grouping to construct the coarsened graph, and the latter restores the original topological structure of the corresponding graph.

Learning a soft cluster assignment. StructPool [84], as a structured graph pooling technique, regards the graph pooling as a graph clustering problem so as to learn a cluster assignment matrix via the feature matrix $\mathbf{X}_V$ and adjacency matrix $\mathbf{A}_G$. Learning the cluster assignment matrix can formulated as a **Conditional Random Field (CRF)**-based [124] probabilistic inference. Specifically, the input feature matrix $\mathbf{X}$ is treated as global observation, and $\mathbf{Y} \in \mathbb{R}^{N \times K}$ is a random field where $\mathbf{Y}[i,:] \in \{1, \ldots, K\}$ is a random variable indicating which clusters the vertex v_i is assigned to. As a result, $(\mathbf{Y}, \mathbf{X})$ can be characterized by a CRF model, i.e.,

$$\begin{aligned} \mathbb{P}(\mathbf{Y}|\mathbf{X}) &= \frac{1}{Z(\mathbf{X})} \exp\left(-\mathcal{E}(\mathbf{Y}|\mathbf{X})\right) \\ &= \frac{1}{Z(\mathbf{X})} \exp\left(\sum_{C \in C_G} \psi_C(\mathbf{Y}_C|\mathbf{X}) \right), \end{aligned}$$

where $\mathcal{E}(\mathbf{Y}|\mathbf{X}) = -\sum_{C \in C_G} \psi_C(\mathbf{Y}_C|\mathbf{X})$ is called an energy function, C_G is a set of cliques, $\psi_C(\mathbf{Y}_C|\mathbf{X})$ is a potential function, and $Z(\mathbf{X})$ is a partition function. The energy function $\mathcal{E}(\mathbf{Y}|\mathbf{X})$ can be characterized by a unary energy $\psi_u(\cdot)$ and a pairwise energy $\psi_p(\cdot,\cdot)$, i.e.,

$$\mathcal{E}(\mathbf{Y}|\mathbf{X}) = \sum_{s=1}^{N} \psi_u(\mathbf{y}_{s,:}|\mathbf{X}) + \sum_{s \neq t} \psi_p(\mathbf{y}_{s,:}, \mathbf{y}_{t,:}|\mathbf{X}) a_{s,t}^{(l)},$$

where $a_{s,t}^{(l)}$ denotes the (s,t)-th entry of the l-hop adjacency matrix $\mathbf{A}_G^{(l)}$. The unary energy matrix $\Psi_u = (\psi_u(\mathbf{y}_{s,:}|\mathbf{X}))_{N \times K}$ can be obtained by a GCNN taking the global observation $\mathbf{X}$ and the adjacency $\mathbf{A}_G$ as input. The pairwise energy matrix $\Psi_p = (\psi_p(\mathbf{y}_{s,:}, \mathbf{y}_{t,:}|\mathbf{X}))_{K \times K}$ can be obtained by

$$\psi_p(\mathbf{y}_{s,:}, \mathbf{y}_{t,:}|X) = \mu(\mathbf{y}_{s,:}, \mathbf{y}_{t,:}) \frac{\mathbf{x}_{s,:}^T \mathbf{x}_{t,:}}{\sum_{j \neq s} \mathbf{x}_{s,:}^T \mathbf{x}_{j,:}},$$

where $\mu(\mathbf{y}_{s,:}, \mathbf{y}_{t,:})$ is a learnable compatibility function. Minimizing the energy function $\mathcal{E}(\mathbf{Y}|\mathbf{X})$ via mean-field approximation results in the most probable cluster assignment matrix $\mathbf{M}$ for a given graph G. As a result, we obtain a new graph $\mathbf{A}_{\text{coar}} = f(\mathbf{M}^T \mathbf{A}_G \mathbf{M})$ and $\mathbf{X}_{\text{coar}} = \mathbf{M}^T \mathbf{X}$. DiffPool [202] is a differentiable graph pooling operator that can generate hierarchical representations of graphs and can be incorporated into various GCNNs in an end-to-end fashion. It maps an adjacency matrix $\mathbf{A}_{G^{(l)}}$ and embedding matrix $\mathbf{Z}^{(l)}$ at the lth layer to a new adjacency matrix $\mathbf{A}_{G^{(l+1)}}$ and a coarsened feature matrix $\mathbf{X}^{(l+1)}$, i.e., $(\mathbf{A}_{G^{(l+1)}}, \mathbf{X}^{(l+1)}) = \text{DiffPool}(\mathbf{A}_{G^{(l)}}, \mathbf{Z}^{(l)})$. More specifically, $\mathbf{X}^{(l+1)} = (\mathbf{M}^{(l)})^T \mathbf{Z}^{(l)}$, $\mathbf{A}_{G^{(l+1)}} = (\mathbf{M}^{(l)})^T \mathbf{A}_{G^{(l)}} \mathbf{M}^{(l)}$. Note that the assignment matrix $\mathbf{M}^{(l)}$ and embedding matrix $\mathbf{Z}^{(l)}$ are respectively computed by two separate GCNNs, namely embedding GCNN and pooling GCNN, i.e., $\mathbf{Z}^{(l)} = \text{GCNN}_{\text{embed}}(\mathbf{A}_{G^{(l)}}, \mathbf{X}^{(l)})$, $\mathbf{M}^{(l)} = \text{Softmax}(\text{GCNN}_{\text{pool}}(\mathbf{A}_{G^{(l)}}, \mathbf{X}^{(l)}))$. The RepPool [125] puts an eye on the non-selected vertices that are overlooked; introduces the concept of representativeness, which is combined with the importance of nodes; and provides a learnable way to integrate non-selected nodes.

Selecting the first k top-rank vertices. The literature [127] proposes a novel **Self-Attention Graph Pooling operator (SAGPool)**. Specifically, SAGPool first employs the GCN [237] to calculate the self-attention scores and then invokes the top-rank function to select the top $\lceil kN \rceil$ vertex indices, i.e.,

$$\begin{array}{lll} \mathbf{Z} = \rho\left(\widetilde{\mathbf{D}}_G^{-\frac{1}{2}}\widetilde{\mathbf{A}}_G\widetilde{\mathbf{D}}_G^{-\frac{1}{2}}\mathbf{X}\Theta\right), & \text{idx} = \text{top-rank}\left(\mathbf{Z}, \lceil kN \rceil\right), & \mathbf{Z}_{\text{mask}} = \mathbf{Z}_{\text{idx}} \\ \mathbf{X}' = \mathbf{X}_{\text{idx}}, & \mathbf{X}_{\text{out}} = \mathbf{X}' \circledast \mathbf{Z}_{\text{mask}}, & \mathbf{A}_{\text{out}} = \mathbf{A}_{\text{idx},\text{idx}}. \end{array}$$

As a result, the selected top-$\lceil kN \rceil$ vertex indices are employed to extract the output adjacency matrix $\mathbf{A}_{\text{out}}$ and feature matrix $\mathbf{X}_{\text{out}}$. In order to exploit the expressive power of an encoder-decoder architecture like U-Net [183], the literature [73] proposes a novel **graph pooling (gPool)** layer and a **graph unpooling (gUnpool)** layer. The gPool adaptively selects the top-k ranked vertex indices by the down-sampling technique to form a coarsened graph ($\mathbf{A}_{\text{coar}} \in \mathbb{R}^{N\times N}$ and $\mathbf{X}_{\text{coar}} \in \mathbb{R}^{N\times d}$) based on scalar projection values on a learnable projection vector, i.e.,

$$\begin{array}{lll} \mathbf{y} = \frac{\mathbf{Xp}}{\|\mathbf{p}\|}, & \text{idx} = \text{top-rank}(\mathbf{y}, k), & \widetilde{\mathbf{y}} = \tanh(\mathbf{y}_{\text{idx}}) \\ \widetilde{\mathbf{X}}_{\text{coar}} = \mathbf{X}_{\text{idx},:}, & \mathbf{A}_{\text{coar}} = \mathbf{A}_{\text{idx},\text{idx}}, & \mathbf{X}_{\text{coar}} = \widetilde{\mathbf{X}}_{\text{coar}} \circledast (\widetilde{\mathbf{y}}\mathbf{1}_C^T), \end{array}$$

where $\mathbf{y} \in \mathbb{R}^d$. The gUnpool performs the inverse operation of the gPool layer so as to restore the coarsened graph into its original structure. To this end, gUnpool records the locations of vertices selected in the corresponding gPool layer and then restores the selected vertices to their original positions in the graph. Specifically, let $\mathbf{X}_{\text{refine}} = \text{Distribute}(\mathbf{0}_{N\times d}, \mathbf{X}_{\text{coar}}, \text{idx})$, where the function $\text{Distribute}(\cdot,\cdot,\cdot)$ distributes row vectors in $\mathbf{X}_{\text{coar}}$ into $\mathbf{0}_{N\times d}$ feature matrix according to the indices idx. Note that row vectors of $\mathbf{X}_{\text{refine}}$ with indices in *idx* are updated by the ones in $\mathbf{X}_{\text{coar}}$, whereas other row vectors remain zero. It is worth noting that the literature [38] adopts the similar pooling strategy as gPool to learn the hierarchical representations of vertices.

3.4 Graph Attention Mechanisms

Attention mechanisms, first introduced in the deep learning community, guide deep learning models to focus on the task-relevant part of its inputs so as to make precise predictions or inferences [12, 60, 248]. Recently, applying the attention mechanisms to GCNNs has gained considerable attention so that various attention techniques have been proposed. Below, we summarize the graph attention mechanisms on graphs from the next four perspectives [122], namely concatenation-based graph attention, similarity-based graph attention, spectral graph attention, and attention-guided walk. Table 4 presents an overview of the main graph attention mechanisms. Without loss of generality, the neighbors of a given vertex v_0 in G are denoted as $v_1, \ldots, v_{d_0}$, and their current feature vectors are respectively denoted as $\mathbf{x}_{0,:}, \mathbf{x}_{1,:}, \ldots, \mathbf{x}_{d_0,:}$, where $d_0 = d_G(v_0)$.

3.4.1 Concatenation-based Graph Attention. The concatenation-based graph attention is typically implemented by employing a softmax with learnable weights [191, 299] to measure the relevance of $v_j, j = 1, \ldots, d_G(v_0)$ to v_0. More specifically, the concatenation-based attention weights between v_0 and v_j can be defined as

$$\begin{aligned} \omega_{0,j} &= \text{Softmax}([e_{0,1}, \ldots, e_{0,d_G(v_0)}]) \\ &= \frac{\exp(\rho(\mathbf{a}^T(\mathbf{Wx}_0 \bowtie \mathbf{Wx}_j)))}{\sum_{k=1}^{d_G(v_0)} \exp(\rho(\mathbf{a}^T(\mathbf{Wx}_0 \bowtie \mathbf{Wx}_k)))}, \end{aligned} \tag{13}$$

Table 4. Overview of Graph Attention Mechanisms

Category	Approach	Task	Architecture	Improvement
concatenation	GAT [191]	node classification, graph classification	graph attention layer with multiple heads	addressing several key challenges of spectral GCNNs, applied to inductive as well as transductive problems
	GTR [299]	few-shot learning, medical abnormality, disease classification, graph classification	inter-graph message passing, intra-graph message passing, attention on source graph, self-attention on target graph	capturing long-range dependency with global attention, enabling dynamic graph structures
similarity	AGNN [141]	node classification	graph linear network, attention mechanisms over neighbors	removing all the intermediate fully connected layers, replacing the propagation layers with attention mechanisms
spectral	SpGAT [87]	node classification	graph wavelet, spectral convolution for low-frequency components and high-frequency components with spectral attention weights	introducing attentions in the spectral domain of graphs, learning representations for different frequency components

where $e_{0,j} = \exp(\rho(\mathbf{a}^T(\mathbf{W}\mathbf{x}_0 \bowtie \mathbf{W}\mathbf{x}_j)))$, $\mathbf{a}$ is a learnable attention vector, and $\mathbf{W}$ is a learnable weight matrix; see Figure 4(a). As a result, the new feature vector of v_0 can be updated by

$$\mathbf{x}'_{0,:} = \rho\left(\sum_{j=1}^{d_G(v_0)} \omega_{0,j}\mathbf{W}\mathbf{x}_{j,:}\right). \tag{14}$$

In practice, multi-head attention mechanisms are usually employed to stabilize the learning process of the single-head attention [191]. For the multi-head attention, assume that the feature vector of each head is $\mathbf{x}_{0,:}^{(h)} = \rho(\sum_{j=1}^{d_G(v_0)} \omega_{0,j}^{(h)}\mathbf{W}^{(h)}\mathbf{x}_{j,:})$. The concatenation-based multi-head attention is computed by $\mathbf{x}'_{0,:} = \bowtie_{h=1}^{H} \mathbf{x}_{0,:}^{(h)} = \bowtie_{h=1}^{H} \rho(\sum_{j=1}^{d_G(v_0)} \omega_{0,j}^{(h)}\mathbf{W}^{(h)}\mathbf{x}_{j,:})$. The average-based multi-head attention is computed by $\mathbf{x}'_{0,:} = \rho(\frac{1}{H}\sum_{h=1}^{H}\sum_{j=1}^{d_G(v_0)} \omega_{0,j}^{(h)}\mathbf{W}^{(h)}\mathbf{x}_{j,:})$.

The conventional multi-head attention mechanism treats all the attention heads equally so that feeding the output of an attention that captures a useless representation may mislead the final prediction of the model. The literature [105] computes an additional soft gate to assign different weights to heads and gets the formulation of the gated multi-head attention mechanism. The **Graph Transformer (GTR)** [299] can capture long-range dependencies of dynamic graphs with softmax-based attention mechanism by propagating features within the same graph structure via an intra-graph message passing. The Graph-BERT [108] is essentially a pre-training method only based on the graph attention mechanism without any graph convolution or aggregation operators. Its key component is called a graph transformer-based encoder, i.e., $\mathbf{X}^{(l+1)} = \text{Softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_h}})\mathbf{V}$, where $\mathbf{Q} = \mathbf{X}^{(l)}\mathbf{W}_Q^{(l+1)}$, $\mathbf{K} = \mathbf{X}^{(l)}\mathbf{W}_K^{(l+1)}$, and $\mathbf{V} = \mathbf{X}^{(l)}\mathbf{W}_V^{(l+1)}$. The Graph2Seq [142] is a general end-to-end graph-to-sequence neural encoder-decoder model converting an input graph to a

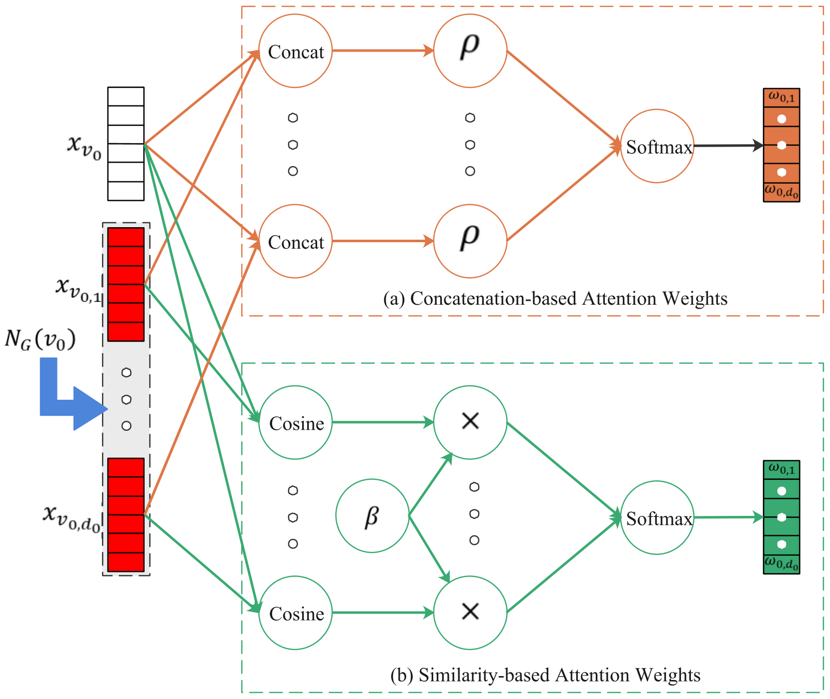

Fig. 4. Two kinds of graph attention mechanisms.

sequence of vectors with the attention-based LSTM model. It is composed of a graph encoder, a sequence decoder, and a vertex attention mechanism. The sequence decoder takes outputs (vertex- and graph-level representations) of the graph encoder as input and employs the softmax-based attention to compute the context vector sequence.

3.4.2 Similarity-based Graph Attention. The similarity-based graph attention depends on the cosine similarities of the given vertex v_0 and its neighbors $v_j, j = 1, \dots, d_G(v_0)$. More specifically, the similarity-based attention weights are computed by

$$\omega_{0,j} = \frac{\exp(\beta \cdot \cos(\mathbf{W}\mathbf{x}_{0,:}, \mathbf{W}\mathbf{x}_{j,:}))}{\sum_{k=1}^{d_G(v_0)} \exp(\beta \cdot \cos(\mathbf{W}\mathbf{x}_{0,:}, \mathbf{W}\mathbf{x}_{k,:}))}, \tag{15}$$

where β is learnable bias and W is a learnable weight matrix; see Figure 4(b). It is well known that $\cos(\mathbf{x}, \mathbf{y}) = \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\|_2 \|\mathbf{y}\|_2}$. **Attention-based Graph Neural Network (AGNN)** [141] adopts the similarity-based attention to construct the propagation matrix $\mathbf{P}^{(l)}$ capturing the relevance of v_j to v_i. As a result, the output hidden representation $\mathbf{X}^{(l+1)}$ at the $(l+1)$-th layer is computed by

$$\mathbf{X}^{(l+1)} = \rho\left(\mathbf{P}^{(l)}\mathbf{X}^{(l)}\mathbf{W}^{(l)}\right),$$

where $\mathbf{P}^{(l)}[i, j] \triangleq \omega_{i,j}$ is defined in Equation (15).

3.4.3 Spectral Graph Attention. The **Spectral Graph Attention Network (SpGAT)** aims to learn representations for different frequency components [87]. The eigenvalues of the normalized graph Laplacian $\overline{L}_G$ can be treated as frequencies on the graph G. As stated in the Preliminary section, $0 = \lambda_1 \le \lambda_2 \le \cdots \le \lambda_N = \lambda_{\max}$. The SpGAT first extracts the low-frequency component

$\mathbf{B}_L = (\mathbf{u}_{:,1}, \ldots, \mathbf{u}_{:,d}) \in \mathbb{R}^{(N \times d)}$ and the high-frequency component $\mathbf{B}_H = (\mathbf{u}_{:,d+1}, \ldots, \mathbf{u}_{:,N}) \in \mathbb{R}^{(N \times (N-d+1))}$ from the graph Fourier bases $\{u_1, u_2, \ldots, u_N\}$. So, we have

$$\begin{aligned} &\mathbf{X}_L = \mathbf{X}^{(l)}\Theta_L, \quad \mathbf{X}_H = \mathbf{X}^{(l)}\Theta_H \\ &\mathbf{X}^{(l+1)} = \rho\left(\textsc{Aggregate}\left(\mathbf{B}_L\mathbf{F}_L\mathbf{B}_L^T\mathbf{X}_L, \mathbf{B}_H\mathbf{F}_H\mathbf{B}_H^T\mathbf{X}_H\right)\right), \end{aligned} \tag{16}$$

where $\mathbf{F}_L$ and $\mathbf{F}_H$ respectively measure the importance of the low and high frequency, and $\textsc{Aggregate}(\cdot,\cdot)$ denotes an aggregation function of two matrices, e.g. $\textsc{Sum}$, $\textsc{Max}$, and so forth. In practice, we exploit a re-parameterization trick to accelerate the training. More specifically, we replace $\mathbf{F}_L$ and $\mathbf{F}_H$ respectively with the learnable attention weights $\Omega_L = \mathrm{diag}(\omega_L, \ldots, \omega_L)$ and $\Omega_H = \mathrm{diag}(\omega_H, \ldots, \omega_H)$ so as to reduce the number of learnable parameters. To ensure that ω_L and ω_H are positive and comparable, we normalize them by the softmax function, i.e., $\omega_L = \frac{\exp(\omega_L)}{\exp(\omega_L)+\exp(\omega_H)}$, $\omega_H = \frac{\exp(\omega_H)}{\exp(\omega_L)+\exp(\omega_H)}$. In addition to the attention weights, another important issue is how to choose the low- and high-frequency components $\mathbf{B}_L$ and $\mathbf{B}_H$. A natural choice is to use the graph Fourier bases, yet the literatures [16, 35] conclude that utilizing the spectral graph wavelet operators can achieve better embedding results than the graph Fourier bases. Therefore, we substitute $\mathbf{B}_L$ and $\mathbf{B}_H$ in Equation (16) for the spectral graph wavelet operator $\Psi_{L,g}^s$ and $\Psi_{H,g}^s$, i.e.,

$$\mathbf{X}^{(l+1)} = \rho\left(\textsc{Aggregate}\left(\left(\Psi_{L,g}^s\right)\mathbf{F}_L\left(\Psi_{L,g}^s\right)^{-1}\mathbf{X}_L, \left(\Psi_{H,g}^s\right)\mathbf{F}_H\left(\Psi_{H,g}^s\right)^{-1}\mathbf{X}_H\right)\right).$$

3.4.4 *Attention-guided Walk.*

The two aforementioned kinds of attention mechanisms focus on incorporating task-relevant information from the neighbors of a given vertex into the updated representations of the pivot. Here, we introduce a new attention mechanism, namely attention-guided walk [123], which has a different purpose from the softmax- and similarity-based attention mechanisms. Suppose a walker walks along $v_0, v_1, \ldots, v_N$ and he currently locates at the vertex v_t. The hidden representation $\mathbf{h}_{t,:}$ of v_t is computed by a recurrent neural network $f_x(\cdot)$ taking the step embedding $\mathbf{s}_{t,:}$ and internal representation of the historical information from the previous step $\mathbf{h}_{t-1,:}$ as input, i.e.,

$$\mathbf{h}_{t,:} = f_x\left(\mathbf{s}_{t,:}, \mathbf{h}_{t-1,:}; \Theta_x\right).$$

The step embedding $\mathbf{s}_{t,:}$ is computed by a step network $f_s(\mathbf{r}_{t-1,:}, \mathbf{x}_{t,:}; \Theta_s)$ taking the ranking vector $\mathbf{r}_{t-1,:}$ and the initial feature vector $\mathbf{x}_{t,:}$ of the top-priority vertex v_t as input, i.e.,

$$\mathbf{s}_{t,:} = f_s\left(\mathbf{r}_{t-1,:}, \mathbf{x}_{t,:}; \Theta_s\right).$$

The hidden representation $\mathbf{h}_{t,:}$ is then fed into a ranking network $f_r(\mathbf{h}_{t,:}; \Theta_r)$ and a predicting network $f_p(\mathbf{h}_{t,:}; \Theta_p)$, i.e.,

$$\mathbf{r}_{t,:} = f_r\left(\mathbf{h}_{t,:}; \Theta_r\right), \quad \hat{\mathbf{l}}_{t,:} = f_p(\mathbf{h}_{t,:}; \Theta_p).$$

The ranking network $f_r(\mathbf{h}_{t,:}; \Theta_r)$ determines which neighbors of v_t should be prioritized in the next step, and the predicting network $f_p(\mathbf{h}_{t,:}; \Theta_p)$ makes a prediction on graph labels. Now, $\mathbf{h}_{t,:}$ and $\mathbf{r}_{t,:}$ are fed into the next vertex to compute its hidden representations. Figure 5 shows the computational framework of the attention-guided walk.

