Graph-based Semi-supervised Learning: A Comprehensive Review

Zixing Song, Xiangli Yang, Zenglin Xu, *Senior Member, IEEE* Irwin King, *Fellow, IEEE,*

***Abstract*—Semi-supervised learning (SSL) has tremendous value in practice due to its ability to utilize both labeled data and unlabelled data. An important class of SSL methods is to naturally represent data as graphs such that the label information of unlabelled samples can be inferred from the graphs, which corresponds to graph-based semi-supervised learning (GSSL) methods. GSSL methods have demonstrated their advantages in various domains due to their uniqueness of structure, the universality of applications, and their scalability to large scale data. Focusing on this class of methods, this work aims to provide both researchers and practitioners with a solid and systematic understanding of relevant advances as well as the underlying connections among them. This makes our paper distinct from recent surveys that cover an overall picture of SSL methods while neglecting fundamental understanding of GSSL methods. In particular, a major contribution of this paper lies in a new generalized taxonomy for GSSL, including graph regularization and graph embedding methods, with the most up-to-date references and useful resources such as codes, datasets, and applications. Furthermore, we present several potential research directions as future work with insights into this rapidly growing field.**

***Index Terms*—Semi-supervised learning, graph-based semi-supervised learning, graph embedding, graph representation learning.**

I. Introduction

SEMI-SUPERVISED learning (SSL) has achieved great successes in various real-world applications where only a few expensive labeled samples are available and abundant unlabeled samples are easily obtained. Moreover, as a typical class of SSL solutions, graph-based SSL (GSSL) is very promising because the graph structure can be naturally used as a reflection for the significant manifold assumption in SSL. More specifically, GSSL methods start with constructing a graph where the nodes represent all the samples and the weighted edges reflect the similarity between a pair of nodes. This way of graph construction implies that nodes connected by edge associated with large weights tend to have the same label, which corresponds to the manifold assumption. The manifold assumption suggests that samples locating near to each other on a low-dimensional manifold should share similar labels. Consequently, the expressive power of graph structure under the manifold assumption contributes to the success of GSSL methods.

In the graph structure commonly used for SSL, each sample is represented by a node, and these nodes are connected by weighted edges that measure the similarity between them. Therefore, the main procedure of GSSL is to create a suitable graph along which the given labels can be easily propagated. To be precise, this goal can be achieved by the following two main steps.

Step 1. **Graph construction.** A similarity graph is constructed based on all the given data, including both the labeled and unlabeled samples. During this step, the biggest challenge is how to make the relationship between original samples well represented.

Step 2. **Label inference.** The label inference is performed so that the label information can be propagated from the labeled samples to the unlabeled ones by incorporating the structure information from the constructed graph in the previous step.

Compared with other SSL methods, which are not involved with any graph structure, GSSL has some advantages that are worth noticing. In the following, we list several advantages of graph-based SSL methods.

- **Universality**. Many common data sets of current trends are represented by graphs like the World Wide Web (WWW), citation networks, and social networks.
- **Convexity**. Since an undirected graph is usually involved in the graph construction step, the symmetric feature of the undirected graph makes it easier to formulate the learning problem into a convex optimization problem, which can be solved with various exciting techniques [1].
- **Scalability**. Many of the GSSL methods are meticulously designed so that the time complexity is linear to the total number of samples. As a result, they are often easily parallelized to handle large scale datasets with ease.

Related work. Several SSL survey papers [2][3] often fail to cover enough methods of GSSL, neglecting its significant role in SSL. Zhu *et al.* [2] conduct a comprehensive review of classic methods involved in SSL, and GSSL is not explored in detail. Similar earlier work like [3] by Pise *et al.* also tries to present a whole picture of SSL methods without covering enough work in GSSL. Recent literature review work, [4] [5] and [6] all follow the footsteps of work [2] and [3] by adding more recent research output. However, they do not cover the recent development in GSSL methods. Instead, our work solely focuses on GSSL and combines both earlier studies with recent advances.

Z. Song and I. King are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China(Email: zxsong@cse.cuhk.edu.hk, king@cse.cuhk.edu.hk).

X. Yang is with the SMILE Lab, School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China(E-mail: xlyang@std.uestc.edu.cn).

Z. Xu is with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China, and also with Peng Cheng Lab, Shenzhen, China (Email: xuzenglin@hit.edu.cn).

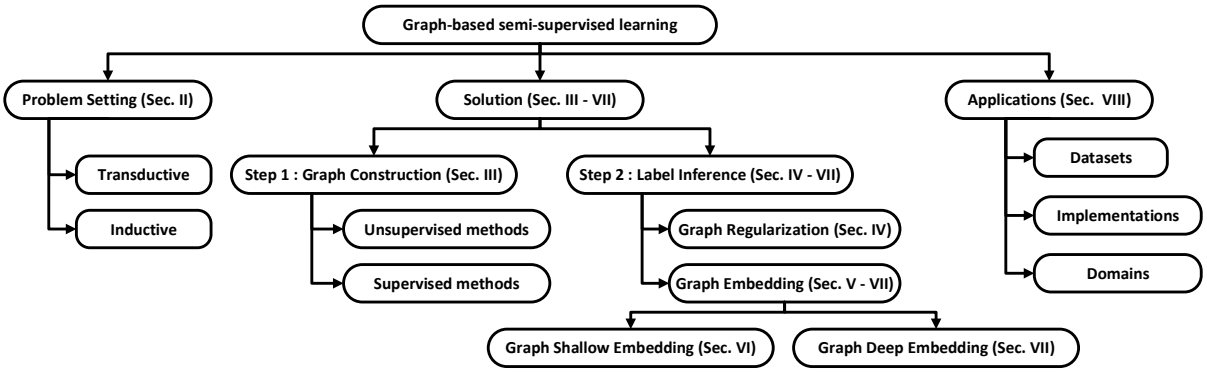

Figure 1: Taxonomy for graph-based semi-supervised learning

The most relevant work to ours is [7] by Chong *et al.*, and it is considered as the most up-to-date survey paper on GSSL. However, there are several noticeable drawbacks of this work that are worth mentioning here. First, [7] reviews work from the perspective of transductive, inductive, and scalability learning. This taxonomy fails to show the context of development and thus does not reveal the relationship of different methods or models. As a result, we provide a novel taxonomy from the perspective of the two main steps in GSSL: graph construction and label inference. Secondly, some of the reviewed methods in [7] are not graph-based models, but rather are some semi-supervised convolutional neural network (CNN) models as shown in Section 3.4 in the original paper[7]. Most importantly, [7] fails to develop a framework to generalize the methods or models reviewed. However, this paper fills all these gaps with several noticeable contributions.

Contributions. To sum up, this paper presents an extensive and systematic review of GSSL with the following contributions.

1) **Comprehensive review.** We provide the most comprehensive and the most up-to-date overview of GSSL methods. For every approach reviewed in this paper, we present the detailed descriptions with key equations, clarify the context of development beneath the algorithms, make the necessary comparison, and summarise the corresponding strengths or limitations.
2) **New taxonomy.** We propose a new taxonomy for graph-based semi-supervised learning with a more generalized framework, as shown in Figure 1. We divide the GSSL process into two steps. The first one is to construct a similarity graph and the second step is to do label inference based on this graph. The latter step is much more challenging and is also the main focus of this paper. Label inference methods are then categorized into two main groups: graph regularization methods and graph embedding methods. For the former group, a generalized framework of regularizers from the perspective of the loss function is presented. For the latter group, we provide a new unified representation for graph embedding methods in SSL with the help of the encoder-decoder framework.
3) **Abundant resources.** We collect abundant resources related to GSSL and build a useful, relevant code base, including the open-source codes for all the reviewed methods or models, some popular benchmark data sets, and pointers to representative practical applications in different areas. This survey can be regarded as a hands-on guide for researchers interested in understanding existing GSSL approaches, using the codes for experiments, and even developing new ideas for GSSL.
4) **Future directions.** We propose some open problems and point out some directions for future research in terms of dynamicity, scalability, noise-resilience, and attack-robustness.

Organization of the paper. The rest of this survey is organized as follows. In Section II, we introduce the background knowledge related to GSSL. Then some necessary notations are listed, and the relevant terms are properly defined. In Section III, we provide a detailed review of graph construction, the first step of GSSL. From Section IV to VII, the label inference, the second step of GSSL, is covered, which is the main focus of this paper. Furthermore, a new taxonomy is provided, as shown in Figure 1. Graph regularization methods are reviewed in Section IV while graph embedding methods are reviewed from Section V to Section VII. Section V discusses the generalized encoder-decoder framework for graph embedding. To provide a more detailed overview of it, we further split it into shallow embedding and deep embedding and review them in Section VI and Section VII respectively. Moreover, in Section IX, four open problems are briefly reviewed as future research directions. Finally, applications of GSSL are extensively explored in the Appendix, along with a list of common datasets and a code base for some popular models.

II. Background and Definition

As is mentioned earlier, a majority of GSSL algorithms requires solving the following two sub-problems:

- Constructing a graph over the input data (if one is not already available).
- Inferring the labels on the unlabeled samples in the input or estimating the model parameters.

GSSL methods run on a specifically designed graph in which training samples are represented as nodes, and each node pair is linked by weight to denote the underlying similarity. Some of the nodes are labeled, while others are not.

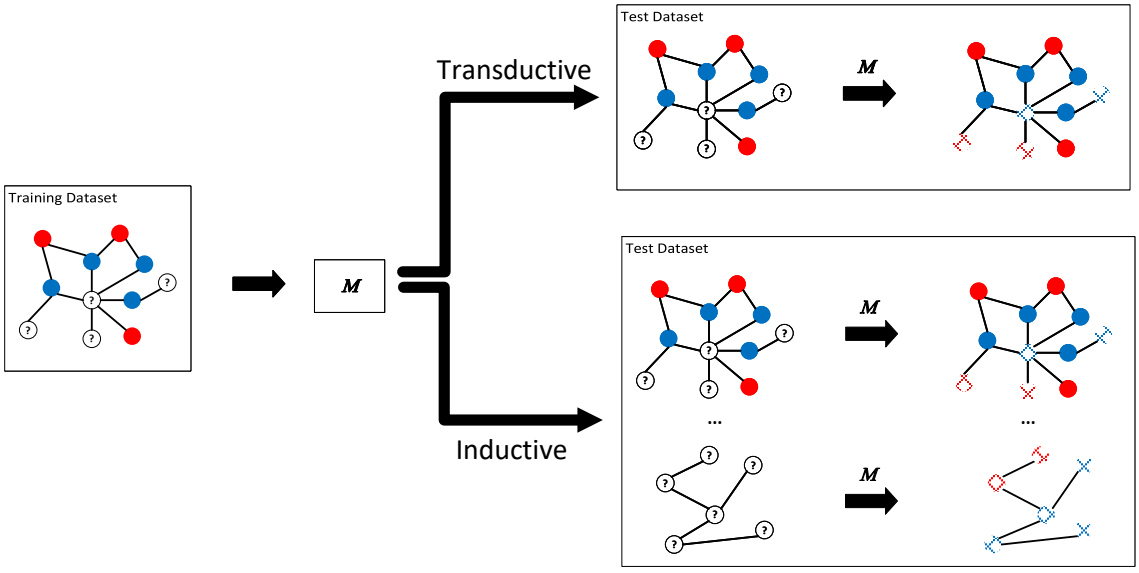

Figure 2: Comparison between transductive and inductive setting in GSSL. For transductive setting, only the labels of unlabeled nodes in the training dataset need to be inferred while for inductive setting, the trained model M can predict the label of any unseen node.

As a result, a graph has to be built to make these problems amenable to the following GSSL approaches.

However, it is worth mentioning that most of the graph-based algorithms are designed for the label inference step. As a result, in this paper, we mainly focus on the label inference techniques used in GSSL, and we only discuss graph construction in Section III.

Once the graph is constructed, the next step in solving an SSL problem using graph-based methods is to inject labeled data on a subset of the nodes in the graph, followed by inferring the labels for the unlabeled nodes. While a majority of the graph-based inference approaches are transductive, there are some inductive GSSL approaches as well.

Following the framework for SSL, the loss function of GSSL approaches can also be generalized within that of SSL, which contains three parts as shown in Eq. (1)

$$\mathcal{L}(f) = \underbrace{\mathcal{L}_s(f, \mathcal{D}_l)}_{\text{supervised loss}} + \lambda \underbrace{\mathcal{L}_u(f, \mathcal{D}_u)}_{\text{unsupervised loss}} + \mu \underbrace{\mathcal{L}_r(f, \mathcal{D})}_{\text{regularization loss}}, \quad (1)$$

where $\mathcal{L}_s(f, \mathcal{D}_l)$ is the supervised loss on the labeled data and $\mathcal{L}_u(f, \mathcal{D}_u)$ is the unsupervised loss on the unlabeled data and $\mathcal{L}_r(f, \mathcal{D})$ is the regularization loss. Additionally, λ and μ are hyperparameters to balance these terms. However, for GSSL, unsupervised loss is often absorbed into the regularization loss since no label information is used in the regularization loss term. Therefore, the loss function for GSSL can be generalized as shown in Eq. (2)

$$\mathcal{L}(f) = \mathcal{L}_s(f, \mathcal{D}_l) + \mu \mathcal{L}_r(f, \mathcal{D}). \quad (2)$$

In this paper, more attention will be paid to how to do label inference when the similarity graph has already been constructed from the given datasets under the setting of semi-supervised learning. Two main groups of GSSL are reviewed following Eq. (2): graph regularization and graph embedding. More details will be provided in Section IV to Section VII.

A. Related concepts

1) Supervised learning and unsupervised learning: Supervised learning and unsupervised learning can be viewed as two extremes of SSL because all the training samples are well labeled in supervised learning settings, while unsupervised learning can only have access to unlabeled data. Semi-supervised learning aims to introduce cheap unlabeled samples to enhance the model's performance with only a few costly labeled samples. Therefore, the problem setting of SSL is a perfect match for many real-world applications.

2) Other semi-supervised learning methods: Throughout the development of SSL, a great number of successful algorithms or models have emerged in roughly three phases. The first phase is the early stage of SSL before 2000, where classic machine learning algorithms are investigated and improved with unlabeled data. Typical examples are S3VM and Co-training. The second phase is the mature stage of SSL between 2000 and 2015, in which many methods flourished, such as mixture model, pseudo label, self-training, manifold learning, and GSSL. The third phase is after 2015, with the advance of deep learning and especially Graph Neural Networks (GNN). Since GSSL methods witness all these three stages, reviewing its development and recent progress is necessary.

3) Transductive and inductive settings: Like other SSL methods, GSSL algorithms can be divided into two categories based on whether to predict data samples' labels out of training data.

Definition II.1 (Transductive setting). Given a training set consisting of labeled and unlabeled data $\mathcal{D} = \{\{\mathbf{x}_i, y_i\}_{i=1}^{n_l}, \{\mathbf{x}_i\}_{i=1}^{n_u}\}$, the goal of a transductive algorithm

Table I: Notations used in the paper

Notations	Descriptions
G	A Graph
V	The set of nodes (vertices) in a graph
E	The set of edges in a graph
i,v	Node i, Node v
(i, j)	The edge linked between node i, j
W	The weight matrix of a graph
W_{ij}	The weight associated with edge (i, j)
A	The adjacency matrix of a graph
A_{ij}	i^{th} row j^{th} column in the adjacency matrix A
D	The degree matrix of a graph
D_{ii}	The degree of node i
X	The attribute matrix of a graph
x_i	The attribute vector for node i
$\mathcal{N}(i)$	The neighborhood of a node i
L	Unnormalized graph Laplacian matrix
$\tilde{L}$	Normalized graph Laplacian matrix
$\mathcal{P}_i^1$	First-order proximity of node i
$\mathcal{P}_{i,j}^2$	Second-order proximity between node i, j
n_l	The number of labeled samples
n_u	The number of unlabeled samples
$\mathcal{D}_l = \{\mathbf{x}_i, y_i\}_{i=1}^{n_l}$	Labeled samples
$\mathcal{D}_u = \{\mathbf{x}_i\}_{i=1}^{n_u}$	Unlabeled samples
$\mathbf{S}$	Similarity matrix of a graph
$\mathbf{S}[u, v]$	Similarity measurement between node u,v
$\mathbf{Z}$	Embedding matrix
$\mathbf{z}_i$	Embedding for node i
$\mathbf{h}_v^{(k)}$	Hidden embedding for node v in k^{th}layer
$\mathbf{m}_{\mathcal{N}(v)}^{(k)}$	Message aggregated from node v's neighborhood in k^{th}layer

is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ so that f is only able to predict the labels for the unlabeled data $\{\mathbf{x}_i\}_{i=1}^{n_u}$.

