Deep Learning on Graphs: A Survey

Ziwei Zhang, Peng Cui and Wenwu Zhu

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

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

arXiv:1812.04202v1 [cs.LG] 11 Dec 2018

1 Introduction

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

- **Irregular domain**. Unlike images, audio and text which have a clear grid structure, graphs lie in an irregular domain, making it hard to generalize some basic mathematical operations to graphs [6]. For example, it is not straight-forward to define convolution and pooling operation for graph data, which are the fundamental operations in Convolutional Neural Networks (CNNs). This is often referred as the geometric deep learning problem [7].
- **Varying structures and tasks**. Graph itself can be complicated with diverse structures. For example, graphs can be heterogenous or homogenous, weighted or unweighted, and signed or unsigned. In addition, the tasks for graphs also vary greatly, ranging from node-focused problems such as node classification and link prediction, to graph-focused problems such as graph classification and graph generation. The varying structures and tasks require different model architectures to tackle specific problems.
- **Scalability and parallelization**. In the big-data era, real graphs can easily have millions of nodes and edges, such as social networks or e-commerce networks [8]. As a result, how to design scalable models, preferably with a linear time complexity, becomes a key problem. In addition, since nodes and edges in the graph are interconnected and often need to be modeled as a whole, how to conduct parallel computing is another critical issue.
- **Interdiscipline**. Graphs are often connected with other disciplines, such as biology, chemistry or social sciences. The interdiscipline provides both opportunities and challenges: domain knowledge can be leveraged to solve specific problems, but integrating domain knowledge could make designing the model more difficult. For example, in generating molecular graphs, the objective function and chemical constraints are often non-differentiable, so gradient based training methods cannot be easily applied.

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

In this paper, we try to fill this gap by comprehensive reviewing deep learning methods on graphs. Specifically, as shown in Figure 1, we divide the existing methods into three main categories: semi-supervised methods, unsupervised methods and recent advancements. Concretely speaking, semi-supervised methods include Graph Neural Networks (GNNs) and Graph Convolutional Networks (GCNs), unsupervised methods are mainly composed of Graph Autoencoders (GAEs) and recent advancements include Graph Recurrent Neural Networks and Graph Reinforcement Learning. We summarize some main distinctions of these categories in Table 1. Broadly speaking, GNNs and GCNs are semi-supervised as they utilize node attributes and node labels to train model parameters end-to-end for a specific task, while GAEs mainly focus on learning representation using unsupervised methods. Recently advanced methods use other unique algorithms

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

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

that do not fall in previous categories. Besides these high-level distinctions, the model architectures also differ greatly. In the following sections, we will provide a comprehensive overview of these methods in detail, mainly following their history of developments and how these methods solve challenges of graphs. We also analyze the differences of these models and how to composite different architectures. In the end, we briefly outline the applications of these methods and discuss potential future directions.

Related works. There are several surveys that are related to our paper. Bronstein et al. [7] summarize some early GCN methods as well as CNNs on manifolds, and study them comprehensively through geometric deep learning. Recently, Battaglia et al. [9] summarize how to use GNNs and GCNs for relational reasoning using a unified framework called graph networks and Lee et al. [10] review the attention models for graphs. We differ from these works in that we systematically and comprehensively review different deep learning architectures on graphs rather than focusing on one specific branch. Another closely related topic is network embedding, trying to embed nodes into a low-dimensional vector space [11]–[13]. The main distinction between network embedding and our paper is that we focus on how different deep learning models can be applied to graphs, and network embedding can be recognized as a concrete example using some of these models (they use non deep learning methods as well).

The rest of this paper is organized as follows. In Section 2, we introduce notations and preliminaries. Then, we review GNNs, GCNs, GAEs and recent advancements in Section 3 to Section 6 respectively. We conclude with a discussion in Section 7.

2 Notations and Preliminaries

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

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

For a deep learning model, we use superscripts to denote layers, e.g. $\mathbf{H}^l$. We use f_l to denote the number of dimensions in layer l. The sigmoid activation function is defined as $\sigma(x) = 1/(1+e^{-x})$ and rectifier linear unit (ReLU) is defined as $\text{ReLU}(x) = max(0,x)$. A general element-wise non-linear activation function is denoted as $\rho(\cdot)$. In this paper, unless stated otherwise, we assume all functions are differentiable so that we can learn model parameters $\mathbf{\Theta}$ through back-propagation [14] using commonly adopted optimizers, such as Adam [15], and training techniques, such as dropout [16]. We summarize the notations in Table 2.

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

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

Note that such distinction is more conceptually than mathematically rigorous. On the one hand, there exist tasks associated with mesoscopic structures such as community detection [17]. In addition, node-focused problems can sometimes be studied as graph-focused problems by transforming the former into ego-centric networks [18]. Nevertheless, we will detail the distinction between these two categories when necessary.

3 Graph Neural Networks (GNNs)

In this section, we review the most primitive semi-supervised deep learning methods for graph data, Graph Neural Networks (GNNs).

The origin of GNNs can be dated back to the "pre-deep-learning" era [19], [20]. The idea of GNN is simple: to encode structural information of the graph, each node v_i can be represented by a low-dimensional state vector $\mathbf{s}_i, 1 \leq i \leq N$. Motivated by recursive neural networks [21], a recursive definition of states is adopted [20]:

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

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

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

For graph-focused tasks, the authors suggest adding a special node with unique attributes corresponding to the whole graph. To learn model parameters, the following semi-supervised method is adopted: after iteratively solving Eq. (1) to a stable point using Jacobi method [22], one step of gradient descend is performed using the Almeida-Pineda algorithm [23], [24] to minimize a task-specific objective function, for example, the square loss between predicted values and the ground-truth for regression tasks; then, this process is repeated until convergence.

With two simple equations in Eqs. (1)(2), GNN plays two important roles. In retrospect, GNN unifies some early methods in processing graph data, such as recursive neural networks and Markov chains [20]. Looking to the future, the concept in GNN has profound inspirations: as will be shown later, many state-of-the-art GCNs actually have a similar formulation as Eq. (1), following the framework of exchanging information with immediate neighborhoods. In fact, GNNs and GCNs can be unified into a common framework and GNN is equivalent to GCN using

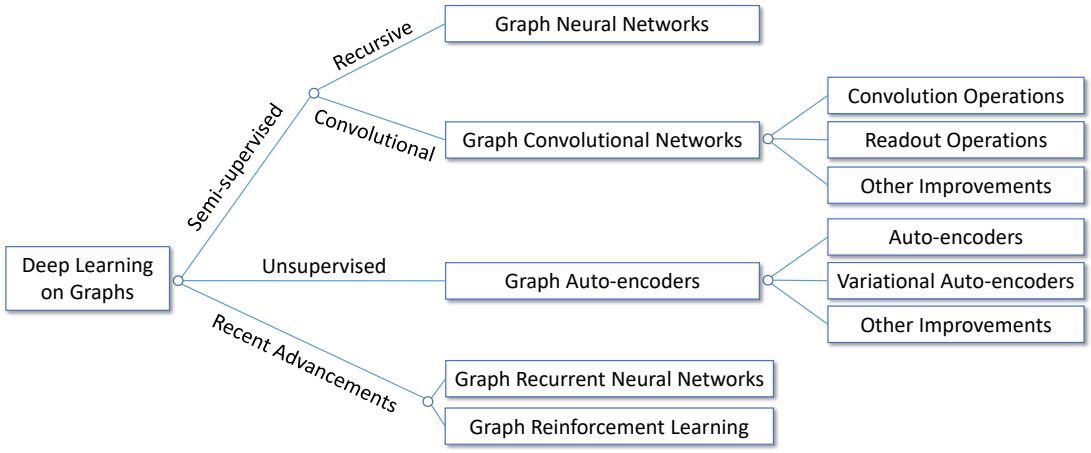

Fig. 1. The categorization of deep learning methods on graphs. We divide existing methods into three categories: semi-supervised, unsupervised and recent advancements. The semi-supervised methods can be further divided into Graph Neural Networks and Graph Convolutional Networks based on their architectures. Recent advancements include Graph Recurrent Neural Networks and Graph Reinforcement Learning methods.

TABLE 1
Some Main Distinctions of Deep Learning Methods on Graphs

Category	Type	Node Attributes/Labels	Counterparts in Traditional Domains
Graph Neural Networks	Semi-supervised	Yes	Recursive Neural Networks
Graph Convolutional Networks	Semi-supervised	Yes	Convolutional Neural Networks
Graph Autoencoders	Unsupervised	Partial	Autoencoders/Variational Autoencoders
Graph Recurrent Neural Networks	Various	Partial	Recurrent Neural Networks
Graph Reinforcement Learning	Semi-supervised	Yes	Reinforcement Learning

TABLE 2
Table of Commonly Used Notations

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

identical layers to reach a stable state. More discussion will be given in Section 4.