3.5 Graph Recurrent Neural Networks

The GRNNs generalize the RNNs to process the graph-structured data. In general, the GRNN can be formulated as

$$\mathbf{h}_{j,:}^{(l+1)} = \mathrm{GRNN}\left(\mathbf{x}_{j,:}^{(l)}, \left\{\mathbf{h}_{k,:}^{(l)} : v_k \in N_G(v_j) \cup \left\{v_j\right\}\right\}\right).$$

Table 5 presents an overview of the main GRNNs.

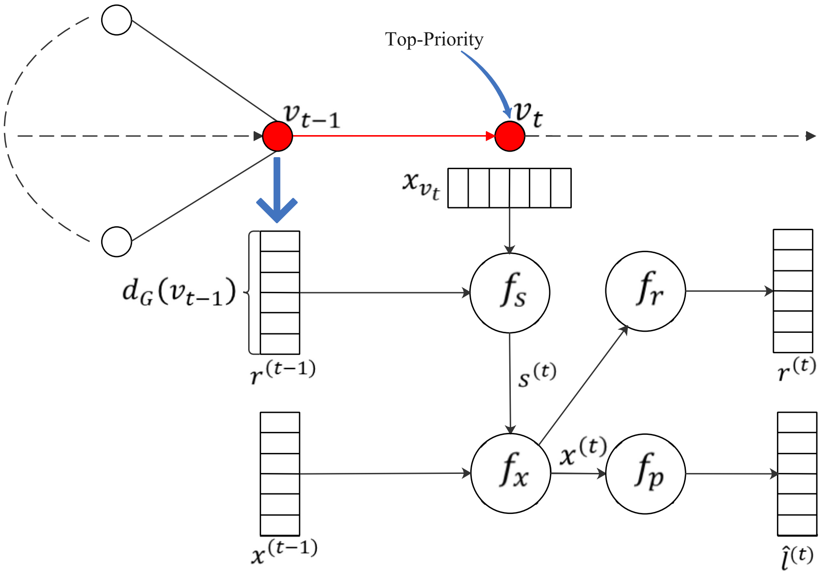

Fig. 5. The computational framework of the attention-guided walk.

Table 5. Overview of MAIN GRNNs

Category	Approach	Task	Architecture	Improvement
graph LSTM	GraphLSTM [296]	graph classification	node embedding, random walk, LSTM	mapping graph nodes to sequences via a parameterized random walk, proposing a new node embedding method, extending RNNs to learn graph-level representation
	GG-NNs [303]	bAbI tasks [98]	node annotation, propagation model, sequence outputs with observed and latent annotations	converting graphs into sequences
dynamic graph	DGNN [279]	link prediction, node classification	update component (interact unit, update unit, merge unit), propagation component	providing a principled approach for the node information update and propagation, modeling establishing orders and time intervals of edges into a coherent framework
vanilla RNN	GraphRNA [260]	node classification	attributed random walks, graph recurrent networks (bidirectional GRU, pooling)	performing joint random walks on attributed networks, boosting the deep node representation learning

3.5.1 Graph LSTM. The **Graph Long Short Term Memroy (Graph LSTM)** [9, 129, 244, 266, 296, 302] generalizes the vanilla LSTM for the sequential data to the ones for general graph-structured data. Specifically, the graph LSTM updates the hidden states and cell states of vertices by the following formula:

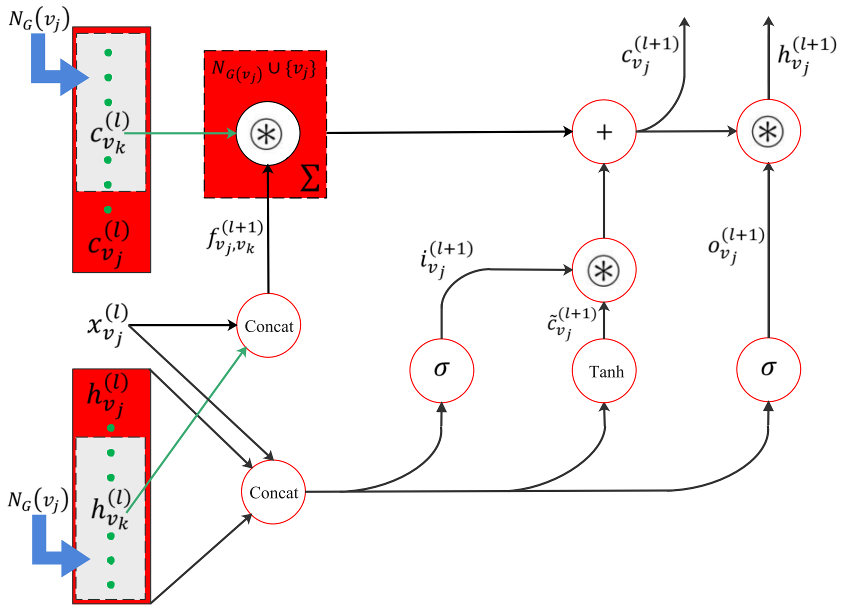

Fig. 6. Computational framework of the Graph LSTM.

$$\mathbf{i}_{j,:}^{(l+1)} = \sigma\left(\mathbf{W}_i\mathbf{x}_{j,:}^{(l)} + \sum_{v_k \in N_G(v_j)\cup\{v_j\}} \mathbf{U}_i\mathbf{h}_{k,:}^{(l)} + \mathbf{b}_i\right),$$
$$\mathbf{o}_{j,:}^{(l+1)} = \sigma\left(\mathbf{W}_o\mathbf{x}_{j,:}^{(l)} + \sum_{v_k \in N_G(v_j)\cup\{v_j\}} \mathbf{U}_o\mathbf{h}_{k,:}^{(l)} + \mathbf{b}_o\right)$$
$$\widetilde{\mathbf{c}}_{j,:}^{(l+1)} = \tanh\left(\mathbf{W}_c\mathbf{x}_{j,:}^{(l)} + \sum_{v_k \in N_G(v_j)\cup\{v_j\}} \mathbf{U}_c\mathbf{h}_{k,:}^{(l)} + \mathbf{b}_c\right),$$
$$\mathbf{f}_{j,k}^{(l+1)} = \sigma\left(\mathbf{W}_f\mathbf{x}_{j,:}^{(l)} + \mathbf{U}_f\mathbf{h}_{k,:}^{(l)} + \mathbf{b}_f\right)$$
$$\mathbf{c}_{j,:}^{(l+1)} = \mathbf{i}_{j,:}^{(l+1)} \circledast \widetilde{\mathbf{c}}_{j,:}^{(l+1)} + \sum_{v_k \in N_G v_j \cup\{v_j\}} \mathbf{f}_{j,k}^{(l+1)} \circledast \mathbf{c}_{k,:}^{(l)},$$
$$\mathbf{h}_{j,:}^{(l+1)} = \mathbf{o}_{j,:}^{(l+1)} \circledast \tanh\left(\mathbf{c}_{j,:}^{(l+1)}\right).$$

See Figure 6. The literature [265] develops a general framework, named structure-evolving LSTM, for learning explainable data representations via the graph LSTM. It progressively evolves the multi-level vertex representations by stochastically merging two adjacent vertices with high compatibilities estimated by the adaptive forget gate of the graph LSTM. As a result, the new graph is produced with a Metropolis-Hastings sampling method. The **Gated Graph Sequence Neural Networks (GGS-NNs)** [303] employ the **Gated Recurrent Unit (GRU)** [143] to modify the vanilla GCNNs so that it can be extended to process the sequential data.

3.5.2 GRNNs for Dynamic Graphs. A dynamic graph is the one whose structure (i.e., adding a vertex, removing a vertex, adding an edge, removing an edge) or features on vertices/edges evolve over time. The GRNNs are a straightforward approach to tackling dynamic graphs. Below, we introduce some studies on the GRNNs for dynamic graphs.

The literature [279] proposes a **Dynamic Graph Neural Network (DGNN)** concerning the dynamic graph only with vertices or edges added. More specifically, the DGNN is composed of two key components: an update component and a propagation component. Suppose an edge (v_s, v_r, t) is added to the input dynamic graph at time t. Let $t-$ denote a time before time t. The update component consists of three sequential units: the interacting unit, the S- or G-update unit, and the merging unit. The interacting unit of an edge (s, r), whose index is k in the edge list, takes the source and target representations before time t as input and outputs the joint representation of the interaction, i.e., $\mathbf{x}_{k,:}^{(e,t)} = \rho(\mathbf{W}_s \mathbf{x}_{s,:}^{(t-)} + \mathbf{W}_r \mathbf{x}_{r,:}^{(t-)} + \mathbf{b}_{k,:})$. The S- and G-update units employ the LSTM [217] to respectively update the cell states and hidden states of the source and target, i.e.,

$$\left(\mathbf{C}_{s,:}^{(t)}, \mathbf{h}_{s,:}^{(t)}\right) = \mathrm{LSTM}_s\left(C_{s,:}^{(t-)}, \mathbf{h}_{s,:}^{(t-)}, \Delta t_s\right), \quad \left(\mathbf{C}_{r,:}^{(t)}, \mathbf{h}_{r,:}^{(t)}\right) = \mathrm{LSTM}_g\left(\mathbf{C}_{r,:}^{(t-)}, \mathbf{h}_{r,:}^{(t-)}, \Delta t_r\right).$$

The merging unit adopts the similar functions to the interacting unit to respectively merge $\mathbf{h}_{s,:}^{(t)}$ and $\mathbf{h}_{s,:}^{t-}$, and $\mathbf{h}_{r,:}^{(t)}$ and $\mathbf{h}_{r,:}^{t-}$. The propagation component can propagate information from two interacting vertices (v_s and v_r) to influence vertices (i.e., their neighbors). It also consists of three units: the interacting unit, the propagation unit, and the merge unit, which are defined similarly to the update component except that they have different learnable parameters. The literature [70] addresses the vertex- and graph-focused prediction tasks on dynamic graphs with a fixed vertex set by combining GCNs, LSTMs, and fully connected layers. The **Variational Graph Recurrent Neural Network (VGRNN)** [62] is essentially a variational graph auto-encoder whose encoder integrates the GCN and RNN into a GRNN framework and the decoder is a joint probability distribution of a multi-variate Gaussian distribution and Bernoulli Distribution. The semi-implicit variational inference is employed to approximate the posterior so as to generate the vertex embedding.

3.5.3 GRNNs Based on Vanilla RNNs. The GRNNs based on vanilla RNNs first employ random walk techniques or traversal methods, e.g., **Breadth-First Search (BFS)** and **Depth-First Search (DFS)**, to obtain a collection of vertex sequences and then leverage an RNN, e.g., LSTM and GRU, to capture long short-term dependencies. The literature [260] performs joint random walks on attributed networks and utilizes them to boost the deep vertex representation learning. The proposed GraphRNA in [260] consists of two key components, namely a collaborative walking mechanism AttriWalk and a tailored deep embedding architecture for joint random walks GRN. Suppose $\mathbf{X}$ denotes the vertex feature matrix of size $N \times d$. The AtriWalk admits the transition matrix of size $\mathbb{R}_{+}^{(N+d)\times(N+d)}$, which is written as

$$\mathcal{T} = \begin{bmatrix} \alpha \mathbf{A}_G & (1-\alpha)\mathbf{X} \\ (1-\alpha)\mathbf{X}^T & \mathbf{0}_{d\times d} \end{bmatrix}.$$

After obtaining the sequences via the collaborative random walk, the bi-directional GRU [143] and pooling operator are employed to learn the global representations of sequences. The literature [61] leverages the BFS vertex ordering and truncation strategy to obtain a collection of vertex representation sequences and then uses the GRU model and variational auto-regression regularization to perform the graph classification.

4 EXTENDED ARCHITECTURES AND APPLICATIONS

The aforementioned architectures essentially provide ingredients of constructing the GNNs for us. This section tells readers how these building blocks are organized. Below, we investigate the extensions of the GNNs from the next 10 aspects: GCNNs on special graphs, capability and explainability, graph pre-training models, GNN-based network embedding, deep graph generative models, combining PI and GNNs, adversarial training of GNNs, graph neural architecture search, graph reinforcement learning, and applications of GNNs.

4.1 GCNNs on Special Graphs

The vanilla GCNNs aim at learning the representations of input graphs (directed or undirected, weighted or unweighted). The real-world graphs may have more additional characteristics, e.g., spatial-temporal graphs, heterogeneous graphs, hyper-graphs, signed graphs, and so on. The GCNN for signed graphs [242] leverages the balance theory to aggregate and propagate information through positive and negative links.

4.1.1 Heterogeneous Graphs. Heterogeneous Graphs are composed of vertices and edges of different types, and each type of edge is called a relation between two types of vertices. For example, a bibliographic information network contains at least four types of vertices, namely Author, Paper, Venue, and Term, and at least three types of edges, namely Author-Paper, Term-Paper, and Venue-Paper [298]. The heterogeneity and rich semantic information brings great challenges for designing heterogeneous graph convolutional neural networks. In general, a heterogeneous graph can be denoted as $H = (V, E, \nu, \zeta)$, where $\nu(v)$ denotes the type of vertex $v \in V$ and $\zeta(e)$ denotes the type of edge $e \in E$. Let $\mathcal{T}^v$ and $\mathcal{T}^e$ respectively denote the set of vertex types and edge types. The literature [22] provides a unified framework to deeply summarize and evaluate existing work on heterogeneous network representation learning. The readers can refer to this article for deeper information about the heterogeneous graph learning.

Vanilla Heterogeneous GCNNs. The **Heterogeneous Graph Neural Networks (HetGNNs)** [34] aims to resolve the issue of jointly considering heterogeneous structural information as well as heterogeneous content information of vertices. It first samples a fixed size of strongly correlated heterogeneous neighbors for each vertex via a **Random Walk with Restart (RWR)** and groups them into different vertex types. Then, it aggregates feature information of those sampled neighboring vertices via a bi-directional LSTM and attention mechanism. Running RWR with a restart probability p from vertex v will yield a collection of a fixed number of vertices, denoted as RWR(v). For each vertex type t, the t-type neighbors $N_G^t(v)$ of vertex v denotes the set of top-k_t vertices from RWR(v) with regard to frequency. Let C_v denote the heterogeneous contents of vertex v, which can be encoded as a fixed-size embedding via a function $f_1(v)$, i.e.,

$$\mathbf{f}_{v,:}^{(1)} = \frac{\sum_{j \in C_v} \left[\overrightarrow{\text{LSTM}} \left(\mathcal{FC}_{\theta_x}(\mathbf{x}_{v,:}^{(j)}) \right) \bowtie \overleftarrow{\text{LSTM}} \left(\mathcal{FC}_{\theta_x}(\mathbf{x}_{v,:}^{(j)}) \right) \right]}{|C_v|},$$

where $\mathcal{FC}_{\theta_x}(\cdot)$ denotes feature transformer, e.g., identity or fully connected neural networks with parameter θ_x, and $\overrightarrow{\text{LSTM}}(\cdot)$ and $\overleftarrow{\text{LSTM}}(.)$ is defined by the Equation (12). The content embedding of the t-type neighbors of vertex v can be aggregated as follows:

$$\begin{aligned} \mathbf{f}_{v,:}^{(2,t)} &= \textsc{Aggregate}^T \left(\left\{ \mathbf{f}_{u,:}^{(1)} : u \in N_G^t(v) \right\} \right) \\ &= \frac{\sum_{u \in N_G^t(v)} \left[\overrightarrow{\text{LSTM}}(\mathbf{f}_1(u)) \bowtie \overleftarrow{\text{LSTM}}(\mathbf{f}_1(u)) \right]}{|N_G^t(v)|}. \end{aligned}$$

Let $F_v = \{f_{v,:}^{(1)}\} \cup \{f_{v,:}^{(2,t)}, t \in \mathcal{T}^v\}$. As a result, the output embedding of vertex v can be obtained via the attention mechanism, i.e., $\mathbf{o}_v = \omega_v^{(1)} \mathbf{f}_{v,:}^{(1)} + \sum_{t \in \mathcal{T}^v} \omega_v^{(2,t)} \mathbf{f}_{v,:}^{(2,t)}$, where $\omega_v^{(1)}$ and $\omega_v^{(2,t)}$ denote the concatenation-based graph attention weights; see Equation (13). In addition, the GraphInception [29] can be employed to learn the hierarchical relational features on heterogeneous graphs by converting the input graph into a multi-channel graph (each meta path as a channel) [290].

Heterogeneous Graph Attention Mechanism. The literature [263] first proposes a hierarchical attention-based heterogeneous GCNN consisting of vertex-level and semantic-level attentions. The vertex-level attention aims to learn the attention weights of a vertex and its meta-path-based neighbors, and the semantic-level attention aims to learn the importance of different meta-paths. More specifically, given a meta-path Φ, the vertex-level attention weight of a vertex v_i and its meta-path-based neighbors $v_j \in N_G^{\Phi}(v_i)$ is defined to be

$$\omega_{j,k}^{\Phi} = \frac{\exp\left(\rho\left(\mathbf{a}_{\Phi}^{T}(\mathbf{M}_{\nu(v_j)}\mathbf{x}_{j,:} \bowtie \mathbf{M}_{\nu(v_k)}\mathbf{x}_{k,:})\right)\right)}{\sum_{v_k \in N_G^{\Phi}(v_j)} \exp\left(\rho\left(\mathbf{a}_{\Phi}^{T}(\mathbf{M}_{\nu(v_j)}\mathbf{x}[j,:] \bowtie \mathbf{M}_{\nu(v_k)}\mathbf{x}_{k,:})\right)\right)},$$

where $\mathbf{M}_{\nu(v_j)}$ transforms the feature vectors of vertices of type $\nu(v_j)$ in different vector spaces into a unified vector space. The embedding of vertex v_i under the meta-path Φ can be computed by

$$\mathbf{x}_{j,:}^{\Phi,l+1} = \bowtie_{k=1}^{K} \rho\left(\sum_{k \in N_G^{\Phi}(v_j)} \omega_{j,k}^{\Phi}\mathbf{M}_{\nu(v_k)}\mathbf{x}_{k,:}^{\Phi,l}\right).$$

Given a meta-path set $\{\Phi_0, \ldots, \Phi_P\}$, performing the vertex-level attention layers under each meta-path will yield a set of semantic-specific vertex representations, namely $\{\mathbf{X}^{\Phi_0}, \ldots, \mathbf{X}^{\Phi_P}\}$. The semantic-level attention weight of the meta-path Φ_j is defined as

$$\beta^{\Phi_j} = \frac{\exp\left(\omega^{\Phi_j}\right)}{\sum_{p=1}^{P} \exp\left(\omega^{\Phi_p}\right)},$$

where $\omega^{\Phi_p} = \frac{1}{|V|}\sum_{v_k \in V} \mathbf{q}^T \tanh(\mathbf{W}\mathbf{x}_k^{\Phi_p} + \mathbf{b})$. As a result, the embedding matrix $\mathbf{X} = \sum_{p=1}^{P} \beta^{\Phi_p}\mathbf{X}^{\Phi_p}$. In addition, there are some available studies on the GCNNs for multi-relational graphs [41, 277] and the transformer for dynamic heterogeneous graphs [306, 323]. The literature [21] leveraged a contextual masking operation at the feature level and a contextual attention mechanism at the node level to model the multipartite networks of target nodes and their intermediate context nodes and achieve interaction contextualization by treating neighboring target nodes based on intermediate context nodes.

4.1.2 Spatio-Temporal Graphs. Spatio-temporal graphs can be used to model traffic networks [15, 221] and skeleton networks [226, 305]. In general, a spatio-temporal graph is denoted as $G_{ST} = (V_{ST}, E_{ST})$, where $V_{ST} = \{v_{t,j} : t = 1, \ldots, T, j = 1, \ldots, N_{ST}\}$. The edge set E_{ST} is composed of two types of edges, namely spatial edges and temporal edges. All spatial edges $(v_{t,j}, v_{t,k}) \in E_{ST}$ are collected in the intra-frame edge set E_S, and all temporal edges $(v_{t,j}, v_{t+1,j}) \in E_{ST}$ are collected in the inter-frame edge set E_T. The literature [15] proposes a novel deep learning framework, namely **Spatio-Temporal Graph Convolutional Networks (STGCN)**, to tackle the traffic forecasting problem. Specifically, STGCN consists of several layers of spatio-temporal convolutional blocks, each of which has a "sandwich" structure with two temporal gated convolution layers (abbreviated as Temporal Gated-Conv) and a spatial graph convolution layer in between (abbreviated as Spatial Graph-Conv). The Spatial Graph-Conv exploits the conventional GCNNs to extract the spatial features, whereas the Temporal Gated-Conv exploits the temporal gated convolution operator to extract temporal features. Suppose that the input of the temporal gated convolution for each vertex is a length-S sequence with C_{in} channels, i.e., $\mathbf{X} \in \mathbb{R}^{S \times C_{\text{in}}}$. The temporal gated convolution kernel $\Gamma \in \mathbb{R}^{K \times C_{\text{in}} \times 2C_{\text{out}}}$ is used to filter the input $\mathbf{Y}$, i.e.,

$$\Gamma *_T \mathbf{Y} = P \circledast \sigma(Q) \in \mathbb{R}^{(S-K+1) \times C_{\text{out}}},$$

to yield an output $\mathbf{P} \bowtie \mathbf{Q} \in \mathbb{R}^{(M-K+1)\times(2C_{\text{out}})}$. The Spatial Graph-Conv takes a tensor $\underline{\mathbf{X}}^{(l)} \in \mathbb{R}^{S\times N_{ST}\times C^{(l)}}$ as input and outputs a tensor $\underline{\mathbf{X}}^{(l+1)} \in \mathbb{R}^{(S-2(K-1))\times N_{ST}\times C^{(l+1)}}$, i.e.,

$$\underline{\mathbf{X}}^{(l+1)} = \Gamma_1^{(l)} *_T \rho\left(\Theta^{(l)} *_G \left(\Gamma_0^{(l)} *_T \underline{\mathbf{X}}^{(l)}\right)\right),$$

where $\Gamma_0^{(l)}, \Gamma_1^{(l)}$ are the upper and lower temporal kernel and $\Theta^{(l)}$ is the spectral kernel of the graph convolution. In addition, some other studies pay attention to the GCNNs on the spatio-temporal graphs from other perspectives, e.g., Structural-RNN [11] via a factor graph representation of the spatio-temporal graph and GCRNN [153, 293] combining the vanilla GCNN and RNN.