Definition II.2 (Inductive setting). Given a training set consisting of labeled and unlabeled data $\mathcal{D} = \{\{\mathbf{x}_i, y_i\}_{i=1}^{n_l}, \{\mathbf{x}_i\}_{i=1}^{n_u}\}$, the goal of an inductive algorithm is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ so that f is able to predict the output y of any input $x \in \mathcal{X}$.

While most of the GSSL approaches are transductive, there are a few inductive GSSL approaches. In most scenarios, transductive SSL methods outperform inductive ones in terms of prediction accuracy while they often suffer from high training costs compared to inductive ones, especially in the context of large scale incremental learning. Figure 2 illustrates the difference between transductive and inductive setting in GSSL.

B. Notations and Definitions

In this section, as a matter of convenience, we first define some useful and common terms used in GSSL, along with relevant notations. Unless otherwise specified, the notations used in this survey paper are illustrated as Table I. After the list of the notations, the minimal set of definitions required to understand this paper is defined.

Definition II.3 (Graph). A graph is an ordered pair $G = (V, E)$ where $V = \{1, \ldots, |V|\}$ is the set of nodes (or vertices) and $E \subseteq \{V \times V\}$ is the set of edges.

GSSL algorithms start by representing the data as a graph. We assume that the node $i \in V$ represents the input sample $\mathbf{x}_i$. We will be using both i and $\mathbf{x}_i$ to refer to the i^{th} node in the graph.

Definition II.4 (Directed and Undirected Graphs). A graph whose edges have no starting or ending nodes is called an undirected graph. In the case of a directed graph, the edges have a direction associated with them.

Definition II.5 (Weighted Graph). A graph G is weighted if there is a number or weight associated with every edge in the graph. Given an edge (i, j), where $i, j \in V$ and W_{ij} is used to denote the weight associated with the edge (i, j) and thus forms the whole weight matrix $W \in \mathcal{R}^{n \times n}$. In most cases, we assume $W_{ij} \geq 0$ and W_{ij} can be 0 if and only if there is no edge between the node pair (i, j).

Definition II.6 (Degree of a Node). The degree D_{ii} of the node i is given by $D_{ii} = \sum_j W_{ij}$. Moreover, in the case of an unweighted graph, the node's degree is equal to its number of neighbors.

Definition II.7 (Neighborhood of a Node). The neighborhood of a node v in a graph G is denoted as $\mathcal{N}(v)$ to indicate the subgraph of G induced by all nodes adjacent to v.

Definition II.8 (Adjacency Matrix). Adjacency matrix is a matrix with a 1 or 0 in each position (i, j) based on whether node i and node j are adjacent or not. If the given graph is undirected, its corresponding adjacency matrix is a symmetric matrix.

Definition II.9 (Graph Laplacian Matrix). The unnormalized graph Laplacian matrix is given by $L = D - W$. Here the $D \in \mathcal{R}^{n \times n}$ is a diagonal matrix such that $D_{i,i}$ is the degree of the node i and otherwise $D_{ij} = 0 \ \forall i \neq j$. It is easy to prove that L is a positive semi-definite matrix.

The normalized graph Laplacian matrix is given by $\tilde{L} = D^{-1/2} L D^{1/2}$ where L is the unnormalized graph Laplacian matrix.

III. Graph Construction

To perform any GSSL methods, a graph must be constructed first, where nodes represent data samples, some of which are labeled while others are not, and edges are associated with a certain weight to reflect each node pair's similarity. In some domains, such as citation networks, there is already an implicit underlying graph. Graph-based methods are thus a natural fit for SSL problems in these domains. For most of the other machine learning tasks, however, it is believed that the data instances are not conveniently represented as a graph structure, and as a result, a graph has to be built to make these problems appropriate for GSSL approaches. The graph construction techniques are involved in the first step mentioned before.

The goal of graph construction is to discover a graph $G = (V, E, W)$ where V is the set of nodes, $E \subseteq V \times V$ are the edges, and $W \in \mathcal{R}^{n \times n}$ are the associated weights on the edges. Each node in the graph represents an input sample, and thus the number of nodes in the graph is $|V| = n$. As the nodes are fixed (assuming that D is fixed, which is often

the case), the task of graph construction involves estimating E and W. The following three assumptions often hold.

Assumption 1. The graph is undirected, so W is symmetric. And all edge weights are non-negative, $W_{ij} \geq 0$, $\forall i \neq j$.

Assumption 2. $W_{ij} = 0$ implies the absence of an edge between nodes i and j.

Assumption 3. There are no self-loops, $W_{ii} = 0$, $\forall 1 \leq i \leq n$.

These three assumptions simplify the problem by adding these constraints and lay the foundations for the following unsupervised and supervised methods.

A. Unsupervised methods

Unsupervised graph construction techniques ignore all the given label information of the training data during the construction process. Among all the unsupervised methods for graph construction, the K-nearest neighbor (KNN) graph and b-Matching methods, along with their extensions, are the most popular ones.

1) KNN-based approaches: For KNN-based graph construction approaches [8], every node is associated based on a pre-configured distance metric with its k nearest neighbors in the resulting graph. Moreover, KNN-based methods link the k nearest neighbors greedily to generate graphs whose nodes' degree is larger than k, which leads to irregular graphs. Note that a graph is said to be regular if every node has the same degree.

KNN-based method needs a proximity function $\texttt{sim}(\mathbf{x}_i, \mathbf{x}_j)$ or distance metric that can quantify the resemblance or disparity between every node pair in the training data. The weight value associated with the edge is given by Eq. (3),

$$W_{ij} = \begin{cases} \texttt{sim}(\mathbf{x}_i, \mathbf{x}_j) & i \in \mathcal{N}(j) \\ 0 & otherwise \end{cases}. \tag{3}$$

In ε-neighborhood-based graph construction method [8], if the distance between a node pair is smaller than ε, where $\varepsilon \geq 0$ is a predefined constant, a connected edge is formed between them. KNN methods enjoy certain favorable properties when compared with ε-neighborhood-based graphs. Specifically, in ε-neighborhood-based graphs, a misleading choice of the parameter ε could lead to generating disconnected graphs [9]. However, KNN-based graphs outperform ε-neighborhood-based ones with better scalability.

In Oziki *et al.* [10], it is contended that a hub or a center situated in the sample space can result in a corresponding hub in the classic KNN graphs. This may downgrade the prediction performance on several classification tasks. In order to handle this issue, [10] proposes a new way of constructing a graph by using mutual KNN in combination with a maximum span tree (M-KNN). In parallel with this work, Vega *et al.* [11] also introduce the sequential KNN (S-KNN) to produce graphs under the new relaxed condition in which the resulting graph contains no hubs but is not necessarily regular.

2) b-Matching: As discussed above, KNN graphs, contrary to their name, often lead to graphs where different nodes have different degrees. Jebara *et al.* [9] propose b-Matching, which guarantees that every node in the resulting graph has exactly b neighbors. Using b-Matching for graph construction involves two steps: (a) graph sparsification and (b) edge re-weighting.

In graph sparsification, there exists an issue where edges are removed in a way of estimating a matrix $P \in \{0,1\}^{n \times n}$. For the entry in P, $P_{ij} = 1$ signifies an existing edge between a node pair in the generated graph, while $P_{ij} = 0$ suggests a lack of an edge. b-Matching provides a solution by formulating an optimization problem with the objective,

$$\begin{aligned} &\min_{P \in \{0,1\}^{n \times n}} \sum_{i,j} P_{ij} \Delta_{ij} \\ &s.t. \sum_j P_{ij} = b, P_{ii} = 0, P_{ij} = P_{ji}, \forall 1 \leq i, j \leq n. \end{aligned} \tag{4}$$

Here, $\Delta \in \mathcal{R}_+^{n \times n}$ is a symmetric distance matrix.

When a selection of edges is made in matrix P from the previous step, the next aim is to determine the chosen edges' associated weights to produce the estimated weight matrix W. There are three popular ways to determine the weight matrix W.

- **Binary Kernel.** The easiest way to estimate W is to set $W = P$. Thus $W_{ij} = P_{ij}$ and each entity in W is also either 0 or 1.
- **Gaussian Kernel.** Here, W can be a little bit complex compared to the previous one. That is $W_{ij} = P_{ij} \exp\left(-\frac{d(x_i, x_j)}{2\sigma^2}\right)$.
- **Locally Linear Reconstruction (LLR).** LLR is derived from the Locally Linear Embedding (LLE) technique by Roweis *et al.* [12]. The goal is to reconstruct $\mathbf{X}_i$ from its neighborhood. It can be formulated to solve the following optimization problem,

$$\begin{aligned} &\min_W \sum_i \left\| x_i - \sum_j P_{ij} W_{ij} x_j \right\|^2 \\ &s.t. \sum_j W_{ij} = 1, W_{ij} \geq 0, i = 1, \ldots, n. \end{aligned} \tag{5}$$

In summary, the b-matching method restricts the constructed similarity graph to be regular so that the given label can be propagated in a more balanced way during the following label inference step.

B. Supervised methods

The existing prevalent strategies of graph construction are unsupervised, i.e., they fail to use any given label information during the construction phase. However, labeled samples can be used as a kind of prior knowledge that can be used to refine the generated graph for the downstream learning tasks. Dhillon *et al.* [13] study the possibility of employing labeled points so as to measure the similarities between node pairs. Rohban *et al.* [14] suggest another supervised method of graph construction, which demonstrates that the optimal solution for a neighborhood graph can be regarded as a subgraph of a KNN graph as long as the manifold sampling rate is large enough.

Driven by previous studies [10], a new method, graph-based on informativeness of labeled instances (GBILI) [15], also

utilizing the label information, is introduced. GBILI not only results in a decent accuracy on classification tasks but also stands out with a quadratic time complexity [16]. Moreover, built on GBILI [15], Lilian *et al.* [17] have upgraded the method for producing more robust graphs by solving an optimization problem with the specific algorithm called the Robust Graph that Considers Labeled Instances (RGCLI). More recently, a new SSL learning method referred to as a low-rank semi-supervised representation is proposed [18] which incorporates labeled data into the low-rank representation (LRR). A follow-up work is by Taherkhani *et al.* [19]. By taking additional supervised information, the generated similarity graph can facilitate the following label inference process to a great extent.

IV. Graph Regularization

All the classic GSSL methods can actually be simplified as searching for a function f on the graph. f has to satisfy two criteria simultaneously: 1) it must be as close to the given labels as possible, and 2) it must be smooth on the entire constructed graph.

These two conditions can be further expressed in a general regularization framework in which loss function can be decomposed into two main parts. The first term is a supervised loss constraint to the first criterion, and the second term is a graph regularization loss constraint to the second criterion. Formally, we have,

$$\mathcal{L}(f) = \sum_{(x_i,y_i)\in\mathcal{D}_l} \underbrace{\mathcal{L}_s(f(x_i), y_i)}_{\text{supervised loss}} + \mu \sum_{x_i\in\mathcal{D}_l+\mathcal{D}_u} \underbrace{\mathcal{L}_r(f(x_i))}_{\text{regularization loss}}, \tag{6}$$

where f is the prediction function and μ is a trade-off hyper-parameter.

In the following sections, we will see that all the graph regularization methods reviewed here are similar. They only differ in the particular choice of the loss function with various regularizers. Table II summarizes all the reviewed graph regularization methods in Section IV from the perspective of decomposing the regularizer. This generalized framework of graph regularization has been carefully examined by Zhou *et al.* [20], and its theoretical analysis from different perspectives has also been provided by [21] [22] [23].

A. Label propagation

Label Propagation (LP) [31] is the most popular method for label inference on GSSL. Label Propagation can be formulated as a problem, in which some of the nodes' labels, also referred to as seeds, propagate to unlabeled nodes based on the similarity of each node pair, which is represented by the constructed graph discussed in Section III. Meanwhile, during the propagation process, given labels need to be fixed. In this way, labeled nodes serve as guides that lead label information flow through the edges within the graph so that unlabeled nodes can also be tagged with predicted labels.

The basic version of label propagation algorithm is as follows:

Step 1. All nodes propagate labels for one step $Y \leftarrow TY$.

Step 2. Row-normalize Y to maintain the class probability interpretation.

Step 3. Clamp the labeled data. Repeat from step 2 until Y converges.

1) Gaussian random fields: Gaussian Random Fields (GRF) [24] is a typical example of the early work in GSSL by using label propagation algorithms. The strategy is to estimate some prediction function f based on the graph G with some constraints to ensure certain necessary properties and afterward attach labels to the unlabeled nodes according to f. In fact, the above-mentioned constraint is to take $f(x_i) = f_l(x_i) \equiv y_i$ on all the labeled nodes. Intuitively, the clustering unlabeled points with strongly connected edges should share common labels. This is why the quadratic energy function is designed as shown in Eq. (7),

$$E(f) = \mathcal{L}_r = \frac{1}{2}\sum_{i,j} W_{ij}\left(f(x_i) - f(x_j)\right)^2. \tag{7}$$

It is noteworthy that the minimum value of energy function $f = \arg\min_{f|_{\mathcal{D}_l}=f_l} E(f)$ is harmonic; namely, it satisfies the constraint $Lf = 0$ on the unlabeled nodes and is equal to f_l on the labeled nodes $\mathcal{D}_l$, where L is the graph Laplacian matrix.

The property of harmonic function indicates that the value of f at every unlabeled node is the mean value of f at its neighboring nodes: $f(x_j) = \frac{1}{d_j}\sum_{i\sim j} W_{ij} f(x_i)$, for $j = l+1, \ldots, l+u$. This constraint is actually compatible with the previous smoothness requirement of f with respect to the graph. It can also be interpreted in an iterative manner as shown in Eq. (8)

$$f^{(t+1)} \leftarrow P \cdot f^{(t)}, \tag{8}$$

where $P = D^{-1}W$. Furthermore, a closed form solution of Eq. (8) can be deduced if weight matrix W is split into four blocks $W = \begin{bmatrix} W_{ll} & W_{lu} \\ W_{ul} & W_{uu} \end{bmatrix}$. Then,

$$f_u = \left(D_{uu} - W_{uu}\right)^{-1} W_{ul} f_l = \left(I - P_{uu}\right)^{-1} P_{ul} f_l. \tag{9}$$

2) Local and global consistency: Zhou *et al.* [25] extend the work [24] to multiclass setting and proposes Local and Global Consistency (LGC) to handle a more general semi-supervised problem. The iterative formula is shown in Eq. (10)

$$Y^{(t)} = \alpha \mathcal{S} Y^{(t-1)} + (1-\alpha) Y^{(0)}, \tag{10}$$

where $\mathcal{S} = D^{-1/2} A D^{-1/2}$, and α is a hyper-parameter. We can also easily derive the closed-form solution for Eq. (10) as shown in Eq. (11)

$$\hat{Y} = \alpha \mathcal{S} \hat{Y} + (1-\alpha) Y^{(0)}. \tag{11}$$

From a perspective of optimization problem, LGC [25] actually tries to minimize the following objective function Eq. (12) associated with prediction function f.

$$\mathcal{L}(f) = \frac{1}{2}\left(\sum_{i,j} W_{ij}\left(\frac{1}{\sqrt{D_{ii}}} f(x_i) - \frac{1}{\sqrt{D_{jj}}} f(x_j)\right)^2\right) + \mu \sum_{i=1}^{n_l} \left(f(x_i) - y_i\right)^2. \tag{12}$$

Table II: Summary on Graph Regularization Methods

Method	Supervised loss $f_s(f, \mathcal{D}_l)$	Graph regularization loss $f_r(\mathcal{D})$
GRF [24]	$\sum_{i=1}^{n_l} (f(x_i) - y_i)^2$	$\sum_{i,j} W_{ij} (f(x_i) - f(x_j))^2$
LRC [25]	$\sum_{i=1}^{n_l} (f(x_i) - y_i)^2$	$\left(\sum_{i,j} W_{ij} \left(\frac{1}{\sqrt{D_{ii}}} f(x_i) - \frac{1}{\sqrt{D_{jj}}} f(x_j)\right)^2\right)$
p-Laplacian [26]	$\sum_{i=1}^{n_l} (f(x_i) - y_i)^2$	$\sum_{i,j} W_{ij} \left\lvert \frac{1}{\sqrt{D_{ii}}} f(x_i) - \frac{1}{\sqrt{D_{jj}}} f(x_j)\right\rvert^p$
Directed regularization [27]	$\sum_{i=1}^{n_l} (f(x_i) - y_i)^2$	$\left(\sum_{i,j} \pi(i) p(i,j) \left(\frac{1}{\sqrt{D_{ii}}} f(x_i) - \frac{1}{\sqrt{D_{jj}}} f(x_j)\right)^2\right)$
Manifold regularization [28]	$\sum_{i=1}^{n_l} (f(x_i) - y_i)^2$	$\gamma_A \lVert f \rVert_K^2 + \gamma_I \frac{1}{(n_l+n_u)^2} \hat{y}^T L \hat{y}$
LPDGL [29]	$\sum_{i=1}^{n_l} (f(x_i) - y_i)^2$	$\left(\sum_{i,j} W_{ij} (f(x_i) - f(x_j))^2\right) + \left(\sum_{i=1}^{n} (1 - \frac{D_{ii}}{\sum_{j=1}^{n} D_{jj}}) (f(x_i))^2\right)$
Poisson learning [30]	$\sum_{i=1}^{n_l} (f(x_i) - y_i)^2$	$\left(\sum_{i,j} W_{ij} \left(f(x_i) - \sum_{j \in \mathcal{N}(i)} f(x_j)\right)^2\right)$

Compared to the GRF objective above, LGC has two important differences: (a) the inferred labels for the "labeled" nodes are no longer required to be exactly equal to the seed values, and this helps with cases where there may be noise in the seed labels, and (b) the label for each node is penalized by the degree of that node $\frac{1}{\sqrt{D_{ii}}}$, ensuring that in the case of irregular graphs, the influence of high degree nodes is regularized.

