Network Representation Learning: A Macro and Micro View

Xueyi Liu and Jie Tang

Abstract—Graph is a universe data structure that is widely used to organize data in real-world. Various real-word networks like the transportation network, social and academic network can be represented by graphs. Recent years have witnessed the quick development on representing vertices in the network into a low-dimensional vector space, referred to as network representation learning. Representation learning can facilitate the design of new algorithms on the graph data. In this survey, we conduct a comprehensive review of current literature on network representation learning. Existing algorithms can be categorized into three groups: shallow embedding models, heterogeneous network embedding models, graph neural network based models. We review state-of-the-art algorithms for each category and discuss the essential differences between these algorithms. One advantage of the survey is that we systematically study the underlying theoretical foundations underlying the different categories of algorithms, which offers deep insights for better understanding the development of the network representation learning field.

Index Terms—Network Representation Learning, Graph Neural Networks, Graph Spectral Theory

1 Introduction

Graph is a highly expressive data structure, based on which various networks exist in the real-world, like the social networks [1], [86], citation networks [109], biological networks [80], chemistry networks [81], traffic networks, and others. Mining information from real-world networks plays a crucial role in many emerging applications. For example, in social networks, classifying people into social communities according to their profile and social connections is useful for many related task, like social recommendation, target advertising, user search [162], etc. In communication networks, detecting community structures can help understand information diffusion. In biological networks, predicting the role of protein can help us reveal the mysteries of life; predicting molecular drugability can promote new drug development. In chemistry networks, predicting the function of molecules can help with the synthesis of new compound and new material. The way in which networks are generally represented cannot supply effective analysis. For example, the only structural information we can get from an adjacency matrix about one node is just its neighbours and the weight of the edges between them. It is not informative enough with respect to the neighbourhood structure and its role in the graph, and also of high space complexity (i.e., $O(N)$ for one node, where N is the number of nodes in the network). It is also hard to design an efficient algorithm based just on the adjacency matrix. Taking community detection as an example, most existing algorithms will involve calculating the spectral decomposition of a matrix [79], whose time complexity is always at least quadratic with respect to the number of vertices. Existing graph analytical methods, like distributed graph data processing framework (e.g., GraphX [40], and GraphLab [77]) suffer from high computational cost and high space complexity. This complexity makes the algorithms hard to be applied to large-scale networks with millions of vertices.

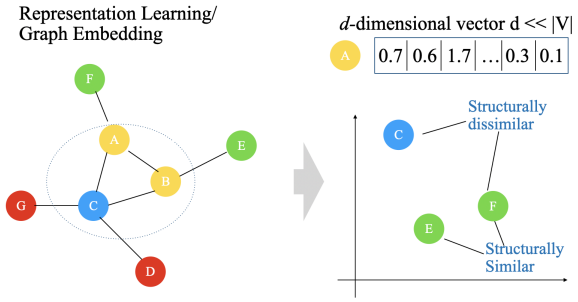

Fig. 1: A toy example for network embedding task. Vertices in the network lying in the left part are embedded into d-dimensional vector space, where d is much smaller than the total number of nodes $|V|$ in the network. Vertices with the same color are structurally similar to each other. Basic structural information should be kept in the embedding space (e.g., Structurally similar vertices E and F are embedded closer to each other than structurally dissimilar vertices C and F).

Recent years have seen the rapid development of network representation learning algorithms. Their purpose is to learn latent, informative and low-dimensional representations for network vertices, which can preserve the network structure, vertex features, labels and other auxiliary information [13], [162], as Fig. 1 illustrates. The vertex representations can help design efficient algorithms since various vector based machine learning algorithms can thus be easily applied to vertex representation vectors.

- *Xueyi Liu is with the Department of Computer Science and Technology, Tsinghua University, Beijing, China.*
  E-mail: xueyi-li18@mails.tsinghua.edu.cn
- *Jie Tang is with the Department of Computer Science and Technology, Tsinghua University, and Tsinghua National Laboratory for Information Science and Technology (TNList), Beijing, China, 100084.*
  E-mail: jietang@tsinghua.edu.cn, corresponding author.

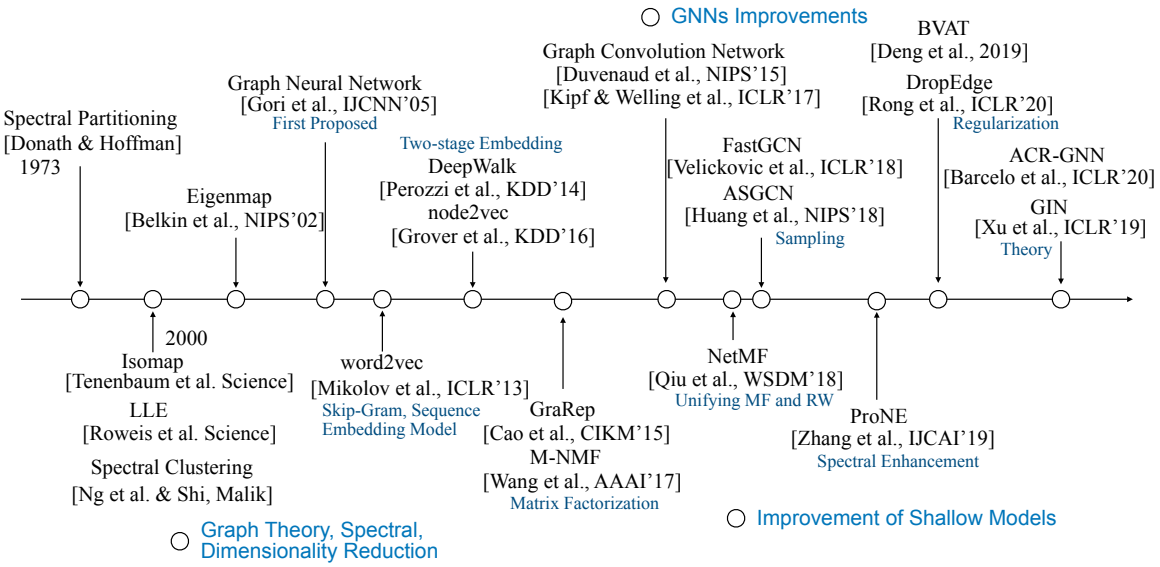

Fig. 2: A brief summary of the development of network embedding techniques. Left Channel: Shallow (Heterogeneous) Neural Embedding Models; Mid Channel: Matrix Factorization Based Models; Right Channel: Graph Neural Network Based Models.

Such works date back to the early 2000s [162], when the proposed algorithms were part of dimensionality-reduction techniques (e.g., Isomap [123], LLE [106], Eigenmap [7], and MFA [150]). These algorithms firstly calculate the affinity graph (e.g., k-nearest-neighbour graph) for the input of high-dimensional data. Then, the affinity graph is embedded into a lower dimensional space. However, the time complexity of those methods is too high to scale to large networks. Later on, there is an emerging number of works [15], [92], [159] focusing on developing efficient and effective embedding method to assign each node a low dimensional representation vector that is aware of structural information, vertex content and other information. Many efficient machine learning models can be designed for downstream tasks based on the learned vertex representations, like node classification [10], [172], link prediction [38], [72], [78], recommendation [146], [158], similarity search [74], visualization [118], clustering [79], and knowledge graph search [73]. Fig. 2 shows a brief summary of the development history of graph embedding models.

In this survey, we provide a comprehensive up-to-date review of network representation learning algorithms, aiming to give readers a macro, covering the some common basic insights under different kinds of embedding algorithms and the relationship between them, as well as a micro, lacking no details of different algorithms and also theories behind them, view on previous effort and achievements in this area. We group existing graph embedding methods into three major categories based on the development dependencies among those algorithms, from *shallow embedding models*, whose objects are basic homogeneous graphs (Def. 1) with only one type of nodes and edges[1], to *heterogeneous embedding models*, most of whose basic ideas are inherited from shallow embedding models designed for homogeneous graphs with the range of graph objects expanded to heterogeneous graphs with more than one types of nodes or edges and also often node or edge features, then further to *graph neural network based models*, many of whose insights are able to be found in shallow embedding models and heterogeneous embedding models, like the inductive learning and neighbourhood aggregation [17], spectral propagation [33], [163], and so on. Though it is hard to say the ideas of which methods are inspired by whose thoughts, the similarity and connections between them can help us understand them better and also always offer some interesting rethinking of the common field they belong to, which are also what the survey focuses on beyond reviewing existing graph embedding techniques.

Table 1 lists some typical graph embedding models and some of their related information, which can help readers get a fast glimpse of existing graph embedding models, their inner mechanisms and underlying relations. *Shallow embedding models* can be roughly grouped into two main categories, shallow neural embedding models and matrix factorization based models. Shallow neural embedding models (S-N) are characterized by embedding look-up tables, which are updated to preserve various proximities lying in the graph. Typical models are DeepWalk [92], node2vec [42], LINE [120] and so on. Matrix factorization (S-MF) based models aim to factorize matrices related with graph structure and other side information to get high-quality node representation vectors. Based on shallow embedding models designed for homogeneous networks, embedding techniques (e.g. PTE [119], metapath2vec [31], GATNE [17]), are designed for heterogeneous networks and we refer these models to *heterogeneous (SH) embedding models*. Different from shallow embedding models, *graph neural networks (GNNs)* are kind of techniques characterized by deep architectures to extract meaningful structural information into node representation

1. The word "homogeneous" is omitted in the category name "shallow embedding models" for brevity.

vectors. In addition to the discussion of the above-mentioned types of models, we also focus on their inner connections, advantages and disadvantages, optimization methods and some related theoretical foundations.

Finally, we summarize some existing challenges and propose possible development directions that can help with further design. We organize the survey as follows. In Section 2, we first summarize some useful definitions which can help readers understand the basic concepts, and then propose our taxonomy for the existing embedding algorithms. Then, in Section 4, 5, and 6, we review typical embedding methods falling into those three categories. In Section 7 and 8 we discuss some relationships within those algorithms of different categorizes and related optimization methods. We then go further to discuss some problems and challenges of existing graph embedding models in Section 9. At last, we discuss some further development directions for network representation learning in Section 10.

2 Preliminaries

We summarize related definitions as follows to help readers understand the algorithms discussed in the following parts.

First we introduce the definition of a graph, which is the basic data structure of real-world networks:

***Definition* 1** (Graph)**.** A graph can be denoted as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges in the graph. When associated with the node type mapping function $\Phi : \mathcal{V} \rightarrow \mathcal{O}$ mapping each node to its specific node type and an edge mapping function $\Psi : \mathcal{E} \rightarrow \mathcal{R}$ mapping each edge to its corresponding edge type, a graph G can be divided into two categories: *homogeneous graph* and *heterogeneous graph*. A homogeneous graph is a graph G with only one node type and one edge type (i.e., $|\mathcal{O}| = 1$ and $|\mathcal{R}| = 1$). A graph is a heterogeneous graph when $|\mathcal{O}| + |\mathcal{R}| > 2$.

Graphs are basic data structure for many kinds of real-world networks, like transportation network [102], social networks, academic networks [104], [152], and so on. They can be modeled by homogeneous graphs or heterogeneous graphs, based on the knowledge we have on nodes and edges in those networks. In the survey, we use *graph embedding* and *network representation learning* alternatively, both of which are high-frequency terms appeared in the literature [17], [42], [92], [153], [163] and both denote the process of generating representative vectors of a finite dimension for nodes in a graph or a network. When we use the term *graph embedding*, we focus mainly on the basic graph models, where we simply care about nodes and edges in the graph, and when we use *network representation learning*, our focus is more on networks in real-world.

Since there is a large number of embedding algorithms based on modeling vertex proximities, we briefly summarize the proposed vertex similarities as follows [162]:

***Definition* 2** (Vertex Proximities)**.** Various vertex proximities can exist in real-world networks, like first-order proximity, second-order proximity and higher-order proximities. The first-order proximity can measure the direct connectivity between two nodes, which is usually defined as the weight of the edge between them. The second-order proximity between two vertices can be defined as the distance between the distributions of their neighbourhood [135]. Higher-order proximities between two vertices v and u can be defined as the k-step transition probability from vertex v to vertex u [162].

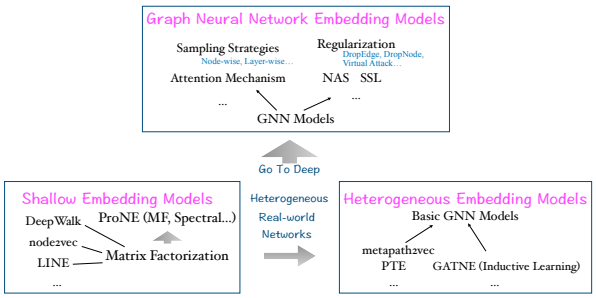

Fig. 3: An overview of existing graph embedding models and their correlation.

***Definition* 3** (Structural Similarity)**.** Structural similarity [33], [50], [75], [94], [102] refers to the similarity of the structural roles of two vertices in their respective communities, although they may not connect with each other.

***Definition* 4** (Intra-community Similarity)**.** The intra-community similarity originates from the community structure of the graph and denotes the similarity between two vertices that are in the same community. Many real-life networks (e.g., social networks, citation networks) have community structure, where vertex-vertex connections in a community are dense, but sparse for nodes between two communities.

Since graph Laplacian matrices are based for understanding embedding algorithms based on graph spectral, or adopt the graph spectral way, which is also a crucial development direction for embedding algorithms, we briefly introduce them as follows:

***Definition* 5** (Graph Laplacian)**.** Following notions in [51], $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$, where $\boldsymbol{A}$ is the adjacency matrix, D is the corresponding degree matrix, is the *combinational graph laplacian*, $\boldsymbol{\mathcal{L}} = \boldsymbol{I} - \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$ is the *normalized graph Laplacian*, $\boldsymbol{L}_{rw} = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{A}$ is the *random walk graph Laplacian*. Meanwhile, let $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \sigma\boldsymbol{I}$ denotes the augmented adjacency matrix, then, $\tilde{\boldsymbol{L}}, \tilde{\boldsymbol{\mathcal{L}}}, \tilde{\boldsymbol{L}}_{rw}$ are the *augmented graph Laplacian*, *augmented normalized graph Laplacian*, *augmented random walk graph Laplacian* respectively.

3 Overview of Graph Embedding Techniques

In this section, we will give graph embedding techniques of each category a brief introduction to help readers get a better understanding of the overall architecture of this paper. Fig. 3 shows a panoramic view of existing embedding models and their connections.

Shallow Embedding Models. These models can be divided into two main streams: shallow neural embedding models and matrix factorization based models. Though there are

Model	Type	Neural	Heter.	A/L	O-S	Proximity	Matrix	Filter
DeepWalk [92]	S-N	✓	×	×	SGNS,G	H-O	TABLE 2 DeepWalk	$h(\lambda)=\frac{1}{T}\sum_{r=1}^{T}\lambda^r$
node2vec [42]		✓	×	×	SGNS,G		TABLE 2 node2vec	-
Diff2vec [107]		✓	×	×	PS		-	-
Walklets [93]		✓	×	×	SGNS,G		-	-
Rol2Vec [2]		✓	×	A	SGNS,G	I-C	-	-
LINE [120]		✓	×	×	PS,NS,D,G	F-O,S-O	TABLE 2 LINE	$h(\lambda)=1-\lambda$
pRBM [138]		✓	×	A	PS,G	F-O	-	-
UPP-SNE [161]		✓	×	A	SGNS,G	H-O	-	-
DDRW [70]		✓	×	L	SGNS,SVM D,G	H-O	TABLE 2 DeepWalk	$h(\lambda)=\frac{1}{T}\sum_{r=1}^{T}\lambda^r$
TLINE [165]		✓	×	L	PS,NS,SVM D,G	F-O S-O	TABLE 2 LINE	$h(\lambda)=1-\lambda$
GraphGAN [137]		✓	×	×	G,D	F-O	-	-
struct2vec [102]		✓	×	×	SGNS,D	ST	-	-
PTE [119]	SH	✓	✓	×	PS,NS,G	S-O	Eq. 13	-
metapath2vec [31]		✓	✓	×	SGNS	H-O	-	-
HIN2vec [37]		✓	✓	×	SGNS	H-O	-	-
GATNE [17]		✓	✓	A	SGNS	H-O	-	-
HERec [111]		✓	✓	L	SGNS MF	H-O	user rating matrix $\boldsymbol{R}$	-
HueRec [142]		✓	✓	L	SGNS	H-O	-	-
HeGAN [54]		✓	✓	×	G,D	F-O	-	-
M-NMF [139]	S-MF	×	×	×	Iter-Update	F-O,S-O, I-C	$\boldsymbol{S}=\boldsymbol{S}^{(1)}+\eta\boldsymbol{S}^{(2)}$, $\boldsymbol{H}$	-
NetMF [97]		×	×	×	tSVD	H-O	TABLE 2 DeepWalk	$h(\lambda)=\frac{1}{T}\sum_{r=1}^{T}\lambda^r$
ProNE [163]		×	×	×	r-tSVD	F-O	Eq. 5	-
GraRep [14]		×	×	×	tSVD	H-O	Eq. 6	-
HOPE [88]		×	×	×	JDGSVD [52]	H-O	General	-
TADW [151]		×	×	A	I-MF	H-O(without homophily)	k-step transition matrix $\boldsymbol{M}$	-
HSCA [160]		×	×	A	Iter-Update	H-O	k-step transition matrix $\boldsymbol{M}$	-
ProNE [163]	S-SS	×	×	×	Spectral Propagation	H-O	$\boldsymbol{I}_N-\boldsymbol{U}g(\Lambda)\boldsymbol{U}^{-1}$	$g(\lambda)=e^{-\frac{1}{2}[(\lambda-\mu)^2-1]\theta}$
GraphZoom [28]		×	×	A		H-O	$(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}})^k$	$h(\lambda)=(1-\lambda)^k$
GraphWave [33]		×	×	×		ST	-	$g_s(\lambda)=e^{-\lambda s}$
GCN [65]	GNN	✓	×	A,L	SGD	H-O	$\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$	$h(\lambda)=1-\lambda$
GraphSAGE [45]		✓	×	A,L	SGD	H-O	Depend on A-M	-
FastGCN [19]		✓	×	A,L	SGD	H-O	$\tilde{\boldsymbol{A}}Q(Q_{ii}=\frac{1}{q_{v_i}^{(l)}})$	$h(\lambda)=1-\lambda$
ASGCN [59]		✓	×	A,L	SGD	H-O	$\tilde{\boldsymbol{A}}\boldsymbol{Q}(Q_{ii}=\frac{q_i}{q*_{v_i}})$	$h(\lambda)=1-\lambda$
GAT [130]		✓	×	A,L	SGD	H-O	$\boldsymbol{P}\tilde{\boldsymbol{A}}+\tilde{\boldsymbol{A}}\boldsymbol{Q}$ [5]	-
GIN [148]		✓	×	A,L	SGD	H-O	$\tilde{\boldsymbol{A}}=\boldsymbol{A}+\boldsymbol{I}_N$	-
gfNN [51]		✓	×	A,L	SGD	H-O	$(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}})^k$	$h(\lambda)=(1-\lambda)^k$
SGC [145]		✓	×	A,L	SGD	H-O	$(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}})^k$	$h(\lambda)=(1-\lambda)^k$
ACR-GNN [6]		✓	×	A,L	SGD	H-O	-	-
RGCN [170]		✓	×	A,L	SGD	H-O	$\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}$	-
BVAT [29]		✓	×	A,L	Adversarial, SGD	H-O	-	-
DropEdge [105]		✓	×	A,L	SGD	H-O	$\hat{\boldsymbol{A}}_{\text{drop}}=\mathcal{N}(\boldsymbol{A}-\boldsymbol{A}')$ [105]	-
R-GCN [108]		✓	✓	A	SGD	H-O	-	-
HetGNN [157]		✓	✓	A	SGNS	H-O	-	-
GraLSP [62]		✓	✓	A	SGNS	H-O,ST	-	-

