# Graph Neural Networks: A Review of Methods and Applications

Jie Zhou*, Ganqu Cui*, Zhengyan Zhang*, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, Maosong Sun



**Abstract**—Lots of learning tasks require dealing with graph data which contains rich relation information among elements. Modeling physics system, learning molecular fingerprints, predicting protein interface, and classifying diseases require a model to learn from graph inputs. In other domains such as learning from non-structural data like texts and images, reasoning on extracted structures, like the dependency tree of sentences and the scene graph of images, is an important research topic which also needs graph reasoning models. Graph neural networks (GNNs) are connectionist models that capture the dependence of graphs via message passing between the nodes of graphs. Unlike standard neural networks, graph neural networks retain a state that can represent information from its neighborhood with arbitrary depth. Although the primitive GNNs have been found difficult to train for a fixed point, recent advances in network architectures, optimization techniques, and parallel computation have enabled successful learning with them. In recent years, systems based on variants of graph neural networks such as graph convolutional network (GCN), graph attention network (GAT), gated graph neural network (GGNN) have demonstrated ground-breaking performance on many tasks mentioned above. In this survey, we provide a detailed review over existing graph neural network models, systematically categorize the applications, and propose four open problems for future research.

**Index Terms**—Deep Learning, Graph Neural Network

## 1 Introduction

Graphs are a kind of data structure which models a set of objects (nodes) and their relationships (edges). Recently, researches of analyzing graphs with machine learning have been receiving more and more attention because of the great expressive power of graphs, i.e. graphs can be used as denotation of a large number of systems across various areas including social science (social networks) [1], [2], natural science (physical systems [3], [4] and protein-protein interaction networks [5]), knowledge graphs [6] and many other research areas [7]. As a unique non-Euclidean data structure for machine learning, graph analysis focuses on node classification, link prediction, and clustering. Graph neural networks (GNNs) are deep learning based methods that operate on graph domain. Due to its convincing performance and high interpretability, GNN has been a widely applied graph analysis method recently. In the following paragraphs, we will illustrate the fundamental motivations of graph neural networks.

The first motivation of GNNs roots in convolutional neural networks (CNNs) [8]. CNNs have the ability to extract multi-scale localized spatial features and compose them to construct highly expressive representations, which led to breakthroughs in almost all machine learning areas and started the new era of deep learning [9]. As we are going deeper into CNNs and graphs, we found the keys of CNNs: local connection, shared weights and the use of multi-layer [9]. These are also of great importance in solving problems of graph domain, because 1) graphs are the most typical locally connected structure. 2) shared weights reduce the computational cost compared with traditional spectral graph theory [10]. 3) multi-layer structure is the key to deal with hierarchical patterns, which captures the features of various sizes. However, CNNs can only operate on regular Euclidean data like images (2D grid) and text (1D sequence) while these data structures can be regarded as instances of graphs. Therefore, it is straightforward to think of finding the generalization of CNNs to graphs. As shown in Fig. 1, it is hard to define localized convolutional filters and pooling operators, which hinders the transformation of CNN from Euclidean domain to non-Euclidean domain.

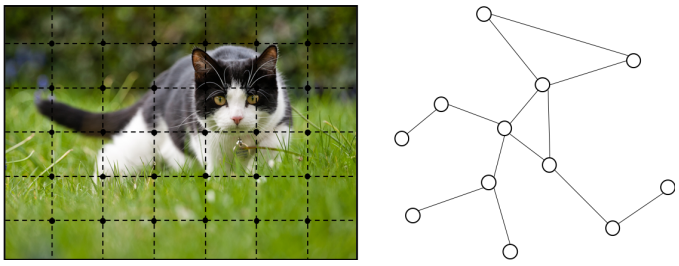


Fig. 1. Left: image in Euclidean space. Right: graph in non-Euclidean space

* *indicates equal contribution.*

- *Jie Zhou, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu (corresponding author) and Maosong Sun are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: {zhoujie18, zhangzhengyan14, cheng-ya14}@mails.tsinghua.edu.cn, {sms, liuzy}@tsinghua.edu.cn*
- *Ganqu Cui is with the Department of Physics, Tsinghua University, Beijing 100084, China. Email: cgq15@mails.tsinghua.edu.cn*
- *Lifeng Wang, Changcheng Li are with the Tencent Incorporation, Shenzhen, China. Email:{fandywang, harrychli}@tencent.com*

The other motivation comes from *graph embedding* [11]–[15], which learns to represent graph nodes, edges or subgraphs in low-dimensional vectors. In the field of graph analysis, traditional machine learning approaches usually rely on hand engineered features and are limited by its inflexibility and high cost. Following the idea of *representation learning* and the success of word embedding [16], DeepWalk [17], which is regarded as the first graph embedding method based on representation learning, applies SkipGram model [16] on the generated random walks. Similar approaches such as node2vec [18], LINE [19] and TADW [20] also achieved breakthroughs. However, these methods suffer two severe drawbacks [12]. First, no parameters are shared between nodes in the encoder, which leads to computationally inefficiency, since it means the number of parameters grows linearly with the number of nodes. Second, the direct embedding methods lack the ability of generalization, which means they cannot deal with dynamic graphs or generalize to new graphs.

Based on CNNs and graph embedding, graph neural networks (GNNs) are proposed to collectively aggregate information from graph structure. Thus they can model input and/or output consisting of elements and their dependency. Further, graph neural network can simultaneously model the diffusion process on the graph with the RNN kernel.

In the following part, we explain the fundamental reasons why graph neural networks are worth investigating. Firstly, the standard neural networks like CNNs and RNNs cannot handle the graph input properly in that they stack the feature of nodes by a specific order. However, there isn't a natural order of nodes in the graph. To present a graph completely, we should traverse all the possible orders as the input of the model like CNNs and RNNs, which is very redundant when computing. To solve this problem, GNNs propagate on each node respectively, ignoring the input order of nodes. In other words, the output of GNNs is invariant for the input order of nodes. Secondly, an edge in a graph represents the information of dependency between two nodes. In the standard neural networks, the dependency information is just regarded as the feature of nodes. However, GNNs can do propagation guided by the graph structure instead of using it as part of features. Generally, GNNs update the hidden state of nodes by a weighted sum of the states of their neighborhood. Thirdly, reasoning is a very important research topic for high-level artificial intelligence and the reasoning process in human brain is almost based on the graph which is extracted from daily experience. The standard neural networks have shown the ability to generate synthetic images and documents by learning the distribution of data while they still cannot learn the reasoning graph from large experimental data. However, GNNs explore to generate the graph from non-structural data like scene pictures and story documents, which can be a powerful neural model for further high-level AI. Recently, it has been proved that an untrained GNN with a simple architecture also perform well [21].

There exist several comprehensive reviews on graph neural networks. [22] proposed a unified framework, MoNet, to generalize CNN architectures to non-Euclidean domains (graphs and manifolds) and the framework could generalize several spectral methods on graphs [2], [23] as well as some models on manifolds [24], [25]. [26] provides a thorough review of geometric deep learning, which presents its problems, difficulties, solutions, applications and future directions. [22] and [26] focus on generalizing convolutions to graphs or manifolds, however in this paper we only focus on problems defined on graphs and we also investigate other mechanisms used in graph neural networks such as gate mechanism, attention mechanism and skip connection. [27] proposed the message passing neural network (MPNN) which could generalize several graph neural network and graph convolutional network approaches. [28] proposed the non-local neural network (NLNN) which unifies several "self-attention"-style methods. However, the model is not explicitly defined on graphs in the original paper. Focusing on specific application domains, [27] and [28] only give examples of how to generalize other models using their framework and they do not provide a review over other graph neural network models. [29] provides a review over graph attention models. [30] proposed the graph network (GN) framework which has a strong capability to generalize other models. However, the graph network model is highly abstract and [30] only gives a rough classification of the applications.

[31] and [32] are the most up-to-date survey papers on GNNs and they mainly focus on models of GNN. [32] categorizes GNNs into five groups: graph convolutional networks, graph attention networks, graph auto-encoders, graph generative networks and graph spatial-temporal networks. Our paper has a different taxonomy with [32]. We introduce graph convolutional networks and graph attention networks in Section 2.2.2 as they contribute to the propagation step. We present the graph spatial-temporal networks in Section 2.2.1 as the models are usually used on dynamic graphs. We introduce graph auto-encoders in Sec 2.2.3 as they are trained in an unsupervised fashion. And finally, we introduce graph generative networks in applications of graph generation (see Section 3.3.1).

In this paper, we provide a thorough review of different graph neural network models as well as a systematic taxonomy of the applications. To summarize, this paper presents an extensive survey of graph neural networks with the following contributions.

- We provide a detailed review over existing graph neural network models. We introduce the original model, its variants and several general frameworks. We examine various models in this area and provide a unified representation to present different propagation steps in different models. One can easily make a distinction between different models using our representation by recognizing corresponding aggregators and updaters.
- We systematically categorize the applications and divide the applications into structural scenarios, non-structural scenarios and other scenarios. We present several major applications and their corresponding methods for each scenario.
- We propose four open problems for future research.

Graph neural networks suffer from over-smoothing and scaling problems. There are still no effective methods for dealing with dynamic graphs as well as modeling non-structural sensory data. We provide a thorough analysis of each problem and propose future research directions.

The rest of this survey is organized as follows. In Sec. 2, we introduce various models in the graph neural network family. We first introduce the original framework and its limitations. Then we present its variants that try to release the limitations. And finally, we introduce several general frameworks proposed recently. In Sec. 3, we will introduce several major applications of graph neural networks applied to structural scenarios, non-structural scenarios and other scenarios. In Sec. 4, we propose four open problems of graph neural networks as well as several future research directions. And finally, we conclude the survey in Sec. 5.

## 2 MODELS

Graph neural networks are useful tools on non-Euclidean structures and there are various methods proposed in the literature trying to improve the model's capability.

In Sec 2.1, we describe the original graph neural networks proposed in [33]. We also list the limitations of the original GNN in representation capability and training efficiency. In Sec 2.2 we introduce several variants of graph neural networks aiming to release the limitations. These variants operate on graphs with different types, utilize different propagation functions and advanced training methods. In Sec 2.3 we present three general frameworks which could generalize and extend several lines of work. In detail, the message passing neural network (MPNN) [27] unifies various graph neural network and graph convolutional network approaches; the non-local neural network (NLNN) [28] unifies several "self-attention"-style methods. And the graph network(GN) [30] could generalize almost every graph neural network variants mentioned in this paper.

Before going further into different sections, we give the notations that will be used throughout the paper. The detailed descriptions of the notations could be found in Table 1.

TABLE 1
Notations used in this paper.

| Notations | Descriptions |
|---|---|
| $\mathbb{R}^m$ | $m$-dimensional Euclidean space |
| $a, \mathbf{a}, \mathbf{A}$ | Scalar, vector, matrix |
| $\mathbf{A}^T$ | Matrix transpose |
| $\mathbf{I}_N$ | Identity matrix of dimension $N$ |
| $\mathbf{g}_\theta \star \mathbf{x}$ | Convolution of $\mathbf{g}_\theta$ and $\mathbf{x}$ |
| $N$ | Number of nodes in the graph |
| $N^v$ | Number of nodes in the graph |
| $N^e$ | Number of edges in the graph |
| $\mathcal{N}_v$ | Neighborhood set of node $v$ |
| $\mathbf{a}_v^t$ | Vector $\mathbf{a}$ of node $v$ at time step $t$ |
| $\mathbf{h}_v$ | Hidden state of node $v$ |
| $\mathbf{h}_v^t$ | Hidden state of node $v$ at time step $t$ |
| $\mathbf{e}_{vw}$ | Features of edge from node $v$ to $w$ |
| $\mathbf{e}_k$ | Features of edge with label $k$ |
| $\mathbf{o}_v^t$ | Output of node $v$ |
| $\mathbf{W}^i, \mathbf{U}^i, \mathbf{W}^o, \mathbf{U}^o, ...$ | Matrices for computing $\mathbf{i}, \mathbf{o}, ...$ |
| $\mathbf{b}^i, \mathbf{b}^o, ...$ | Vectors for computing $\mathbf{i}, \mathbf{o}, ...$ |
| $\sigma$ | The logistic sigmoid function |
| $\rho$ | An alternative non-linear function |
| $\tanh$ | The hyperbolic tangent function |
| LeakyReLU | The LeakyReLU function |
| $\odot$ | Element-wise multiplication operation |
| $\|$ | Vector concatenation |

### 2.1 Graph Neural Networks

The concept of graph neural network (GNN) was first proposed in [33], which extended existing neural networks for processing the data represented in graph domains. In a graph, each node is naturally defined by its features and the related nodes. The target of GNN is to learn a state embedding $\mathbf{h}_v \in \mathbb{R}^s$ which contains the information of neighborhood for each node. The state embedding $\mathbf{h}_v$ is an $s$-dimension vector of node $v$ and can be used to produce an output $\mathbf{o}_v$ such as the node label. Let $f$ be a parametric function, called *local transition function*, that is shared among all nodes and updates the node state according to the input neighborhood. And let $g$ be the *local output function* that describes how the output is produced. Then, $\mathbf{h}_v$ and $\mathbf{o}_v$ are defined as follows:

$$\mathbf{h}_v = f(\mathbf{x}_v, \mathbf{x}_{co[v]}, \mathbf{h}_{ne[v]}, \mathbf{x}_{ne[v]}) \tag{1}$$

$$\mathbf{o}_v = g(\mathbf{h}_v, \mathbf{x}_v) \tag{2}$$

where $\mathbf{x}_v, \mathbf{x}_{co[v]}, \mathbf{h}_{ne[v]}, \mathbf{x}_{ne[v]}$ are the features of $v$, the features of its edges, the states, and the features of the nodes in the neighborhood of $v$, respectively.

Let $\mathbf{H}$, $\mathbf{O}$, $\mathbf{X}$, and $\mathbf{X}_N$ be the vectors constructed by stacking all the states, all the outputs, all the features, and all the node features, respectively. Then we have a compact form as:

$$\mathbf{H} = F(\mathbf{H}, \mathbf{X}) \tag{3}$$

$$\mathbf{O} = G(\mathbf{H}, \mathbf{X}_N) \tag{4}$$

where $F$, the *global transition function*, and $G$, the *global output function* are stacked versions of $f$ and $g$ for all nodes in a graph, respectively. The value of $\mathbf{H}$ is the fixed point of Eq. 3 and is uniquely defined with the assumption that $F$ is a contraction map.

With the suggestion of Banach's fixed point theorem [34], GNN uses the following classic iterative scheme for computing the state:

$$\mathbf{H}^{t+1} = F(\mathbf{H}^t, \mathbf{X}) \tag{5}$$

where $\mathbf{H}^t$ denotes the $t$-th iteration of $\mathbf{H}$. The dynamical system Eq. 5 converges exponentially fast to the solution of Eq. 3 for any initial value $\mathbf{H}(0)$. Note that the computations described in $f$ and $g$ can be interpreted as the feedforward

neural networks.

