sensors

MDPI

Review

Graph Representation Learning and Its Applications: A Survey

Van Thuy Hoang [1], Hyeon-Ju Jeon [2], Eun-Soon You [1], Yoewon Yoon [3], Sungyeop Jung [4] and O-Joun Lee [1,*]

[1] Department of Artificial Intelligence, The Catholic University of Korea, 43, Jibong-ro, Bucheon-si 14662, Gyeonggi-do, Republic of Korea; hoangvanthuy90@gmail.com (V.T.H.); tesniere@naver.com (E.-S.Y.)
[2] Data Assimilation Group, Korea Institute of Atmospheric Prediction Systems (KIAPS), 35, Boramae-ro 5-gil, Dongjak-gu, Seoul 07071, Republic of Korea; hjjeon@kiaps.org
[3] Department of Social Welfare, Dongguk University, 30, Pildong-ro 1-gil, Jung-gu, Seoul 04620, Republic of Korea; yyoon@dongguk.edu
[4] Semiconductor Devices and Circuits Laboratory, Advanced Institute of Convergence Technology (AICT), Seoul National University, 145, Gwanggyo-ro, Yeongtong-gu, Suwon-si 16229, Gyeonggi-do, Republic of Korea; sungyeop.jung@snu.ac.kr
* Correspondence: ojlee@catholic.ac.kr; Tel.: +82-2-2164-5516

Abstract: Graphs are data structures that effectively represent relational data in the real world. Graph representation learning is a significant task since it could facilitate various downstream tasks, such as node classification, link prediction, etc. Graph representation learning aims to map graph entities to low-dimensional vectors while preserving graph structure and entity relationships. Over the decades, many models have been proposed for graph representation learning. This paper aims to show a comprehensive picture of graph representation learning models, including traditional and state-of-the-art models on various graphs in different geometric spaces. First, we begin with five types of graph embedding models: graph kernels, matrix factorization models, shallow models, deep-learning models, and non-Euclidean models. In addition, we also discuss graph transformer models and Gaussian embedding models. Second, we present practical applications of graph embedding models, from constructing graphs for specific domains to applying models to solve tasks. Finally, we discuss challenges for existing models and future research directions in detail. As a result, this paper provides a structured overview of the diversity of graph embedding models.

Keywords: graph embedding; graph representation learning; graph transformer; graph neural networks

Citation: Hoang, V.T.; Jeon, H.-J.; You, E.-S.; Yoon, Y.; Jung, S.; Lee, O.-J. Graph Representation Learning and Its Applications: A Survey. *Sensors* **2023**, *23*, 4168. https://doi.org/10.3390/s23084168

Academic Editor: Alessandro Bevilacqua

Received: 8 March 2023
Revised: 16 April 2023
Accepted: 17 April 2023
Published: 21 April 2023

1. Introduction

Graphs are a common language for representing complex relational data, including social media, transportation system networks, and biological protein–protein networks [1,2]. Since most graph data are complex and high-dimensional, it is difficult for researchers to extract valuable knowledge. Therefore, processing graph data and transforming them into a form (fixed-dimensional vectors) is an important process that researchers can then apply to different downstream tasks [3]. The objective of graph representation learning is to obtain vector representations of graph entities (e.g., nodes, edges, subgraphs, etc.) to facilitate various downstream tasks, such as node classification [4], link prediction [5,6], community detection [7], etc. As a result, graph representation learning plays an important role since it could significantly promote the performance of the downstream tasks.

Representation of the graph data, however, is challenging and different from image and text data [8]. In textual data, words are linked together in a sentence, and they have a fixed position in that sentence. In image data, pixels are arranged on an ordered grid space and can be represented by a grid matrix. However, the nodes and edges in graphs are non-ordered and have their features. This leads to mapping graph entities to latent space

while preserving the graph structure, and proximity relationships are challenging. In the case of a social network, a user can have many friends (neighbors) and various personal information, such as hometown, education level, and hobbies, which makes preserving the graph structure and properties significantly problematic. In addition, many real-world networks show dynamic behaviors in which graph structures and node features could be changed over time [9,10]. These could deliver challenges in capturing the graph structure and mapping graph entities into vector space.

Over decades, various graph representation learning models have been proposed to project graph entities into fixed-length vectors [11–13]. Graph embedding models are mainly divided into five main groups: graph kernels, matrix factorization models, shallow models, deep neural network models, and non-Euclidean models. Figure 1 presents the popularity of different graph representation learning models from 2010 to 2022. The number of graph representation learning studies increased considerably over the period of 12 years. Furthermore, there was significant growth in the frequency of research studies on graph neural networks, graph convolutional networks, and graph transformer models. In contrast, the number of studies in graph kernels, graph autoencoder, and matrix factorization-based models increased slightly over the period of 12 years. We obtained the frequency of academic publications including each keyword from Scopus (https://www.scopus.com (accessed on 16 April 2023)).

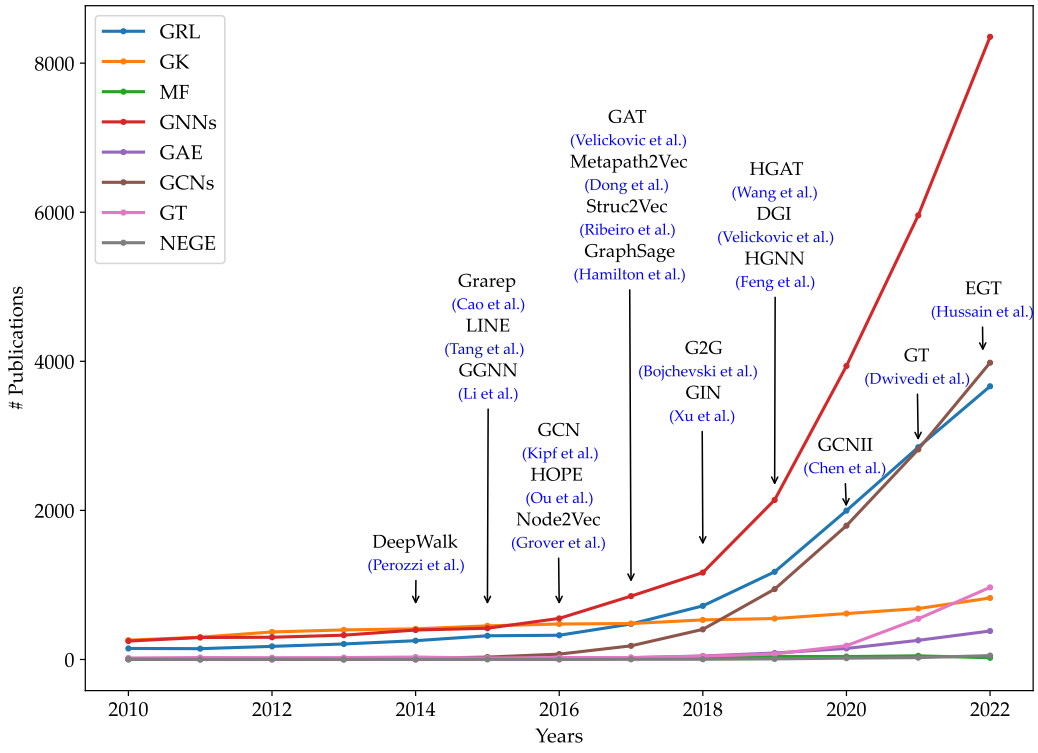


Figure 1. The popularity of graph representation learning models in the Scopus database. The line plot shows changes in the number of publications in different types of graph representation learning models from 2010 to 2022. The y-axis denotes the number of publications on the popularity of graph representation learning models over the years. There are seven keywords, including graph representation learning (GRL), graph kernels (GK), matrix factorization-based graph embedding (MF), graph neural networks (GNNs), graph autoencoder (GAE), graph convolution networks (GCNs), graph transformer (GT), and non-Euclidean graph embedding (NEGE). There are nineteen representative models, including DeepWalk [14], Grarep [15], LINE [16], GGCN [17], GCN [18], HOPE [5], Node2Vec [4], GAT [19], Metapath2Vec [20], Struc2Vec [21], GraphSage [22], G2G [23], GIN [24], HGAT [25], DGI [26], HGNN [27], GCNII [28], GT [29], and EGT [30].

Historically, the first graph representation learning models were graph kernels. The idea of graph kernel methods perhaps comes from the most essential and well-known Weisfeiler–Lehman (WL) isomorphic testing in 1968 [31]. Graph kernels are kernel functions that aim to measure the similarity between graphs and their entities [32]. The main idea of graph kernels is to decompose original graphs into substructures and construct vector embeddings based on the substructure features. There are two main types of graph kernels: kernels for graphs and kernels on graphs. The former aims to measure the similarity between pairs of graphs, while the latter estimates the similarity between graph nodes. Several strategies to estimate the similarity of graph pairs have been proposed to represent various graph structures, such as graphlet kernels, random walk, and the shortest path, which started in the 2000s. Based on WL isomorphic testing, various graph kernels are built to compute the similarity of pairs of graph entities, such as WL kernels [31], WL subtree kernels [33–35], and random walks [36,37]. However, one of the limitations of graph kernels is the computational complexity when working with large-scale graphs since computing graph kernels is an NP-hard class.

Early models for graph representation learning primarily focused on matrix factorization methods, which are motivated by traditional techniques for dimensionality reduction in 2002 [38]. Several matrix factorization-based models have been proposed to handle large graphs with millions of nodes [39,40]. The objective of matrix factorization models is to decompose the proximity matrix into a product of small-sized matrices and then learn the embeddings that fit the proximity. Based on the ways to learn vector embeddings, there are two main lines of matrix factorization models: Laplacian eigenmaps and node proximity matrix factorization. Starting in the 2000s, Laplacian eigenmaps methods [41,42] aim to represent each node by Laplacian eigenvectors along with the first k eigenvalues. In contrast, the node proximity matrix factorization methods [5,15] aim to gain node embeddings by the singular value decomposition in 2015. Various proximity matrix factorization models have successfully handled large graphs and achieved great performance [15,43]. However, matrix factorization models suffer from capturing high-order proximity due to computational complexity when performing with high transition matrices.

In 2014 and 2016, early shallow models, DeepWalk [14] and Node2Vec [4] were proposed, which learn node embeddings based on shallow neural networks. Remarkably, the primary concept is to learn node embeddings by maximizing the neighborhood probability of target nodes using the skip-gram model started in the natural language processing area. The purpose of this strategy could then be optimized with SGD on neural network layers, thus reducing computational complexity. With this historic milestone, various models have been developed by improving multiple sampling strategies and training processes. Shallow models are the embedding models that aim to map graph entities to low-dimensional vectors by conducting an embedding lookup for each graph entity [3]. From this perspective, the embedding of node v_i could be represented as $Z_i = Mx_i$, where M denotes an embedding matrix of all nodes and x_i is a one-hot vector of node v_i. Various shallow models have been proposed to learn embeddings with different strategies to preserve graph structures and the similarity between graph entities. Structure-preservation models aim to preserve the structural connection between entities (e.g., DeepWalk [14], Node2Vec [4]). In 2015, Tang et al. [16] proposed the LINE model, a proximity reconstruction method that aims to preserve proximity between nodes in graphs. After that, various models have been proposed to preserve the node proximity with higher-order proximity and capture more global graph structure. However, most of the above models focus on transductive learning and ignore node features, which may have several limitations to practical applications.

Breakthroughs in deep learning have led to a new research perspective on applying deep neural networks to the graph domain. Since the 2000s, there have been several early models on GNNs designed to learn node embeddings based on neighborhood information using an aggregation mechanism [44,45]. Graph neural networks (GNNs) have shown a significant expressive capacity to represent graph embeddings in an inductive learning manner and solve the limitations of aforementioned shallow models [46,47]. Recurrent GNNs

are also the first studies on GNNs based on recurrent neural network architecture [48,49] in 2005. These models aim to learn node embeddings via recurrent layers with the same weights in each hidden layer and run recursively until convergence. Several recurrent GNNs with different strategies have been proposed by the power of recurrent neural network architecture and the combinations with several sampling strategies. However, using the same weights at each hidden layer of the RGNN model may cause the model to be incapable of distinguishing the local and global structure. Since 2016, several graph autoencoder models have been proposed based on the original autoencoder architecture, which could learn complex graph structures by reconstructing the input graph structure [50,51]. The graph autoencoders comprise two main layers: encoder layers take the adjacency matrix as input and squeeze it to generate node embeddings, and decoder layers reconstruct the input data. By contrast, the idea of CGNNs is to use convolutional operators with different weights in each hidden layer, which are more efficient in capturing and distinguishing the local and global structures [18,52–54]. Many studies have been proposed with various variants of CGNNs, including spectral CGNNs [55–57] started in 2014, spatial CGNNs [22,24,52] started in 2016, and attentive CGNNs [19,58] started in 2017. Nevertheless, most GNNs suffer limitations such as over-smoothing problems and noise from neighbor nodes when stacking more GNN layers [59,60].

Motivated by transformer architecture started from natural language processing applications in 2017, several graph transformer models were proposed using the transformer architecture to the graph domain in 2019 [61,62]. Graph transformer models have shown competitive and superior performance against GNNs in learning complex graph structures [30,63]. Graph transformer models can be divided into three main groups: transformer for tree-like graphs, transformer with GNNs, and transformer with global self-attention. Early graph transformer models aim to learn tree-like graphs, which mainly aim at learning node embeddings in tree-like graphs where nodes are arranged hierarchically [64,65] since 2019. These models encode the node positions through their relative and absolute positional encoding in trees as constraints with root nodes and neighbor nodes at the same level. Second, several models leverage the power of GNNs as an auxiliary module in computing attention scores [66]. In addition, some models put GNN layers on top of the model to overcome the over-smoothing problem and make the model remember the local structure [61]. Most above graph transformer models adopt vanilla transformer architecture to learn embeddings that rely on multi-head self-attention. Third, several graph transformer models use a global self-attention mechanism to learn node embeddings, which implements self-attention independently and does not require constraints from the neighborhood [30,67]. These models work directly on input graphs and can capture the global structure with global self-attention.

Most of the above models learn embeddings in Euclidean space and represent graph entities as vector points in latent space. However, graphs in the real world could have complex structures and different forms, such that Euclidean space may be inadequate to represent the graph structure and ultimately lead to structural loss [68,69]. Early models learn complex graphs in non-Euclidean geometry by developing efficient algorithms for learning node embeddings based on manifold optimization [70] in 2017. Following the line, several models aim to represent graph data in non-Euclidean space and gain desirable results [68,69,71]. Two typical non-Euclidean spaces, including spherical and hyperbolic geometry, have their advantages. Spherical space could represent graph structures with large cycles, while hyperbolic space is suitable for hierarchical graph structures. Most non-Euclidean models aim to design an efficient algorithm for learning node embeddings since it is challenging to implement operators directly in non-Euclidean. Furthermore, to deal with uncertainty, several Gaussian graph models have been introduced to represent graph entities as density-based embeddings [23] started in 2016. Node embeddings could be defined as a continuous density mostly based on Gaussian distribution [72].

To the extent of our knowledge, no comparable paper in the literature focuses on a wide range of graph embedding models for static and dynamic graphs in different geometric

spaces. Most current papers only presented specific approaches for graph representation learning. Wu et al. [8] focused on graph neural network models, which are presented as a section in this paper. Several surveys [13,73,74] summarized graph embedding models for various types of graphs, but they did not mention either graph transformer models or non-Euclidean models. From applying graph embedding models to practical applications, several papers only list the applications for specific and narrow tasks [12,75]. However, we discuss how graphs are constructed in specific applications and how graph embedding models are implemented in various domains.

This paper presents a comprehensive picture of graph embedding models in static and dynamic graphs in different geometric spaces. In particular, we recognize five general categories of models for addressing graph representation learning, including graph kernels, matrix factorization models, shallow models, deep neural network models, and non-Euclidean models. The contribution of this study can be categorized as follows:

- This paper presents a taxonomy of graph embedding models based on various algorithms and strategies.
- We provide readers with an in-depth analysis of an overview of graph embedding models with different types of graphs ranging from static to dynamic and from homogeneous to heterogeneous graphs.
- This paper presents graph transformer models, which have achieved remarkable results in a deeper understanding of graph structures in recent years.
- We cover applications of graph representation learning in various areas, from constructing graphs to applying models in specific tasks.
- We discuss the challenges and future directions of existing graph embedding models in detail.

Since abundant graph representation learning models have been proposed recently, we employed different approaches to find related studies. We built a search strategy by defining keywords and analyzing reliable sources. The list of keywords includes graph embedding, graph representation learning, graph neural networks, graph convolution, graph attention, graph transformer, graph embedding in non-Euclidean space, Gaussian graph embedding, and applications of graph embedding. We found related studies at famous top-tier conferences and journals such as AAAI, IJCAI, SIGKDD, ICML, WSDM, Nature Machine Intelligence, Pattern Recognition, Intelligent Systems with Applications, the Web, and so on.

The following sections of this paper are summarized as follows. Section 2 describes fundamental concepts and backgrounds related to graph representation learning. In Section 3, all the graph embedding models will be presented, such as graph kernels, matrix factorization models, shallow models, deep neural network models, and non-Euclidean models. Section 4 discusses a wide range of practical applications of graph embedding models in the real world. Section 5 summarizes the latest benchmarks, downstream tasks, evaluation metrics, and libraries. Challenges for existing graph embedding models and future research directions will be discussed in Section 6. The last section, Section 7 is the conclusion.

2. Problem Description

Graph representation learning aims to project the graph entities into low-dimensional vectors while preserving the graph structure and the proximity of entities in graphs. With the desire to map graph entities into vector space, it is necessary to model the graph in mathematical form. Therefore, we begin with several fundamental definitions of graphs. The list of standard notations used in this survey is detailed in Table 1. Mathematically, a graph G can be defined as follows:

Definition 1 (Graph [3]). *A graph is a discrete structure consisting of a set of nodes and the edges connecting those nodes. The graph can be described mathematically in the form:* $G = (V, E, A)$, *where* $V = \{v_1, v_2, \cdots, v_N\}$ *is the set of nodes,* $E = \{(v_i, v_j) | (v_i, v_j) \in V \times V\}$ *is the set of*

edges, and A is an adjacency matrix. A is a square matrix of size N × N where N is the number of nodes in graphs. This can be formulated as follows:

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1N} \\ \vdots & \ddots & \vdots \\ A_{N1} & \cdots & A_{NN} \end{bmatrix}, \; A_{ij} = \begin{cases} 1, & \text{if } e_{ij} \in E. \\ 0, & \text{otherwise.} \end{cases}, \tag{1}$$

where A_{ij} indicates adjacency between node v_i and node v_j.

Table 1. A summary of notations.

Notations	Descriptions
V	The set of nodes in the graph G
E	The set of edges in graph G
N	The number of nodes in graph G
E^t	The set of edges with type t in heterogeneous graphs
v_i	The node v_i in the graph G
e_{ij}	The edge (v_i, v_j) in the graph G
A	The adjacency matrix of the graph G
X	The feature matrix of nodes in graph G
D	The degree matrix of nodes in graph G
ϕ	Projection function
Z_i	The embedding vector of node v_i
M	The transition matrix
$N(v_i)$	The set of neighbors of node v_i
k	The k-hop distance from a target node to other nodes
d	The dimension of vector in latent space
y_i	The label of node v_i

When A_{ij} is binary, the matrix A represents only the existence of connections between nodes. By extending the definition of matrix A, we could expand to abundant different types of graph G:

- Directed graph: When $A_{ij} = A_{ji}$ for any $1 \leq i, j \leq n$, then the graph G is called an undirected graph, and G is directed graph otherwise.
- Weighted graph: is a graph in which each edge is assigned a specific weight value. Therefore, the adjacency matrix could be presented as: $A_{ij} = w_{ij}$, where $w_{ij} \in \mathbb{R}$ is the weight of the edge e_{ij}.
- Signed graph: When $A_{ij} \in [-\infty, \infty]$, the graph G is called signature/signed graph. The graph G could have all positive signed edges when $A_{ij} > 0$ for any $1 \leq i, j \leq n$, and G could have all negative signed edges otherwise.
- Attributed graph: A graph $G = (V, E, A, X)$ is an attributed graph where V, E is the set of nodes and edges, respectively, and X is the matrix of node attributes with size $n \times d$. Furthermore, we could also have the matrix X as the matrix of edge input attribute with size $m \times d$ where m is the number of edges $e_{ij} \in E$ for any $1 \leq i, j \leq n$.
- Hyper graph: A hyper graph G could be represented as $G = (V, E, W)$, where V denotes the set of nodes and E denotes a set of hyperedge. Each hyperedge e_{ij} can connect multiple nodes and is assigned a weight $w_{ij} \in W$. The hypergraph G could be represented by an incidence matrix H size $|V| \times |E|$ with entries $h(v_i, v_j) = 1$ if $e_{ij} \in E$, and $h(v_i, v_j) = 0$ otherwise.
- Heterogeneous graph: A heterogeneous graph is defined as $G = (V, E, T, \varphi, \rho)$ where V, and E are the set of nodes and edges, respectively, φ is the mapping function: $\varphi : V \rightarrow T_v$, and the mapping function $\rho : E \rightarrow T_e$ with T_v, T_e describe the set of node types and edge types, respectively, and $T = T_v + T_e$ is the sum of the number of node types and edge types.

According to the definitions of graph $G = (V, E)$ that have been represented mathematically above, the idea of graph embedding is to map graph entities into low-dimensional

vectors with the number of dimensions d with $d \ll N$. Mathematically, the graph embedding is formulated as follows:

Definition 2 (Graph embedding [14]). *Given a graph $G = (V, E)$ where V is the set of nodes, and E is the set of edges, graph embedding is a projection function $\phi(\cdot)$, where $\phi : V \to \mathbb{R}^d$ ($d \ll |V|$) and $k(v_i, v_j) \simeq \langle \phi(v_i), \phi(v_j) \rangle$ describes the proximity of two nodes v_i and v_j in the graph and $\langle \phi(v_i), \phi(v_j) \rangle$ is the distance of two vectors $\phi(v_i)$ and $\phi(v_j)$ in the vector space.*

Graph representation learning aims to project graph entities into the vector space while preserving the graph structure and entity proximity. For example, if two nodes v_i and v_j in the graph G are connected directly, then in vector space, the distance between two vectors $\phi(v_i)$ and $\phi(v_j)$ must be minimal. Figure 2 shows an example of a graph embedding model that transforms nodes in a graph to low-dimensional vectors $(Z_1\ Z_2 \cdots Z_n)$ in the vector space.

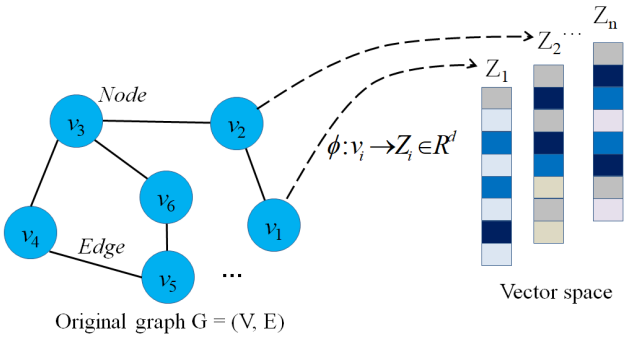


Figure 2. A comprehensive view of graph embedding. Given a spare, high-dimensional graph $G = (V, E)$ where V and E denote the set of nodes and edges. Graph embedding learning aims to find a function ϕ that maps nodes from graph space to d-dimensional vector space with $d \ll |V|$.