TABLE 1: An overview of network representation learning algorithms (selected). Symbols in some formulas can refer to Def. 5. For others, "A" $\sim$ w/o vertex attributes;"L" $\sim$ w/o vertex labels; "Heter." $\sim$ heterogeneous networks. Abbreviations used: "F-O", "S-O", "H-O", "I-C", "ST" refer to First-Order, Second-Order, High-Order, Intra-Community and Structural similarities; "SN", "SHN", "MF", "SS" refer to Shallow Neural Embedding models, Shallow Heterogeneous Network Embedding Model, Matrix Factorization Based models and Shallow Spectral models; "O-S" denotes optimization strategies, in which "PS", "NS" refer to positive sampling and negative sampling; "(r)-(t)SVD" refers to (randomized)-(truncated) singular value decomposition; "SGNS" refers to "Skip-Gram with Negative Sampling"; "Iter-Update" refers to iteratively updating; "I-MF" refers to inductive matrix factorization [87]; "G" $\sim$ generative method; "D" $\sim$ discriminative methods; "A-M" $\sim$ aggregation methods.

some differences between those two embedding genres, it has been shown that some shallow embedding based models, especially those adopt random walk to sample vertex sequences and perform skip-gram model to get vertex embeddings, have close connections with matrix factorization models: they are actually implicitly factorizing their equivalent matrices [97], to be specific.

Besides, matrices being factorized by shallow embedding models also have close relationship with graph spectral theories. Apart from models like GraphWave [33] which are based on graph spectral directly (see Sec. 4.4), other models like DeepWalk [92], node2vec [42], LINE [120] can also be proved to have close relationship with graph spectral by proving that their equivalent matrices are filter matrices [97].

Then, the explicit combination of traditional shallow embedding methods like matrix factorization and spectral embedding models, can be seen in the embedding model ProNE [163], where vertex embeddings are firstly obtained by factorizing a sparse matrix and then propagated by band-pass filter matrix in the spectral domain. Moreover, such close connections can also be seen int the university of spectral propagation technique proposed in ProNE, which is proved to be a universal embedding enhancement method, improving the quality of vertex embeddings obtained by other shallow embedding models effectively [163].

Such associations enable some basic ideas of those shallow embedding models can be regarded as the basis of GNN models.

Heterogeneous Embedding Models. Based on shallow embedding models, many embedding models for heterogeneous networks can be developed by some techniques, like metapath2vec [31], which applies certainty constrictions on the random sampling process and PTE [119] which splits the heterogeneous graph into several homogeneous graphs.

Moreover, various graph content in heterogeneous models, like vertex and edge features and labels evokes the thoughts on how to effectively utilize graph content in the embedding process and also how to become inductive when being applied on dynamic graphs, which is a common feature of real-world graphs. For example, the proposed embedding model GATNE [17] applies attention mechanism on vertex features during the embedding process, and try to learn the transformation function applied on vertex contents to make the model become inductive (GATNE-I).

Such design ideas can be seen as basic models for Graph Neural Networks.

Graph Neural Networks. Different from above mentioned shallow embedding models, Graph Neural Networks (GNNs) are some kind of deep, inductive embedding models, which can utilize graph contents better and can also be trained with supervised information. The basic idea of GNNs is iteratively aggregating neighbourhood information from vertex neighbours to get a successive view over the whole graph structure.

Based on vanilla GNN models, there is a huge amount of works focusing on developing enhancement techniques [35], [53], [59], [105] to improve the efficiency and effectiveness of GNN models.

Despite the advantages of GNN models, there are also many problems lying in GNN architecture, with also methods proposed to solve such problems, most of which focus on graph regularisation [29], [133], basic theories [6], self-supervised learning [57], [95], architecture search [169] and so on.

4 Shallow Embedding Models

4.1 Neural Based

There is a kind of model that is characterized by looking-up embedding tables containing node embeddings as row or column vectors, which are treated as parameters and can be updated during the training process. There are many approaches for updating vertex embeddings. Some extract vertex-context pairs by performing random walks on the graph (e.g., DeepWalk [92], node2vec [42]). They tend to maximize the log-likelihood of observing context vertices for the given target node. These methods are treated as generative models in [137]. In generative models, it is assumed that there existing a true connectivity distribution $p_{true}(\cdot|v)$ for each node v and the graph is generated by the connectivity distribution. The co-occurrence frequencies for the vertex-context pairs are then treated as the observed empirical distributions for the underlying connectivity distribution. Some try to model edges directly through the similarities between vertex embeddings of each connected pair (e.g., first-order proximity in LINE [120]) or training a discriminative model (or a classifier) to predict their existence.

In this section, we will review a large class of methods based on random walk, make comparisons between different random walk strategies and examine some models adopting other methods.

4.1.1 Random Walk Family

Random walk and its variants are kind of effective methods transferring the sub-linear structure of graph to the linear structure (i.e.,node sequences), since the generated walks can well preserve structural information of the original graph [76], [82].

Random walk strategy is firstly used to generate node sequences in DeepWalk [92], and we refer to the proposed random walk technique as *Vanilla Random Walk*. It can be seen as a Markov process on the graph and has been well studied [76]. Readers can refer to [76] for more details. After node sequences are generated, the Skip-Gram model [83] is applied to extract positive vertex-context pairs from them. Based on the distributional hypothesis [47], the Skip-Gram model is first proposed in [83] to capture semantic similarity between words in natural language. It is then generalized to networks based on the hypothesis that vertices that share similar structural contexts tend to be close in the embedding space.

Development of Random Walks. Based on the vanilla random walk, which is proposed in DeepWalk [92] and has achieved the state-of-the-art performance at that time when applied to downstream tasks (e.g., multi-label node classification), the biased random walk is proposed in [42] by introducing a return parameter p, and a in-out parameter q in the calculation for the transition probability at each step (Fig. 4 Right Channel). Thus, it is also the second-order random walk, whose transition probability also depends on

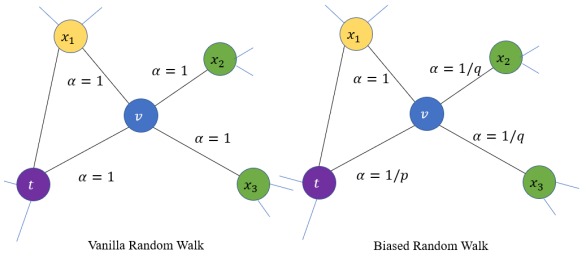

Fig. 4: An illustration for the transition probabilities in vanilla and biased random walk. Right Panel: Assuming the previous node is t and the current node is v, then $\alpha_{pa}(v,x)$ for node x_1, x_2, x_3 depend on their distances from the previous node t. The transition probability from current node v to node x is calculated by $\pi_{vx} = \alpha_{pq}(t,x) \cdot w_{vx}$, where w_{vx} is the weight of edge (v,x). Left Panel: The vanilla random walk can be regarded as a special case of the biased random walk, where $p = q = 1$. Adapted from [42].

the previous node. Euler walk is proposed in Diff2Vec [107], which perform a euler tour in the diffusion subgraph centered at each node. Walklets is proposed to separated mixed node proximities information from each order in [93]. Thus, it can get embeddings with successively coarser node proximity information preserved as the order k increases. Besides, attribute random walk is proposed in Rol2Vec [2] to design a kind of random walk that can incorporate vertex attributes and structural information.

Comparison and Discussion. Compared with the vanilla random walk, the introduced parameters p and q can help the biased random walk interpolate smoothly between DFS and BFS [24]. Thus, the biased random walk can explore various node proximities that may exist in the real-world network (e.g., second-order similarity, structural equivalence). It can also fit in a new network more easily by changing parameters to change the preference of proximities being explored since different proximities may dominate in different networks [42]. But these two parameters will need tuning to fit in a new graph if there is no labeled data that can be used to learn them.

Both biased random walk and vanilla random walk need calculating transition probabilities for each adjacent node of the current node at each step, which is time-consuming [107]. Compared with them, Euler tour is easy to find in the subgraph [144]. It can also get a more comprehensive view over the neighbourhood since the Euler tour will include all the adjacencies in the subgraph. Thus, fewer diffusion subgraphs and fewer Euler walks need generating centered at each node, compared with vanilla random walks, which tend to revisit a vertex many times, thus producing redundant information [4], [107]. However, the BFS strategy which is used to generate diffusion subgraphs is rather rigid, and cannot explore the various node proximities flexibly. Besides, the effectiveness of Diff2Vec is not well proved, since its performance in popular downstream tasks that are widely used in previous works (e.g., node classification and link prediction) [33], [42], [92], [97], [120], [163] have not been studied [107].

TABLE 2: Matrices that are implicitly factorized by DeepWalk, LINE and node2vec, same with [97]. "DW" refers to DeepWalk, "n2v" refers to node2vec.

Model	Matrix
DW	$\log\left(\mathrm{vol}(G)(\frac{1}{T}\sum_{r=1}^{T}(\boldsymbol{D}^{-1}\boldsymbol{A})^r)\boldsymbol{D}^{-1}\right) - \log b$
LINE	$\log\left(\mathrm{vol}(G)\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{D}^{-1}\right) - \log b$
n2v	$\log\left(\frac{\frac{1}{2T}\sum_{r=1}^{T}\left(\sum_u \boldsymbol{X}_{w,u}\underline{\boldsymbol{P}}^r_{c,w,u} + \sum_u \boldsymbol{X}_{c,u}\underline{\boldsymbol{P}}^r_{w,c,u}\right)}{(\sum_u \boldsymbol{X}_{w,u})(\sum_u \boldsymbol{X}_{c,u})}\right) - \log b$

4.1.2 Others Methods

Random walk based methods can be seen as kinds of generative models [137]. There are also other methods coming out of the random walk and Skip-Gram range, which can be seen as discriminative models, or as implicitly both generative and discriminative (e.g., LINE [120]), or as adversarial generative training method [41] (e.g., GraphGAN [137]).

In LINE, both the existence of edges and the connectivity distribution for each node are modeled, which can be seen as its discriminative and generative parts respectively. Existence of edges is modeled by maximizing following probability for each two connected node pair (v_i, v_j):

$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-\vec{u}_i^T \cdot \vec{u}_j)}, \tag{1}$$

where $\vec{u}_i$ is the vertex embedding for node v_i. The connectivity distribution for each node $p_2(\cdot|v_i)$(can be calculated by Eq. 3) is forced to be similar with the empirical distribution $\hat{p}_2(\cdot|v_i)$ by minimizing the following objective to model the second-order proximity:

$$O_2 = \sum_{i \in V} \lambda_i d(\hat{p}_2(\cdot|v_i), p_2(\cdot|v_i)), \tag{2}$$

where $d(\cdot,\cdot)$ is the distance between two distributions, λ_i is the weight for each node, which represents its prestige in the network and can be measured by vertex degree or other algorithms (e.g. PageRank [89]).

$$p_2(v_j|v_i) = \frac{\exp(\vec{u}_j'^T \cdot \vec{u}_i)}{\sum_{k=1}^{|V|}\exp(\vec{u}_k'^T \cdot \vec{u}_i)} \tag{3}$$

4.2 Matrix Factorization Based.

Matrix factorization is an effective method to get high-quality vertex embedding vectors.

Matrices to be factorized can be defined to preserve various node proximities, like the first-order, second-order and intra-community proximities preserved in M-NMF [139], the asymmetric high-order node proximity preserved in HOPE [88]. Or they can be defined as the matrix implicitly factorized by shallow neural embedding models discussed before, since some of these methods are proved to be inherently related to matrix factorization. It will be discussed in the next subsection.

Moreover, there are many techniques to factorize the matrix. In addition to making the factor matrices obtained by factorizing preserve the properties of the original matrix, time efficiency is also of great importance for matrix factorization methods. Detailed discussion can be seen in Section 8.2.

4.3 Connection between Neural Based and Matrix Factorization Based Models

Recent years have seen many works focusing on the exploring the equivalence between some of shallow neural embedding models and matrix factorization models by proving that some neural based models are factorizing matrices implicitly. Such connections can also help with the analysis of robustness of random walk based embedding models [11]. Moreover, it is empirically proved that embedding vectors obtained by factorizing the corresponding matrix can preform better in downstream tasks than those optimized by stochastic gradient descent in DeepWalk [97].

4.3.1 Matrices in Natural Language Models

This concern for equivalence does not originate from graph representation learning models. It is proposed in [69] that the word2vec model [84] or the SGNS procedure in it is implicitly factorizing the following word-context matrix:

$$M_{ij}^{SGNS} = \log\left(\frac{\#(w,c)\cdot|D|}{\#(w)\cdot\#(c)}\right) - \log(k), \tag{4}$$

where $\#(w,c)$ is the number of co-occurrence of the word pair (w,c) in the corpus, which is selected by sliding a certain length of window over word sequences, $\#(w)$ is the number of occurrences of the word w. It is worth noting that the term $\log\left(\frac{\#(w,c)\cdot|D|}{\#(w)\cdot\#(c)}\right)$ is actually the well known pointwise mutual information (PMI) of the word pair (w,c) and has been widely used in word embedding models [23], [25], [127], [128].

Moreover, PMI is also the basis for deriving the matrices factorized by random walk based models in [97], which are finally presented in matrix form.

4.3.2 From Natural Language to Graph

For graph representation learning models, some typical algorithms (e.g. DeepWalk [92], node2vec [42], LINE [120]) can also be shown to factorize their corresponding matrices implicitly (TABLE 2). Based on SGNS's implicit matrix M^{SGNS} (Eq. 4), the proof focuses on building the bridge between PMI of word-context pair (w,c) and the transition probability matrix of the network.

Factorizing Log-Empirical-Distribution Matrices. Theoretical results for the connections between shallow neural embedding algorithms and matrix factorization open a new direction for the optimization process of some neural based methods. Since each entry for this kind of matrices can be seen as the empirical connectivity preference [137] between the corresponding vertex-context pair (w,c), we refer to these matrices as *Log-Empirical-Distribution Matrices*. In [97], Qiu et al. try to factorize the matrix of DeepWalk [92] directly. Embedding vectors generated this way can outperform the embedding vectors obtained by the SGNS process employed in the original DeepWalk algorithm in the downstream tasks. In the matrix factorization part of ProNE [163], a matrix (Eq. 5, where λ is the negative sampling ratio and $P_{D,j}$ are negative samples associated with node v_j.) with only the first-order node proximities preserved (thus a sparse one) is generated through a similar way in [69] and is factorized to get raw embedding vectors.

$$M_{i,j} = \begin{cases} \ln p_{i,j} - \ln(\lambda P_{D,j}), (v_i,v_j) \in \mathcal{D} \\ 0, (v_i,v_j) \notin \mathcal{D} \end{cases} \tag{5}$$

In GraRep [14], SGNS matrices preserving each k-order proximity:

$$Y_{i,j}^k = \log\left(\frac{A_{i,j}^k}{\sum_t A_{i,j}^k}\right) - \log(\beta), \tag{6}$$

where A is the adjacency matrix, are generated and then factorized to get the embedding vectors preserving each k-order proximities. These embedding vectors, preserving different orders of node proximity, are then concatenated together to get the final embedding vectors for each node.

4.3.3 Differences Between Neural Based Embedding and Matrix Factorization Based Models

Although SGNS can be shown to implicitly factorize a matrix, there are also many differences between them.

SGNS needs to sample node pairs explicitly, which is time-consuming if we want to preserve high-order node proximities. At the same time, the matrix being generated is also a dense one, if high-order proximites are preserved. But a dense matrix can sometimes be approximated or replaced by a sparse one and then adopt other refinement methods [96], [163]. Thus, matrix factorization methods are more likely to be scaled to large-scale networks since complexity for factorizing a sparse matrix can be controlled to $O(|E|)$ with the development of numerical computation [36], [163]. Besides, factorizing matrices does not require tuning learning rates or other hyper-parameters.

However, factorizing matrices always suffer from unobserved data, which can be weighted naturally in sampling based methods [69]. In contrast, exactly weighting for matrix factorization is a hard computational problem.

4.4 Enhancing via Graph Spectral Filters

Apart from the shallow neural embedding models and models which adopt matrix factorization to generate vertex embeddings, recent literature has seen the wide application of graph spectral filters in generating high-quality and structure-aware vertex embeddings.