When we have the framework of GNN, the next question is how to learn the parameters of $f$ and $g$. With the target information ($\mathbf{t}_v$ for a specific node) for the supervision, the loss can be written as follow:

$$loss = \sum_{i=1}^{p}(\mathbf{t}_i - \mathbf{o}_i) \tag{6}$$

where $p$ is the number of supervised nodes. The learning algorithm is based on a gradient-descent strategy and is composed of the following steps.

- The states $\mathbf{h}_v^t$ are iteratively updated by Eq. 1 until a time $T$. They approach the fixed point solution of Eq. 3: $\mathbf{H}(T) \approx \mathbf{H}$.
- The gradient of weights $\mathbf{W}$ is computed from the loss.
- The weights $\mathbf{W}$ are updated according to the gradient computed in the last step.

**Limitations** Though experimental results showed that GNN is a powerful architecture for modeling structural data, there are still several limitations of the original GNN. Firstly, it is inefficient to update the hidden states of nodes iteratively for the fixed point. If relaxing the assumption of the fixed point, we can design a multi-layer GNN to get a stable representation of node and its neighborhood. Secondly, GNN uses the same parameters in the iteration while most popular neural networks use different parameters in different layers, which serve as a hierarchical feature extraction method. Moreover, the update of node hidden states is a sequential process which can benefit from the RNN kernel like GRU and LSTM. Thirdly, there are also some informative features on the edges which cannot be effectively modeled in the original GNN. For example, the edges in the knowledge graph have the type of relations and the message propagation through different edges should be different according to their types. Besides, how to learn the hidden states of edges is also an important problem. Lastly, it is unsuitable to use the fixed points if we focus on the representation of nodes instead of graphs because the distribution of representation in the fixed point will be much smooth in value and less informative for distinguishing each node.

## 2.2 Variants of Graph Neural Networks

In this subsection, we present several variants of graph neural networks. Sec 2.2.1 focuses on variants operating on different graph types. These variants extend the representation capability of the original model. Sec 2.2.2 lists several modifications (convolution, gate mechanism, attention mechanism and skip connection) on the propagation step and these models could learn representations with higher quality. Sec 2.2.3 describes variants using advanced training methods which improve the training efficiency. An overview of different variants of graph neural networks could be found in Fig. 2.

### 2.2.1 Graph Types

In the original GNN [33], the input graph consists of nodes with label information and undirected edges, which is the simplest graph format. However, there are many variants of graphs in the world. In this subsection, we will introduce some methods designed to model different kinds of graphs.

**Directed Graphs** The first variant of graph is directed graph. Undirected edge which can be treated as two directed edges shows that there is a relation between two nodes. However, directed edges can bring more information than undirected edges. For example, in a knowledge graph where the edge starts from the head entity and ends at the tail entity, the head entity is the parent class of the tail entity, which suggests we should treat the information propagation process from parent classes and child classes differently. DGP [35] uses two kinds of weight matrix, $\mathbf{W}_p$ and $\mathbf{W}_c$, to incorporate more precise structural information. The propagation rule is shown as follows:

$$\mathbf{H}^t = \sigma(\mathbf{D}_p^{-1}\mathbf{A}_p\sigma(\mathbf{D}_c^{-1}\mathbf{A}_c\mathbf{H}^{t-1}\mathbf{W}_c)\mathbf{W}_p) \tag{7}$$

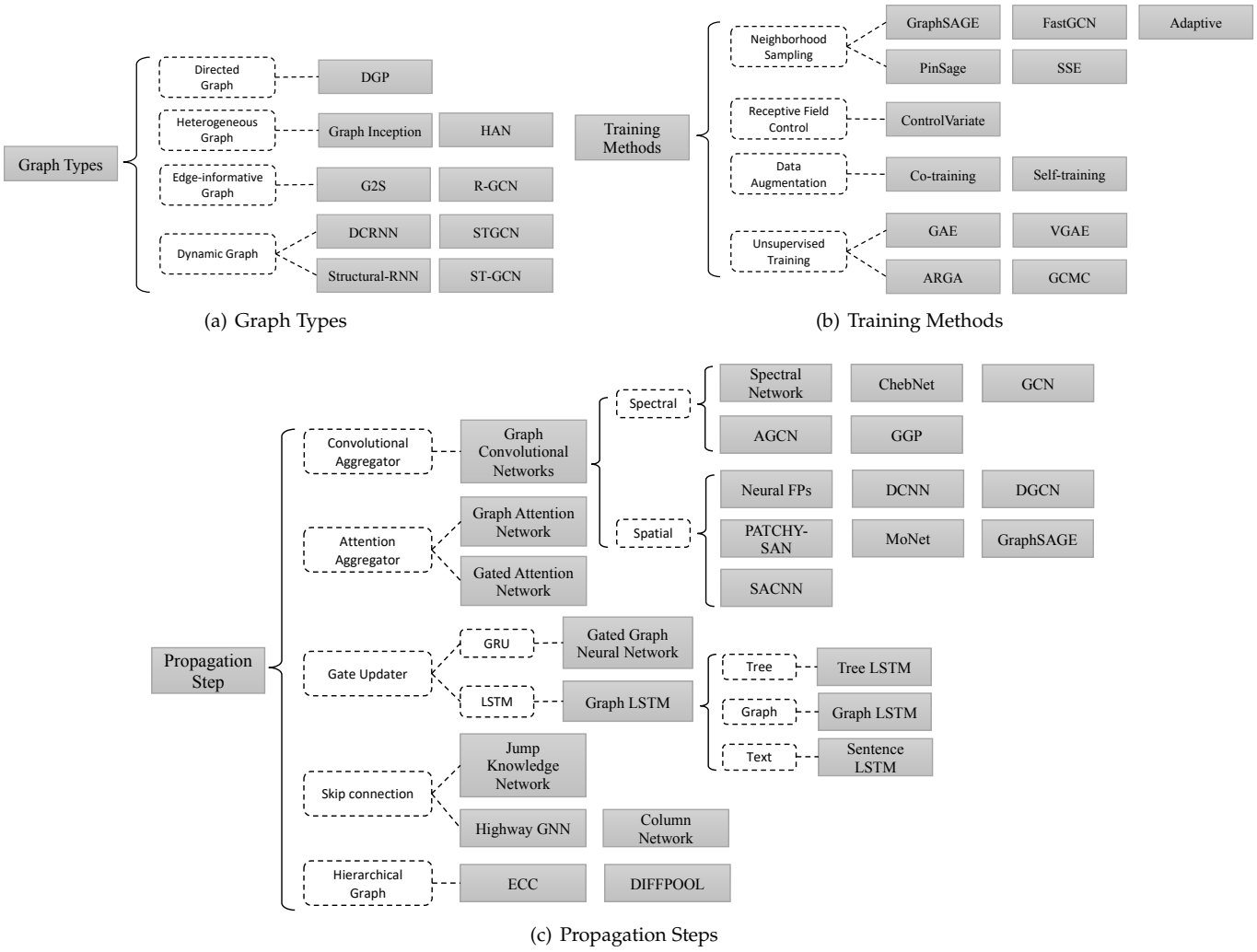
where $\mathbf{D}_p^{-1}\mathbf{A}_p$, $\mathbf{D}_c^{-1}\mathbf{A}_c$ are the normalized adjacency matrix for parents and children respectively.

**Heterogeneous Graphs** The second variant of graph is heterogeneous graph, where there are several kinds of nodes. The simplest way to process heterogeneous graph is to convert the type of each node to a one-hot feature vector which is concatenated with the original feature. What's more, GraphInception [36] introduces the concept of metapath into the propagation on the heterogeneous graph. With metapath, we can group the neighbors according to their node types and distances. For each neighbor group, GraphInception treats it as a sub-graph in a homogeneous graph to do propagation and concatenates the propagation results from different homogeneous graphs to do a collective node representation. Recently, [37] proposed the heterogeneous graph attention network (HAN) which utilizes node-level and semantic-level attentions. And the model have the ability to consider node importance and meta-paths simultaneously.

**Graphs with Edge Information** In another variant of graph, each edge has additional information like the weight or the type of the edge. We list two ways to handle this kind of graphs: Firstly, we can convert the graph to a bipartite graph where the original edges also become nodes and one original edge is split into two new edges which means there are two new edges between the edge node and begin/end nodes. The encoder of G2S [38] uses the following aggregation function for neighbors:

$$\mathbf{h}_v^t = \rho(\frac{1}{|\mathcal{N}_v|}\sum_{u\in\mathcal{N}_v}\mathbf{W}_r(\mathbf{r}_v^t \odot \mathbf{h}_u^{t-1}) + \mathbf{b}_r) \tag{8}$$

where $\mathbf{W}_r$ and $\mathbf{b}_r$ are the propagation parameters for different types of edges (relations). Secondly, we can adapt different weight matrices for the propagation on different kinds of edges. When the number of relations is very large, r-GCN [39] introduces two kinds of regularization to reduce the number of parameters for modeling amounts of relations: *basis*- and *block-diagonal*-decomposition. With the basis




(a) Graph Types (b) Training Methods

(c) Propagation Steps

Fig. 2. An overview of variants of graph neural networks.

decomposition, each $\mathbf{W}_r$ is defined as follows:

$$\mathbf{W}_r = \sum_{1}^{B} a_{rb}\mathbf{V}_b \tag{9}$$

Here each $\mathbf{W}_r$ is a linear combination of basis transformations $\mathbf{V}_b \in \mathbb{R}^{d_{in}\times d_{out}}$ with coefficients $a_{rb}$. In the block-diagonal decomposition, r-GCN defines each $\mathbf{W}_r$ through the direct sum over a set of low-dimensional matrices, which needs more parameters than the first one.

**Dynamic Graphs** Another variant of graph is dynamic graph, which has static graph structure and dynamic input signals. To capture both kind of information, DCRNN [40] and STGCN [41] first collect spatial information by GNNs, then feed the outputs into a sequence model like sequence-to-sequence model or CNNs. Differently, Structural-RNN [42] and ST-GCN [43] collect spatial and temporal messages at the same time. They extend static graph structure with temporal connections so they can apply traditional GNNs on the extended graphs.

### 2.2.2 Propagation Types

The propagation step and output step are of vital importance in the model to obtain the hidden states of nodes (or edges). As we list below, there are several major modifications in the propagation step from the original graph neural network model while researchers usually follow a simple feed-forward neural network setting in the output step. The comparison of different variants of GNN could be found in Table 2. The variants utilize different aggregators to gather information from each node's neighbors and specific updaters to update nodes' hidden states.

**Convolution.** There is an increasing interest in generalizing convolutions to the graph domain. Advances in this direction are often categorized as spectral approaches and non-spectral (spatial) approaches.

Spectral approaches work with a spectral representation of the graphs.

*Spectral Network.* [45] proposed the spectral network. The convolution operation is defined in the Fourier domain by computing the eigendecomposition of the graph Laplacian. The operation can be defined as the multiplication of a sig-

TABLE 2
Different variants of graph neural networks.