Though conceptually important, GNN has several drawbacks. First, to ensure that Eq. (1) has a unique solution, $\mathcal{F}(\cdot)$ has to be a "contraction map" [25], which severely limits the modeling ability. Second, since many iterations are needed between gradient descend steps, GNN is computationally expensive. Because of these drawbacks and perhaps the lack of computational power (e.g. Graphic Processing Unit, GPU, is not widely used for deep learning those days) and lack of research interests, GNN was not widely known to the community.

A notable improvement to GNN is Gated Graph Sequence Neural Networks (GGS-NNs) [26] with several modifications. Most importantly, the authors replace the recursive definition of Eq. (1) with Gated Recurrent Units (GRU) [27], thus remove the requirement of "contraction map" and support the usage of modern optimization techniques. Specifically, Eq. (1) is replaced by:

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

where $\mathbf{z}$ are calculated by update gates, $\widetilde{\mathbf{s}}$ are candidates for updating and t is the pseudo time. Secondly, the authors propose using several such networks operating in sequence to produce a sequence output, which can be applied to applications such as program verification [28].

GNN and its extensions have many applications. For example, CommNet [29] applies GNN to learn multi-agent communication in AI systems by regarding each agent as a node and updating the states of agents by communication with others for several time steps before taking an action. Interaction Network (IN) [30] uses GNN for physical reasoning by representing objects as nodes, relations as edges and using pseudo-time as a simulation system. VAIN [31] improves CommNet and IN by introducing attentions to weigh different interactions. Relation Networks (RNs) [32] propose using GNN as a relational reasoning module to augment other neural networks and show promising results in visual question answering problems.

4 Graph Convolutional Networks (GCNs)

Besides GNNs, Graph Convolutional Networks (GCNs) are another class of semi-supervised methods for graphs. Since GCNs usually can be trained with task-specific loss via back-propagation like standard CNNs, we focus on the architectures adopted. We will first discuss the convolution operations, then move to the readout operations and improvements. We summarize main characteristics of GCNs surveyed in this paper in Table 3.

TABLE 3
Comparison of Different Graph Convolutional Networks (GCNs)

Method	Type	Convolution	Readout	Scalability	Multiple Graphs	Other Improvements
Bruna et al. [33]	Spectral	Interpolation Kernel	Hierarchical Clustering + FC	No	No	-
Henaff et al. [34]	Spectral	Interpolation Kernel	Hierarchical Clustering + FC	No	No	Constructing Graph
ChebNet [35]	Spatial	Polynomial	Hierarchical Clustering	Yes	Yes	-
Kipf&Welling [36]	Spatial	First-order	-	Yes	-	Residual Connection
Neural FPs [37]	Spatial	First-order	Sum	No	Yes	-
PATCHY-SAN [38]	Spatial	Polynomial + Order	Order + Pooling	Yes	Yes	Order for Nodes
DCNN [39]	Spatial	Polynomial Diffusion	Mean	No	Yes	Edge Features
DGCN [40]	Spatial	First-order + Diffusion	-	No	-	-
MPNNs [41]	Spatial	First-order	Set2set	No	Yes	General Framework
GraphSAGE [42]	Spatial	First-order + Sampling	-	Yes	-	General Framework
MoNet [43]	Spatial	First-order	Hierarchical Clustering	Yes	Yes	General Framework
GNs [9]	Spatial	First-order	Whole Graph Representation	Yes	Yes	General Framework
DiffPool [44]	Spatial	Various	Hierarchical Clustering	No	Yes	Differentiable Pooling
GATs [45]	Spatial	First-order	-	Yes	Yes	Attention
CLN [46]	Spatial	First-order	-	Yes	-	Residual Connection
JK-Nets [47]	Spatial	Various	-	Yes	Yes	Jumping Connection
ECC [48]	Spatial	First-order	Hierarchical Clustering	Yes	Yes	Edge Features
R-GCNs [49]	Spatial	First-order	-	Yes	-	Edge Features
Kearnes et al. [50]	Spatial	Weave module	Fuzzy Histogram	Yes	Yes	Edge Features
PinSage [51]	Spatial	Random Walk	-	Yes	-	-
FastGCN [52]	Spatial	First-order + Sampling	-	Yes	Yes	Inductive Setting
Chen et al. [53]	Spatial	First-order + Sampling	-	Yes	-	-

4.1 Convolution Operations

4.1.1 Spectral Methods

For CNNs, convolution is the most fundamental operation. However, standard convolution for image or text can not be directly applied to graphs because of the lack of a grid structure [6]. Bruna et al. [33] first introduce convolution for graph data from spectral domain using the graph Laplacian matrix $\mathbf{L}$ [54], which plays a similar role as the Fourier basis for signal processing [6]. Specifically, the convolution operation on graph $*_G$ is defined as:

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

where $\mathbf{u}_1, \mathbf{u}_2 \in \mathbb{R}^N$ are two signals defined on nodes and $\mathbf{Q}$ are eigenvectors of $\mathbf{L}$. Then, using the convolution theorem, filtering a signal $\mathbf{u}$ can be obtained as:

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

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

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

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

However, directly using Eq. (6) requires $O(N)$ parameters to be learned, which may not be feasible in practice. In addition, the filters in spectral domain may not be localized in the spatial domain. To alleviate these problems, Bruna et al. [33] suggest using the following smooth filters:

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

where $\mathcal{K}$ is a fixed interpolation kernel and $\alpha_{l,i,j}$ are learnable interpolation coefficients. The authors also generalize this idea to the setting where the graph is not given but constructed from some raw features using either a supervised or an unsupervised method [34]. However, two fundamental limitations remain unsolved. First, since the full eigenvectors of the Laplacian matrix are needed during each calculation, the time complexity is at least $O(N^2)$ per forward and backward pass, which is not scalable to large-scale graphs. Second, since the filters depend on the eigen-basis $\mathbf{Q}$ of the graph, parameters can not be shared across multiple graphs with different sizes and structures.

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

4.1.2 Efficiency Aspect

To solve the efficiency problem, ChebNet [35] proposes using a polynomial filter as follows:

$$\mathbf{\Theta}(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k\mathbf{\Lambda}^k, \tag{8}$$

where $\theta_0, ..., \theta_{K-1}$ are learnable parameters and K is the polynomial order. Then, instead of performing the eigen-decomposition, the authors rewrite Eq. (8) using the Chebyshev expansion [55]:

$$\mathbf{\Theta}(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k\mathcal{T}_k(\tilde{\mathbf{\Lambda}}), \tag{9}$$

where $\tilde{\mathbf{\Lambda}} = 2\mathbf{\Lambda}/\lambda_{max} - \mathbf{I}$ are the rescaled eigenvalues, λ_{max} is the maximum eigenvalue, $\mathbf{I} \in \mathbb{R}^{N\times N}$ is the identity matrix and $\mathcal{T}_k(x)$ is the Chebyshev polynomial of order k. The rescaling is necessary because of the orthonormal basis of Chebyshev polynomials. Using the fact that polynomial of the Laplacian acts

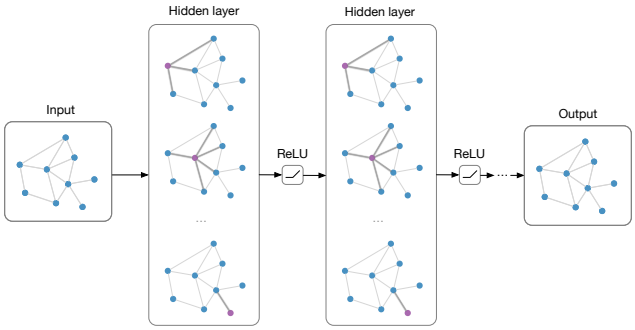

Fig. 2. An illustrating example of convolution operation proposed by Kipf & Welling [36]. Reprinted with permission.

as a polynomial of its eigenvectors, the filter operation in Eq. (5) can be rewritten as:

$$\begin{aligned}\mathbf{u}' = \mathbf{Q}\boldsymbol{\Theta}(\boldsymbol{\Lambda})\mathbf{Q}^T\mathbf{u} &= \sum_{k=0}^{K-1}\theta_k\mathbf{Q}\mathcal{T}_k(\tilde{\boldsymbol{\Lambda}})\mathbf{Q}^T\mathbf{u} \\ &= \sum_{k=0}^{K-1}\theta_k\mathcal{T}_k(\tilde{\mathbf{L}})\mathbf{u} = \sum_{k=0}^{K-1}\theta_k\bar{\mathbf{u}}_k,\end{aligned} \tag{10}$$

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

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

with $\bar{\mathbf{u}}_0 = \mathbf{u}, \bar{\mathbf{u}}_1 = \tilde{\mathbf{L}}\mathbf{u}$. Now, since only the matrix product of $\tilde{\mathbf{L}}$ and some vectors needs to be calculated, the time complexity is $O(KM)$, where M is the number of edges and K is the polynomial order, i.e. linear with respect to the graph size. It is also easy to see that such polynomial filter is strictly K-localized: after one convolution, the representation of node v_i will only be affected by its K-step neighborhood $\mathcal{N}_K(i)$. Interestingly, this idea is independently used in network embedding to preserve the high-order proximity [56], of which we omit the details for brevity.