There exist quite a few variants of LGC method, a representative one is p-Laplacian regularization [26]. The first term in Eq. (12) can be substituted by a more general one as $\sum_{i,j} W_{ij} \left\lvert \frac{1}{\sqrt{D_{ii}}} f(x_i) - \frac{1}{\sqrt{D_{jj}}} f(x_j)\right\rvert^p$, where p is a positive integer. Slepcev *et al.* [26] provide a comprehensive analysis of its theoretical grounds. In addition, many applications based on LGC are proven to be successful in various domains. For example, Iscen *et al.* [32] utilize LGC method to facilitate the training process of deep neural networks (DNNs) by generating pseudo-label for the unlabeled data.

B. Directed regularization

In the previous label propagation methods, only undirected graphs are applicable. A new regularization framework for directed graphs, such as citation networks, is provided to solve this issue [27]. To fully take the directionality of the edges into consideration, the idea of naive random walk is incorporated into this regularization framework. π is used to denote a unique probability distribution satisfying the following equations,

$$\pi(i) = \sum_{j \to i} \pi(j) p(j,i), \forall i \in V, \tag{13}$$

where

$$p(i,j) = \frac{W_{ij}}{d^+(i)} = \frac{W_{ij}}{\sum_{j \leftarrow i} W_{ij}}. \tag{14}$$

In Eq. (13) and Eq. (14), $j \to i$ denotes the set of vertices adjacent to the vertex i while $j \leftarrow i$ denotes the set of vertices adjacent from the vertex i. Thus, we can define a loss function that sums the weighted variation of each edge in the directed graph as shown in Eq. (15).

$$\mathcal{L}(f) = \frac{1}{2} \left(\sum_{i,j} \pi(i) p(i,j) \left(\frac{1}{\sqrt{D_{ii}}} f(x_i) - \frac{1}{\sqrt{D_{jj}}} f(x_j)\right)^2\right) + \mu \sum_{i=1}^{n_l} (f(i) - y_i)^2. \tag{15}$$

It is also worth noting that Eq. (12) for undirected graphs can be regarded as a specific case of Eq. (15) for directed graphs. The stationary distribution of the random walk in an undirected graph is $\pi(j) = D_{jj} / \sum_{i \in V} D_{ii}$. By substituting this expression into Eq. (15), we can easily derive Eq. (12), which is the exactly the regularizer of LGC by Zhou *et al.* [25].

C. Manifold regularization

The manifold regularization [28] [33] is actually a general framework that allows for developing a great number of algorithms ranging from supervised learning to unsupervised learning. However, it is viewed as a natural fit for GSSL since it combines the spectral graph theory with manifold learning to search for a low-dimensional representation with smoothness constraint in the original commonly high-dimensional space.

The manifold regularization framework fully utilizes the geometry property of the unknown probability distribution, which the data samples obey. Therefore, it introduces another term as a regularizer to control the complexity of the prediction function in the intrinsic space, measured by the geometry of the probability distribution.

Formally, for a Mercer kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, we denote the associated Reproducing Kernel Hilbert Space (RKHS) of the prediction function f. Then the loss function can be formulated in Eq. (16) as

$$\mathcal{L}(f) = \frac{1}{n_l} \sum_{i=1}^{n_l} (f(x_i) - y_i)^2 + \gamma_A \lVert f \rVert_K^2 + \gamma_I \lVert f \rVert_I^2, \tag{16}$$

where γ_A balances the complexity of the prediction function in the ambient space and γ_I is the weighting parameter for the smoothness constraint term $\lVert f \rVert_I^2$ induced by both labeled and unlabeled samples.

It is noted that the added regularization term $\|f\|_I^2$ usually takes the following form,

$$\|f\|_I^2 = \frac{1}{(n_l+n_u)^2}\hat{y}^T L \hat{y}, \tag{17}$$

where $\hat{y} = [f(x_1), f(x_2), \ldots, f(x_{n_l+n_u})]^T$ and L is the Laplacian matrix of the graph.

According to the Representer Theorem [34], it is well-known that Eq. (16) has a closed-form solution when $\|f\|_I^2$ takes the form as shown in Eq. (17). However, it suffers from the high computational cost [35] which makes the algorithm unscalable when faced with large graphs. Popular solutions to alleviate this problem would be to accelerate either the construction of the Laplacian graph [36] [37] or the kernel matrix operation [38] [39].

D. LPDGL

The above three methods [24] [25] [27] all prove to be ineffective for handling ambiguous examples [40]. Gong *et al.* [29] introduce deformed graph Laplacian (DGL) and provides the corresponding label prediction algorithm via DGL (LPDGL) for SSL. A new smoothness term that considers local information is added to the regularizer. The whole regularizer becomes Eq. (18) as

$$\begin{aligned}\mathcal{L}(f) = &\frac{1}{2}\left(\alpha\sum_{i,j} W_{ij}\left(f(x_i)-f(x_j)\right)^2\right)\\ &+\frac{1}{2}\left(\beta\sum_{i=1}^{n}(1-\frac{D_{ii}}{\sum_{j=1}^{n}D_{jj}})\left(f(x_i)\right)^2\right)\\ &+\mu\sum_{i=1}^{n_l}\left(f(x_i)-y_i\right)^2,\end{aligned} \tag{18}$$

where both α and β are trade-off parameters. It has been shown by theoretical analysis that LPDGL achieves a globally optimal prediction function. Additionally, the performance is robust to the hyper-parameters setting so this model is not difficult to fine-tune.

E. Poisson learning

The most recent work under the regularization framework is called poisson learning by [30], which is motivated by the need to address the degeneracy of previous graph regularization methods when the label rate is meager. The new proposed approach replaces the given label values with the assignment of sources and sinks like flow in the graph. Thus, a resulting Poisson equation based on the graph can be nicely solved. The loss function of poisson learning is shown in Eq. (19).

$$\begin{aligned}\mathcal{L}(f) = &\frac{1}{2}\left(\sum_{i,j} W_{ij}\left(f(x_i)-\sum_{j\in\mathcal{N}(i)} f(x_j)\right)^2\right)\\ &+\mu\sum_{i=1}^{n_l}\left(f(x_i)-y_i\right)^2,\end{aligned} \tag{19}$$

V. Graph Embedding

Generally speaking, there are two types of graph embedding at two levels commonly seen in the literature. The first one is at the entire graph level while the second one is at the single node level [41]. Both of them aim to represent the target object in a low-dimensional vector space. For GSSL, we focus on node embeddings since such representations can be easily used for SSL tasks. The main objective of node embedding is to encode the nodes as vectors with lower dimensions, which in the meantime can reflect their positions and the structure of their local neighborhood.

Formally, we have the following definition for node embedding on graphs. Given a graph $G = (V, E)$, a node embedding on it is a mapping $f_{\mathbf{z}}\colon v \to \mathbf{z}_v \in \mathbb{R}^d, \forall v \in V$ such that $d \ll |V|$ and the function $f_{\mathbf{z}}$ preserves some proximity measure defined on graph G. The generalized form of the loss function for graph embedding methods is shown in Eq. (20) as

$$\begin{aligned}\mathcal{L}(f) = &\sum_{(x_i,y_i)\in\mathcal{D}_l}\mathcal{L}_s(f(f_{\mathbf{z}}(x_i)), y_i)\\ &+\mu\sum_{x_i\in\mathcal{D}_l+\mathcal{D}_u}\mathcal{L}_r(f(f_{\mathbf{z}}(x_i))),\end{aligned} \tag{20}$$

where $f_{\mathbf{z}}$ is the embedding function. It is obvious that Eq. (20) is almost the same as Eq. (6) for graph regularization except that for graph embedding methods, classifiers are trained based on the nodes' embedding results rather than nodes' attributes directly.

A. Generalization: Perspective of encoder-decoder

Following the generalization methods on graph representation learning by Hamilton *et al.* [41], all the node embedding methods mentioned in this section can be generalized under an encoder-decoder framework. From this perspective, the node embedding problem in graphs can be viewed as involving two key steps. First, an encoder model tries to map every node into a low-dimensional vector. Second, a decoder model is constructed to take the low-dimensional node embeddings as input and use them to reconstruct the information related to each node's neighborhood in the original graph, like an adjacency matrix.

1) Encoder: Formally, the encoder can be viewed as a function that maps nodes $v \in V$ to vector embeddings $\mathbf{z}_v \in \mathbb{R}^d$. The resulting embeddings are more discriminative in the latent space with more dimensions. Furthermore, they can be transformed back to the original feature vector more easily in the following decoder module. From a mathematical view, we have $\mathrm{Enc}\colon V \to \mathbb{R}^d$.

2) Decoder: The decoder module's main goal is to reconstruct certain graph statistics from the node embeddings generated by the encoder in the previous step. For example, given a node embedding $\mathbf{z}_u$ of a node u, the decoder might attempt to predict u's set of neighbors $\mathcal{N}(u)$ or its row $\mathbf{A}[u]$ in the graph adjacency matrix.

Decoders are often defined in a pairwise form, which can be illustrated as predicting each pair of nodes' similarity. Formally, we have, $\mathrm{Dec}\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$.

3) Reconstruction: The reconstruction process of a pair of node embeddings $\mathbf{z}_u, \mathbf{z}_v$ involves applying the pairwise decoder to them. The overall goal is to solve an optimization problem that minimizes the reconstruction loss so that the similarity measures produced by the decoder are as close to the ones defined in the original graph as possible. In a more formal way, we have

$$\mathrm{Dec}(\mathrm{Enc}(u), \mathrm{Enc}(v)) = \mathrm{Dec}\left(\mathbf{z}_u, \mathbf{z}_v\right) \approx \mathbf{S}[u, v]. \tag{21}$$

Here, we assume that $\mathbf{S}[u, v]$ is a certain kind of similarity measure between a pair of nodes. For example, the commonly used reconstruction objective of predicting whether two nodes are neighbors would be minimizing the gap between $\mathbf{S}[u, v]$ and $\mathbf{A}[u, v]$.

To achieve the reconstruction objective Eq. (21), the standard practice is to minimize the empirical reconstruction loss $\mathcal{L}$ defined for all the training data $\mathcal{D}$, including both labeled and unlabeled nodes,

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} \ell\left(\mathrm{Dec}\left(\mathrm{Enc}[u], \mathrm{Enc}[v]\right), \mathbf{S}[u, v]\right), \tag{22}$$

where $\ell \colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a loss function for every node pair to compute the inconsistency between the true similarity values and the decoded ones.

B. Shallow embedding and deep embedding

In most of the work on node embedding, the encoder can be classified into a shallow embedding approach, in which this encoder is simply a lookup function based on the node ID. Additionally, the encoder can use both node features and the local graph structure around each node as the input to generate an embedding, like graph neural networks (GNNs). These methods are further categorized into the deep embedding method.

VI. Shallow Graph Embedding

Some specialized optimization methods based on matrix factorization can be employed as a deterministic way to solve the optimization problem Eq. (22). Generally speaking, the whole task can be considered as using matrix factorization methods to learn a low-dimensional approximation of a similarity matrix $\mathbf{S}$, where S encodes the information related to the original adjacency matrix or other matrix measurements.

Unlike the deterministic factorization methods, recent years have witnessed a surge in successful methods that use stochastic measures of neighborhood overlap to generate shallow embeddings. The key innovation in these approaches is that node embeddings are optimized under the assumption that if two nodes in the graph co-occur on some short-length random walks with high probability, they tend to share similar embeddings [42].

A. Factorization-based methods

For the category of factorization-based methods, a matrix that indicates the relationship between every node pair is factorized to obtain the node embedding. This matrix typically contains some underlying structural information about the constructed similarity graph, such as adjacency matrix and normalized Laplacian matrix. Different matrix properties can lead to different ways of factorizing these matrices. For instance, it is obvious that the normalized Laplacian matrix is positive semi-definite, so eigenvalue decomposition is a natural fit.

Table III applies the encoder-decoder perspective to summarize some representative factorization-based shallow embedding approach on node level for graphs. The most critical benefit of the previously mentioned encoder-decoder framework in Section V-A is that it provides a general overview of their respective components so that it is much easier to compare different embedding methods.

1) Locally linear embedding (LLE): The most fundamental assumption in LLE [43] is that the embedding result of each node is just a linear combination of the nodes in its neighborhood. More specifically, each entry W_{ij} in the weight matrix W for the constructed graph can denote how much the node j contributes to the embedding of node i, namely the weight factor for the node j in the linear combination of the node i. Formally, given the definition of Y_i,

$$Y_i \approx \sum_j W_{ij} Y_j, \quad \forall i \in V. \tag{23}$$

The embedding can be obtained as,

$$\phi(Y) = \sum_i \left| Y_i - \sum_j W_{ij} Y_j \right|^2. \tag{24}$$

Adding another two constraints $\frac{1}{N} Y^T Y = I$ and $\sum_i Y_i = 0$ into the above optimization equation, translational invariance can be eliminated since the embedding is forced to around the origin. It has been proven that the solution to this problem is to compute all the eigenvectors of the sparse matrix $(I - W)^T (I - W)$, sort the corresponding eigenvalues in the descending order and take the first $d + 1$ eigenvectors as the final embedding result.

2) Laplacian eigenmaps: Laplacian Eigenmaps [44] makes strongly connected nodes close to each other in the embedding space. Unlike the LLE [43], the objective function is designed in a pairwise manner,

$$\begin{aligned} \phi(Y) &= \frac{1}{2} \sum_{i,j} |Y_i - Y_j|^2 W_{ij}, \\ &= \mathrm{tr}\left(Y^T L Y\right), \end{aligned} \tag{25}$$

where L is the Laplacian matrix. Similar to LLE [43], it is necessary to add another constraint $Y^T D Y = I$ so that some trivial solutions can be removed. The optimal solution is achieved by choosing the eigenvectors of the normalized Laplacian matrix whose corresponding eigenvalues are among the d smallest ones.

3) Graph factorization: Graph Factorization (GF) [45] is the first algorithm to reduce the time complexity of previous graph embedding algorithms to $O(E)$. Instead of targeting at factorizing Laplacian matrix like LLE [43] and Laplacian

Table III: Summary on Factorization-based Shallow Graph Embedding Methods

Method	Decoder	Similarity measure	Loss function	Time complexity
LLE [43]	$\mathbf{z}_u - \sum_v A_{uv}\mathbf{z}_v$	A_{uv}	$\sum_u \|\mathbf{z}_u - \sum_v A_{uv}\mathbf{z}_v\|^2$	$O\left(\|E\|d^2\right)$
Laplacian Eigenmaps [44]	$\|\mathbf{z}_u - \mathbf{z}_v\|_2^2$	A_{uv}	$\mathrm{Dec}\left(\mathbf{z}_u, \mathbf{z}_v\right) \cdot \mathbf{S}[u, v]$	$O\left(\|E\|d^2\right)$
Graph Factorization [45]	$\mathbf{z}_u^\top \mathbf{z}_v$	A_{uv}	$\|\mathrm{Dec}\left(\mathbf{z}_u, \mathbf{z}_v\right) - \mathbf{S}[u, v]\|_2^2$	$O\left(\|E\|d\right)$
GraRep [46]	$\mathbf{z}_u^\top \mathbf{z}_v$	$A_{uv}, \ldots, A_{uv}^k$	$\|\mathrm{Dec}\left(\mathbf{z}_u, \mathbf{z}_v\right) - \mathbf{S}[u, v]\|_2^2$	$O\left(\|V\|^3\right)$
HOPE [47]	$\mathbf{z}_u^\top \mathbf{z}_v$	General Similarity Matrix $\mathbf{S}$	$\|\mathrm{Dec}\left(\mathbf{z}_u, \mathbf{z}_v\right) - \mathbf{S}[u, v]\|_2^2$	$O\left(\|E\|d^2\right)$

Eigenmaps [44], GF directly employs the adjacency matrix and minimizes the objective function,

$$\phi(Y, \mu) = \frac{1}{2} \sum_{(i,j) \in E} \left(W_{ij} - < Y_i, Y_j >\right)^2 + \frac{\mu}{2} \sum_i \|Y_i\|^2, \tag{26}$$

where μ is a hyper-parameter for the introduced regularization term. Because the adjacency matrix is not necessarily positive semidefinite, the summation over all the observed edges can be regarded as an approximation for the sake of scalability.