| Name | Variant | Aggregator | Updater |
|---|---|---|---|
| Spectral Methods | ChebNet | $\mathbf{N}_k = \mathbf{T}_k(\tilde{\mathbf{L}})\mathbf{X}$ | $\mathbf{H} = \sum_{k=0}^{K} \mathbf{N}_k \mathbf{\Theta}_k$ |
| | $1^{st}$-order model | $\mathbf{N}_0 = \mathbf{X}$<br>$\mathbf{N}_1 = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}$ | $\mathbf{H} = \mathbf{N}_0\mathbf{\Theta}_0 + \mathbf{N}_1\mathbf{\Theta}_1$ |
| | Single parameter | $\mathbf{N} = (\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{X}$ | $\mathbf{H} = \mathbf{N}\mathbf{\Theta}$ |
| | GCN | $\mathbf{N} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}$ | $\mathbf{H} = \mathbf{N}\mathbf{\Theta}$ |
| Non-spectral Methods | Neural FPs | $\mathbf{h}^t_{\mathcal{N}_v} = \mathbf{h}^{t-1}_v + \sum_{k=1}^{\mathcal{N}_v} \mathbf{h}^{t-1}_k$ | $\mathbf{h}^t_v = \sigma(\mathbf{h}^t_{\mathcal{N}_v}\mathbf{W}^{\mathcal{N}_v}_L)$ |
| | DCNN | Node classification:<br>$\mathbf{N} = \mathbf{P}^*\mathbf{X}$<br>Graph classification:<br>$\mathbf{N} = 1^T_N\mathbf{P}^*\mathbf{X}/N$ | $\mathbf{H} = f\left(\mathbf{W}^c \odot \mathbf{N}\right)$ |
| | GraphSAGE | $\mathbf{h}^t_{\mathcal{N}_v} = \mathrm{AGGREGATE}_t\left(\{\mathbf{h}^{t-1}_u, \forall u \in \mathcal{N}_v\}\right)$ | $\mathbf{h}^t_v = \sigma\left(\mathbf{W}^t \cdot [\mathbf{h}^{t-1}_v \| \mathbf{h}^t_{\mathcal{N}_v}]\right)$ |
| Graph Attention Networks | GAT | $\alpha_{vk} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \| \mathbf{W}\mathbf{h}_k]\right)\right)}{\sum_{j\in\mathcal{N}_v}\exp(\mathrm{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \| \mathbf{W}\mathbf{h}_j]))}$<br>$\mathbf{h}^t_{\mathcal{N}_v} = \sigma\left(\sum_{k\in\mathcal{N}_v}\alpha_{vk}\mathbf{W}\mathbf{h}_k\right)$<br>Multi-head concatenation:<br>$\mathbf{h}^t_{\mathcal{N}_v} = \Big\|_{m=1}^{M}\sigma\left(\sum_{k\in\mathcal{N}_v}\alpha^m_{vk}\mathbf{W}^m\mathbf{h}_k\right)$<br>Multi-head average:<br>$\mathbf{h}^t_{\mathcal{N}_v} = \sigma\left(\frac{1}{M}\sum_{m=1}^{M}\sum_{k\in\mathcal{N}_v}\alpha^m_{vk}\mathbf{W}^m\mathbf{h}_k\right)$ | $\mathbf{h}^t_v = \mathbf{h}^t_{\mathcal{N}_v}$ |
| Gated Graph Neural Networks | GGNN | $\mathbf{h}^t_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{h}^{t-1}_k + \mathbf{b}$ | $\mathbf{z}^t_v = \sigma(\mathbf{W}^z\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{U}^z\mathbf{h}^{t-1}_v)$<br>$\mathbf{r}^t_v = \sigma(\mathbf{W}^r\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{U}^r\mathbf{h}^{t-1}_v)$<br>$\widetilde{\mathbf{h}^t_v} = \tanh(\mathbf{W}\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{U}(\mathbf{r}^t_v \odot \mathbf{h}^{t-1}_v))$<br>$\mathbf{h}^t_v = (1 - \mathbf{z}^t_v) \odot \mathbf{h}^{t-1}_v + \mathbf{z}^t_v \odot \widetilde{\mathbf{h}^t_v}$ |
| Graph LSTM | Tree LSTM (Child sum) | $\mathbf{h}^t_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{h}^{t-1}_k$ | $\mathbf{i}^t_v = \sigma(\mathbf{W}^i\mathbf{x}^t_v + \mathbf{U}^i\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{b}^i)$<br>$\mathbf{f}^t_{vk} = \sigma\left(\mathbf{W}^f\mathbf{x}^t_v + \mathbf{U}^f\mathbf{h}^{t-1}_k + \mathbf{b}^f\right)$<br>$\mathbf{o}^t_v = \sigma(\mathbf{W}^o\mathbf{x}^t_v + \mathbf{U}^o\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{b}^o)$<br>$\mathbf{u}^t_v = \tanh(\mathbf{W}^u\mathbf{x}^t_v + \mathbf{U}^u\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{b}^u)$<br>$\mathbf{c}^t_v = \mathbf{i}^t_v \odot \mathbf{u}^t_v + \sum_{k\in\mathcal{N}_v}\mathbf{f}^t_{vk} \odot \mathbf{c}^{t-1}_k$<br>$\mathbf{h}^t_v = \mathbf{o}^t_v \odot \tanh(\mathbf{c}^t_v)$ |
| | Tree LSTM (N-ary) | $\mathbf{h}^{ti}_{\mathcal{N}_v} = \sum_{l=1}^{K}\mathbf{U}^i_l\mathbf{h}^{t-1}_{vl}$<br>$\mathbf{h}^{tf}_{\mathcal{N}_v k} = \sum_{l=1}^{K}\mathbf{U}^f_{kl}\mathbf{h}^{t-1}_{vl}$<br>$\mathbf{h}^{to}_{\mathcal{N}_v} = \sum_{l=1}^{K}\mathbf{U}^o_l\mathbf{h}^{t-1}_{vl}$<br>$\mathbf{h}^{tu}_{\mathcal{N}_v} = \sum_{l=1}^{K}\mathbf{U}^u_l\mathbf{h}^{t-1}_{vl}$ | $\mathbf{i}^t_v = \sigma(\mathbf{W}^i\mathbf{x}^t_v + \mathbf{h}^{ti}_{\mathcal{N}_v} + \mathbf{b}^i)$<br>$\mathbf{f}^t_{vk} = \sigma(\mathbf{W}^f\mathbf{x}^t_v + \mathbf{h}^{tf}_{\mathcal{N}_v k} + \mathbf{b}^f)$<br>$\mathbf{o}^t_v = \sigma(\mathbf{W}^o\mathbf{x}^t_v + \mathbf{h}^{to}_{\mathcal{N}_v} + \mathbf{b}^o)$<br>$\mathbf{u}^t_v = \tanh(\mathbf{W}^u\mathbf{x}^t_v + \mathbf{h}^{tu}_{\mathcal{N}_v} + \mathbf{b}^u)$<br>$\mathbf{c}^t_v = \mathbf{i}^t_v \odot \mathbf{u}^t_v + \sum_{l=1}^{K}\mathbf{f}^t_{vl} \odot \mathbf{c}^{t-1}_{vl}$<br>$\mathbf{h}^t_v = \mathbf{o}^t_v \odot \tanh(\mathbf{c}^t_v)$ |
| | Graph LSTM in [44] | $\mathbf{h}^{ti}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{U}^i_{m(v,k)}\mathbf{h}^{t-1}_k$<br>$\mathbf{h}^{to}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{U}^o_{m(v,k)}\mathbf{h}^{t-1}_k$<br>$\mathbf{h}^{tu}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{U}^u_{m(v,k)}\mathbf{h}^{t-1}_k$ | $\mathbf{i}^t_v = \sigma(\mathbf{W}^i\mathbf{x}^t_v + \mathbf{h}^{ti}_{\mathcal{N}_v} + \mathbf{b}^i)$<br>$\mathbf{f}^t_{vk} = \sigma(\mathbf{W}^f\mathbf{x}^t_v + \mathbf{U}^f_{m(v,k)}\mathbf{h}^{t-1}_k + \mathbf{b}^f)$<br>$\mathbf{o}^t_v = \sigma(\mathbf{W}^o\mathbf{x}^t_v + \mathbf{h}^{to}_{\mathcal{N}_v} + \mathbf{b}^o)$<br>$\mathbf{u}^t_v = \tanh(\mathbf{W}^u\mathbf{x}^t_v + \mathbf{h}^{tu}_{\mathcal{N}_v} + \mathbf{b}^u)$<br>$\mathbf{c}^t_v = \mathbf{i}^t_v \odot \mathbf{u}^t_v + \sum_{k\in\mathcal{N}_v}\mathbf{f}^t_{vk} \odot \mathbf{c}^{t-1}_k$<br>$\mathbf{h}^t_v = \mathbf{o}^t_v \odot \tanh(\mathbf{c}^t_v)$ |

nal $\mathbf{x} \in \mathbb{R}^N$ (a scalar for each node) with a filter $\mathbf{g}_\theta$ =diag($\theta$) parameterized by $\theta \in \mathbb{R}^N$ :

$$\mathbf{g}_\theta \star \mathbf{x} = \mathbf{U}\mathbf{g}_\theta(\mathbf{\Lambda})\mathbf{U}^T\mathbf{x} \tag{10}$$

where $\mathbf{U}$ is the matrix of eigenvectors of the normalized graph Laplacian $\mathbf{L} = \mathbf{I}_N - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$ ($\mathbf{D}$ is the degree matrix and $\mathbf{A}$ is the adjacency matrix of the graph), with a diagonal matrix of its eigenvalues $\mathbf{\Lambda}$.

This operation results in potentially intense computations and non-spatially localized filters. [46] attempts to make the spectral filters spatially localized by introducing a parameterization with smooth coefficients.

*ChebNet.* [47] suggests that $\mathbf{g}_\theta(\mathbf{\Lambda})$ can be approximated by a truncated expansion in terms of Chebyshev polynomials $\mathbf{T}_k(x)$ up to $K^{th}$ order. Thus the operation is:

$$\mathbf{g}_\theta \star \mathbf{x} \approx \sum_{k=0}^{K} \theta_k \mathbf{T}_k(\tilde{\mathbf{L}})\mathbf{x} \tag{11}$$

with $\tilde{\mathbf{L}} = \frac{2}{\lambda_{max}}\mathbf{L} - \mathbf{I}_N$. $\lambda_{max}$ denotes the largest eigenvalue of $\mathbf{L}$. $\theta \in \mathbb{R}^K$ is now a vector of Chebyshev coefficients. The Chebyshev polynomials are defined as $\mathbf{T}_k(\mathbf{x}) = 2\mathbf{x}\mathbf{T}_{k-1}(\mathbf{x}) - \mathbf{T}_{k-2}(\mathbf{x})$, with $\mathbf{T}_0(\mathbf{x}) = 1$ and $\mathbf{T}_1(\mathbf{x}) = \mathbf{x}$. It can be observed that the operation is $K$-localized since it is a $K^{th}$-order polynomial in the Laplacian. [48] proposed the ChebNet. It uses this $K$-localized convolution to define a convolutional neural network which could remove the need to compute the eigenvectors of the Laplacian.

*GCN.* [2] limits the layer-wise convolution operation to $K = 1$ to alleviate the problem of overfitting on local neighborhood structures for graphs with very wide node degree distributions. It further approximates $\lambda_{max} \approx 2$ and the equation simplifies to:

$$\mathbf{g}_{\theta'} \star \mathbf{x} \approx \theta'_0\mathbf{x} + \theta'_1(\mathbf{L} - \mathbf{I}_N)\mathbf{x} = \theta'_0\mathbf{x} - \theta'_1\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{x} \tag{12}$$

with two free parameters $\theta'_0$ and $\theta'_1$. After constraining the number of parameters with $\theta = \theta'_0 = -\theta'_1$, we can obtain the following expression:

$$\mathbf{g}_\theta \star \mathbf{x} \approx \theta\left(\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right)\mathbf{x} \tag{13}$$

Note that stacking this operator could lead to numerical instabilities and exploding/vanishing gradients, [2] introduces the *renormalization trick*: $\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} \rightarrow \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$, with $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ and $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$. Finally, [2] generalizes the definition to a signal $\mathbf{X} \in \mathbb{R}^{N \times C}$ with $C$ input channels and $F$ filters for feature maps as follows:

$$\mathbf{Z} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}\mathbf{\Theta} \tag{14}$$

where $\mathbf{\Theta} \in \mathbb{R}^{C \times F}$ is a matrix of filter parameters and $\mathbf{Z} \in \mathbb{R}^{N \times F}$ is the convolved signal matrix.

All of these models use the original graph structure to denote relations between nodes. However, there may have implicit relations between different nodes and the Adaptive Graph Convolution Network (AGCN) is proposed to learn the underlying relations [49]. AGCN learns a "residual" graph Laplacian and add it to the original Laplacian matrix. As a result, it is proven to be effective in several graph-structured datasets.

What's more, [50] presents a Gaussian process-based Bayesian approach (GGP) to solve the semi-supervised learning problems. It shows parallels between the model and the spectral filtering approaches, which could give us some insights from another perspective.

However, in all of the spectral approaches mentioned above, the learned filters depend on the Laplacian eigenbasis, which depends on the graph structure, that is, a model trained on a specific structure could not be directly applied to a graph with a different structure.

Non-spectral approaches define convolutions directly on the graph, operating on spatially close neighbors. The major challenge of non-spectral approaches is defining the convolution operation with differently sized neighborhoods and maintaining the local invariance of CNNs.

*Neural FPs.* [51] uses different weight matrices for nodes with different degrees,

$$\begin{aligned} \mathbf{x} &= \mathbf{h}_v^{t-1} + \sum_{i=1}^{|\mathcal{N}_v|} \mathbf{h}_i^{t-1} \\ \mathbf{h}_v^t &= \sigma\left(\mathbf{x}\mathbf{W}_t^{|\mathcal{N}_v|}\right) \end{aligned} \tag{15}$$

where $\mathbf{W}_\mathbf{t}^{|\mathcal{N}_\mathbf{v}|}$ is the weight matrix for nodes with degree $|\mathcal{N}_v|$ at layer $t$. And the main drawback of the method is that it cannot be applied to large-scale graphs with more node degrees.

*DCNN.* [23] proposed the diffusion-convolutional neural networks (DCNNs). Transition matrices are used to define the neighborhood for nodes in DCNN. For node classification, it has

$$\mathbf{H} = f\left(\mathbf{W}^c \odot \mathbf{P}^*\mathbf{X}\right) \tag{16}$$

where $\mathbf{X}$ is an $N \times F$ tensor of input features ($N$ is the number of nodes and $F$ is the number of features). $\mathbf{P}^*$ is an $N \times K \times N$ tensor which contains the power series $\{\mathbf{P}, \mathbf{P}^2, ..., \mathbf{P}^K\}$ of matrix $\mathbf{P}$. And $\mathbf{P}$ is the degree-normalized transition matrix from the graphs adjacency matrix $\mathbf{A}$. Each entity is transformed to a diffusion convolutional representation which is a $K \times F$ matrix defined by $K$ hops of graph diffusion over $F$ features. And then it will be defined by a $K \times F$ weight matrix and a non-linear activation function $f$. Finally $\mathbf{H}$ (which is $N \times K \times F$) denotes the diffusion representations of each node in the graph.

As for graph classification, DCNN simply takes the average of nodes' representation,

$$\mathbf{H} = f\left(\mathbf{W}^c \odot 1_N^T\mathbf{P}^*\mathbf{X}/N\right) \tag{17}$$

and $1_N$ here is an $N \times 1$ vector of ones. DCNN can also be applied to edge classification tasks, which requires converting edges to nodes and augmenting the adjacency matrix.

*DGCN.* [52] proposed the dual graph convolutional network (DGCN) to jointly consider the local consistency and global consistency on graphs. It uses two convolutional networks to capture the local/global consistency and adopts an unsupervised loss to ensemble them. The first convolutional network is the same as Eq. 14. And the second network re-

places the adjacency matrix with positive pointwise mutual information (PPMI) matrix:

$$\mathbf{H}' = \rho(\mathbf{D}_P^{-\frac{1}{2}}\mathbf{X}_P\mathbf{D}_P^{-\frac{1}{2}}\mathbf{H}\mathbf{\Theta}) \tag{18}$$

where $\mathbf{X}_P$ is the PPMI matrix and $\mathbf{D}_P$ is the diagonal degree matrix of $\mathbf{X}_P$.

*PATCHY-SAN.* The PATCHY-SAN model [53] extracts and normalizes a neighborhood of exactly $k$ nodes for each node. And then the normalized neighborhood serves as the receptive field for the convolutional operation.

*LGCN.* LGCN [54] also leverages CNNs as aggregators. It performs max pooling on nodes' neighborhood matrices to get top-k feature elements and then applies 1-D CNN to compute hidden representations.

*MoNet.* [22] proposed a spatial-domain model (MoNet) on non-Euclidean domains which could generalize several previous techniques. The Geodesic CNN (GCNN) [24] and Anisotropic CNN (ACNN) [25] on manifolds or GCN [2] and DCNN [23] on graphs could be formulated as particular instances of MoNet.

*GraphSAGE.* [1] proposed the GraphSAGE, a general inductive framework. The framework generates embeddings by sampling and aggregating features from a node's local neighborhood.

$$\begin{aligned}\mathbf{h}_{\mathcal{N}_v}^t &= \mathrm{AGGREGATE}_t\left(\{\mathbf{h}_u^{t-1}, \forall u \in \mathcal{N}_v\}\right)\\ \mathbf{h}_v^t &= \sigma\left(\mathbf{W}^t \cdot [\mathbf{h}_v^{t-1} \| \mathbf{h}_{\mathcal{N}_v}^t]\right)\end{aligned} \tag{19}$$

However, [1] does not utilize the full set of neighbors in Eq.19 but a fixed-size set of neighbors by uniformly sampling. And [1] suggests three aggregator functions.

- Mean aggregator. It could be viewed as an approximation of the convolutional operation from the transductive GCN framework [2], so that the inductive version of the GCN variant could be derived by

  $$\mathbf{h}_v^t = \sigma(\mathbf{W} \cdot \mathrm{MEAN}(\{\mathbf{h}_v^{t-1}\} \cup \{\mathbf{h}_u^{t-1}, \forall u \in \mathcal{N}_v\}) \tag{20}$$

  The mean aggregator is different from other aggregators because it does not perform the concatenation operation which concatenates $\mathbf{h}_v^{t-1}$ and $\mathbf{h}_{\mathcal{N}_v}^t$ in Eq.19. It could be viewed as a form of "skip connection" [55] and could achieve better performance.
- LSTM aggregator. [1] also uses an LSTM-based aggregator which has a larger expressive capability. However, LSTMs process inputs in a sequential manner so that they are not permutation invariant. [1] adapts LSTMs to operate on an unordered set by permutating node's neighbors.
- Pooling aggregator. In the pooling aggregator, each neighbor's hidden state is fed through a fully-connected layer and then a max-pooling operation is applied to the set of the node's neighbors.

  $$\mathbf{h}_{\mathcal{N}_v}^t = \max(\{\sigma\left(\mathbf{W}_{\text{pool}}\mathbf{h}_u^{t-1} + \mathbf{b}\right), \forall u \in \mathcal{N}_v\}) \tag{21}$$

  Note that any symmetric functions could be used in place of the max-pooling operation here.