When mapping graph entities to latent space, preserving the proximity of graph entities is one of the most important factors in preserving the graph structure and the relationship between nodes. In other words, if two nodes v_i and v_j are connected or close in the graph, the distance between the two vectors Z_i and Z_j must be minimal in the vector space. Several models [16,76–78] aim to preserve k-order proximity between graph entities in vector space. Formally, the k-order proximity is defined as follows:

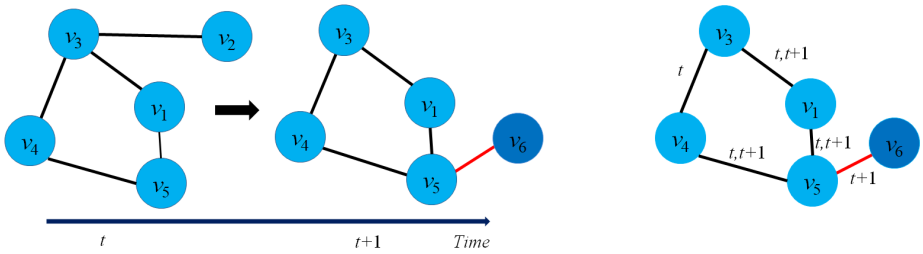
Definition 3 (k-order proximity [79]). *Given a graph $G = (V, E)$ where V is the set of nodes, and E is the set of edges, k-order proximity describes the similarity of nodes with the distance captured from the k-hop in the graph G. When $k = 1$, it is 1st-order proximity that captures the local pairwise proximity of two nodes in graphs. When k is higher, it could capture the global graph structure.*

There is another way to define graph embedding from the perspective of Encoder-Decoder architecture [3]. From this perspective, the task of the encoder part is to encode graph entities into low-dimensional vectors, and the decoder part tries to reconstruct the graph from the latent space. In the real world, many graphs show dynamic behaviors, including node and edge evolution, and feature evolution [80]. Dynamic graphs are found widely in many applications [81], such as social networks where connections between friends could be added or removed over time.

Definition 4 (Dynamic graph [80]). *A dynamic graph $\mathbb{G}$ is formed of three entities: $\mathbb{G} = (\mathbb{V}, \mathbb{E}, T)$ where $\mathbb{V} = \{V(t)\}$ is the group of node sets, $\mathbb{E} = \{E(t)\}$ with $t \in T$ is the group of edge sets over time span T, and T denotes the time span. From the statistic perspective, we could also consider a dynamic graph $\mathbb{G} = \{G(t_0)\ G(t_1) \cdots G(t_n)\}$ as a collection of static graphs $G(t_k)$*

where $G(t_k) = (V(t_k), E(t_k))$ denotes the static graph G at time t_k, and $V(t_k)$, $E(t_k)$ denotes the set of nodes and set of edges at time t_k, respectively.

Figure 3a presents an example of dynamic graph representation. At time $t+1$, there are several changes in the graph $G(t+1)$ such as: the edge e_{23} will be removed, node v_6 will be added and new edge e_{56}. Casteigts et al. [80] proposed an alternative definition of a dynamic graph with five components: $\mathbb{G} = (\mathbb{V}, \mathbb{E}, T, \rho, \zeta)$ where $\rho : \mathbb{V} \times T \to \{0,1\}$ describes the existence of each node at time t, and $\zeta : \mathbb{E} \times T \to \Psi$ describes the existence of an edge at time t.



(**a**) (**b**)

Figure 3. Methods for modeling dynamic graphs over time. (**a**) The representation of a dynamic graph by a series of snapshots; (**b**) The evolution of edges and nodes in the dynamic graph from time t to $t+1$. In (**a**), the graph G is the collection of $G(t)$ (i.e., $G = \{G(1), G(2), \cdots, G(t)\}$) which t is the time span, and the entities of G change from time t to $t+1$. (**b**) depicts the evolution of edges in the same dynamic graph from (**a**) which each edge contains the series of the time spans from t to $t+1$. At time t, the graph has five nodes (v_1, v_2, v_3, v_4, v_5) and five edges ($e_{13}\, e_{15}\, e_{34}\, e_{45}\, e_{23}$). However, at time $t+1$, the edge e_{23} and node v_2 are removed, and a new node v_6, a new edge e_{56} are added in the graph.

There is another way to model a dynamic graph based on the changes of the graph entities (edges, nodes) taking place on the graph $\mathbb{G}$ over a time span t or by an edge stream. From this perspective, a dynamic $\mathbb{G}$ could be modeled as $\mathbb{G} = (\mathbb{V}, E_t, T)$ where E_t presents the collection of edges of dynamic graph $\mathbb{G}$ at time t, and function $f : \mathbb{E} \to R^+$ to map edges into integer numbers. It notices that all the edges at time t will have the same labels. Figure 3b describes the evolution of the edges of a graph from time (t) to $(t+1)$.

Definition 5 (Dynamic graph embedding [82]). *Given a dynamic graph $\mathbb{G} = (\mathbb{V}, \mathbb{E}, T)$ where $\mathbb{V} = \{V(t)\}$ is the group of node sets, and $\mathbb{E} = \{E(t)\}$ is the group of edge sets over time span T, a dynamic graph embedding is a projection function $\phi(\cdot)$, where $\phi(\cdot) : \mathbb{G} \times T \to R^d \times \mathbb{T}$. $\mathbb{T}$ describes the time domain in latent space and T is the time span. When $\mathbb{G}$ is represented as the collection of snapshots: $\mathbb{G} = \{G(t_0)\, G(t_1) \cdots G(t_n)\}$, then the projection function ϕ will be defined as: $\phi = \{\phi(0)\, \phi(1) \cdots \phi(n)\}$ where $\phi(t)$ is the vector embedding of the graph $G(t)$ at time t.*

There are two ways to represent a dynamic graph $\mathbb{G}$, including a temporal dynamic graph embedding (changes over a period of time) and topological dynamic graph embedding (changes in the graph structure over time).

- Temporal dynamic graph embedding: A temporal dynamic embedding is a projection function $\phi(\cdot)$, where $\phi_t : \mathbb{G}_{t-k,t} \times T \to R^d \times \mathbb{T}$ and $\mathbb{G}_{t-k,t} = \{G_{t-k}\, G_{t-k+1} \cdots G_t\}$ describes the collection of graph $\mathbb{G}$ during time interval $[t-k, t]$.
- Topological dynamic graph embedding: A topological dynamic graph embedding for graph $\mathbb{G}$ for nodes is a mapping function ϕ, where $\phi : \mathbb{V} \times T \to \mathbb{R}^d \times \mathbb{T}$.

3. Graph Representation Learning Models

This section presents a taxonomy of existing graph representation learning models in the literature. We categorize the existing graph embedding models into five main groups based on strategies to preserve graph structures and proximity of entities in graphs, including graph kernels, matrix factorization-based models, shallow models, deep neural network models, and non-Euclidean models. Figure 4 presents the proposed taxonomy of the graph representation learning models. Furthermore, we deliver open-source implementations of graph embedding models in Appendix A.

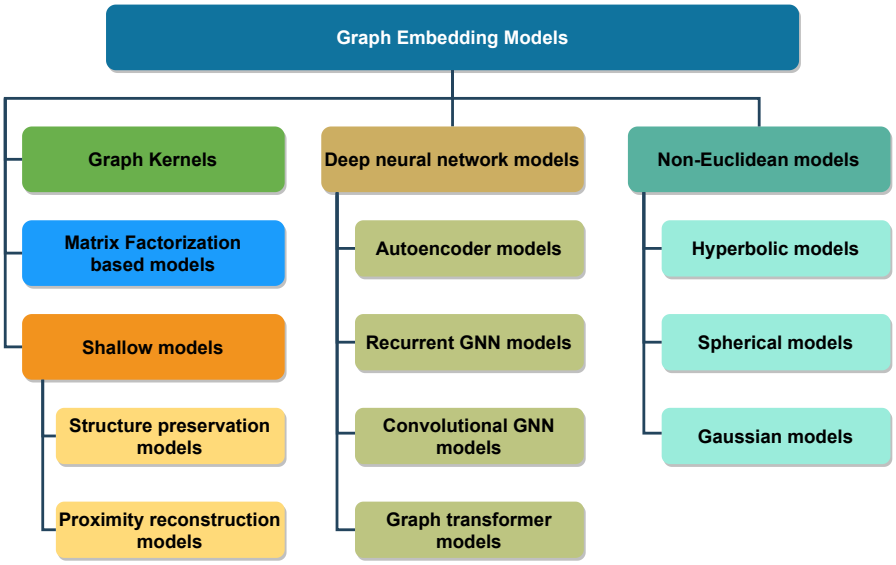


Figure 4. The proposed taxonomy for graph representation learning models.

Graph kernels and matrix factorization-based models are one of the pioneer models for graph representation learning. Graph kernels are prevalent in learning graph embeddings using a deterministic mapping function in solving graph classification tasks [83–85]. There are two types of graph kernels: kernels for graphs, which aim to compare the similarity between graphs, and kernels on graphs aim to find the similarity between nodes in graphs. Second, matrix factorization-based models aim to represent the graph as matrices and gain embeddings by decomposing the matrices [5,86]. There are several strategies for factorization modeling, and most of these models aim to approximate high-order proximity between nodes. However, graph kernels and matrix factorization-based models suffer from computational complexity when handling large graphs and capturing high-order proximity.

Shallow models aim to construct an embedding matrix to transform each graph entity into vectors. We categorize shallow models into two main groups: structure preservation and proximity reconstruction. Structure-preservation strategies aim to conserve structural relationships between nodes in graphs [4,14,87]. Depending on specific tasks, several sampling strategies could be employed to capture graph structures, such as random walks [4,14], graphlets [88], motifs [89–91], etc. By contrast, the objective of the proximity reconstruction models is to preserve the proximity of nodes in graphs [16,92]. The proximity strategies can vary across different models based on their objectives. For example, the LINE model [16] aims to preserve 1st-order and 2nd-order proximity between nodes, while PALE [77] preserves pairwise similarities.

Graph neural networks have shown great performance in learning complex graph structures [18,50]. GNNs can be categorized into three main groups: graph autoencoder [50,51], recurrent GNNs [17,93], and convolutional GNNs. Graph autoencoders and recurrent GNNs are mostly pioneer studies of GNNs based on autoencoder architecture and recurrent neural networks, respectively. Graph autoencoders are composed of an encoder layer and a decoder layer. The encoder layer aims to compress a proximity graph matrix to

vector embeddings, and the decoder layer reconstructs the proximity matrix. Most graph autoencoder models employ multilayer perceptron-based layers or recurrent GNNs as the core of autoencoder architecture. Recurrent GNNs aim to learn node embeddings based on recurrent neural network architecture in which connections between neurons can make a cycle. Therefore, earlier RGNNs mainly aimed to learn embeddings on directed acyclic graphs [94]. Recurrent GNNs employ the same weights in all hidden layers to capture local and global structures. Recently, convolutional GNNs have been much more efficient and can gain outstanding performance compared to RGNNs. The main difference between RGNNs and CGNNs is that CGNNs use different weights in each hidden layer, which could distinguish local and global structures. Various CGNN models have been proposed and mainly fall into two categories: spectral CGNNs, and spatial CGNNs [22,52,95]. Spectral CGNNs aim to transform graph data to the frequency domain and learn node embeddings in this domain [56,96]. By contrast, spatial CGNNs work directly on the graph using convolutional filters [53,54]. By staking multiple GNN layers, the models could learn node embeddings more efficiently and capture higher-order structural information [97,98]. However, stacking many layers could cause the over-smoothing problem, which most GNNs have not fully solved in a whole extent.

Recently, several models have enabled transformer architecture to learn graph structures which gain significant results compared to other deep-learning models [30,46,99]. We categorize graph transformer models into three main groups: transformer for tree-like graphs [64,65], transformer with GNNs [99,100], and transformer with global self-attention [30,67]. Different types of graph transformer models aim to handle distinct types of graphs. The transformer for tree-like graphs aims to learn node embeddings in tree-like hierarchical graphs [64,65,101]. The hierarchical relationships from the target nodes to their parents and neighbors are presented as absolute and relative positional encoding, respectively. Several graph transformer models employ the message-passing mechanism from GNNs as an auxiliary module in computing the attention score matrix [61,100]. GNN layers can be used to aggregate information as input to graph transformer models or put on top of the model, which aims to preserve local structures. In addition, some graph transformer models can directly process graph data without support from GNN layers [30,67]. These models implement a global self-attention to learn local and global structures in a graph input without neighborhood constraints.

Most existing graph embedding models aim to learn embeddings in Euclidean space, which may not deliver good geometric representations and metrics. Recent studies have shown that non-Euclidean spaces are more suitable for representing complex graph structures. The non-Euclidean models could be categorized as hyperbolic, spherical, and Gaussian. Hyperbolic and spherical space are two types of non-Euclidean geometry that could represent different graph structures. Hyperbolic space [102] is more suitable for representing hierarchical graph structures that follow the power law, while the power of spherical space is to represent large circular graph structures [103]. Moreover, since the information about the embedding space is unknown and uncertain, several models aim at learning node embeddings as Gaussian distribution [23,104].

3.1. Graph Kernels

Graph kernels aim to compare graphs or their substructures (e.g., nodes, subgraphs, and edges) by measuring their similarity [105]. The problem of measuring the similarity of graphs is, therefore, at the core of learning graphs in an unsupervised manner. Measuring the similarity of large graphs is problematic since the graph isomorphism problem is assigned to the NP (nondeterministic polynomial time) class. However, it is an NP-complete for subgraphs isomorphism problem. Table 2 describes a summary of graph kernel models.

Kernel methods applied to the graph embedding problem can be understood in two forms, including the isomorphism testing of N graphs (kernels for graphs) and embedding entities of graphs to Hilbert space (kernels on graphs).

- Kernels for graphs: Kernels for graphs aim to measure the similarity between graphs. The similarity between the two graphs (isomorphism) could be explained as follows: Given two undirected graphs $G_1 = (V_1\, E_1)$ and $G_2 = (V_2\, E_2)$, G_1 and G_2 are isomorphic if they exist a bimodal mapping function $\phi : V_1 \to V_2$ such that $\forall a\, b \in V_1$, a and b are contiguous on G_1 if $\phi(a)$ and $\phi(b)$ are contiguous on G_2.
- Kernels on graphs: To embed nodes in graphs, kernel methods refer to finding a function that maps pairs of nodes to latent space using particular similarity measures. Formally, graph kernels could be defined as: Given a graph $G = (V, E)$, a function $K = V \times V \to R$ is a kernel on G if there is a mapping function $\phi : V \to H$ such that $K(v_i\, v_j) = \langle \phi(v_i)\, \phi(v_j) \rangle$ for any node pairs $(v_i\, v_j)$.

There are several strategies to measure the similarity of pairs of graphs, such as graphlet kernels, WL kernels, random walk, and shortest paths [31,83]. Among the kernel methods, graphlet kernels are one of the simple kernels that could measure the similarity between graphs by counting subgraphs with a limited size k [83,106]. For instance, Shervashidze et al. [83] introduced a graphlet kernel with the main idea of finding the graph feature by counting the number of different graphlets in graphs. Formally, given an unlabeled graph G, a graphlet list $V_k = (G_1 + G_2 + \cdots + G_{n_k})$ is the set of the graphlets with size k where n_k depicts the number of graphlets. The graphlet kernel for two unlabeled graphs G and G' could be defined as:

$$K(G, G') = \langle \phi(G), \phi(G') \rangle, \tag{2}$$

where ϕ^G and $\phi^{G'}$ are vectors that depict the number of graphlets in a G_i and G_i', respectively. By counting all graphlets with size k for a graph, the computation time is expensive by the enumeration n^k with n depicts the number of nodes in G. One of the practical solutions to overcome this limitation is to design the feature ${\phi_i}^G$ more effectively, called Weisfeiler–Lehman.

Weisfeiler–Lehman (WL) test [31] is considered to be a traditional strategy to test the homomorphism of two graphs using color refinements. Figure 5 presents the main idea of the WL homomorphism test for two graphs in detail. By updating node labels, all the structure information of nodes in graphs could be stored at each node, including both local and global information, depending on the number of iterations. We can then compute histograms or other summary statistics over these labels as a vector representation for graphs.

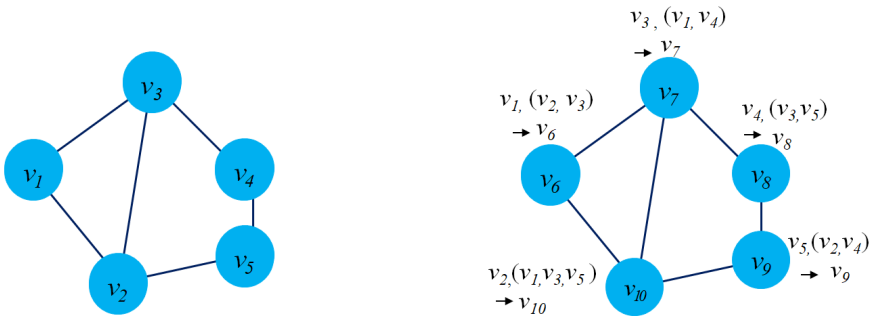
Table 2. A summary of graph kernel models.

Models	Graph Types	Tasks	Loss Function
[83]	Static graphs	Graph comparison	$\sum_{v_i \in V} \max(0, 1 - y_i^\intercal \hat{y}_i) + L_2$
[106]	Static graphs	Graph comparison	$\sum_{v_i \in V} \max(0, 1 - y_i^\intercal \hat{y}_i) + L_2$
[33]	Static graphs	Graph classification	$\sum_{v_i \in V} \max(0, 1 - y_i^\intercal \hat{y}_i) + L_2$
[34]	Static graphs	Graph classification	$\sum_{v_i \in V} \max(0, 1 - y_i^\intercal \hat{y}_i) + L_2$
[84]	Static graphs	Graph classification	$\sum_{v_i \in V} \max(0, 1 - y_i^\intercal \hat{y}_i) + L_2$
[35]	Static graphs	Graph classification	$\sum_{v_i \in V} \max(0, 1 - y_i^\intercal \hat{y}_i) + L_2$
[85]	Static graphs	Graph comparison	$\sum_{v_i \in V} \max(0, 1 - y_i^\intercal \hat{y}_i) + L_2$
[107]	Attributed graphs	Graph classification	$\sum_{v_i \in V} \max(0, 1 - y_i^\intercal \hat{y}_i) + L_2$
[37]	Attributed graphs	Graph classification	$\sum_{v_i \in V} \max(0, 1 - y_i^\intercal \hat{y}_i) + L_2$
[108]	Attributed graphs	Graph classification	$\sum_{v_i \in V} \max(0, 1 - y_i^\intercal \hat{y}_i) + L_2$
[36]	Attributed graphs	Graph classification	$\sum_{v_i \in V} \max(0, 1 - y_i^\intercal \hat{y}_i) + L_2$
[109]	Attributed graphs	Graph classification	$\sum_{v_i \in V} \max(0, 1 - y_i^\intercal \hat{y}_i) + L_2$
[110]	Attributed graphs	Graph classification	$\sum_{v_i \in V} \max(0, 1 - y_i^\intercal \hat{y}_i) + L_2$
GraTFEL [111]	Dynamic graphs	Graph reconstruction Link prediction	$\frac{1}{N} \sum_{v_i \in V} \left\| Z_i - \hat{Z}_i \right\|_2^2 + L_1 + L_2$
[112]	Dynamic graphs	Link prediction	$\frac{1}{N} \sum_{v_i \in V} \left\| Z_i - \hat{Z}_i \right\|_2^2 + L_1 + L_2$
[113]	Dynamic graphs	Link prediction	$\frac{1}{N} \sum_{v_i \in V} \left\| Z_i - \hat{Z}_i \right\|_2^2$

Several models improved the idea from WL isomorphism test [34,84]. The concept of the WL isomorphism test inspired various GNN models later, which aim to be expressive as powerful as the WL test to distinguish different graph structures. Shervashidze et al. [33] presented three instances of WL kernels, including the WL subtree kernel, WL edge kernel, and WL shortest-path kernel with an enrichment strategy for labels. The key idea of [33] is to represent a graph G as WL sequences with the height of h. The WL sequences of two graphs G and G' can be defined as:

$$k_{WL}^{(h)}(G, G') = k(G_0, G_0') + k(G_1, G_1') + \cdots + k(G_h, G_h') \tag{3}$$

where $k(G_i, G_i') = \langle \phi(G_i), \phi(G_i') \rangle$. For N graphs, the WL subtree kernel could be computed in a runtime of $O(Nhm + N^2hn)$, where h and m are the numbers of interactions and edges in G, respectively. Therefore, the algorithm could capture more information about the graph G after h interactions and compare graphs at different levels.



(**a**) (**b**)

Figure 5. The Weisfeiler–Lehman isomorphism test. (**a**) Original labels, $i = 0$; (**b**) Relabeled labels, $i = 1$. There are two interactions of WL relabeling for the graph with five nodes $\{v_1, v_2, v_3, v_4, v_5\}$. In (**a**), labels of nodes are initialized consisting of 5 nodes. In (**b**), in the first iteration, new labels of the nodes will be reassigned and calculated based on the connection information to its adjacent nodes. For example, node v_1 is adjacent to node v_2 and node v_3, therefore the new label of v_1 is calculated as $\{v_1, \langle v_2, v_3 \rangle\}$ and resigned as new label v_6. The same steps are repeated until a steady state for the nodes is reached.

However, the vanilla WL isomorphism test requires massive resources since the methods are an NP-hard class. Following the WL isomorphism idea, Morris et al. [34] presented a set of k-set forms $V(G)_k$ and built a local and global neighborhood of the $k-$sets. Instead of working on each node in graphs, the models calculate and update the labels based on the $k-$set. The feature vectors of graph G then could be calculated by counting the number of occurrences of $k-$sets. Several models [84,114] improved the Wasserstein distance based on the WL isomorphism test, and the models could estimate weights of subtree patterns before the kernel construction [35]. Several models adopted a random-walk sampling strategy to capture the graph structure that could help reduce the computational complexity to handle large graphs [36,37,85,107].

However, the above methods only focus on homogeneous graphs in which nodes do not have side information. In the real world, graph nodes could contain labels and attributes and change over time, making it challenging to learn node embeddings. Several models have been proposed with slight variations from the traditional WL isomorphism test and random walk methods [109–113]. For example, Borgwardt et al. [109] presented random-walk sampling on attributed edges to capture the graph structure. Since existing kernel models primarily work on small-scale graphs or a subset of graphs, improving similarity based on shortest paths could achieve better computational efficiency for graph kernels in polynomial time. An all-paths kernel K could be defined as:

$$K(P(G_1), P(G_2)) = \sum_{p_1 \in P(G_1)} \sum_{p_2 \in P(G_2)} \mathrm{k}_{path}(p_1, p_2), \tag{4}$$

where $P(G_1)$ and $P(G_2)$ are the set of random-walk paths in G_1 and G_2, respectively, and $k_{path}(p_1, p_2)$ depicts a positive definite kernel on two paths p_1 and p_2. The model then applied Floyd–Warshall algorithm [115] to find k shortest-path kernels in graphs. One of the disadvantages of this model is the runtime complexity, which is about $O(k \times n^4)$, where n depicts the number of nodes in graphs. Morris et al. [108] introduced a variation of the WL subtree kernel for attributed graphs by improving existing shortest-path kernels. The key idea of this model is to use a hash function that maps continuous attributes to label codes, and then it normalizes the discrete label codes.

To sum up, graph kernels are effective models and bring several advantages:

- Coverage: The graph kernels are one of the most useful functions to measure the similarity between graph entities by performing several strategies to find a kernel in graphs. This could be seen as a generalization of the traditional statistical methods [116].
- Efficiency: Several kernel tricks have been proposed to reduce the computational cost of kernel methods on graphs [117]. Kernel tricks could reduce the number of spatial dimensions and computational complexity on substructures while still providing efficient kernels.