An improvement to ChebNet introduced by Kipf and Welling [36] further simplifies the filtering by only using the first-order neighbors as follows:

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

where $\mathbf{h}_i^l \in \mathbb{R}^{f_l}$ is the hidden representation of node v_i in the l^{th} layer[2], $\tilde{\mathbf{D}} = \mathbf{D} + \mathbf{I}$ and $\tilde{\mathcal{N}}(i) = \mathcal{N}(i) \cup \{i\}$. This can be written equivalently in the matrix form:

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

where $\tilde{\mathbf{A}} = \mathbf{A}+\mathbf{I}$, i.e. adding a self-connection. The authors show that Eq. (13) is a special case of Eq. (8) by setting $K = 1$ with a few minor changes. Then, the authors argue that stacking K such layers has a similar modeling capacity as ChebNet and leads to better results. The architecture is illustrated in Figure 2.

An important point of ChebNet and its extension is that they connect the spectral graph convolution with the spatial architecture as in GNNs. Actually, the convolution in Eq. (12) is very similar to the definition of states in GNN in Eq. (1), except the convolution definition replaces the recursive definition. In this aspect, GNN can be regarded as GCN using a large number of identical layers to reach stable states [7].

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

4.1.3 Multiple Graphs Aspect

In the meantime, a parallel of works focus on generalizing convolution operation to multiple graphs of arbitrary sizes. Neural FPs [37] propose a spatial method also using the first-order neighbors:

$$\mathbf{h}_i^{l+1} = \sigma\left(\sum\nolimits_{j\in\hat{\mathcal{N}}(i)}\mathbf{h}_j^l\boldsymbol{\Theta}^l\right). \tag{14}$$

Since the parameters $\boldsymbol{\Theta}$ can be shared across different graphs and are independent of graph sizes, Neural FPs can handle multiple graphs of arbitrary sizes. Note that Eq. (14) is very similar to Eq. (12). However, instead of considering the influence of node degrees by adding a normalization term, Neural FPs propose learning different parameters $\boldsymbol{\Theta}$ for nodes with different degrees. This strategy performs well for small graphs such as the molecular graphs, i.e. atoms as nodes and bonds as edges, but may not be scalable to large-scale graphs.

PATCHY-SAN [38] adopts a different idea to assign a unique order of nodes using the graph labeling procedure such as the Weisfeiler-Lehman kernel [57] and arranges nodes in a line using this pre-defined order. To mimic conventional CNNs, PATCHY-SAN defines a "receptive field" for each node v_i by selecting a fixed number of nodes from their k-step neighborhoods $\mathcal{N}_k(i)$ and then adopts standard 1-D CNN with proper normalization. Since now nodes in different graphs all have a "receptive field" with fixed size and order, PATCHY-SAN can learn from multiple graphs like normal CNNs. However, the drawbacks are that the convolution depends heavily on the graph labeling procedure which is a preprocessing step that is not learned, and enforcing a 1-D ordering of nodes may not be a natural choice.

DCNN [39] adopts another approach to replace the eigen-basis of convolution by a diffusion-basis, i.e. the "receptive field" of nodes is determined by the diffusion transition probability between nodes. Specifically, the convolution is defined as:

$$\mathbf{H}^{l+1} = \rho\left(\mathbf{P}^K\mathbf{H}^l\boldsymbol{\Theta}^l\right), \tag{15}$$

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

DGCN [40] further proposes to jointly adopt diffusion and adjacency basis using a dual graph convolutional network. Specifically, DGCN uses two convolutions: one as Eq. (13), and the other replaces the adjacency matrix with positive pointwise mutual information (PPMI) matrix [58] of the transition probability, i.e.

$$\mathbf{Z}^{l+1} = \rho\left(\mathbf{D}_P^{-\frac{1}{2}}\mathbf{X}_P\mathbf{D}_P^{-\frac{1}{2}}\mathbf{Z}^l\boldsymbol{\Theta}^l\right), \tag{16}$$

where $\mathbf{X}_P$ is the PPMI matrix and $\mathbf{D}_P(i,i) = \sum_{j\neq i}\mathbf{X}_P(i,j)$ is the diagonal degree matrix of $\mathbf{X}_P$. Then, two convolutions are ensembled by minimizing the mean square differences between $\mathbf{H}$ and $\mathbf{Z}$. A random walk sampling procedure is also proposed to accelerate the calculation of transition probability. Experiments

demonstrate that such dual convolutions are effective even for single-graph problems.

4.1.4 Frameworks

Based on above two lines of works, MPNNs [41] propose a unified framework for the graph convolution operation in the spatial domain using a message passing function:

$$\begin{aligned}\mathbf{m}_i^{l+1} &= \sum_{j\in\mathcal{N}(i)} \mathcal{F}^l\left(\mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{F}_{i,j}^E\right)\\ \mathbf{h}_i^{l+1} &= \mathcal{G}^l\left(\mathbf{h}_i^l, \mathbf{m}_i^{l+1}\right),\end{aligned} \tag{17}$$

where $\mathcal{F}^l$ and $\mathcal{G}^l$ are message functions and vertex update functions that need to be learned, and $\mathbf{m}^l$ are the "messages" passed between nodes. Conceptually, MPNNs propose a framework that each node sends messages based on its states and updates its states based on messages received from immediate neighbors. The authors show that the above framework includes many previous methods such as [26], [33], [34], [36], [37], [50] as special cases. Besides, the authors propose adding a "master" node that is connected to all nodes to accelerate the passing of messages across long distances and split the hidden representation into different "towers" to improve the generalization ability. The authors show that a specific variant of the MPNNs can achieve state-of-the-art performance on predicting molecular properties.

Concurrently, GraphSAGE [42] takes a similar idea as Eq. (17) with multiple aggregating functions as follows:

$$\begin{aligned}\mathbf{m}_i^{l+1} &= \text{AGGREGATE}^l(\{\mathbf{h}_j^l, \forall j \in \mathcal{N}(i)\})\\ \mathbf{h}_i^{l+1} &= \rho\left(\mathbf{\Theta}^l\left[\mathbf{h}_i^l, \mathbf{m}_i^{l+1}\right]\right),\end{aligned} \tag{18}$$

where $[\cdot,\cdot]$ is concatenation and $\text{AGGREGATE}(\cdot)$ is the aggregating function. The authors suggest three aggregating functions: element-wise mean, long short-term memory (LSTM) [59] and pooling as follows:

$$\text{AGGREGATE}^l = \max\{\rho(\mathbf{\Theta}_{\text{pool}}\mathbf{h}_j^l + \mathbf{b}_{\text{pool}}), \forall j \in \mathcal{N}(i)\}, \tag{19}$$

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

Mixture model network (MoNet) [43] also tries to unify previous works of GCNs as well as CNN for manifolds into a common framework using "template matching":

$$h_{ik}^{l+1} = \sum_{j\in\mathcal{N}(i)} \mathcal{F}_k^l(\mathbf{u}(i,j))\mathbf{h}_j^l, k = 1, ..., f_{l+1}, \tag{20}$$

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

$$\mathcal{F}_k^l(\mathbf{u}) = \exp\left(-\frac{1}{2}(\mathbf{u}-\boldsymbol{\mu}_k^l)^T(\mathbf{\Sigma}_k^l)^{-1}(\mathbf{u}-\boldsymbol{\mu}_k^l)\right), \tag{21}$$

where $\boldsymbol{\mu}_k^l$ are mean vectors and $\mathbf{\Sigma}_k^l$ are diagonal covariance matrices to be learned. The pseudo-coordinates are set to be degrees as in [36], i.e.

$$\mathbf{u}(i,j) = \left(\frac{1}{\sqrt{\mathbf{D}(i,i)}}, \frac{1}{\sqrt{\mathbf{D}(j,j)}}\right). \tag{22}$$

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

$$\begin{aligned}\mathbf{m}_i^l &= \mathcal{G}^{E\to V}(\{\mathbf{h}_j^l, \forall j \in \mathcal{N}(i)\})\\ \mathbf{m}_V^l &= \mathcal{G}^{V\to G}(\{\mathbf{h}_i^l, \forall v_i \in V\})\\ \mathbf{m}_E^l &= \mathcal{G}^{E\to G}(\{\mathbf{h}_{ij}^l, \forall (v_i,v_j) \in E\})\\ \mathbf{h}_i^{l+1} &= \mathcal{F}^V(\mathbf{m}_i^l, \mathbf{h}_i^l, \mathbf{z}^l)\\ \mathbf{e}_{ij}^{l+1} &= \mathcal{F}^E(\mathbf{e}_{ij}^l, \mathbf{h}_i^l, \mathbf{h}_j^l, \mathbf{z}^l)\\ \mathbf{z}^{l+1} &= \mathcal{F}^G(\mathbf{m}_E^l, \mathbf{m}_V^l, \mathbf{z}^l),\end{aligned} \tag{23}$$

where $\mathcal{F}^V(\cdot), \mathcal{F}^E(\cdot), \mathcal{F}^G(\cdot)$ are corresponding updating functions for nodes, edges and the whole graph respectively and $\mathcal{G}(\cdot)$ are message-passing functions with superscripts denoting message-passing directions. Compared with MPNNs, GNs introduce edge representations and the whole graph representation, thus making the framework more general.