For example, in ProNE [163], the band-pass filter $g(\lambda) = e^{-\frac{1}{2}[(\lambda-\mu)^2-1]\theta}$ [46], [114] is designed to propagate raw vertex embedding vectors generated by factorizing a sparse matrix (Eq. 5) in the first stage. The propagation operation is also empirically proved to be an effective and universal method that can improve the quality of vertex embedding vectors obtained by many other embedding algorithms (e.g. DeepWalk [92], node2vec [42], GraRep [14], HOPE [88], LINE [120]).

In the embedding refinement stage of GraphZoom [28], it is found that the solution of the refinement problem:

$$\min_{\boldsymbol{E}} \|\boldsymbol{E}_i - \hat{\boldsymbol{E}}_i\|_2^2 + \mathrm{tr}(\boldsymbol{E}_i^T \boldsymbol{L}_i \boldsymbol{E}_i), \tag{7}$$

where $\hat{\boldsymbol{E}}_i$ is the embedding matrix to be refined, $\boldsymbol{E}_i$ is the desired matrix after refinement, $\boldsymbol{L}_i$ is the corresponding graph Laplacian, is:

$$\boldsymbol{E}_i = (\boldsymbol{I} + \boldsymbol{L}_i)^{-1} \hat{\boldsymbol{E}}_i. \tag{8}$$

It is equal to passing the original vertex embeddings through the low-pass filter $h(\lambda) = (1+\lambda)^{-1}$ in the spectral domain. The filter $h(\lambda) = (1+\lambda)^{-1}$ is further approximated by its first-order approximation $\tilde{h}(\lambda) = 1 - \lambda$ and then generalized to the k-order multiplication form: $\tilde{h}_k(\lambda) = (1-\lambda)^k$. Its matrix form $(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}})^k$, where $\tilde{\boldsymbol{A}}$ is the augmented adjacency matrix, is used to filter the embedding matrix $\hat{\boldsymbol{E}}_i$ to get the refined embedding matrix $\boldsymbol{E}_i$.

In GraphWave [33], the low-pass filter $g_s(\lambda) = e^{-\lambda s}$ is used to generate the spectral graph wavelet Ψ_a for each node a in the graph:

$$\Psi_a = \boldsymbol{U}\text{diag}(g_s(\lambda_1), \ldots, g_s(\lambda_N))\boldsymbol{U}^T\delta_a, \tag{9}$$

where $\boldsymbol{U}$, $\lambda_1, ..., \lambda_N$ are the eigenvector matrix and eigenvalues of the combinational graph Laplacian $\boldsymbol{L}$ respectively, $\delta_a = \mathbb{1}(a)$ is the one hot vector for node a. And m-th wavelet coefficient of this column vector Ψ_a is denoted by Ψ_{ma}. By characterizing the distribution via empirical characteristic functions:

$$\Phi_a(t) = \frac{1}{N}\sum_{m=1}^{N} e^{it\Psi_{ma}}, \tag{10}$$

and concatenating $\Phi_a(t)$ at d evenly spaced points $t_1, \ldots, t_d$ as follows (Eq. 11), a $2d$-dimension embedding vector for node a can be generated:

$$\chi_a = [\text{Re}(\Phi_a(t_i)), \text{Im}(\Phi_a(t_i))]_{t_1,\ldots,t_d}. \tag{11}$$

It can be proved that the k-hop structural equivalent and similar nodes a and b will have ϵ-structural similar wavelets Ψ_a and Ψ_b, where ϵ is the K-th order polynomial approximation error of the low-pass kernel $g_s(\lambda)$. Thus, the embedding vectors generated by wavelets can preserve the structural similarity.

Universal Graph Spectral Filters. Graph spectral filters have close connections with spatial properties. In fact, many models have the corresponding spectral filters as their kernels, which are further discussed in Section 7.1. Readers can refer to [46], [113], [115], [125], [134] for more details.

5 Heterogeneous Embedding Models

Although the embedding models discussed above are designed for homogeneous networks, they are actually the basis of many heterogeneous networks.

Heterogeneous networks are widespread in real-world, which have more than one type of vertices or edges. Thus, algorithms for heterogeneous network embedding are supposed to not only incorporate vertex attributes or labels with structural information, but also leverage vertex types, edge types, and also the semantic information that lies behind the connections between two vertices [32]. This is exactly where the challenge of heterogeneous network representation learning lies in.

Since there are already surveys for heterogeneous networks representation learning algorithms [16], [32], we will focus on the correlations between heterogeneous and homogeneous network embedding techniques in this section.

5.1 Heterogeneous LINE

In PTE [119], the heterogeneous network that has words, documents, labels as its vertices and the connections within them as the edges, is projected to three homogeneous networks first (word-word network, word-document network and word-label network). Then, for each bipartite network $G = (\mathcal{V}_A \cup \mathcal{V}_B, \mathcal{E})$, where $\mathcal{V}_A$ and $\mathcal{V}_B$ are two disjoint vertex sets, $\mathcal{E}$ is the edge set, the conditional probability of vertex v_j in set $\mathcal{V}_A$ generated by vertex v_i in set $\mathcal{V}_B$ is defined as:

$$p(v_j|v_i) = \frac{\exp(\vec{u}_j^T \cdot \vec{u}_i)}{\sum_{k\in A}\exp(\vec{u}_k^T \cdot \vec{u}_i)}, \tag{12}$$

similar with $p_2(v_j|v_i)$ (Eq. 3) in LINE [120]. Then the conditional distribution $p(\cdot|v_j)$ is forced to be close to its empirical distribution $\hat{p}(\cdot|v_j)$ by jointly minimizing the corresponding loss function similar with the one in LINE (Eq. 2).

Moreover, it is also proved in [97] that the implicit matrix factorized by PTE is in the following form:

$$\log\left(\begin{bmatrix} \alpha\text{vol}(G_{\text{ww}})(\boldsymbol{D}_{\text{row}}^{\text{ww}\,-1})\boldsymbol{A}_{\text{ww}}(\boldsymbol{D}_{\text{col}}^{\text{ww}-1}) \\ \beta\text{vol}(G_{\text{dw}})(\boldsymbol{D}_{\text{row}}^{\text{dw}\,-1})\boldsymbol{A}_{\text{dw}}(\boldsymbol{D}_{\text{col}}^{\text{ww}-1}) \\ \gamma\text{vol}(G_{\text{lw}})(\boldsymbol{D}_{\text{row}}^{\text{lw}\,-1})\boldsymbol{A}_{\text{lw}}(\boldsymbol{D}_{\text{col}}^{\text{ww}-1}) \end{bmatrix}\right) - \log \boldsymbol{b}, \tag{13}$$

where G_{ww}, G_{dw}, G_{lw} are word-word, document-word, label-word graphs respectively, with A_{ww}, A_{dw}, A_{lw} as their adjacency matrices and D^{ww}, D^{dw}, D^{lw} as their degreee matrices respectively, $\text{vol}(G) = \Sigma_i\Sigma_j A_{ij} = \Sigma_i d_i$ is the volume of the weighted graph G.

5.2 Heterogeneous Random Walk

The proposed meta-path based random walk in [31] provides a natural way to transform the heterogeneous networks into vertex sequences with both structural information and semantic information underlying different types of vertices and edges preserved. The key idea is to design specific meta paths which can restrict transitions between only specified types of vertices. To be specific, given a heterogeneous network $G = (\mathcal{V}, \mathcal{E})$ and a meta path scheme $\mathcal{P}: V_1 \xrightarrow{R_1} V_2 \xrightarrow{R_2} V_3 \cdots V_t \xrightarrow{R_t} \cdots \xrightarrow{R_{l-1}} V_l$, where $V_i \in \mathcal{O}$ are vertex types in the network, the transition probability is defined as:

$$P_{v^{i+1},v_t^i,\mathcal{P}} = \begin{cases} \frac{1}{|\mathcal{N}_{t+1}(v_t^i)|} & (v^{i+1}, v_t^i) \in \mathcal{E}, \Phi(v^{i+1}) = t+1 \\ 0 & (v^{i+1}, v_t^i) \in \mathcal{E}, \Phi(v^{i+1}) \neq t+1 \\ 0 & (v^{i+1}, v_t^i) \notin \mathcal{E} \end{cases} \tag{14}$$

where $\Phi(v_t^i) = V_t$, $\mathcal{N}_{t+1}(v_t^i)$ is the V_{t+1} type of neighbourhood of vertex v_t^i. Then the SGNS framework is applied to the generated random walks to optimize vertex embeddings. Moreover, the type-dependent negative sampling strategy is also proposed to better capture the structural and semantic information in heterogeneous networks.

Combined with Neighbourhood Aggregation. Different from the paradigm where node embedding and edge embedding vectors are defined directly as parameters to be optimized, attention mechanism is used in GATNE [17] to calculate embedding vectors for each node based on neighbourhood aggregation operation. Two embedding methods

are introduced: GATNE-T (transductive) and GATNE-I (inductive).

In GATNE, the overall embedding of node v_i on edge r is split into base embedding which is shared between different edge types and edge embedding. The k-th level edge embedding $u_{i,r}^{(k)} \in \mathbb{R}^s$, $(1 \leq k \leq K)$ of node v_i on edge type r is aggregated from neighbours' edge embeddings:

$$u_{i,r}^{(k)} = aggregator(u_{j,r}^{(k-1)}), \forall v_j \in \mathcal{N}_{i,r}, \tag{15}$$

where $\mathcal{N}_{i,r}$ is the neighbours of node v_i on edge type r. After the K-th level edge embeddings are calculated, the overall embedding $v_{i,r}$ of node v_i on edge type r is computed by applying self-attention mechanism on the concatenated embedding vector of node v_i:

$$U_i = (u_{i,1}, u_{i,2}, ..., u_{i,m}). \tag{16}$$

The base embedding b_i for node v_i is then added as the embedding bias on the self-attention result.

The difference between transductive model (GATNE-T) and the inductive model (GATNE-I) lies in how the 0-th level edge embedding vector of each node v_i on each edge type r and the base embedding vector of each node v_i is calculated. In GATNE-T, they are optimized directly as parameters, while in GATNE-I, they are computed by applying transformation functions h_z and $g_{z,r}$ on the raw feature x_i of each node v_i: $b_i = h_z(x_i), u_{i,r}^{(0)} = g_{z,r}(x_i)$. Transformation functions h_z and $g_{z,r}$ are optimized during the training process.

The neighbourhood aggregation mechanism increases the model's inductive bias and also make it easier combining with node features, which is similar with the core idea of inductive embedding models based on graph neural networks.

Meta Paths Augmentation. Meta-paths can also be treated as the relations or "edges" between the corresponding connected vertices to augment the network. In HIN2vec [37], meta-paths are treated as the relations between vertices connected by them with learnable embeddings. Then probability of the two vertices x and y connected by meta-path r is modeled by:

$$P(r|x,y) = \text{sigmoid}\left(\sum \boldsymbol{W}'_X \vec{x} \odot \boldsymbol{W}'_Y \vec{y} \odot f_{01}(\boldsymbol{W}'_R \vec{r})\right), \tag{17}$$

where $\boldsymbol{W}_X$, $\boldsymbol{W}_Y$ are vertex embedding matrices, $\boldsymbol{W}_R$ is the relation embedding matrix, $\boldsymbol{W}'_X$ is the transpose of matrix $\boldsymbol{W}_X$, $\vec{x}, \vec{y}, \vec{r}$ are one-hot vectors for two connected vertices x, y and the relation between them respectively, $f_{01}(\cdot)$ is the regularization function. Parameters are optimized by maximizing the following objective:

$$\begin{aligned} \log O_{x,y,r}(x,y,r) = L(x,y,r) \log P(r|x,y) + \\ (1 - L(x,y,r)) \log(1 - P(r|x,y)), \end{aligned} \tag{18}$$

where $L(x,y,r) = 1$ if vertices x, y is connected by relation r, otherwise $L(x,y,r) = 0$.

In TapEM [21], the proximity between two vertices i, j on two sides of the given meta-path of type r is explicitly preserved by modeling the conditional probability $P(j|i;r)$, where i, j are vertices on two sides of the meta-path, r is the type of the meta-path.

In HeteSpaceyWalk [49], the heterogeneous personalized spacey random walk algorithm is proposed, which is a space-friendly and efficient approximation for meta-path based random walks and can converge to the same limiting stationary distribution.

Summary. Compared with PTE, random walk for heterogeneous networks can capture the structural dependencies between different types of vertices better and also preserve higher-order proximities. But they both need manual design with expert knowledge in advance (how to separate networks in PTE and how to design meta paths). Moreover, just using the information of types of the meta path between two connected vertices may lose some information (e.g., vertex or edge types, vertex attributes) passing through the meta path [53].

5.3 Other Models

Apart from the random walks and Skip-Gram framework, there are also many other homogeneous embedding models that can be used in heterogeneous network embedding algorithms (e.g., label propagation, matrix factorization, generate adversarial approach applied in GraphGAN [137]). The assumption of label propagation in homogeneous networks that "two connected vertices tend to have the same labels" is generalized to the heterogeneous networks in LSHM [60] by assuming that two vertices of the same type connected by a path tend to have similar latent representations. GAN [41] is used in HeGAN [54] with relation-aware discriminator and generator to perform better negative sampling.

Moreover, supervised information can also be added for downstream tasks. For example, the idea of PTE, meta paths and matrix factorization are combined in HERec [111] with supervised information from recommendation task. To be specific, vertex and item embedding vectors are firstly generated by performing meta path-based random walks on the graph and then the user-item rating matrix is introduced to help learn fusion functions on those embeddings.

6 Graph Neural Network Based Models

Graph neural networks (GNNs) are kind of powerful feature extractor for graph structured data and have been widely used in graph embedding problems. There are some inherent problems in shallow embedding models, which will be discussed in later sections, and the presence of GNNs can alleviate these problems to some extent.

Past few years have seen the rapid development of Graph Neural Networks in graph mining tasks. GNNs' architecture can enable them to effectively model structural and relational data. Compared with shallow embedding models that have been discussed before, GNNs have a deep architecture and can model vertex attributes as well as network structure naturally. These are typically neglected, or cannot be modeled efficiently in shallow embedding models.

There are two main streams in designing GNNs. The first is in the graph spectral fashion, in which the convolutional operation can be seen as passing vertex features through a low-pass filter in the spectral domain. We refer to these GNNs as *graph spectral GNNs*. The classical graph convolution network (GCN) [64], [65], ChebyNet [27], FastGCN [19],

Model	T	S	A-M	Aggregation Function	Appendix
GCN [65]	E	×	×	$\boldsymbol{H}^{(l+1)} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}^{(l)}\Theta$	$\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_N$
GraphSAGE [45]	E	N	×	$h^k_{\mathcal{N}(v)} \leftarrow \text{AGGREGATE}_k(\{h^{k-1}_u, \forall u \in \mathcal{N}(v)\})$ $\text{h}^k_v \leftarrow \sigma\left(\boldsymbol{W}^k \cdot \text{CONCAT}\left(h^{k-1}_v, h^k_{\mathcal{N}(v)}\right)\right)$	AGGREGATE ∈ {MAX POOL, MEAN, LSTM}
FastGCN [19]	E	L	×	$\boldsymbol{H}^{(l+1)}(v,:) = \sigma\left(\frac{1}{t_l}\sum_{j=1}^{t_l}\frac{\hat{A}(v,u_j^{(l)})\boldsymbol{H}^{(l)}(u_j^{(l)},:)\boldsymbol{W}^{(l)}}{q(u_j^{(l)})}\right)$, (19) $u_j^{(l)} \sim q, l = 0, 1, \ldots, M$	$q(u) = \frac{\|\hat{A}(:,u)\|^2}{\sum_{u' \in V}\|\hat{A}(:,u')\|^2}, u \in V$
ASGCN [59]	E	L	×	$h^{(l+1)}(v_i) = \sigma_{W^{(l)}}\left(N(v_i)\frac{1}{n}\sum_{j=1}^{n}\frac{p(\hat{u}_j\|v_i)}{q(\hat{u}_j\|v_1,\ldots,v_n)}h^{(l)}(\hat{u}_j)\right)$ (20) $\hat{u}_j \sim q(\hat{u}_j\|v_1,\ldots,v_n)$	Eq. 26; $g(x(u_j)) = \boldsymbol{W}_g x(u_j)$
GAT [130]	A	×	✓	$h_i^{(l+1)} = CONCAT_{k=1}^{K}\left[\sigma\left(\sum_{j\in\mathcal{N}_i}\alpha_{ij}^k\boldsymbol{W}^k h_j^{(l)}\right)\right]$	Eq. 27
RGCN [170]	E	×	×	$\boldsymbol{M}^{(l+1)} = \rho\left(\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\left(\boldsymbol{M}^{(l)} \odot \boldsymbol{\mathcal{A}}^{(l)}\right)\boldsymbol{W}_\mu^{(l)}\right)$ $\boldsymbol{\Sigma}^{(l+1)} = \rho\left(\tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1}\left(\boldsymbol{\Sigma}^{(l)} \odot \boldsymbol{\mathcal{A}}^{(l)} \odot \boldsymbol{\mathcal{A}}^{(l)}\right)\boldsymbol{W}_\sigma^{(l)}\right)$	$\boldsymbol{\mathcal{A}^{(l)}} = \exp(-\boldsymbol{\gamma\Sigma^{(l)}})$
SGC [145]	E	×	×	$Y_{SGC} = \text{softmax}\left((\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}})^K\boldsymbol{X}\boldsymbol{\Theta}\right)$	$\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_N$
GIN [148]	A	×	×	$h_v^{(l+1)} = \text{MLP}^{(l+1)}\left(\left(1+\epsilon^{(l+1)}\right)\cdot h_v^{(l)} + \sum_{u\in\mathcal{N}(v)} h_u^{(l)}\right)$	-
ACR-GNN [6]	A	×	×	Eq. 31	-
R-GCN [108]	A	×	×	$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}\boldsymbol{W}_r^{(l)}h_j^{(l)} + \boldsymbol{W}_0^{(l)}h_i^{(l)}\right)$	$\boldsymbol{W}_r^{(l)} = \sum_{b=1}^{B} a_{rb}^{(l)}V_b^{(l)}$ or $\boldsymbol{W}_r^{(l)} \bigoplus_{b=1}^{B}\boldsymbol{Q}_{br}^{(l)}$
GraLSP [62]	A	N	✓	$a_i^{(k)} = \text{MEAN}_{\boldsymbol{w}\in\mathcal{W}^{(i)}, p\in[1,r_{\boldsymbol{w}}]}\left(\lambda_{i,w}^{(k)}\left(q_{i,w}^{(k)} \odot h_{w_p}^{(k-1)}\right)\right)$ $h_i^{(k)} = \text{ReLU}\left(\boldsymbol{U}^{(k)}h_i^{(k-1)} + \boldsymbol{V}^{(k)}a_i^{(k)}\right), k = 1, 2, ..., K$ (21) $h_i = h_i^{(K)}$	$\lambda_{i,\boldsymbol{w}}$: attention coefficient $q_{i,\boldsymbol{w}}$: amplification coefficient $r_{\boldsymbol{w}}$: receptive window, Eq. 24

TABLE 3: A summary of Graph Neural Networks. Part of the symbols in the formula can refer to Def. 5. For others, $\boldsymbol{H}^{(l)}$ denotes to the feature matrix in layer l, h_v^k denotes feature vector of node v in layer k, $h^{(l)}(v_i)$ or $h_i^{(l)}$ denotes feature vector of node v_i (or i) in layer l, $\boldsymbol{\Theta}$ and $\boldsymbol{W}$ are trainable parameters, $\boldsymbol{M}^{(l)}$ and $\boldsymbol{\Sigma}^{(l)}$ are mean and variance matrices of vertex features in layer l respectively, $\mathcal{N}(v)$ denotes the set of node v's neighbours or the sampled neighbours, $h^k_{\mathcal{N}(v)}$ denotes the feature vector aggregated from node v's sampled neighbours in layer k, $q(\cdot)$ denotes the sampling distribution. For abbreviations used, "E" denotes "Spectral", "A" denotes "Spatial", "A-M" denotes "Attention Mechanism", "T" denotes "Type", "S" denotes "Sampling Strategy", "N" refers to "Node-Wise Sampling", "L" refers to "Layer-Wise Sampling".