Although kernel methods have several advantages, several disadvantages make the kernels difficult to scale:

- Missing entities: Most kernel models could not learn node embeddings for new nodes. In the real world, graphs are dynamic, and their entities could evolve. Therefore, the graph kernels must re-learn graphs every time a new node is added, which is time-consuming and difficult to apply in practice.
- Dealing with weights: Most graph kernel models do not consider the weighted edges, which could lead to structural information loss. This could reduce the possibility of graph representation in the hidden space.
- Computational complexity: Graph kernels are an NP-hard class [109]. Although several kernel-based models aim to reduce the computational time by considering the distribution of substructures, this may increase the complexity and reduce the ability to capture the global structure.

Although the graph kernels delivered good results when working with small graphs, they remain limitations when working with large and complex graphs [118]. To address the issue, matrix factorization-based models could bring far more advantages to learning node embeddings by decomposing the large original graphs into small-sized components. Therefore, we discuss matrix factorization-based models for learning node embeddings in the next section.

3.2. Matrix Factorization-Based Models

Matrix factorization aims to reduce the high-dimensional matrix that describes graphs (e.g., adjacency matrix, Laplacian matrix) into a low-dimensional space. Several well-known decomposition models (e.g., SVD, PCA, etc.) are widely applied in graph representation learning and recommendation system problems. Table 3 and Table 4 present matrix factorization-based models for static and dynamic graphs, respectively. Based on the strategy to preserve the graph structures, matrix factorization models could be categorized into two main groups: graph embedding Laplacian eigenmaps and node proximity matrix factorization.

- The Laplacian eigenmaps: To learn representations of a graph $G = (V, E)$, these approaches first represent G as a Laplacian matrix L where $L = D - A$ and D is the degree matrix [41]. In the matrix L, the positive values depict the degree of nodes, and negative values are the weights of the edges. The matrix L could be decomposed to find the smallest number from eigenvalues which are considered node embeddings. The optimal node embedding Z^*, therefore, could be computed using an objective function:

$$Z^* = \arg\min_{Z} Z^\intercal L Z \,. \tag{5}$$

- Node proximity matrix factorization: The objective of these models is to decompose node proximity matrix into small-sized matrices directly. In other words, the proximity of nodes in graphs will be preserved in the latent space. Formally, given a proximity matrix M, the models try to optimize the distance between two pair nodes v_i and v_j, which could be defined as:

$$Z^* = \arg\min_{Z} \left\| M_{ij} - Z_i {Z_j}^T \right\|. \tag{6}$$

Hofmann et al. [119] proposed an MSDC (Multidimensional Scaling and Data Clustering) model based on matrix factorization. The key idea of MSDC is to represent data points as a bipartite graph and then learn node embeddings based on node similarity in the graph. This method requires a symmetric proximity matrix $M \in \mathbb{R}^{N \times N}$ as input and learns a latent representation of the data in Euclidean space by minimizing the loss that could be defined as:

$$Z^* = \arg\min_{Z} \frac{1}{2|V|} \sum_{(v_i, v_j) \in E} \left[\left| Z_i - Z_j \right|^2 - M_{ij} \right]^2. \tag{7}$$

However, the limitation of the MSDC model is that the model focuses only on the pairwise nodes, which cannot capture the global graph structure. Furthermore, the model investigated the proximity of all the data points in the graph, which could increase computational complexity when working on large graphs. Several models [39,120] adopted k-nearest methods to search neighbor nodes which can capture more graph structure. The k-nearest methods, therefore, could bring the advantage of reducing computational complexity since the models only take k neighbors as inputs. For example, Han et al. [120] proposed the similarity S_{ij} between two nodes v_i and v_j as:

$$S_{ij} = \begin{cases} \exp\left(-\frac{||v_i - v_j||^2}{\delta^2} \right), & \text{if } v_j \in N_k(v_i). \\ 0, & \text{otherwise.} \end{cases}, \tag{8}$$

where $N_k(v_i)$ depicts the set of k nearest neighbors of v_i in graphs. The model could measure the infringement of the constraints between pairs of nodes regarding label distribution. In addition, the model can estimate the correlation between features which would be beneficial to combine common features during the training process.

Several models [7,40,120–122] have been proposed to capture side information in graphs such as attributes and labels. He et al. [42] used the locality-preserving projection technique, a nonlinear Laplacian Eigenmap, to preserve the local structural information in graphs. The model first constructs an adjacency matrix with k nearest neighbors for each pair of nodes. The model then computes the objective function as:

$$\mathrm{a}^* = \arg\min_{\mathrm{a}} \mathrm{a}^\intercal X L X^\intercal \mathrm{a} \tag{9}$$

$$\text{subject to: } \mathrm{a}^\intercal X L X^\intercal \mathrm{a} = 1 \tag{10}$$

where D is a diagonal matrix, $L = D - A$ is the Laplacian matrix, and a is the transformation matrix in the linear embedding $x_i \rightarrow y_i = A^\intercal x_i$. Nevertheless, the idea from [42] only captures the structure within k nearest neighbors, which fails to capture the global similarity between nodes in the graph. Motivated by these limitations, Cao et al. [15] introduced the GraRep model, which considers a k-hop neighborhood of each target node. Accordingly, GraRep could capture global structural information in graphs. The model works with k-order probability transition matrix (proximity matrix) M^k which could be defined as:

$$M^k = \underbrace{M \cdots M}_{k} \tag{11}$$

where $M = D^{-1}A$, D is the degree matrix, A is the adjacent matrix, and M_{ij}^k presents the transition probability from node v_i to v_j. The loss function, thus, is the sum of k transition loss functions:

$$\mathfrak{L}_k(v_i) = \sum_{v_j \in N(v_i)} M_{ij}^k \log \sigma\left(Z_i^\intercal Z_j\right) - \left|N_{neg}\right| \sum_{v_m \sim P_n(v)} M_{im}^k \log \sigma\left(-Z_i^\intercal Z_m\right). \quad (12)$$

To construct the vector embeddings, GraRep decomposed the transition matrix into small-sized matrices using SVD matrix factorization. Similarly, Li [123] introduced NECS (Learning network embedding with the community) to capture the high-order proximity using Equation (11).

Table 3. A summary of matrix factorization-based models for static graphs. C indicates the number of clusters in graphs, $N(Z_i|\mu_c, \Sigma_c)$ refers to the multivariate Gaussian distribution for each cluster, L means the Laplacian matrix, $H \in \mathbb{R}^{n \times k}$ is the probability matrix that a node belongs to a cluster, U denotes the coefficient vector, and W_{ij} is the weight on (v_i, v_j).

Models	Graph Types	Tasks	Loss Function
SLE [39]	Static graphs	Node classification	$\sum_{v_i \in V} \sum_{v_j \in V} \left\|W_{ij}\right\| \left\|Z_i - s_{ij} Z_j\right\|_2^2$
[120]	Attributed graphs	Node classification	$\arg\min_W Tr(U^\intercal X^\intercal L X U) + \alpha \sum_{\substack{v_i \in V \\ v_j \in N(v_i)}} \left\|Z_i - Z_j\right\|_2^2 + \alpha_1 L_1 + \alpha_2 L_2$
[7]	Attributed graphs	Community detection	$-\sum_{(v_i,v_j) \in E} \log \sigma\left(Z_i^\intercal Z_j\right) - \sum_{\substack{v_i \in V \\ v_j \in N(v_i)}} \log \sigma\left(Z_i^\intercal Z_j\right)$ $-\left\|N_{neg}\right\| \sum_{v_k \sim P_n(v)} \log \sigma\left(-Z_i^\intercal Z_k\right) - \sum_{v_i \in V, c=1}^{C} N(Z_i \mid \mu_c, \Sigma_c)$
LPP [42]	Attributed graphs	Node classification	$\frac{1}{\|V\|} \sum_{v_i \in V} \|y_i - \hat{y}_i\|_2^2$
[121]	Attributed graphs	Graph reconstruction	$\frac{1}{\|V\|} \sum_{v_i \in V} \|y_i - \hat{y}_i\|_2^2$
[40]	Static graphs	Node clustering	$\sum_{v_i \in V} \sum_{c=1}^{C} \|Z_i - \mu_c\|$
GLEE [122]	Attributed graphs	Graph reconstruction, Link prediction	$\left\|L - \hat{L}\right\|^2$
LPP [42]	Static graphs	Node classification	$\sum_{(v_i,v_j) \in E} \left\|Z_i - Z_j\right\|_2^2$
Grarep [15]	Static graphs	Node classification, Node clustering	$-\sum_{v_i \in V} \sum_{v_j \in N(v_i)} A_{ij}^l \log \sigma\left(Z_i^\intercal Z_j\right)$ $-\left\|N_{neg}\right\| \sum_{v_k \sim P_n(v)} A_{ik}^l \log \sigma\left(-Z_i^\intercal Z_k\right)$
NECS [123]	Static graphs	Graph reconstruction, Link prediction, Node classification	$\left\|M - \hat{M}\right\|_F^2 + \alpha_1 \left\|H - \hat{H}\right\|_F^2 + \alpha_2 \left\|H^\intercal \hat{H} - I\right\|_F^2$
HOPE [5]	Static graphs	Graph reconstruction Link prediction, Node classification	$\|M - Z \cdot Z^\intercal\|_F^2$
[124]	Static graphs	Link prediction	$\sum_{(v_i,v_j) \in S} A_{ij} \left\|Z_i - Z_j\right\|_2^2$
AROPE [86]	Static graphs	Graph reconstruction, Link prediction, Node classification	$\|M - Z \cdot Z^\intercal\|_F^2$
ProNE [43]	Static graphs	Node classification	$-\sum_{v_i \in V} \left[\sum_{v_j \in N(v_i)} \log \sigma\left(Z_i^\intercal Z_j\right) + \sum_{v_k \sim P_n(v)} \log \sigma\left(-Z_i^\intercal Z_k\right)\right]$
ATP [6]	Static graphs	Link prediction	$\|M - Z \cdot Z^\intercal\|_F^2$
[125]	Static graphs	Graph partition	$\sum_{\substack{(v_i,v_j) \in E \\ v_i, v_j \in V_k}} \left(W_{ij} - \left\langle Z_i^{(k)}, Z_j^{(k)} \right\rangle\right)^2 + \sum_{v_i \in V_k} \left\|Z_i - \hat{Z}_i\right\|_2^2$
NRL-MF [126]	Static graphs	Node classification	$\sum_{\substack{v_i \in V \\ v_j \in N(v_i)}} \left\|Z_i - Z_j\right\|_2^2$

In terms of considering the node proximity based on neighbor relations, Ou et al. [5] presented HOPE, an approach for preserving structural information in graphs using k-order proximity. In contrast to GraRep, HOPE tried to solve the asymmetric transitivity problem in directed graphs by approximating high-order proximity. The objective function needs to be minimized for the approximation proximity could be defined as:

$$Z^* = \arg\min_{Z} \left\| M_{ij} - Z_i^\intercal Z_j \right\|_2^2 \tag{13}$$

where M is the high-order proximity matrix, for instance, M_{ij} presents the proximity of two nodes v_i and v_j, Z_i and Z_j denote vector embeddings of v_i and v_j, respectively. The proximity matrix M can be measured by decomposing into two small-sized matrices $\left(M = M_g^{-1} \cdot M_l\right)$. Several common criteria could measure the node proximity, such as Katz Index [127], Rooted PageRank [128], Adamic-Adar [129], and Common Neighbors. Coskun and Mustafa [124] suggested changes in the proximity measure formulas of the HOPE model. For nodes that have a small degree, singular values could be zero after measuring the node proximity. Therefore, to solve this problem, they added a parameter σ to regularize the Laplacian graph.

A few models have been proposed with the same idea as HOPE and GraRep [43,86]. For example, ProNE model [43] aimed to use k number of the Chebyshev expansion to avoid Eigen decomposition, instead of using k-order proximity in HOPE models. Sun et al. [6] introduced a similar approach for preserving asymmetric transitivity with high-order proximity. However, the significant difference is that they proposed a strategy to break directed acyclic graphs while preserving the graph structure. The non-negative matrix factorization could then be applied to produce an embedding matrix. Several models [125,130,131] mainly focused on the pointwise mutual information (PMI) of nodes in graphs which calculates the connection between nodes in terms of linear and nonlinear independence. Equation (5) is used to learn node embeddings.

Several models aimed to reduce computational complexity from matrix factorization by improving the sampling strategies [126,132,133]. For instance, the key idea of the NRL-MF model [126] was to deal with a hashing function for computing dot products. Each node is presented as a binarized vector by a hashing function, which can be calculated faster by XOR operators. The model could learn the binary and quantized codes based on matrix factorization and preserve high-order proximity. Jiezhong [133] targeted sparse matrix factorization. They implemented random-walk sampling on graphs to construct a NetMF Matrix Sparsifier. RNP model [132] explored in-depth vector embeddings based on personalized PageRank values, then approximated the PPR matrices.

Table 4. A summary of matrix factorization-based models for heterogeneous graphs and dynamic graphs. $H \in \mathbb{R}^{n \times k}$ is the probability matrix that a node belongs to a cluster, $E^{(t)}$ is the edge matrix with type t, W_{ij} is the weight on (v_i, v_j), r denotes the relation type, and $E^{(1,2)}$ is the set of edges in two component graphs G_1 and G_2.

Models	Graph Types	Tasks	Loss Function
DBMM [134]	Dynamic graphs	Node classification, Node clustering	$\left\| A - \hat{A} \right\|_F^2$
[135]	Dynamic graphs	Link prediction	$\left\| A - \hat{A} \right\|_2^2 + \alpha L_2$
[136]	Dynamic graphs	Link prediction	$\left\| A - \hat{A} \right\|_F^2$
LIST [137]	Dynamic graphs	Link prediction	$\left\| A - \hat{A} \right\|_F^2 + L_2$
TADW [131]	Attributed graphs	Node classification	$\|M - W^\intercal HX\|_F^2 + \alpha\left(\|W\|_F^2 + \|H\|_F^2\right)$
PME [138]	Heterogeneous graphs	Link prediction	$\sum_{\substack{(v_i,v_j)\in E^{(r)} \\ (v_i,v_k)\notin E^{(r)}}} \left(\left\|Z_i - Z_j\right\|_2^2 - \|Z_i - Z_{n_k}\|_2^2 + m\right)$

Table 4. *Cont.*

Models	Graph Types	Tasks	Loss Function
EOE [139]	Heterogeneous graphs	Node classification	$\sum_{(v_i,v_j)\in E^{(1,2)}} \sigma\left(Z_i^\intercal Z_j\right) - \sum_{(v_l,v_k)\notin E^{(1,2)}} \sigma\left(-Z_i^\intercal Z_j\right) + \alpha_1 L_1 + \alpha_2 L_2$
[130]	Heterogeneous graphs	Link prediction	$\sum_{(v_i,v_j)\in E^{(t)}} \left\|Z_i^{(t)} - \hat{Z}_i^{(t)}\right\|_F^2 + \alpha_1 L_1 + \alpha_2 L_2$
ASPEM [140]	Heterogeneous graphs	Node classification, Link prediction	$-\sum_{\substack{v_i\in V \\ (v_i,v_j,r)\in E}} \log\left(p(v_i\|v_j,r)\right)$
MELL [141]	Heterogeneous graphs	Link prediction	$-\sum_{(v_i,v_j)\in E} \sigma\left(Z_i^\intercal Z_j\right) - \sum_{(v_i,v_k)\notin E} \sigma\left(1 - Z_i^\intercal Z_k\right) + \alpha L_2$
PLE [142]	Attributed graphs	Node classification	$\sum_{(v_i,v_j)\in E} \log\sigma(Z_i Z_j) + \left\|N_{neg}\right\| E_{v_k\sim P_n(v_k)}(\log\sigma(-Z_i Z_k))$

In the real world, several graphs often contain attributes for nodes and edges, such as user profiles on a social network. These attributes provide helpful information to improve the node representation and help to learn node embedding. Yang et al. [131] proposed the TADW model by representing the DeepWalk model as a matrix factorization and integrating text features into the factorization model. Ren et al. [142] introduced the PLE model to learn jointly different types of nodes and edges with text attributes. Since existing models often ignore the noise of labels, PLE is the first work to investigate the noisy type labels by measuring the similarity between entities and type labels.

Beyond static and homogeneous graphs, several models have been proposed to learn embeddings in dynamic and heterogeneous graphs. The embedding models for dynamic graphs are essentially the same as for static graphs, including Laplacian eigenmaps methods and node proximity matrix factorization to model relations in dynamic graphs over time. For Laplacian eigenmaps methods, Li et al. [81] presented DANE (Dynamic Attributed Network Embedding) model to learn node embeddings in dynamic graphs. The main idea of the DANE model is to represent a Laplacian matrix as $L_A^{(t)} = D_A^{(t)} - A^{(t)}$, where $A^{(t)} \in R^{n\times n}$ is the adjacency matrix of dynamic graphs at time t, D_A is the diagonal matrix, then the model could be able to learn node embeddings by time in an online manner. To preserve the node proximity, the DANE model aimed to minimize the loss function:

$$\mathfrak{L}(v_i, v_j) = \sum_{\substack{(v_i,v_j) \\ i\neq j}} A_{ij}^{(t)} \left\|Z_i - Z_j\right\|_2^2 . \tag{14}$$

The eigenvectors λ of the Laplacian matrix L can be calculated by solving the generalized eigenproblem: $L_A^{(t)} a = \lambda D_A^{(t)}$, where $a = \langle a_0\, a_1\, \cdots\, a_N\rangle$ is the eigenvectors.

Several models applied node proximity matrix factorization directly to dynamic graphs by updating the proximity matrix between entities in the dynamic graphs. Rossi et al. [134] presented dynamic graphs as a set of static graph snapshots: $G = \{G(t_0)\, G(t_1)\, \cdots\, G(t_N)\}$. The model then learned a transition proximity matrix T, which describes all transitions from the dynamic graphs. For evaluation, they predict the graph G at time $t+1$: $\widehat{G}_{t+1} = G_t T_{t+1}$, then estimate the error using Frobenius loss: $\left\|\widehat{G}_{t+1} - G_{t+1}\right\|_F$. Zhu et al. [135,137] aimed to preserve the graph structure based on temporal matrix factorization during the network evolution. Given an adjacency matrix $A(t)$ at time t, two temporal rank-k matrix factorization U and $V(t)$ are factorized as $A(t) = f(UV(t)^\intercal)$, and the objective is to minimize the loss function $\mathfrak{L}(A)$ which could be defined as:

$$\mathfrak{L}(A) = \sum_{t=1}^{T} \frac{D(t)}{2} \left\|A(t) - \hat{A}(t)\right\|_F^2 . \tag{15}$$

Matrix factorization models have been successfully applied to graph embedding, mainly for the node embedding problem. Most models are based on singular value

decomposition to find eigenvectors in the latent space. There are several advantages of matrix factorization-based models:

- Training data requirement: The matrix factorization-based models do not need much data to learn embeddings. Compared to other methods, such as neural network-based models, these models bring advantages in case there is little training data.
- Coverage: Since the graphs are presented as Laplacian matrix L, or transition matrix M, then the models could capture all the proximity of the nodes in the graphs. The connection of all the pairs of nodes is observed at least once time under the matrix that makes the models could be able to handle sparsity graphs.

Although matrix factorization is widely used in graph embedding problems, it still has several limitations:

- Computational complexity: The matrix factorization suffers from time complexity and memory complexity for large graphs with millions of nodes. The main reason is the time it takes to decompose the matrix into a product of small-sized matrices [15].
- Missing values: Models based on matrix factorization cannot handle incomplete graphs with unseen and missing values [143,144]. When the graph data are insufficient, the matrix factorization-based models could not learn generalized vector embeddings. Therefore, we need neural network models that can generalize graphs and better predict entities in graphs.

3.3. Shallow Models

This section focuses on shallow models for mapping graph entities into vector space. These models mainly aim to map nodes, edges, and subgraphs as low-dimensional vectors while preserving the graph structure and entity proximity. Typically, the models first implement a sampling technique to capture graph structure and proximity relation and then learn embeddings based on shallow neural network algorithms. Several sampling strategies could be taken to capture the local and global information in graphs [14,145,146]. Based on the sampling strategy, we divide shallow models into two main groups: structure preservation and proximity reconstruction.

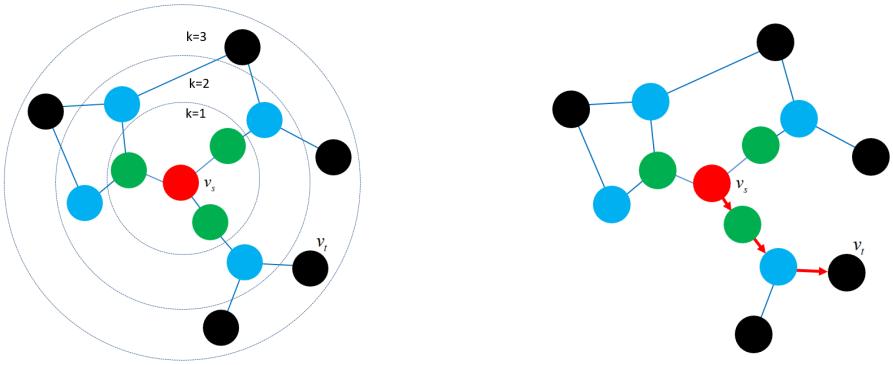
- Structure preservation: The primary concept of these approaches is to define sampling strategies that could capture the graph structure within fixed-length samples. Several sampling techniques could capture both local and global graph structures, such as random-walk sampling, role-based sampling, and edge reconstruction. The model then applies shallow neural network algorithms to learn vector embeddings in the latent space in an unsupervised learning manner. Figure 6a shows an example of a random-walk-based sampling technique in a graph from a source node v_s to a target node v_t.
- Proximity reconstruction: It refers to preserving a k-hop relationship between nodes in graphs. The relation between neighboring nodes in the k-hop distance should be preserved in the latent space. For instance, Figure 6b presents a 3-hop proximity from the source node v_s.

In general, shallow models have achieved many successes in the past decade [4,14,21]. However, there are several disadvantages of shallow models:

- Unseen nodes: When there is a new node in graphs, the shallow models cannot learn embeddings for new nodes. To obtain embedding for new nodes, the models must update new patterns, for example, re-execute random-walk sampling to generate new paths for new nodes, and then the models must be re-trained to learn embeddings. The re-sampling and re-training procedures make it impractical to apply them in practice.
- Node features: Shallow models such as DeepWalk and Node2Vec mainly work suitably on homogeneous graphs and ignore information about the attributes/labels of nodes. However, in the real world, many graphs have attributes and labels that could be informative for graph representation learning. Only a few studies have investigated the attributes and labels of nodes, and edges. However, the limitations of domain

knowledge when working with heterogeneous and dynamic graphs have made the model inefficient and increased the computational complexity.

- Parameter sharing: One of the problems of shallow models is that these models cannot share the parameters during the training process. From the statistical perspective, parameter sharing could reduce the computational time and the number of weight updates during the training process.



(**a**) (**b**)

Figure 6. Node sampling techniques. (**a**) *k*-hop sampling; (**b**) Random-walk sampling. The source node v_s and the target node v_t are taken as the source node and the target node in the graph. In (**a**), the *k*-hop proximity sampling strategy begins from source node v_s, and the green nodes are considered to be the 1*st*-hop proximity of node v_s. The blue and the black nodes are considered 2nd-hop and 3rd-hop proximity of node v_s, respectively. In (**b**), the random-walk sampling strategy takes a random walk (red arrow) from the source node v_s to the target node v_t.