In summary, the convolution operations have evolved from the spectral domain to the spatial domain and from multi-step neighbors to the immediate neighbors. Currently, gathering information from immediate neighbors like Eq. (13) and following the framework of Eqs. (17) (18) (23) are the most common choices for the graph convolution operation.

4.2 Readout Operations

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

Order invariance. A critical requirement for the graph readout operation is that the operation should be invariant to the order of nodes, i.e. if we change the indices of nodes and edges using a bijective function between two vertex sets, representation of the whole graph should not change. For example, whether a drug can treat certain disease should be independent of how the drug is represented as a graph. Note that since this problem is related to the graph isomorphism problem which is known to be NP [60], we can only find a function that is order invariant but not vice versa in polynomial time, i.e. even two graphs are not isomorphism, they may have the same representation.

4.2.1 Statistics

The most basic operations that are order invariant are simple statistics like taking sum, average or max-pooling [37], [39], i.e.

$$\mathbf{h}_G = \sum_{i=1}^N \mathbf{h}_i^L \text{ or } \mathbf{h}_G = \frac{1}{N}\sum_{i=1}^N \mathbf{h}_i^L \text{ or } \mathbf{h}_G = \max\left\{\mathbf{h}_i^L, \forall i\right\}, \tag{24}$$

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

In [50], the authors suggest considering the distribution of node representations by using fuzzy histograms [61]. The basic idea of fuzzy histograms is to construct several "histogram

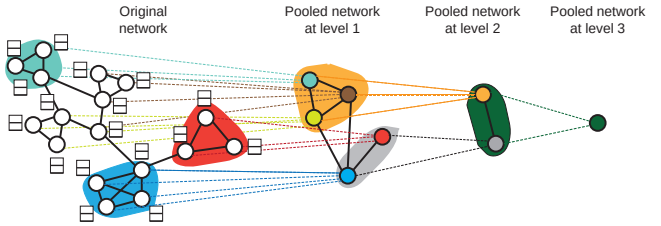

Fig. 3. An example of hierarchical clustering of a graph. Reprinted from [44] with permission.

bins" and then calculate the memberships of $\mathbf{h}_i^L$ to these bins, i.e. regarding representations of nodes as samples and match them to some pre-defined templates, and return concatenation of the final histograms. In this way, nodes with the same sum/average/maximum but with different distributions can be distinguished.

Another commonly used approach to gather information is to add a fully connected (FC) layer as the final layer [33], i.e.

$$\mathbf{h}_G = \mathbf{\Theta}_{FC}\mathbf{H}^L, \tag{25}$$

where $\mathbf{\Theta}_{FC}$ are parameters of the FC layer. Eq. (25) can be regarded as a weighted sum of combing node-level features. One advantage is that the model can learn different weights for different nodes, at the cost of being unable to guarantee order invariance.

4.2.2 Hierarchical clustering

Rather than a dichotomy between node or graph level structure, graphs are known to exhibit rich hierarchical structures [62], which can be explored by hierarchical clustering methods as shown in Figure 3. For example, a density based agglomerative clustering [63] is used in Bruna et al. [33] and multi-resolution spectral clustering [64] is used in Henaff et al. [34]. ChebNet [35] and MoNet [43] adopt Graclus [65], another greedy hierarchical clustering algorithm to merge two nodes at a time, together with a fast pooling method by rearranging the nodes into a balanced binary tree. ECC [48] adopts another hierarchical clustering method by eigen-decomposition [66]. However, these hierarchical clustering methods are all independent of the convolution operation, i.e. can be done as a pre-processing step and not trained end-to-end.

To solve that problem, DiffPool [44] proposes a differentiable hierarchical clustering algorithm jointly trained with graph convolutions. Specifically, the authors propose learning a soft cluster assignment matrix in each layer using the hidden representations:

$$\mathbf{S}^l = \mathcal{F}\left(\mathbf{A}^l, \mathbf{H}^l\right), \tag{26}$$

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

$$\mathbf{H}^{l+1} = (\mathbf{S}^l)^T\hat{\mathbf{H}}^{l+1}, \mathbf{A}^{l+1} = (\mathbf{S}^l)^T\mathbf{A}^l\mathbf{S}^l, \tag{27}$$

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

4.2.3 Others

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

In GNNs [20], the authors suggest adding a special node that is connected to all nodes to represent the whole graph. Similarly, GNs [9] take the idea of directly learning the representation of the whole graph by receiving messages from all nodes and edges.

MPNNs adopt set2set [67], a modification of seq2seq model that is invariant to the order of inputs. Specifically, set2set uses a Read-Process-and-Write model that receives all inputs at once, computes internal memories using attention mechanism and LSTM, and then writes the outputs.

As mentioned earlier, PATCHY-SAN [38] takes the idea of imposing an order of nodes using a graph labeling procedure and then resorts to standard 1-D pooling as in CNNs. Whether such method can preserve order invariance depends on the graph labeling procedure, which is another research field that is beyond the scope of this paper [68]. However, imposing an order for nodes may not be a natural choice for gathering node information and could hinder the performance of downstream tasks.

In short, statistics like taking average or sum are most simple readout operations, while hierarchical clustering algorithm jointly trained with graph convolutions is more advanced but sophisticated solution. For specific problems, other methods exist as well.

4.3 Improvements and Discussions

4.3.1 Attention Mechanism

In previous GCNs, the neighborhoods of nodes are combined with equal or pre-defined weights. However, the influence of neighbors can vary greatly, which should better be learned during training than pre-determined. Inspired by the attention mechanism [69], Graph Attention Networks (GATs) [45] introduce attentions into GCNs by modifying the convolution in Eq (12) as follows:

$$\mathbf{h}_i^{l+1} = \rho\left(\sum_{j\in\hat{\mathcal{N}}(i)} \alpha_{ij}^l \mathbf{h}_j^l \mathbf{\Theta}^l\right), \tag{28}$$

where α_{ij}^l is the attention defined as:

$$\alpha_{ij}^l = \frac{\exp\left(\text{LeakyReLU}\left(\mathcal{F}\left(\mathbf{h}_i^l\mathbf{\Theta}^l, \mathbf{h}_j^l\mathbf{\Theta}^l\right)\right)\right)}{\sum_{k\in\hat{\mathcal{N}}(i)}\exp\left(\text{LeakyReLU}\left(\mathcal{F}\left(\mathbf{h}_i^l\mathbf{\Theta}^l, \mathbf{h}_k^l\mathbf{\Theta}^l\right)\right)\right)}, \tag{29}$$

where $\mathcal{F}(\cdot,\cdot)$ is another function to be learned such as a small fully connected network. The authors also suggest using multiply independent attentions and concatenating the results, i.e. the multi-head attention in [69], as illustrated in Figure 4.

4.3.2 Residual and Jumping Connections

Similar to ResNet [70], residual connections can be added into existing GCNs to skip certain layers. For example, Kipf & Welling [36] add residual connections into Eq. (13) as follows:

$$\mathbf{H}^{l+1} = \rho\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^l\mathbf{\Theta}^l\right) + \mathbf{H}^l. \tag{30}$$

They show experimentally that adding such residual connections can increase the depth of the network, i.e. number of convolution layers in GCNs, which is similar to the results of ResNet.

Column Network (CLN) [46] takes a similar idea using the following residual connections with learnable weights:

$$\mathbf{h}_i^{l+1} = \boldsymbol{\alpha}_i^l \odot \widetilde{\mathbf{h}}_i^{l+1} + (1-\boldsymbol{\alpha}_i^l)\odot \mathbf{h}_i^l, \tag{31}$$

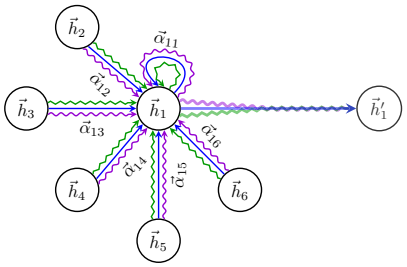

Fig. 4. An illustration of multi-head attentions proposed in GATs [45] where each color denotes an independent attention. Reprinted with permission.