ASGCN [59], GWNN [147] and the graph filter network (gfNN) proposed in [51] are examples of *graph spectral GNNs*. The convolution process is performed in the spectral domain in such GNNs, in which node features are first transferred to spectral domain and then multiply with a spectral filter matrix. Desired spectral convolution process should be economic in computation and also localized in spatial domain [27], [147]. Then, the second type is *graph spatial GNNs*, which operate on vertex features in the spatial domain directly. They update each node's features by linearly combining (or aggregating) its neighbours' features. It is similar to the spatial convolution discussed in [46]. GraphSAGE [45], Graph Isomorphism Network (GIN) [148], and MPNN [39] are examples for this kind of GNNs.

Compared with spectral GNNs, the spatial convolution employed in the spatial GNNs usually just focus on 1-st neighbours of each node. However, the local property of spatial convolution operation can help spatial GNNs be inductive.

Compared with shallow embedding models, GNNs can better combine the structural information with vertex attributes, but the need for vertex attributes also make GNNs hard to be applied to homogeneous networks without vertex features. Although it has been proposed in [65] that we can set the feature matrix $\boldsymbol{X} = \boldsymbol{I}_N$, where $\boldsymbol{I}_N \in \mathbb{R}^{N\times N}$ is the identity matrix and N is the number of vertices, for featureless graphs, it cannot be scaled to large networks.

Apart from GNNs' advantages in content augmentation, they can also be trained in the supervised or semi-supervised fashion easily (in fact, GCN is proposed for semi-supervised classification). Label augmentation can improve the discriminative property of the learned features [162]. Moreover, neural network architecture can help with the design of an end-to-end model, fitting in downstream tasks better.

6.1 GNN Models

While the powerful CNNs can also be effectively applied to data that can be organized in grid structures (e.g. images [66], sentences [63], and videos [122]), they cannot be generalized to graphs directly. Inspired by the effectiveness of CNNs in extracting features from grid structures, many previous works focus on properly defining the convolution operation for graph data to capture structural information.

To the best of our knowledge, it was in [12] that convolutions for graph data were first introduced based on graph spectral theory [27] and graph signal processing [113], where both multilevel convolutional neural networks in spectral and spatial domains were built with few parameters to

learn, which preserve nice qualities for CNNs. The spectral convolution operation in [12] is actually a low-pass filtering operation, consistent with the ideas for building graph neural networks in the following works [45], [51], [65], [130].

GCN is proposed in [65], which uses the first-order approximation of the graph spectral convolution and the augmented graph adjacency matrix to design the feature convolution layer's architecture.

After the proposition of GCN [65], many GNN models are designed based on it. They try to make some improvements, such as introducing sampling strategies [19], [45], [59], adding attention mechanism [124], [130], or improving the filter kernel [51], [145]. We briefly summarize parts of existing GNN models in TABLE 3 and discuss some examples in the following parts.

6.1.1 Sampling

Sampling techniques are introduced to reduce the time complexity of GCN or introduce the inductive bias. There are various sampling strategies and we will introduce some of them in the following parts, such as node-wise sampling [45], [62], layer-wise sampling [19], [59] and subgraph sampling [156].

Node-Wise Sampling. Based on GCN, GraphSAGE [45] introduces node-wise sampling to randomly sample a fixed size neighbourhood for each node in each layer and also shift to the spatial domain to help it become inductive.

However, as proposed in ASGCN [59], its node-wise sampling strategy would probability lead to the number of sampled nodes grows exponentially with the number of layers. If the depth of the network is d, then the number of sampled nodes in the input layer will increase to $O(n^d)$, where n is the number of sampled neighbours of each node, leading to significant computational burden for large d.

Different from the random sampling strategy introduced in GraphSAGE, an adaptive node-wise sampling strategy is proposed in GraLSP [62]. In GraLSP, the vertex v's neighbourhood is sampled by performing random walks of length l starting at vertex v. The aggregation process is also combined with attention mechanism. Structural information is preserved by introducing *Random Anonymous Walk* [82], which is calculated based on the sampled random walk $\boldsymbol{w} = (w_1, w_2, ..., w_l)$:

$$\text{aw}(\boldsymbol{w}) = (\text{DIS}(\boldsymbol{w}, w_1), \text{DIS}(\boldsymbol{w}, w_2), ..., \text{DIS}(\boldsymbol{w}, w_l)), \quad (22)$$

where $\text{DIS}(\boldsymbol{w}, w_i)$ denotes the number of distinct nodes in $\boldsymbol{w}$ when w_i first appears in $\boldsymbol{w}$:

$$\text{DIS}(\boldsymbol{w}, w_i) = |\{w_1, w_2, ..., w_p\}|, p = \min_j\{w_j = w_i\}. \quad (23)$$

Then the adaptive receptive radius is defined as:

$$r_{\boldsymbol{w}} = \left\lfloor \frac{2l}{\max(\text{aw}(\boldsymbol{w}))} \right\rfloor, \quad (24)$$

where $\max(\text{aw}(\boldsymbol{w}))$ is equal to the number of distinct nodes visited by walk $\boldsymbol{w}$. Then, the first $r_{\boldsymbol{w}}$ nodes in the random walk $\boldsymbol{w}$ started at node v are chosen to pass their features to node v.

Layer-Wise Sampling. Different from node-wise sampling strategies, nodes in the current layer are sampled based on all the nodes in the previous layer. To be specific, layer-wise sampling strategies aim to find the best and tractable sampling distribution $q(\cdot|v_1, \ldots, v_{t_{l-1}})$ for each layer l based on nodes sampled in layer $l-1$: $\{v_1, \ldots, v_{t_{l-1}}\}$. The sampling distributions aim to minimize the variance introduced by performing sampling. However, the best sampling distributions are always cannot be calculated directly, thus some tricks and relaxations are introduced to obtain sampling distributions that can be used in practice [19], [59]. The sampling distribution is

$$q(u) = \|\hat{A}(:, u)\|^2 / \sum_{u' \in V} \|\hat{A}(:, u')\|^2, u \in V, \quad (25)$$

in FastGCN [19] and

$$q^*(u_j) = \frac{\sum_{i=1}^n p(u_j|v_i)|g(x(u_j))|}{\sum_{j=1}^N \sum_{i=1}^n p(u_j|v_i)|g(x(u_j))|}, \quad (26)$$

in ASGCN [59], where $g(x(u_j))$ is a linear function (i.e., $g(x(u_j)) = W_g x_{u_j}$) applied on vertex features of node u_j.

Subgraph Sampling. Apart from node-wise and layer-wise sampling strategies, which sample a set of nodes in each layer, a subgraph sampling strategy is proposed in [156], which samples a set of nodes and edges in each training epoch and perform the whole graph convolution operation on the sampled subgraph.

Edge sampling rates are proposed aiming to reduce the variance of node features in each layer introduced by performing subgraph sampling strategy. It is set to $p_{u,v} \propto \frac{1}{\deg(u)} + \frac{1}{\deg(v)}$ in practice. Based on edge sampling rates, different samplers can be designed to sample the subgraph. For random edge sampler, edges are sampled just using the edge sampling distribution discussed above. For random node sampler, a certain number of nodes are sampled under the node sampling rate $P(u) \propto \|\tilde{A}_{:,u}\|^2$, where $\tilde{A}$ is the normalized graph adjacency matrix. For random walk based sampler, the sampling rate for the node pair (u, v) is set to $p_{u,v} \propto B_{u,v} + B_{v,u}$, where $\boldsymbol{B} = \tilde{\boldsymbol{A}}^L$ and L is the length of the random walk. Besides, there are also many other random walk based samplers proposed in previous literature [56], [68], [101], which can also be used to sample the subgraph.

6.1.2 Attention Mechanism

Introducing an attention mechanism can help improve models' capacities and interpretability [130] by assigning different weights to nodes in a same neighborhood explicitly. It is interesting that the attention mechanism used in GAT [130] will make the model easy to be attacked due to the aggregation process's dependency on neighbours' features. But the one used in RGCN can help improve model's robustness by assigning features with a larger variance lower weights.

Besides, the comparison between attention mechanism and the sampling and LSTM-aggregation strategy used in GraphSAGE can cast some similar insights with the comparison between RNN based models and attention based models for sequence modeling in NLP domain. Attention mechanism can obtain a more comprehensive view over nodes' neighbourhood than RNN based aggregation strategies.

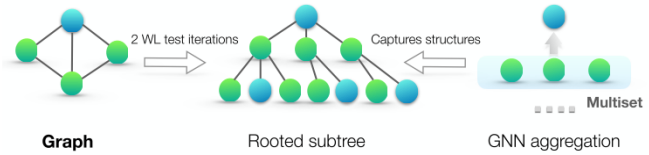

Fig. 5: Illustration for WL test and the relationship with GNNs. Middle Panel: rooted subtree of the blue node in the left panel. Right Panel: if a GNN's aggregation function can capture the full multiset of node neighbours, then it can capture the rooted subtree and be as powerful as WL test in distingusihing different graphs. Reprinted from [148]

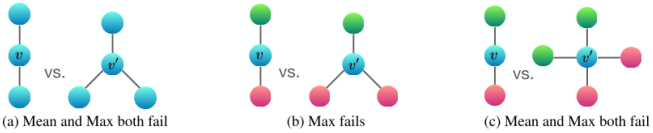

Fig. 6: Examples where max aggregator and mean aggregator will fail. For each subimage, node v and node v' will get the same embeddings under corresponding aggregators even though their neighbourhood structures are different from each other. Reprinted from [148]

Performing attention mechanism on neighbours' features can also be seen as a feature rescaling process, which can be used to unify GNN models in a same framework. The choice of specific attention strategy depends on our purpose and practice.

In GAT [130], the calculation for k-th head's attention weight α_{ij}^k between two nodes i and j is Eq. 27, where $\vec{h}_i \in \mathbb{R}^{d\times 1}$ is the feature vector for vertex i, $\boldsymbol{W}^k \in \mathbb{R}^{d\times d'}$, $\vec{\alpha} \in \mathbb{R}^{2d'\times 1}$ are corresponding parameters, d, d' are the dimension for feature vector in the previous layer and current layer respectively. Different from GAT, the attention weight between nodes i and j is calculated based on the cosine similarity between their hidden representations (Eq. 28, where $\beta(l)$ are trained attention-guided parameters of layer l.) in [124], where $\cos(\cdot,\cdot)$ represents the cosine similarity.

$$\alpha_{ij}^k = \frac{\exp(\text{LeakyReLU}(\vec{a}^T[\boldsymbol{W}^k\vec{h}_i \| \boldsymbol{W}^k\vec{h}_j]))}{\sum_{k\in\mathcal{N}_i}\exp(\text{LeakyReLU}(\vec{a}^T[\boldsymbol{W}^k\vec{h}_i \| \boldsymbol{W}^k\vec{h}_k]))} \tag{27}$$

$$\alpha_{ij} = \frac{\exp(\beta(l)\cos(H_i, H_j))}{\sum_{j\in\mathcal{N}_i}\exp(\beta(l)\cos(H_i, H_j))} \tag{28}$$

Besides, attention mechanism is also widely used in heterogeneous network embedding algorithms, by applying which the various semantic information underlying different kind of connections between vertices. More discussions can be seen in Section 6.1.4.

6.1.3 Discriminative Power

Weisfeiler-Lehman (WL) Graph Isomorphism Test. GNN's inner mechanism is similar with the Weisfeiler-Lehman (WL) graph isomorphism test (Fig. 5) [45], [110], [143], [148], which is a powerful test [110] known to distinguish a broad class of graphs, despite of some corner cases. Comparisons between GNNs and WL test allow us to understand the capabilities and limitations of GNNs more clearly.

It is proved in [148] that GNNs are at most as powerful as the WL test in distinguishing graph structures and can be as powerful as WL test only if using proper neighbour aggregation functions and graph readout functions ([148] Theorem 3). Those functions are applied on the set of neighbours' features, which can be treated as a multi-set [148]. For neighbourhood aggregation functions, it is concluded that other multi-set functions like mean, max aggregators are not as expressive as the sum aggregator (Fig. 6).

One kind of powerful GNNs is proposed by taking "SUM" as its aggregation function over neighbours' feature vectors and MLP as its transformation function, whose feature updating function in the k-th layer is:

$$h_v^k = MLP^k((1+\epsilon^k)\cdot h_v^{k-1} + \sum_{u\in\mathcal{N}(v)} h_u^{k-1}). \tag{29}$$

Logical Classifier. In [6], Boolean classifiers expressible as formulas in the logic FOC_2, which is a well-studied fragment of first-order logic, are studied and used to judge the GNNs' logical expressiveness. It is shown that a popular class of GNNs, called AC-GNNs (Aggregate-Combine GNNs, whose feature updating function can be written as Eq. 30, where COM = COMBINE, AGG = AGGREGATE, $x_v^{(i)}$ is the feature vector of vertex v in layer i) in which the features of each node in the successive layers are only updated in terms of node features of its neighbourhood, can only capture a specific part of FOC_2 classifiers.

$$x_v^{(i)} = \text{COM}^{(i)}(x_v^{(i-1)}, \text{AGG}^{(i)}(\{x_u^{(i-1)} | u\in\mathcal{N}_G(v)\})), \quad \text{for } i = 1,\dots,L \tag{30}$$

By simply extending AC-GNNs, another kind of GNNs are proposed (i.e., ACR-GNN(Aggregate-Combine-Readout GNN)), which can capture all the FOC_2 classifiers:

$$x_v^{(i)} = \text{COM}^{(i)}(x_v^{(i-1)}, \text{AGG}^{(i)}(\{x_u^{(i-1)} | u\in\mathcal{N}_G(v)\}), \text{READ}^{(i)}(\{x_u^{(i-1)} | u\in G\})), \text{for } i = 1,\dots,L, \tag{31}$$

where READ = READOUT. Since the global computation can be costly, it is further proposed that just one readout function together with the final layer is enough to capture each FOC_2 classifier instead of adding the global readout function for each layer.

6.1.4 From Homogeneous to Heterogeneous

Different from GNN models for homogeneous networks, GNNs for heterogeneous networks concern how to aggregate vertex features of different vertex types or connected with edge of different types. Attention mechanism is widely used in the design process of GNNs for heterogeneous networks [140], [174].

In [108], the relational graph convolution network (R-GCN) is proposed to model large-scale relation data based on the message-passing frameworks. Different weight matrices are used for different relations in each layer to aggregate and transform hidden representations from each node's neighbourhood.

In HetGNN [157], a feature type-specific LSTM model is used to extract features of different types for each node, followed by another vertex-type specific LSTM model which is used to aggregate extracted feature vectors from different types of neighbours. Then, the attention mechanism is used to combine representation vectors from different types of neighbours.

Heterogeneous Graph Attention Network (HAN) is proposed in [140], where meta paths are treated as edges between the connected two nodes. Here an attention mechanism based on meta paths and nodes is used to calculate neighbourhood aggregation vector and embedding matrix.

Heterogeneous Graph Transformer is proposed in [174]. A node type-specific attention mechanism is used and weights for different meta paths are learned automatically.

Apart from embedding different types of nodes to the same latent space, it is also proposed in HetSANN [53] to assign different latent dimensions to them. During the aggregation process, transformation matrices are applied on the feature vectors of the target node's neighbours to transform them to the same latent space with the target node. Then, the attention based aggregation process is applied on the projected neighbourhood vectors.

7 Theoretical Foundations

Understanding different models in a universal framework can cast some insights on the design process of corresponding embedding models (like the powerful spectral filters). Thus, in this section, we will review some theoretical basis and recent understandings for models discussed above, which can benefit the further development of related algorithms.

7.1 Underlying Kernels: Graph Spectral Filters

We want to show that most of the models discussed above, whether in a shallow architecture or based on graph neural networks, have some connections with graph spectral filters.

For shallow embedding models, it has been shown in [97] that the some neural based models (e.g. DeepWalk [92], node2vec [42], LINE [120]) are implicitly factorizing matrices (TABLE 2). Furthermore, DeepWalk matrix can also be seen as filtering [97].

Moreover, the convolutional operation in spectral GNNs can be interpreted as a low-pass filtering operation [51], [145]. The understanding can also be easily extended to spatial GNNs since the spatial aggregation operation can be transferred to the spectral domain, according to [113].