3.3.1. Structure-Preservation Models

Choosing a strategy to capture the graph structure is essential for shallow models to learn vector embeddings. The graph structure can be sampled through connections between nodes in graphs or substructures (e.g., subgraphs, motifs, graphlets, roles, etc.). Table 5 briefly summarizes structure-preservation models for static and homogeneous graphs.

Over the last decade, various models have been proposed to capture the graph structure and learn embeddings [4,21,147,148]. Among those models, random-walk-based strategies could be considered one of the most typical strategies to sample the graph structures [4,14]. The main idea of the random-walk strategy is to gather information about the graph structure to generate paths that can be treated as sentences in documents. The definition of random walks could be defined as:

Definition 6 (Random walk [14])**.** *Given a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges, a random walk with length l is a process starting at a node $v_i \in V$ and moving to its neighbors for each time step. The next steps are repeated until the length l is reached.*

Two models, DeepWalk [14] and Node2Vec [4] could be considered to be pioneer models to open a new direction for learning node embeddings.

Inspired by the disadvantages of the matrix factorization-based models, the DeepWalk model could preserve the node neighborhoods based on random-walk sampling, which could capture global information in graphs. Moreover, both DeeWalk and Node2Vec aim to maximize the probability of observing node neighbors by stochastic gradient descent on each single-layer neural network. Therefore, these models reduce running time and computational complexity. DeepWalk [14] is a simple node embedding model using the random-walk sampling strategy to generate node sequences and treat them as word sentences. The objective of DeepWalk is to maximize the probability of the set of neighbor

nodes $N(v_i)$ given a target node v_i. Formally, the optimization problem could be defined as:

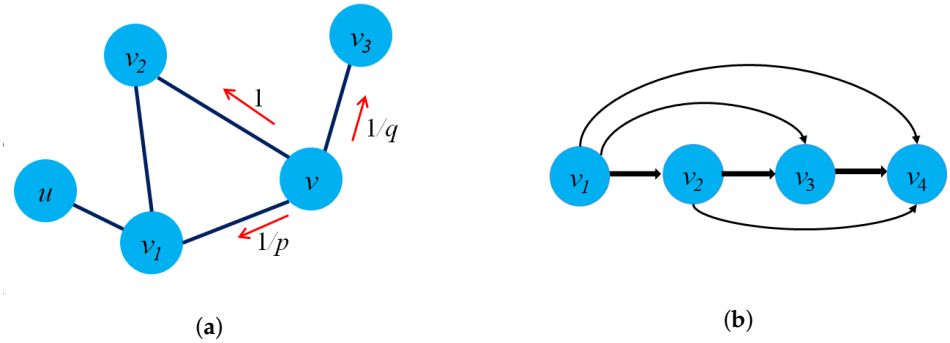
$$\phi^*(\cdot) = \arg\min_{\phi(\cdot)} -\log p(N(v_i)|\phi(v_i)) \tag{16}$$

where v_i denotes the target node, $N(v_i)$ is the set of neighbors of v_i which could be generated from random-walk sampling, $\phi(v_i)$ is the mapping function $\phi : v_i \in V \rightarrow R^{|V|\times d}$. The model uses two strategies for finding neighbors given a source node, based on the Breadth-First Search (BFS) and Depth First Search (DFS) strategies. The BFS strategy aims to represent a microscopic view that captures the local structure. In contrast, the DFS strategy delivers the global structure information in graphs. The DeepWalk then uses a skip-gram model and stochastic gradient descent (SGD) to learn latent representations.

Table 5. A summary of structure-preservation models for homogeneous and static graphs. K indicates the number of clusters in the graph, and μ_k refers to the mean value of cluster k.

Models	Graph Types	Tasks	Loss Function
DeepWalk [14]	Static graphs	Node classification	$-\sum_{v_i\in V} y_i^\intercal \log(\hat{y}_i)$
Node2Vec [4]	Static graphs	Node classification, Link prediction	$-\sum_{v_i\in V} y_i^\intercal \log(\hat{y}_i)$
WalkLets [147]	Static graphs	Node classification	$-\sum_{v_i\in V} y_i^\intercal \log(\hat{y}_i)$
Div2Vec [149]	Static graphs	Link prediction	$-\sum_{v_i\in V} y_i^\intercal \log(\hat{y}_i)$
	Static graphs	Node classification	$-\sum_{v_i\in V} y_i^\intercal \log(\hat{y}_i)$
Node2Vec+ [148]	Static graphs	Node classification	$-\sum_{v_i\in V} y_i^\intercal \log(\hat{y}_i)$
Struct2Vec [21]	Static graphs	Node classification	$-\sum_{v_i\in V, \langle v_i,v_j\rangle\in E} \log\left(\sigma(Z_i^\intercal Z_j)\right) - \lvert N_{neg}\rvert \sum_{v_k\sim P_n(v)} \log(\sigma(-Z_i^\intercal Z_k))$
DiaRW [150]	Static graphs	Node classification, Link prediction	$\sum_{v_i\in V} -y_i^\intercal \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i)$
Role2Vec [151]	Attributed graphs	Link prediction	$\sum_{k=1}^{K} \sum_{v_i\in V_k} \lVert Z_i - \mu_k\rVert_2^2$
NERD [152]	Directed graphs	Link Prediction, Graph Reconstruction, Node classification	$\sum_{(v_i,v_j)\in E} \log\sigma(Z_i Z_j) + \lvert N_{neg}\rvert E_{v_k\sim P_n(v_k)}(\log\sigma(-Z_i Z_k))$
Sub2Vec [153]	Static graphs	Community detection, graph classification	$\sum_{k=1}^{K} \sum_{v_i\in V_k} \lVert Z_i - \mu_k\rVert_2^2$
Subgraph2Vec [145]	Static graphs	Graph classification, Clustering	$-\sum_{v_i\in V} y_i^\intercal \log(\hat{y}_i)$
RUM [89]	Static graphs	Node classification, Graph reconstruction	$-\sum_{v_i\in V} y_i^\intercal \log(\hat{y}_i)$
Gat2Vec [154]	Attributed graphs	Node classification, Link prediction	$-\sum_{v_i\in V, \langle v_i,v_j\rangle\in E} \log\left(\sigma(Z_i^\intercal Z_j)\right) - \lvert N_{neg}\rvert \sum_{v_k\sim P_n(v)} \log(\sigma(-Z_i^\intercal Z_k))$
ANRLBRW [155]	Attributed graphs	Node classification	$-\sum_{v_i\in V, \langle v_i,v_j\rangle\in E} \log\left(\sigma(Z_i^\intercal Z_j)\right) - \lvert N_{neg}\rvert \sum_{v_k\sim P_n(v)} \log(\sigma(-Z_i^\intercal Z_k))$
Gl2Vec [88]	Static graphs	Node classification	$-\sum_{v_i\in V} y_i^\intercal \log(\hat{y}_i)$

One of the limitations of DeepWalk is that the model can only capture the graph structure but fail to navigate the random-walk sampling to enrich the quality of the sampling graph structure. To overcome the limitations of DeepWalk, Grover and Leskovec introduced Node2Vec [4] with a flexible random-walk sampling strategy to navigate random walks via each time step. The key difference between DeepWalk and Node2Vec is that instead of using a truncated random walk, the model used a biased random-walk sampling process with two parameters (p and q) to adjust the random walk on graphs. Figure 7a presents two parameters p and q in Node2Vec model in detail. The model could capture more information on the graph structure locally and globally by introducing constraints when deciding the subsequent nodes visited.



(**a**) (**b**)

Figure 7. Sampling strategy in Node2Vec and WalkLets model. (**a**) Sampling strategy in Node2Vec model; (**b**) Sampling strategy in WalkLets model. In (**a**), assume a random path from the DeepWalk model is of the form: $(v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4)$, then the corpus of random walk pairs at scale $k = 3$ is: $A_1 = \{(v_1, v_2), (v_2, v_3), (v_3, v_4)\}$, $A_2 = \{(v_1, v_3), (v_2, v_4)\}$, and $A_3 = \{(v_1, v_4)\}$. In (**b**), there are two parameters: the return parameter p and the in–out parameter q. Parameters $1, 1/p$, and $1/q$ are conditional probabilities. Starting at node u and now at v, the random walk looks at the next node based on the probabilities $1/p$ and $1/q$.

Perozzi et al. [147] presented the WalkLets model, which was extended from the DeepWalk model. They modified the random-walk sampling strategy to capture more graph structure information by skipping and passing over multiple nodes at each time step. Therefore, these sampling strategies can capture more global graph structure by the power of the transition matrix when passing over multiple nodes. The main idea of the WalkLets model is to represent the random-walk paths as pairs of nodes in the multi-scale direction. Figure 7b depicts the sampling strategy of the WalkLets model using multi-scale random-walk paths. However, one of the limitations of the WalkLets is that the model could not distinguish local and global structures when passing and skipping over nodes in graphs. Jisu et al. [149] presented a variation of DeepWalk, named Div2Vec model. The main difference between the two models is the way that Div2Vec chooses the next node in the random-walk path, which will be visited based on the degree of neighboring nodes. The focus on the degree of neighboring nodes could help the models learn the importance of nodes that are popular in social networks. Therefore, at the current node v_i, the probability of choosing the next node v_j in a random-walk path is calculated as:

$$p(v_j|v_i) = \frac{f(\deg(v_j)}{\sum_{v_i \in N} f(\deg(v_i))} \tag{17}$$

where $\deg(v_j)$ depicts the degree of node v_j, and $f(\deg(v_j)) = \frac{1}{\deg(v_j)}$. Renming et al. [148] presented Node2Vec+, an improved version of Node2Vec. One limitation of the Node2Vec model is that it cannot determine the following nodes based on the target nodes. There is a significant difference between Node2Vec and Node2Vec+. The Node2Vec+ model can determine the state of the potential edge for a given node, therefore enhancing the navigability of the Node2Vec model to capture more graph structure. In particular, they introduced three neighboring edge states from a current node (out edge, noisy edge, and in edge) which are calculated to decide the next step. With potential out edges $(v_i, v_j) \in E$ from previous node t, the in–out parameters p and q of Node2Vec model could then be re-defined as bias factor α as:

$$\alpha_{pq}(t, v_i, v_j) = \begin{cases} \frac{1}{p} & \text{if } t = x. \\ 1 & \text{if } w(v_j, t) \geq \tilde{d}(v_j). \\ \min\{1, \frac{1}{p}\} & \text{if } w(v_j, t) < \tilde{d}(v_j) \text{ and } w(v_j, t) < \tilde{d}(v_i). \\ \frac{1}{q} + (1 - \frac{1}{q}) \frac{w(v_j, t)}{d(x)} & \text{if } w(v_j, t) < \tilde{d}(v_j) \text{ and } w(v_j, t) \geq \tilde{d}(v_i). \end{cases} \tag{18}$$

where $\tilde{d}(v_i)$ denotes a noisy edge threshold which could consider the next node state v_j from the current node t and could be viewed as the weights of edges, $w(v_i, v_j)$ is the weight of the edge between v_i and v_j.

In contrast to preserving graph topology which mainly focuses on distance relations, several models aimed to preserve the role and importance of nodes in graphs. In the case of social networks, for example, we could discover influencers with the ability to impact several activities of communities. In contrast to the random-walk-based technique, several studies [21,150] used the term "role-based" to preserve the nodes' role, which random-walk-based sampling strategies cannot capture in a fixed length. Therefore, by preserving the role of nodes, role-based models could capture the structural equivalent. Ribeiro et al. [21] introduced the Struc2Vec model to capture graph structure based on the nodes' role. Nodes that have the same degree should be encoded close in the vector space. Given a graph G, they introduced k graphs, each graph can be considered in one layer. Each layer denotes a graph that describes the weighted node degree from different hop distances. Specifically, at layer L_k, for each node $v_i \in V$, there are three probabilities of going to node v_j in the same layer, jumping to the previous layer L_{k-1} and next layer L_{k+1}:

$$p_k(v_i^k, v_j^k) = \frac{e^{-f_k(v_i^k, v_j^k)}}{Z_k(v_i^k)} \tag{19}$$

$$p_k(v_i^k, v_i^{k+1}) = \frac{w(v_i^k, v_i^{k+1})}{w(v_i^k, v_i^{k+1}) + w(v_i^k, v_i^{k-1})} \tag{20}$$

$$p_k(v_i^k, v_i^{k-1}) = 1 - p_k(v_i^k, v_i^{k+1}) \tag{21}$$

where $f_k(v_i\, v_j)$ presents the role-based distance between nodes v_i and v_j, and $w(\cdot)$ denotes the edge weight. Zhang et al. [150] presented the DiaRW model, which uses a random-walk strategy based on the node degree. The difference between other role-based models and the DiaRW model is that they used random walks that can vary in length based on the node degree. One of the limitations of the Struc2Vec model is that the model could not preserve the similarity of nodes in graphs. Motivated by this limitation, the DiaRW model aims to capture structural identity based on node degree and the neighborhood in which nodes have a high degree. The purpose of this model is to collect structural information around higher-order nodes, which is a limitation of models based on fixed-length random walks. Ahmed et al. [151] introduced the Role2Vec model that could capture the node's similarity and structure by introducing a node-type parameter to guide random-walk paths. The core idea of Role2Vec is that nodes in the same cluster should be sampled together in the random-walk path. By only sampling nodes in the same clusters, Role2Vec could learn correct patterns with reduced computational complexity. The model then uses the skip-gram model to learn node embeddings. Unlike Rol2Vec, the NERD model [152] considers nodes' asymmetric roles for directed graphs. The model sampled the neighbor's nodes using an alternative random walk. The probability of the next node v_{i+1} from the current node v_i in the random-walk path could be defined as:

$$p(v_{i+1}|v_i) = \begin{cases} \frac{1}{d^{out}(v_i)} \cdot w(v_i, v_j) & \text{if } (v_i, v_j) \in \text{E}\,. \\ \frac{1}{d^{in}(v_i)} \cdot w(v_i, v_j) & \text{if } (v_j, v_i) \notin \text{E}\,. \\ 0 & \text{otherwise}\,. \end{cases} \tag{22}$$

where $w(v_i, v_j)$ is the weight of the edge e_{ij} $d^{in}(v_i)$ and $d^{out}(v_i)$ present the total in-degree and out-degree of the node v_i, respectively.

In some types of graphs, nodes in the same subgraphs tend to have similar labels. Studying low-level node representation could not bring significant generalization. Instead of embedding individual nodes in graphs, several studies aimed to learn subgraph similarity or the whole graphs. Inspired by representations of sentences and documents in the

NLP (natural language processing) area, Bijaya et al. [153] proposed the Sub2Vec model to embed each subgraph into a vector embedding.

To learn a subgraph embedding $S = \{G_1\, G_2\, \cdots\, G_n\}$ from an original graph G, two properties should be preserved: similarity and structural property. The former ensures the connection between subgraph nodes by collecting sets of paths in a subgraph. The latter ensures that each node in a subgraph should be densely connected to all other nodes in the same subgraph. Figure 8 presents two subgraph properties that could capture each subgraph connection and structure.

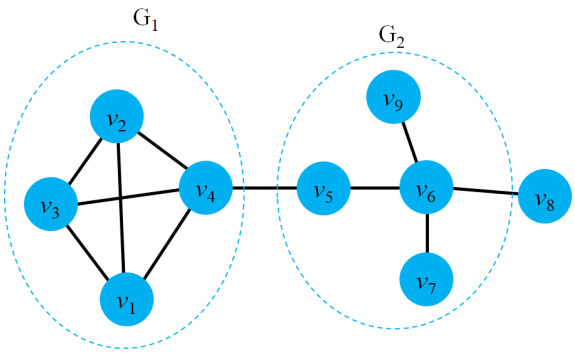


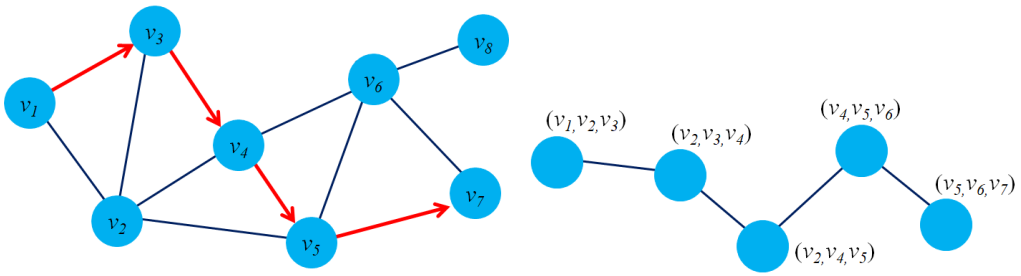
Figure 8. Sampling strategy in Sub2Vec model. Assume that there are two subgraphs $G_1 = \{v_1, v_2, v_3, v_4\}$, and $G_2 = \{v_5, v_6, v_7, v_9\}$. For neighborhood properties, the model uses random-walk sampling on all nodes in subgraphs G_1 and G_2 to capture the subgraph structure. For structural properties, they introduced a ratio of node degree when sampling. With the length of the random-walk path is 3, then the degree path for G_1 is $0.75 \rightarrow 0.75 \rightarrow 0.75$, while the degree path from node v_5 to v_9 is: $0.25 \rightarrow 0.75 \rightarrow 0.25$.

In contrast to Sub2Vec, Subgraph2Vec [145] aimed to learn rooted subgraph embeddings for detecting Android malware. One of the advantages of this model with the Sub2Vec model is that Subgraph2Vec could consider different degrees of rooted subgraphs surrounding the target subgraph while Sub2Vec tried to detect the community. Annamalai et al. [156] targeted embedding the entire graph into the latent space. With the same idea as the Subgraph2Vec model, they extracted the set of subgraphs from the original graph using the WL relabeling strategy. However, the difference is that they used the Doc2Vec model by treating documents as graphs to learn graph embeddings.

Most models mentioned above aim to capture the graph structure based on low-level node representation, which could fail to represent the higher-level structure. Therefore, finding the community structure can be difficult for models based on random-walk sampling strategies. Motif-based models are one of the strategies to preserve the local structure and discover the global structure of graphs. Yanlei et al. [89] proposed the RUM (network Representation learning Using Motifs) model to learn small groups of nodes in graphs. The main idea of RUM was to build a new graph $G' = (V', E')$ based on the original graph by constructing new nodes and edges as follows:

- Generating nodes in graph G': Each new node v in graph G' is a tuple $v_{ijk} = \langle v_i, v_j, v_k \rangle$ in the original graph G. Therefore, they can map the triangle patterns of the original graph to the new graph for structure preservation.
- Generating edges of graph G': Each edge of the new graph is formed from two nodes that have two edges in common in the original graph. For example, the edge $e = (v_{ijk}, v_{ijl})$ denotes that we the edge $(v_i, v_j) \in E$ in the original graph G.

The model then used the skip-gram model to learn the node and motif embeddings. Figure 9b depicts the details of the random-walk sampling strategy based on motifs.



(**a**) (**b**)

Figure 9. The random-walk sampling based on motif. (**a**) Random-walk sampling; (**b**) Motif-based random-walk sampling. (**a**) presents a random-walk path from node v_1 to v_7: $v_1 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_7$. In (**b**), the motif-based path is: $\langle v_1, v_2, v_3 \rangle \rightarrow \langle v_2, v_3, v_4 \rangle \rightarrow \langle v_2, v_4, v_5 \rangle \rightarrow \langle v_4, v_5, v_6 \rangle$.

There are also several models based on motifs for heterogeneous graphs [87,90,91]. For instance, Qian et al. [90] proposed the MBRep model (Motif-based representation) with the same idea from the RUM model to generate a hyper-network based on a triangle motif. However, the critical difference is that the MBRep model could extract motifs based on various node and edge types in heterogeneous graphs.

Most of the above models aim to learn node embeddings without side information, which could be informative for learning graph structure. However, graphs in the real world could be composed of side information, such as attributes of nodes and edges. Several models tried to learn node embeddings in attributed graphs by adding node properties presented as attributed graphs. Nasrullah et al. [154] proposed the Gat2Vec model to capture the contextual attributes of nodes. Given by a graph $G = (V, E, X)$ where X is the attribute function $X : V \rightarrow 2^X$, they generated a structural graph G_s and a bipartite graph G_a as:

$$G_s = (V_s\, E) \tag{23}$$

$$G_a = (V_a\, X\, E_a) \tag{24}$$

where $V_s \subseteq V$, $V_a = \{v_i : X(v_i) \neq \varnothing\}$, $V_a \subseteq V$, and $E_a = \{(v_i, a), a : X(v_i)\}$. They then used the random-walk sampling strategy to capture the graph structure in both types of graphs. Similar to Gat2Vec, Wei et al. [155] introduced the ANRLBRW model (Attributed Network Representation Learning Based on Biased Random Walk) with the idea of splitting the original graph G into a geological graph and attributed graph. However, there is a slight difference between the two models. ANRLBRW model used a biased random-walk sampling inspired by Node2Vec, which includes two parameters p and q in the sampling strategy. Kun et al. [88] introduced the Gl2Vec model to learn node embeddings based on graphlets. To generate the feature representation for graphs, they capture the proportion of graphlet occurrences in a graph compared with random graphs.

For social networks, the connections of nodes are far more complex than the node-to-node edge relationship, which constructs hypergraphs. In contrast to homogeneous graphs, edges in hypergraphs could connect more than two nodes in graphs which leads to difficult learning node embeddings. Several models have been proposed to learn node and edge embeddings in the hypergraphs [157,158]. For example, Yang et al. [157] proposed the LBSN2Vec (Location Based Social Networks) model, a hypergraph embedding model to learn hyperedges including both user-user connection and user-check-in locations over time. Since most existing models fail to capture mobility features and co-location rates dynamically, the model could learn the impact of user mobility in social networks for prediction tasks. The objective of this model is to use a random-walk-based sampling strategy on hyperedges with a sequence length to capture the hypergraph structure. They then use cousin similarity to preserve nodes' proximity in the random-walk sequences. Table 6 lists a summary of representative models for heterogeneous graphs.

Table 6. A summary of structure-preservation models for heterogeneous graphs and dynamic graphs. K is the number of clusters in graphs, N_{neg} refers to the number of negative samples, and P_n means the noise distribution.