4) GraRep: GraRep [46] utilizes the node transition probability matrix which is defined as $T = D^{-1}W$ and k-order proximity is preserved by minimizing the loss $\|X^k - Y_s^k Y_t^{kT}\|_F^2$ where X^k is derived from T^k, Y_s and Y_t are source and target embedding vectors respectively. It then concatenates Y_s^k for all k to form Y_s. The drawback of GraRep is scalability, since T_k can have $O(|V|^2)$ non-zero entries.

5) HOPE: Similar to GraRep [46], HOPE [47] preserves higher-order proximity by minimizing another objective function $\left\|S - Y_s Y_t^T\right\|_F^2$, where S is now the proximity matrix. The similarity measurement is defined in the form of $S = M_g^{-1} M_l$, where M_g and M_l are both sparse matrices. In this fashion, Singular Value Decomposition (SVD) can be applied so as to acquire node embeddings in an efficient manner.

6) M-NMF: While previous methods merely center around the microscopic structure (i.e., the first-order and second-order proximity), the mesoscopic community structure is incorporated into the embedding process for Modularized Nonnegative Matrix Factorization (M-NMF) [48]. The cooperation between the microscopic structure and the mesoscopic structure is established by exploiting the consensus relationship between the representations of nodes and the community structure.

B. Random-walk-based methods

The random walk is a powerful tool to gain approximate results about certain properties of the given graph, such as node centrality [49] and similarity [50]. Consequently, random-walk-based node embedding methods are effective under some scenarios when only part of the graph is accessible or the graph's scale is too large to handle efficiently.

The key points of random-walk-based node embeddings approaches are summarized in Table IV from the encoder-decoder perspective. The similarity function $p_G(v \mid u)$ corresponds to the probability of visiting v on a fixed-length random walk starting from u.

1) DeepWalk: Inspired by the skip-gram model [55], DeepWalk [51] follows the main goal of HOPE [47] and thus preserves higher-order proximity of each node pair. However, DeepWalk takes another approach by maximizing the possibility of encountering the previous k nodes and the following k nodes along one specific random walk with center v_i. In other words, DeepWalk maximizes the log-likelihood function which is defined as $\log \Pr\left(v_{i-k}, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{i+k} \mid Y_i\right)$, where $2k+1$ is the length of the random walk. The decoder is a basic form of a dot-product to reconstruct graph information from the encoded node embeddings.

2) Planetoid: Yang *et al.* [52] propose another GSSL method based on the random walk, called Planetoid, where the embedding of a node is jointly trained to predict the class label and also the context in the given graph. The highlight of Planetoid is that it can be employed both in transductive and inductive settings. The context sampling behind Planetoid is actually built upon DeepWalk. In contrast to DeepWalk, Planetoid can handle graphs with real-value attributes by incorporating negative samples and injecting supervised information. The inductive variant of Planetoid views each node's embedding as a parameterized function of input feature vectors, while the transductive variant only embeds graph structure information.

3) node2vec: Following the same idea of DeepWalk [51], node2vec [56] also tries to preserve higher-order proximity for each node pair but makes full use of biased random walks so that it can balance between the breadth-first (BFS) and depth-first (DFS) search on the given graph to generate more expressive node embeddings. To be more specific, many random walks with fixed length are sampled, and then the possibility of occurrence of subsequent nodes along these biased random walks is maximized.

4) LINE: Previously mentioned methods do not scale in large real-world networks; Tang *et al.* propose LINE [53] to fix this issue by preserving both local and global graph structures with scalability. In particular, LINE combines first-order and second-order proximity, and they are optimized using the KL divergence metric. A decoder based on the sigmoid function is used in the first-order objective, while another decoder identical to the one in node2vec and DeepWalk is used in the second-order objective. Unlike node2vec and DeepWalk, LINE explicitly factorizes proximity measurement instead of implicitly incorporating it with sampled random walks.

5) PTE: PTE [54] is proposed as a new semi-supervised representation learning method for text data. PTE fills the gap that many graph embedding methods are not particularly tuned for any task. The labeled information and various levels of information on word co-occurrence are first interpreted as a text network, which is then embedded into a low-dimensional

Table IV: Summary on Random-Walk-based Shallow Graph Embedding Methods

Method	Decoder	Similarity measure	Loss function	Time complexity
DeepWalk [51]	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k\in V} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v \mid u)$	$-\mathbf{S}[u,v]\log\left(\mathrm{Dec}\left(\mathbf{z}_u,\mathbf{z}_v\right)\right)$	$O(\lvert V\rvert d)$
Planetoid [52]	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k\in V} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v \mid u)$	$\mathbb{E}_{v_n\sim P_n(V)}\left[\log\left(-\sigma\left(\mathbf{z}_u^\top \mathbf{z}_{v_n}\right)\right)\right]$	$O(\lvert V\rvert d)$
node2vec [51]	$\frac{e^{\mathbf{z}_u^\top \mathbf{z}_v}}{\sum_{k\in V} e^{\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v \mid u)$	$\sum_{(u,v)\in\mathcal{D}} -\log\left(\sigma\left(\mathbf{z}_u^\top \mathbf{z}_v\right)\right) - \gamma\mathbb{E}_{v_n\sim P_n(V)}\left[\log\left(-\sigma\left(\mathbf{z}_u^\top \mathbf{z}_{v_n}\right)\right)\right]$	$O(\lvert V\rvert d)$
LINE [53]	$\frac{1}{1-e^{-\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v \mid u)$	$\sum_{(u,v)\in\mathcal{D}} -\log\left(\sigma\left(\mathbf{z}_u^\top \mathbf{z}_v\right)\right) - \gamma\mathbb{E}_{v_n\sim P_n(V)}\left[\log\left(-\sigma\left(\mathbf{z}_u^\top \mathbf{z}_{v_n}\right)\right)\right]$	$O(\lvert V\rvert d)$
PTE [54]	$\frac{1}{1-e^{-\mathbf{z}_u^\top \mathbf{z}_k}}$	$p_{\mathcal{G}}(v \mid u)$	$-\mathbf{S}[u,v]\log\left(p_{\mathcal{G}}(v \mid u)\right)$	$O(\lvert V\rvert d)$

space. This stochastic embedding method preserves the semantic meaning of words and shows a strong representational power for the particular downstream task.

6) HARP: HARP [57] is a general strategy to improve the above-mentioned solutions [51] [56] [53] by avoiding local optima with the help of better weight initialization. The hierarchy of nodes is created by node aggregation in HARP using graph coarsening technique based on the preceding hierarchy layer. After that, the new embedding result can be generated from the coarsen graph, and the refined graph (i.e., the graph in the next level up in the hierarchy) can be initialized with the previous embedding. HARP propagates these node embeddings level by level so that it can be used in combination with random-walk-based approaches in order to achieve better performance.

C. Relationship between random-walk-based and factorization-based methods

Even though shallow embedding methods can be divided into two groups based on whether it is deterministic or stochastic, random-walk-based methods can actually be transformed into the factorization-based group in general. Qiu *et al.* [58] provide a theoretical analysis of the aforementioned random-walk-based methods to show that they all essentially perform implicit matrix factorization and have closed-form solutions. Qiu *et al.* [58] also propose a new framework, NetMF, to factorize these underlying matrices in random-walk-based methods explicitly. An impactful follow-up work, NetSMF [59] extends NetMF [58] to large-scale graphs based on sparse matrix factorization, making it more scalable for large networks in the real world.

D. Limitations of shallow embedding

Although shallow embedding methods have achieved impressive success on many SSL related tasks, it is worth noting that it also has some critical drawbacks that researchers found it hard to overcome with ease.

1) **Lack of shared parameters.** In the encoder module, parameters are not shared between nodes since the encoder directly produces a unique embedding vector for each node. The lack of parameter sharing means that the number of parameters necessarily grows as $O(|V|)$, which can be intractable in massive graphs.
2) **No use of node features.** Another key problem with shallow embedding approaches is that they fail to leverage node features. However, rich feature information could potentially be informative in the encoding process. This is especially true for SSL tasks where each node represents valuable feature information.
3) **Failure in inductive applications.** Shallow embedding methods are inherently transductive [41]. Generating embeddings for new nodes that are observed after the training phase is not possible. This restriction prevents shallow embedding methods from being used on inductive applications.

VII. Deep Graph Embedding

In recent years, a great number of deep embedding approaches have been proposed to handle some of the limitations discussed in Section VI-D. It should be emphasized that these deep embedding approaches differ from the shallow embedding approaches explained in Section VI in that a much more complex encoder, which is often based on deep neural networks (DNN) [60], is constructed and employed. In this manner, the encoder module would incorporate both the structural and attribute information of the graph. For SSL tasks, a top-level classifier needs to be trained to predict class labels for unlabelled nodes under the transductive setting, based on the node embeddings generated by these deep learning models.

A. AutoEncoder-based methods

Apart from the use of deep learning models, autoencoder-based methods also vary from the shallow embedding methods in that a unary decoder is employed instead of a pairwise one. Under the framework of autoencoder-based methods, every node, i, is represented by a high-dimensional vector extracted from a row in the similarity matrix, namely, $\mathbf{s}_i = i^{\text{th}}$ row of $\mathbf{S}$, where $\mathbf{S}_{i,j} = s_{\mathcal{G}}(i,j)$. The autoencoder-based methods aims to first encode each node based on the corresponding vector $\mathbf{s}_i$ and then reconstruct it again from the embedding results, subject to the constraint that the reconstructed one should be as close to the original one as possible (Figure 3):

$$\mathrm{Dec}\left(\mathrm{Enc}\left(\mathbf{s}_i\right)\right) = \mathrm{Dec}\left(\mathbf{z}_i\right) \approx \mathbf{s}_i. \tag{27}$$

From the perspective of the loss function for autoencoder-based methods, it commonly keeps the following form:

$$\mathcal{L} = \sum_{i\in V} \left\|\mathrm{Dec}\left(\mathbf{z}_i\right) - \mathbf{s}_i\right\|_2^2. \tag{28}$$

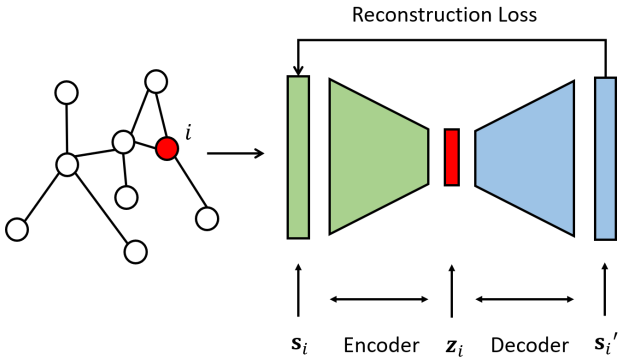

Figure 3: For AutoEncoder-based methods, a high-dimensional vector $\mathbf{s}_i$ is extracted and fed into the AutoEncoder for generating a low-dimensional $\mathbf{z}_i$ embedding

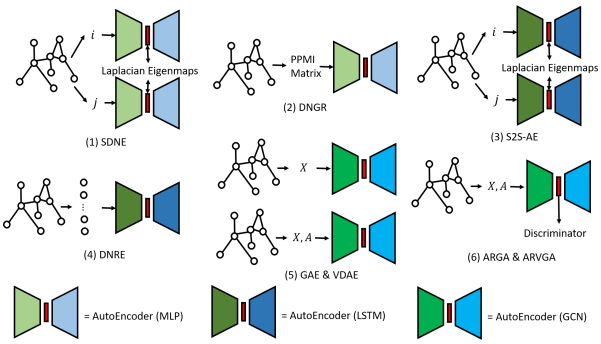

Figure 4: Summary on the architectures of AutoEncoder-based methods

From Eq. (27), it should be pointed out that the encoder module actually depends on the given $\mathbf{s}_i$ vector. This allows autoencoder-based deep embedding approaches to incorporate local structural information into the encoder, while it is simply impossible for the shallow embedding approaches to do so. The primary components of these methods are summarized as Table V, and the architectures of them are compared as Figure 4.

Despite this noticeable enhancement, the autoencoder-based methods may still suffer from some problems. Particularly, the computational cost of it is still intolerable for large scale graphs. Moreover, the structure of the autoencoder is predefined and unchanged during the training, so it is strictly transductive and thus fails to cope with evolving graphs. The up-to-date, relevant representative works to tackle these issues are [67] [68].

1) SDNE: Wang *et al.* [61] proposes Structural deep network integration (SDNE) with the help of deep autoencoders to preserve the proximity for first and second orders. The first-order proximity describes the similarity between each node pair, while the second-order proximity between each node pair describes the proximity of their neighborhood structure. The method takes advantage of non-linear functions to acquire the embedding results. It actually contains two modules: 1) the unsupervised part and 2) the supervised part. The former is an autoencoder designed to produce an embedding result for each node that can be used to rebuild its corresponding vector $\mathbf{s}_i$. For the latter part, Laplacian Eigenmaps is utilized so that penalty is imposed if connected nodes are encoded far away in the embedding space.

2) DNGR: Deep neural networks for learning graph representations (DNGR) [62] integrates random surfing with autoencoders to generate node embeddings. This model has three components: 1) random surfing, 2) estimation of positive pointwise mutual information (PPMI) matrix and 3) stacked denoising autoencoders. For the input graph, random surfing is first applied to produce a co-occurrence probability matrix similar to HOPE. This probabilistic matrix is then converted into a PPMI matrix and fed into a stacked denoising autoencoder to generate the final embedding result. The feedback of the PPMI matrix guarantees the high order proximity is captured and maintained by the autoencoder. Moreover, the introduction of stacked denoising autoencoders enhances the model's robustness when the noise is present and the model's capability to detect the underlying structure required for some downstream tasks like node classification.

3) S2S-AE: Unlike previous methods whose encoders are all based on MLP, Taheri *et al.* [63] extend the form of the encoder to RNN models. S2S-AE [63] uses long short-term memory (LSTM) [69] autoencoders to embed the graph sequences generated from random walks into a continuous vector space. The final representation is computed by averaging its graph sequence representations. The advantage of S2S-AE is that it can support arbitrary-length sequences, unlike others, which often suffer from the limitation of the fixed-length inputs.

4) DRNE: Deep recursive network embedding (DRNE) [64] holds an assumption that the embedding of a node needs to approximate the aggregation of the embeddings of nodes within its neighborhood. It also uses LSTM [69] to aggregate a node's neighbors, so the reconstruction loss is different from the one in S2S-AE [63] as suggested in Table V. In this way, DRNE can solve the issue that the LSTM model is not invariant when the given nodes' sequence permutes in different ways.

5) GAE & VGAE: Both MLP-based and RNN-based methods only consider structural information and ignore the nodes' feature information. GAE [65] leverages GCN [70] to encode both. The encoder takes the form that,

$$\mathrm{Enc}(A, X) = \mathrm{GraphConv}\left(\sigma(\mathrm{GraphConv}(A, X))\right), \quad (29)$$

where $\mathrm{GraphConv}(\cdot)$ is a graph convolutional layer defined in [70], $\sigma(\cdot)$ is the activation function, A is the adjacency matrix, and X is the attribute matrix. The decoder of GAE is defined as

$$\mathrm{Dec}\left(\mathbf{z}_u, \mathbf{z}_v\right) = \mathbf{z}_u^T \mathbf{z}_v. \quad (30)$$

It may have some overfitting issue when the adjacency matrix is reconstructed in a direct way. Variational GAE (VGAE) [65] learns the distribution of data, in which the variational lower bound $\mathcal{L}$ is optimized.