Apart from those implicitly filtering models, graph filters have also been explicitly used in ProNE [163], GraphZoom [28] and GraphWave [33] to refine vertex embeddings or generate vertex embeddings preserving certain kind of vertex proximities.

7.1.1 Spectral Filters as Feature Extractors

Spectral filters can be seen and used as the effective feature extractors based on their close connection with graph spatial properties.

For example, the band-pass filter $g(\lambda) = e^{-\frac{1}{2}[(\lambda-\mu)^2-1]\theta}$ is used in ProNE [163] to propagate vertex embeddings obtained by factorizing a sparse matrix in the first stage. The idea for "band-pass" is inspired by the Cheeger's inequality:

$$\frac{\lambda_k}{2} \leq \rho_G(k) \leq O(k^2)\sqrt{\lambda_k}, \tag{32}$$

where $\rho_G(k)$ is the k-way Cheeger constant, a smaller value of which means a better k-way partition. A well-known property can be concluded from Eq. 32 when setting $\lambda_k = 0$: the number of connected components in an undirected graph is equal to the number of eigenvalue zero in the graph Laplacian [22]. Then the band-pass filter is hoped to extract the both global and local network information from raw embeddings.

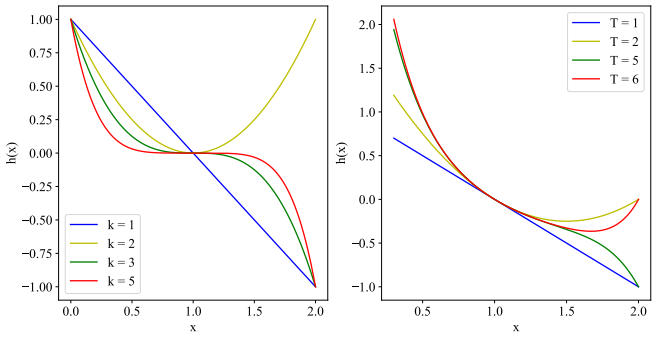

Fig. 7: Image of the filter function $h_k(x) = (1-x)^k$, $k \in \{1, 2, 3, 5\}$ (Left Panel) and DeepWalk matrix filter function $h(x) = -(-x^{-1}+1+x^{-1}(1-x)^{T+1})$, $T \in \{1, 2, 5, 6\}$ (Right Panel). Left Panel: Increasing the value of k can increase the band-stop characteristic of the filter function. Right Panel: The effect of increasing the window size T on the filter function.

In addition, heat kernel $g_s(\lambda) = e^{-\lambda s}$ is used in GraphWave [33] to generate wavelets for each vertex (Eq. 9), based on which vertex embeddings are calculated via empirical characteristic functions. More importantly, structural similarities can be preserved by calculating vertex embeddings in this way.

Apart from band-pass filters and heat kernels, low-pass filters are kind of more widely used filters not only in shallow embedding models [163], but the aggregation matrices in GNNs can be seen as low-pass matrices and the corresponding graph convolution operations can be treated as low-pass filtering operations [51], [145].

For shallow embedding models, the low-pass filter $\tilde{h}_k(\lambda) = (1-\lambda)^k$ is used to propagate the embedding matrix $\hat{\boldsymbol{E}}_i$ to get the refined embedding matrix $\boldsymbol{E}_i$ in the embedding refinement statement of GraphZoom [28].

As for GNNs, passing vertex features through the matrix $\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}} = \boldsymbol{I}_N - \tilde{\boldsymbol{\mathcal{L}}}$ in GCN [65], where N is the number of vertices, is equal to filtering features with the filter $h(\lambda) = 1-\lambda$ in the spectral domain (Eq. 33), where Λ, $\boldsymbol{U}$ are the eigenvalue matrix and eigenvector matrix of $\tilde{\boldsymbol{\mathcal{L}}}$ respectively.

$$\boldsymbol{Z} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{X}\boldsymbol{\Theta} = \boldsymbol{U}(\boldsymbol{I}_N - \boldsymbol{\Lambda})\boldsymbol{U}^T\boldsymbol{X}\boldsymbol{\Theta} \tag{33}$$

The filter $h(\lambda) = 1-\lambda$ is a low-pass filter since the eigenvalues of the normalized graph Laplacian satisfy the following property:

$$0 = \lambda_0 < \lambda_1 \leq \cdots \leq \lambda_{max} \leq 2, \tag{34}$$

and $\lambda_{max} = 2$ if and only if the graph has a bipartite subgraph as its connected component [51], [97], [113].

The low-pass property of propagation matrices in GNNs is further explored in [51], [145]. It is proved that two techniques can enhance the low-pass property for the filters. Let $\lambda_i(\sigma)$ be the i-th smallest generalized eigenvalue of the augmented normalized graph Laplacian $(\tilde{\boldsymbol{D}}, \tilde{\boldsymbol{\mathcal{L}}}) = \boldsymbol{D} + \sigma\boldsymbol{I}$,

then $\lambda_i(\sigma)$ is a non-negative number, and monotonically decreases as the non-negative value σ increases ($\lambda_i(\sigma) = 0$ for all the $\sigma > 0$ if $\lambda_i(0) = 0$). Thus, the high frequency components will be gradually attenuated by the corresponding spectral filtering operation as σ increases. Besides, increasing the power of the graph filter $h_k(\lambda) = (1-\lambda)^k$ (i.e., stacking several GCN layers or directly using the filter $h_k(\lambda) = (1-\lambda)^k, k > 1$ in SGC [145] and gfNN [51]) can help increase the low-pass property of the spctral filter (Fig. 7 Left channel).

Moreover, the propagation matrix used in gfNN is k-th power of the augmented random walk adjacency matrix $\tilde{\boldsymbol{A}}_{rw}^k = (\tilde{\boldsymbol{D}}^{-1}\tilde{\boldsymbol{A}})^k$, whose corresponding spectral filter is $h_k(\lambda) = (1-\lambda)^k$, where λ is the generalized eigenvalues and matrix $\boldsymbol{\Lambda} = \boldsymbol{U}^T\tilde{\boldsymbol{D}}\tilde{\boldsymbol{L}}_{rw}\boldsymbol{U}$. $\boldsymbol{U}$ is the generalized eigenvector matrix with the property $\boldsymbol{U}^T\tilde{\boldsymbol{D}}\boldsymbol{U} = \boldsymbol{I}$. The generalized eigenpair (λ, u) satisfies $\boldsymbol{L}u = \lambda\tilde{\boldsymbol{D}}u$. They are also solutions of the generalized eigenvalue problem in variation form [51], which aims to find $u_1, \ldots, u_n \in \mathbb{R}^n$ such that for each $i \in 1, \ldots, n$, u_i is a solution of the following optimization problem:

$$\begin{aligned}\text{minimize } \Delta(u) \text{ subject to } (u,u)_{\tilde{\boldsymbol{D}}} = 1, (u,u_j)_{\tilde{\boldsymbol{D}}} = 0,\\ j \in 1, \ldots, n\end{aligned} \tag{35}$$

where $\Delta(u) = u^T\boldsymbol{L}u$ is the *variantion* of the signal u and $(u,u_j)_{\tilde{\boldsymbol{D}}} = u^T\tilde{\boldsymbol{D}}u$ is the *inner product* between signal u and u_j. If (λ, u) is a generalized eigenpair, then $(\lambda, \tilde{\boldsymbol{D}}^{1/2}u)$ is an eigenpair of $\tilde{\boldsymbol{\mathcal{L}}}$.

Compared with the eigenvalues and eigenvectors of the normalized graph Laplacian, generalized eigenvectors with smaller generalized eigenvalues are smoother in terms of the variation Δ.

Solution for Optimization Problems. The embedding refinement problem in GraphZoom has seen that the low-pass filter matrix can serve as the close form solution of the optimization problem related with Laplacian regularization.

Another example is the Label Propagation (LP) problem for graph based semi-supervised learning [9], [168], [173], the close form of whose optimization objective function (Eq. 36) is Eq. 37, where Y is the label matrix and Z is the objective of LP that is consistent with the label matrix Y as well as being smoothed on the graph to force nearby vertices to have similar embeddings.

$$\boldsymbol{Z} = \arg\min_{\boldsymbol{Z}} \|\boldsymbol{Z} - \boldsymbol{Y}\|_2^2 + \alpha \cdot \text{tr}(\boldsymbol{Z}^T\boldsymbol{L}\boldsymbol{Z}) \tag{36}$$

$$\boldsymbol{Z} = (\boldsymbol{I} + \alpha\boldsymbol{L})^{-1}\boldsymbol{Y} \tag{37}$$

7.1.2 Spectral Filters as Kernels of Matrices being Factorized

We want to show that some matrices being factorized by matrix factorization algorithms can also be seen as filter matrices.

Take the matrix factorized by DeepWalk [92] as an example. It has been shown in [97] that the matrix term $\left(\frac{1}{T}\sum_{r=1}^T \boldsymbol{P}^r\right)\boldsymbol{D}^{-1}$ in DeepWalk's matrix can be written as

$$\left(\frac{1}{T}\sum_{r=1}^T \boldsymbol{P}^r\right)\boldsymbol{D}^{-1} = \left(\boldsymbol{D}^{-\frac{1}{2}}\right)\left(\boldsymbol{U}\left(\frac{1}{T}\sum_{r=1}^T \boldsymbol{\Lambda}^r\right)\boldsymbol{U}^T\right)\left(\boldsymbol{D}^{-\frac{1}{2}}\right) \tag{38}$$

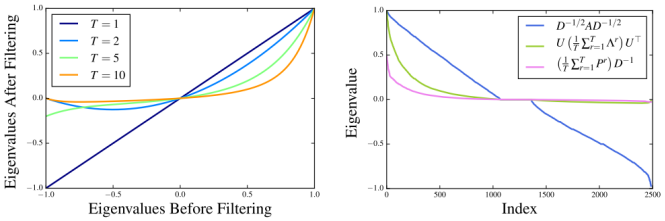

Fig. 8: Filtering characteristic of DeepWalk matrix's function. Left Panel: Image of the function $f(x) = \frac{1}{T}\sum_{r=1}^T x^r$ with $\textbf{dom}f = [-1, 1]$, $T = 1, 2, 5, 10$. Right Panel: Eigenvalues of $\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$, $\boldsymbol{U}\left(\frac{1}{T}\sum_{r=1}^T \boldsymbol{\Lambda}^r\right)\boldsymbol{U}^T$, $\left(\frac{1}{T}\sum_{r=1}^T \boldsymbol{P}^r\right)\boldsymbol{D}^{-1}$ for Cora network ($T = 10$). Reprinted from [97].

where $\boldsymbol{\Lambda}$, $\boldsymbol{U}$ are the eigenvalue matrix and eigenvector matrix of the matrix $\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}} = \boldsymbol{I} - \boldsymbol{\mathcal{L}}$ respectively. The matrix $\boldsymbol{U}\left(\frac{1}{T}\sum_{r=1}^T \boldsymbol{\Lambda}^r\right)\boldsymbol{U}^T$ has eigenvalues $\frac{1}{T}\sum_{r=1}^T \lambda_i^r, i = 1, \ldots, n$, where $\lambda_i, i = 1, \ldots, n$ are eigenvalues of the matrix $\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{A}\boldsymbol{D}^{-\frac{1}{2}}$, which can be seen as the transformation(a kind of filter) applied on the eigenvalue λ_i. This filter has the two properties (Fig. 8): (1) it prefers positive large eigenvalues; (2) the preference becomes stronger as the T (the window size) increases.

Besides, we also want to show the relationship between the DeepWalk matrix and the its corresponding normalized graph Laplacian. Since $\boldsymbol{D}^{-1}\boldsymbol{A} = \boldsymbol{I} - \boldsymbol{L}_{rw}$ and the eigenvectors and eigenvalues of $\boldsymbol{L}_{rw}$ and $\boldsymbol{\mathcal{L}}$ have the following relationship: if (λ_i, u_i) is an eigenvalue-eigenvector pair for normalized graph Laplacian $\boldsymbol{\mathcal{L}}$, then $\boldsymbol{L}_{rw}\boldsymbol{D}^{-\frac{1}{2}}u_i = \lambda_i\boldsymbol{D}^{-\frac{1}{2}}u_i$. Thus, we can write $\boldsymbol{L}_{rw}$ in the following form (Eq. 39) for $\boldsymbol{\mathcal{L}}$ can be written as $\boldsymbol{\mathcal{L}} = \boldsymbol{U}\Lambda\boldsymbol{U}^T$.

$$\boldsymbol{L}_{rw} = \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{U}\Lambda\boldsymbol{U}^T\boldsymbol{D}^{\frac{1}{2}} \tag{39}$$

For each r the term $(\boldsymbol{D}^{-1}\boldsymbol{A})^r$ can be written as $(\boldsymbol{I} - \boldsymbol{L}_{rw})^r = \boldsymbol{I} - C_r^1\boldsymbol{L}_{rw} + C_r^2\boldsymbol{L}_{rw}^2 + \cdots + (-1)^r\boldsymbol{L}_{rw}^r$, adding all the terms for different rs up and using a simple property for combinational number: $C_{n-1}^{m-1} + C_{n-1}^m = C_n^m$, we can have:

$$\begin{aligned}\sum_{r=1}^T (\boldsymbol{D}^{-1}\boldsymbol{A})^r = T\boldsymbol{I} - C_{T+1}^2\boldsymbol{L}_{rw} + C_{T+1}^3\boldsymbol{L}_{rw}^2\\ + \cdots + (-1)^T C_{T+1}^{T+1}\boldsymbol{L}_{rw}^T.\end{aligned} \tag{40}$$

Then viewing Eq. 40 as the binomial expansion with some transformation, it is equal to:

$$-\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{U}(-\boldsymbol{\Lambda}^{-1} + \boldsymbol{I} + \boldsymbol{\Lambda}^{-1}(\boldsymbol{I} - \boldsymbol{\Lambda})^{T+1})\boldsymbol{U}^T\boldsymbol{D}^{\frac{1}{2}} \tag{41}$$

Then the equivalent filter function in the spectral domain can be written as $h(\lambda) = -(-\lambda^{-1} + 1 + \lambda^{-1}(1-\lambda)^{T+1})$, which can be seen as a low-pass filter(Fig. 7, Right channel).

The matrix in the log term of DeepWalk's matrix(TABLE 2 DeepWalk) can be written as $-\boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{U}(-\boldsymbol{\Lambda}^{-1} + \boldsymbol{I} + \boldsymbol{\Lambda}^{-1}(\boldsymbol{I} - \boldsymbol{\Lambda})^{T+1})\boldsymbol{U}^T\boldsymbol{D}^{-\frac{1}{2}}$, where $\boldsymbol{U}$ is the eigenvector matrix for the normalized graph Laplacian.

The matrix factorized by LINE [120] is a trivial example. Ignoring constant items and taking the matrix in the log item, we can have the following form $\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{D}^{-1}$, which is equal to $\boldsymbol{D}^{-\frac{1}{2}}(I - \boldsymbol{\mathcal{L}})\boldsymbol{D}^{-\frac{1}{2}}$. $(I - \boldsymbol{\mathcal{L}})$ can be seen as the filter matrix with the filter function $h(\lambda) = 1 - \lambda$, and the multiplication items $\boldsymbol{D}^{-\frac{1}{2}}$ can be seen as the signal rescaling matrices.

7.2 Universal Attention Mechanism

Attention mechanisms are both explicitly and implicitly widely used in many algorithms.

For shallow embedding models, the positive sampling strategy, like sliding a window in the sampled node sequences obtained by different kind of random walks on the graph [42], [92] or just sample the adjacent nodes for each target node [120], can be seen as applying different attention weights on different nodes. Meanwhile, the negative sampling distribution given a positive sampling distribution for each node, which can also be seen as performing attention mechanism on different nodes, is proposed with the negative sampling strategy in [84] and further discussed in [153]. It is also closely related with the performance of the generated embedding vectors.

The positive samples and negative samples are then used in the optimization process and can help maintain the corresponding node proximities.

For GNNs, following the idea in an unsubmitted manuscript [5], by writing the aggregation function in the following universal form:

$$\boldsymbol{H} = \sigma(\mathcal{N}(\boldsymbol{\mathcal{L}}\boldsymbol{Q}\boldsymbol{H}\boldsymbol{W}), \tag{42}$$

where Q is a diagonal matrix, $\boldsymbol{L}$ is the matrix related with graph adjacency matrix, $\mathcal{N}(\cdot)$ is the normalization function, $\sigma(\cdot)$ is the non-linearity transformation function perhaps with post-propagation rescaling, the aggregation process of graph neural networks can be interpreted and separated as the following four stages: pre-propagation signal rescaling, propagate, re-normalization, and post-propagation signal rescaling.

The proposed paradigm for GNNs can help with the design of neural architecture search algorithms for GNNs. The search space can be designed based on the pre- and post-propagation rescaling matrices, the normalization function $\mathcal{N}(\cdot)$, and feature transformation function $\sigma(\cdot)$, and so on.

The pre-propagation signal rescaling process is usually used as the attention mechanism in many GCN variants, like the attention mechanism in GAT [130], whose pre-propagation rescaling scheme can be seen as multiplying a diagonal matrix $\boldsymbol{Q}$ to the right side of the matrix $\boldsymbol{\mathcal{L}} = \boldsymbol{A}+\boldsymbol{I}_N$, with each of the element in matrix $Q_{jj} = (\boldsymbol{W}\boldsymbol{H}\vec{q})_j$, where $\vec{q} \in \mathbb{R}^{d\times 1}$ is a trainable parameter. Besides, the post-propagation signal rescaling can also be combined with attention mechanism, which can be seen as left multiplying a diagonal matrix $\boldsymbol{P}$ to the propagated signal matrix $\mathcal{N}(\boldsymbol{L}\boldsymbol{Q})\boldsymbol{H}\boldsymbol{W}$. For example, the edge attention can be performed by setting $\boldsymbol{P} = \frac{1}{\boldsymbol{\mathcal{L}}\boldsymbol{Q}\vec{1}}$, k-hop edge attention can be performed in the similar form: $\boldsymbol{P} = \frac{1}{\boldsymbol{\mathcal{L}}^k\boldsymbol{Q}\vec{1}}$, matrix $\boldsymbol{P}$ for k-hop path attention can have the following form:

$$\boldsymbol{\mathcal{L}}' = \boldsymbol{\mathcal{L}}\boldsymbol{Q}_k\boldsymbol{\mathcal{L}}\boldsymbol{Q}_{k-1}...\boldsymbol{\mathcal{L}}\boldsymbol{Q}_1, \quad \vec{p} = \frac{1}{\boldsymbol{\mathcal{L}}'\vec{1}}. \tag{43}$$

Moreover, the aggregation process in RGCN [170], where features with large variance can be attenuated, can also be seen as the attention process applied on vertex feature variance and can help improve the robustness of GCN.