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

$$\boldsymbol{\alpha}_i^l = \rho\left(\mathbf{b}_\alpha^l + \boldsymbol{\Theta}_\alpha^l \mathbf{h}_i^l + \boldsymbol{\Theta}_\alpha^{'l} \sum_{j \in \mathcal{N}(i)} \mathbf{h}_j^l\right), \tag{32}$$

where $\mathbf{b}_\alpha^l, \boldsymbol{\Theta}_\alpha^l, \boldsymbol{\Theta}_\alpha^{'l}$ are some parameters to be learned. Note that Eq. (31) is very similar to the GRU as in GGS-NNs [26], but the overall architecture is based on convolutional layers instead of pseudo time.

Jumping Knowledge Networks (JK-Nets) [47] propose another architecture to connect the last layer of the network with all previous hidden layers, i.e. "jumping" all representations to the final output as illustrated in Figure 5. In this way, the model can learn to selectively exploit information from different localities. Formally, JK-Nets can be formulated as:

$$\mathbf{h}_i^{final} = \text{AGGREGATE}(\mathbf{h}_i^0, \mathbf{h}_i^1, ..., \mathbf{h}_i^K), \tag{33}$$

where $\mathbf{h}_i^{final}$ is final representation for node v_i, AGGREGATE$(\cdot)$ is the aggregating function and K is the number of hidden layers. JK-Nets use three aggregating functions similar to GraphSAGE [42]: concatenation, max-pooling and LSTM attention. Experimental results show that adding jumping connections can improve the performance of multiple GCN architectures.

4.3.3 Edge Features

Previous GCNs mostly focus on utilizing node features. Another important source of information is the edge features, which we briefly discuss in this section.

For simple edge features with discrete values, such as edge types, a straight-forward method is to train different parameters for different edge types and aggregate the results. For example, Neural FPs [37] train different parameters for nodes with different degrees, which corresponds to the hidden edge feature of bond types in a molecular graph, and sum over the results. CLN [46] trains different parameters for different types of edges in a heterogenous graph and average the results. Edge-Conditioned Convolution (ECC) [48] also trains different parameters based on edge types and applies it to graph classification. Relational GCNs (R-GCNs) [49] take a similar idea in knowledge graphs by training different weights for different relation types. However, these methods can only handle limited discrete edge features.

DCNN [39] proposes another method to convert each edge into a node connected to the head and tail node of the edge. Then, edge features can be treated as node features.

Kearnes et al. [50] propose another architecture using the "weave module". Specifically, they learn representations for both

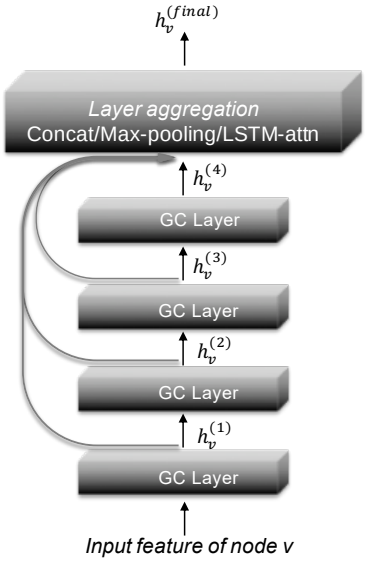

Fig. 5. Jumping Knowledge Networks proposed in [47] where the last layer is connected to all hidden layers to selectively exploit information from different localities. Reprinted with permission.

nodes and edges and exchange information between them in each weave module with four different functions: Node-to-Node (NN), Node-to-Edge (NE), Edge-to-Edge (EE) and Edge-to-Node (EN):

$$\begin{aligned}
\mathbf{h}_i^{l'} &= \mathcal{F}_{NN}(\mathbf{h}_i^0, \mathbf{h}_i^1, ..., \mathbf{h}_i^l) \\
\mathbf{h}_i^{l''} &= \mathcal{F}_{EN}(\{\mathbf{e}_{ij}^l | j \in \mathcal{N}(i)\}) \\
\mathbf{h}_i^{l+1} &= \mathcal{F}_{NN}(\mathbf{h}_i^{l'}, \mathbf{h}_i^{l''}) \\
\mathbf{e}_{ij}^{l'} &= \mathcal{F}_{EE}(\mathbf{e}_{ij}^0, \mathbf{e}_{ij}^1, ..., \mathbf{e}_{ij}^l) \\
\mathbf{e}_{ij}^{l''} &= \mathcal{F}_{NE}(\mathbf{h}_i^l, \mathbf{h}_j^l) \\
\mathbf{e}_{ij}^{l+1} &= \mathcal{F}_{EE}(\mathbf{e}_{ij}^{l'}, \mathbf{e}_{ij}^{l''}),
\end{aligned} \tag{34}$$

where $\mathbf{e}_{ij}^l$ are representations for edge (v_i, v_j) in the l^{th} layer and $\mathcal{F}(\cdot)$ are learnable functions with subscripts representing message passing directions. By stacking multiple such modules, information can propagate through alternative passing between nodes and edges representations. Note that in Node-to-Node and Edge-to-Edge functions, jumping connections similar to JK-Nets [47] are implicitly added. Graph Networks [9] also propose learning edge representation and update both nodes and edges representations using message passing functions as discussed in Section 4.1, which contain the "weave module" as a special case.

4.3.4 Accelerating by Sampling

One critical bottleneck of training GCNs for large-scale graphs is the efficiency problem. In this section, we review some acceleration methods for GCN.

As shown previously, many GCNs follow the framework of aggregating information from neighborhoods. However, since many real graphs follow the power-law distribution [71], i.e. few nodes have very large degrees, the expansion of neighbors can grow extremely fast. To deal with this problem, GraphSAGE [42] uniformly samples a fixed number of neighbors for each node during training. PinSage [51] further proposes sampling neighbors using random walks on graphs together with several implementation improvements, e.g. coordination between CPU

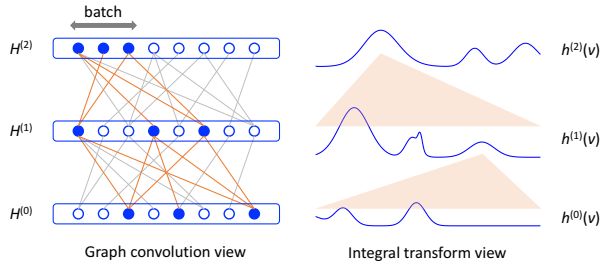

Fig. 6. Node sampling method of FastGCN reprinted from [52] with permission. FastGCN interprets graph convolutions as integral transforms and samples nodes in each convolutional layer.

and GPU, a map-reduce inference pipeline, etc. PinSage is shown to be working well in a real billion-scale graph.

FastGCN [52] adopts a different sampling strategy. Instead of sampling neighbors of each node, the authors suggest sampling nodes in each convolutional layer by interpreting nodes as i.i.d. samples and graph convolutions as integral transforms under probability measures as shown in Figure 6. FastGCN also shows that sampling nodes via their normalized degrees can reduce variance and lead to better performance.

Chen et al. [53] further propose another sampling method to reduce variance. Specifically, historical activations of nodes are used as a control variate, which allows for arbitrarily small sample sizes. The authors also prove this method has theoretical guarantee and outperforms existing acceleration methods in experiments.

4.3.5 *Inductive Setting*

Another important aspect of GCNs is applying to the inductive setting, i.e. training on a set of nodes/graphs and testing on another set of nodes/graphs unseen during training. In principle, this is achieved by learning a mapping function on the given features, which are not dependent on the graph basis and can be transferred across nodes/graphs. The inductive setting is verified in GraphSAGE [42], GATs [45] and FastGCN [52]. However, existing inductive GCNs are only suitable for graphs with features. How to conduct inductive learning for graphs without features, usually called the out-of-sample problem [72], largely remains open in the literature.

4.3.6 *Random Weights*

GCNs with random weights are also an interesting research direction, similar to the case of general neural networks [73]. For example, Neural FPs [37] with large random weights are shown to have similar performance as circular fingerprints, some handcrafted features for molecules with hash indexing. Kipf and Welling [36] also show that untrained GCNs can extract certain meaningful node features. However, a general understanding of GCNs with random weights remain unexplored.

5 Graph Autoencoders (GAEs)

Autoencoder (AE) and its variations are widely used for unsupervised learning [74], which are suitable to learn node representations for graphs without supervised information. In this section, we will first introduce graph autoencoders and then move to graph variational autoencoders and other improvements. Main characteristics of GAEs surveyed are summarized in Table 4.