Recently, the structure-aware convolution and Structure-Aware Convolutional Neural Networks (SACNNs) have been proposed [56]. Univariate functions are used to perform as filters and they can deal with both Euclidean and non-Euclidean structured data.

**Gate.** There are several works attempting to use the gate mechanism like GRU [57] or LSTM [58] in the propagation step to diminish the restrictions in the former GNN models and improve the long-term propagation of information across the graph structure.

[59] proposed the gated graph neural network (GGNN) which uses the Gate Recurrent Units (GRU) in the propagation step, unrolls the recurrence for a fixed number of steps $T$ and uses backpropagation through time in order to compute gradients.

Specifically, the basic recurrence of the propagation model is

$$\begin{aligned}\mathbf{a}_v^t &= \mathbf{A}_v^T[\mathbf{h}_1^{t-1} \dots \mathbf{h}_N^{t-1}]^T + \mathbf{b}\\ \mathbf{z}_v^t &= \sigma\left(\mathbf{W}^z\mathbf{a}_v^t + \mathbf{U}^z\mathbf{h}_v^{t-1}\right)\\ \mathbf{r}_v^t &= \sigma\left(\mathbf{W}^r\mathbf{a}_v^t + \mathbf{U}^r\mathbf{h}_v^{t-1}\right)\\ \widetilde{\mathbf{h}_v^t} &= \tanh\left(\mathbf{W}\mathbf{a}_v^t + \mathbf{U}\left(\mathbf{r}_v^t \odot \mathbf{h}_v^{t-1}\right)\right)\\ \mathbf{h}_v^t &= \left(1 - \mathbf{z}_v^t\right) \odot \mathbf{h}_v^{t-1} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}_v^t}\end{aligned} \tag{22}$$

The node $v$ first aggregates message from its neighbors, where $\mathbf{A}_v$ is the sub-matrix of the graph adjacency matrix $\mathbf{A}$ and denotes the connection of node $v$ with its neighbors. The GRU-like update functions incorporate information from the other nodes and from the previous timestep to update each node's hidden state. $\mathbf{a}$ gathers the neighborhood information of node $v$, $\mathbf{z}$ and $\mathbf{r}$ are the update and reset gates.

LSTMs are also used in a similar way as GRU through the propagation process based on a tree or a graph.

[60] proposed two extensions to the basic LSTM architecture: the *Child-Sum Tree-LSTM* and the *N-ary Tree-LSTM*. Like in standard LSTM units, each Tree-LSTM unit (indexed by $v$) contains input and output gates $\mathbf{i}_v$ and $\mathbf{o}_v$, a memory cell $\mathbf{c}_v$ and hidden state $\mathbf{h}_v$. Instead of a single forget gate, the Tree-LSTM unit contains one forget gate $\mathbf{f}_{vk}$ for each child $k$, allowing the unit to selectively incorporate information from each child. The Child-Sum Tree-LSTM transition equations are the following:

$$\begin{aligned}\widetilde{\mathbf{h}_v^{t-1}} &= \sum_{k \in \mathcal{N}_v} \mathbf{h}_k^{t-1}\\ \mathbf{i}_v^t &= \sigma\Big(\mathbf{W}^i\mathbf{x}_v^t + \mathbf{U}^i\widetilde{\mathbf{h}_v^{t-1}} + \mathbf{b}^i\Big)\\ \mathbf{f}_{vk}^t &= \sigma\Big(\mathbf{W}^f\mathbf{x}_v^t + \mathbf{U}^f\mathbf{h}_k^{t-1} + \mathbf{b}^f\Big)\\ \mathbf{o}_v^t &= \sigma\Big(\mathbf{W}^o\mathbf{x}_v^t + \mathbf{U}^o\widetilde{\mathbf{h}_v^{t-1}} + \mathbf{b}^o\Big)\\ \mathbf{u}_v^t &= \tanh\Big(\mathbf{W}^u\mathbf{x}_v^t + \mathbf{U}^u\widetilde{\mathbf{h}_v^{t-1}} + \mathbf{b}^u\Big)\\ \mathbf{c}_v^t &= \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{k \in \mathcal{N}_v} \mathbf{f}_{vk}^t \odot \mathbf{c}_k^{t-1}\\ \mathbf{h}_v^t &= \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)\end{aligned} \tag{23}$$

$\mathbf{x}_v^t$ is the input vector at time $t$ in the standard LSTM setting.

If the branching factor of a tree is at most $K$ and all children of a node are ordered, $i.e.$, they can be indexed

from 1 to $K$, then the $N$-ary Tree-LSTM can be used. For node $v$, $\mathbf{h}_{vk}^t$ and $\mathbf{c}_{vk}^t$ denote the hidden state and memory cell of its $k$-th child at time $t$ respectively. The transition equations are the following:

$$\begin{aligned}
\mathbf{i}_v^t &= \sigma\Big(\mathbf{W}^i\mathbf{x}_v^t + \sum_{l=1}^{K}\mathbf{U}_l^i\mathbf{h}_{vl}^{t-1} + \mathbf{b}^i\Big)\\
\mathbf{f}_{vk}^t &= \sigma\Big(\mathbf{W}^f\mathbf{x}_v^t + \sum_{l=1}^{K}\mathbf{U}_{kl}^f\mathbf{h}_{vl}^{t-1} + \mathbf{b}^f\Big)\\
\mathbf{o}_v^t &= \sigma\Big(\mathbf{W}^o\mathbf{x}_v^t + \sum_{l=1}^{K}\mathbf{U}_l^o\mathbf{h}_{vl}^{t-1} + \mathbf{b}^o\Big)\\
\mathbf{u}_v^t &= \tanh\Big(\mathbf{W}^u\mathbf{x}_v^t + \sum_{l=1}^{K}\mathbf{U}_l^u\mathbf{h}_{vl}^{t-1} + \mathbf{b}^u\Big)\\
\mathbf{c}_v^t &= \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{l=1}^{K}\mathbf{f}_{vl}^t \odot \mathbf{c}_{vl}^{t-1}\\
\mathbf{h}_v^t &= \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)
\end{aligned} \tag{24}$$

The introduction of separate parameter matrices for each child $k$ allows the model to learn more fine-grained representations conditioning on the states of a unit's children than the Child-Sum Tree-LSTM.

The two types of Tree-LSTMs can be easily adapted to the graph. The graph-structured LSTM in [61] is an example of the $N$-ary Tree-LSTM applied to the graph. However, it is a simplified version since each node in the graph has at most 2 incoming edges (from its parent and sibling predecessor). [44] proposed another variant of the Graph LSTM based on the relation extraction task. The main difference between graphs and trees is that edges of graphs have their labels. And [44] utilizes different weight matrices to represent different labels.

$$\begin{aligned}
\mathbf{i}_v^t &= \sigma\Big(\mathbf{W}^i\mathbf{x}_v^t + \sum_{k\in\mathcal{N}_v}\mathbf{U}_{m(v,k)}^i\mathbf{h}_k^{t-1} + \mathbf{b}^i\Big)\\
\mathbf{f}_{vk}^t &= \sigma\Big(\mathbf{W}^f\mathbf{x}_v^t + \mathbf{U}_{m(v,k)}^f\mathbf{h}_k^{t-1} + \mathbf{b}^f\Big)\\
\mathbf{o}_v^t &= \sigma\Big(\mathbf{W}^o\mathbf{x}_v^t + \sum_{k\in\mathcal{N}_v}\mathbf{U}_{m(v,k)}^o\mathbf{h}_k^{t-1} + \mathbf{b}^o\Big)\\
\mathbf{u}_v^t &= \tanh\Big(\mathbf{W}^u\mathbf{x}_v^t + \sum_{k\in\mathcal{N}_v}\mathbf{U}_{m(v,k)}^u\mathbf{h}_k^{t-1} + \mathbf{b}^u\Big)\\
\mathbf{c}_v^t &= \mathbf{i}_v^t \odot \mathbf{u}_v^t + \sum_{k\in\mathcal{N}_v}\mathbf{f}_{vk}^t \odot \mathbf{c}_k^{t-1}\\
\mathbf{h}_v^t &= \mathbf{o}_v^t \odot \tanh(\mathbf{c}_v^t)
\end{aligned} \tag{25}$$

where $m(v,k)$ denotes the edge label between node $v$ and $k$.

[62] proposed the Sentence LSTM (S-LSTM) for improving text encoding. It converts text into a graph and utilizes the Graph LSTM to learn the representation. The S-LSTM shows strong representation power in many NLP problems. [63] proposed a Graph LSTM network to address the semantic object parsing task. It uses the confidence-driven scheme to adaptively select the starting node and determine the node updating sequence. It follows the same idea of generalizing the existing LSTMs into the graph-structured data but has a specific updating sequence while methods we mentioned above are agnostic to the order of nodes.

**Attention.** The attention mechanism has been successfully used in many sequence-based tasks such as machine translation [64]–[66], machine reading [67] and so on. [68] proposed a graph attention network (GAT) which incorporates the attention mechanism into the propagation step. It computes the hidden states of each node by attending over its neighbors, following a *self-attention* strategy.

[68] defines a single *graph attentional layer* and constructs arbitrary graph attention networks by stacking this layer. The layer computes the coefficients in the attention mechanism of the node pair $(i,j)$ by:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \| \mathbf{W}\mathbf{h}_k]\right)\right)} \tag{26}$$

where $\alpha_{ij}$ is the attention coefficient of node $j$ to $i$, $\mathcal{N}_i$ represents the neighborhoods of node $i$ in the graph. The input set of node features to the layer is $\mathbf{h} = \{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_N\}, \mathbf{h}_i \in \mathbb{R}^F$, where $N$ is the number of nodes and $F$ is the number of features of each node, the layer produces a new set of node features(of potentially different cardinality $F'$), $\mathbf{h}' = \{\mathbf{h}_1', \mathbf{h}_2', \ldots, \mathbf{h}_N'\}, \mathbf{h}_i' \in \mathbb{R}^{F'}$, as its output. $\mathbf{W} \in \mathbb{R}^{F'\times F}$ is the *weight matrix* of a shared linear transformation which applied to every node, $\mathbf{a} \in \mathbb{R}^{2F'}$ is the weight vector of a single-layer feedforward neural network. It is normalized by a softmax function and the LeakyReLU nonlinearity(with negative input slop $\alpha = 0.2$) is applied.

Then the final output features of each node can be obtained by (after applying a nonlinearity $\sigma$):

$$\mathbf{h}_i' = \sigma\Big(\sum_{j\in\mathcal{N}_i}\alpha_{ij}\mathbf{W}\mathbf{h}_j\Big) \tag{27}$$

Moreover, the layer utilizes the *multi-head attention* similarly to [66] to stabilize the learning process. It applies $K$ independent attention mechanisms to compute the hidden states and then concatenates their features(or computes the average), resulting in the following two output representations:

$$\mathbf{h}_i' = \Big\|_{k=1}^{K}\sigma\Big(\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\mathbf{h}_j\Big) \tag{28}$$

$$\mathbf{h}_i' = \sigma\Big(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\mathbf{W}^k\mathbf{h}_j\Big) \tag{29}$$

where $\alpha_{ij}^k$ is normalized attention coefficient computed by the $k$-th attention mechanism.

The attention architecture in [68] has several properties: (1) the computation of the node-neighbor pairs is parallelizable thus the operation is efficient; (2) it can be applied to graph nodes with different degrees by specifying arbitrary weights to neighbors; (3) it can be applied to the inductive learning problems easily.

Besides GAT, Gated Attention Network (GAAN) [69] also uses the multi-head attention mechanism. However, it uses a self-attention mechanism to gather information from

different heads to replace the average operation of GAT.

**Skip connection.** Many applications unroll or stack the graph neural network layer aiming to achieve better results as more layers (i.e $k$ layers) make each node aggregate more information from neighbors $k$ hops away. However, it has been observed in many experiments that deeper models could not improve the performance and deeper models could even perform worse [2]. This is mainly because more layers could also propagate the noisy information from an exponentially increasing number of expanded neighborhood members.

A straightforward method to address the problem, the residual network [70], could be found from the computer vision community. But, even with residual connections, GCNs with more layers do not perform as well as the 2-layer GCN on many datasets [2].

[71] proposed a Highway GCN which uses layer-wise gates similar to highway networks [72]. The output of a layer is summed with its input with gating weights:

$$\begin{aligned} \mathbf{T}(\mathbf{h}^t) &= \sigma\left(\mathbf{W}^t\mathbf{h}^t + \mathbf{b}^t\right) \\ \mathbf{h}^{t+1} &= \mathbf{h}^{t+1} \odot \mathbf{T}(\mathbf{h}^t) + \mathbf{h}^t \odot (1 - \mathbf{T}(\mathbf{h}^t)) \end{aligned} \tag{30}$$

By adding the highway gates, the performance peaks at 4 layers in a specific problem discussed in [71]. The Column Network (CLN) proposed in [73] also utilizes the highway network. But it has different function to compute the gating weights.

[74] studies properties and resulting limitations of neighborhood aggregation schemes. It proposed the Jump Knowledge Network which could learn adaptive, *structure-aware* representations. The Jump Knowledge Network selects from all of the intermediate representations (which "jump" to the last layer) for each node at the last layer, which makes the model adapt the effective neighborhood size for each node as needed. [74] uses three approaches of **concatenation**, **max-pooling** and **LSTM-attention** in the experiments to aggregate information. The Jump Knowledge Network performs well on the experiments in social, bioinformatics and citation networks. It could also be combined with models like Graph Convolutional Networks, GraphSAGE and Graph Attention Networks to improve their performance.

**Hierarchical Pooling.** In the area of computer vision, a convolutional layer is usually followed by a pooling layer to get more general features. Similar to these pooling layers, a lot of work focuses on designing hierarchical pooling layers on graphs. Complicated and large-scale graphs usually carry rich hierarchical structures which are of great importance for node-level and graph-level classification tasks.

To explore such inner features, Edge-Conditioned Convolution (ECC) [75] designs its pooling module with recursively downsampling operation. The downsampling method is based on splitting the graph into two components by the sign of the largest eigenvector of the Laplacian.

DIFFPOOL [76] proposed a learnable hierarchical clustering module by training an assignment matrix in each layer:

$$\mathbf{S}^{(l)} = softmax(\mathbf{GNN}_{\mathbf{l,pool}}(A^{(l)}, X^{(l)})) \tag{31}$$

where $X^{(l)}$ is node features and $A^{(l)}$ is coarsened adjacency matrix of layer $l$.

### 2.2.3 Training Methods

The original graph convolutional neural network has several drawbacks in training and optimization methods. Specifically, GCN requires the full graph Laplacian, which is computational-consuming for large graphs. Furthermore, The embedding of a node at layer $L$ is computed recursively by the embeddings of all its neighbors at layer $L-1$. Therefore, the receptive field of a single node grows exponentially with respect to the number of layers, so computing gradient for a single node costs a lot. Finally, GCN is trained independently for a fixed graph, which lacks the ability for inductive learning.

**Sampling.** GraphSAGE [1] is a comprehensive improvement of original GCN. To solve the problems mentioned above, GraphSAGE replaced full graph Laplacian with learnable aggregation functions, which are key to perform message passing and generalize to unseen nodes. As shown in Eq.19, they first aggregate neighborhood embeddings, concatenate with target node's embedding, then propagate to the next layer. With learned aggregation and propagation functions, GraphSAGE could generate embeddings for unseen nodes. Also, GraphSAGE uses neighbor sampling to alleviate receptive field expansion.