8 Optimization Methods

The optimization strategies we choose for a certain embedding model will affect its time and space efficiency and even the quality of embedding vectors. Some tricks and rethinkings can help reinforce the theoretical basis of related algorithms and further improve their expressiveness as well as reduce the time consumption. Thus, in this section, we will review optimization strategies of some typical embedding models, which are also of great importance in the design process.

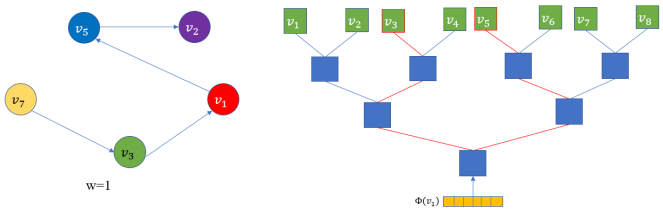

Fig. 9: An illustration for hierarchical softmax.Left Panel: A toy example of random walk with window size $w = 1$. Right Panel: Suppose we want to maximize the probability $\Pr(v_3|\Phi(v_1))$ and $\Pr(v_5|\Phi(v_1))$, they are factorized out over sequences of probability distributions corresponding to the paths starting at the root and ending at v_3 and v_5 respectively. Adapted from [92].

8.1 Optimizations for Random Walk Based Models

Optimization strategies for random walk based models can start with those for natural language embedding problems, which focus on word sequences. We refer these optimization problems to *Sequence Optimization Problems*, which can date bask to the classical N-gram models. But since the the calculation complexity will increase significantly as the word sequence's length grows [92], the problem is relaxed in [83] with two proposed models, CBOW and Skip-Gram.

Skip-Gram. We will focus on the Skip-Gram model, which uses the target node to predict context nodes by maximizing the following probability:

$$\Pr(\{v_{i-w},...,v_{i+w}\}/\{v_i\}|\Phi(v_i)), \tag{44}$$

where $\{v_{i-w},...,v_{i+w}\}$ are context vertices chosen by a sliding window over the node sequence with window size w. By applying the i.i.d. assumption and ignoring the order of context nodes, Eq. 44 can be factorized to

$$\prod_{i-w\leq j\leq i+w, j\neq i} \Pr(v_j|\Phi(v_i)). \tag{45}$$

The order independent assumption can better capture the "nearness" in graph structures [92].

However, the calculation for the probability $\Pr(v_j|\Phi(v_i))$ is also time-consuming ($O(|N|)$, N is the number of nodes in the graph), which is always in the form of softmax (Eq. 46). Thus, two strategies are proposed to alleviate this problem, that is hierarchical softmax [83] and negative sampling [84].

8.1.1 Hierarchical Softmax

In hierarchical softmax (Fig. 9), node embeddings that need optimizing are orgnized into a binary tree and then calculation for Eq. 46 is transferred into the multiplication of softmax in the path from root to the target context

vertex (Eq. 47). Then the time complexity can be reduced to $O(\log |V|)$.

$$\Pr(v_j|\Phi(v_i)) = \frac{e^{\alpha_i \cdot \beta_j}}{\sum_{v_k \in \mathcal{V}} e^{\alpha_i \cdot \beta_k}} \tag{46}$$

$$\Pr(u_k|\Phi(v_i)) = \prod_{l=1}^{\log |V|} \Pr(b_l|\Phi(v_j)) \tag{47}$$

8.1.2 Negative Sampling

An alternative for hierarchical softmax is Noise Contrastive Estimation (NCE) as proposed in [44], based on which the Negative Sampling strategy is introduced in [84] and has gain a wide application [42], [92], [120].

The objective of NCE can be shown to approximately maximize the log probability of the softmax [84]. Negative Sampling strategy is a simplified NCE concerned only with learning high-quality vector representations. Eq. 48, where w_I is the target word, w_O is the context word, w_i is the sampled negative word, ${v'}_{w_O}$ is the context embedding vector for word w_O and v_{w_I} is the word embedding vector for word w_I, $P_n(\cdot)$ is the negative sampling distribution, can be used to replace every $\log P(w_O|w_I)$ term in the Skip-Gram objective. Maximizing Eq. 48 can be seen as distinguishing the target word w_O from k negative samples drew from the noise distribution $P_n(w)$ using logistic regression.

$$\log \sigma({v'}_{w_O}^T v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)}[\log \sigma(-{v'}_{w_i}^T v_{w_I})] \tag{48}$$

The negative sampling distribution $P_n(w)$ is a free parameter, which is set to the unigram distribution $U(w)$ raised to $3/4$rd power (i.e., $U(w)^{3/4}/Z$) in [84] since it can outperform significantly the unigram and the uniform distributions.

Further Understanding for Negative Sampling (NS). However, neither the unigram distribution nor the $3/4$rd power of the unigram ditribution that is employed in word2vec [84] is the best negative sampling distribution. In [153], it is theoretically proved that the best negative sampling distribution should be positively but sub-linearly correlated to the corresponding positive sampling distribution.

Although the proposed design principle seems contrary to the intuition that nodes with high positive sampling rates should have lower negative sampling rates, NS distribution designed by following this theory can indeed improve the performance of existing algorithms.

In [69], it is proved that the optimal dot-product of the word-context pair's representations should take the form of PMI (Eq. 4) by replacing the NS distribution with the uniform distribution. Keeping the negative sampling distribution term, the close form of $\vec{u}^T\vec{v}$, where $\vec{v}$ is the representation vector of node v, is as follows:

$$\vec{u}^T\vec{v} = -\log\left(\frac{k \cdot p_n(u|v)}{p_d(u|v)}\right), \tag{49}$$

from which it can be seen that the positive and negative sampling distribution (p_d and p_n) have the same level influence on embedding results.

The effect of negative sampling can be further seen from the process of derivating the optimal NS distribution. The best NS distribution should minimize the gap between the theoretical solution of corresponding parameters: $\theta = [\vec{u}_0^T\vec{v}, \ldots, \vec{u}_{N-1}^T\vec{v}]$, and their empirical solution, since only limited positive and negative samples can be obtained in practice. One way is to minimize the mean square error between the theoretical and empirical results of $\vec{u}^T\vec{v}$:

$$\mathbb{E}[\|(\theta_T - \theta^*)_u\|^2] = \frac{1}{T}\left(\frac{1}{p_d(u|v)} - 1 + \frac{1}{kp_n(u|v)} - \frac{1}{k}\right). \tag{50}$$

It claims that nodes with high positive sampling rates should also be negatively sampled sufficiently, otherwise the expectation of the mean square error would increase.

By setting the NS rate for node u given node v positively but sub-linearly correlated to its positive sampling rate, i.e., $p_n(u|v) \propto p_d^{\alpha}(u|v)$, the dot-product $\vec{u}^T\vec{v}$ can also have the following monotonicity:

$$\begin{aligned} \vec{u}_i^T\vec{v} &= \log p_d(u_i|v) - \alpha \log p_d(u_i|v) + c \\ &> (1-\alpha)\log p_d(u_j|v) + c = \vec{u}_j^T\vec{v}, \end{aligned} \tag{51}$$

if $p_d(u_i|v) > p_d(u_j|v)$.

8.2 Optimization Strategies for Matrix Factorization

8.2.1 SVDs

SVD is a basic algorithm from linear algebra which is used to factorize matrix $\boldsymbol{M}$ into the product of three matrices $\boldsymbol{U} \cdot \boldsymbol{\Sigma} \cdot \boldsymbol{V}^T$, where $\boldsymbol{\Sigma}$ is the diagonal singular value matrix, $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal matrices with each of their row vector as an unit vector. The time complexity of the economic SVD on $m \times n$ matrix is $O(mn\min(m,n))$.

One property is that the optimal rank d approximation of matrix $\boldsymbol{M}$ can be obtained by taking $\boldsymbol{M}_d = \boldsymbol{U}_d \cdot \boldsymbol{\Sigma}_d \cdot \boldsymbol{V}_d^T$, where $\boldsymbol{\Sigma}_d$ is the diagonal matrix formed from the top d singular values, $\boldsymbol{U}_d$ and $\boldsymbol{V}_d$ are matrices formed by selecting the corresponding columns from $\boldsymbol{U}$ and $\boldsymbol{V}$. To be specific, $\boldsymbol{M}_d = \underset{\text{Rank}(\boldsymbol{M}')=d}{\arg\min} \|\boldsymbol{M}' - \boldsymbol{M}\|_2$.

Truncated Singular Value Decomposition (tSVD) aims to find the matrix factorization results $\boldsymbol{U}_d, \boldsymbol{\Sigma}_d, \boldsymbol{V}_d$ given a specific d.

r-tSVD. Randomized matrix method can be used to accelerate the basic tSVD algorithm. The idea is to find an orthogonal matrix $\boldsymbol{Q}$ by performing the iterative QR decomposition on the random projected matrix $\boldsymbol{H} = \boldsymbol{M}\boldsymbol{\Omega}$, where $\boldsymbol{\Omega}$ is a Gaussian random matrix with fewer columns than $\boldsymbol{M}$. Then tSVD can be preformed on the matrix $\boldsymbol{B} = \boldsymbol{Q}^T\boldsymbol{M}$, which will cost less time than performing tSVD directly on matrix $\boldsymbol{M}$, since $\boldsymbol{B}$ is a small matrix with less rows than $\boldsymbol{M}$. Then $\boldsymbol{U}_d = \boldsymbol{Q}\boldsymbol{U}_d$, $\boldsymbol{\Sigma}_d$, $\boldsymbol{V}_d$ are the approximated results of tSVD on the matrix $\boldsymbol{M}$, where $[\boldsymbol{U}_d, \boldsymbol{\Sigma}_d, \boldsymbol{V}_d] = \text{tSVD}(\boldsymbol{B})$. r-tSVD is used in the sparse matrix factorization stage in ProNE [163], whose time complexity is linear with respect to the number of edges in the graph (i.e., $O(|E|)$).

8.2.2 Acceleration of rPCA for Sparse Matrix

PCA is similar with tSVD when being used to factorize a specific matrix $\boldsymbol{M}$ [36]. Some methods can be used to accelerate the calculation process of the basic rPCA algorithm due to the sparse matrices' properties, like replacing the QR decomposition in the iteration process of rPCA with LU decomposition, replacing the Gaussian random matrix Ω with the uniform random matrix, and so on.

For example, two acceleration versions (*fr*PCA and its variant *fr*PCAt) of rPCA are proposed in [36] based on the modified power iteration scheme, which is faster than rPCA and can also provide more flexible trade-off between runtime and accuracy. Actually, they can accelerate ProNE by about 5-10 times.

8.2.3 Iterative Updating Strategy

Apart from tSVD and its variants, the target matrix can be obtained by iteratively updating corresponding matrices after the optimization objective is defined, like algorithmes for non-negative matrix factorization proposed in [67] and the joint non-negative matrix factorization method stated in [3]. Their applications can be seen in M-NMF [139].

8.3 Optimization Strategies for GNNs

Optimization strategies for GNNs focus on defining corresponding objective functions, which are then optimized under the stochastic gradient descent paradigm. The objective function can be supervised or unsupervised, which can be defined according to downstream tasks. In this subsection, we will only focus on the design of unsupervised objective functions.

Vertex Proximity Based. For example, the Laplacian regularization term (Eq. 52) is widely used in [8], [168], [173] based on the assumption that connected nodes tend to share the same label.

$$O_{\text{reg}} = \sum_{i,j} A_{ij} \|f(X_i) - f(X_j)\|^2 = f(\boldsymbol{X})^T \boldsymbol{L} f(\boldsymbol{X}) \quad (52)$$

However, as stated in [65], this assumption may restrict the modeling capacity, since graph edges are always half observed and are not necessarily encode node similarity.

The random walk-like loss function is used in [45] to encourage vertices with high co-occurrence frequency to have similar representations:

$$O(z_u) = -\log(\sigma(z_u^T z_v)) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log(\sigma(-z_u^T z_{v_n})), \quad (53)$$

where z_u is output representation of node u, v is a node that co-occurs near u in a fixed-length random walk.

Others. Graph information can also be utilized differently. For instance, Graph Infomax [131] and InfoGraph [116] maximize the mutual information between vertex representations and the pooled graph representation. In Variational Graph Auto-Encoder [64], vertex representations are used to reconstruct the graph structure.

9 Challenges and Problems

9.1 Shallow Embedding Modles

Random Walk Based Models. Despite the theoretical bound lying behind shallow embedding models, which give them connections with the matrix factorization models [97]. The equivalent can be satisfied only when the walk length goes to infinite, which leading that fact that random walk based models cannot outperform matrix factorization based methods, which has also been shown empirically [97]. Moreover, the sampling process is time-consuming if high order proximities are wished to be preserved [107].

Dataset	training ratio	0.1	0.3	0.5	0.7	0.9
PPI	DeepWalk	16.4	19.4	21.1	22.3	22.7
	LINE	16.3	20.1	21.5	22.7	23.1
	node2vec	16.2	19.7	21.6	23.1	24.1
	GraRep	15.4	18.9	20.2	20.4	20.9
	HOPE	16.4	19.8	21.0	21.7	22.5
	ProNE (SMF)	15.8	20.6	22.7	23.7	24.2
	ProNE	**18.2**	**22.7**	**24.6**	**25.4**	**25.9**
	($\pm\sigma$)	($\pm$0.5)	($\pm$0.3)	($\pm$0.7)	($\pm$1.0)	($\pm$1.1)
Wiki	DeepWalk	40.4	45.9	48.5	49.1	49.4
	LINE	**47.8**	50.4	51.2	51.6	52.4
	node2vec	45.6	47.0	48.2	49.6	50.0
	GraRep	47.2	49.7	50.6	50.9	51.8
	HOPE	38.5	39.8	40.1	40.1	40.1
	ProNE (SMF)	47.6	51.6	53.2	53.5	53.9
	ProNE	47.3	**53.1**	**54.7**	**55.2**	**57.2**
	($\pm\sigma$)	($\pm$0.7)	($\pm$0.4)	($\pm$0.8)	($\pm$0.8)	($\pm$1.3)
BlogCatalog	DeepWalk	36.2	39.6	40.9	41.4	42.2
	LINE	28.2	30.6	33.2	35.5	36.8
	node2vec	**36.3**	39.7	41.1	42.0	42.1
	GraRep	34.0	32.5	33.3	33.7	34.1
	HOPE	30.7	33.4	34.3	35.0	35.3
	ProNE (SMF)	34.6	37.6	38.6	39.3	39.0
	ProNE	36.2	**40.0**	**41.2**	**42.1**	**42.7**
	($\pm\sigma$)	($\pm$0.5)	($\pm$0.3)	($\pm$0.6)	($\pm$0.7)	($\pm$1.2)
Dataset	training ratio	0.01	0.03	0.05	0.07	0.09
DBLP	DeepWalk	49.3	55.0	57.1	57.9	58.4
	LINE	48.7	52.6	53.5	54.1	54.5
	node2vec	48.9	55.1	57.0	58.0	58.4
	GraRep	50.5	52.6	53.2	53.5	53.8
	HOPE	**52.2**	55.0	55.9	56.3	56.6
	ProNE (SMF)	50.8	54.9	56.1	56.7	57.0
	ProNE	48.8	**56.2**	**58.0**	**58.8**	**59.2**
	($\pm\sigma$)	($\pm$1.0)	($\pm$0.5)	($\pm$0.2)	($\pm$0.2)	($\pm$0.1)
Youtube	DeepWalk	38.0	40.1	41.3	42.1	42.8
	LINE	33.2	35.5	37.0	38.2	39.3
	ProNE (SMF)	36.5	40.2	41.2	41.7	42.1
	ProNE	**38.2**	**41.4**	**42.3**	**42.9**	**43.3**
	($\pm\sigma$)	($\pm$0.8)	($\pm$0.3)	($\pm$0.2)	($\pm$0.2)	($\pm$0.2)

Fig. 10: The classification performance in terms of Micro-F1 (%). Repring from [163].

Matrix Factorization Based Models. Factorizing matrices which encode high-order node proximities, structural information and other side information is guaranteed to obtain high-quality vertex embeddings. However, factorizing large, dense matrices is still time-consuming, though it has been proved that the factorizing process can be accelerated by random matrix theory when the matrix is sparse [36], [163]. And matrices being factorized are destined to be dense ones if high-order vertex proximities, and structural information are wished to be preserved.

Summary. Ways to solve problems of shallow embedding models mentioned above can be found in an embedding model ProNE [163], where a spare matrix is designed to be factorized to get raw node embeddings, then the spectral propagation process is applied on the obtained embeddings to make sure that the propagated embeddings are aware of high-order vertex and structural information. Since the matrix being factorized is a spare one, and the propagation process is also economical based on some mathematical properties, the model ProNE is a fast and effective model, combining advantages of different embedding models and also staying time-efficient (Fig. 10).

However, there are also some inherent problems lying in

shallow embedding models:

In the first place, the look-up embedding table in shallow neural embedding models and matrices in matrix factorization based embedding methods decide that those models are inherently transductive. Generating embedding vectors for new nodes needs to be calculated from scratch or it will take a long time. Moreover, if there are encoders in such models, they are relatively simple, and it is hard to incorporate vertex content in the encoding process. Even though the deep neural encoders are adopted in DNGR [15] and SDNE [135], features that are fed into the encoders are $|V|$-dimensional connectivity proximity vectors and the reconstruction architecture makes it hard to encode vertex content with connectivity information.