Models	Graph Types	Tasks	Loss Function
MBRep [90]	Hypergraphs	Link prediction	$-\sum_{v_i \in V, \langle v_i, v_j \rangle \in E} \log\left(\sigma(Z_i^\intercal Z_j)\right) - \lvert N_{neg} \rvert \sum_{v_k \sim P_n(v)} \log(\sigma(-Z_i^\intercal Z_k))$
Motif2Vec [87]	Heterogeneous graphs	Node classification link prediction	$\sum_{v_i \in V} \sum_{v_j \in N(v_i)} -\log\left(p(v_j \lvert Z_i)\right)$
JUST [159]	Heterogeneous graphs	Node classification Node clustering	$\sum_{(v_i,v_j)\in E} \log \sigma(Z_i Z_j) + \lvert N_{neg} \rvert E_{v_k \sim P_n(v_k)}(\log \sigma(-Z_i Z_k))$
[160]	Multiplex graphs	Link prediction	$\sum_{(v_i,v_j)\in E} \log \sigma(Z_i Z_j) + \lvert N_{neg} \rvert E_{v_k \sim P_n(v_k)}(\log \sigma(-Z_i Z_k))$
BHIN2Vec [161]	Heterogeneous graph	Node classification	$-\sum_{v_i \in V} \left(y_i^\intercal \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right)$
[162]	Heterogeneous graphs	Link prediction	$\frac{1}{\lvert V \rvert} \sum_{v_i \in V} \lVert y_i - \hat{y}_i \rVert_2^2$
[163]	Heterogeneous graphs	Link prediction	$\frac{1}{\lvert V \rvert} \sum_{v_i \in V} \lVert y_i - \hat{y}_i \rVert_2^2$
[164]	Heterogeneous graphs	Link prediction	$\frac{1}{\lvert V \rvert} \sum_{v_i \in V} \lVert y_i - \hat{y}_i \rVert_2^2$
[165]	Heterogeneous graphs	Link prediction	$\frac{1}{\lvert V \rvert} \sum_{v_i \in V} \lVert y_i - \hat{y}_i \rVert_2^2$
[166]	Heterogeneous graphs	Entities prediction	$\frac{1}{\lvert V \rvert} \sum_{v_i \in V} \lVert y_i - \hat{y}_i \rVert_2^2$
MrMine [167]	Multiplex graphs	Graph classification	$-\sum_{v_i \in V, \langle v_i, v_j \rangle \in E} \log\left(\sigma(Z_i^\intercal Z_j)\right) - \lvert N_{neg} \rvert \sum_{v_k \sim P_n(v)} \log(\sigma(-Z_i^\intercal Z_k))$
[168]	Heterogeneous graph	Link prediction	$\sum_{(v_i,v_j)\in E} \left[\log \sigma\left(Z_i^\intercal Z_j\right) - \sum_{k=1}^{n} \left(E_{v_k \sim P(v_i)} \log \sigma(Z_i^\intercal Z_k)\right)\right]$
[169]	Dynamic graphs	Node classification	$-\sum_{v_i \in V} y_i^\intercal \log(\hat{y}_i)$
[170]	Dynamic graphs	Node classification	$-\sum_{v_i \in V, \langle v_i, v_j \rangle \in E} \log\left(\sigma(Z_i^\intercal Z_j)\right) - \lvert N_{neg} \rvert \sum_{v_k \sim P_n(v)} \log(\sigma(-Z_i^\intercal Z_k))$
[171]	Dynamic graphs	Link prediction	$-\sum_{i=1}^{N} [y_i \log \hat{y}_i + \alpha(1-y_i)\log(1-\hat{y}_i)]$
STWalk [172]	Dynamic graphs	Node classification	
[173]	Dynamic graphs	Node classification, Link prediction	$-\sum_{v_i \in V} y_i^\intercal \log(\hat{y}_i)$
[174]	Dynamic graphs	Link prediction, Node classification	$-\sum_{v_i \in V} \sum_{v_j \in N(v_i)} \log(\hat{Z}_j)$
Dyn2Vec [10]	Dynamic graphs	Node classification	$-\sum_{v_i \in V, \langle v_i, v_j \rangle \in E} \log\left(\sigma(Z_i^\intercal Z_j)\right) - \lvert N_{neg} \rvert \sum_{v_k \sim P_n(v)} \log(\sigma(-Z_i^\intercal Z_k))$
[92]	Dynamic graphs	Link prediction	$\frac{1}{\lvert V \rvert} \sum_{v_i \in V} [y_i \log \hat{y}_i + \alpha(1-y_i)\log(1-\hat{y}_i]$
T-EDGE [175]	Dynamic graphs	Node classification	$-\sum_{v_i \in V, \langle v_i, v_j \rangle \in E} \log\left(\sigma(Z_i^\intercal Z_j)\right) - \lvert N_{neg} \rvert \sum_{v_k \sim P_n(v)} \log(\sigma(-Z_i^\intercal Z_k))$
LBSN2Vec [157]	Hyper graphs	Link prediction	$\sum_{i=1}^{n} \left(1 - \cos\left(Z_i, Z_j\right)\right)$
[158]	Hyper graphs	Link prediction	$\arg\min_{Z} Z^\intercal L Z$

Several types of graphs in the real world are heterogeneous, with different node and edge types. Most of the above models fail to capture heterogeneous graphs. Several models have been proposed to capture the heterogeneous graph structure [159,164,166]. Dong et al. [20] introduced the Metapath2Vec model, the idea based on random walks to learn node embeddings in heterogeneous graphs. One of the powers of meta-path is that it can capture the relationship between various types of nodes and edges in heterogeneous

graphs. To capture the structure of heterogeneous graphs with different types of nodes and edges, they presented meta-path random walks P with length l:

$$P : v_1 \xrightarrow{t_1} v_2 \xrightarrow{t_2} \cdots \xrightarrow{t_{k-1}} v_k \xrightarrow{t_k} \cdots \xrightarrow{t_{l-1}} v_l \tag{25}$$

where t_i presents the relation type between nodes v_i and v_{i+1}. Therefore, the transition probability of node v_{i+1} given by node v_i in the meta-path P could be defined as:

$$p(v_{i+1}|v_i^t, P) = \begin{cases} \frac{1}{|N_{t+1}(v_i^t)|} & \text{if } (v_{i+1}, v_i^t) \in E \; E^{(t)}(v_{i+1}) = t+1 \, . \\ 0, & \text{otherwise} \, . \end{cases}, \tag{26}$$

where $N_{t+1}(v_i^t)$ is the number of the neighbors of node v_i with node type $t+1$. Then, similar to DeepWalk and Node2Vec models, they used the skip-gram model to learn node embeddings. The approach from JUST [159] was conceptually similar to Metapath2Vec but the sampling strategy is performed differently. The model introduced a biased random-walk strategy with two parameters (jumping and staying) which aims to change the current domain or stay in the same domain for the next step.

Since the vanilla meta-path sampling strategy fails to capture different types of graphs, such as multiplex graphs and sparse graphs, several sampling strategies have been proposed for heterogeneous graphs based on meta-path strategies. The work of Zhang et al. [160] was similar to Metapath2Vec which implements random-walk sampling of all node types in the multiplex network. Lee et al. [161] introduced a BHIN2vec model which uses the random-walk strategy to capture sparse and rare patterns in heterogeneous graphs. Some models [162–165] have been applied to biological areas based on random-walk strategies. Lee et al. [166] used the WL relabeling strategy to capture temporal substructures of graphs. The model targeted the proximity of substructures in graphs instead of node proximity to learn the bibliographic entities in heterogeneous graphs. There are several models [167,168,176,177] that aim to capture entities from multiple networks. Du and Tong et al. [167] presented the MrMine model (Multi-resolution Multi-network) to learn embeddings with multi-resolutions. They first used WL label transformation to label nodes by the degree sequences, then adopted a dynamic time wrapping measure [21] to calculate the distance of each sequence to generate a relation network. The truncated random-walk sampling strategy is adopted to capture the graph structure. In contrast to the MrMine model, Lee and colleagues [168,176,177] explored in-depth multi-layered structure to represent the relation and proximity of individual characters, substructures, and the story network as a whole. To embed the substructure and story network, they first used WL relabeling [33] to extract substructures in the story network and then used Subgraph2Vec and Doc2Vec models to learn node embeddings.

Several types of graphs in the real world, however, show dynamic behaviors. Since most graph embedding models aim to learn node embeddings in static graphs, several models have been applied to learn node embeddings in dynamic graphs [10,92,173–175]. Most of them were based on the idea of DeepWalk and Node2Vec to capture the graph structure. By representing dynamic graphs as a set of static graphs, some models captured changes in the dynamic graph structure and updated changes in random walks over time. Then, the skip-gram model is used to learn node embeddings. For instance, the key idea of Sajjad et al. [169] is to generate random-walk paths on the first snapshot and then update random-walk paths in the corpus by time. Most existing models re-generate node embeddings for each graph snapshot to capture the dynamic behaviors. By contrast, the model introduced a set of dynamic random walks, which are frequently updated when there are any changes in dynamic graphs. This could reduce the computational complexity when the model handles large graphs. Figure 10 shows an example of how random-walk paths are updated in dynamic graphs.

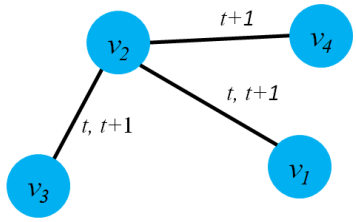


Figure 10. Updating random-walk paths to the corpus on dynamic graphs. At time t, the graph has 3 nodes: v_1, v_2, v_3 with two edges:(v_1, v_2) and (v_2, v_3). Assuming the length of the random walk is 3, then the set of random walks: $\langle v_1, v_2, v_1 \rangle$, $\langle v_1, v_2, v_3 \rangle$, $\langle v_2, v_1, v_2 \rangle$, $\langle v_2, v_3, v_2 \rangle$, $\langle v_3, v_2, v_1 \rangle$, $\langle v_3, v_2, v_3 \rangle$. At the time $t + 1$: the graph has a new node v_4 and a new edge (v_2, v_4). Then, new random walks will be updated on the corpus are: $\langle v_4, v_2, v_1 \rangle$, $\langle v_4, v_2, v_3 \rangle$, and $\langle v_4, v_2, v_4 \rangle$.

Since the evolution of graphs only takes place at every few nodes and within a specific range of neighbors, updating the entire random walk is time-consuming. Several models [169–172,174,175] suggested updating dynamic steps over time for a few nodes and their local neighbors' relationship. For example, Sedigheh et al. [174] presented the Dynnode2Vec model to capture the temporal evolution from graph G_t to G_{t+1} by a set of new nodes and edges (V_{new}, E_{new}) and a set of removed nodes and edges (V_{del}, E_{del}). Motivated by Node2Vec architecture, the Dynnode2Vec model could learn the dynamic structure by inducing an adequate group of random walks for only dynamic nodes. The random-walk strategy, therefore, could be more computational efficiency when the model handles large graphs. Furthermore, the proposed dynamic skip-gram model could learn node embeddings at time t by adopting the results of the previous time $t - 1$ as initial weights. As a result, the dynamic skip-gram model could learn the dynamic behaviors over time.

Therefore, the changes in nodes at time $t + 1$ could be described as:

$$\Delta V_t = V_{add} \cup \{v_i \in V_{t+1} | \exists e = (v_i, v_j) \in (E_{add} \cup E_{del})\} . \tag{27}$$

In summary, structure-preservation methods have succeeded in learning embeddings over the past decade. There are several key advantages of these models:

- Computational complexity: Unlike kernel models and matrix factorization-based models, which require considerable computational costs, structure preservation models could learn embeddings with an efficient time. This effectiveness comes from search-based sampling strategies and the model generalizability from the training process.
- Classification tasks: Since the models aim to find structural neighbor relationships from a target node, these show power in problems involving node classification. In almost all graphs, nodes that have the same label tend to be connected at a small, fixed-length distance. This is a strength of models based on preserving structure in problems related to classification tasks.

However, there are a few limitations that these models suffer when preserving the graph structure:

- Transductive learning: Most models cannot learn node embeddings that have not been seen in the training data. To learn new node embeddings, the model should re-sample the graph structure and learn the new samples again which could be time-consuming.
- Missing connection problem: Many graphs have sparse connections and missing connections between nodes in the real world. However, most structure-preservation models cannot handle missing connections between nodes since the sampling strategies could not be able to capture these connections. In the case of a random-walk-based sampling strategy, for example, these models only capture graph structure when nodes are linked together.

- Parameter sharing: These models could only learn node embeddings for individual nodes and do not share parameters. The absence of sharing parameters could reduce the effectiveness of learning representation.

3.3.2. Proximity Reconstruction Models

The purpose of graph embedding models is not only to preserve the graph structure but also to preserve the proximity of nodes in graphs. Most proximity reconstruction-based models are used for link prediction or node recommendation tasks [178–180] due to the nature of the similarity strategies. In this part, we discuss various models attempting to preserve the proximity of entities in graphs. Table 7 describes a summary of representative proximity reconstruction-based graph embedding models.

One of the typical models is LINE [16], which aims to preserve the symmetric proximity of node pairs in graphs. The advantage of the LINE model is that it could learn the node similarity which most structure-preservation models cannot represent this structural information. The main goal of the LINE model is to preserve the 1st-order and 2nd-order proximity of node pairs in graphs. The 1st-order proximity can be defined as follows:

Definition 7 (1st-order proximity [16]). *The 1st-order proximity describes the local pairwise similarity between two nodes in graphs. Let w_{ij} be the weight of an edge between two nodes v_i and v_j, and the 1st-order proximity is defined as w_{ij} when two nodes are connected and $w_{ij} = 0$ when there is no link between them.*

In the case of binary graphs, $w_{ij} = 1$ if two nodes v_i and v_j are connected, and $w_{ij} = 0$ otherwise. To preserve the 1st-order proximity, the objective function of two distribution $\hat{p}_1(v_i, v_j)$ and $p_1(v_i, v_j)$ should be minimized:

$$\mathfrak{L}_1(\theta) = \arg\min_{\theta} d\big(\hat{p}_1(v_i, v_j), p_1(v_i, v_j)|\theta\big) \tag{28}$$

$$\hat{p}_1(v_i, v_j) = \frac{w_{ij}}{\sum_{(v_k, v_l)\in E} Z_k^\intercal Z_l} \quad p_1(v_i, v_j) = \frac{\exp(Z_i^\intercal Z_j)}{\sum_{(v_k, v_l)\in E} Z_k^\intercal Z_l} \tag{29}$$

where $\hat{p}_1(v_i, v_j)$ and $p_1(v_i, v_j)$ depict the empirical probability, and the actual probability of the 1st-order proximity, respectively, v_i and v_j are two nodes in G, Z_i and Z_j are embedding vectors in latent space corresponding to v_i and v_j, respectively, $d(\cdot\,\cdot)$ is the distance between the two distributions. The statistical distance, Kullback–Leibler divergence [181], is usually used to measure the difference between two distributions. In addition to preserving the proximity of two nodes that are connected directly, the LINE model also introduced 2nd-order proximity, which could be defined as follows:

Definition 8 (2nd-order proximity [16]). *The 2nd-order proximity when $k = 2$ captures the relationship of neighbors of each pair of nodes in the graph G. The idea of the 2nd-order proximity is that nodes should be closed if they share the same neighbors.*

Let Z_i and Z_j are vector embeddings of nodes v_i and v_j, respectively, the probability of the specific context v_j given by the target node v_i could be defined as:

$$p_2(v_j|v_i) = \frac{\exp(Z_j^\intercal Z_i)}{\sum_{v_k\in V} \exp(Z_k^\intercal Z_i)}\,. \tag{30}$$

Therefore, the minimization of the objective function $\mathfrak{L}_2$ could be defined as:

$$\mathfrak{L}_2(\theta) = \arg\min_{\theta} \sum_{v_i\in V} D_{KL}(\hat{p}_2(.|v_i;\theta), p_2(.|v_i)) \tag{31}$$

where $\hat{p}_2(v_j|v_i) = \frac{w_{ij}}{\sum_{k\in N(i)} w_{ik}}$ is the observed distribution, w_{ij} is the weighted edge between v_i and v_j.

Table 7. A summary of proximity reconstruction models. $v_i^{(t)}$ denotes the type t of node v_i, w_{ij} is the weight between node v_i and v_j, P is a meta-path in heterogeneous graphs, N_2 is the 1st-order and 2nd-order proximity of a node v_i, and $P_n(v)$ is the noise distribution for negative sampling.

Models	Graph Types	Objective	Loss Function
LINE [16]	Static graphs	Node classification	$-\sum_{v_i\in V,\langle v_i,v_j\rangle\in E} \log\left(\sigma(Z_i^\intercal Z_j)\right) - \lvert N_{neg}\rvert \sum_{v_k\sim P_n(v)} \log(\sigma(-Z_i^\intercal Z_k))$
APP [76]	Static graphs	Link prediction	$-\sum_{v_i\in V,\langle v_i,v_j\rangle\in E} \log\left(\sigma(Z_i^\intercal Z_j)\right) - \lvert N_{neg}\rvert \sum_{v_k\sim P_n(v)} \log(\sigma(-Z_i^\intercal Z_k))$
PALE [77]	Static graphs	Link prediction	$-\sum_{v_i\in V,\langle v_i,v_j\rangle\in E} \log\left(\sigma(Z_i^\intercal Z_j)\right) - \lvert N_{neg}\rvert \sum_{v_k\sim P_n(v)} \log(\sigma(-Z_i^\intercal Z_k))$ $\sum_{\langle v_i,v_j\rangle\in E} \left\lVert Z_i - Z_j\right\rVert_F^2$
CVLP [182]	Attributed graphs	Link prediction	$-\sum_{\substack{(v_i,v_j)\in E\\(v_i,v_k)\notin E}} \log\sigma\left(Z_i^\intercal Z_j - Z_i^\intercal Z_k\right) + \alpha_1\left\lVert Z_i - Z_j\right\rVert_2^2 + \alpha_2 L_1 + \alpha_3 L_2$
[183]	Static graphs	Link prediction	$-\sum_{\langle v_i,v_j\rangle\in E} w_{ij}\log\left(p_1(v_i\|v_j)\right) - \sum_{\langle v_i,v_j\rangle\in E} w_{ij}\log\left(p_2(v_j\|v_i)\right)$
HARP [178]	Static graphs	Node classification	$-\sum_{v_i\in V,\langle v_i,v_j\rangle\in E} \log\left(\sigma(Z_i^\intercal Z_j)\right) - \lvert N_{neg}\rvert \sum_{v_k\sim P_n(v)} \log(\sigma(-Z_i^\intercal Z_k))$
PTE [179]	Heterogeneous graphs	Link prediction	$-\sum_{\left\langle v_i^{(t)},v_j^{(t)}\right\rangle\in E^{(t)}} w_{ij}\log(v_i^{(t)}\|v_j^{(t)})$
Hin2Vec [180]	Heterogeneous graphs	Node classification, link prediction	$\sum_{v_i\in V} -y_i^\intercal \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i)$
[78]	Heterogeneous graphs	Node classification	$\sum_{(v_i,v_j)\in E}\log\sigma(Z_iZ_j) + \lvert N_{neg}\rvert E_{v_k\sim P_n(v_k)}(\log\sigma(-Z_iZ_k))$
[184]	Signed graphs	Link prediction	$\sum_{v_i\in V} -y_i^\intercal \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i)$
[185]	Heterogeneous graphs	Node classification, Node clustering	$-\sum_{(v_i,v_j)\in P}\log\left(1+e^{-Z_iZ_j}\right) + \lvert N_{neg}\rvert E_{v_k\sim P_n(v_k)}\left[\log\left(1+e^{-Z_iZ_k}\right)\right]$
[186]	Heterogeneous graphs	Link prediction	$\sum_{(v_i,v_j)\in N_2}\log\sigma(Z_iZ_j) + \lvert N_{neg}\rvert E_{v_k\sim P_n(v_k)}(\log\sigma(-Z_iZ_k))$
[187]	Static graphs	Node classification	$\sum_{(v_i,v_j)\in N_2}\log\sigma(Z_iZ_j) + \lvert N_{neg}\rvert E_{v_k\sim P_n(v_k)}(\log\sigma(-Z_iZ_k))$
[188]	Heterogeneous graph	Graph reconstruction, link prediction, node classification	$\sum_{v_i\in V}\left\lVert(Z_i-\hat{Z}_i)\odot B\right\rVert_2^2 + \alpha L_2$
ProbWalk [189]	Static graphs	Node classification, link prediction	$\sum_{v_i\in V,\langle v_i,v_j\rangle\in E}\log\left(\sigma(Z_i^\intercal Z_j)\right) - \lvert N_{neg}\rvert\sum_{v_k\sim P_n(v)}\log(\sigma(-Z_i^\intercal Z_k))$
[190]	Static graphs	Node classification, link prediction	$\frac{1}{\lvert V\rvert}\sum_{v_i\in V}\left[y_i\log\hat{y}_i + \alpha(1-y_i)\log(1-\hat{y}_i)\right]$
NEWEE [191]	Static graphs	Node classification, link prediction	$-\sum_{v_i\in V,\langle v_i,v_j\rangle\in E} \log\left(\sigma(Z_i^\intercal Z_j)\right) - \lvert N_{neg}\rvert \sum_{v_k\sim P_n(v)} \log(\sigma(-Z_i^\intercal Z_k))$
DANE [192]	Attributed graphs	Node classification, Link prediction	$\sum_{v_i\in V}\left\lVert X_i-\hat{X}_i\right\rVert_2^2 + \sum_{v_i\in V}\left\lVert M_i-\hat{M}_i\right\rVert_2^2 - \sum_{(v_i,v_j)\in E}\log p_{ij} - \sum_{(v_i,v_j)\in E}\left[\log p_{ij} - \sum_{(v_i,v_j)\notin E}\log(1-p_{ij})\right]$
CENE [193]	Attributed graphs	Node classification	$\sum_{v_i\in V} -y_i^\intercal \log(\hat{y}_i) - (1-y_i)\log(1-\hat{y}_i)$
HSCA [194]	Attributed graphs	Node classification	$\lVert M - W^\intercal HX\rVert_F^2 + \alpha\left(\lVert W\rVert_F^2 + \lVert H\rVert_F^2\right)$

However, the LINE model had several limitations as it only handles symmetric proximity pairs of nodes, and the proximity of node pairs was only considered up to 2nd-order proximity. To deal with directed graphs, Chang et al. [76] introduced the APP model, which

could preserve the asymmetric proximity of node pairs. They introduced two roles for each node $v_i \in V$ as the source role s_{v_i} and target role t_{v_i}. The probability of each pair of nodes that start from a source node to the target node could be defined as:

$$p(v_i|v_j) = \frac{\exp(s_{v_j} \cdot t_{v_i})}{\sum_{v_k \in V} \exp(s_{v_j} \cdot t_{v_k})} . \tag{32}$$

Tong et al. [77] presented the PALE (Predicting Anchor Links via Embedding) model to predict the anchor links in social networks. The idea of the PALE model was the same as that of the LINE model, but they sampled only 1st-order proximity. The loss function with the negative sampling could be defined as:

$$\mathfrak{L}(V) = -\sum_{(v_i,v_j)\in E} \log \sigma(Z_i Z_j) - \left|N_{neg}\right| E_{v_k \sim P_n(v_k)} (\log \sigma(-Z_i Z_k)) . \tag{33}$$

Wei et al. [182] presented the CVLP (Cross-View Link Prediction) model that could predict the connections of nodes in the context of missing and noisy attributes. Given by a triplet (v_i, v_j, v_k) where $(v_i, v_j) \in E$ and $(v_i, v_k) \notin E$, the probability of proximity preservation is defined as:

$$P(s_{ij} > s_{ik}|U^g) = \sigma(s_{ij} - s_{ik}) \tag{34}$$

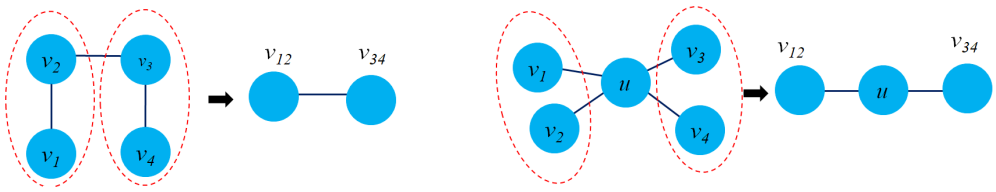
where U^g is the latent representation, s_{ij} is the inner product of the representation $s_{ij} = U_i^g (U_j^g)^\intercal$, and $\sigma(\cdot)$ is the sigmoid function. Li et al. [183] performed a similar study to deeply learn follower-ship and followee-ship between users across different social networks. The main idea of this model is that the proximity between nodes in a social network should be preserved in another social network. For each node v_i in a graph, there are three vector representations (a node vector Z_i, an input context vector $Z_i^{(1)}$, and output context vector $Z_i^{(2)}$). In particular, if a node v_i is following a node v_j in a social network, then vector Z_i should contribute to the input context of $Z_j^{(1)}$, and vector Z_j should contribute to the output context of $Z_i^{(2)}$. Therefore, given a node v_i, the input and output context probability of node v_j could be defined as follows:

$$p_{input}(v_j|v_i) = \frac{\exp({Z_j^{(1)}}^\intercal Z_i)}{\sum_{k=1}^{N} ({Z_k^{(1)}}^\intercal Z_i)} \quad p_{output}(v_i|v_j) = \frac{\exp({Z_j^{(2)}}^\intercal Z_j)}{\sum_{k=1}^{N} ({Z_k^{(2)}}^\intercal Z_j)} . \tag{35}$$

Haochen et al. [178] presented HARP (Hierarchical Representation) model with a meta-strategy to capture more global proximity of each pair node in graphs. The critical difference between HARP and LINE models is that they presented the original graph G as a series of graphs $G_1, G_2, \cdots, G_L$ where each graph can represent the collapse of adjacent edges and nodes. Figure 11 shows the way that two edges and nodes are collapsed in a graph. By representing L graphs after multiple collapses of edges and nodes, the graph can compress the proximity of nodes through supernodes.