PinSage [77] proposed importance-based sampling method. By simulating random walks starting from target nodes, this approach chooses the top T nodes with the highest normalized visit counts.

FastGCN [78] further improves the sampling algorithm. Instead of sampling neighbors for each node, FastGCN directly samples the receptive field for each layer. FastGCN uses importance sampling, which the importance factor is calculated as below:

$$q(v) \propto \frac{1}{|\mathcal{N}_v|} \sum_{u \in \mathcal{N}_v} \frac{1}{|\mathcal{N}_u|} \tag{32}$$

In contrast to fixed sampling methods above, [79] introduces a parameterized and trainable sampler to perform layer-wise sampling conditioned on the former layer. Furthermore, this adaptive sampler could find optimal sampling importance and reduce variance simultaneously.

Following reinforcement learning, SSE [80] proposed Stochastic Fixed-Point Gradient Descent for GNN training. This method views embedding update as value function and parameter update as value function. While training, the algorithm will sample nodes to update embeddings and sample labeled nodes to update parameters alternately.

**Receptive Field Control.** [81] proposed a control-variate based stochastic approximation algorithms for GCN by utilizing the historical activations of nodes as a control variate. This method limits the receptive field in the 1-hop neighborhood, but use the historical hidden state as

an affordable approximation.

**Data Augmentation.** [82] focused on the limitations of GCN, which include that GCN requires many additional labeled data for validation and also suffers from the localized nature of the convolutional filter. To solve the limitations, the authors proposed Co-Training GCN and Self-Training GCN to enlarge the training dataset. The former method finds the nearest neighbors of training data while the latter one follows a boosting-like way.

**Unsupervised Training.** GNNs are typically used for supervised or semi-supervised learning problems. Recently, there has been a trend to extend auto-encoder (AE) to graph domains. Graph auto-encoders aim at representing nodes into low-dimensional vectors by an unsupervised training manner.

Graph Auto-Encoder (GAE) [83] first uses GCNs to encode nodes in the graph. Then it uses a simple decoder to reconstruct the adjacency matrix and computes the loss from the similarity between the original adjacency matrix and the reconstructed matrix.

$$\begin{aligned} \mathbf{Z} &= \mathrm{GCN}(\mathbf{X}, \mathbf{A}) \\ \widetilde{\mathbf{A}} &= \rho(\mathbf{Z}\mathbf{Z}^T) \end{aligned} \tag{33}$$

[83] also trains the GAE model in a variational manner and the model is named as the variational graph auto-encoder (VGAE). Furthermore, Berg et al. use GAE in recommender systems and have proposed the graph convolutional matrix completion model (GC-MC) [84], which outperforms other baseline models on the MovieLens dataset.

Adversarially Regularized Graph Auto-encoder (ARGA) [85] employs generative adversarial networks (GANs) to regularize a GCN-based graph auto-encoder to follow a prior distribution. There are also several graph auto-encoders such as NetRA [86], DNGR [87], SDNE [88] and DRNE [89], however, they don't use GNNs in their framework.

## 2.3 General Frameworks

Apart from different variants of graph neural networks, several general frameworks are proposed aiming to integrate different models into one single framework. [27] proposed the message passing neural network (MPNN), which unified various graph neural network and graph convolutional network approaches. [28] proposed the non-local neural network (NLNN). It unifies several "self-attention"-style methods [66], [68], [90]. [30] proposed the graph network (GN) which unified the MPNN and NLNN methods as well as many other variants like Interaction Networks [4], [91], Neural Physics Engine [92], CommNet [93], structure2vec [7], [94], GGNN [59], Relation Network [95], [96], Deep Sets [97] and Point Net [98].

### 2.3.1 Message Passing Neural Networks

[27] proposed a general framework for supervised learning on graphs called Message Passing Neural Networks (MPNNs). The MPNN framework abstracts the commonalities between several of the most popular models for graph-structured data, such as spectral approaches [45] [2], [48] and non-spectral approaches [51] in graph convolution, gated graph neural networks [59], interaction networks [4], molecular graph convolutions [99], deep tensor neural networks [100] and so on.

The model contains two phases, a message passing phase and a readout phase. The message passing phase (namely, the propagation step) runs for $T$ time steps and is defined in terms of message function $M_t$ and vertex update function $U_t$. Using messages $\mathbf{m}_v^t$, the updating functions of hidden states $\mathbf{h}_v^t$ are as follows:

$$\begin{aligned} \mathbf{m}_v^{t+1} &= \sum_{w \in \mathcal{N}_v} M_t\left(\mathbf{h}_v^t, \mathbf{h}_w^t, \mathbf{e}_{vw}\right) \\ \mathbf{h}_v^{t+1} &= U_t\left(\mathbf{h}_v^t, \mathbf{m}_v^{t+1}\right) \end{aligned} \tag{34}$$

where $\mathbf{e}_{vw}$ represents features of the edge from node $v$ to $w$. The readout phase computes a feature vector for the whole graph using the readout function $R$ according to

$$\hat{\mathbf{y}} = R(\{\mathbf{h}_v^T | v \in G\}) \tag{35}$$

where T denotes the total time steps. The message function $M_t$, vertex update function $U_t$ and readout function $R$ could have different settings. Hence the MPNN framework could generalize several different models via different function settings. Here we give an example of generalizing GGNN, and other models' function settings could be found in [27]. The function settings for GGNNs are:

$$\begin{aligned} M_t\left(\mathbf{h}_v^t, \mathbf{h}_w^t, \mathbf{e}_{vw}\right) &= \mathbf{A}_{\mathbf{e}_{vw}} \mathbf{h}_w^t \\ U_t &= GRU\left(\mathbf{h}_v^t, \mathbf{m}_v^{t+1}\right) \\ R &= \sum_{v \in V} \sigma\left(i(\mathbf{h}_v^T, \mathbf{h}_v^0)\right) \odot \left(j(\mathbf{h}_v^T)\right) \end{aligned} \tag{36}$$

where $\mathbf{A}_{\mathbf{e}_{vw}}$ is the adjacency matrix, one for each edge label $e$. The GRU is the Gated Recurrent Unit introduced in [57]. $i$ and $j$ are neural networks in function $R$.

### 2.3.2 Non-local Neural Networks

[28] proposed the Non-local Neural Networks (NLNN) for capturing long-range dependencies with deep neural networks. The non-local operation is a generalization of the classical non-local mean operation [101] in computer vision. A non-local operation computes the response at a position as a weighted sum of the features at all positions. The set of positions can be in space, time or spacetime. Thus the NLNN can be viewed as a unification of different "self-attention"-style methods [66], [68], [90]. We will first introduce the general definition of non-local operations and then some specific instantiations.

Following the non-local mean operation [101], the generic non-local operation is defined as:

$$\mathbf{h}_i' = \frac{1}{\mathcal{C}(\mathbf{h})} \sum_{\forall j} f(\mathbf{h}_i, \mathbf{h}_j) g(\mathbf{h}_j) \tag{37}$$

where $i$ is the index of an output position and $j$ is the index that enumerates all possible positions. $f(\mathbf{h}_i, \mathbf{h}_j)$ computes a scalar between $i$ and $j$ representing the relation between them. $g(\mathbf{h}_j)$ denotes a transformation of the input $\mathbf{h}_j$ and a

factor $\frac{1}{\mathcal{C}(\mathbf{h})}$ is utilized to normalize the results.

There are several instantiations with different $f$ and $g$ settings. For simplicity, [28] uses the linear transformation as the function $g$. That means $g(\mathbf{h}_j) = \mathbf{W}_g\mathbf{h}_j$, where $W_g$ is a learned weight matrix. Next we list the choices for function $f$ in the following.

**Gaussian.** The Gaussian function is a natural choice according to the non-local mean [101] and bilateral filters [102]. Thus:

$$f(\mathbf{h}_i, \mathbf{h}_j) = e^{\mathbf{h}_i^T \mathbf{h}_j} \tag{38}$$

Here $\mathbf{h}_i^T\mathbf{h}_j$ is dot-product similarity and $\mathcal{C}(\mathbf{h}) = \sum_{\forall j} f(\mathbf{h}_i, \mathbf{h}_j)$.

**Embedded Gaussian.** It is straightforward to extend the Gaussian function by computing similarity in the embedding space, which means:

$$f(\mathbf{h}_i, \mathbf{h}_j) = e^{\theta(\mathbf{h}_i)^T \phi(\mathbf{h}_j)} \tag{39}$$

where $\theta(\mathbf{h}_i) = \mathbf{W}_\theta\mathbf{h}_i$, $\phi(\mathbf{h}_j) = W_\phi\mathbf{h}_j$ and $\mathcal{C}(\mathbf{h}) = \sum_{\forall j} f(\mathbf{h}_i, \mathbf{h}_j)$.

It could be found that the self-attention proposed in [66] is a special case of the Embedded Gaussian version. For a given $i$, $\frac{1}{\mathcal{C}(\mathbf{h})} f(\mathbf{h}_i, \mathbf{h}_j)$ becomes the *softmax* computation along the dimension $j$. So that $\mathbf{h}' = \textit{softmax}(\mathbf{h}^T\mathbf{W}_\theta^T\mathbf{W}_\phi\mathbf{h})g(\mathbf{h})$, which matches the form of self-attention in [66].

**Dot product.** The function $f$ can also be implemented as a dot-product similarity:

$$f(\mathbf{h}_i, \mathbf{h}_j) = \theta(\mathbf{h}_i)^T\phi(\mathbf{h}_j). \tag{40}$$

Here the factor $\mathcal{C}(\mathbf{h}) = N$, where $N$ is the number of positions in $\mathbf{h}$.

**Concatenation.** Here we have:

$$f(\mathbf{h}_i, \mathbf{h}_j) = \text{ReLU}(\mathbf{w}_f^T[\theta(\mathbf{h}_i) \| \phi(\mathbf{h}_j)]). \tag{41}$$

where $\mathbf{w}_f$ is a weight vector projecting the vector to a scalar and $\mathcal{C}(\mathbf{h}) = N$.

[28] wraps the non-local operation mentioned above into a non-local block as:

$$\mathbf{z}_i = \mathbf{W}_z\mathbf{h}'_i + \mathbf{h}_i, \tag{42}$$

where $\mathbf{h}'_i$ is given in Eq.37 and "$+\mathbf{h}_i$" denotes the residual connection [70]. Hence the non-local block could be insert into any pre-trained model, which makes the block more applicable.

### 2.3.3 Graph Networks

[30] proposed the Graph Network (GN) framework which generalizes and extends various graph neural network, MPNN and NLNN approaches [27], [28], [33]. We first introduce the graph definition in [30] and then we describe the GN block, a core GN computation unit, and its computational steps, and finally we will introduce its basic design principles.

**Graph definition.** In [30], a graph is defined as a 3-tuple $G = (\mathbf{u}, H, E)$ (here we use $H$ instead of $V$ for notation's consistency). $\mathbf{u}$ is a global attribute, $H = \{\mathbf{h}_i\}_{i=1:N^v}$ is the set of nodes (of cardinality $N^v$), where each $\mathbf{h}_i$ is a node's attribute. $E = \{(\mathbf{e}_k, r_k, s_k)\}_{k=1:N^e}$ is the set of edges (of cardinality $N^e$), where each $\mathbf{e}_k$ is the edge's attribute, $r_k$ is the index of the receiver node and $s_k$ is the index of the sender node.

**GN block**. A GN block contains three "update" functions, $\phi$, and three "aggregation" functions, $\rho$,

$$\begin{aligned} \mathbf{e}'_k &= \phi^e(\mathbf{e}_k, \mathbf{h}_{r_k}, \mathbf{h}_{s_k}, \mathbf{u}) & \bar{\mathbf{e}}'_i &= \rho^{e\to h}(E'_i) \\ \mathbf{h}'_i &= \phi^h(\bar{\mathbf{e}}'_i, \mathbf{h}_i, \mathbf{u}) & \bar{\mathbf{e}}' &= \rho^{e\to u}(E') \\ \mathbf{u}' &= \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{h}}', \mathbf{u}) & \bar{\mathbf{h}}' &= \rho^{h\to u}(H') \end{aligned} \tag{43}$$

where $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i,\ k=1:N^e}$, $H' = \{\mathbf{h}'_i\}_{i=1:N^v}$, and $E' = \bigcup_i E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$. The $\rho$ functions must be invariant to permutations of their inputs and should take variable numbers of arguments.

**Computation steps.** The computation steps of a GN block are as follows:

1) $\phi^e$ is applied per edge, with arguments $(\mathbf{e}_k, \mathbf{h}_{r_k}, \mathbf{h}_{s_k}, \mathbf{u})$, and returns $\mathbf{e}'_k$. The set of resulting per-edge outputs for each node $i$ is, $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i,\ k=1:N^e}$. And $E' = \bigcup_i E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$ is the set of all per-edge outputs.
2) $\rho^{e\to h}$ is applied to $E'_i$, and aggregates the edge updates for edges that project to vertex $i$, into $\bar{\mathbf{e}}'_i$, which will be used in the next step's node update.
3) $\phi^h$ is applied to each node $i$, to compute an updated node attribute, $\mathbf{h}'_i$. The set of resulting per-node outputs is, $H' = \{\mathbf{h}'_i\}_{i=1:N^v}$.
4) $\rho^{e\to u}$ is applied to $E'$, and aggregates all edge updates, into $\bar{\mathbf{e}}'$, which will then be used in the next step's global update.
5) $\rho^{h\to u}$ is applied to $H'$, and aggregates all node updates, into $\bar{\mathbf{h}}'$, which will then be used in the next step's global update.
6) $\phi^u$ is applied once per graph and computes an update for the global attribute, $\mathbf{u}'$.

Note here the order is not strictly enforced. For example, it is possible to proceed from global, to per-node, to per-edge updates. And the $\phi$ and $\rho$ functions need not be neural networks though in this paper we only focus on neural network implementations.

**Design Principles.** The design of Graph Network based on three basic principles: flexible representations, configurable within-block structure and composable multi-block architectures.

- **Flexible representations.** The GN framework supports flexible representations of the attributes as well as different graph structures. The global, node and edge attributes can use arbitrary representational formats but real-valued vectors and tensors are most common. One can simply tailor the output of a GN block according to specific demands of tasks. For example, [30] lists several *edge-focused* [103], [104], *node-focused* [3], [4], [92], [105] and *graph-focused* [4],

[27], [96] GNs. In terms of graph structures, the framework can be applied to both structural scenarios where the graph structure is explicit and non-structural scenarios where the relational structure should be inferred or assumed.

- **Configurable within-block structure.** The functions and their inputs within a GN block can have different settings so that the GN framework provides flexibility in within-block structure configuration. For example, [104] and [3] use the full GN blocks. Their $\phi$ implementations use neural networks and their $\rho$ functions use the elementwise summation. Based on different structure and functions settings, a variety of models (such as MPNN, NLNN and other variants) could be expressed by the GN framework. And more details could be found in [30].
- **Composable multi-block architectures.** GN blocks could be composed to construct complex architectures. Arbitrary numbers of GN blocks could be composed in sequence with shared or unshared parameters. [30] utilizes GN blocks to construct an *encode-process-decode* architecture and a recurrent GN-based architecture. These architectures are demonstrated in Fig. 3. Other techniques for building GN based architectures could also be useful, such as skip connections, LSTM- or GRU-style gating schemes and so on.