Those problems can be partly solved by Graph Neural Network based models.

9.2 Graph Neural Networks

Compared with shallow embedding models, GNNs present their potential in exploring graph structures [148], exploiting content information of nodes and edges [129], [130], being inductive [17], as well as dealing with heterogeneous structures better [17], [140], [157]. Although they can solve problems of shallow embedding models to some extend, generating node features that are more meaningful, better aware of node structural information and content information, there are some inherent problems lying in GNNs' architecture [29], [105], [167], [170]: (1) GNN models always tend to increase the number of GNN layers to capture information from high-order neighbours, which can lead to three problems:

- *over-fitting.* Since the signal propagation process is always coupled with non-linear transformation. Thus, increasing the number of layers will lead to increasing the number of parameters at the same time, which will raise the risk of over-fitting;
- *over-smoothing,* it has been proved that the convolution operation is essentially a special form of Laplacian smoothing [71]. The spatial form of the convolution operation centered at a target node is just linearly aggregating features from its neighbourhood. Thus, directly stacking many layers will make each node incorporate too many features from others but lose specific information of itself [18], [167], leading to the over-smoothing problem;
- *non-robust,* which comes from the propensity to over-fitting towards noisy part of input features [51] as the number of parameters increases.

(2) The propagation process in GNN models will always make each node too dependent on its neighbours, thus leading to the *non-robust* problem as stated above. Moreover, since edges in real-world networks are always only partially observed, even with false edges, it will make the model be more sensitive to adversarial attack on graph data and hard to learn true features for each node [170]. The attacker can indirectly attack the target node by just manipulating long-distance neighbours [178]. (3) Moreover, different from shallow embedding models, like random walk based models, matrix factorization based models, which rely more on graph structures the advantages that GNNs have over shallow embedding like easily incorporating with node and edge features, labels, also make GNNs rely on labels too much. Thus it is hard for GNNs to perform well when there are only scarce labels available. (4) Designing the best GNN for a certain task requires manual tuning to adjust the network architecture and hyper-parameters, such as the attention mechanism in the neighbourhood aggregation process, the activation functions and the number of hidden dimensions. Apart from those problems regard to the architecture of GNN models mentioned above, huge parameters in GNNs that require tuning also lead to the heavy manual labors in GNNs' design process, which is also a common problem in deep learning community.

Proposed Solutions. Problems stated above are also opportunities to help us design better models. There are many works aiming to tackle the problems for graph neural networks [29], [105], [133], [167], [170]. We will discuss some basic insights later.

9.2.1 Graph Regularization

Random Propagation. Just as the effectiveness of Dropout for the training process of neural networks, which can be seen an adaptive L_2 regularization strategy, introducing random factors in the feature propagation process of GNNs [35], [105] has been proved as an effective way to help improve the model's robustness, alleviate over-fitting and over-smoothing problems. In DropEdge [105], a certain rate of edges will be droped out in each training epoch, while in DropNode [35], a certain rate of nodes' features will be droped out. Both of them can help block the some information propagation ways, reduce vertices' dependency on certain neighbours and thus can help the model go deep, alleviate the over-smoothing problem and improve the robustness.

Compared with DropEdge, dropping out the entire features of some nodes in the training epoch can further help decouple the feature propagation and transformation process, which are closely connected in the deterministic GNN architectures. Moreover, the consistency loss is introduced to help force the model output similar predictions with different augmentations (with different nodes randomly dropped) as input, whose effectiveness is empirically proved.

Data Augmentation. Data augmentation can help increase data varieties, thus helping avoid over-fitting problems.

For example, a graph data interpolation method is used in GraphMix [133] to augment data. Moreover, the random propagation [35], [105] can also be seen as a data augmentation technique since random deformed copies of the original graph can increase the randomness and diversity of the input data. Thus, it can help increase data varieties and help avoid over-fitting problems.

Moreover,several data augmentation techniques are proposed in [141] to improve the performance of GNNs during inference time. A consistency loss is introduced and minimized to make the prediction of node labels under different augmentations consistent with each other:

$$\mathcal{L}_C = \frac{1}{|\mathcal{V}|}\sum_{i\in\mathcal{V}} \mathcal{D}_{KL}\left(p\left(y|\mathcal{G}_i, x_{j\,j\in\mathcal{V}_i}, \tilde{\Theta}\right) \| p\left(y|\hat{\mathcal{G}}_i, \hat{x}_{j\,j\in\hat{\mathcal{V}}_i}, \tilde{\Theta}\right)\right), \tag{54}$$

where $\mathcal{G}_i$ is the subgraph of node v_i, which corresponds to the receptive field of node v_i, $\mathcal{V}_i$ is the node set of the subgraph,

x_i denotes the attributes of node v_i, Θ is the parameter set. $\hat{\mathcal{G}}_i$ is the subgraph of node v_i after augmentation, $\tilde{\Theta}$ is a fixed copy of the current parameter Θ, indicating that the gradient is not propagated through Θ.

In addition, a *"parallel universe"* data augmentation scheme is introduced to conduct data augmentation for different nodes individually and separately, since the modification on the subgraph of a node v_i will influence the input features of other nodes.

Adversarial Virtual attack. Performing adversarial virtual attacks and introducing corresponding loss term to the optimization objective can help improve the smoothness of output vectors with respect to perturbation around the local structure [29].

BVAT. In [29], two virtual attack strategies are proposed to improve the adversarial virtual attack techniques in VAT [85], to get close to worst-case attack for each node. The first one is S-BVAT, where a set of nodes $\mathcal{V}_S \subset \mathcal{V}$ is sampled. The receptive field for each node in the set will not overlap with other nodes' receptive fields. Then the regularization term for training is the average LDS loss over nodes in $\mathcal{V}_S$:

$$\mathcal{R}_{vadv}(\mathcal{V}_S, \mathcal{W}) = \frac{1}{B} \sum_{u \in \mathcal{V}_S} \text{LDS}(\boldsymbol{X}_u, \mathcal{W}, r_{vadv,u}), \tag{55}$$

where $\mathcal{W}$ is the trainable parameters, $\boldsymbol{X}_u$ is the input feature matrix of all nodes in node u's receptive field. Then, $\mathcal{R}_{vadv}(\mathcal{V}_\mathcal{S}, \mathcal{W})$ can be seen as an approximate estimation of $\mathcal{R}_{vadv}(\mathcal{V}, \mathcal{W})$. The second one is called O-BVAT, where the average LDS loss with respect to the whole perturbation matrix $\boldsymbol{R}$ corresponding to the whole feature matrix $\boldsymbol{X}$ is maximized by solving:

$$\max_{\boldsymbol{R}} \frac{1}{N} \sum_{u \in \mathcal{V}} D_{KL}(p(y|\boldsymbol{X}_u, \hat{\mathcal{W}}) \| p(y|\boldsymbol{X}_u + \boldsymbol{R}_u, \mathcal{W})) - \gamma \cdot \|\boldsymbol{R}\|_F^2, \tag{56}$$

where $\|\boldsymbol{R}\|_F$ is the Frobenius norm of $\boldsymbol{R}$ to make the optimal perturbation have a small norm and γ is a hyper-parameter to balance the loss terms. $\boldsymbol{R}$ is optimized for T iterations, which is more powerful than one-step gradient-based methods.

RGCN. Different from VAT and BVAT, which introduce perturbation to the training process deliberately, a robust GCN model (RGCN) is introduced to improve model's robustness by reducing aggregation of features with large variance. The basic idea of RGCN is replacing direct message propagation processes in GCN and message passing GNN models with the convolutional operation on Gaussian distributions. The technique to avoid attacking on GNNs models was an attention mechanism, which assigns different weights to features according their variances since larger variances may indicate more uncertainties in the latent representations and larger probability of having been attacked [170]. Meanwhile, the "reparameterization trick" [30] is used to optimize the loss function using back propagation and an explicit regularization term was used to constrain the latent representations in the first layer to Gaussian distribution:

$$\mathcal{L}_{reg1} = \sum_{i=1}^{N} KL(\mathcal{N}(\boldsymbol{\mu}_i^{(1)}, \text{diag}(\boldsymbol{\sigma}_i^{(1)})) \| \mathcal{N}(0, \boldsymbol{I})), \tag{57}$$

where $\boldsymbol{\mu}_i^{(1)}$ is the mean vector for i-th feature in the first layer, $\boldsymbol{\sigma}_i^{(1)}$ is the variance vector for i-th feature in the first layer, $\mathcal{N}(\mu, \sigma)$ is the normal distribution with μ as its mean vector and σ as its covariance matrix, and $KL(\cdot\|\cdot)$ is the KL-divergence between two distributions.

Category	Method	Cora	Citeseer	Pubmed
Graph Convolution	GCN [27]	81.5	70.3	79.0
	GAT [42]	83.0±0.7	72.5±0.7	79.0 ±0.3
	Graph U-Net [15]	84.4±0.6	73.2±0.5	79.6±0.2
	MixHop [2]	81.9 ± 0.4	71.4±0.8	80.8±0.6
	GMNN [36]	83.7	72.9	81.8
	GraphNAS [16]	84.2±1.0	73.1±0.9	79.6±0.4
Regularization based GCNs[2]	VBAT [13]	83.6 ± 0.5	74.0 ± 0.6	79.9 ± 0.4
	G^{3}NN [31]	82.5 ±0.2	74.4±0.3	77.9 ± 0.4
	GraphMix [43]	83.9 ± 0.6	74.5 ±0.6	81.0 ± 0.6
	DropEdge [37]	82.8	72.3	79.6
Sampling based GCNs[3]	GraphSAGE [22]	78.9±0.8	67.4±0.7	77.8±0.6
	FastGCN [9]	81.4±0.5	68.8±0.9	77.6±0.5
Our methods	GRAND	**85.4±0.4**	**75.4±0.4**	**82.7±0.6**
	GRAND_GCN	84.5±0.3	74.2±0.3	80.0±0.3
	GRAND_GAT	84.3±0.4	73.2± 0.4	79.2±0.6
	GRAND_dropout	84.9±0.4	75.0±0.3	81.7±1.0

Fig. 11: Experiment results of several GNN models in node classification tasks on three public datasets. "Our methods" denotes GRAND model [35]. Reprint from [35].

Summary. Fig. 11 taken from [35] summarizes and compares the experiment results with regard to different GNN models' effectiveness on several public datasets in node classification task. Moreover, it is also shown that graph regularization techniques can address non-robust, over-smoothing and over-fitting problems to some extend, which demonstrates the effectiveness of these methods as a way to improve GNN models.

9.2.2 Self-supervised Learning for GNNs

Self-supervised learning (SSL) is an effective technique that is widely adopted in NLP and computer vision domain to extract expressive representations for words, sentences and images.

Recently, SSL on graph data has attracted a lot of interests, especially for GNNs.

Self-supervised learning for GNNs focuses on defining proper pretext tasks and training techniques.

Basic pretext tasks are defined on characteristics of graph data, such as attribute completion [57], [61], [164], [171], edge prediction [57], [61], [171]. Some models try to define pretext tasks based on graph topology, like vertex distance prediction [61], context prediction [57], graph structure recovery [164], pair-wise proximity prediction [91], and so on. Moreover, Graph Contrastive Coding (GCC) is proposed in [95] based on the contrastive learning paradigm [20], [48]. A generative pre-training model (GPT-GNN) is proposed in [58] to pre-train GNNs based on vertex attributes generation and edge generation tasks. Labels are also combined in the pretext tasks in [61] to align with down stream tasks.

As for training techniques, pre-training under pretext tasks and fine-tuning on down stream tasks is a widely used paradigm [57], [164]. Besides, self-training is also a effective training technique and is used in [155]. It pre-trains a model

in the labeled data and then uses it to generate pseudo-labels for unlabeled data, which are included into labeled data for the next round of training [175]. Moreover, the multi-task training can also be defined to combine self-supervised pretext tasks and down stream supervised tasks.

In conclusion, self-supervised learning broadens the idea of training GNNs and expands our exploration space.

9.2.3 Neural Architecture Search for GNNs

Aiming to alleviate the heavy manual tuning labors lying in GNNs' design process, the developing of neural architecture search (NAS) on GNN models focus on how to automatically design GNN architectures [169]. Designing the best GNN for a certain task requires manual tuning to adjust the network architecture and hyper-parameters, such as the attention mechanism in the neighbourhood aggregation process, the activation functions and the number of hidden dimensions. Thus, the developing of neural architecture search (NAS) on GNN models focus on how to automatically design GNN architectures [169].

Although neural architecture search has rose a lot of interests [34] and has outpreformed handcrafted ones at many other domains or tasks (e.g., image classification [176], [177], image generation [136]), it is not a trivial thing to generate such strategies to design auto-GNN models as stated in [169].

A neural architecture search model should define the searching space, a controller which is used to judge the most prosperous models and are also supposed to do efficient parameter sharing, which can avoid training a new model from scratch. However, as for the search space, search space for GNNs is different from those of existing NAS work. For example, the search space in CNN models involves the kernel size and the number of convolutional layers, while in GNN models, the search space is defined on the activation functions, aggregation strategies, etc. Moreover, the traditional controller is inefficient to discover the potentially well-performed GNN architectures due to the inherent property of GNN models which determines the performance of GNNs varies significantly with slight architecture modification. The problem partly comes from the difficulty of evaluating GNN models, which reveals the importance of understanding GNN models deeply and defining good evaluation methods as a recent paper does [126] in NLP domain. Thirdly, the widely adopted techniques such as the parameter sharing is not suitable for heterogeneous GNN models which will have different weight shapes or output statistics [43].

Thus, in [169], an efficient controller is designed based on the property of GNNs and define the concept of heterogeneous graph neural networks and permit parameter sharing only between two homogeneous GNNs. Given the best architecture at the time, the architecture modification is realized by the following three steps: (1) For each class, remove it and treat the remaining GNN architecture as the current stage. Then, use a RNN encoder to generate actions of this class for each layer. (2) Use an action guider to sample a list of classes to be modified based on the decision entropy of each class. The decision entropy of class c is defined as follows:

$$E_c \triangleq \sum_{i=1}^{n}\sum_{j=1}^{m_c} -P_{ij}\log P_{ij}, \tag{58}$$

where n is the number of layers, m_c is the number of actions that can be chosen from in class c, $\vec{P}_i$ is the action probability distribution in layer i of class c. (3) Modify the GNN architecture of each class in the generated class list.

Besides, regrading the network architecture search of deep convolutional neural networks in the field of computer vision, a recent paper [100] moves from the traditional design paradigm which focuses on designing individual network instances tuned to a single setting to searching design spaces [99] which aims to discover general and interpretable design principles.

The discovery of the general design principles can better guide the future design of individual well-performed networks, which is also a meaningful direction that the neural architecture search for GNNs can focus on.

10 Future Development Directions

In Section 9 we summarize the existing challenges and problems in both shallow and GNN based embedding models. Although there have already been some works trying to solve those challenges, they still cannot be addressed completely and elegantly. Designing an embedding algorithm which is both effective and efficient is not an easy thing.

In this section, we will further review challenges of designing embedding algorithms on real-world networks and also some promising developing directions, hoping to be helpful for the future development.

Dynamic. Networks in the real world are always evolving, such as new users (new vertices) in social networks, new citations (new edges) in citation networks. Although there are some works trying to develop embedding algorithms for evolving networks, there are also many underlying challenges in such researches since the corresponding embedding algorithms should deal with the changing networks and be able to update embedding vectors efficiently [162].

Robustness. In the past two years, attacks and defenses on graph data have attracted widespread attention [117]. It is shown that whether unsupervised models or models with supervision from downstream tasks can be fooled even by unnoticeable perturbations [11], [178]. Moreover, edges and vertices in real-world networks are always uncertain and noisy [162]. It is crucial to learn representations that are robust with respect to those uncertainties and possible adversarial attacks on graphs. Some universal techniques are widely adopted to improve the embedding robustness, like using the adversarial attack as a regularizer (e.g., ANE [26], ATGA [90] VBAT [29]), modeling graph structure using probability distribution methods (e.g., URGE [55] and RGCN [170]).

Generating Real-World Networks. Generating real-world networks is a meaningful thing. For example, generating molecular graphs can help with the drug design and discovery process [112], [154], generating real-world citation networks or social networks can help design more reasonable benchmarks and defend adversarial attacks. However, designing efficient density estimation and generating models on graphs is a challenging thing due to graphs' inherent combinational property and worth researching on.

Reasoning Ability of GNNs. Recently, there are also some works digging into the reasoning ability of GNNs. They try to explore GNNs' potential in executing algorithms [132], [149], or focus on the logical expressiveness of GNNs [6], [166]. Both of them can help us better understand the internal mechanism of GNNs and thus help promote the development of GNN models to generate more expressive and powerful vertex embeddings. There are also attempts trying to combine GNNs with the statistical relational learning (SRL) problem, which have been well explored in Markov networks [121], Markov logic networks [103] usually based on conditional random field, to help GNNs model relational data [98].

REFERENCES

[1] C. C. Aggarwal. An introduction to social network data analytics. In *Social network data analytics*. 2011.

[2] N. K. Ahmed, R. Rossi, J. B. Lee, T. L. Willke, R. Zhou, X. Kong, and H. Eldardiry. Learning role-based graph embeddings, 2018.

[3] Z. Akata, C. Thurau, and C. Bauckhage. Non-negative matrix factorization in multimodality data for segmentation and label prediction. 2011.

[4] N. Alon, C. Avin, M. Koucky, G. Kozma, Z. Lotker, and M. R. Tuttle. Many random walks are faster than one, 2007.

[5] Anonymous. Understanding graph convolutional networks as signal rescaling. 2020.

[6] P. Barceló, E. V. Kostylev, M. Monet, J. Pérez, J. Reutter, and J. P. Silva. The logical expressiveness of graph neural networks. In *ICLR*, 2020.

[7] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, 2002.

[8] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 2006.

[9] Y. Bengio, O. Delalleau, and N. Le Roux. Label propagation and quadratic criterion. 2006.

[10] S. Bhagat, G. Cormode, and S. Muthukrishnan. Node classification in social networks. In *Social network data analytics*. 2011.