Several variations and extensions of the LINE model are applied to heterogeneous and dynamic graphs. Jian et al. [179] presented the PTE model to preserve the 1st-order and 2nd-order proximity for heterogeneous graphs. By considering heterogeneous graphs as the set of bipartite graphs, they could independently construct the 1st-order and 2nd-order proximity for each homogeneous graph. Specifically, a bipartite graph G could be defined as $G = (V_A \cup V_B, E)$ where V_A and V_B are the set of nodes with different types. The probability of a node v_i in the set V_A given by a node v_j in the set V_B could be defined as follows:

$$p(v_i, v_j) = \frac{\exp(Z_i^\intercal \cdot Z_j)}{\sum_{v_k \in V_A} \exp(Z_k^\intercal \cdot Z_j)} . \tag{36}$$



(**a**) (**b**)

Figure 11. The strategy of edge and node collapsing of HARP model. (**a**) Edge compression; (**b**) Node compression. In (**a**), the super nodes $\langle v_1, v_2 \rangle$ and $\langle v_3, v_4 \rangle$ are formed by merging edges e_{12} and e_{34}, respectively. In (**b**), the super nodes $\langle v_1, v_2 \rangle$ and $\langle v_3, v_4 \rangle$ are formed by merging node pairs (v_1, v_2) and (v_3, v_4), respectively.

The PTE model decomposes heterogeneous graphs into k homogeneous graphs, and the loss function is the sum of the component loss functions, which could be formulated as:

$$\mathfrak{L}(V) = -\sum_{\left(v_i^{(t)}, v_j^{(t)}\right) \in E^{(t)}} w_{ij} \log p\left(v_i^{(t)} | v_j^{(t)}\right), \tag{37}$$

where K is the number of bipartite graphs extracted from the heterogeneous graphs. Similar to the PTE model, Tao-yang et al. [180] proposed the Hin2Vec model to capture the 2nd-order proximity in heterogeneous graphs. However, instead of treating heterogeneous graphs as sets of bipartite graphs, the Hin2Vec model captured the relationship between two nodes within 2-hop distance. For instance, in the DBLP network, the relationship set is $R = \{P-P, P-A, A-P, P-P-P, P-P-A, P-A-P, A-P-P, A-P-A\}$ where P is the paper node type and A is the author node type. Zhipeng and Nikos [185] presented the HINE model (Heterogeneous Information Network Embedding) to preserve the truncated proximity of nodes. They defined an empirical joint probability of two entities in a graph as:

$$\hat{p}(v_i, v_j) = \frac{s(v_i, v_j)}{\sum_{v_k \in V} s(v_i, v_k)} \tag{38}$$

where v_i and v_j are nodes, and $s(v_i, v_j)$ depicts the proximity between v_i and v_j in G. The proximity score $s(v_i, v_j)$ could be measured by counting the number of instances of the meta-path containing two nodes or a probability gained from implementing a random-walk sampling from v_i to v_j.

Graphs in the real world, however, could contain attributes where several existing models, such as LINE and APP, fail to capture this information. Several models have been proposed to learn structural similarity in attributed graphs [193,195]. Sun et al. [193] proposed a CENE (content-enhanced network embedding) model to learn structural graphs and side information jointly. The objective of the CENE model is to preserve the similarity between node pairs and node-content pairs. Zhang et al. [194] proposed the HSCA (Homophily, Structure, and Content Augmented network) model to learn the homophily property of node sequences. To gain the node sequences, HSCA uses the DeepWalk model to capture the short random-walk sampling, which could represent the node context. The model then learns node embeddings based on matrix factorization by decomposing the probability transition matrix.

Most models mentioned above mainly consider the edge's existence and ignore the dissimilarities between edges. Beyond preserving the topology and proximity of the aforementioned nodes, there are a variety of studies on edge reconstruction. The main idea of edge initialization-based models is that the edge weights can be transformed as transition probability. Wu et al. [189] introduced the ProbWalk model to learn weighted edges based on random-walk paths for edges and the skip-gram model to learn edge embeddings. The advantage of random walk on weighted edges is that this could help the model to generate

more accurate node sequences and capture more useful structural information. To calculate the probability of weighted edges in graphs, they introduced a joint distribution:

$$p(v_1, v_2 \cdots v_k | v_i) = \prod_{v_j \in C} \frac{e^{Z_j \cdot Z_i}}{\sum_{m=1}^{n} e^{Z_m \cdot Z_i}} \tag{39}$$

where v_i is the target node, $C = \{v_1, v_2, \cdots, v_k\}$ is the context of node v_i, and Z_i is vector embedding of node v_i.

Alternatively, several tasks need to preserve the proximity between different relationship types of nodes. Qi et al. [190,191] proposed the NEWEE model to learn edge embeddings and then adopted a biased random-walk sampling to capture the graph structure. To learn edge embeddings, they first look for a self-centered network of each graph node. In this situation, the model could explore the similarity between edges in the self-centered network since their score tends to be higher than those in the different self-centered networks. Given a node v_i in G, the self-centered network is a set of nodes containing v_i and its neighbors. For example, Figure 12 depicts two self-centered networks C_1 and C_2 of node v_1. The objective of the model is to make all edges embedded in the same self-centered network should be close in the vector space. Therefore, given a self-centered network $G' = (V', E')$, the objective function aims to maximize the proximity between edges in the same network, which could be defined as:

$$\mathfrak{L}(E) = -\sum_{v_i \in V'} \sum_{\substack{e_{ij} \in E' \\ e_{ik} \notin E'}} \log\left[\sigma\left(e_{ij}^{\intercal} Z_i\right)\right] + \log\left[1 - \sigma\left(e_{ij}^{\intercal} Z_k\right)\right] \tag{40}$$

where e_{ij} denotes the edge between node v_i and v_j in a self-centered network G', e_{ik} denotes a negative edge that v_i and v_k coming from different self-centered network.

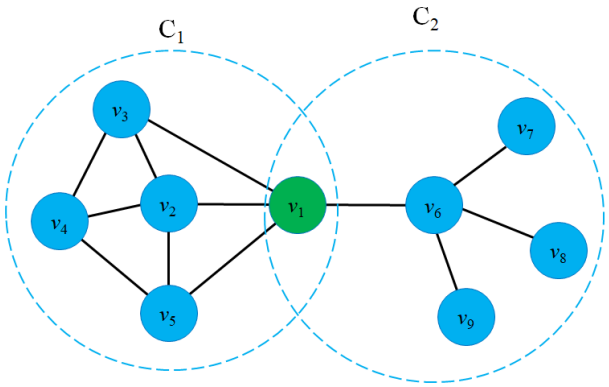


Figure 12. The self-centered network of NEWEE model. For instance, the self-centered of node v_2 could be defined as $G' = (V', E')$ where $V' = (v_1, v_2, v_3, v_4, v_5)$ and E' is the set of edges in G'.

In summary, compared with structure-preservation models, the proximity construction models bring several advantages:

- Inter-graph proximity: Proximity-based models not only explore proximity between nodes in a single graph but can also are applied for proximity reconstruction across different graphs with common nodes [183]. These methods can preserve the structural similarity of nodes in other graphs which are entirely different from other models. In the case of models based on structure-preservation strategies, these must re-learn node embeddings in other graphs.
- Proximity of nodes belonging to different clusters: In the context of clusters with different densities and sizes, proximity reconstruction-based models could capture nodes that are close to each other but in different clusters. This feature shows an

advantage over structure reconstruction-based models, which tend to favor searching for neighboring nodes in the same cluster.

- Link prediction and node classification problem: Since structural identity is based on proximity between nodes, two nodes with similar neighborhoods should be close in the vector space. For instance, the LINE model considered preserving the 1st-order and 2nd-order proximity between two nodes. As a result, proximity reconstruction provides remarkable results for link prediction and node classification tasks [16,76,77].

However, besides the advantages of these models, there are also a few disadvantages of the proximity-based models:

- Weighted edges problems: Most proximity-based models do not consider the weighted edges between nodes. These models consider proximity based only on the number of connections shared without weights which could lead to structural loss.
- Capturing the whole graph structure: Proximity-based models mostly focus on 1st-order and 2nd-order proximity which cannot specify the global structure of graphs. A few models try to capture the higher-order proximity of nodes in graphs, but there is a problem with the computational complexity.

To overcome these limitations, shallow models should be replaced by models based on deep neural networks. Deep neural network-based models can better generalize and capture more of graph entity relationships and graph structure.

3.4. Deep Neural Network-Based Models

In recent years, large-scale graphs have challenged the ability of numerous graph embedding models. Traditional models, such as shallow neural networks or statistical methods, cannot efficiently capture complex graph structures due to their simple architecture. Recently, there have been various studies on deep graph neural networks, which are exploding rapidly because of their ability to work with complex and large graphs [11,14,23,196]. Based on the model architecture, we separate deep graph neural networks into four main groups: graph autoencoders, recurrent GNNs, convolutional GNNs, and graph transformer models. This section provides a detailed picture of deep neural network-based methods.

Unlike earlier models, most deep neural network-based models adopt the graph structure (represented as A) and node attributes/features (represented as X) to learn node embeddings. For instance, users in the social network could have text data, such as profile information. For nodes with missing attribute information, the attributes/features could be represented as node degree or one-hot vectors [72].

3.4.1. Graph Autoencoders

Graph autoencoder models are unsupervised learning algorithms that aim to encode graph entities into the latent space and reconstruct these entities from the encoded information. Based on the encoder and decoder architecture, we can classify graph autoencoder models into multilayer perceptron-based models and recurrent graph neural networks.

Early-stage graph autoencoder models are primarily based on multilayer perceptron (MLP) to learn embeddings [50,51,196]. Table 8 lists a summary of fully connected graph autoencoder models. Daixin et al. [50] introduced the SDNE model (Structural Deep Network Embedding) to capture the graph structure based on autoencoder architecture. Similar to the LINE model, the SDNE model aimed to preserve the 1st-order and 2nd-order proximity between two nodes in graphs, but it used the autoencoder-based architecture. Figure 13 presents the general architecture of the SDNE model with the corresponding encoder and decoder layers. The joint loss function that combines two loss functions for 1st-order proximity and 2nd-order proximity can be formulated as:

$$\mathfrak{L}(Z, X) = \sum_{i,j=1}^{n} \left\| (\hat{X} - X) \odot B \right\|_F^2 + \lambda \sum_{i,j=1}^{n} s_{ij} \left\| Z_i - Z_j \right\|_2^2 + L_2 \tag{41}$$

where s_{ij} denotes the proximity between two nodes v_i and v_j. However, the SDNE model has been proposed to learn node embeddings in homogeneous graphs. Extension of the SDNE model to heterogeneous graphs was suggested by several graph autoencoder models [51,196]. Ke et al. [51] presented the DHNE model (Deep Hyper-Network Embedding) to preserve neighborhood structures, ensuring that the nodes with similar neighborhood structures will have similar embeddings. The autoencoder layer adopts an adjacency matrix A of a hypergraph as an input, which can be formulated as:

$$A = HH^{\intercal} - D_v \tag{42}$$

where D_v is the diagonal matrix of node degree, and H is a matrix of size $|V| \times |E|$ presents the relation between nodes and hyperedges in graphs. The autoencoder includes two main layers: an encoder layer and a decoder layer. The encoder part takes the adjacency matrix as input and compresses it to generate node embeddings, and then the decoder part tries to reconstruct the input. Formally, the output of the encoder and decoder layer of node v_i could be defined as follows:

$$Z_i = \sigma(WA_i + b) \quad \hat{A}_i = \sigma(\hat{W}Z_i + \hat{b}) \, . \tag{43}$$

Table 8. A summary of fully connected graph autoencoder models. A and $\hat{A}$ are the input adjacency matrix and reconstructed adjacency matrix, respectively, B is the penalty matrix, A^t is the adjacency matrix of node type t, L denotes the number of layers, k is the length of random-walk steps, s_{ij} denotes the proximity between v_i and v_j, and $Z_i^{(l)}$ is the hidden vector of node v_i at layer l.

Models	Graph Types	Objective	Loss Function
SDNE [50]	Static graphs	1st-order proximity, 2nd-order proximity	$\left\|(\hat{A} - A) \odot B\right\|_F^2 + \lambda \sum_{i,j=1}^{n} s_{ij} \left\|Z_i - Z_j\right\|_2^2 + L_2.$
DHNE [51]	Hyper graphs	1st-order proximity, 2nd-order proximity	$\left\|\hat{A} - A\right\|_F^2 + \lambda \sum_{t}^{n} \left\|A^t - \hat{A}^t\right\|_F^2.$
DNE-SBP [197]	Signed graphs	1st-order proximity	$\sum_{l=1}^{L} \left(\left\|(\hat{A}^{(l)} - A^{(l)}) \odot B\right\|_F^2 + \alpha_l A \left\|Z_i^{(l)} - Z_j^{(l)}\right\|_F^2 + \beta_l L_1 + \gamma_l L_2 \right).$
DynGEM [198]	Dynamic graphs	1st-order proximity, 2nd-order proximity	$\left\|(\hat{A} - A) \odot B\right\|_F^2 + \lambda \sum_{i,j=1}^{n} s_{ij} \left\|Z_i - Z_j\right\|_2^2 + L_1 + L_2.$
NetWalk [199]	Dynamic graphs	Random walk	$\sum_{l=1}^{L} \left\|Z_i^{(l)} - Z_j^{(l)}\right\|_2^2 + \sum_{l=1}^{L} \left\||A_i^{(l)} - A_j^{(l)}\right\|_2^2 + L_2.$
DNGR [196]	Static graphs	PPMI matrix	$\left\|\hat{A} - A\right\|_F^2.$

One of the limitations of SDNE and DHNE models is that these models cannot handle signed graphs. Shen and Chung [197] proposed the DNE-SBP model (Deep Network Embedding with Structural Balance Preservation) to preserve the proximity of nodes in signed graphs. The DNE-SBP model constructed the input and output of the autoencoder which could be defined as:

$$H^{(l)} = \sigma\left(X^{(l)}\left(W_1^{(l)}\right)^T + B_1^{(l)}\right) \quad \hat{X}^{(l)} = \sigma\left(\hat{H}^{(l)}(W_2^{(l)})^T\right) + B_2^{(l)} \tag{44}$$

where $X^{(1)} = A$, $X^{(l)} = H^{(l-1)}$, and σ is an activation function. The joint loss function is then composed of reconstruction errors with ML and CL pairwise constraints [200].

For dynamic graphs, graph autoencoder models take snapshots of graphs as inputs, and the model tries to rebuild snapshots. In several models, the output can predict future graphs by reconstructing coming snapshot graphs. Inspired by the SDNE model for static graphs, Palash et al. [198] presented the DynGEM model for dynamic graph embedding. Figure 14 presents the overview architecture of the DynGEM model. Given a

sequence of graph snapshots $G = \{G_1, G_2, \cdots, G_T\}$ and a sequence of a mapping function $\phi = \{\phi_1, \phi_2, \cdots, \phi_T\}$, the DynGEM model aims to generate an embedding $Z_{t+1} = \phi_{t+1}(G_{t+1})$. The stability of embeddings is the ratio of the difference between embeddings over the difference between adjacency matrices over time which could be defined as:

$$A_{abs}(\phi; t) = \left\| \frac{Z_{t+1}(V_t) - Z_t(V_t)}{A_{t+1}(V_t) - A_t(V_t)} \right\| \tag{45}$$

where A_t is the weighted adjacency matrix of graph G_t, $Z_t(V_t)$ presents embeddings of all nodes V_t at time t. The model learns parameter θ for each graph snapshot G_t at time t. Similar to the SDNE model, the loss function of the DynGEM model could be defined as:

$$\mathfrak{L}(Z, X) = \left\| (\hat{A} - A) \odot B \right\|_F^2 + \lambda \sum_{i,j=1}^{n} s_{ij} \left\| Z_i - Z_j \right\|_2^2 + L_1 + L_2 \tag{46}$$

where L_1 and L_2 are regularization terms to prevent the over-fitting, and s_{ij} is the similarity between v_i and v_j. Similar to SDNE, Palash et al. [201] used autoencoder architecture and adopted the adjacency matrix of graph snapshots as input of the encoder layer. However, they updated parameters θ_t at time t based on parameter θ_{t-1} from the previous graph G_{t-1}.

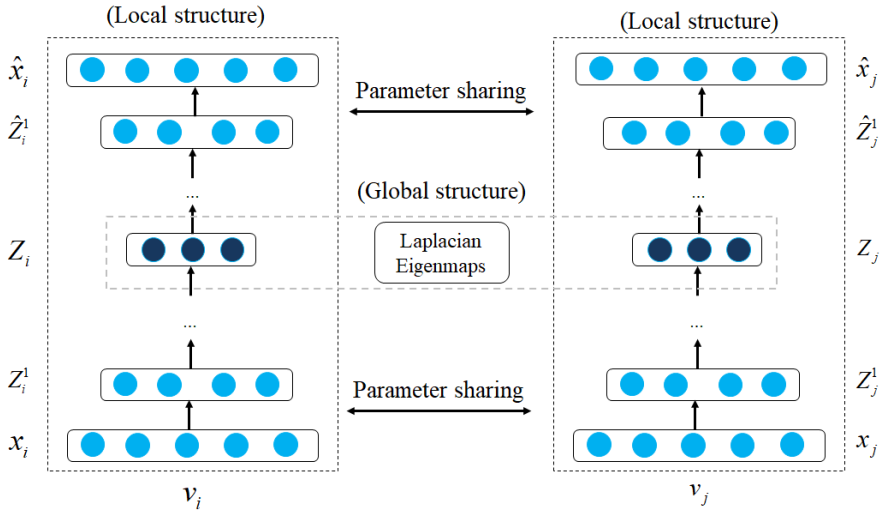


Figure 13. The architecture of SDNE model. The features of nodes x_i and x_j are the inputs of the SDNE model. The encoder layer compresses the feature data x_i and x_j into vectors Z_i and Z_j in the latent space. The decoder layer aims to reconstruct the node features.

Unlike the aforementioned models, Wenchao et al. [199] presented the NetWalk model that composes initial embeddings first and then updates the embeddings by learning paths in graphs, which are sampled by a reservoir sampling strategy. NetWalk model sampled the graph structure using a random-walk strategy as input to the autoencoder model. If there are any changes in dynamic graphs, the Netwalk model first updates the list of neighbors for each node and corresponding edges and then only learns embeddings again for the changes.

The aforementioned autoencoder models, which are based on feedforward neural networks, only focus on preserving pairs of nodes in graphs. Several models focus on integrating recurrent neural networks and LSTM into the autoencoder architecture, bringing prominent results, which we cover in the following section.

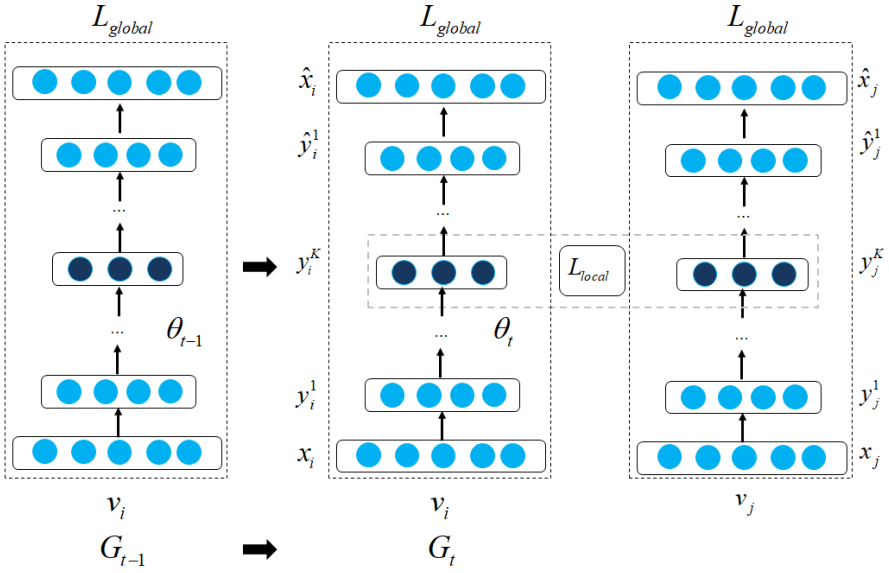


Figure 14. The architecture of DynGEM model. Similarity to the SDNE model, the DynGEM model could capture the 1st-order and 2nd-order proximity between two nodes in graphs with the encoder and decoder layers. The difference is vector embedding θ_t parameters at time t are updated from vector embedding θ_{t-1} at time $t-1$.

3.4.2. Recurrent Graph Neural Networks

One of the first models applying deep neural networks to graph representation learning was based on graph neural networks (GNNs). The main idea of GNNs is that it considers messages shared between target nodes and their neighbors until a steady balance is acquired. Table 9 summarizes graph recurrent autoencoder models.

Scarselli et al. [44,45] proposed a GNN model which could learn embeddings directly for different graphs, such as acyclic/cyclic and directed/undirected graphs. These models assumed that if nodes are directly connected in graphs, the distance between them should be minimized in the latent space. The GNN models used a data diffusion mechanism to aggregate signals from neighbor nodes (units) to target nodes. Therefore, the state of a node describes the context of its neighbors and can be used to learn embeddings. Mathematically, given a node v_i in a graph, the state of v_i and its output can be defined as:

$$H_i = \sum_{v_j \in N(v_i)} f_w(y_i, e_{ij}, H_j, y_j), \quad Z_i = g_w(H_i, y_i), \tag{47}$$

where $f_w(\cdot\,\cdot\,\cdot)$ and $g_w(\cdot\,\cdot)$ are transition functions, and y_i, e_{ij} denote the label of node v_i, edge (v_i, v_j), respectively. By considering the state H_i that is revised by the shift process, H_i and its output at layer l could be defined as:

$$H_i^{(l)} = f_w\left(y_i, e_{ij}, H_j^{(l-1)}, y_j\right), \quad Z_i^{(l)} = g_w\left(H_i^{(l)}, y_i\right). \tag{48}$$

However, one of the limitations of GNNs is that the model learns node embeddings as single output, which could cause problems with sequence output. Several studies tried to improve GNNs using recurrent graph neural networks [17,48,49]. Unlike the GNNs which could represent a single output for each entity in a graph, Li et al. [17] attempted to output sequences by applying gated recurrent units. The model used two gated graph neural networks $F_x^{(l)}$ and $F_o^{(l)}$ to predict the output O^l and the following hidden states. Therefore, the output of node v_i at layer $l+1$ could be computed as:

$$H_i^{(l+1)} = \sigma\left(H_i^{(l)}, \sum_{v_j \in N(v_i)} WH_j^{(l)}\right), \tag{49}$$

where $N(v_i)$ denotes the set of neighbors of node v_i.