4.1.3 Hypergraphs. The aforementioned GCNN architectures are concerned with the conventional graphs consisting of pairwise connectivity between two vertices. However, there could be even more complicated connections between vertices beyond the pairwise connectivity, e.g., co-authorship networks. Under such circumstances, a hypergraph, as a generalization to the convectional graph, provides a flexible and elegant modeling tool to represent these complicated connections between vertices. A hypergraph is usually denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \omega)$, where $\mathcal{V} = \{v_1, \ldots, v_N\}$ like the conventional graph, and $\mathcal{E} = \{e_1, \ldots, e_M\}$ is a set of M hyperedges. $\omega(e_k)$ denote weights of hyperedges $e_k \in \mathcal{E}$. A non-trivial hyperedge is a subset of $\mathcal{V}$ with at least two vertices. In particular, a trivial hyperedge, called a self-loop, is composed of a single vertex. The hypergraph $\mathcal{G}$ can also be denoted by an incidence matrix $\mathbf{H}_{\mathcal{G}} \in \mathbb{R}^{N\times M}$, i.e.,

$$\mathbf{H}_{\mathcal{G}}[j,k] = \begin{cases} 0, & v_j \notin e_k \\ 1, & v_j \in e_k. \end{cases}$$

For a vertex $v_j \in \mathcal{V}$, its degree $d_{\mathcal{G}}^{(v)}(v_j) = \sum_{e_k\in\mathcal{E}} \omega(e_k)\mathbf{H}_{\mathcal{G}}[j,k]$. For a hyperedge $e_k \in \mathcal{E}$, its degree $\mathrm{d}_{\mathcal{G}}^{(e)}(e_k) = \sum_{v_j\in e_k} \mathbf{H}_{\mathcal{G}}[j,k]$. Let

$$\mathbf{D}_{\mathcal{V}} = \operatorname{diag}\left(d_{\mathcal{G}}^{(v)}(v_1), \ldots, d_{\mathcal{G}}^{(v)}(v_N)\right), \quad \mathbf{D}_{\mathcal{E}} = \operatorname{diag}\left(d_{\mathcal{G}}^{(e)}(e_1), \ldots, d_{\mathcal{G}}^{(e)}(e_M)\right),$$

and $\mathbf{W}_{\mathcal{G}} = \operatorname{diag}(\omega(e_1), \ldots, \omega(e_M))$. The hypergraph Laplacian [284] $\mathbf{L}_{\mathcal{G}}$ of $\mathcal{G}$ is defined to be

$$\mathbf{L}_{\mathcal{G}} = \mathbf{I}_N - \mathbf{D}_{\mathcal{V}}^{-\frac{1}{2}}\mathbf{H}_{\mathcal{G}}\mathbf{W}_{\mathcal{G}}\mathbf{D}_{\mathcal{E}}^{-1}\mathbf{H}_{\mathcal{G}}^T\mathbf{D}_{\mathcal{V}}^{-\frac{1}{2}}.$$

It can also be factorized by the eigendecomposition, i.e., $\mathbf{L}_{\mathcal{G}} = \mathbf{U}\Lambda\mathbf{U}^T$. The spectral hypergraph convolution operator, the Chebyshev hypergraph convolutional neural network, and the hypergraph convolutional network can be defined in analogy to Equations (4), (6), and (8). The **HyperGraph Neural Network (HGNN)** architecture proposed in the literature [284] is composed of multiple layers of the hyperedge convolution, and each layer is defined as

$$\mathbf{X}^{(l+1)} = \rho\left(\mathbf{D}_{\mathcal{V}}^{-\frac{1}{2}}\mathbf{H}_{\mathcal{G}}\mathbf{W}_{\mathcal{G}}\mathbf{D}_{\mathcal{E}}^{-1}\mathbf{H}_{\mathcal{G}}^T\mathbf{D}_{\mathcal{V}}^{-\frac{1}{2}}\mathbf{X}^{(l)}\Theta^{(l)}\right).$$

In essence, the HGNN essentially views each hyperedge as a complete graph so that the hypergraph is converted into a conventional graph. Treating each hyperedge as a complete graph obviously incurs expensive computational cost. Hence, some studies [174, 175] propose various approaches to approximate the hyperedges. The HNHN [285] interleaves updating the vertex representations with the hyperedge representations by the following formulas:

$$\mathbf{X}_{\mathcal{V}}^{(l+1)} = \rho\left(\mathbf{D}_{\mathcal{V}}^{-1}\mathbf{H}_{\mathcal{G}}\mathbf{X}_{\mathcal{E}}^{(l)}\Theta_{\mathcal{V}}^{(l)}\right), \quad \mathbf{X}_{\mathcal{E}}^{(l+1)} = \rho\left(\mathbf{D}_{\mathcal{E}}^{-1}\mathbf{H}_{\mathcal{G}}^T\mathbf{X}_{\mathcal{V}}^{(l)}\Theta_{\mathcal{E}}^{(l)}\right).$$

4.2 Capability and Explainability

The GCNNs have achieved tremendous empirical successes over the supervised, semi-supervised, and unsupervised learning on graphs. Recently, many studies have started to examine the capability and explainability of the GCNNs.

4.2.1 Capability. The capability of the GCNNs refers to their expressive power. If two graphs are isomorphic, they will obviously output the same representations of vertices/edges/graph. Otherwise, they should output different representations. However, two non-isomorphic graphs may output the same representations in practice. This is the theoretical limitations of the GCNNs. As described in the literatures [30, 138, 210], the 1-**hop spatial GCNNs (1-GCNNs)** have the same expressive power as the 1-**dimensional Weisfeiler-Leman (1-WL)** graph isomorphism test in terms of distinguishing non-isomorphic graphs. The 1-WL iteratively updates the colors of vertices according to the following formula:

$$C^{(l+1)}(v) = \textsc{Hash}\left(C^{(l)}(v), \left\{C^{(l)}(u) : u \in N_G(v)\right\}\right).$$

According to the literatures [30, 138], we have that the 1-GCNN architectures do not have more power in terms of distinguishing two non-isomorphic graphs than the 1-WL heuristic. Nevertheless, they have equivalent power if the aggregation and update functions are injective. In order to overcome the theoretical limitations of the GCNNs, the literature [138] proposes a **Graph Isomorphism Network (GIN)** architecture, i.e.,

$$\begin{aligned}
\mathbf{a}_{v,:}^{(l+1)} &= \psi^{e\to v}\left(\left\{\mathbf{X}^{(l)}[u,:] : u \in N_G(v)\right\}\right) \\
&\triangleq \sum_{u\in N_G(v)} \mathbf{X}^{(l)}[u,:] \\
\mathbf{X}^{(l+1)}[v,:] &= \phi^{v}\left(\mathbf{X}^{(l)}[v,:], \mathbf{a}_{v,:}^{(l+1)}\right) \\
&\triangleq \mathrm{MLP}\left((1+\epsilon^{(l+1)})\mathbf{X}^{(l)}[v,:] + \mathbf{a}_{v,:}^{(l+1)}\right),
\end{aligned}$$

where $\epsilon^{(l)}$ is a scalar parameter. The literature [8] studies the expressive power of the spatial GCNNs and presents two results: (1) the spatial GCNNs are shown to be a universal approximator under sufficient conditions on their depth, width, initial vertex features, and layer expressiveness, and (2) the power of the spatial GCNNs is limited when their depth and width are restricted. In addition, there are some other studies on the capability of the GCNNs from different perspectives, e.g., the first-order logic [186], p-order graph moments [181], algorithmic alignment with the dynamic programming [137], and generalization and representational limits of the GNNs [76].

4.2.2 Explainability. Explainability plays a vital role in constructing a reliable and intelligent learning system. Although some studies have started to explore the explainability of the conventional deep learning models, few studies have examined the explainability of the GNNs [64]. The literature [196] extended three prominent explainability methods: (1) contrastive gradient-based saliency maps, (2) class activation mapping, and (3) excitation back-propagation, to explain graph convolutional neural networks. The GNNExplainer [317] is a general and model-agnostic approach for providing explainable explanations for any spatial GCNN-based model in terms of graph machine learning tasks. Given a trained spatial GCNN model Φ and a set of predictions, the GNNExplainer will generate a single-instance explanation by identifying a subgraph of the computation graph and a subset of initial vertex features, which are the most vital for the prediction of the model Φ. In general, the GNNExplainer can be formulated as an optimization problem

$$\max_{G_S, \mathbf{X}_{S,:}^F} I\left(\mathbf{Y}; (G_S, \mathbf{X}_{S,:}^F)\right) = H(\mathbf{Y}) - H\left(\mathbf{Y} | G = G_S, \mathbf{X} = \mathbf{X}_{S,:}^F\right), \tag{17}$$

where $I(\cdot;\cdot)$ denotes the mutual information of two random variables, G_S is a small subgraph of the computation graph, and $\mathbf{X}^F_{S,:}$ is a small subset of vertex features $\{\mathbf{X}^F[j,:] : v_j \in G_S\}$. The entropy term $H(\mathbf{Y})$ is constant because the spatial GCNN model Φ is fixed. In order to improve the tractability and computational efficiency of the GNNExplainer, the final optimization framework is reformulated as

$$\min_{\mathbf{M},\mathbf{F}} -\sum_{c=1}^{C} \mathbb{I}\left(y=c\right) \log P_\Phi\left(\mathbf{Y}=y|G=\mathbf{A}_G \circledast \sigma(\mathbf{M}), \mathbf{X}=\mathbf{X}^F_{S,:}\right).$$

In addition, the GNNExplainer also provides multi-instance explanations based on graph alignments and prototypes so as to answer questions like "How did a GCNN predict that a given set of vertices all have label c?" The literature [83] reviewed current GNN explainability methods and provided a unified categorization of taxonomy for this topic. The purpose of this article is to shed light on the commonalities and differences of existing methods and set the stage for further methodological developments. The literature [161] explained and unified the GNNs with an optimization framework. They formulated different propagation mechanisms as a unified optimization objective,

$$\min_{\mathbf{Z}} \zeta \|\mathbf{F}_1\mathbf{Z} - \mathbf{F}_2\mathbf{Z}\|_F^2 + \xi \operatorname{tr}\left(\mathbf{Z}^T\overline{\mathbf{L}}\mathbf{Z}\right),$$

where ξ is a non-negative coefficient, ζ is usually chosen from $[0, 10]$, and Z is the transformation on original input feature matrix $\mathbf{X}$. $\mathbf{F}_1$ and $\mathbf{F}_2$ are defined as arbitrary graph convolutional kernels. The conventional GNN models, e.g., GCN [237], PPNP/APPNP [121], JKNet [136], and DAGNN [162], have been explained by the proposed optimization framework.

4.3 Graph Pre-training Frameworks

Pre-training plus fine-tuning have become a prevalent learning paradigm in the field of natural language processing [273]. Recently, pre-training GNNs to learn transferable knowledge for downstream tasks has also achieved remarkable success in the GNN community. In general, there are three kinds of pre-training methods for GNNs [280], i.e., predictive self-supervised pre-training, contrastive self-supervised pre-training, and self-attention based pre-training. The first two schemes belong to the field of self-supervised graph learning, which is an impressive learning paradigm and has become a hot and promising pre-training method for graphs.

Before proceeding, we first introduce three types of fine-tuning schemes [289], i.e., parameter-transferring fine-tuning, joint learning, and parameter-freezing fune-tuning. For the **parameter-transferring fine-tuning scheme**, the encoder is first pre-trained on the pretext tasks and then transferred to downstream tasks. For the **joint learning scheme**, the encoder is jointly trained on both the pretext and downstream tasks. Their contributions are balanced by a trade-off hyperparameter. For the parameter-freezing fine-tuning scheme, the encoder is first trained on the pretext tasks in an unsupervised manner and then frozen and incorporated into the downstream training models.

4.3.1 Predictive Self-supervised Pre-training. Predictive self-supervised pre-training frameworks train a graph encoder f together with a pretext head g under the supervision of informative labels yielded by some topological characteristics on graphs. The literature [180] first introduced the pre-training techniques on graph data. It defined a Siamese network induced by GNNs to approximate the kernel values of two graphs, which are computed by a graph kernel [132]. For a graph dataset $\mathcal{D}_G = \{G^{(i)} : i = 1, \ldots, D\}$ and a graph kernel $K(\cdot,\cdot)$, the Siamese network was trained on the dataset $\{(G^{(i)}, G^{(j)}, K(G^{(i)}, G^{(j)})) : i, j \in 1, \ldots, D\}$, where $K(\cdot,\cdot)$ is a graph kernel. The literatures [276, 320, 321] exploited one or more existing tasks on graphs, e.g., graph reconstruction, centrality score ranking, cluster preserving, or denoising graph reconstruction, to pre-train GNNs preserving generic structural information. It is worth noting that [320] took four generic node-level features: degree, core number, collective influence, and local clustering coefficient as input yet the others did not. The literature [144] proposed a general class of structure-related features called distance encoding and applied it to pre-training GNNs in two ways: (1) as extra node features and (2) as controllers of message aggregation. The **Iterative Graph Self-Distillation (IGSD)** [82] iteratively performs the teacher-student distillation with graph augmentations and predicts the teacher network representation of the graph pairs under different augmented views. The graph augmentations transform a graph with the transition matrix $\mathbf{D}^{-1}\mathbf{A}$ via graph diffusion and sparsification. The L2P-GNN [300] attempts to narrow the gap between the pre-training and fine-tuning. It encodes the local and global information into the prior via a dual adaptation mechanism at both node and graph levels. The **Generative Pre-Training framework for GNNs (GPT-GNN)** [322] aims to pre-train GNNs by maximizing the graph likelihood of the graph distribution $p_\theta(\mathbf{X}^\pi, E^\pi)$ over all possible permutations, i.e.,

$$p\,(G;\theta) = E_\pi\ \left[p_\theta\,(\mathbf{X}^\pi, E^\pi)\right],$$

where E (and E^π) is a set of edges (under a permutation π).

4.3.2 Contrastive Self-supervised Pre-training. Contrastive learning has made remarkable progress in natural language processing and computer vision. Recent studies are inspired by this trend and would like to apply this self-supervised learning paradigm to training graph data. In general, contrastive learning frameworks include three components [280], i.e., transformations, encoders, and learning objectives. The transformations aim at computing multiple views of input graphs, the encoders aim at embedding each view into a low-dimensional space, and the learning objectives are optimized in order to optimize model parameters in the encoders. According to the multiple graph views, the contrastive self-supervised pre-training frameworks can be categorized into five classes: node-original graph contrast, node-structurally transformed graph contrast, node-subgraph contrast, contrast on subgraphs, and contrast on randomly transformed graphs.

Node-original graph contrast. This kind of contrastive learning formulates the learning objective by contrasting node-level representations with graph-level ones. A canonical learning objective is the mutual information between patch representations and corresponding high-level summaries of graphs. The deep graph infomax [192] adopts this contrastive learning scheme to pre-train GNNs in a self-supervised manner. Specifically, its objective is

$$\mathcal{L} = \frac{1}{N+M}\left(\sum_{i=1}^{N}\mathbb{E}_{(\mathbf{X}_V,\mathbf{A})}\left[\log I\left(\mathbf{h}_{i,:}^{(L)};\mathbf{s}\right)\right] + \sum_{j=1}^{M}\mathbb{E}_{(\widetilde{\mathbf{X}}_V,\widetilde{\mathbf{A}})}\left[\log\left(1 - I\left(\widetilde{\mathbf{h}}_{j,:}^{(L)};\widetilde{\mathbf{s}}_{j,:}\right)\right)\right]\right),$$

where the term on the right side of "+" is yielded by negative samples, $\mathbf{s}$ (and $\widetilde{\mathbf{s}}$) is a summary of the patch representations $\mathbf{h}_{i,:}, i = 1, \ldots, N$ (and $\widetilde{\mathbf{h}}_{j,:}, j = 1, \ldots, M$) via a readout function, and $\mathbf{h}_{i,:}$ (and $\widetilde{\mathbf{h}}_{j,:}$) is computed by a GNN. The literature [311] proposed a novel concept, called **Graphical Mutual Information (GMI)**, to measure the correlation between input graphs and high-level representations. Inspired by the deep graph infomax, there are some studies [37, 63] to maximize the mutual information between the graph-level representation and the representations of substructures of different scales, e.g., nodes, edges, triangles, and clusters. The **Contrastive GCNs with Graph Generation (CG^3)** obtain node representations generated from global and local views respectively and then use the semi-supervised contrastive loss to maximize the agreement between the representations learned from these views. Because the graph topology itself contains valuable information, it leverages a generative term to contrast the correlation between node- and graph-level representations.

Node-structurally transformed graph contrast. This kind of contrastive learning formulates the learning objective by contrasting views between nodes and structurally transformed graphs. The literature [86] employed a graph diffusion to generate an additional structural view of the input graph and then sub-sampled the generated and regular views. The resulting samples are fed into two dedicated GNNs, one for each view, followed by a shared MLP, to learn node representations for both views. The resulting representations are fed into a shared graph pooling layer and a shared MLP. At last, a discriminator is trained by contrasting node representations from one view with graph representations from another view and vice versa.

Node-subgraph contrast. This kind of contrastive learning formulates the learning objective by contrasting views between nodes and subgraphs. The literature [251] develops a new strategy and self-supervised methods for pre-training GNNs at the level of individual nodes as well as context subgraphs. The node-level pre-training includes the context prediction and attribute masking. The former is to pre-train GNNs so that nodes with similar structural contexts are mapped to nearby embeddings, and the latter aims to capture domain knowledge by learning the regularities of the node/edge attributes distributed over graphs. The graph-level one aims to transfer robust graph representations to downstream tasks. The Subg-Con [107] utilizes the strong correlation between central vertices and their sampled subgraphs to capture regional structural information. It does not take the complete input graph as input but the sampled subgraphs as the input of the encoder. Then, it optimizes the correlation of the center vertex representations and the corresponding subgraph representations.

Contrast on subgraphs. This kind of contrastive learning formulates the learning objective by contrasting different subgraph views. The **graph contrastive coding (GCC)** [118] exploits the contrastive learning at the level of subgraphs to learn transferable and informative node-level representations. Its goal is to train a discriminator to map similar subgraphs into nearby embeddings and dissimilar subgraphs into distant embeddings. Specifically, its objective is

$$\mathcal{L} = -\log \frac{\exp\left(\mathbf{x}_q^T \mathbf{x}_k^+ / \tau\right)}{\sum_{k=0}^{K} \exp\left(\mathbf{x}_q^T \mathbf{x}_k^{i-} / \tau\right)},$$

where $\mathbf{x}_q$ is a query representation of the target subgraph, $\mathbf{x}_k^+$ is a key representation of the positive subgraph, and $\mathbf{x}_k^{i-}$ are key representations of the negative subgraphs. All of $\mathbf{x}_q$, $\mathbf{x}_k^+$, and $\mathbf{x}_k^{i-}$ are approximated by GNNs. The literature [222] employs GNNs to extract motifs from large-scale graph datasets and then leverage the learned motifs to sample informative subgraphs for contrastive learning of graphs. The motif learning is formulated as a differentiable clustering problem and adopts EM-clustering to classify similar and frequent subgraphs into several motifs.

Contrast on randomly transformed graphs. This kind of contrastive learning formulates the learning objective by contrasting different views yielded by randomly transformed graphs. The GraphCL [292] designs four types of graph augmentation strategies, i.e., node dropping, edge perturbation, attribute masking, subgraph sampling, and then maximizes the agreement between two augmented views of the same graph via a contrastive loss. Specifically, it is composed of four major components: graph data augmentation; GNN-based encoder; projection head, which maps augmented representations into another latent space; and contrastive loss. The literature [318] adopts adaptive graph augmentation strategies (i.e., topology-level augmentation and node-attribute-level augmentation), which are different from the uniform augmentation operations in [292], to incorporate various priors for topological and semantic aspects of the graph.

4.3.3 Self-attention-based Pre-training. The GraphBert [108], just like the BERT [54], solely employs self-attention mechanisms, instead of the graph aggregation and update operations, to construct a GNN, and can be trained with sampled linkless subgraphs within their local contexts. A core module in the Graph-Bert is its graph transformer, i.e.,

$$\begin{aligned}\mathbf{X}_V^{(l+1)} &= \text{G-Transformer}\left(\mathbf{X}_V^{(l)}\right)\\ &= \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_h}}\right)\mathbf{V} + \text{G-Res}\left(\mathbf{X}_V^{(l-1)}, \mathbf{X}_V\right),\end{aligned}$$

where $\mathbf{Q} = \mathbf{X}_V^{(l)}\mathbf{W}_Q^{(l+1)}$, $\mathbf{K} = \mathbf{X}_V^{(l)}\mathbf{W}_K^{(l+1)}$, and $\mathbf{V} = \mathbf{X}_V^{(l)}\mathbf{W}_V^{(l+1)}$, and G-Res$(\cdot,\cdot)$ represents the graph residual term.