5.1 Autoencoders

The use of AEs for graphs is originated from Sparse Autoencoder (SAE) [75][3]. The basic idea is that, by regarding adjacency matrix or its variations as the raw features of nodes, AEs can be leveraged as a dimension reduction technique to learn low-dimensional node representations. Specifically, SAE adopts the following L2-reconstruction loss:

$$\begin{aligned} \min_{\mathbf{\Theta}} \mathcal{L}_2 &= \sum_{i=1}^{N} \left\| \mathbf{P}\left(i,:\right) - \hat{\mathbf{P}}\left(i,:\right) \right\|_2 \\ \hat{\mathbf{P}}\left(i,:\right) &= \mathcal{G}\left(\mathbf{h}_i\right), \mathbf{h}_i = \mathcal{F}\left(\mathbf{P}\left(i,:\right)\right), \end{aligned} \tag{35}$$

where $\mathbf{P}$ is the transition matrix, $\hat{\mathbf{P}}$ is the reconstructed matrix, $\mathbf{h}_i \in \mathbb{R}^d$ is the low-dimensional representation of node v_i, $\mathcal{F}(\cdot)$ is the encoder, $\mathcal{G}(\cdot)$ is the decoder, $d \ll N$ is the dimensionality and $\mathbf{\Theta}$ are parameters. Both encoder and decoder are multi-layer perceptron with many hidden layers. In other words, SAE tries to compress the information of $\mathbf{P}(i,:)$ into low-dimensional vectors $\mathbf{h}_i$ and reconstruct the original vector. SAE also adds another sparsity regularization term. After getting the low-dimensional representation $\mathbf{h}_i$, k-means [85] is applied for the node clustering task, which proves empirically to outperform non deep learning baselines. However, since the theoretical analysis is incorrect, the mechanism underlying such effectiveness remains unexplained.

Structure Deep Network Embedding (SDNE) [76] fills in the puzzle by showing that the L2-reconstruction loss in Eq. (35) actually corresponds to the second-order proximity, i.e. two nodes share similar latten representations if they have similar neighborhoods, which is well studied in network science such as in collaborative filtering or triangle closure [5]. Motivated by network embedding methods, which show that the first-order proximity is also important [86], SDNE modifies the objective function by adding another term similar to the Laplacian Eigenmaps [54]:

$$\min_{\mathbf{\Theta}} \mathcal{L}_2 + \alpha \sum_{i,j=1}^{N} \mathbf{A}(i,j) \left\| \mathbf{h}_i - \mathbf{h}_j \right\|_2, \tag{36}$$

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

$$\mathcal{L}_2 = \sum_{i=1}^{N} \left\| \left(\mathbf{A}\left(i,:\right) - \mathcal{G}\left(\mathbf{h}_i\right)\right) \odot \mathbf{b}_i \right\|_2, \tag{37}$$

where $b_{ij} = 1$ if $\mathbf{A}(i,j) = 0$ and $b_{ij} = \beta > 1$ else and β is another hyper-parameter. The overall architecture of SDNE is shown in Figure 7.

Motivated by another line of works, a contemporary work DNGR [77] replaces the transition matrix $\mathbf{P}$ in Eq. (35) with the positive pointwise mutual information (PPMI) [58] matrix of a random surfing probability. In this way, the raw features can associate with some random walk probability of the graph [87]. However, constructing the input matrix can take $O(N^2)$ time complexity, which is not scalable to large-scale graphs.

GC-MC [78] further takes a different approach for autoencoders by using GCN in [36] as the encoder:

$$\mathbf{H} = GCN\left(\mathbf{F}^V, \mathbf{A}\right), \tag{38}$$

3. The original paper [75] motivates the problem by analyzing the connection between spectral clustering and Singular Value Decomposition, which is mathematically incorrect as pointed out in [84]. We keep their work for completeness of the literature.

TABLE 4
Comparison of Different Graph Autoencoders (GAEs)

Method	Type	Objective	Scalability	Node Features	Other Improvements
SAE [75]	AE	L2-Reconstruction	Yes	No	-
SDNE [76]	AE	L2-Reconstruction + Laplacian Eigenmaps	Yes	No	-
DNGR [77]	AE	L2-Reconstruction	No	No	-
GC-MC [78]	AE	L2-Reconstruction	Yes	Yes	Convolutional Encoder
DRNE [79]	AE	Recursive Reconstruction	Yes	No	-
G2G [80]	AE	KL + Ranking	Yes	Yes	Nodes as distributions
VGAE [81]	VAE	Pairwise Probability of Reconstruction	No	Yes	Convolutional Encoder
DVNE [82]	VAE	Wasserstein + Ranking	Yes	No	Nodes as distributions
ARGA/ARVGA [83]	AE/VAE	L2-Reconstruction + GAN	Yes	Yes	Convolutional Encoder

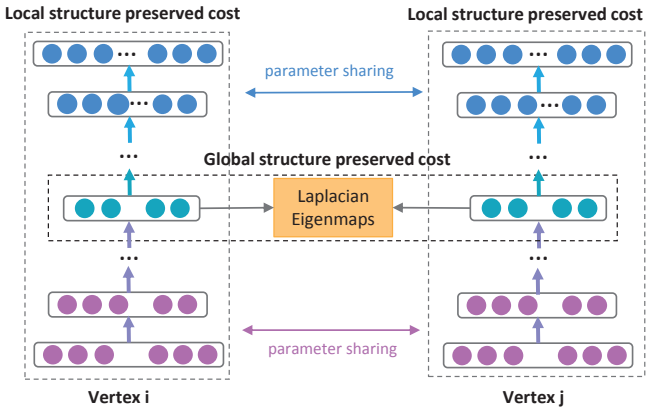

Fig. 7. The framework of SDNE reprinted from [76] with permission. Both the first and the second order proximity of nodes are preserved using deep autoencoders.

and the decoder is a simple bilinear function:

$$\hat{\mathbf{A}}(i,j) = \mathbf{H}(i,:)\mathbf{\Theta}_{de}\mathbf{H}(j,:)^T, \tag{39}$$

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

Instead of reconstructing the adjacency matrix or its variations, DRNE [79] proposes another modification to directly reconstruct the low-dimensional vectors of nodes by aggregating neighborhood information using LSTM. Specifically, DRNE minimizes the following objective function:

$$\mathcal{L} = \sum_{i=1}^{N} \left\| \mathbf{h}_i - \text{LSTM}\left(\{\mathbf{h}_j | j \in \mathcal{N}(i)\}\right) \right\|. \tag{40}$$

Since LSTM requires a sequence of inputs, the authors suggest ordering the neighborhoods of nodes according to their degrees. A sampling of neighbors is also adopted for nodes with large degrees to prevent the memory from being too long. The authors prove that such method can preserve regular equivalence and many centrality measures of nodes such as PageRank [88].

Unlike previous works which map nodes into low-dimensional vectors, Graph2Gauss (G2G) [80] proposes encoding each node as a Gaussian distribution $\mathbf{h}_i = \mathcal{N}\left(\mathbf{M}(i,:), diag\left(\mathbf{\Sigma}(i,:)\right)\right)$ to capture the uncertainties of nodes. Specifically, the authors use a deep mapping from node attributes to the means and variances of the Gaussian distribution as the encoder:

$$\mathbf{M}(i,:) = \mathcal{F}_{\mathbf{M}}(\mathbf{F}^V(i,:)), \mathbf{\Sigma}(i,:) = \mathcal{F}_{\mathbf{\Sigma}}(\mathbf{F}^V(i,:)), \tag{41}$$

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

$$\begin{aligned} &\text{KL}\left(\mathbf{h}_j || \mathbf{h}_i\right) < \text{KL}\left(\mathbf{h}_{j'} || \mathbf{h}_i\right) \\ &\forall i, \forall j, \forall j' \; s.t. \; d(i,j) < d(i,j'), \end{aligned} \tag{42}$$

where $d(i,j)$ is the shortest distance from node v_i to v_j and $\text{KL}[q(\cdot)||p(\cdot)]$ is the Kullback-Leibler (KL) divergence between $q(\cdot)$ and $p(\cdot)$ [89]. In other words, the constrains ensure that KL-divergence between node pairs has the same relative order as the graph distance. However, since Eq. (42) is hard to optimize, energy-based loss [90] is resorted to as an relaxation:

$$\mathcal{L} = \sum_{(i,j,j') \in \mathcal{D}} \left(E_{ij}^2 + \exp^{-E_{ij'}} \right), \tag{43}$$

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

5.2 Variational Autoencoders

As opposed to previous autoencoders, Variational Autoencoder (VAE) is another type of deep learning method that combines dimension reduction with generative models [91]. VAE was first introduced into modeling graph data in [81], where the decoder is a simple linear product:

$$p\left(\mathbf{A}|\mathbf{H}\right) = \prod_{i,j=1}^{N} \sigma\left(\mathbf{h}_i \mathbf{h}_j^T\right), \tag{44}$$

where $\mathbf{h}_i$ are assumed to follow a Gaussian posterior distribution $q\left(\mathbf{h}_i|\mathbf{M}, \mathbf{\Sigma}\right) = \mathcal{N}\left(\mathbf{h}_i|\mathbf{M}(i,:), diag\left(\mathbf{\Sigma}(i,:)\right)\right)$. For the encoder of mean and variance matrices, the authors adopt GCN in [36]:

$$\mathbf{M} = GCN_{\mathbf{M}}\left(\mathbf{F}^V, \mathbf{A}\right), \log \mathbf{\Sigma} = GCN_{\mathbf{\Sigma}}\left(\mathbf{F}^V, \mathbf{A}\right). \tag{45}$$