Wang et al. [49] proposed Topo-LSTM model to capture the diffusion structure by representing graphs as a diffusion cascade to capture active and inactive nodes in graphs. Given by a cascade sequence $s = \{(v_1, 1) \cdots (v_T, T)\}$, the hidden state can be represented as follows:

$$h_t^{'(p)} = \phi(h_v | v \in P_v), \tag{50}$$

$$h_t^{'(q)} = \phi(h_v | v \in Q_v \backslash P_v), \tag{51}$$

where p and q denote the input aggregation for active nodes connected with v_t and not connected with the node v_t, respectively, P_v depicts the precedent sets of active nodes at time t, and Q_v depicts the set of activated nodes before time t. Figure 15 presents an example of the Topo-LSTM model. However, these models could not capture global graph structure since they only capture the graph structure within k-hop distance. Several models have been proposed by combining graph recurrent neural network architecture with random-walk sampling structure to capture higher structural information [48,93]. Huang et al. [93] introduced the GraphRNA model to combine a joint random-walk strategy on attributed graphs with recurrent graph networks. One of the powers of the random-walk sampling strategy is to capture the global structure. By considering the node attributes as a bipartite network, the model could perform joint random walks on the bipartite matrix containing attributes to capture the global structure of graphs. After sampling the node attributes and graph structure through joint random walks, the model uses graph recurrent neural networks to learn embeddings. Similar to GraphRNA model, Zhang et al. [48] presented the SHNE model to analyze the attributes' semantics and global structure in attributed graphs. The SHNE model also used a random-walk strategy to capture the global structure of graphs. However, the main difference between SHNE and GraphRNA is that the SHNE model first applied GRU (gated recurrent units) model to learn the attributes and then combined them with graph structure via random-walk sampling.

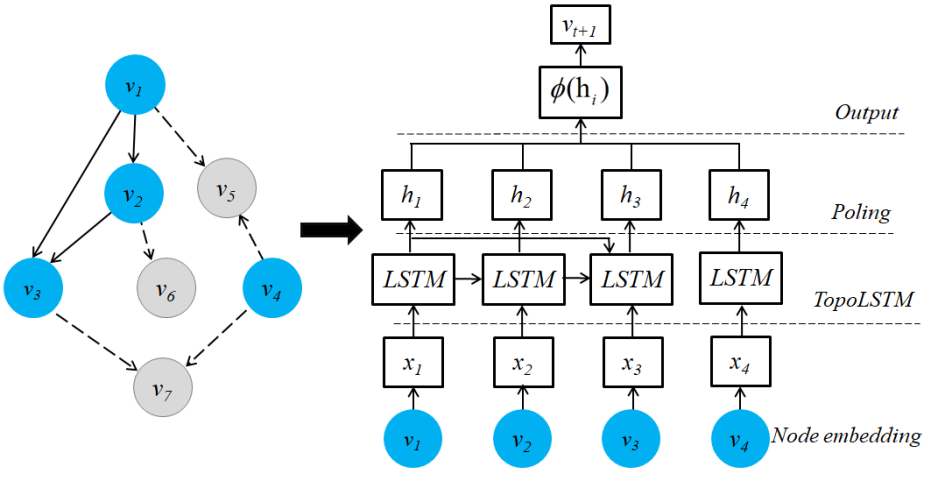


Figure 15. An example of the Topo-LSTM model. Given by a cascade sequence $S = \{(v_1, 1), (v_2, 2), (v_3, 3), (v_4, 4)\}$, the model first takes features of each node x_1, x_2, x_3, x_4 as inputs and then infers embeddings via Topo-LSTM model.

Table 9. A summary of graph recurrent autoencoder models. $G_{i,t}$ is the diffusion graph of a cascade at time t, y_i is the label of node v_i, T is the timestamp window, A^t_{ij} is the adjacency matrix at time t, $\sigma(\cdot)$ is the sigmoid function. $w_{i,j}$ is the weight between two nodes v_i and v_j, $N_s(v_i)$ is the set of neighbors of node v_i, and triple (v_i, v_j, v_k) denotes $(v_i, v_j) \in P$, and v_k is the negative sample.

Model	Graph Type	Sampling Strategy	Loss Function
[44]	Hypergraphs	Local transition function	$\sum_{v_i \in V} \left\| Z_i - \hat{Z}_i \right\|_2^2$
[45]	Homogeneous graphs	Local transition function	$\sum_{v_i \in V} \left\| Z_i - \hat{Z}_i \right\|_2^2$
[57]	Weighted graphs	Node-weight sequences	$\frac{1}{N} \sum_{v_i \in V} \left\| Z_i - \hat{Z}_i \right\|_2^2$, $\sum_{<v_i, v_j> \in E} w_{ij} \log\Big(p(v_i, v_j)\Big)$ $p(v_i, v_j) = \text{SoftMax}(Z_i^\intercal Z_j)$
[202]	Dynamic graphs	Random walk, Shortest paths, BFS	$\frac{1}{N} \sum_{v_i \in V} \left\| Z_i - \hat{Z}_i \right\|_2^2$
LSTM-Node2Vec [203]	Dynamic graphs	Temporal random walk	$-\sum_{v_i \in V} \log p(N_s(v_i) \mid Z_i)$
E-LSTM-D [204]	Dynamic graphs	1st-order proximity	$\left\| (A_t - \hat{A}_t) \odot B \right\|_F^2 + \lambda L_2$
Dyngraph2Vec-AERNN [201]	Dynamic graphs	Adjacency matrix	$\left\| (A_t - \hat{A}_t) \odot B \right\|_F^2$
Topo-LSTM [49]	Directed graphs	Diffusion structure	$-\sum_{v_i \in V} \sum_{t=1}^{T} \log p\left(v_{i,t} \mid G_t\right) + \alpha L_2$
SHNE [48]	Heterogeneous graphs	Random walk Meta-path	$\sum_{\langle v_i, v_j, v_k \rangle} \log \sigma\Big(Z_j \cdot Z_i\Big) + \log \sigma(-Z_k \cdot Z_i)$
[17]	Directed graphs	Transition matrix	$-\sum_{v_i \in V} y_i^\intercal \log(\hat{y}_i)$
GraphRNA [93]	Attributed graph	Random walk	$-\sum_{v_i \in V} y_i^\intercal \log(\hat{y}_i)$
[205]	Labeled graphs	Random walks, shortest paths, and breadth-first search.	$\sum_{v_i \in V} \left\| Z_i - \hat{Z}_i \right\|_2^2$ $-\sum_{v_i \in V} y_i^\intercal \log(\hat{y}_i)$
[206]	Dynamic graphs	Graph reconstruction	$\sum_{t=1}^{T} \sum_{<v_i, v_j> \in E} A^t_{i,j} \log(\hat{A}^t_{i,j})$
Camel [207]	Heterogeneous graphs	Link prediction	$\sum_{\substack{(v_i, v_j) \in E \\ (v_i, v_k) \notin E}} \left[\left\| Z_i - Z_j \right\|_2^2 - \left\| Z_i - Z_k \right\|_2^2 \right]$ $+ \alpha_1 \sum_{(v_i, v_k) \notin P} \log \sigma\Big(-Z_i^\intercal Z_j\Big) + \alpha_2 L_2$
TaPEm [208]	Heterogeneous graphs	Link prediction	$-\sum_{v_i \in V} \left(y_i^\intercal \log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i) \right)$
[209]	Heterogeneous graphs	Link prediction	$-\sum_{v_i \in V} \left(y_i^\intercal \log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i) \right)$

Since the power of autoencoders architecture is to learn compressed representations, several studies [57,205] aimed to combine RGNNs and autoencoders with learning node embeddings in weighted graphs. For instance, Seo and Lee [57] adopted an LSTM autoencoder to learn node embeddings for weighted graphs. They used the BFS algorithm to travel nodes in graphs and extract the node-weight sequences of graphs as inputs for the LSTM autoencoder. The model then could leverage the graph structure reconstruction based on autoencoder architecture and the node attributes by the LSTM model. Figure 16 presents the sampling strategy of this model, which lists the nodes and their respective weighted edges. To capture local and global graph structure, Aynaz et al. [205] proposed a sequence-to-sequence autoencoder model, which could represent inputs with arbitrary lengths. The LSTM-based autoencoder model architecture consists of two main parts: the encoder layer LSTM_{enc} and the decoder layer LSTM_{dec}. For the sequence-to-sequence autoencoder, at each time step l, the hidden vectors in the encoder and decoder layers can be defined as:

$$h_{enc}^{t} = \text{LSTM}_{enc}\left(Z_i^{(t)}, h_{enc}^{t-1}\right), \quad h_{dec}^{t} = \text{LSTM}_{dec}\left(Z_i^{(t-1)}, h_{dec}^{t-1}\right) \tag{52}$$

where h_{enc}^{t} and h_{dec}^{t} are the hidden states at step t in the encoder and decoder layers, respectively. To generate the sequences of nodes, the model implemented different sampling strategies, including random walks, shortest paths, and breadth-first search with the WL algorithm to encode the information of node labels.

Since the aforementioned models learn node embeddings for static graphs, Shima et al. [203] presented an LSTM-Node2Vec model by combining an LSTM-based autoencoder architecture with the Node2Vec model with learning embeddings for dynamic graphs. The idea of the LSTM-Node2Vec model is that it uses an LSTM autoencoder to preserve the history of node evolution with a temporal random-walk sampling. It then adopted the Node2Vec model to generate the vector embeddings for the new graphs. Figure 17 presents a temporal random-walk sampling strategy to travel a dynamic graph.

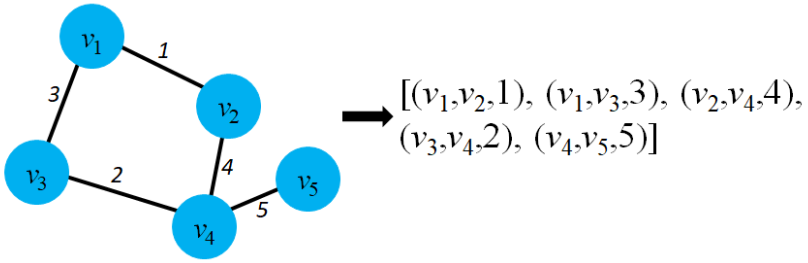
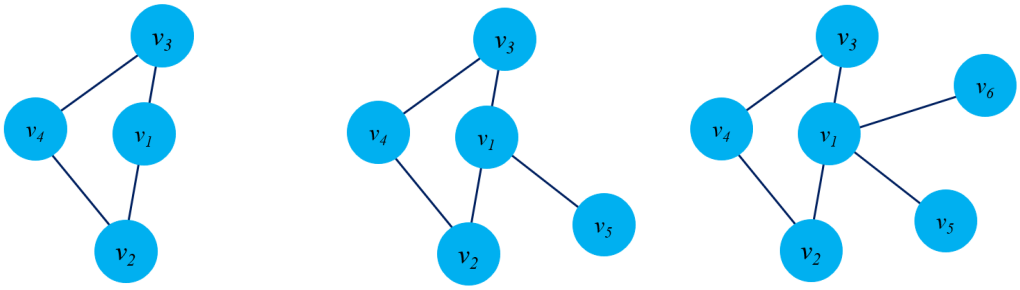


Figure 16. The sampling strategy of [57]. The model lists all the node pairs in respective weights as input of the autoencoder model.



(**a**) (**b**) (**c**)

Figure 17. The temporal random-walk sampling strategy of LSTM-Node2Vec model during the graphs' evolution. (**a**) t; (**b**) $t+1$; (**c**) $t+2$. At the time t, the graph has four nodes and four edges between nodes. At the time $t+1$ and $t+2$, the graph has new nodes v_5 and v_6, respectively. A temporal random walk for node v_1 with length $L = 3$ could be: $P = \{(v_2, v_3, v_4), (v_3, v_2, v_5), (v_3, v_5, v_6), \cdots\}$.

Jinyin et al. [204] presented the E-LSTM-D model (Encoder-LSTM-Decoder) to learn embeddings for dynamic graphs by combining autoencoder architecture and LSTM layers. Given by a set of graph snapshots $S = \{G_{t-k}, G_{t-k+1}, \cdots, G_{t-1}\}$, the objective of the model is to learn a mapping function $\phi : \phi(S) \rightarrow G_t$. The model takes the adjacency matrix as the input of the autoencoder model, and the output of the encoder layer could be defined as:

$$H_{e,i}^{(1)} = \text{ReLU}\left(W_e^{(1)} s_i + b_i^{(1)}\right) \tag{53}$$

$$H_{e,i}^{(l)} = \text{ReLU}\left(W_e^{(l)} b_{e,i}^{(l-1)} + b_e^{(l)}\right) \tag{54}$$

$$H_e^{(l)} = \left[H_{e,0}^{(l)}, H_{e,1}^{(l)}, \cdots, H_{e,N-1}^{(l)}\right] \tag{55}$$

where s_i denotes the i-th graph in the series of graph snapshots, $\text{ReLU}(\cdot) = max(0, \cdot)$ is the activation function. For the decoder layer, the model tried to reconstruct the original adjacency matrix from vector embeddings, which could be defined as follows:

$$H_d^{(1)} = \text{ReLU}\left(W_d^{(1)} H_e + b_d^{(1)}\right) \quad (56)$$

$$H_d^{(l)} = \text{ReLU}\left(W_d^{(l)} H_d^{(l-1)} + b_d^{(l)}\right) \quad (57)$$

where H_e depicts the output of the stacked LSTM model, which captures the current graph's structure G_t. Similar to E-LSTM-D model, Palash et al. [201] proposed a variant of Dyngraph2Vec model, named Dyngraph2VecAERNN (Dynamic Graph to Vector Autoencoder Recurrent Neural Network) which also considers the adjacency matrix as input for the model. However, the critical difference between the E-LSTM-D model and the Dyngraph2VecAERNN model is that they feed the LSTM layers directly into the encoder part to learn embeddings. The decoder layer is composed of fully connected neural network layers to reconstruct the inputs.

There are several advantages of recurrent graph neural networks compared to shallow learning techniques:

- Diffusion pattern and multiple relations: RGNNs show superior learning ability when dealing with diffuse information, and they can handle multi-relational graphs where a single node has many relations. This feature is achieved due to the ability to update the states of each node in each hidden layer.
- Parameter sharing: RGNNs could share parameters across different locations, which could be able to capture the sequence node inputs. This advantage could reduce computational complexity during the training process with fewer parameters and increase the performance of the models.

However, one of the disadvantages of the RGNNs is that these models use recurrent layers with the same weights during the weight update process. This leads to inefficiencies in representing different relationship constraints between neighbor and target nodes. To overcome the limitation of RGNNs, convolutional GNNs have shown remarkable ability in recent years when it uses different weights in each hidden layer.

3.4.3. Convolutional Graph Neural Networks

CNNs have achieved remarkable success in the image processing area. Since image data can be considered to be a special case of graph data, convolution operators can be defined and applied to graph mining. There are two strategies to implement when applying convolution operators to the graph domain. The first strategy is based on graph spectrum theory which transforms graph entities from the spatial domain to the spectral domain and applies convolution filters on the spectral domain. The other strategy directly employs the convolution operators in the graph domain (spatial domain). Table 10 summarizes spectral CGNN models.

Table 10. A summary of spectral CGNN models.

Model	Graph Type	Tasks	Loss Function
[56]	Static graphs	Node classification	$-\sum_{v_i \in V} y_i^\intercal \log(\hat{y}_i)$
[96]	Static graphs	Node classification	$-\sum_{v_i \in V} y_i^\intercal \log(\hat{y}_i) + L_2$
[210]	Static graphs	Multi-task prediction Node classification	$\left[\frac{1}{\lvert V \rvert} \sum_{v_i \in V} (\hat{y}_i - y_i)^2\right]^{\frac{1}{2}}$
[211]	Static graphs	Label classification	$-\frac{1}{\lvert V \rvert} \sum_{v_i \in V} y_i^\intercal \log(\hat{y}_i) + L_2$
GCN [18]	Knowledge graphs	Node classification	$-\sum_{v_i \in V} y_i^\intercal \log(\hat{y}_i)$

Table 10. *Cont.*

Model	Graph Type	Tasks	Loss Function
EGCN [55]	Static graphs	Multi-task classification, Link prediction	$\left[\frac{1}{\lvert V\rvert}\sum_{v_i\in V}(\hat{y}_i - y_i)^2\right]^{\frac{1}{2}}$
LNPP [212]	Static graphs	Graph Reconstruction	$\left\lVert A - \hat{A}\right\rVert_F^2$
[213]	Static graphs	Node classification	$\sum_{v_i\in V}\lVert y_i - \hat{y}_i\rVert_D^2)$
[214]	Static graphs	Node classification	$-\sum_{v_i\in V} y_i^{\intercal}\log(\hat{y}_i)$
[215]	Heterogeneous graphs	Node classification	$\sum_{(v_i,v_j)\in E}\log\sigma\left(Z_i^{\intercal}Z_j\right) - \sum_{(v_i,v_k)\notin E}\log\sigma\left(-Z_i^{\intercal}Z_j\right)$

When computing power is insufficient for implementing convolution operators directly on the graph domain, several studies focus on transforming graph data to the spectral domain and applying filtering operators to reduce computational time [18,55,213]. The signal filtering process acts as the feature extraction on the Laplacian matrix. Most models adopted single and undirected graphs and presented graph data as a Laplacian matrix:

$$L = I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \tag{58}$$

where D denotes the diagonal matrix of the node degree, A is the adjacency matrix. The matrix L is a symmetric positive definite matrix describing the graph structure. Considering a matrix U as a graph Fourier basis, the Laplacian matrix then could be decomposed into three components: $L = U\Lambda U^{\intercal}$ where Λ is the diagonal matrix which denotes the spectral representation of graph topology and $U = [u_0, u_1, \cdots, u_{n-1}]$ is eigenvectors matrix. The filter function g_θ resembles a k-order polynomial, and the spectral convolution acts as diffusion convolution in graph domains. The spectral graph convolution given by an input x with a filter g_θ is defined as:

$$g_\theta * x = Ug_\theta U^{\intercal}x \tag{59}$$

where $*$ is the convolution operation. Bruna et al. [56] transformed the graph data to the spectral domain and applied filter operators on a Fourier basis. The hidden state at the layer l could be defined as:

$$H_i^{(l)} = \sigma\left(V\sum_{j=1}^{c_{l-1}} D_{ij}^{(l)}V^{\intercal}H_j^{(l)}\right) \tag{60}$$

where $D_{ij}^{(l)}$ is a diagonal matrix at layer l, c_{l-1} denotes the number of filters at layer $l-1$, and V denotes the eigenvectors of the L matrix. Typically, most of the energy of the D matrix is concentrated in the first d elements. Therefore, we can obtain the first d values of the matrix V, and the number of parameters that should be trained is $c_{l-1}\cdot c_l \cdot d$.

Several studies focused on improving spectral filters to reduce computational time and capture more graph structure in the spectral domain [210,216]. For instance, Defferrard et al. [216] presented a strategy to re-design convolutional filters for graphs. Since the spectral filter $g_\theta(\Lambda)$ indeed generates a kernel on graphs, the key idea is that they consider $g_\theta(\Lambda)$ as a polynomial which includes k-localized kernel:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1}\theta_k\Lambda^{(k)} \tag{61}$$

where θ is a vector of polynomial coefficients. This k-localized kernel provides a circular distribution of weights in the kernel from a target node to k-hop nodes in graphs.

Unlike the above models, Zhuang and Ma [211] tried to capture the local and global graph structures by introducing two convolutional filters. The first convolutional operator,

local consistency convolution, captures the local graph structure. The output of a hidden layer Z^l, then, could be defined as:

$$Z^{(l)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}Z^{(l-1)}W^{(l)}) \tag{62}$$

where $\tilde{A} = A + I$ denotes the self-loops adjacency matrix, and $\tilde{D}_{i.i} = \sum_j \tilde{A}_{ij}$ is the diagonal matrix presenting the degree information of nodes. In addition to the first filter, the second filter aims to capture the global structure of graphs which could be defined as:

$$Z^{(l)} = \sigma(D^{-\frac{1}{2}}APD^{-\frac{1}{2}}Z^{(l-1)}W^{(l)}) \tag{63}$$

where P denotes the PPMI matrix, which can be calculated via frequency matrix using random-walk sampling.

Most of the above models learn node embeddings by transforming graph data to signal domain and use convolutional filters which lead to increased computational complexity. In 2016, Kipf and Welling [18] introduced graph convolutional networks (GCNs), which were considered to be a bridge between spectral and spatial approaches. The spectral filter $g_\theta(\Lambda)$ and the hidden layers of the GCN model followed the layer-wise propagation rule can be defined as follows:

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} {\theta'}_k T_k(\tilde{\Lambda}) \tag{64}$$

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}A\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{65}$$

where $\tilde{\Lambda} = \frac{2}{\lambda_{\max}}\Lambda - I_N$ and $\lambda_{\max}$ is the largest eigenvalue of Laplacian matrix L, $\theta' \in \mathbb{R}^K$ is Chebyshev coefficients vector, $T_k(x)$ is Chebyshev polynomials could be defined as:

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x) \tag{66}$$

where $T_0(x) = 1$ and $T_1(x) = x$. Consequently, the convolution filter of an input x is defined as:

$$g_{\theta'} * x \approx \sum_{k=0}^{K} {\theta'}_k T_k(\tilde{L})x, \;\; \tilde{L} = \frac{2}{\lambda_{\max}}L - I_N. \tag{67}$$

Although spectral CGNNs are effective in applying convolution filters on the spectral domain, they have several limitations as follows:

- Computational complexity: The spectral decomposition of the Laplacian matrix into matrices containing eigenvectors is time-consuming. During the training process, the dot product of the U, Λ, and U^T matrices also increase the training time.
- Difficulties for handling large-scale graphs: Since the number of parameters for the kernels also corresponds to the number of nodes in graphs. Therefore, spectral models could not be suitable for large-scale graphs.
- Difficulties for considering graph dynamicity: To apply convolution filters to graphs and train the model, the graph data must be transformed to the spectral domain in the form of a Laplacian matrix. Therefore, when the graph data changes, in the case of dynamic graphs, the model is not applicable to capture changes in dynamic graphs.

Motivated by the limitations of spectral domain-based CGNNs, spatial models apply convolution operators directly to the graph domain and learn node embeddings in an effective way. Recently, various spatial CGNNs have been proposed showing remarkable results in handling different graph structures compared to spectral models [52,95]. Based on the mechanism of aggregation from graphs and how to apply the convolution operators, we divide CGNN models into the following main groups: (i) Aggregation mechanism improvement, (ii) Training efficiency improvement, (iii) Attention-based models, and

(iv) Autoencoder-CGNN models. Tables 11 and 12 present a summary of spatial CGNN models for all types of graphs ranging from homogeneous to heterogeneous graphs.

Table 11. A summary of spatial CGNN models for static and homogeneous graphs. m is the total weight of the degrees of the Graph, V_t is the number of clusters in the graph. $P_n(v)$ is a negative sampling distribution, $A^{(k)}$ is the transition matrix at time k, and B is the batch of nodes used to calculate the gradient estimation.