## 3 Applications

Graph neural networks have been explored in a wide range of problem domains across supervised, semi-supervised, unsupervised and reinforcement learning settings. In this section, we simply divide the applications in three scenarios: (1) Structural scenarios where the data has explicit relational structure, such as physical systems, molecular structures and knowledge graphs; (2) Non-structural scenarios where the relational structure is not explicit include image, text, etc; (3) Other application scenarios such as generative models and combinatorial optimization problems. Note that we only list several representative applications instead of providing an exhaustive list. The summary of the applications could be found in Table 3.

### 3.1 Structural Scenarios

In the following subsections, we will introduce GNN's applications in structural scenarios, where the data are naturally performed in the graph structure. For example, GNNs are widely being used in social network prediction [1], [2], traffic prediction [71], [144], recommender systems [77], [84] and graph representation [76]. Specifically, we are discussing how to model real-world physical systems with object-relationship graphs, how to predict chemical properties of molecules and biological interaction properties of proteins and the methods of reasoning about the out-of-knowledge-base(OOKB) entities in knowledge graphs.

#### 3.1.1 Physics

Modeling real-world physical systems is one of the most basic aspects of understanding human intelligence. By representing objects as nodes and relations as edges, we can perform GNN-based reasoning about objects, relations, and physics in a simplified but effective way.

[4] proposed *Interaction Networks* to make predictions and inferences about various physical systems. The model takes objects and relations as input, reasons about their interactions, and applies the effects and physical dynamics to predict new states. They separately model relation-centric and object-centric models, making it easier to generalize across different systems. In CommNet [93], interactions are not modeled explicitly. Instead, an interaction vector is obtained by averaging all other agents' hidden vectors. VAIN [90] further introduced attentional methods into agent interaction process, which preserves both the complexity advantages and computational efficiency as well.

*Visual Interaction Networks* [91] could make predictions from pixels. It learns a state code from two consecutive input frames for each object. Then, after adding their interaction effect by an Interaction Net block, the state decoder converts state codes to next step's state.

[3] proposed a Graph Network based model which could either perform state prediction or inductive inference. The inference model takes partially observed information as input and constructs a hidden graph for implicit system classification.

#### 3.1.2 Chemistry and Biology

**Molecular Fingerprints** Calculating molecular fingerprints, which means feature vectors that represent moleculars, is a core step in computer-aided drug design. Conventional molecular fingerprints are hand-made and fixed. By applying GNN to molecular graphs, we can obtain better fingerprints.

[51] proposed *neural graph fingerprints* which calculate substructure feature vectors via GCN and sum to get overall representation. The aggregation function is

$$\mathbf{h}_{\mathcal{N}_v}^t = \sum_{u \in \mathcal{N}(v)} \text{CONCAT}(\mathbf{h}_u^t, \mathbf{e}_{uv}) \tag{44}$$

Where $\mathbf{e}_{uv}$ is the edge feature of edge $(u, v)$. Then update node representation by

$$\mathbf{h}_v^{t+1} = \sigma(\mathbf{W}_t^{deg(v)} \mathbf{h}_{\mathcal{N}_v}^t) \tag{45}$$

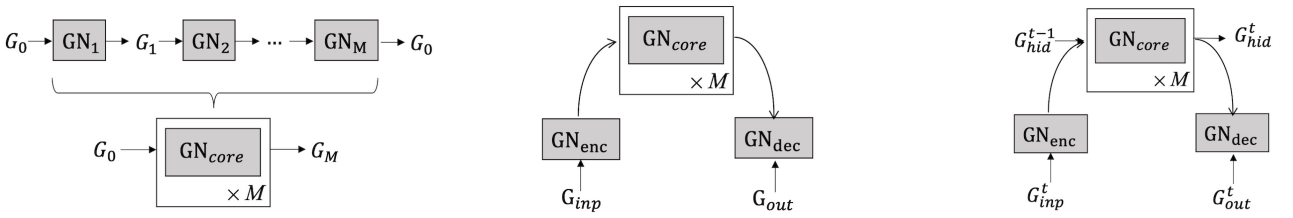
Where $deg(v)$ is the degree of node $v$ and $\mathbf{W}_t^N$ is a learned matrix for each time step $t$ and node degree $N$.

[99] further explicitly models atom and atom pairs independently to emphasize atom interactions. It introduces edge representation $\mathbf{e}_{uv}^t$ instead of aggregation function, i.e. $\mathbf{h}_{\mathcal{N}_v}^t = \sum_{u \in \mathcal{N}(v)} \mathbf{e}_{uv}^t$. The node update function is

$$\mathbf{h}_v^{t+1} = \text{ReLU}(\mathbf{W}_1(\text{ReLU}(\mathbf{W}_0 \mathbf{h}_u^t), \mathbf{h}_{\mathcal{N}_v}^t)) \tag{46}$$

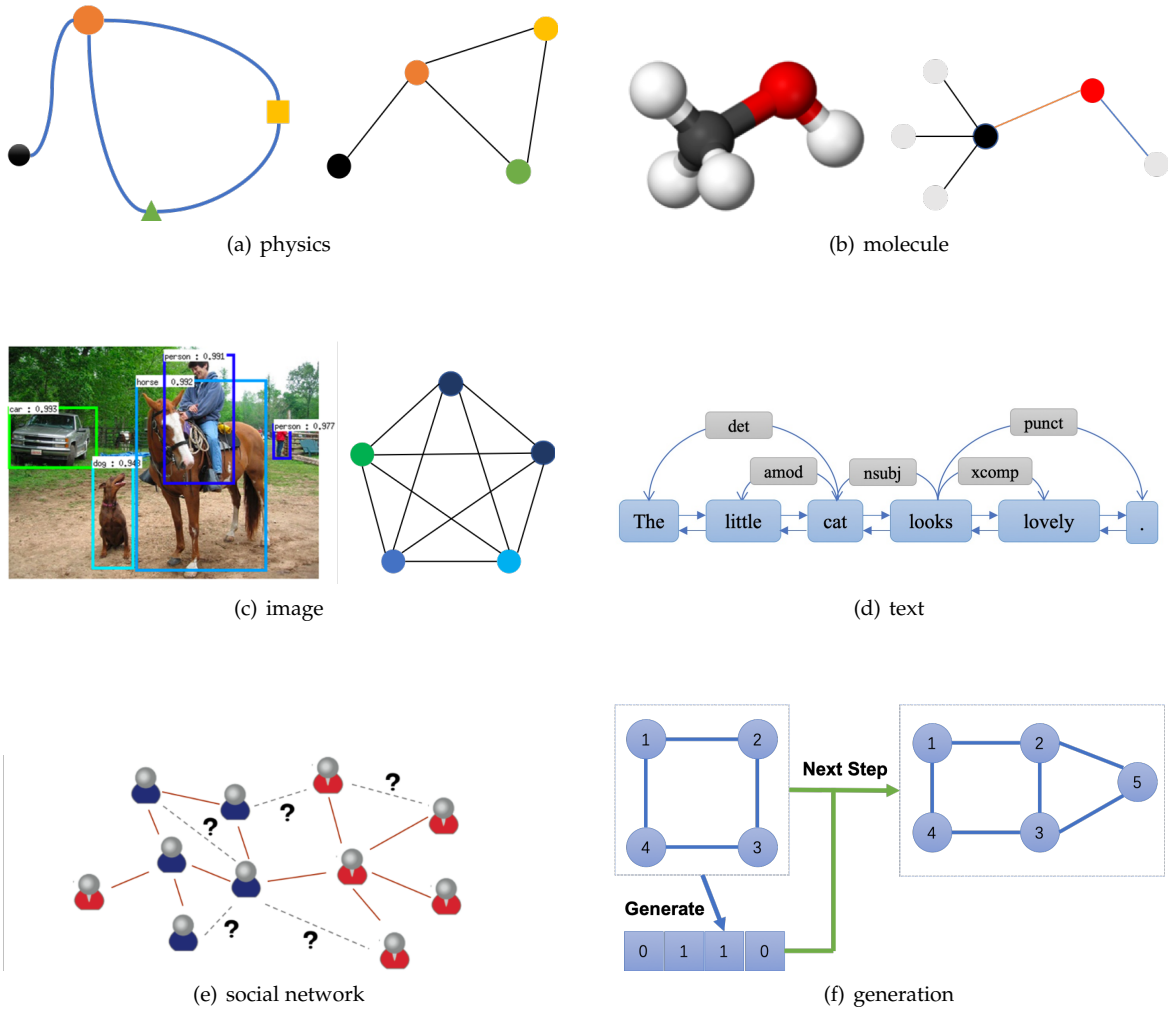
TABLE 3
Applications of graph neural networks.

| Area | Application | Algorithm | Deep Learning Model | References |
|---|---|---|---|---|
| Text | Text classification | GCN | Graph Convolutional Network | [1], [23], [48] [2], [22], [46] |
| | | GAT | Graph Attention Network | [68] |
| | | DGCNN | Graph Convolutional Network | [106] |
| | | Text GCN | Graph Convolutional Network | [107] |
| | | Sentence LSTM | Graph LSTM | [62] |
| | Sequence Labeling (POS, NER) | Sentence LSTM | Graph LSTM | [62] |
| | Sentiment classification | Tree LSTM | Graph LSTM | [60] |
| | Semantic role labeling | Syntactic GCN | Graph Convolutional Network | [108] |
| | Neural machine translation | Syntactic GCN | Graph Convolutional Network | [109], [110] |
| | | GGNN | Gated Graph Neural Network | [38] |
| | Relation extraction | Tree LSTM | Graph LSTM | [111] |
| | | Graph LSTM | Graph LSTM | [44], [112] |
| | | GCN | Graph Convolutional Network | [113] |
| | Event extraction | Syntactic GCN | Graph Convolutional Network | [114], [115] |
| | AMR to text generation | Sentence LSTM | Graph LSTM | [116] |
| | | GGNN | Gated Graph Neural Network | [38] |
| | Multi-hop reading comprehension | Sentence LSTM | Graph LSTM | [117] |
| | Relational reasoning | RN | MLP | [96] |
| | | Recurrent RN | Recurrent Neural Network | [118] |
| | | IN | Graph Neural Network | [4] |
| Image | Social Relationship Understanding | GRM | Gated Graph Neural Network | [119] |
| | Image classification | GCN | Graph Convolutional Network | [120], [121] |
| | | GGNN | Gated Graph Neural Network | [122] |
| | | DGP | Graph Convolutional Network | [35] |
| | | GSNN | Gated Graph Neural Network | [123] |
| | Visual Question Answering | GGNN | Gated Graph Neural Network | [119], [124], [125] |
| | Object Detection | RN | Graph Attention Network | [126], [127] |
| | Interaction Detection | GPNN | Graph Neural Network | [128] |
| | | Structural-RNN | Graph Neural Network | [42] |
| | Region Classification | GCNN | Graph CNN | [129] |
| | Semantic Segmentation | Graph LSTM | Graph LSTM | [63], [130] |
| | | GGNN | Gated Graph Neural Network | [131] |
| | | DGCNN | Graph CNN | [132] |
| | | 3DGNN | Graph Neural Network | [133] |
| Science | Physics Systems | IN | Graph Neural Network | [4] |
| | | VIN | Graph Neural Network | [91] |
| | | GN | Graph Networks | [3] |
| | Molecular Fingerprints | NGF | Graph Convolutional Network | [51] |
| | | GCN | Graph Convolutional Network | [99] |
| | Protein Interface Prediction | GCN | Graph Convolutional Network | [5] |
| | Side Effects Prediction | Decagon | Graph Convolutional Network | [134] |
| | Disease Classification | PPIN | Graph Convolutional Network | [135] |
| Knowledge Graph | KB Completion | GNN | Graph Neural Network | [6] |
| | KG Alignment | GCN | Graph Convolutional Network | [136] |
| Combinatorial Optimization | | structure2vec | Graph Convolutional Network | [7] |
| | | GNN | Graph Neural Network | [137] |
| | | GCN | Graph Convolutional Network | [138] |
| | | AM | Graph Attention Network | [139] |
| Graph Generation | | NetGAN | Long short-term memory | [140] |
| | | GraphRNN | Rucurrent Neural Network | [137] |
| | | Regularizing VAE | Variational Autoencoder | [141] |
| | | GCPN | Graph Convolutional Network | [142] |
| | | MolGAN | Relational-GCN | [143] |




(a) Sequential GN blocks (b) Encode-process-decode (c) Recurrent GN blocks

Fig. 3. Examples of architectures composed by GN blocks. (a) The sequential processing architecture; (b) The encode-process-decode architecture; (c) The recurrent architecture.




(a) physics (b) molecule

(c) image (d) text

(e) social network (f) generation

Fig. 4. Application Scenarios

while the edge update function is

$$\mathbf{e}_{uv}^{t+1} = \text{ReLU}(\mathbf{W}_4(\text{ReLU}(\mathbf{W}_2\mathbf{e}_{uv}^t), \text{ReLU}(\mathbf{W}_3(\mathbf{h}_v^t, \mathbf{h}_u^t)))) \tag{47}$$

**Protein Interface Prediction** [5] focused on the task named protein interface prediction, which is a challenging problem with important applications in drug discovery and design. The proposed GCN based method respectively learns ligand and receptor protein residue representation and merges them for pairwise classification.

GNN can also be used in biomedical engineering. With Protein-Protein Interaction Network, [135] leverages graph convolution and relation network for breast cancer subtype classification. [134] also suggests a GCN based model for polypharmacy side effects prediction. Their work models the drug and protein interaction network and separately deals with edges in different types.

### 3.1.3 *Knowledge graph*

[6] utilizes GNNs to solve the out-of-knowledge-base (OOKB) entity problem in knowledge base completion (KBC). The OOKB entities in [6] are directly connected to

the existing entities thus the embeddings of OOKB entities can be aggregated from the existing entities. The method achieves satisfying performance both in the standard KBC setting and the OOKB setting.

[136] utilizes GCNs to solve the cross-lingual knowledge graph alignment problem. The model embeds entities from different languages into a unified embedding space and aligns them based on the embedding similarity.

### 3.2 Non-structural Scenarios

In this section we will talk about applications on non-structural scenarios such as image, text, programming source code [59], [145] and multi-agent systems [90], [93], [103]. We will only give detailed introduction to the first two scenarios due to the length limit. Roughly, there are two ways to apply the graph neural networks on non-structural scenarios: (1) Incorporate structural information from other domains to improve the performance, for example using information from knowledge graphs to alleviate the zero-shot problems in image tasks; (2) Infer or assume the relational structure in the scenario and then apply the model to solve the problems defined on graphs, such as the method in [62] which models text into graphs.