4.4 GNN-based Network Embedding

Network embedding (also called graph representation learning) is a paradigm of unsupervised learning on graphs. It gains a large amount of popularity since the DeepWalk [19]. Subsequently, many studies exploit deep learning techniques to learn low-dimensional representations of vertices [256]. In general, the network embedding via the vanilla deep learning techniques learn low-dimensional feature vectors of vertices by utilizing either stacked auto-encoders to reconstruct the adjacent or positive point-wise mutual information features [39, 56, 134, 219, 262] or RNNs to capture long- and short-term dependencies of vertex sequences yielded by random walks [135, 254]. In the following, we introduce the network embedding approaches based on GNNs.

In essence, the GNNs provide an elegant and powerful framework for learning vertex/edge/graph representations. The majority of the GCNNs and GRNNs are concerned with semi-supervised learning (i.e., vertex-focused tasks) or supervised learning (i.e., graph-focused) tasks. Here, we review the GNN-based unsupervised learning on graphs. In general, the network embedding based on GNNs first utilizes the GCNNs and variational auto-encoders to generate Gaussian-distributed hidden states of vertices and then reconstructs the adjacency matrix and/or the feature matrix of the input graph [32, 216, 223, 236, 312, 314]. A representative approach among these ones is the **Variational Graph Auto-Encoder (VGAE)** [236] consisting of a GCN-based encoder and an inner product decoder. The GCN-based encoder is defined to be

$$q(\mathbf{Z}|\mathbf{X}, \mathbf{A}_G) = \prod_{j=1}^{N} q(\mathbf{Z}[j,:]|\mathbf{X}, \mathbf{A}_G) = \prod_{j=1}^{N} \mathcal{N}\left(\mathbf{Z}[j,:]|\mu_j, \text{diag}(\sigma_j^2)\right),$$

where $\mu_j = \text{GCN}_\mu(\mathbf{X}, \mathbf{A}_G)$ and $\log\sigma_j = \text{GCN}_\sigma(\mathbf{X}, \mathbf{A}_G)$. The inner product decoder is defined to be

$$p(\mathbf{A}_G|\mathbf{Z}) = \prod_{j=1}^{N}\prod_{k=1}^{N} p\left(\mathbf{A}_G[j,k]|\mathbf{Z}[j,:], \mathbf{Z}[k,:]\right) = \prod_{j=1}^{N}\prod_{k=1}^{N} \sigma\left(\mathbf{Z}[j,:]\mathbf{Z}[k,:]^T\right).$$

They adopt the evidence lower bound [51] as their objective function. The adversarially regularized (variational) graph auto-encoder [223] extends the VGAE by adding an adversarial regularization term to the evidence lower bound. The literature [119] proposes a symmetric graph convolutional auto-encoder, which produces low-dimensional latent vertices representations. Its encoder employs Laplacian smoothing [197] to jointly encode the structural and attributed information, and its decoder is designed based on Laplacian sharpening as the counterpart of the Laplacian smoothing of the encoder. The Laplacian sharpening is defined to be $\mathbf{X}^{(l+1)} = (1+\gamma)\mathbf{X}^{(l)} - \gamma\mathbf{D}^{-1}\mathbf{A}\mathbf{X}^{(l)} = \mathbf{X}^{(l)} + \gamma(\mathbf{I}_N - \mathbf{D}^{-1}\mathbf{A})\mathbf{X}^{(l)}$, which allows utilizing the graph structure in the whole process of the proposed auto-encoder architecture. In addition, there are some other methods to perform the unsupervised learning on graphs, which do not rely on the reconstruction of the adjacency and/or feature matrix, e.g., the graph auto-encoder on directed acyclic graphs [310].

4.5 Deep Graph Generative Models

The aforementioned work concentrates on embedding an input graph into a low-dimensional vector space so as to perform semi-supervised/supervised/unsupervised learning tasks on graphs. This subsection introduces deep graph generative models aiming to mimic real-world complex graphs. Generating complex graphs from latent representations is confronted with great challenges due to high nonlinearity and arbitrary connectivity of graphs. Note that graph translation [267] is akin to graph generation. However, their difference lies in that the former takes two graphs, i.e., input graph and target graph, as input, and the latter only takes a single graph as input. The NetGAN [19] utilizes the generative adversarial network [93] to mimic the input real-world graphs. More specifically, it is composed of two components, i.e., a generator G and a discriminator D, as well. The discriminator D is modeled as an LSTM in order to distinguish real vertex sequences, which are yielded by the second-order random walks scheme vertex2vec [2], from faked ones. The generator G aims to generate faked vertex sequences via another LSTM, whose generating process is as follows:

$$v_0 = 0, \quad mathbfz \sim N_m(\mathbf{0}_m, \mathbf{I}_m), \quad (\mathbf{c}_0, \mathbf{h}_0) = g_{\theta'}(\mathbf{z}),$$
$$(\mathbf{c}_j, \mathbf{h}_j, \mathbf{o}_j) = \mathrm{LSTM}_G(\mathbf{c}_{j-1}, \mathbf{h}_{j-1}, \mathbf{v}_{j-1}), \quad v_j \sim \mathrm{Cat}\left(\mathrm{Softmax}(\mathbf{o}_j)\right).$$

The motif-targeted GAN [10] generalizes random-walk-based architecture of the NetGAN to characterize mesoscopic context of vertices. Different from [10, 19], the GraphVAE [157] adopts a probabilistic graph decoder to generate a probabilistic fully connected graph and then employs approximate graph matching to reconstruct the input graph. Its reconstruction loss is the cross-entropy between the input and reconstructed graphs. The literature [304] defines a sequential decision-making process to add a vertex/edge via the graph network [195], readout operator, and softmax function. The GraphRNN [111] is a deep autoregressive model, which generates graphs by training on a representative set of graphs and decomposes the graph generation process into a sequence of vertex and edge formations conditioned on the current generated graph. The literature [274] formulated a novel problem of generating graphs from given related context spaces and proposed a novel unified model of graph variational generative adversarial nets that can tackle the challenges of flexible context-structure conditioning and permutation-invariant generation.

4.6 Combining PI and GNNs

The GNNs and PI are two different learning paradigms for complicated real-world data. The former specializes in learning hierarchical representations based on local and global structural information, and the latter learning the dependencies between random variables. This subsection provides a summarization of studies combining these two paradigms.

4.6.1 Similarity-Preserving CRF Layer. The literature [89] proposes a CRF layer for the GCNNs to enforce hidden layers to preserve similarities between vertices. Specifically, the input $\mathbf{X}^{(l)}$ to the $(l+1)$-th layer is a random matrix around the output $\mathbf{B}^{(l)} = \mathrm{GCNN}(\mathbf{X}^{(l-1)}, \mathbf{A}_G, \mathbf{X})$ of the $(l-1)$-th layer. The objective function for the GCNN with a CRF layer can be reformulated as

$$J(\mathbf{W}; \mathbf{X}, \mathbf{A}_G, \mathbf{Y}) = \mathcal{L}\left(\mathbf{Y}; \mathbf{B}^{(L)}\right) + \sum_{l=1}^{L-1}\left(\frac{\gamma}{2}\|\mathbf{X}^{(l)} - \mathbf{B}^{(l)}\|_F^2 + \mathcal{R}(\mathbf{X}^{(l)})\right),$$

where the first term after "=" is the conventional loss function for the semi-supervised vertex classification problem, and the last term is a regularization one implementing similarity constraint. The similarity constraint $\mathcal{R}(\mathbf{X}^{(l)})$ is modeled as a CRF, i.e., $p(\mathbf{X}^{(l)}|\mathbf{B}^{(l)}) = \frac{1}{Z(\mathbf{B}^{(l)})}\exp(-E(\mathbf{X}^{(l)}|\mathbf{B}^{(l)}))$, where the energy function $E(\mathbf{X}^{(l)}|\mathbf{B}^{(l)})$ is formulated as

$$\begin{aligned} & E\left(\mathbf{X}^{(l)}|\mathbf{B}^{(l)}\right) \\ = & \sum_{v\in V}\varphi_v(\mathbf{X}^{(l)}[v,:],\mathbf{B}^{(l)}[v,:]) + \sum_{(u,v)\in E}\varphi_{u,v}\left(\mathbf{X}^{(l)}[u,:],\mathbf{X}^{(l)}[v,:],\mathbf{B}^{(l)}[u,:],\mathbf{B}^{(l)}[v,:]\right). \end{aligned}$$

Let $s_{u,v}$ denote the similarity between u and v. The unary energy component $\varphi_v(\cdot,\cdot)$ and pairwise energy component $\varphi_{u,v}(\cdot,\cdot,\cdot,\cdot)$ for implementing the similarity constraint are respectively formulated as

$$\begin{aligned} \varphi_v\left(\mathbf{X}^{(l)}[v,:],\mathbf{B}^{(l)}[v,:]\right) &= \|\mathbf{X}^{(l)}[v,:]-\mathbf{B}^{(l)}[v,:]\|_2^2, \\ \varphi_{u,v}\left(\mathbf{X}^{(l)}[u,:],\mathbf{X}^{(l)}[v,:],\mathbf{B}^{(l)}[u,:],\mathbf{B}^{(l)}[v,:]\right) &= s_{u,v}\|\mathbf{X}^{(l)}[u,:]-\mathbf{X}^{(l)}[v,:]\|_2^2. \end{aligned}$$

The mean-field variational Bayesian inference is employed to approximate the posterior $p(\mathbf{B}^{(l)}|\mathbf{X}^{(l)})$. Consequently, the CRF layer is defined as

$$\left(\mathbf{X}^{(l)}[v,:]\right)^{(k+1)} = \frac{\alpha\mathbf{B}^{(l)}[v,:]+\beta\sum_{u\in N_G(v)}s_{u,v}\left(\mathbf{X}^{(l)}[u,:]\right)^{(k)}}{\alpha+\beta\sum_{u\in N_G(v)}s_{u,v}}.$$

4.6.2 *Semi-supervised Conditional GCNNs.* The conditional GCNNs incorporate the CRF into the conventional GCNNs so that the semi-supervised vertex classification can be enhanced by both the powerful vertex representations and the dependencies of vertex labels. The GMNN [163] performs the semi-supervised vertex classification by incorporating the GCNN into the **Statistical Relational Learning (SRL)**. Specifically, the SRL usually models the conditional probability $p(Y_V|X_V)$ with the CRF, i.e., $p(\mathbf{Y}_V|\mathbf{X}_V) = \frac{1}{Z(X_V)}\prod_{(i,j)\in E}\varphi_{i,j}(\mathbf{y}_{i,:},\mathbf{y}_{j,:},\mathbf{X}_V)$. Note that $\mathbf{Y}_V$ is composed of the observed vertex labels $\mathbf{Y}_L$ and hidden vertex labels $\mathbf{Y}_U$. The variational Bayesian inference is employed to estimate the posterior $p(\mathbf{Y}_U|\mathbf{Y}_L,\mathbf{X}_V)$. The objective function is defined as

$$\text{ELBO}\left(q_{\theta_v}(\mathbf{Y}_U|\mathbf{X}_V)\right) = \mathbb{E}_{q_{\theta_v}(\mathbf{Y}_U|\mathbf{X}_V)}\left[\log\left(p_{\theta_l}(\mathbf{Y}_L,\mathbf{Y}_U|\mathbf{X}_V)\right) - \log\left(q_{\theta_v}(\mathbf{Y}_U|\mathbf{X}_V)\right)\right].$$

This objective can be optimized by the variational Bayesian EM algorithm [198], which iteratively updates the variational distribution $q_{\theta_v}(\mathbf{Y}_U|\mathbf{X}_V)$ and the likelihood $p_{\theta_l}(\mathbf{Y}_U|\mathbf{Y}_L,\mathbf{X}_V)$. In the VBE stage, $q_{\theta_v}(\mathbf{Y}_U|\mathbf{X}_V) = \prod_{v\in U}q_{\theta_v}(\mathbf{y}_v|\mathbf{X}_V)$, and $q_{\theta_v}(\mathbf{y}_v|\mathbf{X}_V)$ is approximated by a GCNN. In the VBM stage, the pseudo-likelihood is employed to approximate

$$\mathbb{E}_{q_{\theta_v}(\mathbf{Y}_U|\mathbf{X}_V)}\left[\sum_{v\in V}\log p_{\theta_l}\left(\mathbf{y}_n|\mathbf{Y}_{V\backslash v},\mathbf{X}_V\right)\right] = \mathbb{E}_{q_{\theta_v}(\mathbf{Y}_U|\mathbf{X}_V)}\left[\sum_{v\in V}\log p_{\theta_l}\left(\mathbf{y}_n|\mathbf{Y}_{N_G(v)},\mathbf{X}_V\right)\right],$$

and $p_{\theta_l}(\mathbf{y}_n|\mathbf{Y}_{N_G(v)},\mathbf{X}_V)$ is approximated by another GCNN. The literature [232] adopts the similar idea to the GMNN. Its posterior is modeled as a CRF with unary energy components and pairwise energy components whose condition is the output of the prescribed GCNN. The maximum likelihood estimation is employed to estimate the model parameters.

4.6.3 *GCNN-based Gaussian Process Inference.* A **Gaussian Process (GP)** defines a distribution over a function space and assumes any finite collection of marginal distributions follows a multivariate Gaussian distribution. A function $f:\mathbb{R}^d\rightarrow\mathbb{R}$ follows a Gaussian Process $\text{GP}(m(\cdot),\kappa(\cdot,\cdot))$ iff $(f(\mathbf{x}_1),\ldots,f(\mathbf{x}_N))^T$ for any N d-dimensional random vectors, and follows an N-dimensional Gaussian distribution $\mathcal{N}_N(\mu,\Sigma)$, where $\mu=(m(\mathbf{x}_1),\ldots,m(\mathbf{x}_N))^T$ and $\Sigma=[\kappa(\mathbf{x}_j,\mathbf{x}_k)]_{N\times N}$. For two d-dimensional random vectors $\mathbf{x}$ and $\mathbf{x}'$, we have $\mathbb{E}[f(\mathbf{x})]=m(\mathbf{x})$ and $\text{Cov}(f(\mathbf{x}),f(\mathbf{x}'))=\kappa(\mathbf{x},\mathbf{x}')$. Given a collection of N samples $\mathcal{D}=\{(\mathbf{x}_j,\mathbf{y}_j):j=1,\ldots N\}$, the GP inference aims to calculate the probability $p(\mathbf{y}|\mathbf{x})$ for predictions, i.e.,

$$f\sim\text{GP}(0(\cdot),\kappa(\cdot,\cdot)), y_j\sim\text{DIST}(\lambda(f(\mathbf{x}_j))),$$

where $\lambda(\cdot)$ is a link function and DIST$(\cdot)$ denotes an arbitrary feasible noise distribution. To this end, the posterior $p(f|\mathcal{D})$ needs to be calculated out first. The literature [147] employs amortized variational Bayesian inference to approximate $p(f|\mathcal{D})$, i.e., $f = \mu + \mathbf{L}\epsilon$, and the GCNNs to estimate μ and $\mathbf{L}$.

4.6.4 Other GCNN-based Probabilistic Inference. The literature [152] combines the GCNNs and variational Bayesian inference to infer the input graph structure. The literature [139] infers marginal probabilities in probabilistic graphical models by incorporating the GCNNs to the conventional message-passing inference algorithm. The literature [308] approximates the posterior in Markov logic networks with the GCNN-enhanced variational Bayesian inference.

4.7 Adversarial Training of GNNs

In many scenarios where classifiers are deployed, adversaries intend to contaminate data so as to fake the classifiers [93, 250]. This is the so-called adversarial attack for the classification problems. For GNNS, the adversaries may also attack graph data so as to fake the node-, edge-, or graph-level classifier. Therefore, it is imperative to defend against the adversarial attacks for GNNs [145].

Given a graph dataset $\mathcal{D} = \{(c_j, G_{c_j}, y_j) : j = 1, \ldots, M\}$ where the target component c_j within a graph $G_{c_j} \in \mathcal{G}$ is associated with a corresponding ground-truth label y_j, after slightly perturbing G_j (denoted as $\widetilde{G}_{c_j}$), the adversarial samples $\widetilde{G}_{c_j}$ and G_{c_j} should be similar under an equivalence indicator, but the performance on node-, edge-, or graph-level tasks sharply decays. In general, given a sample $(c, G_c, \mathrm{y}) \in \mathcal{D}$ and a classifier f, the graph adversarial attacker $g : \mathcal{F} \times \mathcal{G} \to \mathcal{G}$ attempts to perturb a graph $G = (V, E)$ into $\widetilde{G} = (\widetilde{V}, \widetilde{E})$, such that

$$\begin{array}{ll} \max_g & \mathbb{I}\left(f(c, \widetilde{G}_c) \neq \mathrm{y}\right) \\ \text{s.t.} & \widetilde{G}_c = g(f, (c, G_c, \mathrm{y})), \\ & \mathcal{I}(G_c, \widetilde{G}_c, c) = 1, \end{array}$$

where the equivalence indicator $\mathcal{I} : \mathcal{G} \times \mathcal{G} \times V \to \{0, 1\}$ checks whether two graphs G_c and $\widetilde{G}_c$ are equivalent under the classification semantics. The equivalence indicators are usually defined in two fashions, namely explicit semantics and small modifications. The explicit semantics are defined as $\mathcal{I}(G_c, \widetilde{G}_c, v) = \mathbb{I}(f^*(G_c, v) = f^*(\widetilde{G}_c, v))$, where f^* is a gold standard classifier, and the small modifications are defined as $\mathcal{I}(G_c, \widetilde{G}_c, c) = \mathbb{I}(|(E - \widetilde{E}) \cup (\widetilde{E} - E)| < m) \cdot \mathbb{I}(\widetilde{E} \subseteq N(G, b))$, where $N(G, b) = \{(u, v) : u, v \in V, \mathrm{dist}_G(u, v) <= b\}$.

4.7.1 Adversarial Attacks on Graphs. The literature [81] first proposes a reinforcement-learning-based attack method, which can learn a generalizable attack policy, on graphs. It provides a definition for graph adversarial attackers. Specifically, the attack procedure is modeled as a finite horizon **Markov Decision Process (MDP)** $\mathcal{M}_m(f, G_c, c, \mathrm{y})$ and is therefore optimized by Q-learning with a hierarchical Q-function. For the MDP $\mathcal{M}_m(f, G_c, c, \mathrm{y})$, its action $a_t \in \mathcal{A} \subseteq V \times V$ at time step t is defined to add or delete edges in the graph, its state $(\widehat{G}_t, c)$ at time step t is a partially modified graph with some of the edges added/deleted from G, and the reward function is defined as

$$R\left(\widetilde{G}_c, c\right) = \begin{cases} 1 & f(\widetilde{G}_c, c) \neq y \\ -1 & f(\widetilde{G}_c, c) = y. \end{cases}$$

Note that the GCNNs are employed to parameterize the Q-function. The NETTACK [45] considers attacker vertices in $\mathcal{V}$ to satisfy a feature attack constraint $\mathbf{X}'[u, j] \neq \mathbf{X}^{(0)}[u, j] \Longrightarrow u \in \mathcal{V}$, a structure attack constraint $\mathbf{A}'_{u,v} \neq \mathbf{A}^{(0)}_{u,v} \Longrightarrow u \in \mathcal{V} \vee v \in \mathcal{V}$, and an equivalence indicator constraint $\sum_u \sum_j |\mathbf{X}^{(0)}[u, j] - \mathbf{X}'[u, j]| + \sum_{u<v} |\mathbf{A}^{(0)}[u, v] - \mathbf{A}'[u, v]| \leq \Delta$, where G' is derived by perturbing

$G^{(0)}$. Let $\mathcal{P}^{G0}_{\Delta,\mathcal{A}}$ denote the set of all perturbed graphs G' satisfying these three constraints. The goal is to find a perturbed graph $G' = (\mathbf{A}', \mathbf{X}')$ that classifies a target vertex v_0 as c_{new} and maximizes the log-probability/logit to c_{old}, i.e., $\max_{(\mathbf{A}',\mathbf{X}')\in\mathcal{P}^{G0}_{\delta,\mathcal{A}}} \max_{c_{\text{new}}\neq c_{\text{old}}} \log \mathbf{Z}[v_0, c_{\text{new}}]^* - \log \mathbf{Z}[v_0, c_{\text{old}}]^*$, where $\mathbf{Z}^* = f_{\theta^*}(\mathbf{A}', \mathbf{X}')$ with $\theta^* = \arg\min_\theta \mathcal{L}(\theta; \mathbf{A}', \mathbf{X}')$. The NETTACK employs the GCNNs to model the classifier. The literature [46] adopts the similar equivalence indicator, and poisoning attacks are mathematically formulated as a bilevel optimization problem, i.e.,

$$\begin{aligned} \min_{\widetilde{G}} \quad & \mathcal{L}_{\text{attack}}\left(f_{\theta^*}(\widetilde{G})\right) \\ s.t. \quad & \widetilde{G} \in \mathcal{P}^{G0}_{\Delta,\mathcal{A}} \\ & \theta^* = \arg\min_\theta \mathcal{L}_{\text{train}}\left(f_\theta(\widetilde{G})\right). \end{aligned} \tag{18}$$

This bilevel optimization problem in Equation (18) is then tackled using meta-gradients, whose core idea is to treat the adjacency matrix of the input graph as a hyperparameter.