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{Z}|X,A)}[\log p(A \mid \mathbf{Z})] - \mathrm{KL}[q(\mathbf{Z} \mid X, A) \| p(\mathbf{Z})], \quad (31)$$

Table V: Summary on AutoEncoder-based Deep Graph Embedding Methods

Method	Encoder	Deconder	Similarity measure	Loss function	Time complexity
SDNE [61]	MLP	MLP	$\mathbf{s}_u$	$\sum_{u\in V}\left\|\mathrm{Dec}\left(\mathbf{z}_u\right)-\mathbf{s}_u\right\|_2^2$	$O(\|V\|\|E\|)$
DNGR [62]	MLP	MLP	$\mathbf{s}_u$	$\sum_{u\in V}\left\|\mathrm{Dec}\left(\mathbf{z}_u\right)-\mathbf{s}_u\right\|_2^2$	$O\left(\|V\|^2\right)$
S2S-AE [63]	LSTM	LSTM	$\mathbf{s}_u$	$\sum_{u\in V}\left\|\mathrm{Dec}\left(\mathbf{z}_u\right)-\mathbf{s}_u\right\|_2^2$	$O\left(\|V\|^2\right)$
DRNE [64]	LSTM	LSTM	$\mathbf{s}_u$	$\sum_{u\in V}\left\|\left(\mathbf{z}_u\right)-\sum_{v\in\mathcal{N}(u)}\mathrm{LSTM}\left(\mathbf{z}_v\right)\right\|_2^2$	$O(\|V\|\|E\|)$
GAE [65]	GCN	$\mathbf{z}_u^\top\mathbf{z}_v$	A_{uv}	$\sum_{u\in V}\left\|\mathrm{Dec}\left(\mathbf{z}_u\right)-A_u\right\|_2^2$	$O(\|V\|\|E\|)$
VGAE [65]	GCN	$\mathbf{z}_u^\top\mathbf{z}_v$	A_{uv}	$\mathbb{E}_{q(\mathbf{Z}\mid X,A)}[\log p(A\mid\mathbf{Z})]-\mathrm{KL}[q(\mathbf{Z}\mid X,A)\|p(\mathbf{Z})]$	$O(\|V\|\|E\|)$
ARGA [66]	GAE	$\mathbf{z}_u^\top\mathbf{z}_v$	A_{uv}	$\min_{\mathcal{G}}\max_{\mathcal{D}}\mathbb{E}_{\mathbf{z}\sim p_z}[\log\mathcal{D}(\mathbf{Z})]+\mathbb{E}_{x\sim p(x)}[\log(1-\mathcal{D}(\mathcal{G}(X,A)))]$	$O(\|V\|\|E\|)$
ARVGA [66]	VGAE	$\mathbf{z}_u^\top\mathbf{z}_v$	A_{uv}	$\min_{\mathcal{G}}\max_{\mathcal{D}}\mathbb{E}_{\mathbf{z}\sim p_z}[\log\mathcal{D}(\mathbf{Z})]+\mathbb{E}_{x\sim p(x)}[\log(1-\mathcal{D}(\mathcal{G}(X,A)))]$	$O(\|V\|\|E\|)$

where $\mathrm{KL}[q(\cdot)\|p(\cdot)]$ is the Kullback-Leibler divergence between $q(\cdot)$ and $p(\cdot)$. Moreover, we have

$$q(\mathbf{Z}\mid X,A)=\prod_{i=1}^{N}\mathcal{N}\left(\mathbf{z}_i\mid\mu_i,\mathrm{diag}\left(\sigma_i^2\right)\right), \tag{32}$$

and

$$p(A\mid\mathbf{Z})=\prod_{i=1}^{N}A_{ij}\sigma\left(\mathbf{z}_i^\top\mathbf{z}_j\right)+\left(1-A_{ij}\right)\left(1-\sigma\left(\mathbf{z}_i^\top\mathbf{z}_j\right)\right). \tag{33}$$

The most recent follow-up work are RWR-GAE [71] which adds a random walk regularizer for GAE and achieves noticeable performance improvement and DGVAE [72] which combines with graph cluster memberships as latent factors to further improve the internal mechanism of VAEs based graph generation.

6) ARGA & ARVGA: To further improve the empirical distribution, $q(\mathbf{Z}\mid\mathbf{X},\mathbf{A})$ in accordance with the prior distribution $p(\mathbf{A}\mid\mathbf{Z})$ in GAE and VGAE, Pan *et al.* [66] propose ARGA and ARVGA with the help of the generative adversarial networks (GANs) [73], in which they take GAE and VGAE as encoder respectively.

B. GNN-based methods

Several up-to-date deep embedding approaches are designed to overcome the shallow embedding approaches' main drawbacks by constructing some specific functions that depend on a node's neighborhood (Figure 5). Graph neural network (GNN), which is heavily utilized in state-of-the-art deep embedding approaches, is considered as a general scheme for defining deep neural networks in the graph structure data.

The main idea is that the representation vectors of nodes can depend not only on the structure of the graph but also on any feature information associated with the nodes. Dissimilar to the previously reviewed methods, graph neural networks use the node features, e.g., node information for a citation network or even simple statistics such as node degree, one-hot vectors, etc., to generate the desired node embeddings.

Like other deep node embedding methods, a classifier is trained on top of the node embeddings generated by the final hidden state in GNN-based models explicitly or implicitly. Afterward, it can be applied to the unlabeled nodes for SSL tasks.

Since GNN consists of two main operations: Aggregate operation and Update operation, these methods will be reviewed from the perspective of the specific operation changed and improved compared with the basic GNN. The main techniques employed in these methods are also listed in Table VI and some representative models.

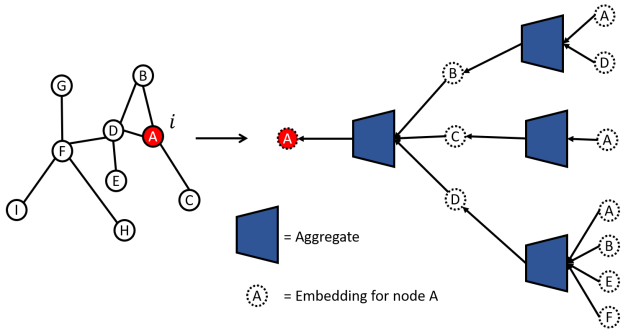

Figure 5: GNN-based methods can generate node embeddings by aggregating embeddings from its neighbors

C. Basic GNN

As Gilmer *et al.* [90] point out, the fundamental feature of a basic GNN is that it takes advantages of *neural message passing* in which messages are exchanged and updated between each pair of the nodes by using neural networks.

More specifically, during each neural message passing iteration in a basic GNN, a hidden embedding $\mathbf{h}_u^{(k)}$ corresponding to each node u is updated according to message or information aggregated from u's neighborhood $\mathcal{N}(u)$. This general message passing update rule can be expressed as follows:

$$\begin{aligned}&\mathbf{h}_u^{(k+1)}\\&=\texttt{Update}^{(k)}\left(\mathbf{h}_u^{(k)},\texttt{Aggregate}^{(k)}\left(\left\{\mathbf{h}_v^{(k)},\forall v\in\mathcal{N}(u)\right\}\right)\right),\\&=\texttt{Update}^{(k)}\left(\mathbf{h}_u^{(k)},\mathbf{m}_{\mathcal{N}(u)}^{(k)}\right).\end{aligned} \tag{34}$$

It is noteworthy that in Eq. (34), both the operation **`Update`** and **`Aggregate`** must be differentiable functions, typically, neural networks. Moreover, $\mathbf{m}_{\mathcal{N}(u)}$ is the exact message that is aggregated from node u's neighborhood $\mathcal{N}(u)$ and tends to encode useful local structure information. Combining the message from neighbourhood with the previous hidden embedding state, the new state is generated according to Eq. (34). After a certain preset number of iterative steps, the last hidden

Table VI: Summary on GNN-based Deep Graph Embedding Methods

Improvement	Technique	Model
Basic GNN (Baseline)	Neural Message Passing	Basic GNN [74]
Generalized Aggregate Operation	Neighborhood Normalization	GCN [70] MixHop [75] SGC [76] DGN [77]
	Pooling	Set Pooling [78] Janossy pooling [79]
	Neighborhood Attention	GAT [80] AGNN [81]
Generalized Update Operation	Concatenation	Column networks [82] Scattering GCN [83] GraphSAGE [84] DropEdge [85]
	Gated Updates	GGNN [86] NeuroSAT [87]
	JK connections	JK Networks [88] InfoGraph* [89]

embedding state converges so that this final state is regarded as the embedding output for each node. Formally, we have,

$$\mathbf{z}_u = \mathbf{h}_u^{(K)}, \forall u \in \mathcal{V}. \tag{35}$$

It should be stressed that both the basic GNN and many of its variants strictly follow this generalized framework. The relationship among them is summarized as Table VI.

Before the review on some of the GNN-based methods designed for SSL tasks, the basic version of GNN is introduced, which is a simplification of the original GNN model proposed by Scarselli *et al.* [74].

The basic GNN message passing is defined as:

$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}_{\text{self}}^{(k)}\,\mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v\in\mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)}\right), \tag{36}$$

where $\mathbf{W}_{\text{self}}^{(k)}, \mathbf{W}_{\text{neigh}}^{(k)}$ are trainable parameters and σ is the activation function. The messages from the neighbors are firstly summarized. Then, the neighborhood information is combined together with the node's previous hidden embedding results by using a basic linear combination. Finally, a non-linearity activation function is applied on the combined information. From the perspective of the key components of the GNN framework, the Aggregation operation is and the Update operation is defined as shown in Eq. (37) and Eq. (38).

$$\texttt{Aggregate}^{(k)}\left(\left\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\right\}\right) = \sum_{v\in\mathcal{N}(u)} \mathbf{h}_v, \tag{37}$$

$$\texttt{Update}\left(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}\right) = \sigma\left(\mathbf{W}_{\text{self}}\,\mathbf{h}_u + \mathbf{W}_{\text{neigh}}\,\mathbf{m}_{\mathcal{N}(u)}\right). \tag{38}$$

Furthermore, it is not uncommon to add some self-loop tricks to the input graph so as to shut out the explicit update step, which can be considered as a straightforward simplification of the neural message passing method used in the basic GNN. To be a little more specific, the message passing process can now be simply defined as shown in Eq. (39).

$$\mathbf{h}_u^{(k)} = \texttt{Aggregate}\left(\left\{\mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}(u) \cup \{u\}\right\}\right). \tag{39}$$

As mentioned before, GNN models have all kinds of variants, which try to improve its performance and robustness to some extent. However, regardless of the variant of GNN, they all follow the neural message passing framework for Eq. (34) examined earlier. In the following two sections, Section VII-D and VII-E, some representative improvements on the two main components of basic GNN, aggregation operation and update operation, are reviewed in detail.

D. Generalized aggregation operation

In general, the Aggregation operation in GNN models has received the most attention from the literature as a large number of researchers have proposed novel architectures or variations based on the original GNN model.

1) Neighborhood normalization: As previously stated, the most basic neighborhood aggregation operation, shown in Eq. (37), solely computes the sum of the neighborhood's embedding states. The main problem with this approach is that it could be unstable and susceptible to the node's degree since nodes with a large degree tend to receive a large total value from more neighbors than those with fewer neighbors.

One typical and simple solution to this issue is to just normalize the aggregation operation based on the degree of the central nodes. The simplest approach is to just take an average rather than the sum by Eq. (40)

$$\mathbf{m}_{\mathcal{N}(u)} = \frac{\sum_{v\in\mathcal{N}(u)} \mathbf{h}_v}{|\mathcal{N}(u)|}, \tag{40}$$

but methods with other normalization factors with similar ideas were proposed and achieved remarkable performance gain, such as the following symmetric normalization employed by Kipf *et al.* [70] in the GCN model as shown in Eq. (41).

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v\in\mathcal{N}(u)} \frac{\mathbf{h}_v}{\sqrt{|\mathcal{N}(u)|\,|\,\mathcal{N}(v)\,|}}. \tag{41}$$

Graph convolutional networks (GCNs). One of the most popular and effective baseline GNN variants is the graph convolutional network (GCN) [70], which is inspired by [91] and [92]. GCN makes full use of the neighborhood normalized aggregation techniques as well as the self-loop update operation. Therefore, the GCN model defines the update operation function as shown in Eq. (42). No aggregation operation is defined since it has been implicitly defined within the update operation function as

$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}^{(k)} \sum_{v\in\mathcal{N}(u)\cup\{u\}} \frac{\mathbf{h}_v}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}\right). \tag{42}$$

There exist a great number of GCN variants to enhance SSL performance from different aspects. Li *et al.* [93] are the first to

provide deep insights into GCN's success and failure on SSL tasks. Later on, extensions to GCN for SSL begin to proliferate. Jiang *et al.* [94] explore the way to do graph construction based on GCN. Yang *et al.* [95] combine the classic graph regularization methods with GCN. Abu *et al.* [96] present a novel N-GCN which marries the random walk with GCN, and a follow-up work GIL [97] with similar ideas is proposed as well. Other research work on GCN extensions can be found in [98] [99] [100] [101] [102] [103] [104] [105] [106].

MixHop. GCN often fails to learn a generalized class of neighborhood with various mixing relationships. In order to overcome this limitation, MixHop [75] is proposed to learn these relationships by repeatedly mixing feature representations of neighbors at various distances. Unlike GCN whose aggregation operator in the matrix form is defined as

$$\mathbf{H}^{(k)} = \sigma\left(A\mathbf{H}^{(k-1)}\mathbf{W}^{(k)}\right), \tag{43}$$

where $\mathbf{H}^{(k-1)}$ and $\mathbf{H}^{(k)}$ are the input and output hidden embedding matrix for layer k. MixHop replaces the Graph Convolution (GC) layer defined in Eq. (43) with

$$\mathbf{H}^{(k)} = \|_{j\in P}\sigma\left(A^j\mathbf{H}^{(k-1)}\mathbf{W}_j^{(i)}\right), \tag{44}$$

where the hyper-parameter P is a set of integer adjacency powers and $\|$ denotes column-wise concatenation. Specifically, by setting $P = \{1\}$, it exactly recovers the original GC layer. In fact, MixHop is interested in higher-order message passing, where each node receives latent representations from their immediate (one-hop) neighbors and from further N-hot neighbors.

Simple graph convolution networks (SGC). GCNs inherit unnecessary complexity and redundant computation cost in nature as it derives inspiration from deep learning methods. Wu *et al.* [76] reduce this excess complexity by eliminating the nonlinearities among every GCN layer and collapsing the original nonlinear function into a simple linear mapping function defined in Eq. (45). More importantly, these simplifications do not harm the prediction performance in many downstream applications.

$$\mathbf{h}_u^{(k)} = \sigma\left(\sum_{v\in\mathcal{N}(u)\cup\{u\}}\frac{\mathbf{h}_v}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}}\right). \tag{45}$$

Differentiable group normalization (DGN). To further mitigate the over-smoothing issue in GCN, DGN [77] also applies a new operation between the successive graph convolutional layers. Taking each embedding matrix $H^{(k)}$ generated from the k^{th} graph convolutional layer as the input, DGN assigns each node into different groups and normalizes them independently to output a new embedding matrix for the next layer. Formally, we have,

$$\mathbf{H}^{(k+1)} = \mathbf{H}^{(k)} + \lambda\sum_{i=1}^{g}\left(\gamma_i(\frac{\mathbf{s}_i^{(k)}\circ\mathbf{H}^{(k)} - \mu_i}{\delta_i}) + \beta_i\right), \tag{46}$$

where $\mathbf{H}^{(k)}$ is the k^{th} layer of the hidden embedding matrix, $\mathbf{s}_i$ is the similarity measure and g is the total number of groups. In particular, μ_i and δ_i denote the vectors of running mean of group i, respectively, and γ_i and β_i denote the trainable scale and shift vectors, respectively.

2) Pooling: Aggregation operation is essentially a mapping from a set of neighborhood embedding results to a single vector with encoded information about the local structure and the feature of neighbor nodes' feature. In the previously reviewed settings of GNN models, the mapping function in the aggregation operation is simply the basic summation or linear functions over neighbor embeddings. Some more sophisticated and successful mapping functions used in the aggregation setting are reviewed in this section.

Set pooling. In fact, according to Wu *et al.* [78], one principal approach for designing an aggregation function is focused on the theory of permutation invariant neural networks. Generally, the permutation invariance property deals with problems concerning a set of objects: the target value for a given set is the same regardless of the order of the objects in the set. A typical example of an invariant permutation model is a convolutional neural network which performs the pooling operation over embedding extracted from a set's elements. Permutation invariance on graphs in general means that the aggregation function does not depend on the arbitrary order of the rows/columns in the adjacency matrix. For example, Zaheer *et al.* [107] show that an aggregation function with the following form can be considered as a universal set function approximator:

$$\mathbf{m}_{\mathcal{N}(u)} = \mathrm{MLP}_\theta\left(\sum_{v\in N(u)}\mathrm{MLP}_\phi\left(\mathbf{h}_v\right)\right). \tag{47}$$

Janossy pooling. Another alternative method, called Janossy pooling, is to enhance the aggregation operation, which is also possibly more efficient than simply taking a sum or mean of the neighbor embeddings used in basic GNN. Janossy pooling [79] uses a completely different approach. Instead of using a permutation invariant reduction (e.g., a sum or a mean), a permutation-sensitive function is applied, and the outcome is averaged over many potential permutations.