#### *3.2.1 Image*

**Image Classification** Image classification is a very basic and important task in the field of computer vision, which attracts much attention and has many famous datasets like ImageNet [146]. Recent progress in image classification benefits from big data and the strong power of GPU computation, which allows us to train a classifier without extracting structural information from images. However, **zero-shot and few-shot learning** become more and more popular in the field of image classification, because most models can achieve similar performance with enough data. There are several works leveraging graph neural networks to incorporate structural information in image classification. First, knowledge graphs can be used as extra information to guide zero-short recognition classification [35], [121]. [121] builds a knowledge graph where each node corresponds to an object category and takes the word embeddings of nodes as input for predicting the classifier of different categories. As over-smoothing effect happens with the deep depth of convolution architecture, the 6-layer GCN used in [121] would wash out much useful information in the representation. To solve the smoothing problem in the propagation of GCN, [35] managed to use single layer GCN with a larger neighborhood which includes both one-hop and multi-hops nodes in the graph. And it proved effective in building a zero-shot classifier with existing ones.

Except for the knowledge graph, the similarity between images in the dataset is also helpful for the few-shot learning [120]. [120] proposed to build a weighted full-connected image network based on the similarity and do message passing in the graph for few-shot recognition. As most knowledge graphs are large for reasoning, [123] selects some related entities to build a sub-graph based on the result of object detection and applies GGNN to the extracted graph for prediction. Besides, [122] proposed to construct a new knowledge graph where the entities are all the categories. And, they defined three types of label relations: super-subordinate, positive correlation, and negative correlation and propagate the confidence of labels in the graph directly.

**Visual Reasoning** Computer-vision systems usually need to perform reasoning by incorporating both spatial and semantic information. So it is natural to generate graphs for reasoning tasks.

A typical visual reasoning task is visual question answering(VQA), [124] respectively constructs image scene graph and question syntactic graph. Then it applies GGNN to train the embeddings for predicting the final answer. Despite spatial connections among objects, [147] builds the relational graphs conditioned on the questions. With knowledge graphs, [119], [125] could perform finer relation exploration and more interpretable reasoning process.

Other applications of visual reasoning include object detection, interaction detection, and region classification. In object detection [126], [127], GNNs are used to calculate RoI features; In interaction detection [42], [128], GNNs are message passing tools between human and objects; In region classification [129], GNNs perform reasoning on graphs which connects regions and classes.

**Semantic Segmentation** Semantic segmentation is a crucial step toward image understanding. The task here is to assign a unique label (or category) to every single pixel in the image, which can be considered as a dense classification problem. However, regions in images are often not grid-like and need non-local information, which leads to the failure of traditional CNN. Several works utilized graph-structured data to handle it.

[63] proposed Graph-LSTM to model long-term dependency together with spatial connections by building graphs in form of distance-based superpixel map and applying LSTM to propagate neighborhood information globally. Subsequent work improved it from the perspective of encoding hierarchical information [130].

Furthermore, 3D semantic segmentation (RGBD semantic segmentation) and point clouds classification utilize more geometric information and therefore are hard to model by a 2D CNN. [133] constructs a K nearest neighbors (KNN) graph and uses a 3D GNN as propagation model. After unrolling for several steps, the prediction model takes the hidden state of each node as input and predict its semantic label.

As there are always too many points, [131] solved large-scale 3D point clouds segmentation by building superpoint graphs and generating embeddings for them. To classify supernodes, [131] leverages GGNN and graph convolution.

[132] proposed to model point interactions through edges. They calculate edge representation vectors by feeding the coordinates of its terminal nodes. Then node embeddings are updated by edge aggregation.

#### 3.2.2 Text

The graph neural networks could be applied to several tasks based on texts. It could be applied to both sentence-level tasks(e.g. text classification) as well as word-level tasks(e.g. sequence labeling). We will introduce several major applications on text in the following.

**Text classification** Text classification is an important and classical problem in natural language processing. The classical GCN models [1], [2], [22], [23], [46], [48] and GAT model [68] are applied to solve the problem, but they only use the structural information between the documents and they don't use much text information. [106] proposed a graph-CNN based deep learning model to first convert texts to graph-of-words, and then use graph convolution operations in [53] to convolve the word graph. [62] proposed the Sentence LSTM to encode text. It views the whole sentence as a single state, which consists of sub-states for individual words and an overall sentence-level state. It uses the global sentence-level representation for classification tasks. These methods either view a document or a sentence as a graph of word nodes or rely on the document citation relation to construct the graph. [107] regards the documents and words as nodes to construct the corpus graph (hence heterogeneous graph) and uses the Text GCN to learn embeddings of words and documents. Sentiment classification could also be regarded as a text classification problem and a Tree-LSTM approach is proposed by [60].

**Sequence labeling** As each node in GNNs has its hidden state, we can utilize the hidden state to address the sequence labeling problem if we consider every word in the sentence as a node. [62] utilizes the Sentence LSTM to label the sequence. It has conducted experiments on POS-tagging and NER tasks and achieves promising performance.

Semantic role labeling is another task of sequence labeling. [108] proposed a Syntactic GCN to solve the problem. The Syntactic GCN which operates on the direct graph with labeled edges is a special variant of the GCN [2]. It integrates edge-wise gates which let the model regulate contributions of individual dependency edges. The Syntactic GCNs over syntactic dependency trees are used as sentence encoders to learn latent feature representations of words in the sentence. [108] also reveals that GCNs and LSTMs are functionally complementary in the task.

**Neural machine translation** The neural machine translation task is usually considered as a sequence-to-sequence task. [66] introduces the attention mechanisms and replaces the most commonly used recurrent or convolutional layers. In fact, the Transformer assumes a fully connected graph structure between linguistic entities.

One popular application of GNN is to incorporate the syntactic or semantic information into the NMT task. [109] utilizes the Syntactic GCN on syntax-aware NMT tasks. [110] incorporates information about the predicate-argument structure of source sentences (namely, semantic-role representations) using Syntactic GCN and compares the results of incorporating only syntactic or semantic information or both of the information into the task. [38] utilizes the GGNN in syntax-aware NMT. It converts the syntactic dependency graph into a new structure called the Levi graph by turning the edges into additional nodes and thus edge labels can be represented as embeddings.

**Relation extraction** Extracting semantic relations between entities in texts is an important and well-studied task. Some systems treat this task as a pipeline of two separated tasks, named entity recognition and relation extraction. [111] proposed an end-to-end relation extraction model by using bidirectional sequential and tree-structured LSTM-RNNs. [113] proposed an extension of graph convolutional networks that is tailored for relation extraction and applied a pruning strategy to the input trees.

Cross-sentence N-ary relation extraction detects relations among $n$ entities across multiple sentences. [44] explores a general framework for cross-sentence $n$-ary relation extraction based on graph LSTMs. It splits the input graph into two DAGs while important information could be lost in the splitting procedure. [112] proposed a graph-state LSTM model. It keeps the original graph structure and speeds up computation by allowing more parallelization.

**Event extraction** Event extraction is an important information extraction task to recognize instances of specified types of events in texts. [114] investigates a convolutional neural network (which is the Syntactic GCN exactly) based on dependency trees to perform event detection. [115] proposed a Jointly Multiple Events Extraction (JMEE) framework to jointly extract multiple event triggers and arguments by introducing syntactic shortcut arcs to enhance information flow to attention-based graph convolution networks to model graph information.

**Other applications** GNNs could also be applied to many other applications. There are several works focus on the AMR to text generation task. A Sentence-LSTM based method [116] and a GGNN based method [38] have been proposed in this area. [60] uses the Tree LSTM to model the semantic relatedness of two sentences. And [117] exploits the Sentence LSTM to solve the multi-hop reading comprehension problem. Another important direction is relational reasoning, relational networks [96], interaction networks [4] and recurrent relational networks [118] are proposed to solve the relational reasoning task based on text. The works cited above are not an exhaustive list, and we encourage our readers to find more works and application domains of graph neural networks that they are interested in.

### 3.3 Other Scenarios

Besides structural and non-structural scenarios, there are some other scenarios where graph neural networks play an important role. In this subsection, we will introduce generative graph models and combinatorial optimization with GNNs.

#### 3.3.1 Generative Models

Generative models for real-world graphs has drawn significant attention for its important applications including modeling social interactions, discovering new chemical structures, and constructing knowledge graphs. As deep learning

methods have powerful ability to learn the implicit distribution of graphs, there is a surge in neural graph generative models recently.

NetGAN [140] is one of the first work to build neural graph generative model, which generates graphs via random walks. It transformed the problem of graph generation to the problem of walk generation which takes the random walks from a specific graph as input and trains a walk generative model using GAN architecture. While the generated graph preserves important topological properties of the original graph, the number of nodes is unable to change in the generating process, which is as same as the original graph. GraphRNN [148] managed to generate the adjacency matrix of a graph by generating the adjacency vector of each node step by step, which can output required networks having different numbers of nodes.

Instead of generating adjacency matrix sequentially, MolGAN [143] predicts discrete graph structure (the adjacency matrix) at once and utilizes a permutation-invariant discriminator to solve the node variant problem in the adjacency matrix. Besides, it applies a reward network for RL-based optimization towards desired chemical properties. What's more, [141] proposed constrained variational autoencoders to ensure the semantic validity of generated graphs. And, GCPN [142] incorporated domain-specific rules through reinforcement learning.

[149] proposed a model which generates edges and nodes sequentially and utilizes a graph neural network to extract the hidden state of the current graph which is used to decide the action in the next step during the sequential generative process.

#### 3.3.2 Combinatorial Optimization

Combinatorial optimization problems over graphs are set of NP-hard problems which attract much attention from scientists of all fields. Some specific problems like traveling salesman problem (TSP) and minimum spanning trees (MST) have got various heuristic solutions. Recently, using a deep neural network for solving such problems has been a hotspot, and some of the solutions further leverage graph neural network because of their graph structure.

[150] first proposed a deep-learning approach to tackle TSP. Their method consists of two parts: a Pointer Network [151] for parameterizing rewards and a policy gradient [152] module for training. This work has been proved to be comparable with traditional approaches. However, Pointer Networks are designed for sequential data like texts, while order-invariant encoders are more appropriate for such work.

[153] and [139] improved the above method by including graph neural networks. The former work first obtain the node embeddings from structure2vec [94] then feed them into a Q-learning module for making decisions. The latter one builds an attention-based encoder-decoder system. By replacing reinforcement learning module with a attention-based decoder, it is more efficient for training. These work achieved better performance than previous algorithms, which proved the representation power of graph neural networks.

[137] focused on Quadratic Assignment Problem i.e. measuring the similarity of two graphs. The GNN based model learns node embeddings for each graph independently and matches them using attention mechanism. This method offers intriguingly good performance even in regimes where standard relaxation-based techniques appear to suffer.

## 4 Open problems

Although GNNs have achieved great success in different fields, it is remarkable that GNN models are not good enough to offer satisfying solutions for any graph in any condition. In this section, we will state some open problems for further researches.

**Shallow Structure** Traditional deep neural networks can stack hundreds of layers to get better performance, because deeper structure has more parameters, which improve the expressive power significantly. However, graph neural networks are always shallow, most of which are no more than three layers. As experiments in [82] show, stacking multiple GCN layers will result in over-smoothing, that is to say, all vertices will converge to the same value. Although some researchers have managed to tackle this problem [59], [82], it remains to be the biggest limitation of GNN. Designing real deep GNN is an exciting challenge for future research, and will be a considerable contribution to the understanding of GNN.

**Dynamic Graphs** Another challenging problem is how to deal with graphs with dynamic structures. Static graphs are stable so they can be modeled feasibly, while dynamic graphs introduce changing structures. When edges and nodes appear or disappear, GNN can not change adaptively. Dynamic GNN is being actively researched on and we believe it to be a big milestone about the stability and adaptability of general GNN.

**Non-Structural Scenarios** Although we have discussed the applications of GNN on non-structural scenarios, we found that there is no optimal methods to generate graphs from raw data. In image domain, some work utilizes CNN to obtain feature maps then upsamples them to form superpixels as nodes [63], while other ones directly leverage some object detection algorithms to get object nodes. In text domain [129], some work employs syntactic trees as syntactic graphs while others adopt fully connected graphs. Therefore, finding the best graph generation approach will offer a wider range of fields where GNN could make contribution.

**Scalability** How to apply embedding methods in web-scale conditions like social networks or recommendation systems has been a fatal problem for almost all graph embedding algorithms, and GNN is not an exception. Scaling up GNN is difficult because many of the core steps are computational consuming in big data environment. There are several examples about this phenomenon: First, graph data are not regular Euclidean, each node has its own

neighborhood structure so batches can not be applied. Then, calculating graph Laplacian is also unfeasible when there are millions of nodes and edges. Moreover, we need to point out that scaling determines whether an algorithm is able to be applied into practical use. Several work has proposed their solutions to this problem [77] and we are paying close attention to the progress.

## 5 Conclusion

Over the past few years, graph neural networks have become powerful and practical tools for machine learning tasks in graph domain. This progress owes to advances in expressive power, model flexibility, and training algorithms. In this survey, we conduct a comprehensive review of graph neural networks. For GNN models, we introduce its variants categorized by graph types, propagation types, and training types. Moreover, we also summarize several general frameworks to uniformly represent different variants. In terms of application taxonomy, we divide the GNN applications into structural scenarios, non-structural scenarios, and other scenarios, then give a detailed review for applications in each scenario. Finally, we suggest four open problems indicating the major challenges and future research directions of graph neural networks, including model depth, scalability, the ability to deal with dynamic graphs and non-structural scenarios.

## References


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

[2] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *ICLR 2017*, 2017.

[3] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, "Graph networks as learnable physics engines for inference and control," *arXiv preprint arXiv:1806.01242*, 2018.

[4] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende *et al.*, "Interaction networks for learning about objects, relations and physics," in *NIPS 2016*, 2016, pp. 4502–4510.

[5] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein interface prediction using graph convolutional networks," in *NIPS 2017*, 2017, pp. 6530–6539.

[6] T. Hamaguchi, H. Oiwa, M. Shimbo, and Y. Matsumoto, "Knowledge transfer for out-of-knowledge-base entities : A graph neural network approach," in *IJCAI 2017*, 2017, pp. 1802–1808.

[7] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *arXiv preprint arXiv:1704.01665*, 2017.

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature 2015*, vol. 521, no. 7553, p. 436, 2015.

[10] F. R. Chung and F. C. Graham, *Spectral graph theory*. American Mathematical Soc., 1997, no. 92.

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

[12] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications." *IEEE Data(base) Engineering Bulletin*, vol. 40, pp. 52–74, 2017.

[13] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Network representation learning: A survey," *IEEE transactions on Big Data*, 2018.

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

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

[16] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[17] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *SIGKDD 2014*. ACM, 2014, pp. 701–710.

[18] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *SIGKDD*. ACM, 2016, pp. 855–864.

[19] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *WWW 2015*, 2015, pp. 1067–1077.

[20] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information." in *IJCAI 2015*, 2015, pp. 2111–2117.

[21] T. Kawamoto, M. Tsubaki, and T. Obuchi, "Mean-field theory of graph neural networks in graph partitioning," in *NeurIPS 2018*, 2018, pp. 4366–4376.

[22] 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," *CVPR 2017*, pp. 5425–5434, 2017.