4.7.2 Defense Against Adversarial Attacks. A robust GCNN requires that it is invulnerable to perturbations of the input graph. The **robust GCN (RGCN)** [133] can fortify the GCNs against adversarial attacks. More specifically, it adopts Gaussian distributions as the hidden representations of vertices, i.e.,

$$\mathbf{X}^{(l+1)}[j,:] \sim N\left(\mu_j^{(l+1)}, \text{diag}(\sigma_j^{(l+1)})\right),$$

in each graph convolution layer so that the effects of adversarial attacks can be absorbed into the variances of the Gaussian distributions. The Gaussian-based graph convolution is defined as

$$\mu_j^{(l+1)} = \rho\left(\sum_{v_k\in N_G(v_j)} \frac{\mu_k^{(l)} \circledast \alpha_k^{(l)}}{\sqrt{\widetilde{\mathbf{D}}_{j,j}\widetilde{\mathbf{D}}_{k,k}}} \mathbf{W}_\mu^{(l)}\right), \sigma_j^{(l+1)} = \rho\left(\sum_{v_k\in N_G(v_j)} \frac{\sigma_k^{(l)} \circledast \alpha_k^{(l)} \circledast \alpha_k^{(l)}}{\widetilde{\mathbf{D}}_{j,j}\widetilde{\mathbf{D}}_{k,k}} \mathbf{W}_\sigma^{(l)}\right),$$

where $\alpha_j^{(k)}$ are attention weights. Finally, the overall loss function is defined as regularized cross-entropy. The literature [316] presents a batch virtual adversarial training method that appends a novel regularization term to the conventional objective function of the GCNNs, i.e.,

$$\mathcal{L} = \mathcal{L}_0 + \alpha \cdot \frac{1}{N}\sum_{u\in V} E(p(\mathbf{y}|\mathbf{X}_u, \mathbf{W})) + \beta \cdot \mathcal{R}_{vadv}(\mathbf{V}, \mathbf{W}),$$

where $\mathcal{L}_0$ is an average cross-entropy loss of all labeled vertices, $E(\cdot)$ is the conditional entropy of a distribution, and $\mathcal{R}_{\text{vadv}}(\mathbf{V}, \mathbf{W})$ is the average **Local Distributional Smoothness (LDS)** loss for all vertices. Specifically, $\mathcal{R}_{\text{vadv}}(\mathbf{V}, \mathbf{W}) = \frac{1}{N}\sum_{u\in V} \text{LDS}(\mathbf{X}[u,:], \mathbf{W}, r_{\text{vadv},u})$, where $\text{LDS}(\mathbf{x}, \mathbf{W}, r_{\text{vadv}}) = D_{KL}(p(\mathbf{y}|\mathbf{x}, \widehat{\mathbf{W}})||p(\mathbf{y}|\mathbf{x} + r_{\text{vadv}}, \mathbf{W}))$ and r_{vadv} is the virtual adversarial perturbation. Additionally, there are some other studies aiming at verifying certifiable (non-)robustness to structure and feature perturbations for the GCNNs and developing a robust training algorithm [18, 47]. The literature [133] proposes to improve GCN generalization by minimizing the expected loss under small perturbations of the input graph. Its basic assumption is that the adjacency matrix A_G is perturbed by some random noise. Under this assumption, the objective function is defined as $\min_{\mathbf{W}} \int q(\epsilon|\alpha)\mathcal{L}(\mathbf{X}, \mathbf{Y}, \mathbf{A}_G(\epsilon), \mathbf{W})d\epsilon$, where $\mathbf{A}_G(\epsilon)$ denotes the perturbed adjacency matrix of G and $q(\epsilon|\alpha)$ is a zero-centered density of the noise ϵ so that the learned GCN is robust to this noise and has good generalization ability.

4.8 Graph Neural Architecture Search

Neural Architecture Search (NAS) [234] has achieved notable success in discovering the optimal neural network architecture for image and language learning tasks. However, existing NAS algorithms cannot be directly generalized to find the optimal GNN architecture. The reason is twofold: (1) the search spaces of GNNs are quite different from the ones of the conventional deep neural networks, and (2) the capabilities of GNNs are usually vulnerable to the architectures of GNNs. Fortunately, there have been some studies to bridge this gap. In general, the **Graph Neural Architecture Search (GNAS)** can be defined as follows. Given search space $\mathcal{F}$, training set $\mathcal{D}_{\text{train}}$, validation set $\mathcal{D}_{valid}$, and evaluation metric $\mathcal{M}$, the GNAS aims to find the optimal GNN architecture $f^*\mathcal{F}$ accompanied with the test metric M^* on set $\mathcal{D}_{\text{valid}}$. Specifically, it can be formulated as

$$f^* = \arg\max_{f \in \mathcal{F}} \mathcal{M}\left(f\left(\theta^*\right), \mathcal{D}_{\text{valid}}\right), \quad \theta^* = \arg\min_{\theta} \mathcal{L}\left(f\left(\theta\right), \mathcal{D}_{\text{train}}\right),$$

where θ^* denotes the model parameter for the architecture f, $\mathcal{L}$ denotes the loss function, and the metric $\mathcal{M}$ could be the F1 score or accuracy for the classification problem. The literature [131] proposed a reinforced conservative neural architecture search algorithm consisting of three components: (1) a conservative explorer screening out the best architecture so far, (2) a guided architecture modifier mutating certain actions in the retained best architecture, and (3) a reinforcement learning trainer learning the architecture mutation causality. The literature [74] employed an RNN to generate variable-length strings that describes the graph neural architecture and then trained the RNN with policy gradient to maximize the expected accuracy of the generated architectures on a validation dataset. The literature [28] proposes a **Graph HyperNetwork (GHN)** to amortize the search cost of training thousands of different networks, which is trained to minimize the training loss of the sampled network with the weights generated by a GCNN.

4.9 Graph Reinforcement Learning

Deep reinforcement learning (DRL) has achieved tremendous success in sequential decision-making problems. Many researchers from the GNN community would like to investigate how DRL can be used to solve optimization problems such as routing on graphs. The literature [189] found that most of the state-of-the-art DRL-based networking techniques fail to generalize onto graphs that cannot be seen in the training stage. This is because existing DRL networking solutions usually use conventional deep neural networks that are unable to extract graph-structured information. This article overcame this limitation and successfully incorporated the GNNs into the DRL. The proposed model DRL+GNN, which employs GCNNs to model the Q-value function, is able to not only operate and optimize decision-making problems on graphs but also generalize onto unseen graphs. The literature [283] learns to walk over a graph from a source vertex toward a target vertex for a given query via reinforcement learning. The proposed agent M-Walk is composed of a deep RNN and **Monte Carlo Tree Search (MCTS)**. The former maps a hidden vector representation h_t, yielded by a special RNN encoding the state s_t at time t, to a policy and Q-values, and the latter is employed to generate trajectories yielding more positive rewards. The literature [239] proposed NerveNet to explicitly model the underlying structure of an agent in the form of a graph. The NerveNet served as the agent's policy network, propagated information over the agent graph, and then inferred actions for different parts of the agent.

4.10 Applications

In this subsection, we introduce the applications of the GNNs, including complex network, combinatorial optimization, knowledge graph, bioinformatics, chemistry, brain science, physical system, source code analysis, intelligent traffics, recommender systems, computer vision, and natural language processing. In the field of complex networks, GNNs can be used to predict users' social influence [117], predict missing links [172], and compute graph similarity [307]. In the field of the combinatorial optimization, GNNs can be used to approximately solve many NP-complete or NP-hard problems, e.g., the **Traveling Sales Problem (TSP)** [154, 258], the minimum dominating set problem and the minimum vertex cover problem [211, 319], and the Boolean satisfiability problem [44, 110, 319]. In the field of knowledge graphs, GNNs can be used to perform multi-hop knowledge reasoning [146], common-sense reasoning [14], and knowledge graph completion [269]. In the field of the bioinformatics, GNNs can be used to model polypharmacy side effects [155], identify disease-gene association [190], and predict protein interface [4]. In the field of chemistry, GNNs can be used to predict chemical reactivity [36], generate small molecular graphs [109, 177], and represent molecular fingerprints [229]. In the field of brain science, GNNs can be used to measure the distance between two functional brain networks [227] and predict clinical neurodevelopmental outcomes from brain networks [99]. In the field of physical systems, GNNs can be used to implement an inductive bias for object- and relation-centric representations of complex, dynamical physical systems [6]; understand physical, social, and team-play systems [282]; learn a physics simulator from video [176]; reason about objects, relations, and physics [193]; and understand our world in terms of objects, relations, and hierarchies [235]. In the field of source code analysis, GNNs can be used to predict the name of a variable given its usage and reason about selecting the correct variable that should be used at a given program location [168], predict both standard types an user-defined types that have not been encountered during training [113], and infer Javascript types [100]. In the field of intelligent traffics, GNNs can be used to predict traffic conditions for time steps ahead at different locations on a road network graph [31], infer road attributes such as lane count and road type from satellite imagery [228], and deduce future travel time by exploiting the data of upcoming traffic volume [207]. In the field of recommender systems, GNNs can be used in session-based recommendation [225], multi-component collaborative filtering [264], knowledge-graph-based recommendation [259], holistic sequential recommendation [25], meta-path-guided intent recommendation [218], and personalized recommendation of micro-video [287]. In the field of computer vision, GNNs can be used to develop a cross-modal graph matching strategy for the multiple-phrase visual grounding task [291], generate image caption with abstract scene graphs [224], reconstruct high-fidelity 3D face [104], detect human object interactions [185], recognize skeleton-based actions [27, 226], learn multi-label image and video classification [199, 238], learn on point clouds [58, 151, 301], generate scene graphs [42, 106, 286], and conduct visual question answering [40, 150, 160]. In the field of natural language processing, GNNs can be used for Chinese named entity recognition [231], event detection [233, 261], neural machine translation [43, 55], reading comprehension [24, 116, 169, 170], relation extraction [85, 182, 241], text classificaiton [92, 270], sentiment classification [17, 53, 315], text generation [204, 249, 295], and text question answering [178, 294].

5 IMPLEMENTATIONS AND EVALUATIONS

This section aims at telling readers how the GNNs are implemented and evaluated. The implementations provide convenient programming interfaces so that researchers can easily implement the existing GNN models and their own ones. The benchmarks provide ground truth for various GNN architectures so that different GNNs can be compared fairly. The evaluation pitfalls empirically show that the existing evaluation criteria have potential pitfalls.

5.1 Implementations

After understanding the principles of diverse GNN models, how to implement existing GNN architectures or your own GNN becomes more and more imperative. As far as I know, there are two widespread Python libraries for deep graph learning, entitled PyTorch Geometric [159] and **Deep Graph Library (DGL)** [171], so that researchers and engineers can use them to implement and deploy various GNN models into their application scenarios.

The PyTorch Geometric is a geometric deep learning extension library for PyTorch. It provides plenty of APIs for deep learning on irregular data structures, e.g., graphs and manifolds, and achieves high data throughput by leveraging sparse GPU acceleration, by providing dedicated CUDA kernels, and by introducing efficient mini-batch handling for input graphs of different sizes. Specifically, the PyTorch Geometric represents a graph as a vertex feature matrix and a sparse adjacency tuple consisting of an edge matrix encoding source and destination indices and edge feature matrix. Its APIs, e.g., data loading routines, model instantiations, GPU acceleration support, message passing schemes, etc., follow the PyTorch style so as to keep them as familiar as possible for users.

The DGL is a comprehensive deep graph learning library, which provides (1) graph abstraction; (2) flexible APIs; (3) support for gigantic, heterogeneous, and dynamic graphs; and (4) efficient memory usage and high training speed. Specifically, it represents a graph (DGLGraph) as a tuple consisting of a source vertex tensor and a destination vertex tensor. The vertex and edge attributes are incorporated into the DGLGraph respectively by two dictionaries "DGLGraph.ndata" and "DGLGraph.edata." After defining the DGLGraphs, another important task is to define the message passing procedure, which is decomposed into three independent functions, i.e., message function, aggregation function, and update function. The DGL provides plenty of APIs to implement the three kinds of functions and deploy them on the DGLGraph. In addition, the DGL can adequately leverage high-performance tensor computing library, autograd operations, and other feature extraction modules to accelerate the training procedure of GNNs.

5.2 Benchmarks

Graph neural networks have become a powerful toolkit for analyzing complex graphs. It becomes more and more critical to evaluate the effectiveness of new GNN architectures and compare different GNN models under a standardized benchmark with consistent experimental settings and large datasets. A feasible benchmark for the GNNs should include appropriate graph datasets, robust coding interfaces, and experimental settings so that different GNN architectures can be compared under the same settings.

The literature [245] constructed a reproducible GNN benchmarking framework providing the facility for researchers to add new graph datasets and GNN models. Specifically, it releases an open-source benchmark infrastructure for GNNs, hosted on GitHub based on PyTorch [1] and DGL libraries [171]; introduces medium-scale graph datasets with 12k to 70k graphs of variable-size 9 to 500 vertices; and identifies important building blocks of GNNs (graph convolutions, anistropic diffusion, residual connections, and normalization layers) with the proposed benchmark infrastructures. The goal of this work is to provide convenience for researchers to use a collection of medium-scale graph datasets on which performance of different GNN models can be gauged under the same circumstances. This article proposed six datasets, which are described in Table 6. The MNIST [275] and CIFAR10 [5] from the computer vision are converted into graphs using so-called super-pixels. The task is then to classify these graphs into the true categories. The ZINC [255] is an existing real-world molecular graph dataset with atoms as vertices and bonds as edges. The task is to regress a molecule property known as the constrained solubility. The remaining graph datasets in Table 6 are constructed according to prescribed models. Specifically, the PATTERN and CLUSTER are generated according to the stochastic block model, and the TSP is based on the

Table 6. Summary Statistics of Proposed Benchmark Datasets [245]

Domain	**Dataset**	**# Graphs**	**# vertices**
Computer Vision	MNIST	70K	40–75
	CIFAR10	60K	85–150
Chemistry	ZINC	12K	9–37
Stochastic Block Model	PATTERN	14K	50–180
	CLUSTER	12K	40–190
Uniform Distribution	TSP	12K	50–500

Table 7. Overview of Currently Available OGB Datasets [91]

Task	**Scale**	**Domain**		
		Nature	**Society**	**Information**
Vertex Level	Small	-	arxiv	-
	Medium	proteins	products	-
	Large	-	-	-
Edge Level	Small	-	collab	-
	Medium	ppa	citation	wikikg
	Large	-	-	-
Graph Level	Small	molhiv	-	-
	Medium	molpcba, ppa	-	-
	Large	-	-	-

Traveling Salesman Problem. These proposed datasets are large enough so that differences observed between various GNN models are statistically relevant.

The literature [91] presents an **Open Graph Benchmark (OGB)**[1] including a diverse set of real-world and large-scale benchmark graph datasets and encompassing multiple important graph deep learning tasks from a diverse range of domains, e.g., social and information networks, biological networks, molecular graphs, and knowledge graphs. Its goal is to facilitate scalable, robust, and reproducible deep learning research on graphs. The OGB, which provides a unified evaluation protocol using application-specific train/validation/test dataset splits and evaluation metrics, releases an end-to-end graph processing pipeline including graph data loading, experimental settings, and model evaluations. The OGB includes five components, i.e., OGB graph datasets, OGB data loader, your learning model, OGB evaluator, and OGB Leaderboards. The OGB graph datasets contain real-world graph benchmark datasets at different scales from diverse application domains (e.g., nature, society, and information), which cover the vertex-, edge-, and graph-level tasks. The currently available OGB datasets are presented in Table 7. The nature domain includes biological networks and molecular graphs, the society domain includes academic graphs and e-commerce networks, and the information domain includes knowledge graphs. The OGB data loaders, which are compatible with the PyTorch Geometric and DGL, can automatically download and process graphs and further split the datasets in a standardized manner. Our learning models can be developed based on the available data loaders. The OGB evaluator can evaluate your models and gauge the model performance. Finally, the OGB provides leaderboards to keep track of recent advances.

[1]https://ogb.stanford.edu.

5.3 Evaluation Pitfalls

The literature [184] compares four typical GCNN architectures: GCN [237], MoNet [65], GAT [191], and GraphSAGE using three aggregation strategies [257] against four baseline models: logistic regression, multi-layer perceptron, label propagation, and normalized Laplacian label propagation, and uses a standardized training and hyperparameter tuning procedure for all these models so as to perform a more fair comparison. The experimental results show that different train/validation/test splits of datasets lead to dramatically different rankings of models. In addition, their findings also demonstrate that simpler GCNN architectures can outperform more sophisticated ones only if the hyperparameters and training procedures are tuned fairly for all models.

6 FUTURE RESEARCH DIRECTIONS

Although the GNNs have achieved tremendous success in many fields, there still exist some open problems. This section aims to tell readers which directions the GNNs will go to in the future.

6.1 Highly Scalable GNNs

The real-world graphs usually contain hundreds of millions of vertices and edges and have dynamically evolving characteristics. It turns out that it is difficult for the existing GNN architectures to scale up to the web-scale graphs. Now, all graph learning libraries leverage high-performance tensor computing libraries based on GPU to accelerate the training of GNNs. Hardware acceleration is only a feasible solution to processing web-scale graphs. Another crucial solution is to design highly scalable GNN architectures to accelerate the training procedure of GNNs. In general, many researchers take the conventional accelerating strategies, e.g., neighbor sampling and mini-batch, to expedite their GNN training. However, we should explore whether there exist more effective accelerating strategies than the conventional ones. This motivates us to design highly scalable GNN architectures that can efficiently and effectively learn vertex/edge/graph representations for gigantic dynamically evolving graphs. It should be noted that designing accelerating strategies must fully take the oversmoothing and overfitting phenomena observed on the GNNs into consideration. Readers interested in this topic can refer to the literatures [78, 79, 90, 149, 162, 253, 297, 327].

6.2 Robust GNNs

The existing GNN architectures are vulnerable to adversarial attacks. That is, the performance of the GNN models will sharply drop once the structure and/or initial features of the input graph are attacked by adversaries. Therefore, we should incorporate the attack-and-defense mechanism into the GNN architectures, i.e., constructing robust GNN architecture, so as to reinforce them against adversarial attacks. Current studies only considered adversarial attacks of specific types with certain assumptions over graph data. In addition, graph benchmarks for adversarial attacks and defenses are also lacking. Therefore, we should conduct our research in the future from two perspectives. One is to formulate a unified adversarial learning framework over graph data. A general adversarial attack on graphs should take the following six factors into account: structure and attribute perturbations, attack stage, attack objective, attack knowledge, attack goal, and attack task. After being attacked by adversaries, we should design adversarial defense schemes on graphs to fortify the robustness of GNNs. In general, there are two manners to implement the adversarial defense on graphs: (1) formulating the adversarial training models according to the adversarial intrusion strategies and (2) identifying, reducing, or even eliminating the attacks over contaminated graphs. The second is to construct graph benchmarks for adversarial attack and defense over graphs and design unified metrics for evaluating attack-and-defense performance on graphs. Readers interested in this topic can refer to the literatures [18, 46, 47, 57, 81, 133, 145].

6.3 GNNs Going beyond WL Test

The capabilities (or expressiveness) of GNNs are limited by the 1-WL test, and the higher-order WL test is computationally expensive. Consequently, two non-isomorphic graphs may yield the same vertex/edge/graph representations under appropriate conditions. This motivates us to develop a novel GNN framework going beyond the WL test, or design an elegant higher-order GNN architecture corresponding to the higher-order WL test. In fact, a critical point is how to determine whether two graphs are isomorphic. As well known, we do not know the complexity of the graph isomorphism problem (polynomial or NP-complete). However, most researchers are inclined to believe that it can be solved in polynomial time. Breaking through the graph isomorphism problem is vital to ameliorate the capabilities of GNNs. As far as this goes, we could focus on how to design injective message, aggregation, and update functions so as to fortify the expressive ability of GNNs. Although the literature [138] proposed a graph isomorphism network that provably satisfies the injectiveness condition, it approximated the message, aggregation, and update functions with neural networks. The injectiveness cannot be guaranteed strictly. Up to now, the expressive power of GNNs has become a hot topic in the GNN community, and lots of researchers are concerned with this topic. Readers interested in this topic can refer to the literatures [181, 186, 210, 243].

6.4 Explainable GNNs

The existing GNNs run in a black box. We do not understand why they achieve state-of-the-art performance on the vertex-, edge-, and graph-level tasks. Studying the explainability of GNNs can help to identify and localize contextual subgraphs relevant to the models' decisions within input graphs. Now, explainability has become a major obstacle to extend the applications of GNNs. Although there have been some studies on the explainability of GNNs, they only focus on different aspects of the GNN models and provide instance-dependent explanations for these models. In general, they answer a few questions such as "Which vertices or edges are more important?", "Which vertex or edge features are more important?", and "What graph patterns are vital to the prediction of a certain class?" In fact, a more important point lies in how to explain the GNNs at the model self level. We need to design a unified explainable framework for diverse GNNs. In addition, we do not yet have a canonical explainability benchmark for GNNs. Readers interested in this topic can refer to the literatures [64, 83, 196, 317].

7 CONCLUSIONS

This article aims at demonstrating a panorama of GNNs for readers so as to let them know the taxonomy, advances, and trends of GNNs. We expand the content of this article from four perspectives: fundamental architectures, extended architectures and applications, implementations and evaluations, and future research directions. The fundamental architectures are expanded from five angles: graph convolutional neural networks, accelerating strategies for GCNN training, graph pooling operators, graph attention mechanisms, and graph recurrent neural networks. The extended architectures are expanded from nine perspectives: GCNNS on special graphs, capability and explainability, self-supervised graph learning, deep graph representation learning, deep graph generative models, combining PI and GNNs, adversarial training of GNNs, graph neural architecture search, and graph reinforcement learning. In the future directions, we propose four prospective research topics on GNNs: highly scalable GNNs, robust GNNs, GNNs going beyond WL test, and explainable GNNs. We expect that the relevant scholars can understand the computational principles of GNNs, consolidate the foundations of GNNs, and apply them to addressing more and more real-world issues through reading this survey.

REFERENCES

[1] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan Trevor Kileen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf Edward Yang, Zach Devito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. *arXiv* (2019). https://arxiv.org/abs/1912.01703.

[2] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'16)*. ACM, 855–864. https://doi.org/10.1145/2939672.2939754

[3] Afshin Rahimi, Trevor Cohn, and Timothy Baldwin. 2018. Semi-supervised user geolocation via graph convolutional networks. In *The Annual Meeting of the Association for Computational Linguistics (ACL'18)*. ACL, 2009–2019.

[4] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. 2017. Protein interface prediction using graph convolutional networks. In *The International Conference on Neural Information Processing Systems*. Curran Associates, Inc., 6530–6539.

[5] Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. (2009). https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[6] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. 2018. Graph networks as learnable physics engines for inference and control. In *The International Conference on Machine Learning (ICML'18)*, Vol. 80. PMLR, 4470–4479.

[7] Amir H. Khasahmadi, Kaveh Hassani, Parsa Moradi, Leo Lee, and Quaid Morris. 2020. Memory-based graph networks. In *The International Conference on Learning Representations (ICLR'20)*. Addis Ababa.

[8] Andreas Loukas. 2020. What graph neural networks cannot learn-depth vs width. In *The International Conference on Learning Representations (ICLR'20)*. Addis Ababa.