[11] A. Bojchevski and S. Günnemann. Adversarial attacks on node embeddings via graph poisoning. In *ICML*, 2019.

[12] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *arXiv*, 2013.

[13] H. Cai, W. V. Zheng, and C.-C. K. Chang. A comprehensive survey of graph embedding: Problems, techniques and applications. *TKDE*, 2018.

[14] S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *CIKM*, 2015.

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

[16] Y. Carl, X. Yuxin, Z. Yu, S. Yizhou, and H. Jiawei. Heterogeneous network representation learning: Survey, benchmark, evaluation, and beyond. 2020.

[17] Y. Cen, X. Zou, J. Zhang, H. Yang, J. Zhou, and J. Tang. Representation learning for attributed multiplex heterogeneous network. *KDD*, 2019.

[18] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view, 2019.

[19] J. Chen, T. Ma, and C. Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling, 2018.

[20] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. *ICML*, 2020.

[21] T. Chen and Y. Sun. Task-guided and path-augmented heterogeneous network embedding for author identification. *WSDM*, 2017.

[22] F. R. Chung and F. C. Graham. *Spectral graph theory*. American Mathematical Soc., 1997.

[23] K. Church and P. Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 1990.

[24] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction to algorithms, 2nd edition. 2001.

[25] I. Dagan, F. Pereira, and L. Lee. Similarity-based estimation of word cooccurrence probabilities. *arXiv*, 1994.

[26] Q. Dai, Q. Li, J. Tang, and D. Wang. Adversarial network embedding. 2017.

[27] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016.

[28] C. Deng, Z. Zhao, Y. Wang, Z. Zhang, and Z. Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. *ICLR*, 2020.

[29] Z. Deng, Y. Dong, and J. Zhu. Batch virtual adversarial training for graph convolutional networks. *arXiv*, 2019.

[30] C. Doersch. Tutorial on variational autoencoders. *arXiv*, 2016.

[31] Y. Dong, V. N. Chawla, and A. Swami. metapath2vec: Scalable representation learning for heterogeneous networks. *KDD*, 2017.

[32] Y. Dong, Z. Hu, K. Wang, Y. Sun, and J. Tang. Heterogeneous network representation learning. *IJCAI*, 2020.

[33] C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec. Spectral graph wavelets for structural role similarity in networks. *CoRR*, 2017.

[34] T. Elsken, H. J. Metzen, and F. Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 2019.

[35] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang. Graph random neural network. *NIPS*, 2020.

[36] X. Feng, Y. Xie, M. Song, W. Yu, and J. Tang. Fast randomized pca for sparse data, 2018.

[37] T.-Y. Fu, W.-C. Lee, and Z. Lei. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. *CIKM*, 2017.

[38] S. Gao, L. Denoyer, and P. Gallinari. Temporal link prediction by integrating content and structure information. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, 2011.

[39] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

[40] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica. Graphx: Graph processing in a distributed dataflow framework. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, 2014.

[41] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014.

[42] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016.

[43] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun. Single path one-shot neural architecture search with uniform sampling. *CVPR*, 2019.

[44] M. U. Gutmann and A. Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 2012.

[45] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, 2017.

[46] D. K. Hammond, P. Vandergheynst, and R. Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 2011.

[47] Z. S. Harris. Distributional structure. *Word*, 1954.

[48] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. *CVPR*, 2019.

[49] Y. He, Y. Song, J. Li, C. Ji, J. Peng, and H. Peng. Hetespaceywalk: A heterogeneous spacey random walk for heterogeneous information network embedding. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019.

[50] K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li. Rolx: Structural role extraction mining in large graphs. In *KDD*, 2012.

[51] N. Hoang and T. Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv*, 2019.

[52] M. E. Hochstenbach. A jacobi-davidson type method for the generalized singular value problem. *Linear Algebra Its Applications*, 2009.

[53] H. Hong, H. Guo, Y. Lin, X. Yang, Z. Li, and J. Ye. An attention-based graph neural network for heterogeneous structural learning. *AAAI*, 2020.

[54] B. Hu, Y. Fang, and C. Shi. Adversarial learning on heterogeneous information networks. 2019.

[55] J. Hu, R. Cheng, Z. Huang, Y. Fang, and S. Luo. On embedding uncertain graphs. In *CIKM*, 2017.

[56] P. Hu and C. W. Lau. A survey and taxonomy of graph sampling. *CoRR*, 2013.

[57] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec. Strategies for pre-training graph neural networks. In *ICLR*, 2020.

[58] Z. Hu, Y. Dong, K. Wang, K.-W. Chang, and Y. Sun. Gpt-gnn: Generative pre-training of graph neural networks. In *KDD*, 2020.

[59] W. Huang, T. Zhang, Y. Rong, and J. Huang. Adaptive sampling towards fast graph representation learning, 2018.

[60] Y. Jacob, L. Denoyer, and P. Gallinari. Learning latent representations of nodes for classifying in heterogeneous social networks. *WSDM*, 2014.

[61] W. Jin, T. Derr, H. Liu, Y. Wang, S. Wang, Z. Liu, and J. Tang. Self-supervised learning on graphs: Deep insights and new direction. *arXiv*, 2020.

[62] Y. Jin, g. song, and C. Shi. Gralsp: Graph neural networks with local structural patterns. *AAAI*, 2020.

[63] Y. Kim. Convolutional neural networks for sentence classification. *arXiv*, 2014.

[64] T. N. Kipf and M. Welling. Variational graph auto-encoders. *arXiv*, 2016.

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

[66] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[67] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *NIPS*, 2001.

[68] J. Leskovec and C. Faloutsos. Sampling from large graphs. *KDD*, pages 631–636, 2006.

[69] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In *NIPS*, 2014.

[70] J. Li, J. Zhu, and B. Zhang. Discriminative deep random walk for network classification. In *ACL*, 2016.

[71] Q. Li, Z. Han, and X.-M. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 2018.

[72] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, 2003.

[73] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, 2015.

[74] Z. Liu, V. W. Zheng, Z. Zhao, F. Zhu, K. C.-C. Chang, M. Wu, and J. Ying. Distance-aware dag embedding for proximity search on heterogeneous graphs. In *AAAI*, 2018.

[75] F. Lorrain and H. C. White. Structural equivalence of individuals in social networks. *Social Networks*, 1977.

[76] L. Lovász et al. Random walks on graphs: A survey. *Combinatorics, Paul erdos is eighty*, 1993.

[77] B. LowY et al. Distributedgraphlab: aframeworkformachine-learninganddata mininginthecloud. *ProceedingsoftheVLDBEndowment*, 2012.

[78] L. Lü and T. Zhou. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 2011.

[79] F. D. Malliaros and M. Vazirgiannis. Clustering and community detection in directed networks: A survey. *Physics reports*, 2013.

[80] Marinka, Zitnik, Rok, Sosič, Marcus, W, Feldman, Jure, and Leskovec. Evolution of resilience in protein interactomes across the tree of life. *Proceedings of the National Academy of Sciences of the United States of America*, 2019.

[81] I. F. Martins, A. L. Teixeira, L. Pinheiro, and A. O. Falcao. A bayesian approach to in silico blood-brain barrier penetration modeling. *Journal of Chemical Information and Modeling*, 2012.

[82] S. Micali and A. Z. Zhu. Reconstructing markov processes from independent and anonymous experiments. *Discrete Applied Mathematics*, 2016.

[83] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space, 2013.

[84] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[85] T. Miyato, S.-i. Maeda, M. Koyama, and S. Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *PAMI*, 2018.

[86] S. A. Myers, A. Sharma, P. Gupta, and J. Lin. Information network or social network? the structure of the twitter follow graph. In *WWW*, 2014.

[87] N. Natarajan and I. S. Dhillon. Inductive matrix completion for predicting gene–disease associations. *Bioinformatics*, 2014.

[88] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In *KDD*, 2016.

[89] L. PAGE. The pagerank citation ranking : Bringing order to the web, online manuscript. *http://www-db.stanford.edu/backrub/pageranksub.ps*, 1998.

[90] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang. Adversarially regularized graph autoencoder for graph embedding. 2018.

[91] Z. Peng, Y. Dong, M. Luo, X.-M. Wu, and Q. Zheng. Self-supervised graph representation learning via global context prediction. *arXiv*, 2020.

[92] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *KDD*, 2014.

[93] B. Perozzi, V. Kulkarni, H. Chen, and S. Skiena. Don't walk, skip! online learning of multi-scale network embeddings, 2016.

[94] N. Pizarro. Structural identity and equivalence of individuals in social networks: Beyond duality. *International Sociology*, 2007.

[95] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang. GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training. *KDD*, 2020.

[96] J. Qiu, Y. Dong, H. Ma, J. Li, C. Wang, K. Wang, and J. Tang. Netsmf: Large-scale network embedding as sparse matrix factorization. *WWW*, 2019.

[97] 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 *WSDM*, 2018.

[98] M. Qu, Y. Bengio, and J. Tang. Gmnn: Graph markov neural networks. 2019.

[99] I. Radosavovic, J. Johnson, S. Xie, W.-Y. Lo, and P. Dollaacute;r. On network design spaces for visual recognition. *ICCV*, pages 1882–1890, 2019.

[100] I. Radosavovic, P. R. Kosaraju, R. Girshick, K. He, and P. Dollár. Designing network design spaces. *CVPR*, 2020.

[101] F. B. Ribeiro and F. D. Towsley. Estimating and sampling graphs with multidimensional random walks. *internet measurement conference*, pages 390–403, 2010.

[102] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo. struc2vec: Learning node representations from structural identity. In *KDD*, 2017.

[103] M. Richardson and P. Domingos. Markov logic networks (vol 62, pg 107, 2006). *Machine Learning*, 2006.

[104] S. C. Ritchie, S. Watts, L. G. Fearnley, K. E. Holt, G. Abraham, and M. Inouye. A scalable permutation approach reveals replication and preservation patterns of network modules in large datasets. *Cell systems*, 2016.

[105] Y. Rong, W. Huang, T. Xu, and J. Huang. Dropedge: Towards deep graph convolutional networks on node classification, 2019.

[106] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 2000.

[107] B. Rozemberczki and R. Sarkar. Fast sequence-based embedding with diffusion graphs, 2020.

[108] M. Schlichtkrull, N. T. Kipf, P. Bloem, v. d. R. Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. *ESWC*, 2018.

[109] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 2008.

[110] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 2011.

[111] C. Shi, B. Hu, X. W. Zhao, and S. P. Yu. Heterogeneous information network embedding for recommendation. *TKDE*, 2019.

[112] C. Shi, M. Xu, Z. Zhu, W. Zhang, M. Zhang, and J. Tang. Graphaf: a flow-based autoregressive model for molecular graph generation. *ICLR*, 2020.

[113] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE*, 2013.

[114] D. I. Shuman, B. Ricaud, and P. Vandergheynst. Vertex-frequency analysis on graphs. *Applied and Computational Harmonic Analysis*, 2016.

[115] I. D. Shuman, B. Ricaud, and P. Vandergheynst. Vertex-frequency analysis on graphs. *Applied and Computational Harmonic Analysis*, 2013.

[116] F.-Y. Sun, J. Hoffmann, and J. Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv*, 2019.

[117] L. Sun, Y. Dou, C. Yang, J. Wang, P. S. Yu, and B. Li. Adversarial attack and defense on graph data: A survey. *arXiv*, 2020.

[118] J. Tang, J. Liu, M. Zhang, and Q. Mei. Visualizing large-scale and high-dimensional data. In *WWW*, 2016.

[119] J. Tang, M. Qu, and Q. Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks. *KDD*, 2015.

[120] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *WWW*, 2015.

[121] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. *Proc.conf.on Uncertainty in Artificial Intelligence*, 2012.

[122] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler. Convolutional learning of spatio-temporal features. In *European conference on computer vision*, 2010.

[123] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 2000.

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

[125] N. Tremblay and P. Borgnat. Graph wavelets for multiscale community mining. *IEEE Transactions on Signal Processing*, 2014.

[126] M. R. Tulio, W. Tongshuang, G. Carlos, and S. Sameer. Beyond accuracy: Behavioral testing of nlp models with checklist. *ACL*, 2020.

[127] P. D. Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In *European conference on machine learning*, 2001.

[128] P. D. Turney and P. Pantel. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 2010.

[129] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, 2017.

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

[131] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm. Deep graph infomax. In *ICLR*, 2019.

[132] P. Veličković, R. Ying, M. Padovano, R. Hadsell, and C. Blundell. Neural execution of graph algorithms. In *ICLR*, 2020.

[133] V. Verma, M. Qu, A. Lamb, Y. Bengio, J. Kannala, and J. Tang. Graphmix: Regularized training of graph neural networks for semi-supervised learning. *arXiv*, 2019.

[134] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 2007.

[135] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *KDD*, 2016.

[136] H. Wang and J. Huan. Agan: Towards automated design of generative adversarial networks. *CoRR*, 2019.

[137] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo. Graphgan: Graph representation learning with generative adversarial nets. *TKDE*, 2017.

[138] S. Wang, J. Tang, F. Morstatter, and H. Liu. Paired restricted boltzmann machine for linked data. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2016.

[139] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang. Community preserving network embedding. In *AAAI*, 2017.

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

[141] Y. Wang, W. Wang, Y. Liang, Y. Cai, J. Liu, and B. Hooi. Nodeaug: Semi-supervised node classification with data augmentation. KDD, 2020.

[142] Z. Wang, H. Liu, Y. Du, Z. Wu, and X. Zhang. Unified embedding model over heterogeneous information network for personalized recommendation. *IJCAI*, 2019.

[143] B. Weisfeiler and A. A. Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 1968.

[144] D. B. West. Introduction to graph theory, 2nd ed. *Networks*, 2001.

[145] F. Wu, T. Zhang, A. H. d. Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger. Simplifying graph convolutional networks. *ICML*, 2019.

[146] M. Xie, H. Yin, H. Wang, F. Xu, W. Chen, and S. Wang. Learning graph-based poi embedding for location-based recommendation. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2016.

[147] B. Xu, H. Shen, Q. Cao, Y. Qiu, and X. Cheng. Graph wavelet neural network. *ICLR*, 2019.

[148] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *ICLR*, 2019.

[149] K. Xu, J. Li, M. Zhang, S. S. Du, K.-i. Kawarabayashi, and S. Jegelka. What can neural networks reason about? *ICLR*, 2020.

[150] S. Yan, D. Xu, B. Zhang, H. Zhang, Q. Yang, and S. Lin. Graph embedding and extensions: A general framework for dimensionality reduction. *PAMI*, 2007.

[151] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Chang. Network representation learning with rich text information. In *IJCAI*, 2015.

[152] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 2015.

[153] Z. Yang, M. Ding, C. Zhou, H. Yang, J. Zhou, and J. Tang. Understanding negative sampling in graph representation learning, 2020.

[154] J. You, B. Liu, R. Ying, S. V. Pande, and J. Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *NeurIPS*, 2018.

[155] Y. You, T. Chen, Z. Wang, and Y. Shen. When does self-supervision help graph convolutional networks? *arXiv*, 2020.

[156] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna. Graphsaint: Graph sampling based inductive learning method. *ICLR*, 2020.

[157] C. Zhang, D. Song, C. Huang, A. Swami, and V. N. Chawla. Heterogeneous graph neural network. 2019.

[158] C. Zhang, K. Zhang, Q. Yuan, H. Peng, Y. Zheng, T. Hanratty, S. Wang, and J. Han. Regions, periods, activities: Uncovering urban dynamics via cross-modal representation learning. In *WWW*, 2017.

[159] D. Zhang, J. Yin, X. Zhu, and C. Zhang. Collective classification via discriminative matrix factorization on sparsely labeled networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, 2016.

[160] D. Zhang, J. Yin, X. Zhu, and C. Zhang. Homophily, structure, and content augmented network representation learning. In *ICDM*, 2016.

[161] D. Zhang, J. Yin, X. Zhu, and C. Zhang. User profile preserving social network embedding. In *IJCAI*, 2017.

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

[163] J. Zhang, Y. Dong, Y. Wang, J. Tang, and M. Ding. Prone: fast and scalable network representation learning. In *IJCAI*, 2019.

[164] J. Zhang, H. Zhang, L. Sun, and C. Xia. Graph-bert: Only attention is needed for learning graph representations. *arXiv*, 2020.

[165] X. Zhang, W. Chen, and H. Yan. Tline: Scalable transductive network embedding. In *Asia Information Retrieval Symposium*, 2016.

[166] Y. Zhang, X. Chen, Y. Yang, A. Ramamurthy, B. Li, Y. Qi, and L. Song. Can graph neural networks help logic reasoning?, 2019.

[167] L. Zhao and L. Akoglu. Pairnorm: Tackling oversmoothing in gnns, 2019.

[168] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *NIPS*, 2004.

[169] K. Zhou, Q. Song, X. Huang, and X. Hu. Auto-gnn: Neural architecture search of graph neural networks, 2019.

[170] D. Zhu, Z. Zhang, P. Cui, and W. Zhu. Robust graph convolutional networks against adversarial attacks. In *KDD*, 2019.

[171] Q. Zhu, B. Du, and P. Yan. Self-supervised training of graph convolutional networks. *arXiv*, 2020.

[172] S. Zhu, K. Yu, Y. Chi, and Y. Gong. Combining content and link for classification using matrix factorization. In *SIGIR*, 2007.

[173] X. Zhu, Z. Ghahramani, and J. D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, 2003.

[174] H. Ziniu, D. Yuxiao, W. Kuansan, and S. Yizhou. Heterogeneous graph transformer. *WWW*, 2020.

[175] B. Zoph, G. Ghiasi, T.-Y. Lin, Y. Cui, H. Liu, E. D. Cubuk, and Q. V. Le. Rethinking pre-training and self-training. *CVPR*, 2020.

[176] B. Zoph and V. Q. Le. Neural architecture search with reinforcement learning. *ICLR*, 2017.

[177] B. Zoph, V. Vasudevan, J. Shlens, and V. Q. Le. Learning transferable architectures for scalable image recognition. *computer vision and pattern recognition*, 2018.

[178] D. Zügner, A. Akbarnejad, and S. Günnemann. Adversarial attacks on neural networks for graph data. In *KDD*, 2018.