Then, the model parameters can be learned by minimizing the variational lower bound [91]:

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{H}|\mathbf{F}^V, \mathbf{A})}\left[\log p\left(\mathbf{A}|\mathbf{H}\right)\right] - \text{KL}\left[q\left(\mathbf{H}|\mathbf{F}^V, \mathbf{A}\right) || p(\mathbf{H})\right]. \tag{46}$$

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

Motivated by SDNE and G2G, DVNE [82] proposes another VAE for graph data by also representing each node as a Gaussian distribution. Unlike previous works which adopt KL-divergence as the measurement, DVNE uses Wasserstein distance [92] to preserve the transitivity of nodes similarities. Similar to SDNE

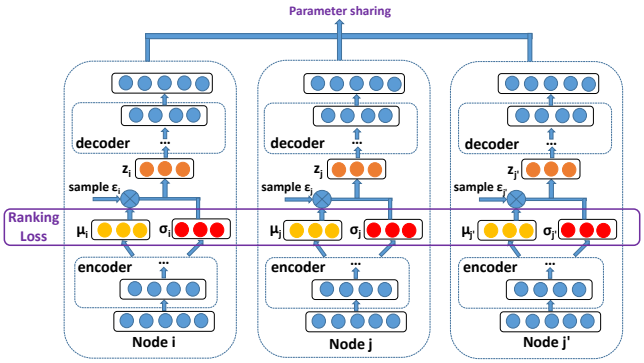

Fig. 8. The framework of DVNE reprinted from [82] with permission. DVNE represents nodes as distributions using VAE and adopts Wasserstein distance to preserve transitivity of nodes similarity.

and G2G, DVNE also preserve both the first and second order proximity in the objective function:

$$\min_{\mathbf{\Theta}} \sum_{(i,j,j')\in\mathcal{D}} \left(E_{ij}^2 + \exp^{-E_{ij'}}\right) + \alpha\mathcal{L}_2, \tag{47}$$

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

$$\mathcal{L}_2 = \inf_{q(\mathbf{Z}|\mathbf{P})} \mathbb{E}_{p(\mathbf{P})}\mathbb{E}_{q(\mathbf{Z}|\mathbf{P})} \left\|\mathbf{P} \odot (\mathbf{P} - \mathcal{G}(\mathbf{Z}))\right\|_2^2, \tag{48}$$

where $\mathbf{P}$ is the transition matrix and $\mathbf{Z}$ are samples drawn from $\mathbf{H}$. The framework is shown in Figure 8. Then, the objective function can be minimized as conventional VAEs using the reparameterization trick [91].

5.3 Improvements and Discussions

Besides these two main categories, there are also several improvements which are worthy discussions.

5.3.1 Adversarial Training

The adversarial training scheme, especially the generative adversarial network (GAN), has been a hot topic in machine learning recently [93]. The basic idea of GAN is to build two linked models, a discriminator and a generator. The goal of the generator is to "fool" the discriminator by generating fake data, while the discriminator aims to distinguish whether a sample is from real data or generated by the generator. Then, both models can benefit from each other by joint training using a minimax game.

The adversarial training scheme is incorporated into GAEs as an additional regularization term in [83]. The overall architecture is shown in Figure 9. Specifically, the encoder is used as the generator, and the discriminator aims to distinguish whether a latent representation is from the generator or from a prior distribution. In this way, the autoencoder is forced to match the prior distribution as regularization. The objective function is:

$$\min_{\mathbf{\Theta}} \mathcal{L}_2 + \alpha\mathcal{L}_{GAN}, \tag{49}$$

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

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \mathbb{E}_{\mathbf{h}\sim p_{\mathbf{h}}} \left[\log \mathcal{D}(\mathbf{h})\right] + \mathbb{E}\left[\log\left(1 - \mathcal{D}\left(\mathcal{G}(\mathbf{F}^V, \mathbf{A})\right)\right)\right], \tag{50}$$

where $\mathcal{G}\left(\mathbf{F}^V, \mathbf{A}\right)$ is the convolutional encoder in Eq. (45), $\mathcal{D}(\cdot)$ is a discriminator with the cross-entropy loss and $p_{\mathbf{h}}$ is the prior distribution. In the paper, a simple Gaussian prior is adopted and experimental results demonstrate the effectiveness of the adversarial training scheme.

5.3.2 Inductive Learning and GCN encoder

Similar to GCNs, GAEs can be applied to the inductive setting if node attributes are incorporated in the encoder. This can be achieved by using GCNs as the encoder such as in [78], [81], [83], or directly learning a mapping function from features as in [80]. Since edge information is only utilized in learning the parameters, the model can be applied to nodes not seen during training. These works also show that, although GCNs and GAEs are based on different architectures, it is possible to use them in conjunction, which we believe is a promising future direction.

5.3.3 Similarity Measures

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

6 Recent Advancements

Besides aforementioned semi-supervised and unsupervised methods, there are some recently advanced categories which we discuss in this section. Main characteristics of methods surveyed are summarized in Table 5.

6.1 Graph Recurrent Neural Networks

Recurrent Neural Networks (RNNs) such as GRU [27] or LSTM [59] are de facto standards in modeling sequential data and are used in GNNs to model states of nodes. In this section, we show that RNNs can also be applied to the graph level. To disambiguate with GNNs which use RNNs in the node level, we refer to these architectures as Graph RNNs.

You et al. [94] apply Graph RNN to the graph generation problem. Specifically, they adopt two RNNs, one for generating new nodes while the other generates edges for the newly added node in an autoregressive manner. They show that such hierarchial RNN architecture can effectively learn from input graphs compared to the traditional rule-based graph generative models while having an acceptable time complexity.

Dynamic Graph Neural Network (DGNN) [95] proposes using time-aware LSTM [100] to learn node representations in dynamic graphs. After a new edge is established, DGNN uses LSTM to update the representation of two interacting nodes as well as their immediate neighbors, i.e. considering one-step propagation effect. The authors show that time-aware LSTM can well model the establishing orders and time intervals of edge formations, which in turn benefits a range of graph applications.

It is also possible to use Graph RNN in conjunction with other architectures, such as GCNs or GAEs. For example, RMGCNN [96] applies LSTM to the results of GCNs to progressively reconstruct the graph as illustrated in Figure 10, aiming to tackle the graph sparsity problem. By using LSTM, the information from

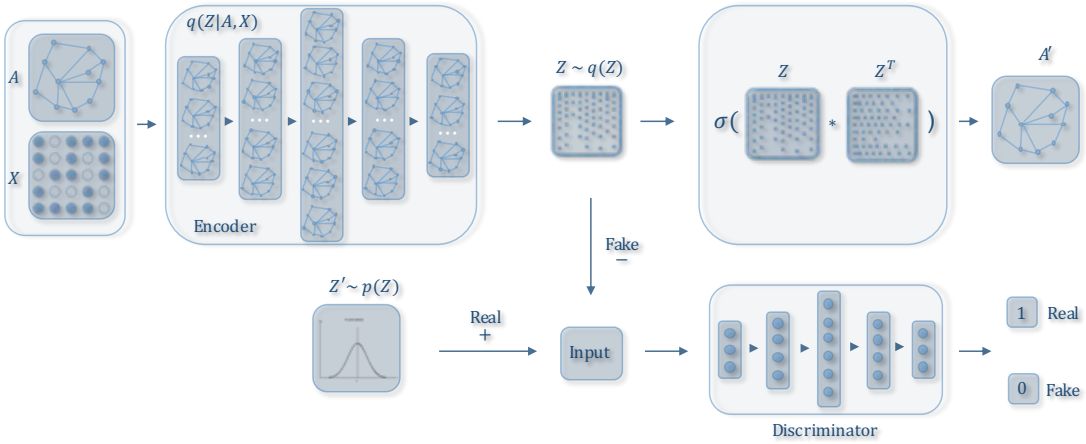

Fig. 9. The framework of ARGA/ARVGA reprinted from [83] with permission. The adversarial training scheme is added into GAEs. The notations are slightly different from the main body where $\mathbf{X}$ and $\mathbf{Z}$ in the figure correspond to $\mathbf{F}^V$ and $\mathbf{H}$ respectively.