[9] Annyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. 2017. Cross-sentence N-ary relation extraction with graph LSTMs. *Transactions of the Association for Computational Linguistics* 5 (2017), 101–115. https://doi.org/10.1162/tacl_a_00049

[10] Anuththari Gamage, Eli Chien, Jianhao Peng, and Olgica Milenkovic. 2020. Multi-MotifGAN (MMGAN): Motif-targeted graph generation and prediction. In *The IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'20)*. IEEE, 4182–4186. https://doi.org/10.1109/ICASSP40776.2020.9053451

[11] Ashesh Jain, Amir R. Zamir, Silvio Savarese, and Ashutosh Saxena. 2016. Structural-RNN: Deep learning on spatio-temporal graphs. In *The International Conference on Computer Vision and Pattern Recognition (CVPR'16)*. IEEE, 5308–5317.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomiz, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *The International Conference on Neural Information Processing (NIPS'17)*. Curran Associates, Inc., Long Beach, CA, 5998–6008.

[13] B. W. Kernighan and S. Lin. 1970. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal* 49, 2 (1970), 291–307. https://doi.org/10.1002/j.1538-7305.1970.tb01770.x

[14] Bill Yuchen Lin, Xinyue Chen, Jamin Chen, and Xiang Ren. 2019. KagNet: Knowledge-aware graph networks for commonsense reasoning. In *The International Conference on Empirical Methods in Natural Language Processing (EMNLP'19)*. ACL, 2829–2839.

[15] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *The International Joint Conference on Artificial Intelligence (IJCAI'18)*. IJCAI Organization, 3634–3640. https://doi.org/10.24963/ijcai.2018/505

[16] Bingbing Xu, Huawei Shen, Qi Cao, Yunqi Qiu, and Xueqi Cheng. 2019. Graph wavelet neural network. In *The International Conference on Learning Representations (ICLR'19)*.

[17] Binxuan Huang and Kathleen Carley. 2019. Syntax-aware aspect level sentiment classification with graph attention networks. In *The Conference on Empirical Methods in Natural Language Processing (EMNLP'19)*. ACL, 5469–5477.

[18] Aleksandar Bojchevski and Stephan Günnemann. 2019. Certifiable robustness to graph perturbations. In *The International Conference on Neural Information Processing Systems (NeurPS'19)*, Vol. 32. Curran Associates, Inc.

[19] Bryan Perozzi, Rami Ai-Rfou, and Steven Skiena. 2014. DeepWalk: Online learning of social representations. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'14)*. ACM, 701–710. https://doi.org/10.1145/2623330.2623732

[20] Caglar Gulcehre, Misha Denil, Mateusz Malinowski, Ali Razavi, Razvan Pascanu, Karl Moritz Hermann, Peter Battaglia, Victor Bapst, David Raposo, Adam Santoro, and Nando de Freitas. 2019. Hyperbolic attention networks. In *The International Conference on Learning Representations (ICLR'19)*.

[21] Carl Yang, Aditya Pal, Andrew Zhai, Nikil Pancha, Jiawei Han, and Charles Rosenberg. 2020. MultiSage: Empowering GCN with contextualized multi-embeddings on web-scale multipartite networks. In *The ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'20)*. ACM, 2434–2443. https://doi.org/10.1145/3394486.3403293

[22] Carl Yang, Yuxin Xiao, Yu Zhang, Yizhou Sun, and Jiawei Han. 2020. Heterogeneous network representation: A unified framework with survey and benchmark. *IEEE Transactions on Knowledge and Data Engineering* (2020). https://doi.org/10.1109/TKDE.2020.3045924

[23] Jianfei Chen, Jun Zhu, and Le Song. 2018. Stochastic training of graph convolutional networks with variance reduction. In *The 35th International Conference on Machine Learning (ICML'18)*, Vol. 80. PMLR, 942–950.

[24] Yu Chen, Lingfei Wu, and Mohammed J. Zaki. 2020. GraphFlow: Exploiting conversation flow with graph neural networks for conversational machine comprehension. In *The International Joint Conference on Artificial Intelligence (IJCAI'20)*. IJCAI Organization, 1230–1236. https://doi.org/10.24963/ijcai.2020/171

[25] Cheng Hsu and Cheng-Te Li. 2021. RetaGNN: Relational temporal attentive graph neural networks for holistic sequential recommendation. In *The World Wide Web Conference (WWW'21)*. 499–508.

[26] Chenyi Zhuang and Qiang Ma. 2018. Dual graph convolutional networks for graph-based semi-supervised classification. In *The World Wide Web Conference (WWW'18)*. 499–508.

[27] Cheyang Si, Wentao Chen, Wei Wang, Liang Wang, and Tieniu Tan. 2019. An attention enhanced graph convolutional LSTM network for skeleton-based action recognition. In *The International Conference on Computer Vision and Pattern Recognition (CVPR'19)*. IEEE, 1227–1236.

[28] Chris Zhang, Mengye Ren, and Raquel Urtasun. 2019. Graph hypernetworks for neural architecture search. In *The International Conference on Learning Representations (ICLR'19)*.

[29] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *The IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'15)*. IEEE, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

[30] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2019. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *The Conference on Artificial Intelligence (AAAI'19)*. Association for the Advance of Artificial Intelligence, 4602–4609. https://doi.org/10.1609/aaai.v33i01.33014602

[31] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. 2020. GMAN: A graph multi-attention network for traffic prediction. In *The Conference on Artificial Intelligence (AAAI'20)*. Association for the Advance of Artificial Intelligence.

[32] Chun Wang, Shrui Pan, Guodong Long, Xingquan Zhu, and Jing Jiang. 2017. MGAE: Marginalized graph autoencoder for graph clustering. In *The International Conference on Information and Knowledge Management (CIKM'17)*. ACM, 889–898. https://doi.org/10.1145/3132847.3132967

[33] Fan R. K. Chung. 1992. *Spectral Graph Theory*. American Mathematical Society.

[34] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V. Chawla. 2019. Heterogeneous graph neural network. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'19)*. ACM, 793–803. https://doi.org/10.1145/3292500.3330961

[35] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. 2018. Learning structural node embeddings via diffusion wavelets. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'18)*. ACM, 1320–1329.

[36] Connor W. Coley, Wengong Jin, Luke Rogers, Timothy F. Jamison, Tommi S. Jaakkola, William H. Green, Regina Barzilay, and Klavs F. Jensen. 2019. A graph-convolutional neural network model for the prediction of chemical reactivity. *Chemical Science* 10, 2 (2019), 370–377. https://doi.org/10.1039/C8SC04228D

[37] Costas Mavromatis and George Karypis. 2020. Graph InfoClust: Leveraging cluster-level node information for unsupervised graph representation learning. *arXiv* (2020). https://arxiv.org/abs/2009.06946v1.

[38] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas N. Kipf, and Pietro Liò. 2018. Towards sparse hierarchical graph classifiers. *arXiv* (2018). https://arxiv.org/abs/1811.01287.

[39] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *The ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'16)*. ACM, New York, NY, 1225–1234. https://doi.org/10.1145/2939672.2939753

[40] Damien Teney, Lingqiao Liu, and Anton van den Hengel. 2017. Graph-structured representations for Visual Question Answering. In *The International Conference on Computer Vision and Pattern Recognition (CVPR'17)*. IEEE, 1–9.

[41] Dan Busbridge, Dane Sherburn, Pietro Cavallo, and Nils Y. Yhammerla. 2019. Relational graph attention networks. *arXiv* (2019). https://arxiv.org/abs/1904.05811.

[42] Danfei Xu, Yuke Zhu, Christopher B. Choy, and Fei-Fei Li. 2017. Scene graph generation by iterative message passing. In *The International Conference on Computer Vision and Pattern Recognition (CVPR'17)*. IEEE, 5140–5419.

[43] Daniel Beck, Gholamreza Haffari, and Trevor Cohn. 2018. Graph-to-sequence learning using gated graph neural networks. In *The Annual Meeting of the Association for Computational Linguistics (ACL'18)*, Vol. 1. ACL, 273–283.

[44] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. 2019. Learning a SAT solver from single-bit supervision. In *The International Conference on Learning Representations (ICLR'19)*.

[45] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *The International Conference on Machine Learning (ICML'18)*. PMLR, 2847–2856. https://doi.org/10.1145/3219819.3220078

[46] Daniel Zügner and Stephan Günnemann. 2019. Adversarial attacks on graph neural networks via meta learning. In *The International Conference on Learning Representations (ICLR'19)*.

[47] Daniel Zügner and Stephan Günnemann. 2019. Certifiable robustness and robust training for graph convolutional networks. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'19)*. ACM, 246–256. https://doi.org/10.1145/3292500.3330905

[48] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gomez-Bombarelli, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P. Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *The International Conference on Neural Information Processing Systems (NIPS'15)*. Curran Associates, Inc., 2224–2232.

[49] David I. Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. 2013. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine* 30, 3 (2013), 83–98.

[50] David K. Hammond, Pierre Vandergheynst, and Rémi Gribonval. 2011. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis* 30, 2 (2011), 129–150. https://doi.org/10.1016/j.acha.2010.04.005

[51] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. 2017. Variational inference: A review for statisticians. *Journal of the Americian Statistical Association* 112, 518 (2017), 859–877. https://doi.org/10.1080/01621459.2017.1285773

[52] Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda. 2020. A gentle introduction to deep learning for graphs. *Neural Networks* 129 (2020), 203–221. https://doi.org/10.1016/j.neunet.2020.06.006

[53] Deepanway Ghosal, Navonil Majumder, Soujanya Poria, Niyati Chhaya, and Alexander Gelbukh. 2019. DialogueGCN: A graph convolutional neural network for emotion recognition in conversation. In *The Conference on Empirical Methods in Natural Language Processing (EMNLP'19)*. ACL, 154–164.

[54] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *The Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'19)*, Vol. 1. ACL, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[55] Diego Marcheggiani, Joost Bastings, and Ivan Titov. 2018. Exploiting semantics in neural machine translation with graph convolutional networks. In *The Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'18)*. ACL, 486–492.

[56] Dingyuan Zhu, Peng Cui, Daixin Wang, and Wenwu Zhu. 2018. Deep variational network embedding in Wasserstein space. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'18)*. ACM, 2827–2836. https://doi.org/10.1145/3219819.3220052

[57] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2019. Robust graph convolutional networks against adversarial attacks. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'19)*. ACM, 1399–1407.

[58] Dong Wook Shu, Sung Woo Park, and Junseok Kwon. 2019. 3D point cloud generative adversarial network based on tree structured graph convolutions. In *The International Conference on Computer Vision (ICCV'19)*. IEEE, 3859–3868.

[59] Dongmian Zou and Gilad Lerman. 2019. Graph convolutional neural networks via scattering. *Applied and Computational Harmonic Analysis* 49, 3 (2019), 1046–1074. https://doi.org/10.1016/j.acha.2019.06.003

[60] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *The International Conference on Learning Representations (ICLR'15)*.

[61] Edouard Pineau and Nathan de Lara. 2019. Variational recurrent neural networks for graph classification. *arXiv* (2019). https://arxiv.org/abs/1902.02721.

[62] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. 2019. Variational graph recurrent neural networks. In *The International Conference on Neural Information Processing Systems (NeurPS'19)*. Curran Associates, Inc., 10701–10711.

[63] Fan-Yun Sun, Jordan Hoffman, Vikas Verma, and Jian Tang. 2019. InfoGraph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *The International Conference on Learning Representations (ICLR'19)*.

[64] Federico Baldassarre and Hossein Azizpour. 2019. Explainability techniques for graph convolutional networks. *arXiv* (2019). https://arxiv.org/abs/1905.13686.

[65] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *The Conference on Computer Vision and Pattern Recognition (CVPR'17)*. IEEE, 5425–5434.

[66] Felipe Petroski Such, Shagan Sah, Miguel Dominguez, Suhas Pillai, Chao Zhang, Andrew Michael, Nathan D. Cahill, and Raymond Ptucha. 2017. Robust spatial filtering with graph convolutional neural networks. *IEEE Journal of Selected Topics in Signal Processing* 11, 6 (2017), 884–896.

[67] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying graph convolutional networks. In *The International Conference on Machine Learning (ICML'19)*, Vol. 97. PMLR, 6861–6871.

[68] Fenyu Hu, Yanqiao Zhu, Shu Wu, Liang Wang, and Tieniu Tan. 2019. Semi-supervised node classification via hierarchical graph convolutional networks. *arXiv* (2019). https://arxiv.org/abs/1902.06667v2.

[69] Fisher Yu and Vladlen Koltun. 2016. Multi-scale context aggregation by dilated convolutions. *arXiv* (2016). https://arxiv.org/abs/1511.07122v3.

[70] Franco Manessi, Alessandro Rozza, and Mario Manzo. 2020. Dynamic graph convolutional networks. *Pattern Recognition* 97 (2020), No. 107000. https://doi.org/10.1016/j.patcog.2019.107000

[71] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2008), 61–80. https://doi.org/10.1109/TNN.2008.2005605

[72] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. Computational capabilities of graph neural networks. *IEEE Transactions on Neural Networks* 20, 1 (2009), 81–102.

[73] Hongyang Gao and Shuiwang Ji. 2019. Graph U-Nets. In *The International Conference on Machine Learning*, Vol. 97. PMLR, 2083–2092.

[74] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. 2020. Graph neural architecture search. In *The International Joint Conference on Artificial Intelligence (IJCAI'20)*. IJCAI Organization, 1403–1409. https://doi.org/10.24963/ijcai.2020/195

[75] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. In *The IEEE International Conference on Computer Vision and Pattern Recognition (CVPR'17)*. IEEE, 2261–2269. https://doi.org/10.1109/CVPR.2017.243

[76] Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. 2020. Generalization and representational limits of graph neural networks. In *The 37th International Conference on Machine Learning (ICML'20)*, Vol. 119. PMLR, 3419–3430.

[77] Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In *The International Joint Conference on Neural Networks (IJCNN'05)*, Vol. 2. 729–734.

[78] Guohao Li, Matthias, Bernard Ghanem, and Vladlen Koltun. 2021. Training graph neural networks with 1000 layers. In *The International Conference on Machine Learning (ICML'21)*. PMLR, 338–348. https://doi.org/10.1145/3394486.3403076

[79] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. 2019. DeepGCNs: Can GCNs go as deep as CNNs. In *The International Conference on Computer Vision (ICCV'19)*. IEEE, 9267–9276.

[80] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. 2019. Invariant and equivariant graph networks. In *The International Conference on Learning Representations (ICLR'19)*.

[81] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial attack on graph structured data. In *The International Conference on Machine Learning (ICML'18)*, Vol. 80. PMLR, 1115–1124.

[82] Hanlin Zhang, Shuai Lin, Weiyang Liu, Pan Zhou, Jian Tang, Xiaodan Liang, and Eric P. Xing. 2020. Iterative graph self-distillation. *arXiv* (2020). https://arxiv.org/abs/2010.12609v1.

[83] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. 2020. Explainability in graph neural networks: A taxonomic survey. *arXiv* (2020). https://arxiv.org/abs/2012.15445.

[84] Hao Yuan and Shuiwang Ji. 2020. StructPool: Structured graph pooling via conditional random fields. In *The International Conference on Learning Representations (ICLR'20)*. Addis Ababa.

[85] Hao Zhu, Yankai Lin, Zhiyuan Liu, Jie Fu, Tat-Seng Chua, and Maosong Sun. 2019. Graph neural networks with generated parameters for relation extraction. In *The Annual Meeting of the Association for Computational Linguistics*. ACL, 1331–1339.

[86] Kaveh Hassani and Amir Hosein Khasahmadi. 2020. Contrastive multi-view representation learning on graphs. In *The 37th International Conference on Machine Learning*, Vol. 119. PMLR, 4116–4126.

[87] Heng Chang, Yu Rong, Tingyang Xu, Wenbing Huang, Somayeh Sojoudi, Junzhou Huang, and Wenwu Zhu. 2020. Spectral graph attention network. *arXiv* (2020). https://arxiv.org/abs/2003.07450.

[88] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric graph convolutional networks. In *The International Conference on Learning Representations (ICLR'20)*.

[89] Hongchang Gao, Jian Pei, and Heng Huang. 2019. Conditional random field enhanced graph convolutional neural networks. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'19)*. ACM, 276–284. https://doi.org/10.1145/3292500.3330888

[90] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'18)*. ACM, 1415–1424. https://doi.org/10.1145/3219819.3219947

[91] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. In *The International Conference on Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 22118–22133.

[92] Hu Linmei, Tianchi Yang, Chuan Shi, Houye Ji, and Xiaoli Li. 2019. Heterogeneous graph attention networks for semi-supervised short text classification. In *The Conference on Empirical Methods in Natural Language Processing (EMNLP'19)*. ACL, 4821–4830.

[93] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Azron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *The International Conference on Neural Information Processing Systems (NIPS'14)*, Vol. 27. 2672–2680.

[94] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. 2004. Kernel k-means, spectral clustering and normalized cuts. In *The ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'04)*, Vol. 32. ACM, 551–556. https://doi.org/10.1145/1014052.1014118

[95] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. 2007. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29, 11 (2007), 1944–1957. https://doi.org/10.1109/TPAMI.2007.1115

[96] Ines Chami, Rex Ying, Christopher Re, and Jure Leskovec. 2019. Hyperbolic graph convolutional neural networks. In *The International Conference on Neural Information Processing Systems (NeurPS'19)*. Curran Associates, Inc., 4868–4879.

[97] James Atwood and Don Towsley. 2016. Diffusion-convolutional neural network. In *The International Conference on Neural Information Processing Systems (NIPS'16)*. Curran Associates, Inc., 1993–2001.

[98] Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. 2015. Towards AI-Complete question answering: A set of prerequisite toy tasks. *arXiv* (2015). https://arxiv.org/abs/1502.05698v1.

[99] Jeremy Kawahara, Colin J. Brown, Steven P. Miller, Brian G. Booth, Vann Chau, Ruth E. Grunau, Jill G. Zwicker, and Ghassan Hamarneh. 2017. BrainNetCNN: Convolutional neural networks for brain networks; towards predicting neurodevelopment. *NeuroImage* 146 (2017), 1038–1049. https://doi.org/10.1016/j.neuroimage.2016.09.046

[100] Jessica Schrouff, Kai Wohlfahrt, Bruno Marnette, and Liam Atkinson. 2019. Inferring Javascript types using graph neural networks. *arXiv* (2019). https://arxiv.org/abs/1905.06707.

[101] Jian Du, Shanghang Zhang, Guanhang Wu, José M. F. Moura, and Soummya Kar. 2018. Topology adaptive graph convolutional networks. *arXiv* (2018). https://arxiv.org/abs/1710.10370.

[102] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, 8 (2000), 888–905. https://doi.org/10.1109/34.868688

[103] Jianfei Chen, Jun Zhu, and Le Song. 2018. Stochastic training of graph convolutional networks with variance reduction. In *The International Conference on Machine Learning (ICML'18)*. PMLR, 942–950.

[104] Jiangke Lin, Yi Yuan, Tianjie Shao, and Kun Zhou. 2020. Towards high-fidelity 3d face reconstruction from in-the-wild images using graph convolutional networks. In *The International Conference on Computer Vision and Pattern Recognition (CVPR'20)*. IEEE.

[105] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. 2018. GaAN: Gated attention networks for learning on large and spatiotemporal graphs. In *The International Conference on Uncertainty in Artificial Intelligence (UAI'18)*. PMLR, No. 139.

[106] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. 2018. Graph R-CNN for scene graph generation. In *The European Conference on Computer Vision (ECCV'18)*. Springer.

[107] Yizhou Jiao, Yun Xiong, Jiawei Zhang, Yao Zhang, Tianqi Zhang, and Yangyong Zhu. 2021. Sub-graph contrast for scalable self-supervised graph representation learning. In *The International Conference on Data Mining (ICDM'21)*. IEEE.

[108] Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. 2020. Graph-Bert: Only attention is needed for learning graph representations. *arXiv* (2020). https://arxiv.org/abs/2001.05140.

[109] Jiaxuan You, Bowen Liu, Rex Ying, and Vijay Pande. 2018. Graph convolutional policy network for goal-directed molecular graph generation. In *The International Conference on Neural Information Processing Systems (NeurPS'19)*. Curran Assoicates, Inc., Montreal, Quebec, Canada, 6412–6422.

[110] Jiaxuan You, Haoze Wu, Clark Barrett, Raghuram Ramanujan, and Jure Leskovec. 2019. G2SAT: Learning to generate SAT formulas. In *The International Conference on Neural Information Processing Systems (NeurPS'18)*. Curran Associates, Inc., 10552–10563.

[111] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *The International Conference on Machine Learing (ICML'18)*, Vol. 80. PMLR, 5708–5717.

[112] Jiaxue You, Rex Ying, and Jure Leskovec. 2019. Position-aware graph neural networks. In *The International Conference on Machine Learning (ICML'19)*, Vol. 97. PMLR, 7134–7143.

[113] Jiayi Wei, Maruth Goyal, Greg Durrett, and Isil Dilling. 2020. LambdaNet: Probabilistic type inference using graph neural networks. In *The International Conference on Learning Representations (ICLR'20)*. Addis Ababa.

[114] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *The International Conference on Learning Representations (ICLR'18)*.

[115] Jie Zhou, Gangqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. *AI Open* 1 (2020), 57–81. https://doi.org/10.1016/j.aiopen.2021.01.001

[116] Jie Zhou, Xu Han, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2019. GEAR: Graph-based evidence aggregating and reasoning for fact verification. In *The Annual Meeting of the Association for Computational Linguistics (ACL'19)*. ACL, 892–901.

[117] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. 2018. DeepInf: Social influence prediction with deep learning. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'18)*. 234, 2110–2119.

[118] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. 2020. GCC: Graph contrastive coding for graph neural network pre-training. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'20)*. ACM, 1150–1160. https://doi.org/10.1145/3394486.3403168

[119] Jiwoong Park, Minsik Lee, Hyung Jin Chang, Kyuewang Lee, and Jin Young Choi. 2019. Symmetric graph convolutional autoencoder for unsupervised graph representation learning. In *The International Conference on Computer Vision (ICCV'19)*. IEEE, Seoul, Korea, 6519–6528.

[120] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and locally connected networks on graphs. In *The International Conference on Learning Representations (ICLR'14)*.

[121] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict the propagate: Graph neural networks meet personalized pagerank. In *The International Conference on Learning Representations (ICLR'19)*.