Let $\pi_i \in \Pi$ denotes a permutation function that maps the set $\{\mathbf{h}_v, \forall v \in \mathcal{N}(u)\}$ to a specific sequence $\left(\mathbf{h}_{v_1}, \mathbf{h}_{v_2}, \ldots, \mathbf{h}_{v_{|\mathcal{N}(u)|}}\right)_{\pi_i}$. Namely, $\pi_i \in \Pi$ takes the unordered set of embedding states from the neighbors and puts them in a sequence dependent on some random ordering arbitrarily. The Janossy pooling approach then performs neighborhood aggregation operation by Eq. (48).

$$\mathbf{m}_{\mathcal{N}(u)} = \mathrm{MLP}_\theta\left(\frac{1}{|\Pi|}\sum_{\pi\in\Pi}\rho_\phi\left(\mathbf{h}_{v_1}, \mathbf{h}_{v_2}, \mathbf{h}_{v_{|\mathcal{N}(u)|}}\right)_{\pi_i}\right), \tag{48}$$

where Π denotes a collection of permutations and ρ_ϕ is a permutation-sensitive function, e.g., a neural network that operates on sequential data. Usually ρ_ϕ is represented as an LSTM in operation, since LSTMs are known to be a powerful architecture for sequences in the neural network.

3) Neighborhood attention: A common approach for enhancing the aggregation layer in GNNs is to implement some attention mechanisms [108], in addition to more general forms of set aggregation. The basic principle is to assign a weight or value of importance to each neighbor, which is used during the aggregation phase to weigh this neighbor's effect.

GAT. The first GNN model to apply this style of attention was Cucurull *et al.*'s Graph Attention Network (GAT) [80], which uses attention weights to define a weighted sum of the neighbors:

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \mathbf{h}_v, \tag{49}$$

where $\alpha_{u,v}$ denotes the attention on neighbor $v \in \mathcal{N}(u)$ when we are aggregating information at node u. In the original GAT paper, the attention weights are defined as

$$\alpha_{u,v} = \frac{\exp\left(\mathbf{a}^\top \left[\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_v\right]\right)}{\sum_{v' \in \mathcal{N}(u)} \exp\left(\mathbf{a}^\top \left[\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_{v'}\right]\right)}, \tag{50}$$

where $\mathbf{a}$ is a trainable attention vector, $\mathbf{W}$ is a trainable matrix, and $\oplus$ denotes the concatenation operation. A similar parallel work is AGNN [81] which reduces the number of parameters in GNN with the help of attention mechanisms.

E. Generalized update operation

As already noted in Section VII-D, lots of research papers focus on generalized aggregate operation. This was especially the case after the GraphSAGE Framework [84], which implements the idea of generalized neighborhood aggregation. This section concentrates on the more diversified Update operation, which also makes the embeddings more suitable for SSL tasks.

1) Concatenation and skip-connections: Over-smoothing is a major issue for GNN. The over-smoothing is almost inevitable after many message iterations when the node-specific information becomes "washed away". In such cases, the modified node representations are too highly dependent on the incoming message aggregated by the neighbors at the cost of previous layers' node hidden states. One reasonable way to mitigate this problem is to use vector concatenations or skip connections, which aim to retain information directly from previous rounds of the update.

These techniques can actually be used in combination with several other update operation methods for the GNN. For general purposes, $\texttt{Update}_{\texttt{base}}$ denotes the simple update rule that will be built on. For instance, the $\texttt{Update}_{\texttt{base}}$ function can be presumed as shown in Eq. (38) in the basic GNN.

GraphSAGE. One of the simplest updates for skip connection is GraphSAGE [84] which uses a concatenation vector to hold more information from node level during message passing process:

$$\texttt{Update}\left(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}}(u)\right) = \left[\texttt{Update}_{\texttt{base}}\left(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}\right) \oplus \mathbf{h}_u\right], \tag{51}$$

where the output from the simple update function is concatenated with the node 's previous layer representation. The core intuition is that the model is encouraged to dissociate information during the message passing.

Column Network (CLN). Besides concatenation methods, some other forms of skip-connections can also be applied, such as the linear interpolation method proposed by Pham *et al.* [82],

$$\begin{aligned}\texttt{Update}\left(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}}(u)\right) = \boldsymbol{\alpha}_1 \circ \texttt{Update}_{\texttt{base}}\left(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}}(u)\right) \\ + \boldsymbol{\alpha}_2 \circ \mathbf{h}_u,\end{aligned} \tag{52}$$

where $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \in [0,1]^d$ are gating vectors with $\boldsymbol{\alpha}_1 + \boldsymbol{\alpha}_2 = 1$ and $\circ$ denotes Hadamard product. In this method the final update is a linear interpolation between the previous output and the current output and is modified depending on the information in the neighborhood.

Scattering GCN. The most recent work on tackling the problem of over-smoothing in GNN is Scattering GCN [83] with the geometric scattering transformation that enables band-pass filtering of graph signals. Geometric scattering is originally introduced in the context of whole-graph classification and consisted of aggregating scattering features. Similar and concurrent work is DropEdge [85] which removes a certain portion of edges from the given graph at each training epoch, acting like a data augmenter and thus alleviate both over-smoothing and over-fitting issues at the same time.

These strategies are also beneficial for node classification tasks with relatively deep GNNs, in a semi-supervised setting, and they are excellent for these SSL tasks where the prediction in each node is closely correlated with the characteristics of the local neighborhood.

2) Gated updates: Parallel to the above-mentioned work, the researchers have also taken inspiration from the approaches used by recurrent neural networks (RNNs) to strengthen stability. One way to interpret the GNN message passing algorithm is to collect an observation from the neighbors from the aggregation operation, which then updates the hidden state of each node. From this perspective, some methods for updating the hidden status of RNN architectures can be directly applied based on the observation.

GatedGNN. For example, one of the earliest GNN variants which put this idea into practice is proposed by Li *et al.* [86], in which the update operation is defined as shown in Eq. (53) as,

$$\mathbf{h}_u^{(k)} = \text{GRU}\left(\mathbf{h}_u^{(k-1)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}\right), \tag{53}$$

where GRU is a gating mechanism function in recurrent neural networks, introduced by Kyunghyun Cho *et al.* [109]. Another approach called NeuroSAT [87] has employed updates based on the LSTM architecture as well.

3) Jumping knowledge (JK) connections: In the previous sections, it is implicitly assumed that the last layer's output is considered as the final embedding result. In other words, the node representations used for a downstream job, such as SSL tasks, are identical to the final layer's node embedding in the GNN. Formally, it is presumed that

$$\mathbf{z}_u = \mathbf{h}_u^{(K)}, \forall u \in \mathcal{V}. \tag{54}$$

JK Net. A complementary approach to increase the effectiveness of final node representations is to use the combination on each layer of the message passing, rather than merely the final layer's output. In a more formal way,

$$\mathbf{z}_u = f_{JK}\left(\mathbf{h}_u^{(0)} \oplus \mathbf{h}_u^{(1)} \oplus \ldots \oplus \mathbf{h}_u^{(K)}\right), \tag{55}$$

This technique is originally introduced and tested by Xu *et al.* [88], called the idea of jumping knowledge connections. The f_{JK} function can be used as the identity function for various applications, meaning that a simple concatenation is essentially performed among the node embeddings from each

layer, but Xu *et al.* [88] also think about other possibilities such as max pooling. This method also leads to significant progress over a wide range of tasks like SSL classification and is usually regarded as an effective strategy.

InfoGraph*. Previous GNN models appear to have low generalization performance because of the model's crafted styles. To lessen the downgrade of generalization performance in the testing phase, InfoGraph* [89] maximizes the mutual information between the embeddings learned by popular supervised learning methods and unsupervised representations learned by InfoGraph, where the given graph is encoded to produce its corresponding feature map by jumping knowledge concatenation. The encoder then learns from unlabeled samples while preserving the implicit semantic information that the downstream task favors.

VIII. Applications

A. Datasets

We summarize the commonly-used datasets in graph-based semi-supervised learning according to seven different domains, namely citation networks, co-purchase networks, web page citation networks, and others. As shown in Table VII in Appendix A, the summary results on selected benchmark datasets on GSSL are listed with their respective statistical analysis.

B. Open-source Implementations

Here we list some open-source implementations for GSSL in Table VIII in Appendix B.

C. Domains

GSSL has a large number of successful applications across various domains. Some domains have graph-structure data in nature, while others do not. The former ones would be scenarios where raw data samples have explicit relational structure and can be easily constructed into a graph, such as traffic network in the cyber-physical systems (CPS), molecular structure in biomedical engineering, and friend recommendation in social networks. From the non-graph-structured data, however, a graph cannot be extracted directly. Typical examples would be more common scenarios, like image classification in computer vision and text classification in NLP.

1) Computer vision: Among many computer vision tasks (CV), hyperspectral image classification (HSI) is a representative example for GSSL applications. For one thing, labeled data in HSI is costly and scarce. For another, among all popular SSL methods, classic GSSL methods have elegant closed-form solutions and are easy to implement. Shao *et al.* [110] [111] propose a spatial and class structure regularized sparse representation graph for semi-supervised HSI classification. Later, Fang *et al.* [112] extend this work [111] by providing a more scalable algorithm based on anchor graph [113].

Pedronette *et al.* [114] also improve the KNN-based graph construction methods [8] in Section III-A1 to facilitate image retrieval. Another similar idea by Shi *et al.* [115] is to use a temporal graph to assist image analysis.

The latest interesting work based on GSSL in CV is related to domain adaptation by He *et al.* [116]. In this work, a novel idea of using graph-based manifold representation to do visual-audio transformation is proposed and examined.

2) Natural language processing: Amarnag *et al.* [117] first introduce GSSL into traditional natural language processing (NLP) tasks and make pioneering work on part-of-speech (POS) tagging based on random fields [24]. The proposed algorithm uses a similarity graph to encourage similar n-grams to have similar POS tags. Later, Aliannejadi *et al.* [118] and Qiu *et al.* [59] extend this work and use some GCN-based methods [70] to make the model more robust on other various natural language understanding (NLU) tasks.

More recent works on how to combine GSSL and NLP tasks center around graph smoothing problems. Mei *et al.* [119] propose a brand-new general optimization framework for smoothing language models with graph structures, which can further enhance the performance of information retrieval tasks. By constructing a similarity graph of documents and words, various types of GSSL methods can be performed.

Unlike the work [119] which studies the long texts, Hu *et al.* [120] focus on short texts in which the labeled data is sparse and limited. In particular, a flexible model based on GAT [80] with a dual-level attention mechanism is presented.

3) Social networks: A social network is a set of people with some pattern of interactions or "ties" between them and has graph-structured data explicitly. As is known to all, Twitter is one of the most famous and large-scale social networks, so various meaningful and interesting tasks based on GSSL can be performed on the Twitter dataset. Alam *et al.* [121] adopt a graph-based deep learning framework by Yang *et al.* [122] for learning an inductive semi-supervised model to classify tweets in a crisis situation. Later, Anand *et al.* [123] improve classic GSSL methods to detect fake users from a large volume of Twitter data.

Another popular topic in social networks is related to POI recommendations, such as friend recommendation and follower suggestion. Yang *et al.* [124] propose a general SSL framework to alleviate data scarcity via smoothing among users and POIs in the bipartite graph. Moreover, Chen *et al.* [113] employ a user profiling approach to boost the performance of POI recommendation based heterogeneous graph attention networks by Wang *et al.* [125].

4) Biomedical science: Graphs are also ubiquitous in the area of biomedical science, such as the semantic biomedical knowledge graphs, molecular graphs for drugs, and protein-drug interaction for drug proposals. Doostparast *et al.* [126] use GSSL methods with genomic data integration to do phenotype classification tasks. In the meantime, Luo *et al.* [127] provide a new graph regularization framework in heterogeneous networks to predict human miRNA-disease [128]. Other typical applications are disease diagnosis [129], medical image segmentation [130] and medical Image classification [131].

IX. Open problem

Here a chronological overview of the mentioned representative methods in this survey is provided in Figure 6. From 2000

to 2012, the mainstream algorithm was centered around graph regularization and matrix factorization in the early years. After the resurgence of deep learning in 2015, the field witnessed the emergence of AutoEncoder-based methods while, in the meantime, random-walk-based methods coexisted. However, with the introduction of GCN [70] in 2017, the GNN-based method became the dominant solution and still is a heated topic now. Based on this trend, we list four potential research topics.

A. Dynamicity

Conventional GSSL methods reviewed in the preceding sections all treat the graph as a fixed observation. Ma *et al.* [132] is the first to apply generative models on GSSL. By viewing the graph as a random variable, the generated joint distribution can extract more general relationships among attributes, labels, and the graph structure. Moreover, this kind of model is more robust to missing data. Some latest follow-up works are [133] [134] and [135]. The most up-to-date work is [136], which provides a multi-source uncertainty framework for GNN and considers different types of uncertainties associated with class probabilities.

B. Scalability

Another open problem is how to make GSSL methods scalable when the input graph size increases rapidly. Pioneering work has been done by Liu *et al.* [137] [138], in which a novel graph is constructed with data points and anchor, namely, anchor graph regularization (AGR). Many successful follow-up works are proposed, such as [139] [140] for graph regularization methods and [141] [142] for GNN methods. These methods focus more on computational complexity instead of classification accuracy. It is worth noting that the most up-to-date work is GBP [143] which invents a new localized bidirectional propagation process from both the feature vectors and the nodes.

C. Noise-resilience

Graphs with noise or missing attributes are also heated topics. Most existing GSSL methods end up fully trusting the given few labels, but in real life, these labels are highly reliable as they are often produced by humans prone to mistakes. Stretcu *et al.* [144] propose Graph Agreement Models (GAM), which introduces an auxiliary model that predicts the probability of two nodes sharing the same label. This co-training approach makes the model more resilient to noise. Zhao *et al.* [136] consider different types of uncertainties associated with class probabilities in the real case scenario. Similar latest works with the same goal but different approaches are [104] [145] [146]. In addition, Dunlop *et al.* [147] provide theoretical insights into this topic.

D. Attack-robustness

Robustness is always a common concern in machine learning systems. Liu *et al.* [148] first propose a general framework for data poisoning attacks to GSSL. While [148] focuses more on how to generate a successful attack, [149] and [150] defend GNN models from these adversarial attacks. Other relevant works via various approaches [151] [152] [153] can be found as well.

X. Conclusion

To sum up, we conduct a comprehensive review of graph-based semi-supervised learning. A new taxonomy is proposed, in which all the popular label inference methods are grouped into two main categories: graph regularization and graph embedding. Moreover, they can be generalized within the regularization framework and the encoder-decoder framework respectively. Then a wide range of applications of GSSL are introduced along with relevant datasets, open-source codes for some of the GSSL methods. A chronological overview of these representative GSSL methods is presented in the Appendix. Finally, four open problems for future research directions are discussed as well.

Appendix A
Datasets collection for GSSL

Appendix B
Open-source Implementations

Appendix C
Chronological overview of GSSL

Here a chronological overview of the mentioned representative methods in this survey is provided in Figure 6. From 2000 to 2012, the mainstream algorithm was centered around graph regularization and matrix factorization in the early years. After the resurgence of deep learning in 2015, the field witnessed the emergence of AutoEncoder-based methods while, in the meantime, random-walk-based methods coexisted. However, with the introduction of GCN [70] in 2017, the GNN-based method became the dominant solution and still is a heated topic now.

References

[1] A. Subramanya and P. P. Talukdar, *Graph-Based Semi-Supervised Learning*, ser. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2014.

[2] X. Zhu, "Semi-supervised learning literature survey," Computer Sciences, University of Wisconsin-Madison, Tech. Rep. 1530, 2005.

[3] N. N. Pise and P. Kulkarni, "A survey of semi-supervised learning methods," in *2008 International Conference on Computational Intelligence and Security*, vol. 2, 2008, pp. 30–34.

[4] V. J. Prakash and L. M. Nithya, "A survey on semi-supervised learning techniques," *CoRR*, vol. abs/1402.4645, 2014.

[5] J. E. Van Engelen and H. H. Hoos, "A survey on semi-supervised learning," *Machine Learning*, vol. 109, no. 2, pp. 373–440, 2020.