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

[24] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on riemannian manifolds," in *ICCV workshops 2015*, 2015, pp. 37–45.

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

[26] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *IEEE SPM 2017*, vol. 34, no. 4, pp. 18–42, 2017.

[27] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *arXiv preprint arXiv:1704.01212*, 2017.

[28] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," *arXiv preprint arXiv:1711.07971*, vol. 10, 2017.

[29] 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.

[30] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[31] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *arXiv preprint arXiv:1812.04202*, 2018.

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

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

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

[35] M. Kampffmeyer, Y. Chen, X. Liang, H. Wang, Y. Zhang, and E. P. Xing, "Rethinking knowledge graph propagation for zero-shot learning," *arXiv preprint arXiv:1805.11724*, 2018.

[36] Y. Zhang, Y. Xiong, X. Kong, S. Li, J. Mi, and Y. Zhu, "Deep collective classification in heterogeneous information networks," in *WWW 2018*, 2018, pp. 399–408.

[37] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," *WWW 2019*, 2019.

[38] D. Beck, G. Haffari, and T. Cohn, "Graph-to-sequence learning using gated graph neural networks," in *ACL 2018*, 2018, pp. 273–283.

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

[40] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," *arXiv preprint arXiv:1707.01926*, 2017.

[41] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," *arXiv preprint arXiv:1709.04875*, 2017.

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

[43] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[44] N. Peng, H. Poon, C. Quirk, K. Toutanova, and W.-t. Yih, "Cross-sentence n-ary relation extraction with graph lstms," *arXiv preprint arXiv:1708.03743*, 2017.

[45] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," *ICLR 2014*, 2014.

[46] M. Henaff, J. Bruna, and Y. Lecun, "Deep convolutional networks on graph-structured data." *arXiv: Learning*, 2015.

[47] 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.

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

[49] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *AAAI 2018*, 2018.

[50] Y. C. Ng, N. Colombo, and R. Silva, "Bayesian semi-supervised learning with graph gaussian processes," in *NeurIPS 2018*, 2018, pp. 1690–1701.

[51] D. K. Duvenaud, D. Maclaurin, J. Aguileraiparraguirre, R. Gomezbombarelli, T. D. Hirzel, A. Aspuruguzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," *NIPS 2015*, pp. 2224–2232, 2015.

[52] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *WWW 2018*, 2018.

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

[54] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proceedings of SIGKDD*. ACM, 2018, pp. 1416–1424.

[55] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *ECCV 2016*. Springer, 2016, pp. 630–645.

[56] J. Chang, J. Gu, L. Wang, G. Meng, S. Xiang, and C. Pan, "Structure-aware convolutional neural networks," in *NeurIPS 2018*, 2018, pp. 11–20.

[57] 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," *EMNLP 2014*, pp. 1724–1734, 2014.

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

[59] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," *arXiv: Learning*, 2016.

[60] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *IJCNLP 2015*, pp. 1556–1566, 2015.

[61] V. Zayats and M. Ostendorf, "Conversation modeling on reddit using a graph-structured lstm," *TACL 2018*, vol. 6, pp. 121–132, 2018.

[62] Y. Zhang, Q. Liu, and L. Song, "Sentence-state lstm for text representation," *ACL 2018*, vol. 1, pp. 317–327, 2018.

[63] X. Liang, X. Shen, J. Feng, L. Lin, and S. Yan, "Semantic object parsing with graph lstm," *ECCV 2016*, pp. 125–143, 2016.

[64] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *ICLR 2015*, 2015.

[65] J. Gehring, M. Auli, D. Grangier, and Y. N. Dauphin, "A convolutional encoder model for neural machine translation," *ACL 2017*, vol. 1, pp. 123–135, 2017.

[66] A. Vaswani, N. Shazeer, N. Parmar, L. Jones, J. Uszkoreit, A. N. Gomez, and L. Kaiser, "Attention is all you need," *NIPS 2017*, pp. 5998–6008, 2017.

[67] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," *EMNLP 2016*, pp. 551–561, 2016.

[68] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *ICLR 2018*, 2018.

[69] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, "Gaan: Gated attention networks for learning on large and spatiotemporal graphs," *arXiv preprint arXiv:1803.07294*, 2018.

[70] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CVPR 2016*, pp. 770–778, 2016.

[71] A. Rahimi, T. Cohn, and T. Baldwin, "Semi-supervised user geolocation via graph convolutional networks," *ACL 2018*, vol. 1, pp. 2009–2019, 2018.

[72] J. G. Zilly, R. K. Srivastava, J. Koutnik, and J. Schmidhuber, "Recurrent highway networks." *ICML 2016*, pp. 4189–4198, 2016.

[73] T. Pham, T. Tran, D. Phung, and S. Venkatesh, "Column networks for collective classification," in *AAAI 2017*, 2017.

[74] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," *ICML 2018*, pp. 5449–5458, 2018.

[75] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3693–3702.

[76] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *NeurIPS 2018*, 2018, pp. 4805–4815.

[77] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *SIGKDD 2018*, 2018.

[78] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," *arXiv preprint arXiv:1801.10247*, 2018.

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

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

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

[82] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," *arXiv preprint arXiv:1801.07606*, 2018.

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

[84] R. van den Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *arXiv preprint arXiv:1706.02263*, 2017.

[85] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding," *arXiv preprint arXiv:1802.04407*, 2018.

[86] W. Yu, C. Zheng, W. Cheng, C. C. Aggarwal, D. Song, B. Zong, H. Chen, and W. Wang, "Learning deep network representations with adversarially regularized autoencoders," in *SIGKDD 2018*, 2018.

[87] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations," in *AAAI 2016*, 2016.

[88] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *SIGKDD 2016*, 2016.

[89] K. Tu, P. Cui, X. Wang, P. S. Yu, and W. Zhu, "Deep recursive network embedding with regular equivalence," in *SIGKDD 2018*, 2018.

[90] Y. Hoshen, "Vain: Attentional multi-agent predictive modeling," in *NIPS 2017*, 2017, pp. 2701–2711.

[91] N. Watters, D. Zoran, T. Weber, P. Battaglia, R. Pascanu, and A. Tacchetti, "Visual interaction networks: Learning a physics simulator from video," in *NIPS 2017*, 2017, pp. 4539–4547.

[92] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum, "A compositional object-based approach to learning physical dynamics," *arXiv preprint arXiv:1612.00341*, 2016.

[93] S. Sukhbaatar, R. Fergus *et al.*, "Learning multiagent communication with backpropagation," in *NIPS 2016*, 2016, pp. 2244–2252.

[94] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *ICML 2016*, 2016, pp. 2702–2711.

[95] D. Raposo, A. Santoro, D. Barrett, R. Pascanu, T. Lillicrap, and P. Battaglia, "Discovering objects and their relations from entangled scene representations," *arXiv preprint arXiv:1702.05068*, 2017.

[96] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap, "A simple neural network module for relational reasoning," in *NIPS 2017*, 2017, pp. 4967–4976.

[97] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *NIPS 2017*, 2017, pp. 3391–3401.

[98] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *CVPR 2017*, vol. 1, no. 2, p. 4, 2017.

[99] 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.

[100] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko, "Quantum-chemical insights from deep tensor neural networks," *Nature communications*, vol. 8, p. 13890, 2017.

[101] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *CVPR 2005*, vol. 2. IEEE, 2005, pp. 60–65.

[102] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Computer Vision 1998*. IEEE, 1998, pp. 839–846.

[103] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel, "Neural relational inference for interacting systems," *arXiv preprint arXiv:1802.04687*, 2018.

[104] J. B. Hamrick, K. R. Allen, V. Bapst, T. Zhu, K. R. McKee, J. B. Tenenbaum, and P. W. Battaglia, "Relational inductive bias for physical construction in humans and machines," *arXiv preprint arXiv:1806.01203*, 2018.

[105] T. Wang, R. Liao, J. Ba, and S. Fidler, "Nervenet: Learning structured policy with graph neural networks," 2018.

[106] H. Peng, J. Li, Y. He, Y. Liu, M. Bao, L. Wang, Y. Song, and Q. Yang, "Large-scale hierarchical text classification with recursively regularized deep graph-cnn," in *WWW 2018*, 2018, pp. 1063–1072.

[107] L. Yao, C. Mao, and Y. Luo, "Graph convolutional networks for text classification," *arXiv preprint arXiv:1809.05679*, 2018.

[108] D. Marcheggiani and I. Titov, "Encoding sentences with graph convolutional networks for semantic role labeling," in *Proceedings of EMNLP*, 2017, pp. 1506–1515.

[109] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, and K. Simaan, "Graph convolutional encoders for syntax-aware neural machine translation," *EMNLP 2017*, pp. 1957–1967, 2017.

[110] D. Marcheggiani, J. Bastings, and I. Titov, "Exploiting semantics in neural machine translation with graph convolutional networks," *arXiv preprint arXiv:1804.08313*, 2018.

[111] M. Miwa and M. Bansal, "End-to-end relation extraction using lstms on sequences and tree structures," *arXiv preprint arXiv:1601.00770*, 2016.

[112] L. Song, Y. Zhang, Z. Wang, and D. Gildea, "N-ary relation extraction using graph state lstm," *arXiv preprint arXiv:1808.09101*, 2018.

[113] Y. Zhang, P. Qi, and C. D. Manning, "Graph convolution over pruned dependency trees improves relation extraction," *arXiv preprint arXiv:1809.10185*, 2018.

[114] T. H. Nguyen and R. Grishman, "Graph convolutional networks with argument-aware pooling for event detection," 2018.

[115] X. Liu, Z. Luo, and H. Huang, "Jointly multiple events extraction via attention-based graph information aggregation," *arXiv preprint arXiv:1809.09078*, 2018.

[116] L. Song, Y. Zhang, Z. Wang, and D. Gildea, "A graph-to-sequence model for amr-to-text generation," *arXiv preprint arXiv:1805.02473*, 2018.

[117] L. Song, Z. Wang, M. Yu, Y. Zhang, R. Florian, and D. Gildea, "Exploring graph-structured passage representation for multi-hop reading comprehension with graph neural networks," *arXiv preprint arXiv:1809.02040*, 2018.

[118] R. B. Palm, U. Paquet, and O. Winther, "Recurrent relational networks," *NeurIPS 2018*, 2018.

[119] Z. Wang, T. Chen, J. Ren, W. Yu, H. Cheng, and L. Lin, "Deep reasoning with knowledge graph for social relationship understanding," *arXiv preprint arXiv:1807.00504*, 2018.

[120] V. Garcia and J. Bruna, "Few-shot learning with graph neural networks," *arXiv preprint arXiv:1711.04043*, 2017.

[121] X. Wang, Y. Ye, and A. Gupta, "Zero-shot recognition via semantic embeddings and knowledge graphs," in *CVPR 2018*, 2018, pp. 6857–6866.

[122] C. Lee, W. Fang, C. Yeh, and Y. F. Wang, "Multi-label zero-shot learning with structured knowledge graphs," in *Proceedings of CVPR*, 2018, pp. 1576–1585.

[123] K. Marino, R. Salakhutdinov, and A. Gupta, "The more you know: Using knowledge graphs for image classification," in *Proceedings of CVPR*, 2017, pp. 20–28.

[124] D. Teney, L. Liu, and A. V. Den Hengel, "Graph-structured representations for visual question answering," in *Proceedings of CVPR*, 2017, pp. 3233–3241.

[125] M. Narasimhan, S. Lazebnik, and A. G. Schwing, "Out of the box: Reasoning with graph convolution nets for factual visual question answering," in *Proceedings of NeurIPS*, 2018, pp. 2654–2665.

[126] H. Hu, J. Gu, Z. Zhang, J. Dai, and Y. Wei, "Relation networks for object detection," in *CVPR 2018*, vol. 2, no. 3, 2018.

[127] J. Gu, H. Hu, L. Wang, Y. Wei, and J. Dai, "Learning region features for object detection," *arXiv preprint arXiv:1803.07066*, 2018.

[128] S. Qi, W. Wang, B. Jia, J. Shen, and S.-C. Zhu, "Learning human-object interactions by graph parsing neural networks," *arXiv preprint arXiv:1808.07962*, 2018.

[129] X. Chen, L.-J. Li, L. Fei-Fei, and A. Gupta, "Iterative visual reasoning beyond convolutions," *arXiv preprint arXiv:1803.11189*, 2018.

[130] X. Liang, L. Lin, X. Shen, J. Feng, S. Yan, and E. P. Xing, "Interpretable structure-evolving lstm," in *CVPR 2017*, 2017, pp. 2175–2184.

[131] L. Landrieu and M. Simonovsky, "Large-scale point cloud semantic segmentation with superpoint graphs," *arXiv preprint arXiv:1711.09869*, 2017.

[132] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *arXiv preprint arXiv:1801.07829*, 2018.

[133] X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun, "3d graph neural networks for rgbd semantic segmentation," in *CVPR 2017*, 2017, pp. 5199–5208.

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

[135] S. Rhee, S. Seo, and S. Kim, "Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification," *arXiv preprint arXiv:1711.05859*, 2017.

[136] Z. Wang, Q. Lv, X. Lan, and Y. Zhang, "Cross-lingual knowledge graph alignment via graph convolutional networks," in *EMNLP 2018*, 2018, pp. 349–357.

[137] A. Nowak, S. Villar, A. S. Bandeira, and J. Bruna, "Revised note on learning quadratic assignment with graph neural networks," in *IEEE DSW 2018*. IEEE, 2018, pp. 1–5.

[138] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *NeurIPS 2018*, 2018, pp. 537–546.

[139] W. Kool and M. Welling, "Attention solves your tsp," *arXiv preprint arXiv:1803.08475*, 2018.

[140] O. Shchur, D. Zugner, A. Bojchevski, and S. Gunnemann, "Netgan: Generating graphs via random walks," in *Proceedings of ICML*, 2018, pp. 609–618.

[141] T. Ma, J. Chen, and C. Xiao, "Constrained generation of semantically valid graphs via regularizing variational autoencoders," in *NeurIPS 2018*, 2018, pp. 7113–7124.

[142] J. You, B. Liu, R. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," *arXiv preprint arXiv:1806.02473*, 2018.

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

[144] Z. Cui, K. Henrickson, R. Ke, and Y. Wang, "Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting," 2018.

[145] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," *arXiv preprint arXiv:1711.00740*, 2017.

[146] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *IJCV 2015*, vol. 115, no. 3, pp. 211–252, 2015.

[147] W. Norcliffebrown, S. Vafeias, and S. Parisot, "Learning conditioned graph structures for interpretable visual question answering," in *Proceedings of NeurIPS*, 2018, pp. 8334–8343.

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

[149] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv preprint arXiv:1803.03324*, 2018.
[150] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," 2017.
[151] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *NIPS 2015*, 2015, pp. 2692–2700.
[152] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[153] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *NIPS 2017*, 2017, pp. 6348–6358.

## 6 Revision History

- Version 2 (2 Jan 2019). Add NuerIPS 2018 papers.
- Version 3 (7 Mar 2019). Add two coauthors and we thank them for their kindly suggestions and contributions.
- Version 4 (10 Jul 2019). Add more models and applications based on recent papers and update figures and references.