[122] John B. Lee, Ryan A. Rossi, Sungchul Kim, Nesreen K. Ahmed, and Eunyee Koh. 2019. Attention models in graphs: A survey. *ACM Transactions on Knowledge Discovery from Data* 13, 6 (2019), No. 62.

[123] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. 2018. Graph classification using structural attention. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'18)*. ACM, 1666–1674. https://doi.org/10.1145/3219819.3219980

[124] John Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random field: Probabilistic models for segmenting and labeling sequence data. In *The International Conference on Machine Learning (ICML'01)*, Vol. 97. PMLR, 282–289.

[125] Juanhui Li, Yao Ma, Yiqi Wang, Charu Aggarwal, Chang-Dong Wang, and Jiliang Tang. 2020. Graph pooling with representativeness. In *the IEEE International Conference on Data Mining (ICDM)*. IEEE, 20424165. https://doi.org/10.1109/ICDM50108.2020.00039

[126] Jun Wu, Jingrui He, and Jiejun Xu. 2019. DEMO-Net: Degree-specific graph neural networks for node and graph classification. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'19)*. ACM, 406–415.

[127] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-attention graph pooling. In *The International Conference on Machine Learning (ICML'19)*, Vol. 97. PMLR, 3734–3743.

[128] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural message passing for quantum chemistry. In *The International Conference on Machine Learning (ICML'17)*, Vol. 70. PMLR, 1263–1272.

[129] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *The Annual Meeting of the Association for Computational Linguistics (ACL'15)*, Vol. 1. ACL, 1556–1566.

[130] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *The International Conference on Computer Vision and Pattern Recognition (CVPR'16)*. IEEE, 770–778. https://doi.org/10.1109/CVPR.2016.90

[131] Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. 2019. Auto-GNN: Neural architecture search of graph neural networks. *arXiv* (2019). https://arxiv.org/abs/1909.03184.

[132] Karsten Borgwardt, Elisabetta Ghisu, Felipe Llinares-López, Leslie O'Bray, and Bastian Rieck. 2020. Graph kernels: State-of-the-art and future challenges. *Foundations and Trends in Machine Learning* 13, 5–6 (2020), 531–712. https://doi.org/10.1561/2200000076

[133] Ke Sun, Piotr Koniusz, and Zhen Wang. 2019. Fisher-Bures adversary graph convolutional networks. In *The International Conference on Uncertainty in Artificial Intelligence (UAI'19)*. PMLR, No. 161.

[134] Ke Tu, Peng Cui, Xiao Wang, Fei Wang, and Wenwu Zhu. 2018. Structural deep embedding for hyper-networks. In *The International Conference on Artificial Intelligence (AAAI'18)*. Association for the Advances of Artificial Intelligence, 426–433.

[135] Ke Tu, Peng Cui, Xiao Wang, and Philip S. Yu. 2018. Deep recursive network embedding with regular equivalence. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'18)*. ACM, 2357–2366. https://doi.org/10.1145/3219819.3220068

[136] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation learning on graphs with jumping knowledge networks. In *The International Conference on Machine Learning (ICML'18)*, Vol. 80. PMLR, 5453–5462.

[137] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2020. What can neural networks reason about. In *The International Conference on Learning Representations (ICLR'20)*. Addis Ababa.

[138] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks. In *The International Conference on Learning Representations (ICLR'19)*.

[139] KiJung Yoon, Renjie Liao, Yuwen Xiong, Lisa Zhang, Ethan Fetaya, Raquel Urtasun, Richard Zemel, and Xaq Pitkow. 2018. Inference in probabilistic graphical models by graph neural networks. In *The International Conference on Learning Representations (ICLR'18)*. Vancouver, Canada.

[140] Kilian Weinberger, Anirban Dasgupta, and John Langford. 2009. Feature hashing for large scale multitask learning. In *The International Conference on Machine Learning (ICML'09)*. PMLR, 1113–1120. https://doi.org/10.1145/1553374.1553516

[141] Kiran K. Thekumparampil, Chong Wang, Sewoong Oh, and Lijia Li. 2018. Attention-based graph neural network for semi-supervised learning. *arXiv* (2018). https://arxiv.org/abs/1803.03735.

[142] Kun Xu, Lingfei Wu, Zhiguo Wang, Yansong Feng, Michael Witbrock, and Vadim Sheinin. 2018. Graph2Seq: Graph to sequence learning with attention-based neural networks. *arXiv* (2018). https://arxiv.org/abs/1804.00823.

[143] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Youshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *The Conference on Empirical Methods in Natural Language Processing (EMNLP'14)*. ACL, 1724–1734. https://doi.org/10.3115/v1/D14-1179

[144] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. 2020. Distance encoding: Design provably more powerful neural networks for graph representation learning. In *The International Conference on Neural Information Processing Systems (NeurPS'20)*, Vol. 33. Curran Associates, Inc., 4465–4478.

[145] Lichao Sun, Yingtong Dou, Carl Yang, Ji Wang, Philip S. Yu, and Bo Li. 2020. Adversarial attack and defense on graph data: A survey. *arXiv* (2020). https://arxiv.org/abs/1812.10528.

[146] Lin Qiu, Yunxuan Xiao, Yanru Qu, Hao Zhou, Lei Li, Weinan Zhang, and Yong Yu. 2019. Dynamically fused graph network for multi-hop reasoning. In *The Annual Meeting of the Association for Computational Linguistics (ACL'19)*. ACL, 6140–6150.

[147] Linfeng Liu and Liping Liu. 2019. Amortized variational inference with graph convolutional networks for gaussian processes. In *The International Conference on Artificial Intelligence and Statistics (AISTATS'19)*, Vol. 89. Society for Artificial Intelligence and Statistics, 2291–2300.

[148] Lingxiao Ma, Zhi Yang, Youshan Miao, Jilong Xue, Ming Wu, Lidong Zhou, and Yafei Dai. 2018. Towards efficient large-scale graph neural network computing. *arXiv* (2018). https://arxiv.org/abs/1810.08403.

[149] Lingxiao Zhao and Leman Akoglu. 2020. PairNorm: Tackling oversmoothing in GNNs. In *The International Conference on Learning Representations (ICLR'20)*. Addis Ababa.

[150] Linjie Li, Zhe Gan, Yu Cheng, and Jingjing Liu. 2019. Relation-aware graph attention network for visual question answering. In *The International Conference on Computer Vision (ICCV'19)*. IEEE, 10313–10322.

[151] Loic Landrieu and Mohamed Boussaha. 2019. Point Cloud over Segmentation with graph-structured deep metric learning. In *The International Conference on Computer Vision and Pattern Recognition (CVPR'19)*. IEEE, 7440–7449.

[152] Louis Tiao, Pantelis Elinas, Harrison Nguyen, and Edwin V. Bonilla. 2019. Variational spectral graph convolutional networks. *arXiv* (2019). https://arxiv.org/abs/1906.01852v1.

[153] Luana Ruiz, Fernando Gama, and Alejandro Ribeiro. 2019. Gated graph convolutional recurrent neural networks. In *The European Signal Processing Conference (EUSIPCO'19)*. IEEE, 1–5. https://doi.org/10.23919/EUSIPCO.2019.8902995

[154] Marcelo Prates, Pedro H. C. Avelar, Henrique Lemos, Luis C. Lamb, and Moshe Y. Vardi. 2019. Learing to solve NP-Complete problems: A graph neural network for decision TSP. In *The Conference on Artificial Intelligence (AAAI'19)*. Association for the Advances of Artificial Intelligence, 4731–4738.

[155] Marnka Zitnik, Monica Agrawal, and Jure Leskovec. 2018. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics* 34, 13 (2018), i457–i466. https://doi.org/10.1093/bioinformatics/bty294

[156] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *The Conference on Computer Vision and Pattern Recognition (CVPR'17)*. IEEE, 29–38. https://doi.org/10.1109/CVPR.2017.11

[157] Martin Simonovsky and Nikos Komodakis. 2018. GraphVAE: Towards generation of small graphs using variational autoencoders. In *The International Conference on Artificial Neural Networks (ICANN'18)*, Vol. 11139. 412–422. https://doi.org/10.1007/978-3-030-01418-6_41

[158] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *The International Conference on Machine Learning (ICLR'16)*, Vol. 48. 2014–2023.

[159] Matthias Fey and Jan E. Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. In *The International Conference on Learning Representations (ICLR'19)*.

[160] Medhini Narasimhan, Svetlana Lazebnik, and Alexander G. Schwing. 2018. Out of the box: Reasoning with graph convolution nets for factual visual question answering. In *The International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates, Inc., 2654–2665.

[161] Meiqi Zhu, Xiao Wang, Chuan Shi, Houye Ji, and Peng Cui. 2021. Interpreting and unifying graph neural networks with an optimization framework. In *The World Wide Web Conference (WWW'21)*, Vol. 80. 1215–1226.

[162] Meng Liu, Hongyang Gao, and Shuiwang Ji. 2020. Towards deeper graph neural networks. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'20)*. 417, 338–348. https://doi.org/10.1145/3394486.3403076

[163] Meng Qu, Yoshua Bengio, and Jian Tang. 2019. GMNN: Graph markov neural networks. In *The International Conference on Machine Learning (ICML'19)*, Vol. 97. PMLR, 5241–5250.

[164] Diego Mesquita, Amauri H. Souza, and Samuel Kaski. 2020. Rethinking pooling in graph neural networks. In *The International Conference Neural Information Processing Systems (NeurPS'20)*, Vol. 33. Curran Associates, Inc., 2220–2231.

[165] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *The International Conference on Neural Information Processing Systems (NIPS'16)*, Curran Associates, Inc. 3844–3852.

[166] Michael M. Bronstein, Joan Bruna, Yan LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.

[167] Mikael Henaff, Joan Bruna, and Yan LeCun. 2015. Deep convolutional networks on graph-structured data. *arXiv* (2015). https://arxiv.org/abs/1506.05163.

[168] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. 2018. Learning to represent programs with graphs. In *The International Conference on Learning Representations (ICLR'18)*.

[169] Ming Ding, Chang Zhou, Qibin Chen, Hongxia Yang, and Jie Tang. 2019. Cognitive graph for multi-hop reading comprehension at scale. In *The Annual Meeting of the Association for Computational Linguistics (ACL'19)*. ACL, 2694–2703.

[170] Ming Tu, Guangtao Wang, Jing Huang, Yun Tang, Xiaodong He, and Bowen Zhou. 2019. Multi-hop reading comprehension across multiple documents by reasoning over heterogeneous graphs. In *The Annual Meeting of the Association for Computational Linguistics (ACL'19)*. ACL, 2704–2713.

[171] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin Lin, Junbo Zhao, Jinyang Li, Alexander Smola, and Zheng Zhang. 2019. Deep graph library: Towards efficient and scalable deep learning on graphs. In *The International Conference on Learning Representations (ICLR Workshop'19)*.

[172] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *The International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates, Inc., 5171–5181.

[173] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *The Conference on Artificial Intelligence (AAAI'18)*. Association for the Advances of Artificial Intelligence, 4438–4445.

[174] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. 2019. HyperGCN: A new method for training graph convolutional networks on hypergraphs. In *The International Conference on Neural Information Processing Systems (NeurPS'19)*. Curran Associates, Inc., 1511–1522.

[175] Naganand Yadati, Tingran Gao, Shahab Asoodeh, Partha Talukdar, and Anand Louis. 2021. Graph neural networks for soft semi-supervised learning on hypergraphs. In *The Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'21)*. Springer, 447–458. https://doi.org/10.1007/978-3-030-75762-5_36

[176] Nicholas Watters, Andrea Tacchetti, Théophane Weber, Razvan Pascanu, Peter Battaglia, and Daniel Zoran. 2017. Visual interaction networks: Learning a physics simulator from video. In *The International Conference on Neural Inforamtion Processing Systems (NIPS'17)*. Curran Associates, Inc., 4539–4547.

[177] Nicola De Cao and Thomas N. Kipf. 2018. MolGAN: An implicit generative model for small molecular graphs. *arXiv* (2018). https://arxiv.org/abs/1805.11973.

[178] Nicola De Cao, Wilker Aziz, and Ivan Titov. 2019. Question answering by reasoning across documents with graph convolutional networks. In *The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'19)*. ACL, 2306–2317.

[179] Nicolas Keriven and Gabriel Peyré. 2019. Universal invariant and equivariant graph neural networks. In *The International Conference on Neural Information Processing Systems (NeurPS'19)*. Curran Associates, Inc., 7092–7101.

[180] Nicoló Navarin, Dinh V. Tran, and Alessandro Sperduti. 2018. Pre-training graph neural networks with kernels. *arXiv* (2018). https://arxiv.org/abs/1811.06930v1.

[181] Nima Dehmamy, Albert-Laszlo Barabasi, and Rose Yu. 2019. Understanding the representation power of graph neural networks in learning graph topology. In *The International Conference on Neural Information Processing Systems (NeurPS'19)*. Curran Associates, Inc., 15413–15423.

[182] Ningyu Zhang, Shuming Deng, Zhanlin Sun, Guanying Wang, Xi Chen, Wei Zhang, and Huajun Chen. 2019. Long-tail relation extraction via knowledge graph embeddings and graph convolution networks. In *The Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'19)*. ACL, 3016–3025.

[183] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional networks for biomedical image segmentation. In *The International Conference on Medical Image Computing and Computer-assisted Intervention (MCCAI'15)*. 234–241.

[184] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Pitfalls of graph neural network evaluation. *arXiv* (2019). https://arxiv.org/abs/1811.05868.

[185] Oytun Ulutan, A. S. M. Iftekhar, and B. S. Manjunath. 2020. VSGNet: Spatial attention network for detecting human object interactions using graph convolutions. In *The International Conference on Computer Vision and Pattern Recognition (CVPR'20)*. IEEE.

[186] Pablo Barceló, Egor V. Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan Pablo Silva. 2020. The logical expressiveness of graph neural networks. In *The International Conference on Learning Representations (ICLR'20)*. Addis Ababa.

[187] Padraig Corcoran. 2019. Function space pooling for graph convolutional networks. *arXiv* (2019). https://arxiv.org/abs/1905.06259.

[188] Pak K. Chan, M. D. F. Schlag, and Z. Y. Zien. 1994. Spectral K-way ratio-cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13, 9 (1994), 1088–1096. https://doi.org/10.1109/43.310898

[189] Paul Almasan, José Suárez-Varela, Arnau Badia-Sampera, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2019. Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case. *arXiv* (2019). https://arxiv.org/abs/1910.07421.

[190] Peng Han, Peng Yang, Peilin Zhao, Shuo Shang, Yong Liu, Jiayu Zhou, Xin Gao, and Panos Kalnis. 2019. GCN-MF: Disease-gene association identification by graph convolutional networks and matrix factorization. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'19)*. ACM, 705–713.

[191] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *The International Conference on Learning Representations (ICLR'18)*.

[192] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep graph infomax. In *The International Conference on Learning Representations (ICLR'19)*.

[193] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. 2016. Interaction networks for learning about objects, relations and physics. In *The International Conference on Neural Information Processing Systems (NIPS'16)*. Curran Associates, Inc., 4502–4510.

[194] Peter Meltzer, Marcelo Daniel Gutierrez Mallea, and Peter J. Bentley. 2019. PiNet: A permutation invariant graph neural network for graph classification. *arXiv* (2019). https://arxiv.org/abs/1905.03046.

[195] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv* (2018). https://arxiv.org/abs/1806.01261.

[196] Phillip E. Pope, Soheil Kolouri, Mohammad Rostami, Charles E. Martin, and Heiko Hoffmann. 2019. Explainability methods for graph convolutional networks. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'19)*. IEEE, 10772–10781.

[197] Qimai Li, Zhichao Han, and Xiaoming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *The Conference on Artificial Intelligence (AAAI'18)*. Association for the Advances of Artificial Intelligence, 3538–3545.

[198] Radford M. Neal and Geoffrey E. Hinton. 1998. A view of the EM algorithm that justifies incremental, sparse and other variants. *Learning in Graphical Models* 89 (1998), 355–368. https://doi.org/10.1007/978-94-011-5014-9_12

[199] Renchun You, Zhiyao Guo, Lei Cui, Xiang Long, Yingze Bao, and Shilei Wen. 2020. Cross-modality attention with semantic graph embedding for multi-label classification. In *The Conference on Artificial Intelligence (AAAI'20)*. Association for the Advances of Artificial Intelligence.

[200] Renjie Liao, Marc Brockschmidt, Daniel Tarlow, Alexander L. Gaunt, Raquel Urtasun, and Richard Zemel. 2018. Graph partition neural networks for semi-supervised classification. In *The International Conference on Learning Representations Workshop*.

[201] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard Zemel. 2019. LanczosNet: Multi-scale deep graph convolutional networks. In *The International Conference on Learning Representations (ICLR'19)*.

[202] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *The International Conference on Neural Information Processing Systems (NeurPS'18)*. Curran Associates, Inc., 4805–4815.

[203] Richard C. Wilson, Edwin R. Hancock, Elżbieta Pekalska, and Robert P. W. Duin. 2014. Spherical and hyperbolic embeddings of data. *IEEE Transactions on Pattern Recognition and Machine Intelligence* 36, 11 (2014), 2255–2269. https://doi.org/10.1109/TPAMI.2014.2316836

[204] Rik Koncel-Kedziorski, Dhanush Bekal, Yi Luan, Mirella Lapata, and Hannaneh Hajishirzi. 2019. Text generation from knowledge graphs with graph transformers. In *The Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'19)*, Vol. 1. ACL, 2284–2293. https://doi.org/10.18653/v1/N19-1238

[205] Risi Kondor, Truong Son Hy, Horace Pan, Brandon M. Anderson, and Shubhendu Trivedi. 2018. Covariant compositional networks for learning graphs. In *The International Conference on Learning Representations Workshop*.

[206] Ron Levie, Federico Monti, Xavier Bresson, and Michael M. Bronstein. 2019. CayleyNets: Graph convolutional neural networks with complex relational spectral filters. *IEEE Transactions on Signal Processing* 67, 1 (2019), 97–109.

[207] Rui Dai, Shenkun Xu, Qian Gu, Chenguang Ji, and Kaikui Liu. 2020. Hybrid spatio-temporal graph convolutional networks: Improving traffic prediction with navigation data. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'20)*. ACM, 3074–3082. https://doi.org/10.1145/3394486.3403358

[208] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. 2018. Adaptive graph convolutional neural networks. In *The Conference on Artificial Intelligence (AAAI'18)*. Associations for the Advances of Artificial Intelligence, 3546–3553. https://doi.org/10.1145/3292500.3330925

[209] Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. 2019. Relational pooling for graph representations. In *The International Conference on Machine Learning (ICML'19)*, Vol. 97. PMLR, 4663–4673.

[210] Ryoma Sato. 2020. A survey on the expressive power of graph neural networks. *arXiv* (2020). https://arxiv.org/abs/2003.04078.

[211] Ryoma Sato, Makoto Yamada, and Hisashi Kashima. 2019. Approximation ratios of graph neural networks for combinatorial problems. In *The International Conference on Neural Information Processing Systems (NeurPS'19)*. Curran Associates, Inc., 4081–4090.

[212] Sami Abu-EL-Haija, Amol Kapoor, Bryan Perozzi, and Joonseok Lee. 2019. N-GCN: Multi-scale graph convolution for semi-supervised node classification. In *The Conference on Uncertainty in Artificial Intelligence (UAI'19)*. PMLR, No. 310.

[213] Sami Abu-EL-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. MixHop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *The International Conference on Machine Learning (ICML'19)*, Vol. 97. PMLR, 21–29.

[214] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. 2017. Dynamic routing between capsules. In *The International Conference on Neural Information Processing Systems (NIPS'17)*, Vol. 30. Curran Associates, Inc., 3856–3866.

[215] Saurabh Verma and Zhili Zhang. 2018. Graph capsule convolutional neural networks. *arXiv* (2018). https://arxiv.org/abs/1805.08090.

[216] Ś{e}bastien Lerique, Jacob Levy Abitol, and Márton Karsai. 2020. Joint embedding of structure and features via graph convolutional networks. *Applied Network Science* 5, 2020 (2020), No. 5. https://doi.org/10.1007/s41109-019-0237-x

[217] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[218] Shaohua Fan, Junxiong Zhu, Xiaotian Han, Chuan Shi, Linmei Hu, Biyu Ma, and Yongliang Li. 2019. Metapath-guided heterogeneous graph neural network for intent recommendation. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'19)*. ACM, 2478–2486. https://doi.org/10.1145/3292500.3330673

[219] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep neural networks for learning graph representations. In *The Conference on Artificial Intelligence (AAAI'16)*. Association for the Advances of Artificial Intelligence, 1145–1152.

[220] Shengding Hu, Meng Qu, Zhiyuan Liu, and Jian Tang. 2020. Transfer active learning for graph neural networks. *OpenReview* (2020). https://openreview.net/forum?id=BklOXeBFDS.

[221] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. 2019. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *The International Conference on Artificial Intelligence (AAAI'19)*. Association for the Advances of Artificial Intelligence, 922–929. https://doi.org/10.1609/aaai.v33i01.3301922

[222] Shichang Zhang, Ziniu Hu, Arjun Subramonian, and Yizhou Sun. 2021. Motif-driven contrastive learning of graph representations. *arXiv* (2021). https://arxiv.org/abs/2012.12533v2.

[223] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. 2018. Adversarially regularized graph autoencoder for graph embedding. In *The International Joint Conference on Artificial Intelligence (IJCAI'18)*. IJCAI Organization, 2609–2615.

[224] Shizhe Chen, Qin Jin, Peng Wang, and Qi Wu. 2020. Say as you wish: Fine-grained control of image caption generation with abstract scene graphs. In *The International Conference on Computer Vision and Pattern Recognition (CVPR'20)*. IEEE.

[225] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. 2019. Session-based recommendation with graph neural networks. In *The Conference on Artificial Intelligence (AAAI'19)*. Association for the Advances of Artificial Intelligence, 346–353.

[226] Sijie Yan, Yuanjun Xiong, and Dahua Lin. 2018. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *The Conference on Artificial Intelligence (AAAI'18)*. Association for the Advances of Artificial Intelligence, 7444–7452.

[227] Sofia Ira Ktena, Sarah Parisot, Enzo Ferrante, Martin Rajchl, Matthew Lee, Ben Glocker, and Daniel Ruekert. 2017. Distance metric learning using graph convolutional networks: Application to functional brain networks. In *The International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI'17)*. 469–477.