[6] Y. Ouali, C. Hudelot, and M. Tami, "An overview of deep semi-supervised learning," *arXiv preprint arXiv:2006.05278*, 2020.

[7] Y. Chong, Y. Ding, Q. Yan, and S. Pan, "Graph-based semi-supervised learning: A review," *Neurocomputing*, vol. 408, pp. 216 – 230, 2020.

[8] O. Chapelle, B. Schlkopf, and A. Zien, *Semi-Supervised Learning*, 1st ed. The MIT Press, 2010.

[9] T. Jebara, J. Wang, and S. Chang, "Graph construction and *b*-matching for semi-supervised learning," in *ICML*, ser. ACM International Conference Proceeding Series, vol. 382. ACM, 2009, pp. 441–448.

[10] K. Ozaki, M. Shimbo, M. Komachi, and Y. Matsumoto, "Using the mutual k-nearest neighbor graphs for semi-supervised classification on natural language data," in *CoNLL*. ACL, 2011, pp. 154–162.

Table VII: Summary of selected benchmark datasets on GSSL.

Category	Dataset	# Nodes	# Edges	# Features	# Classes	Source
Citation networks	Cora	2,708	5,429	1,433	7	[154]
	Citeseer	3,327	4,732	3,703	6	[154]
	Pubmed	19,717	44,338	500	3	[154]
	DBLP	29,199	133,664	0	4	[155]
Co-purchase networks	Amazon Computers	13,752	245,861	767	10	[156]
	Amazon Photo	7,650	119,081	745	8	[156]
	Coauthor CS	18,333	81,894	6,805	15	[156]
	Coauthor Physics	34,493	247,962	8,415	5	[156]
Webpage citation networks	Cornell	195	286	1,703	5	[157]
	Texas	187	298	1,703	5	[157]
	Washington	230	417	1,703	5	[157]
	Wsicsonsin	265	479	1,703	5	[157]
Social networks	Zachary's karate club	34	77	0	2	[158]
	Reddit	232965	11606919	602	41	[84]
	BlogCatalog	10312	333983	0	39	[159]
	Flickr	1,715,256	22,613,981	0	5	[160]
	Youtube	1,138,499	2,990,443	0	47	[160]
Language networks	Wikipedia	1,985,098	1,000,924,086	0	7	[161]
Bio-chemical	PPI	56944	818716	50	121	[162]
	NCI-1	29	32	37	2	[163]
	MUTAG	17	19	7	2	[164]
	D&G	284	715	82	2	[165]
	PROTEIN	39	72	4	2	[166]
	PTC	25		19	2	[167]
Knowledge graph	Nell	65755	266144	61278	210	[168]
Image	MNIST	10,000	65,403	128	10	[169]
	SVHN	10,000	68,844	128	10	[170]
	CIFAR10	10,000	70,391	128	10	[171]

[11] D. A. Vega-Oliveros, L. Berton, A. M. Eberle, A. de Andrade Lopes, and L. Zhao, "Regular graph construction for semi-supervised learning," in *Journal of physics: Conference series*, vol. 490. IOP Publishing, 2014, p. 012022.

[12] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[13] P. S. Dhillon, P. P. Talukdar, and K. Crammer, "Learning better data representation using inference-driven metric learning," in *ACL (Short Papers)*. The Association for Computer Linguistics, 2010, pp. 377–381.

[14] M. H. Rohban and H. R. Rabiee, "Supervised neighborhood graph construction for semi-supervised classification," *Pattern Recognition*, vol. 45, no. 4, pp. 1363 – 1372, 2012.

[15] L. Berton and A. d. A. Lopes, "Graph construction based on labeled instances for semi-supervised learning," in *2014 22nd International Conference on Pattern Recognition*, 2014, pp. 2477–2482.

[16] L. Berton and A. de Andrade Lopes, "Graph construction for semi-supervised learning," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[17] L. Berton, T. de Paulo Faleiros, A. Valejo, J. Valverde-Rebaza, and A. de Andrade Lopes, "Rgcli: Robust graph that considers labeled instances for semi-supervised learning," *Neurocomputing*, vol. 226, pp. 238 – 248, 2017.

[18] L. Zhuang, Z. Zhou, S. Gao, J. Yin, Z. Lin, and Y. Ma, "Label information guided graph construction for semi-supervised learning," *IEEE Transactions on Image Processing*, vol. 26, no. 9, pp. 4182–4192, 2017.

[19] F. Taherkhani, H. Kazemi, and N. M. Nasrabadi, "Matrix completion for graph-based deep semi-supervised learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 5058–5065.

[20] D. Zhou and B. Schölkopf, "A regularization framework for learning from graph data," in *ICML 2004 Workshop on Statistical Relational Learning and Its Connections to Other Fields (SRL 2004)*, 2004, pp. 132–137.

[21] R. K. Ando and T. Zhang, "Learning on graph with laplacian regularization," in *Advances in neural information processing systems*, 2007, pp. 25–32.

[22] J. Calder and D. Slepčev, "Properly-weighted graph laplacian for semi-supervised learning," *Applied Mathematics & Optimization*, pp. 1–49, 2019.

[23] F. Hoffmann, B. Hosseini, Z. Ren, and A. M. Stuart, "Consistency of semi-supervised learning algorithms on graphs: Probit and one-hot methods," *Journal of Machine Learning Research*, vol. 21, no. 186, pp. 1–55, 2020.

Table VIII: A Summary of Open-source Implementations

		Model	Project Link
Graph Regularization	Label Propagation	GRF [24]	https://github.com/parthatalukdar/junto
		LRC [25]	https://github.com/provezano/lgc
Shallow Graph Embedding	Matrix Factorization	Laplacian Eigenmaps [44]	https://github.com/thunlp/OpenNE
		Graph Factorization [45]	https://github.com/thunlp/OpenNE
		GraRep [46]	https://github.com/benedekrozemberczki/GraRep
		HOPE [47]	https://github.com/ZW-ZHANG/HOPE
		M-NMF [48]	https://github.com/benedekrozemberczki/M-NMF
	Random Walk	DeepWalk [51]	https://github.com/phanein/deepwalk
		Planetoid [52]	https://github.com/kimiyoung/planetoid
		node2vec [56]	https://github.com/eliorc/node2vec
		LINE [53]	https://github.com/tangjianpku/LINE
		PTE [54]	https://github.com/mnqu/PTE
		HARP [57]	https://github.com/GTmac/HARP
	Hybid	NetMF [58]	https://github.com/xptree/NetMF
		NetSMF [59]	https://github.com/xptree/NetSMF
Deep Graph Embedding	AutoEncoder	SDNE [61]	https://github.com/suanrong/SDNE
		DNGR [62]	https://github.com/ShelsonCao/DNGR
		DRNE [64]	https://github.com/tadpole/DRNE
		GAE [65]	https://github.com/tkipf/gae
		VGAE [65]	https://github.com/DaehanKim/vgae_pytorch
		ARGA [66]	https://github.com/Ruiqi-Hu/ARGA
	GNN	GCN [70]	https://github.com/tkipf/gcn
		MixHop [75]	https://github.com/samihaija/mixhop
		SGC [76]	https://github.com/Tiiiger/SGC
		DGN [77]	https://github.com/Kaixiong-Zhou/DGN
		Janossy pooling [79]	https://github.com/PurdueMINDS/JanossyPooling
		GAT [80]	https://github.com/PetarV-/GAT
		AGNN [81]	https://github.com/dawnranger/pytorch-AGNN
		GraphSage [84]	https://github.com/williamleif/GraphSAGE
		DropEdge [85]	https://github.com/DropEdge/DropEdge
		Column networks [82]	https://github.com/trangptm/Column_networks
		Scattering GCN [83]	https://github.com/dms-net/scatteringGCN
		GGNN [86]	https://github.com/yujiali/ggnn
		NeuroSAT [87]	https://github.com/dselsam/neurosat
		JK Networks [88]	https://github.com/mori97/JKNet-dgl
		InfoGraph* [89]	https://github.com/fanyun-sun/InfoGraph

[24] X. Zhu, Z. Ghahramani, and J. D. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *Proceedings of the 20th International conference on Machine learning (ICML-03)*, 2003, pp. 912–919.

[25] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, "Learning with local and global consistency," in *Advances in neural information processing systems*, 2004, pp. 321–328.

[26] D. Slepcev and M. Thorpe, "Analysis of p-laplacian regularization in semisupervised learning," *SIAM Journal on Mathematical Analysis*, vol. 51, no. 3, pp. 2085–2120, 2019.

[27] D. Zhou, J. Huang, and B. Schölkopf, "Learning from labeled and unlabeled data on a directed graph," in *Proceedings of the 22nd International Conference on Machine Learning*, ser. ICML '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 1036–1043.

[28] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples," *J. Mach. Learn. Res.*, vol. 7, pp. 2399–2434, 2006.

[29] C. Gong, T. Liu, D. Tao, K. Fu, E. Tu, and J. Yang, "Deformed graph laplacian for semisupervised learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 10, pp. 2261–2274, 2015.

[30] J. Calder, B. Cook, M. Thorpe, and D. Slepcev, "Poisson learning: Graph based semi-supervised learning at very low label rates," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 1306–1316.

[31] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," Carnegie Mellon University, Tech. Rep., 2002.

[32] A. Iscen, G. Tolias, Y. Avrithis, and O. Chum, "Label propagation for deep semi-supervised learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 5070–5079.

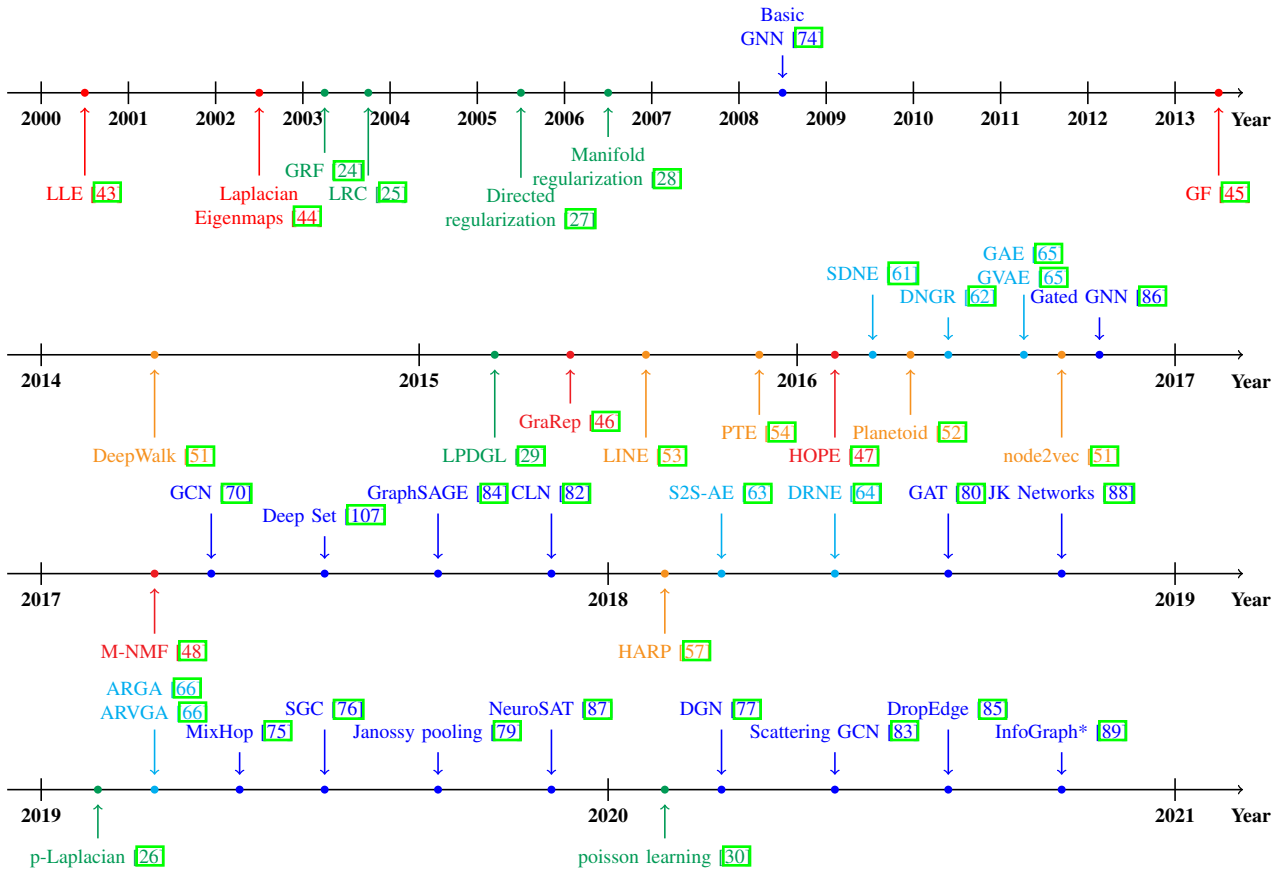

Figure 6: Chronological overview of representative GSSL methods. Green indicates graph regularization methods. Red indicates matrix-factorization-based methods. Orange indicates random-walk-based methods. Cyan indicates AutoEncoder-based methods. Blue indicates GNN-based methods. Models shown above the chronological axis are involved with deep learning techniques while those below the chronological axis are not. (Better viewed in color.)

[33] Z. Xu, I. King, M. R. Lyu, and R. Jin, "Discriminative semi-supervised feature selection via manifold regularization," *IEEE Trans. Neural Networks*, vol. 21, no. 7, pp. 1033–1047, 2010.

[34] A. Argyriou, C. A. Micchelli, and M. Pontil, "When is there a representer theorem? vector versus matrix regularizers," *J. Mach. Learn. Res.*, vol. 10, pp. 2507–2529, 2009.

[35] P. Niyogi, "Manifold regularization and semi-supervised learning: some theoretical analyses," *J. Mach. Learn. Res.*, vol. 14, no. 1, pp. 1229–1250, 2013.

[36] A. Talwalkar, S. Kumar, M. Mohri, and H. A. Rowley, "Large-scale SVD and manifold learning," *J. Mach. Learn. Res.*, vol. 14, no. 1, pp. 3129–3152, 2013.

[37] Y. Zhang, X. Zhang, X. Yuan, and C. Liu, "Large-scale graph-based semi-supervised learning via tree laplacian solver," in *AAAI*. AAAI Press, 2016, pp. 2344–2350.

[38] X. Chang, S. Lin, and D. Zhou, "Distributed semi-supervised learning with kernel ridge regression," *J. Mach. Learn. Res.*, vol. 18, pp. 46:1–46:22, 2017.

[39] J. Li, Y. Liu, R. Yin, and W. Wang, "Approximate manifold regularization: Scalable algorithm and generalization analysis," in *IJCAI*. ijcai.org, 2019, pp. 2887–2893.

[40] X. Li and Y. Guo, "Adaptive active learning for image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 859–866.

[41] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Eng. Bull.*, vol. 40, no. 3, pp. 52–74, 2017.

[42] F. Spitzer, *Principles of random walk*. Springer Science & Business Media, 2013, vol. 34.

[43] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

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

[45] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, "Distributed large-scale natural graph factorization," in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 37–48.

[46] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *Proceedings of the 24th ACM international on conference on information and knowledge management*, 2015, pp. 891–900.

[47] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1105–1114.

[48] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, "Community preserving network embedding." in *AAAI*, vol. 17, 2017, pp. 203–209.

[49] M. E. Newman, "A measure of betweenness centrality based on random walks," *Social networks*, vol. 27, no. 1, pp. 39–54, 2005.

[50] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens, "Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation," *IEEE Transactions on knowledge and data engineering*, vol. 19, no. 3, pp. 355–369, 2007.

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

[52] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *International conference on ma-*

chine learning. PMLR, 2016, pp. 40–48.

[53] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.

[54] J. Tang, M. Qu, and Q. Mei, "Pte: Predictive text embedding through large-scale heterogeneous text networks," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015, pp. 1165–1174.

[55] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[56] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[57] H. Chen, B. Perozzi, Y. Hu, and S. Skiena, "HARP: hierarchical representation learning for networks," in *AAAI*. AAAI Press, 2018, pp. 2127–2134.

[58] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 2018, pp. 459–467.

[59] Z. Qiu, E. Cho, X. Ma, and W. M. Campbell, "Graph-based semi-supervised learning for natural language understanding," in *TextGraphs@EMNLP*. Association for Computational Linguistics, 2019, pp. 151–158.

[60] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

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

[62] S. Cao, W. Lu, and Q. Xu, "Deep neural networks for learning graph representations." in *AAAI*, vol. 16, 2016, pp. 1145–1152.