TABLE 5
Main Characteristics of Recent Advancements

Category	Method	Type	Scalability	Node Features	Other Improvements
Graph RNNs	You et al. [94]	Unsupervised	No	No	-
	DGNN [95]	Semi-supervised/Unsupervised	Yes	No	-
	RMGCNN [96]	Unsupervised	Yes	-	Convolutional layers
	Dynamic GCN [97]	Semi-supervised	Yes	Yes	Convolutional layers
Graph RL	GCPN [98]	Semi-supervised	No	Yes	Convolutional layers + GAN
	MolGAN [99]	Semi-supervised	No	Yes	Convolutional layers + GAN

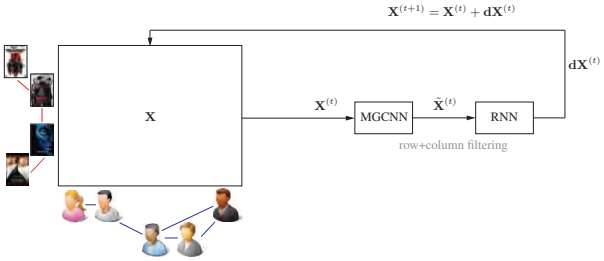

Fig. 10. The framework of RMGCNN reprinted from [96] with permission. RMGCNN adds LSTM into GCN to progressively reconstruct the graph.

different parts of the graph can diffuse across long ranges without needing too many GCN layers. Dynamic GCN [97] applies LSTM to gather results of GCNs of different time slices in dynamic networks, aiming to capture both spatio and temporal graph information.

6.2 Graph Reinforcement Learning

One aspect of deep learning that has not been discussed so far is reinforcement learning (RL), which has been shown effective in AI tasks such as game playing [101]. RL is also known to be able to handle non-differentiable objectives and constraints. In this section, we review graph RL methods.

GCPN [98] utilizes RL for goal-directed molecular graph generation to deal with non-differential objectives and constrains. Specifically, the authors model graph generation as a Markov decision process and regard the generative model as a RL agent operating in the graph generation environment. By resembling agent actions as a link prediction problem, using domain-specific as well as adversarial rewards and using GCNs to learn node representations, GCPN can be trained end-to-end using policy gradient [102]. Experimental results demonstrate the effectiveness of GCPN in various graph generation problems.

A concurrent work, MolGAN [99], takes a similar idea of using RL for generating molecular graphs. Instead of generating the graph by a sequence of actions, MolGAN proposes directly generating the full graph, which works well for small molecules.

7 Conclusion and Discussion

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

Applications. Besides standard graph inference tasks such as node or graph classification, graph-based deep learning methods have also been applied to a wide range of disciplines, such as modeling social influence [103], recommendation [51], [78], [96], chemistry [37], [41], [50], [98], [99], physics [104], [105] , disease or drug prediction [106]–[108], natural language processing (NLP) [109], [110], computer vision [111]–[114], traffic forecasting [115], [116], program induction [117] and solving graph-based NP problems [118], [119].

Though a thorough review of these methods is beyond the scope of this paper due to the diversity of these applications, we list several key inspirations. First, it is important to incorporate domain knowledge into the model, e.g. in constructing the graph or choosing architectures. For example, building a graph based on relative distance may be suitable for traffic forecasting problems, but may not work well for a weather prediction problem because the geographical location is also important. Second, the graph-based model can usually be build on top of other architectures rather than working alone. For example, RNs [32] use GNN as a

relational module to enhance visual question answering problem and [109], [110] adopt GCNs as syntactic constraints for NLP problems. As a result, how to integrate different modules is usually the key challenge. These applications also show that graph-based deep learning not only help mining the rich value underlying existing graph data, but also help advancing other disciplines by naturally modeling relational data as graphs, which greatly generalizes the applicability of graph-based deep learning.

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

- **Different types of graphs**. Due to the extremely varying structures of graph data, the existing methods cannot handle all of them. For example, most methods focus on homogeneous graphs, while heterogenous graphs are seldom studied, especially those containing different modalities like in [120]. Signed networks, where negative edges represent conflicts between nodes, also have unique structures and pose additional challenges to the existing methods [121]. Hypergraphs, representing complex relations between more than two objects [122], are also understudied. An important next step is to design specific deep learning models to handle these different types of graphs.
- **Dynamic graphs**. Most existing methods focus on static graphs. However, many real graphs are dynamic in nature, where nodes, edges and their features can change over time. For example, in social networks, people may establish new social relations, remove old relationships and their characters like hobbies and occupations can change over time. New users may join the network while old users can leave. How to model the evolving characteristics of dynamic graphs and support incrementally updating model parameters largely remains open in the literature. Some preliminary works try to tackle this problem using Graph RNN architectures with encouraging results [95], [97].
- **Interpretability**. Since graphs are often related to other disciplines, interpreting deep learning models for graphs is critical towards decision making problems. For example, in medicine or disease related problems, interpretability is essential in transforming computer experiments to clinical usage. However, interpretability for graph-based deep learning is even more challenging than other black-box models since nodes and edges in the graph are heavily interconnected.
- **Compositionality**. As shown in previous sections, many existing architectures can work together, for example using GCN as a layer in GAEs or Graph RNNs. Besides designing new building blocks, how to composite these architectures in a principled way is an interesting future direction. A recent work, Graph Networks [9], takes the first step and focuses on using a general framework of GNNs and GCNs for relational reasoning problems.

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

ACKNOWLEDGEMENT

We thank Jie Chen, Thomas Kipf, Federico Monti, Shirui Pan, Petar Velickovic, Keyulu Xu, Rex Ying for providing their figures.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.
[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.
[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.
[4] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proceedings of the 4th International Conference on Learning Representations*, 2015.
[5] A.-L. Barabasi, *Network science*. Cambridge university press, 2016.
[6] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, 2013.
[7] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.
[8] C. Zang, P. Cui, and C. Faloutsos, "Beyond sigmoids: The nettide model for social network growth, and its applications," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 2015–2024.
[9] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.
[10] J. B. Lee, R. A. Rossi, S. Kim, N. K. Ahmed, and E. Koh, "Attention models in graphs: A survey," *arXiv preprint arXiv:1807.07984*, 2018.
[11] S. Yan, D. Xu, B. Zhang, H.-J. Zhang, Q. Yang, and S. Lin, "Graph embedding and extensions: A general framework for dimensionality reduction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 1, pp. 40–51, 2007.
[12] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.
[13] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE Transactions on Knowledge and Data Engineering*, 2018.
[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, 1986.
[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proceedings of the 3rd International Conference on Learning Representations*, 2014.
[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
[17] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding," in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2017.
[18] J. Leskovec and J. J. Mcauley, "Learning to discover social circles in ego networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 539–547.
[19] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *IEEE International Joint Conference on Neural Networks Proceedings*, vol. 2. IEEE, 2005, pp. 729–734.
[20] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.
[21] P. Frasconi, M. Gori, and A. Sperduti, "A general framework for adaptive processing of data structures," *IEEE transactions on Neural Networks*, vol. 9, no. 5, pp. 768–786, 1998.
[22] M. J. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *The computer journal*, vol. 7, no. 2, pp. 155–162, 1964.
[23] L. B. Almeida, "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment." in *Proceedings, 1st First International Conference on Neural Networks*. IEEE, 1987.
[24] F. J. Pineda, "Generalization of back-propagation to recurrent neural networks," *Physical Review Letters*, vol. 59, no. 19, p. 2229, 1987.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[49] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. V. D. Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," pp. 593–607, 2018.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[73] R. Giryes, G. Sapiro, and A. M. Bronstein, "Deep neural networks with random gaussian weights: A universal classification strategy?" *IEEE Trans. Signal Processing*, vol. 64, no. 13, pp. 3444–3457, 2016.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[111] V. Garcia and J. Bruna, "Few-shot learning with graph neural networks," in *Proceedings of the 7th International Conference on Learning Representations*, 2018.

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

[113] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun, "3d graph neural networks for rgbd semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[114] K. Marino, R. Salakhutdinov, and A. Gupta, "The more you know: Using knowledge graphs for image classification," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 20–28.

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

[116] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *Proceedings of the 7th International Conference on Learning Representations*, 2018.

[117] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," in *Proceedings of the 7th International Conference on Learning Representations*, 2018.

[118] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *Advances in Neural Information Processing Systems*, 2018, pp. 536–545.

[119] M. O. Prates, P. H. Avelar, H. Lemos, L. Lamb, and M. Vardi, "Learning to solve np-complete problems-a graph neural network for the decision tsp," *arXiv preprint arXiv:1809.02721*, 2018.

[120] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang, "Heterogeneous network embedding via deep architectures," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 119–128.

[121] T. Derr, Y. Ma, and J. Tang, "Signed graph convolutional network," in *Data Mining (ICDM), 2018 IEEE International Conference on*. IEEE, 2018, pp. 559–568.

[122] K. Tu, P. Cui, X. Wang, F. Wang, and W. Zhu, "Structural deep embedding for hyper-networks," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.