[228] Songtao He, Favyen Bastani, Satvat Jagwani, Edward Park, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Samuel Madden, and Mohammad Amin Sadeghi. 2020. RoadTagger: Robust road attribute inference with graph neural networks. In *The Conference on Artificial Intelligence (AAAI'20)*. Association for the Advances of Artificial Intelligence.

[229] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay S. Pande, and Patrick F. Riley. 2016. Molecular graph convolutions: Moving beyond fingerprints. *Journal of Computer-Aided Molecular Design* 30, 8 (2016), 595–608. https://doi.org/10.1007/s10822-016-9938-8

[230] Sungmin Rhee, Seokjun Seo, and Sun Kim. 2018. Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. In *The International Joint Conference on Artificial Intelligence (IJCAI'18)*. IJCAI Organization, 3527–3534. https://doi.org/10.24963/ijcai.2018/490

[231] Tao Gui, Yicheng Zou, Qi Zhang, Minlong Peng, Jinlan Fu, Zhongyu Wei, and Xuanjing Huang. 2019. A lexicon-based graph neural network for Chinese NER. In *The Conference on Empirical Methods in Natural Language Processing (EMNLP'19)*. ACL, 1040–1050.

[232] Tengfei Ma, Junyuan Shang, and Jimeng Sun. 2019. CGNF: Conditional graph neural fields. *OpenReview* (2019). https://openreview.net/forum?id=ryxMX2R9YQ.

[233] Thien Huu Nguyen and Ralph Grishman. 2018. Graph convolutional networks with argument-aware pooling for event detection. In *The Conference on Artificial Intelligence (AAAI'18)*. Association for the Advances of Artificial Intelligence, 5900–5907.

[234] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *Journal of Machine Learning Research* 20 (2019), 1–21. https://doi.org/10.1002/j.1538-7305.1970.tb01770.x

[235] Thomas N. Kipf, Elise van der Pol, and Max Welling. 2020. Contrastive learning of structured world models. In *The International Conference on Learning Representations (ICLR'20)*. Addis Ababa.

[236] Thomas N. Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv* (2016). https://arxiv.org/abs/1611.07308.

[237] Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *The International Conference on Learning Representations (ICLR'17)*.

[238] Tinghuai Wang, Guangming Wang, Kuan Eeik Tan, and Donghui Tan. 2020. Spectral pyramid graph attention network for hyperspectral image classification. *arXiv* (2020). https://arxiv.org/abs/2001.07108.

[239] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. 2018. NerveNet: Learning structured policy with graph neural networks. In *The International Conference on Learning Representations (ICLR'18)*.

[240] Trang Pham, Truyen Tran, Dinh Phung, and Svetha Venkatesh. 2017. Column networks for collective classification. In *The Conference on Artificial Intelligence (AAAI'17)*. Association for the Advances of Artificial Intelligence, 2485–2491.

[241] Tsu-Jui Fu, Peng-Hsuan Li, and Wei-Yun Ma. 2019. GraphRel: Modeling text as relational graphs for joint entity and relation extraction. In *The Annual Meeting of the Association for Computational Linguistics (ACL'19)*. ACL, 1409–1418.

[242] Tyler Derr, Yao Ma, and Jiliang Tang. 2018. Signed graph convolutional network. In *The International Conference on Data Mining (ICDM'18)*. IEEE, 929–934. https://doi.org/10.1109/ICDM.2018.00113

[243] Uri Alon and Eran Yahav. 2020. On the bottleneck of graph neural networks and its practical implications. *arXiv* (2020). https://arxiv.org/abs/2006.05205v2.

[244] Victoria Zayats and Mari Ostendorf. 2018. Conversation modeling on Reddit using a graph-structured LSTM. *Transactions of the Association for Computational Linguistics* 6 (2018), 121–132. https://doi.org/10.1162/tacl_a_00009

[245] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. 2020. Benchmarking graph neural networks. *arXiv* (2020). https://arxiv.org/abs/2003.00982.

[246] Juho Kannala and Jian Tang. 2019. GraphMix: Regularized training of graph neural networks for semi-supervised learning. arXiv(2020), https://arxiv.org/abs/1909.11715v1.
[247] Vincenzo Di Massa, Cabriele Monfardini, Lorenzo Sarti, Franco Scarselli, Marco Maggini, and Marco Gori. 2006. A comparison between recursive neural networks and graph neural networks. In *The International Joint Conference on Neural Networks (IJCNN'06)*. IEEE, 778–785. https://doi.org/10.1109/IJCNN.2006.246763
[248] Volodymyr Minh, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. 2014. Recurrent models of visual attention. In *The International Conference on Neural Information Processing Systems (NIPS'14)*, Vol. 2. Curran Associates, Inc., 2204–2212.
[249] Wei Li, Jingjing Xu, Yancheng He, Shengli Yan, Yunfang Wu, and Xu Sun. 2019. Coherent comments generation for Chinese articles with a graph-to-sequence model. In *The Annual Meeting of the Association for Computational Linguistics (ACL'19)*. ACL, Florence, Italy, 4843–4852.
[250] Wei Liu and Sanjay Chawla. 2009. A game theoretical model for adversarial learning. In *The International Conference on Data Mining Workshops (ICDM Workshop'09)*. IEEE, 25–30.
[251] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay S. Pande, and Jure Leskovec. 2020. Strategies for pre-training graph neural networks. In *The International Conference on Learning Representations (ICLR'20)*. Addis Ababa.
[252] Weilin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'19)*. ACM, 257–266.
[253] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive sampling towards fast graph representation learning. In *The International Conference on Neural Information Processing Systems (NeurPS'18)*. Curran Associates, Inc., 4563–4572.
[254] Wenchao Yu, Cheng Zheng, and Wei Cheng. 2018. Learning deep network representations with adversarially regularized autoencoders. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'18)*. ACM, 2663–2671. https://doi.org/10.1145/3219819.3220000
[255] Wengong Jin, Regina Barzilay, and Tommi Jaakkola. 2018. Junction tree variational autoencoder for molecular graph generation. In *The International Conference on Machine Learning (ICML'18)*, Vol. 80. PMLR, 2323–2332.
[256] Wenwu Zhu, Xin Wang, and Peng Cui. 2020. Deep learning for learning graph representations. *Deep Learning: Concepts and Architectures* 866 (2020), 169–210. https://doi.org/10.1007/978-3-030-31756-0_6
[257] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *The International Conference on Neural Information Processing Systems (NIPS'17)*. Curran Associates, Inc., 1024–1034.
[258] Wouter Kool, Herke van Hoof, and Max Welling. 2019. Attention, learn to solve routing problems. In *The International Conference on Learning Representations (ICLR'19)*. https://openreview.net/forum?id=ByxBFsRqYm.
[259] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. KGAT: Knowledge graph attention network for recommendation. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'19)*. ACM, 950–958.
[260] Xiao Huang, Qingquan Song, Yuening Li, and Xia Hu. 2019. Graph recurrent networks with attributed random walks. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'19)*. ACM, 732–740. https://doi.org/10.1145/3292500.3330941
[261] Xiao Liu, Zhunchen Luo, and Heyan Huang. 2018. Jointly multiple events extraction via attention-based graph information aggregation. In *The Conference on Empirical Methods in Natural Language Processing (EMNLP'18)*. ACL, 1247–1256.
[262] Xiao Shen and Fulai Chung. 2020. Deep network embedding for graph representation learning in signed networks. *IEEE Transactions on Cybernetics* 50, 4 (2020), 1556–1568. https://doi.org/10.1109/TCYB.2018.2871503
[263] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Peng Cui, Philip S. Yu, and Yanfang Ye. 2018. Heterogeneous graph attention network. In *The World Wide Web Conference (WWW'18)*. 2022–2032. https://doi.org/10.1145/3308558.3313562
[264] Xiao Wang, Ruijia Wang, Chuan Shi, Guojie Song, and Qingyong Li. 2020. Multi-component graph convolutional collaborative filtering. In *The Conference on Artificial Intelligence (AAAI'20)*. Association for the Advances of Artificial Intelligence.
[265] Xiaodan Liang, Liang Lin, Xiaohui Shen, Jiashi Feng, Shuicheng Yan, and Eric P. Xing. 2017. Interpretable structure-evolving LSTM. In *The International Conference on Computer Vision and Pattern Recognition (CVPR'17)*. IEEE, 1010–1019.
[266] Xiaodan Liang, Xiaohui Shen, Jiashi Feng, Liang Lin, and Shuicheng Yan. 2016. Semantic object parsing with graph LSTM. In *The European Conference on Computer Vision (ECCV'16)*. Springer, 125–143.
[267] Xiaojie Guo, Lingfei Wu, and Liang Zhao. 2018. Deep graph translation. *arXiv* (2018). https://arxiv.org/abs/1805.09980.

[268] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. 2018. Non-local neural networks. In *The International Conference on Computer Vision and Pattern Recognition (CVPR'18)*. IEEE, 7794–7803.

[269] Xiaoran Xu, Wei Feng, Yunsheng Jiang, Xiaohui Xie, Zhiqing Sun, and Zhihong Zhang. 2020. Dynamically pruned message passing networks for large-scale knowledge graph reasoning. In *The International Conference on Learning Representations (ICLR'20)*. Addis Ababa.

[270] Xien Liu, Xinxin You, Xiao Zhang, Ji Wu, and Ping Lv. 2020. Tensor graph convolutional networks for text classification. *arXiv* (2020). https://arxiv.org/abs/2001.05313.

[271] Xin Jiang, Kewei Cheng, Song Jiang, and Yizhou Sun. 2019. Chordal-GCN: Exploiting sparsity in training large-scale graph convolutional networks. *OpenReview* (2019). https://openreview.net/forum?id=rJl05AVtwB.

[272] Xinyi Zhang and Lihui Chen. 2019. Capsule graph neural network. In *The International Conference on Learning Representations (ICLR'19)*.

[273] Xipeng Qiu, TianXiang Sun, YiGe Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences* 63 (2020), 1872–1897. https://doi.org/10.1007/s11431-020-1647-3

[274] Carl Yang, Peiye Zhuang, Wenhan Shi, Alan Luu, and Pan Li. 2019. Conditional structure generation through graph variational generative adversarial nets. In *The International Conference on Neural Information Processing Systems (NeurPS'19)*, Vol. 32. Curran Associates, Inc.

[275] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 2009. Gradient-based learning applied to document recognition. *The IEEE* 86, 11 (2009), 22778–2324. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[276] Yanqiao Zhu, Yichen Xu, Feng Yu, Shu Wu, and Liang Wang. 2020. CAGNN: Cluster-aware graph neural networks for unsupervised graph representation learning. *arXiv* (2020). https://arxiv.org/abs/2009.01674.

[277] Yao Ma, Suhang Wang, Charu C. Aggarwal, Dawei Yin, and Jiliang Tang. 2019. Multi-dimensional graph convolutional networks. In *The SIAM International Conference on Data Mining (SDM'19)*. SIAM, 657–665. https://doi.org/10.1137/1.9781611975673.74

[278] Yao Ma, Suhang Wang, Charu C. Aggarwal, and Jiliang Tang. 2019. Graph convolutional networks with EigenPooling. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'19)*. ACM, 723–731. https://doi.org/10.1145/3292500.3330982

[279] Yao Mao, Ziyi Guo, Zhaochun Ren, Eric Zhao, Jiliang Tang, and Dawei Yin. 2020. Streaming graph neural networks. In *The International Conference on Research and Development in Information Retrieval (SIGIR'20)*. ACM, Virtual Event, 719–728. https://doi.org/10.1145/3397271.3401092

[280] Yaochen Xie, Zhao Xu, Zhengyang Wang, and Shuiwang Ji. 2021. Self-supervised learning of graph neural networks: A unified review. *arXiv* (2021). https://arxiv.org/abs/2102.10757v2.

[281] Yawei Luo, Tao Guan, Junqing Yu, Ping Liu, and Yi Yang. 2020. Every node counts: Self-ensembling graph convolutional networks for semi-supervised learning. *Pattern Recognition* 106 (2020), No. 107451. https://doi.org/10.1016/j.patcog.2020.107451

[282] Yedid Hoshen. 2017. VAIN: Attentional multi-agent predictive modeling. In *The International Conference on Neural Information Processing Systems (NIPS'17)*. Curran Associates, Inc., 2698–2708.

[283] Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. 2018. M-Walk: Learning to walk over graphs using monte carlo tree search. In *The International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates, Inc., 6786–6797.

[284] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. 2019. Hypergraph neural networks. In *The Conference on Artificial Intelligence (AAAI'19)*. Association for the Advances of Artificial Intelligence, 3558–3565. https://doi.org/10.1609/aaai.v33i01.33013558

[285] Yihe Dong, Will Sawin, and Yoshua Bengio. 2020. HNHN: Hypergraph networks with hyperedge neurons. *arXiv* (2020). https://arxiv.org/abs/2006.12278.

[286] Yikang Li, Wanli Quyang, Bolei Zhou, Jianping Shi, Chao Zhang, and Xiaogang Wang. 2018. Factorizable Net: An efficient subgraph-based framework for scene graph generation. In *The European Conference on Computer Vision (ECCV'18)*. Springer, Munich, Germany.

[287] Yinwei Wei, Xiang Wang, Liqiang Nie, Xiangnan He, Richang Hong, and Tat-Seng Chua. 2019. MMGCN: Multimodal graph convolution network for personalized recommendation of micro-video. In *The International Conference on Multimedia (MM'19)*. ACM, 1437–1445.

[288] Yiqing Xie, Sha Li, Carl Yang, Raymond Chi-Wing Wong, and Jiawei Han. 2020. When do GNNs work: Understanding and improving neighborhood aggregation. In *The International Joint Conference on Artificial Intelligence (IJCAI'20)*. IJCAI Organization, 1303–1309. https://doi.org/10.24963/ijcai.2020/181

[289] Yixin Liu, Shirui Pan, Ming Jin, Chuan Zhou, Feng Xia, and Philip S. Yu. 2021. Graph self-supervised learning: A survey. *arXiv* (2021). https://arxiv.org/abs/2103.00111v1.

[290] Yizhou Zhang, Yun Xiong, Xiangnan Kong, Shanshan Li, Jinhong Mi, and Yangyong Zhu. 2018. Deep collective classification in heterogeneous information networks. In *The World Wide Web Conference (WWW'18)*. 399–408. https://doi.org/10.1145/3178876.3186106

[291] Yongfei Liu, Bo Wan, Xiaodan Zhu, and Xuming He. 2020. Learning cross-modal context graph for visual grounding. In *The Conference on Artificial Intelligence (AAAI'20)*. Association for the Advances of Artificial Intelligence.

[292] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. In *The International Conference on Neural Information Processing Systems (NeurPS'20)*, Vol. 33. Curran Associates, Inc., 5812–5823.

[293] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. 2018. Structured sequence modeling with graph convolutional recurrent networks. In *The International Conference on Neural Information Processing (ICONIP'18)*, Vol. 11301. 362–373. https://doi.org/10.1007/978-3-030-04167-0_33

[294] Yu Cao, Meng Fang, and Dacheng Tao. 2019. BAG: Bi-directional attention entity graph convolutional network for multi-hop reasoning question answering. In *The Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'19)*. ACL, 357–362.

[295] Yu Chen, Lingfei Wu, and Mohammed J. Zaki. 2020. Reinforcement learning based graph-to-sequence model for natural question generation. In *The International Conference on Learning Representations (ICLR'20)*. Addis Ababa.

[296] Yu Jin and Joseph F. JaJa. 2018. Learning graph-level representations with recurrent neural networks. *arXiv* (2018). https://arxiv.org/abs/1805.07683.

[297] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. DropEdge: Towards deep graph convolutional networks on node classification. In *The International Conference on Learning Representations (ICLR'20)*. Addis Ababa.

[298] Yu Zhou, Jianbin Huang, Heli Sun, Yizhou Sun, Shaojie Qiao, and Stephen Wambura. 2019. Recurrent meta-structure for robust similarity measure in heterogeneous information networks. *ACM Transactions on Knowledge Discovery from Data* 13, 6 (2019), No. 64. https://doi.org/10.1145/3364226

[299] Yuan Li, Xiaodan Liang, Zhiting Hu, Yinbo Chen, and Eric P. Xing. 2019. Graph transformer. *OpenReview* (2019). https://openreview.net/forum?id=HJei-2RcK7.

[300] Yuanfu Lu, Xunqiang Jiang, Yuan Fang, and Chuan Shi. 2021. Learning to pre-train graph neural networks. In *The International Conference on Artificial Intelligence (AAAI'21)*. Association for the Advances of Artificial Intelligence.

[301] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, and Michael M. Bronstein. 2019. Dynamic graph CNN for learning on point clouds. *ACM Transactions on Graphics* 38, 5 (2019), No. 146. https://doi.org/10.1145/3326362

[302] Yue Zhang, Qi Liu, and Linfeng Song. 2018. Sentence-State LSTM for text representation. In *The Annual Meeting of the Association for Computational Linguistics (ACL'18)*, Vol. 1. ACL, 317–327. https://doi.org/10.18653/v1/P18-1030

[303] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated graph sequence neural networks. In *The International Conference on Learning Representations (ICLR'16)*. Caribe Hilton.

[304] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. 2018. Learning deep generative models of graphs. In *The International Conference on Learning Representations Workshop*.

[305] Yujun Cai, Liuhao Ge, Jun Liu, Jianfei Cai, Tat-Jen Cham, Junsong Yuan, and Nadia Magnenat Thalmann. 2019. Exploiting spatial-temporal relationships for 3d pose estimation via graph convolutional networks. In *The International Conference on Computer Vision (ICCV'19)*. IEEE, 2272–2281.

[306] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. In *The International Conference on Neural Information Processing Systems (NeurPS'19)*, Vol. 32. Curran Associates, Inc., 1–11.

[307] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. 2019. SimGNN: A neural network approach to fast graph similarity computation. In *The International Conference on Web Search and Data Mining (WSDM'19)*. ACM, 384–392.

[308] Yuyu Zhang, Xinshi Chen, Yuan Yang, Arun Ramamurthy, Bo Li, Yuan Qi, and Le Song. 2020. Efficient probabilistic logic reasoning with graph neural networks. In *The International Conference on Learning Representations (ICLR'20)*. Addis Ababa.

[309] Yuzhou Chen, Yulia R. Gel, and Konstanin Avrachenkov. 2020. Fractional graph convolutional networks (FGCN) for semi-supervised learning. (2020). https://openreview.net/forum?id=BygacxrFwS.

[310] Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. 2019. D-VAE: A variational autoencoder for directed acyclic graphs. In *The International Conference on Neural Information Processing Systems (NeurPS'19)*, Vol. 32. Curran Associates, Inc.

[311] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. 2020. Graph representation learning via graphical mutual information maximization. In *The World Wide Web Conference (WWW'20)*. 259–270.

[312] Zhen Zhang, Hongxia Yang, Jiajun Bu, Sheng Zhou, Pinggang Yu, Jianwei Zhang, Martin Ester, and Can Wang. 2018. ANRL: Attributed network representation learning via deep neural networks. In *The International Joint Conference on Artificial Intelligence (IJCAI'18)*. IJCAI Organization, 3155–3161. https://doi.org/10.24963/ijcai.2018/438

[313] Zhihong Zhang, Dongdong Chen, Jianjia Wang, Lu Bai, and Edwin R. Hancock. 2019. Quantum-based subgraph convolutional neural networks. *Pattern Recognition* 88, 2019 (2019), 38–49. https://doi.org/10.1016/j.patcog.2018.11.002
[314] Zhihong Zhang, Dongdong Chen, Zeli Wang, Heng Li, Lu Bai, and Edwin R. Hancock. 2019. Depth-based subgraph convolutional auto-encoder for network representation learning. *Pattern Recognition* 90 (2019), 363–376. https://doi.org/10.1016/j.patcog.2019.01.045
[315] Zhijiang Guo, Yan Zhang, and Wei Lu. 2019. Attention guided graph convolutional networks for relation extraction. In *The Annual Meeting of the Association for Computational Linguistics (ACL'19)*. ACL, 241–251.
[316] Zhijie Deng, Yinpeng Dong, and Jun Zhu. 2019. Batch virtual adversarial training for graph convolutional networks. In *The International Conference on Machine Learning Workshop*. PMLR.
[317] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. GNNExplainer: Generating explanations for graph neural networks. In *The International Conference on Neural Information Processing Systems (NeurPS'19)*. Curran Associates, Inc., 9244–9255.
[318] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2021. Graph contrastive learning with adaptive augmentation. In *The World Wide Web Conference (WWW'21)*. Ljubljana, 1.
[319] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. 2018. Combinatorial optimization with graph convolutional networks and guided tree search. In *The International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates, Inc., 573–546.
[320] Ziniu Hu, Changjun Fan, Ting Chen, Kai-Wei Chang, and Yizhou Sun. 2019. Pre-training graph neural networks for generic structural feature extraction. *arXiv* (2019). https://arxiv.org/abs/1905.13728.
[321] Ziniu Hu, Changjun Fan, Ting Chen, Kai-Wei Chang, and Yizhou Sun. 2019. Unsupervised pre-training of graph convolutional networks. In *The International Conference on Learning Representations Workshop*. https://arxiv.org/abs/1905.13728.
[322] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020. GPT-GNN: Generative pre-training of graph neural networks. In *The International Conference on Knowledge Discovery and Data Mining (SIGKDD'20)*. ACM, 1157–1167. https://doi.org/10.1145/3394486.3403237
[323] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *The World Wide Web Conference (WWW'20)*. Taipei, 2704–2710. https://doi.org/10.1145/3366423.3380027
[324] Ziqi Liu, Chaochao Chen, Longfei Li, Jun Zhou, Xiaolong Li, Le Song, and Yuan Qi. 2019. GeniePath: Graph neural networks with adaptive receptive paths. In *The Conference on Artificial Intelligence (AAAI'19)*. Association for the Advances of Artificial Intelligence, 4424–4431.
[325] Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2020. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering* 34, 1 (2020), 249–270. https://doi.org/10.1109/TKDE.2020.2981333
[326] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* 32, 1 (2021), 4–24.
[327] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. In *The International Conference on Neural Information Processing Systems (NeurPS'19)*, Vol. 32. Curran Associates, Inc.

Received December 2020; revised August 2021; accepted October 2021