[63] A. Taheri, K. Gimpel, and T. Berger-Wolf, "Learning graph representations with recurrent neural network autoencoders," *KDD Deep Learning Day*, 2018.

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

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

[66] S. Pan, R. Hu, S.-f. Fung, G. Long, J. Jiang, and C. Zhang, "Learning graph embedding with adversarial training methods," *IEEE Transactions on Cybernetics*, vol. 50, no. 6, pp. 2475–2487, 2019.

[67] C. Wang, S. Pan, G. Long, X. Zhu, and J. Jiang, "Mgae: Marginalized graph autoencoder for graph clustering," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 889–898.

[68] T. Ma, J. Chen, and C. Xiao, "Constrained generation of semantically valid graphs via regularizing variational autoencoders," in *Advances in Neural Information Processing Systems*, 2018, pp. 7113–7124.

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

[70] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR (Poster)*. OpenReview.net, 2017.

[71] P.-Y. Huang, R. Frederking *et al.*, "Rwr-gae: Random walk regularization for graph auto encoders," *arXiv preprint arXiv:1908.04003*, 2019.

[72] J. Li, J. Yu, J. Li, H. Zhang, K. Zhao, Y. Rong, H. Cheng, and J. Huang, "Dirichlet graph variational autoencoder," in *NeurIPS*, 2020.

[73] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[74] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[75] S. Abu-El-Haija, B. Perozzi, A. Kapoor, N. Alipourfard, K. Lerman, H. Harutyunyan, G. V. Steeg, and A. Galstyan, "Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 21–29.

[76] F. Wu, A. H. S. Jr., T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 6861–6871.

[77] K. Zhou, X. Huang, Y. Li, D. Zha, R. Chen, and X. Hu, "Towards deeper graph neural networks with differentiable group normalization," in *NeurIPS*, 2020.

[78] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[79] R. L. Murphy, B. Srinivasan, V. A. Rao, and B. Ribeiro, "Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs," in *ICLR (Poster)*. OpenReview.net, 2019.

[80] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR (Poster)*. OpenReview.net, 2018.

[81] K. K. Thekumparampil, C. Wang, S. Oh, and L.-J. Li, "Attention-based graph neural network for semi-supervised learning," *arXiv preprint arXiv:1803.03735*, 2018.

[82] T. Pham, T. Tran, D. Q. Phung, and S. Venkatesh, "Column networks for collective classification," in *AAAI*. AAAI Press, 2017, pp. 2485–2491.

[83] Y. Min, F. Wenkel, and G. Wolf, "Scattering GCN: overcoming oversmoothness in graph convolutional networks," in *NeurIPS*, 2020.

[84] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[85] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," in *ICLR*. OpenReview.net, 2020.

[86] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," in *ICLR (Poster)*, 2016.

[87] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill, "Learning a SAT solver from single-bit supervision," in *ICLR (Poster)*. OpenReview.net, 2019.

[88] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 2018, pp. 5449–5458.

[89] F. Sun, J. Hoffmann, V. Verma, and J. Tang, "Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization," in *ICLR*. OpenReview.net, 2020.

[90] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 1263–1272.

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

[92] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, 2016, pp. 3844–3852.

[93] Q. Li, Z. Han, and X. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *AAAI*. AAAI Press, 2018, pp. 3538–3545.

[94] B. Jiang, Z. Zhang, D. Lin, J. Tang, and B. Luo, "Semi-supervised learning with graph learning-convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 313–11 320.

[95] H. Yang, K. Ma, and J. Cheng, "Rethinking graph regularization for graph neural networks," *arXiv preprint arXiv:2009.02027*, 2020.

[96] S. Abu-El-Haija, A. Kapoor, B. Perozzi, and J. Lee, "N-gcn: Multi-scale graph convolution for semi-supervised node classification," in *Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 841–851.

[97] C. Xu, Z. Cui, X. Hong, T. Zhang, J. Yang, and W. Liu, "Graph inference learning for semi-supervised classification," in *ICLR*. OpenReview.net, 2020.

[98] R. Liao, M. Brockschmidt, D. Tarlow, A. L. Gaunt, R. Urtasun, and R. S. Zemel, "Graph partition neural networks for semi-supervised classification," in *ICLR (Workshop)*. OpenReview.net, 2018.

[99] Y. Zhang, S. Pal, M. Coates, and D. Ustebay, "Bayesian graph convolutional neural networks for semi-supervised classification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 5829–5836.

[100] S. Vashishth, P. Yadav, M. Bhandari, and P. P. Talukdar, "Confidence-based graph convolutional networks for semi-supervised learning," in *AISTATS*, ser. Proceedings of Machine Learning Research, vol. 89. PMLR, 2019, pp. 1792–1801.

[101] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," in *WWW*. ACM, 2018, pp. 499–508.

[102] S. Wan, S. Pan, J. Yang, and C. Gong, "Contrastive and generative graph convolutional networks for graph-based semi-supervised learning," *arXiv preprint arXiv:2009.07111*, 2020.

[103] M. T. Kejani, F. Dornaika, and H. Talebi, "Graph convolution networks with manifold regularization for semi-supervised learning," *Neural Networks*, 2020.

[104] B. Xu, H. Shen, Q. Cao, K. Cen, and X. Cheng, "Graph convolutional networks using heat kernel for semi-supervised learning," *arXiv preprint arXiv:2007.16002*, 2020.

[105] F. Hu, Y. Zhu, S. Wu, L. Wang, and T. Tan, "Hierarchical graph convolutional networks for semi-supervised node classification," in *IJCAI*. ijcai.org, 2019, pp. 4532–4539.

[106] K. Sun, Z. Lin, and Z. Zhu, "Adagcn: Adaboosting graph convolutional networks into deep models," *CoRR*, vol. abs/1908.05081, 2019.

[107] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Advances in neural information processing systems*, 2017, pp. 3391–3401.

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

[109] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[110] Y. Shao, N. Sang, C. Gao, and L. Ma, "Probabilistic class structure regularized sparse representation graph for semi-supervised hyperspectral image classification," *Pattern Recognit.*, vol. 63, pp. 102–114, 2017.

[111] ——, "Spatial and class structure regularized sparse representation graph for semi-supervised hyperspectral image classification," *Pattern Recognit.*, vol. 81, pp. 81–94, 2018.

[112] F. He, R. Wang, and W. Jia, "Fast semi-supervised learning with anchor graph for large hyperspectral images," *Pattern Recognition Letters*, vol. 130, pp. 319–326, Feb. 2020.

[113] Y. Chen, Z. Lai, Y. Ding, K. Lin, and W. K. Wong, "Deep supervised hashing with anchor graph," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 9796–9804.

[114] D. C. G. Pedronette, Y. Weng, A. Baldassin, and C. Hou, "Semi-supervised and active learning through Manifold Reciprocal kNN Graph for image retrieval," *Neurocomputing*, vol. 340, pp. 19–31, May 2019.

[115] X. Shi, H. Su, F. Xing, Y. Liang, G. Qu, and L. Yang, "Graph temporal ensembling based semi-supervised convolutional neural network with noisy labels for histopathology image analysis," *Medical Image Analysis*, vol. 60, p. 101624, Feb. 2020.

[116] G. He, X. Liu, F. Fan, and J. You, "Classification-aware semi-supervised domain adaptation," in *CVPR Workshops*. IEEE, 2020, pp. 4147–4156.

[117] A. Subramanya, S. Petrov, and F. C. N. Pereira, "Efficient graph-based semi-supervised learning of structured tagging models," in *EMNLP*. ACL, 2010, pp. 167–176.

[118] M. Aliannejadi, M. Kiaeeha, S. Khadivi, and S. S. Ghidary, "Graph-based semi-supervised conditional random fields for spoken language understanding using unaligned data," in *Proceedings of the Australasian Language Technology Association Workshop 2014*, Melbourne, Australia, Nov. 2014, pp. 98–103.

[119] Q. Mei, D. Zhang, and C. Zhai, "A general optimization framework for smoothing language models on graph structures," in *SIGIR*. ACM, 2008, pp. 611–618.

[120] L. Hu, T. Yang, C. Shi, H. Ji, and X. Li, "Heterogeneous graph attention networks for semi-supervised short text classification," in *EMNLP/IJCNLP (1)*. Association for Computational Linguistics, 2019, pp. 4820–4829.

[121] F. Alam, S. R. Joty, and M. Imran, "Graph based semi-supervised learning with convolution neural networks to classify crisis related tweets," in *ICWSM*. AAAI Press, 2018, pp. 556–559.

[122] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *ICML*, ser. JMLR Workshop and Conference Proceedings, vol. 48. JMLR.org, 2016, pp. 40–48.

[123] M. Balaanand, N. Karthikeyan, S. Karthik, R. Varatharajan, G. Manogaran, and C. B. Sivaparthipan, "An enhanced graph-based semi-supervised learning algorithm to detect fake users on twitter," *J. Supercomput.*, vol. 75, no. 9, pp. 6085–6105, 2019.

[124] C. Yang, L. Bai, C. Zhang, Q. Yuan, and J. Han, "Bridging collaborative filtering and semi-supervised learning: A neural approach for POI recommendation," in *KDD*. ACM, 2017, pp. 1245–1254.

[125] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *WWW*. ACM, 2019, pp. 2022–2032.

[126] A. D. Torshizi and L. R. Petzold, "Graph-based semi-supervised learning with genomic data integration using condition-responsive genes applied to phenotype classification," *J. Am. Medical Informatics Assoc.*, vol. 25, no. 1, pp. 99–108, 2018.

[127] J. Luo, P. Ding, C. Liang, and X. Chen, "Semi-supervised prediction of human miRNA-disease association based on graph regularization framework in heterogeneous networks," *Neurocomputing*, vol. 294, pp. 29–38, Jun. 2018.

[128] X. Zeng, W. Wang, G. Deng, J. Bing, and Q. Zou, "Prediction of potential disease-associated micrornas by using neural networks," *Molecular Therapy-Nucleic Acids*, vol. 16, pp. 566–575, 2019.

[129] R. Lang, R. Lu, C. Zhao, H. Qin, and G. Liu, "Graph-based semi-supervised one class support vector machine for detecting abnormal lung sounds," *Applied Mathematics and Computation*, vol. 364, p. 124487, Jan. 2020.

[130] D. Mahapatra, "Semi-supervised learning and graph cuts for consensus based medical image segmentation," *Pattern Recognition*, vol. 63, pp. 700 – 709, 2017.

[131] Q. Liu, L. Yu, L. Luo, Q. Dou, P. A. Heng, and P. A. Heng, "Semi-supervised medical image classification with relation-driven self-ensembling model," *IEEE Transactions on Medical Imaging*, pp. 1–1, 2020.

[132] J. Ma, W. Tang, J. Zhu, and Q. Mei, "A flexible generative framework for graph-based semi-supervised learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 3281–3290.

[133] S. Pal, S. Malekmohammadi, F. Regol, Y. Zhang, Y. Xu, and M. Coates, "Non parametric graph learning for bayesian graph neural networks," in *Conference on Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 1318–1327.

[134] M. Esmaeili and A. Nosratinia, "New gcnn-based architecture for semi-supervised node classification," *arXiv preprint arXiv:2009.13734*, 2020.

[135] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang, "Graph random neural networks for semi-supervised learning on graphs," in *NeurIPS*, 2020.

[136] X. Zhao, F. Chen, S. Hu, and J. Cho, "Uncertainty aware semi-supervised learning on graph data," in *NeurIPS*, 2020.

[137] W. Liu, J. He, and S.-F. Chang, "Large graph construction for scalable semi-supervised learning," in *ICML*, 2010.

[138] W. Liu, J. Wang, and S.-F. Chang, "Robust and scalable graph-based semisupervised learning," *Proceedings of the IEEE*, vol. 100, no. 9, pp. 2624–2638, 2012.

[139] M. Wang, W. Fu, S. Hao, D. Tao, and X. Wu, "Scalable semi-supervised learning by efficient anchor graph regularization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 7, pp. 1864–1877, 2016.

[140] F. He, F. Nie, R. Wang, H. Hu, W. Jia, and X. Li, "Fast semi-supervised learning with optimal bipartite graph," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

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

[142] Z. Li, C. Li, L. Yang, S. Y. Philip, and Z. Li, "Mixture distribution modeling for scalable graph-based semi-supervised learning," *Knowledge-Based Systems*, p. 105974, 2020.

[143] M. Chen, Z. Wei, B. Ding, Y. Li, Y. Yuan, X. Du, and J. Wen, "Scalable graph neural networks via bidirectional propagation," in *NeurIPS*, 2020.

[144] O. Stretcu, K. Viswanathan, D. Movshovitz-Attias, E. Platanios, S. Ravi, and A. Tomkins, "Graph agreement models for semi-supervised learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 8713–8723.

[145] F. Zhou, T. Li, H. Zhou, H. Zhu, and Y. Jieping, "Graph-based semi-supervised learning with non-ignorable non-response," in *Advances in Neural Information Processing Systems*, 2019, pp. 7015–7025.

[146] B. K. de Aquino Afonso and L. Berton, "Analysis of label noise in graph-based semi-supervised learning," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 1127–1134.

[147] M. M. Dunlop, D. Slepčev, A. M. Stuart, and M. Thorpe, "Large data and zero noise limits of graph-based semi-supervised learning algorithms," *Applied and Computational Harmonic Analysis*, vol. 49, no. 2, pp. 655–697, 2020.

[148] X. Liu, S. Si, J. Zhu, Y. Li, and C. Hsieh, "A unified framework for data poisoning attack to graph-based semi-supervised learning," in *NeurIPS*, 2019, pp. 9777–9787.

[149] P. Liao, H. Zhao, K. Xu, T. Jaakkola, G. Gordon, S. Jegelka, and R. Salakhutdinov, "Graph adversarial networks: Protecting information against adversarial attacks," *arXiv preprint arXiv:2009.13504*, 2020.

[150] X. Zhang and M. Zitnik, "Gnnguard: Defending graph neural networks against adversarial attacks," in *NeurIPS*, 2020.

[151] H. Gan, Z. Li, W. Wu, Z. Luo, and R. Huang, "Safety-aware graph-based semi-supervised learning," *Expert Systems with Applications*, vol. 107, pp. 243–254, 2018.

[152] X. Gao, W. Hu, and Z. Guo, "Exploring structure-adaptive graph learning for robust semi-supervised classification," in *ICME*. IEEE, 2020, pp. 1–6.

[153] P. Elinas, E. V. Bonilla, and L. C. Tiao, "Variational inference for graph convolutional networks in the absence of graph data and adversarial settings," in *NeurIPS*, 2020.

[154] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[155] B. Perozzi, V. Kulkarni, H. Chen, and S. Skiena, "Don't walk, skip! online learning of multi-scale network embeddings," in *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, 2017, pp. 258–265.

[156] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *arXiv preprint arXiv:1811.05868*, 2018.

[157] W. Wang, X. Liu, P. Jiao, X. Chen, and D. Jin, "A unified weakly supervised framework for community detection and semantic matching," in *Advances in Knowledge Discovery and Data Mining*, D. Phung, V. S. Tseng, G. I. Webb, B. Ho, M. Ganji, and L. Rashidi, Eds. Cham: Springer International Publishing, 2018, pp. 218–230.

[158] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[159] L. Tang and H. Liu, "Relational learning via latent social dimensions," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 817–826.

[160] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data. Jun. 2014.

[161] M. Mahoney, "Large text compression benchmark," 2011.

[162] M. Zitnik and J. Leskovec, "Predicting multicellular function through multi-layer tissue networks," *Bioinformatics*, vol. 33, no. 14, pp. i190–i198, 2017.

[163] N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," *Knowl. Inf. Syst.*, vol. 14, no. 3, p. 347–375, Mar. 2008.

[164] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of medicinal chemistry*, vol. 34, no. 2, pp. 786–797, 1991.

[165] P. D. Dobson and A. J. Doig, "Distinguishing enzyme structures from non-enzymes without alignments," *Journal of molecular biology*, vol. 330, no. 4, pp. 771–783, 2003.

[166] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," *Bioinformatics*, vol. 21, no. suppl_1, pp. i47–i56, 2005.

[167] H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, and C. Helma, "Statistical evaluation of the predictive toxicology challenge 2000–2001," *Bioinformatics*, vol. 19, no. 10, pp. 1183–1193, 2003.

[168] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell, "Toward an architecture for never-ending language learning," in *AAAI*. AAAI Press, 2010.

[169] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[170] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *ICML*. Omnipress, 2011, pp. 689–696.

[171] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on cifar-10," *Unpublished manuscript*, vol. 40, no. 7, pp. 1–9, 2010.