Model	Graph Type	Tasks	Loss Function
HCNP [217]	Static graphs	Node classification	$-\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
CDMG [218]	Static graphs	Community detection	$-trace\left(H^{\intercal} A^{(k)} H\right)$
[219]	Static graphs	Passenger Prediction	$\left[\frac{1}{\lvert V \rvert} \sum_{v_i \in V} (\hat{y}_i - y_i)^2\right]^{\frac{1}{2}}$
ST-GDN [220]	Static graphs	Link prediction	$\sum_{v_i \in V} \lVert y_i - \hat{y}_i \rVert_2^2$
[221]	Static graphs	Node classification	$-\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
MPNNs [222]	Static graphs	Node prediction	$\sum_{v_i \in V} \lVert y_i - \hat{y}_i \rVert_2^2$
GraphSAGE [22]	Static graphs	Node classification	$\sum_{v_i \in V} -\log\left(\sigma(y_i^{\intercal} y_j)\right) - \lvert N_{neg} \rvert \mathrm{E}_{v_k \sim P_n(v)} \log(\sigma(-y_i^{\intercal} y_k))$
FastGCN [52]	Static graphs	Node classification, link prediction	$\sum_{v_i \in V} \lVert y_i - \hat{y}_i \rVert_2^2$
SACNNs [223]	Static graphs	Node classification Regression tasks	$-\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$ $\frac{1}{\lvert V \rvert} \sum_{v_i \in V} \lVert y_i - \hat{y}_i \rVert_2^2$
Cluster-GCN [95]	Static graphs	Node classification	$-\frac{1}{\lvert B \rvert} \sum_{v_i \in B} y_i^{\intercal} \log(\hat{y}_i)$
[18]	Static graphs	Node classification	$\sum_{v_i \in V} \lVert y_i - \hat{y}_i \rVert_2^2$
[224]	Static graphs	Node classification	$-\frac{1}{\lvert B \rvert} \sum_{v_i \in B} y_i^{\intercal} \log(\hat{y}_i)$
GraphSAINT [53]	Static graphs	Node classification Community prediction	$\sum_{v_i \in V} \lVert y_i - \hat{y}_i \rVert_2^2$
VGAE [72]	Static graphs	Link prediction	$E_{q(Z\lvert X,A)}[\log p(A\lvert Z)] - KL[q(Z\lvert X,A) \lvert\rvert p(Z)]$
PinSAGE [225]	Static graphs	Link prediction	$E_{n_k \sim P_n(i)} \max\left(0, Z_i \cdot Z_{n_k} - Z_i \cdot Z_j + m\right)$
Hi-GCN [54]	Static graphs	Classification tasks	$-\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
[226]	Static graphs	Link prediction	$\sum_{\langle v_i, v_j, v_k \rangle} \log \sigma\left(Z_i Z_j - Z_i Z_k\right) + \alpha L_2$
[28]	Static graph	Node classification	$\frac{1}{2} \sum_{v_i \in V} \lVert y_i - \hat{y}_i \rVert_2^2 + \alpha L_1$
[26]	Static graph	Node classification	$-\frac{1}{\lvert V \rvert} \sum_{v_i \in V} \left(y_i^{\intercal} \log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)\right)$
[227]	Static graphs	Classification tasks	$-\frac{1}{\lvert V \rvert} \sum_{v_i \in V} \left(y_i^{\intercal} \log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)\right)$
[228]	Static graph	Node Classification Link prediction	$\sum_{v_i \in V} \lVert y_i - \hat{y}_i \rVert_2^2$
[229]	Static graphs	Node classification Link prediction	$-\sum_{v_i \in V} \left(y_i^{\intercal} \log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)\right)$
DCRNN [230]	Static graphs	Node classification	$\frac{1}{\lvert V \rvert} \sum_{v_i \in V} \lvert y_i - \hat{y}_i \rvert$
PinSAGE [225]	Static graphs	Link prediction	$E_{n_k \sim P_n(i)} \max\left(0, Z_i \cdot Z_{n_k} - Z_i \cdot Z_j + m\right)$
E-GraphSAGE [231]	Static graph	Edge classification	$-\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
GraphNorm [232]	Static graphs	Graph classification	$\sum_{v_i \in V} \frac{1}{2} \lVert y_i - \hat{y}_i \rVert_2^2$
GIN [24]	Heterogeneous graphs	Node classification, Graph classification	$\sum_{v_i \in V} \frac{1}{2} \lVert y_i - \hat{y}_i \rVert_2^2$
DeeperGCN [98]	Static graphs	Node property prediction, Graph property prediction	$\sum_{v_i \in V} \frac{1}{2} \lVert y_i - \hat{y}_i \rVert_2^2$
PHC-GNNs [233]	Static graphs	Graph classification	$-\sum_{v_i \in V} \left(y_i^{\intercal} \log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)\right)$
HGNN [27]	Hypergraphs	Node classification, Recognition tasks.	$-\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
HyperGCN [234]	Hypergraphs	Node classification	$-\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$

Gilmer et al. [222] presented the MPNN (Message-Passing Neural Network) model to employ the concept of messages passing over nodes in graphs. Given a pair of nodes (v_i, v_j), a message from v_j to v_i could be calculated by a message function M_{ij}. During the

message-passing phase, a hidden state at layer l of a node v_i could be calculated based on the message-passing from its neighbors, which could be defined as:

$$m_i^{(l+1)} = \sum_{v_j \in N(v_i)} M_{(l)}\left(h_i^{(l)}, h_j^{(l)}, e_{ij}\right), \tag{68}$$

$$h_i^{(l+1)} = \sigma\left(h_i^{(l)}, m_i^{(l+1)}\right), \tag{69}$$

where $M_{(l)}$ denotes the message function at layer l which could be a MLP function, σ is an activation function, and $N(v_i)$ denotes the set of neighbors of node v_i.

Most previous graph embedding models work in transductive learning which cannot handle unseen nodes. In 2017, Hamilton et al. [22] introduced the GraphSAGE model (SAmple and aggreGatE) to generate inductive node embeddings in an unsupervised manner. The hidden state at layer $l+1$ of a node v_i could be defined as:

$$h_i^{(l+1)} = \mathrm{AGG}_{(l+1)}\left(\{h_j^{(l)}, \forall v_j \in N(v_i)\}\right) \tag{70}$$

where $N(v_i)$ denotes the set of neighbors of node v_i, $h_j^{(l)}$ is the hidden state of node v_j at layer l. The function $\mathrm{AGG}(\cdot\cdot)$ is a differentiable aggregator function. There are three aggregators (e.g., Mean, LSTM, and Pooling) to aggregate information from neighboring nodes and separate nodes into mini batches. Algorithm 1 presents the algorithm of the GraphSAGE model.

Algorithm 1: GraphSAGE algorithm. The model first takes the node features as inputs. For each layer, the model aggregates the information from neighbors and then updates the hidden state of each node v_i.

Input : $G = (V, E)$: The graph G with set of nodes V and set of edges E.
x_i: The input features of node v_i
L: The depth of hidden layers, $\forall l \in \{1 \cdots L\}$
AGG_k: Differentiable aggregator functions
$N(v_i)$: The set of neighbors of node v_i.
Output: Z_i: Vector representations for v_i.

$h_i^0 \leftarrow x_i, \forall v_i \in V$
for *k=1 to L* **do**
  for $v_i \in V$ **do**
    $h_{N(v_i)}^l \leftarrow \mathrm{AGG}_l\left(\{h_j^{l-1}, \forall v_j \in N(v_i)\}\right)$;
    $h_i^l \leftarrow \sigma\left(W^l \cdot \left(h_i^{l-1} \parallel h_{N(v_i)}^l\right)\right)$
  end
  $h_i^l \leftarrow h_i^k / ||h_i^l||_2, \ \forall v_i \in V$
end
$Z_i \leftarrow h_i^L, \ \forall v_i \in V$

Lo et al. [231] aimed to apply the GraphSAGE model to detect computer attackers in computer network systems, named E-graphSAGE. The main difference between the two models is that E-graphSAGE used the edges of graphs as aggregation information for learning embeddings. The edge information between two nodes is the data flow between two source IP addresses (Clients) and destination IP addresses (Servers).

By evaluating the contribution of neighboring nodes to target nodes, Tran et al. [229] proposed convolutional filters with different parameters. The key idea of this model is to rank the contributions of different distances from the set of neighbor nodes to target nodes

using short path sampling. Formally, the hidden state of a node at layer $l+1$ could be defined as multiple graph convolutional filters:

$$h^{r,l+1} = \|_{j=0}^{r} \left((D^j)^{-1} SP^j h^l W^{j,l} \right) \tag{71}$$

where $\|$ denotes the concatenation, r and SP^j denote the r-hop distance and the shortest-path distance j, respectively. Ying et al. [225] considered random-walk sampling as the aggregation information that can be aggregated to the hidden state of CGNNs. To collect the neighbors of node v, the idea of the model is to gather a set consisting of random-walk paths from node v and then select the top k nodes with the highest probability.

Table 12. A summary of spatial CGNN models for dynamic and heterogeneous graphs, m is the margin.

Model	Graph Type	Tasks	Loss Function
SHARE [235]	Dynamic graphs	Availability prediction	$\frac{1}{N}\left(\sum_{v_i \in V} \|y_i - \hat{y}_i\|_2^2 + y_i^\intercal \log(\hat{y}_i)\right)$
Dyn-GRCNN [236]	Dynamic graphs	Traffic flow forecasting	$\sum_{v_i \in V} \|y_i - \hat{y}_i\|_2^2$ $\left[\frac{1}{\|V\|}\sum_{v_i \in V}(\hat{y}_i - y_i)^2\right]^{\frac{1}{2}}$
STAN [237]	Dynamic graphs	Fraud detection	$\sum_{v_i \in V}\left[y_i^\intercal log(\hat{y}_i) + \alpha(1-y_i^\intercal)log(1-\hat{y}_i)\right]$
SeqGNN [238]	Dynamic graphs	Traffic speed prediction	$\sum_{v_i \in V} \|y_i - \hat{y}_i\|_2^2$
DMVST-Net [239]	Dynamic graphs	Taxi demand prediction	$\sum_{v_i \in V}\left(\|y_i - \hat{y}_i\|_2^2 + \left\|\frac{y_i - \hat{y}_i}{y_i}\right\|_2^2\right)$
ST-ResNet [240]	Dynamic graphs	Flow prediction	$\sum_{v_i \in V} \|y_i - \hat{y}_i\|_2^2$
R-GCNs [241]	Knowledge graphs	Entity classification	$-\sum_{v_i \in V} y_i^\intercal \log(\hat{y}_i)$
HDMI [242]	Multiplex graphs	Node clustering, Node classification	$-\frac{1}{\|V\|}\sum_{v_i \in V}\left(y_i^\intercal \log(\hat{y}_i) + (1-y_i)log(1-\hat{y}_i)\right)$
DMGI [243]	Multiplex graphs	Link Prediction, Clustering, Node classification	$-\frac{1}{\|V\|}\sum_{v_i \in V}\left(y_i^\intercal \log(\hat{y}_i) + (1-y_i)log(1-\hat{y}_i)\right)$
LDANE [244]	Dynamic graphs	Graph reconstruction, Link prediction, Node classification	$\sum_{v_i \in V}\left\|\hat{A}_i - A_i\right\|_2^2 + \alpha \sum_{\langle v_i, v_j \rangle \in E}\left\|Z_i - Z_j\right\|_2^2 + L_1 + L_2$
EvolveGCN [245]	Dynamic graphs	Link prediction, Node, edge classification	$-\sum_{v_i \in V} y_i^\intercal \log(\hat{y}_i)$

For hypergraphs, several GNN models have been proposed to learn high-order graph structure [27,44,234]. Feng et al. [27] proposed HGNN (Hypergraph Neural Networks) model to learn hypergraph structure based on spectral convolution. They first learn each hyperedge feature by aggregating all the nodes connected by the hyperedge. Then, each node's attribute is updated with a vector embedding based on all the hyperedges connecting to the nodes. By contrast, Yadati [234] presented the HyperGCN model to learn hypergraphs based on spectral theory. Since each hyperedge could connect several nodes between them, this model's idea is to filter far apart nodes. Therefore, they adopt the Laplacian operator first to learn node embedding and filter edges, which connect two nodes at a high distance. The GCNs could then be used to learn node embeddings.

One of the limitations of GNN models is that the models consider the set of neighbors as permutation invariant. This limitation then makes the models cannot distinguish between isomorphic subgraphs. By considering the message-passing set from neighbors of nodes as permutation invariant, several works aimed to improve the message-passing mechanism by simple aggregation functions. Xu et al. [24] proposed GIN (Graph Isomorphism Network) model, which aims to learn vector embeddings as powerful as the

1-dimensional WL isomorphism test. Formally, the hidden state of node v_i at layer l could be defined as:

$$h_i^{(l)} = \text{MLP}^{(l)}\left(\left(1+\varepsilon^{(l)}\right)\cdot h_i^{(l-1)} + \sum_{v_j \in N(v_i)} h_j^{(l-1)}\right) \tag{72}$$

where MLP denotes multilayer perceptions and ε is a parameter that could be learnable or fixed scalar. Another problem of GNNs is the over-smoothing problem when stacking more layers in the models. DeeperGCN [98] was a similar approach that aims to solve the over-smoothing problem by generalized aggregations and skip connections. The DeeperGCN model defined a simple normalized message-passing, which could be defined as:

$$m_{ij}^{(l)} = \text{ReLU}\left(h_i^{(l)} + \mathbb{1}\left(h_{e_{ij}}^{(l)} \cdot h_{e_{ij}}^{(l)}\right)\right) + \varepsilon \tag{73}$$

$$h_i^{(l+1)} = \text{MLP}\left(h_i^{(l)} + s \cdot \left\|h_i^{(l)}\right\|_2 \cdot \frac{m_i^{(l)}}{\left\|m_i^{(l)}\right\|_2}\right) \tag{74}$$

where m_{ij} denotes the message-passing from node v_j to node v_i, $h_{e_{ij}}$ is the edge feature of the edge e_{ij}, $\mathbb{1}(\cdot)$ presents an indicator procedure which is being 1 if two nodes v_i and v_j are connected. Le et al. [233] presented the PHC-GNN model, which improves the message-passing compared to the GIN model. The main difference between PHC-GNN and GIN models is that the PHC-GNN model added edge embeddings and a residual connection after the message-passing. Formally, the message-passing and hidden state of a node v_i at layer $l+1$ could be defined as:

$$m_i{}^{(l+1)} = \sum_{v_j \in N(v_i)} \alpha_{ij}\left(h_i^{(l)} + h_{e_{ij}}^{(l)}\right), \tag{75}$$

$$\tilde{h}_i^{(l+1)} = \text{MLP}^{(l+1)}\left(h_i^{(l)} + m_i^{(l+1)}\right), \tag{76}$$

$$h_i{}^{(l+1)} = h_i{}^{(a)} + \tilde{h}_i^{(l+1)}. \tag{77}$$

A few studies focused on building pre-trained GNN models, which could be used to initialize other tasks [209,246,247]. These pre-trained models are also beneficial to handle the little availability of node labels. For example, the main objective of the GPT-GNN model [247] is to reconstruct the graph structure and the node features by masking the attributes and edges. Given a permutated order, the model maximizes the node attributes based on observed edges and then generates the remaining edges. Formally, the conditional probability could be defined as:

$$p(X_i, E_i | X_{<i}, E_{<i}) = \sum_m p(X_i, E_{i,\neg m} | E_{i,m} X_{<i}, E_{<i}) \cdot p(E_{i,m} | X_{<i}, E_{<i}) \tag{78}$$

where $E_{i,m}$ and $E_{i,\neg m}$ depict the observed and masked edges, respectively.

Since learning node embeddings in the whole graphs is time-consuming, several approaches aim to apply standard cluster algorithms (e.g., METIS, K-means, etc.) to cluster nodes into different subgraphs, then use GCNs to learn node embeddings. Chiang et al. [95] proposed a Cluster-GCN model to increase the computational efficiency during the training of the CGNNs. Given a graph G, the model first separates G into c clusters $G = \{G_1, G_2, \cdots, G_c\}$ where $G_i = \{V_i, E_i\}$ using Metis clustering algorithm [248]. The model then aggregates information within each cluster. GraphSAINT model [53] had a similar structure to Cluster-GCN and [249] model. GraphSAINT model aggregated neighbor information and samples nodes directly on a subgraph at each hidden layer. The probability of keeping a connection from a node u at layer l to a node v in layer $l+1$ could be based on the node degree. Figure 18 presents an example of aggregation strategy for the GraphSAINT model. By contrast, Jiang et al. [54] presented a hi-GCN model (hierarchical GCN) that

could effectively model the brain network with two-level GCNs. Since individual brain networks have multiple functions, the first level GCN aims to capture the graph structure. The objective of the 2nd GCN level is to provide the correlation between network structure and contextual information to improve the semantic information. The work from Huang et al. [250] was similar to GraphSAGE and FastGCN models. However, instead of using node-wise sampling at each hidden layer, the model provided two strategies: a layer-wise sampling strategy and a skip-connection strategy that could directly share the aggregation information between hidden layers and improve message-passing. The main idea of the skip-connection strategy is to reuse the information from previous layers that could usually be forgotten in dense graphs.

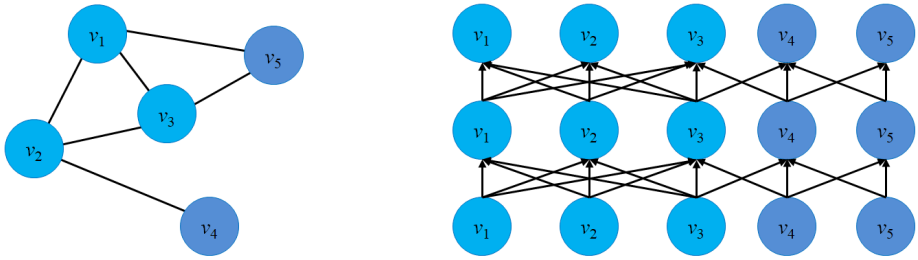
One of the limitations of the CGNNs is that at the hidden layer, the model updates the state of all neighboring nodes. This can lead to slow training and updating because of inactive nodes. Some models aimed to enhance CGNNs by improving the sampling strategy [52,223,224]. For example, Chen et al. [52] presented a FastGCN model to improve the training time and the model performance compared to CGNNs. One of the problems with existing GNN models is scalability which expands the neighborhood and increases computational complexity. The model could learn neighborhood sampling at each convolution layer which mainly focuses on essential neighbor nodes. Therefore, the model could learn the essential neighbor nodes for every batch.

By considering each hidden layer as an embedding layer of independent nodes, FastGCN aims to subsample the receptive area at each hidden layer. For each layer, they chose t_k i.i.d. nodes $u_1^{(l)}, u_2^{(l)}, \cdots, u_k^{(l)}$ and compute the hidden state which could be defined as:

$$\tilde{h}_{k+1}^{(l+1)}(v) = \frac{1}{k}\sum_{j=1}^{k}\tilde{A}(v,u_j^{(l)})h_k^{(l)}u_j^{(l)}W^{(l)} \tag{79}$$

$$h_{k+1}^{(l+1)}(v) = \sigma(\tilde{h}_{k+1}^{(l+1)}(v)) \tag{80}$$

where $\tilde{A}(v,u_j^{(l)})$ denotes the kernel, and σ denotes the activation function. Wu et al. [214] introduced SGC (Simple Graph Convolution) model, which could improve 1st-order proximity in the GCN model. The model removed nonlinear activation functions at each hidden layer. Instead, they used a final SoftMax function at the last layer to acquire probabilistic outputs. Chen et al. [224] presented a model to improve the updating of the nodes' state. Instead of collecting all the information from the neighbors of each node, the model proposed an option to keep track of the activation history states of the nodes to reduce the receptive scope. The model aimed to maintain the history state $\bar{h}_v^{(l)}$ for each state $h_v^{(l)}$ of each node v.



(**a**) (**b**)

Figure 18. An example of the GraphSAINT model. (**a**) A subgraph has five nodes v_1, v_2, v_3, v_4, and v_5; (**b**) A full GCN has three layers. (**a**) presents a subgraph with nodes. In the subgraph, there are 3 nodes (v_1, v_2, v_3) with higher order than the other two nodes (v_4, v_5). (**b**) presents a full CGNN-based model with three layers. Three nodes with higher degrees should be sampled from each other in the next layers.

Similar to [250], Chen et al. [28] presented a GCNII model using an initial residual connection and identity mapping to overcome the over-smoothing problem. The GCNII model aimed to maintain the structural identity of target nodes to overcome the over-smoothing problem. They introduced an initial residual connection H^0 at the first convolution layer and identity mapping I_n. Mathematically, the hidden state at layer $l+1$ could be defined as:

$$\mathrm{H}^{(l+1)} = \sigma\Big(\Big((1-a_l)\tilde{\mathrm{P}}\cdot\mathrm{H}^{(l)} + a_l\mathrm{H}^{(0)}\Big)\Big((1-b_l)\mathrm{I}_n + b_l\mathrm{W}^{(l)}\Big)\Big) \tag{81}$$

where $\tilde{P} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ denotes the convolutional filter with normalization. Adding two parameters $\mathrm{H}^{(0)}$ and I_n is for the purpose of tackling the over-smoothing problem.

Several models aim to maximize the node representation and graph structure by matching a prior distribution. There have been a few studies based on the idea of Deep Infomax [227] from image processing to learn graph embeddings [26,242]. For example, Velickovic et al. [26] introduced the Deep Graph Infomax (DGI) model, which could adopt the GCN as an encoder. The main idea of mutual information is that the model trains the GCN encoder to maximize the understanding of local and global graph structure in actual graphs and minimize that in fake graphs. There are four components in the DGI model, including:

- A corruption function $\mathcal{C}$: This function aims to generate negative examples from an original graph with several changes in structure and properties.
- An encoder $\phi : \mathbb{R}^{N\times M} \times \mathbb{R}^{N\times N} \to \mathbb{R}^{N\times D}$. The goal of function ϕ is to encode nodes into vector space so that $\phi(X, A) = H = \{h_1, h_2, \cdots h_N\}$ presents vector embeddings of all nodes in graphs.
- Readout function $\mathcal{R} : \mathbb{R}^{N\times D} \to \mathbb{R}^D$. This function maps all embedding nodes into a single vector (supernode).
- A discriminator $\mathcal{D} : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}$ compares vector embeddings against the global vector of the graph by calculating a score between 0 and 1 for each vector embedding.

One of the limitations of the DGI model is that it only works with attributed graphs. Several studies have improved DGI to work with heterogeneous graphs with attention and semantic mechanisms [242,243]. Similar to the DGI model, Park et al. [243] presented the DMGI model (Deep Multiplex Graph Infomax) for attributed multiplex graphs. Given a specific node with relation type r, the hidden state could be defined as:

$$\mathrm{H}^{(r)} = \sigma\left(\hat{D}_r^{-\frac{1}{2}}\hat{A}^{(r)}\hat{D}_r^{-\frac{1}{2}}\mathrm{X}\mathrm{W}^r\right) \tag{82}$$

where $\hat{A}^{(r)} = \mathrm{A}^{(r)} + \alpha\mathrm{I}_n$, and $\hat{D}_{ii} = \sum_j \hat{A}_{ij}$, $W^r \in \mathbb{R}^{n\times d}$ is trainable weights, and σ is the activation function. Similar to the DGI model, the readout function and discriminator can be employed as:

$$\mathrm{S}^{(r)} = \mathrm{Readout}(H^{(r)}) = \sigma\left(\frac{1}{N}\sum_{i=1}^{N}\mathrm{h}_i^{(r)}\right) \tag{83}$$

$$D\left(h_i^{(r)}, \mathrm{S}^{(r)}\right) = \sigma\left(\mathrm{h}_i^{(r)^T}\mathrm{M}^{(r)}\,\mathrm{s}^{(r)}\right) \tag{84}$$

where $h_i^{(r)}$ is the i-th vector of matrix $H^{(r)}$, M^r denotes a trainable scoring matrix, S^r is a function with $S^r = \sigma\left(\frac{1}{N}\sum_{i=1}^{N} h_i{}^r\right)$. The attention mechanism is adopted from [251], which could capture the importance of node type to generate the vector embeddings at the last layer. Similarly, Jing et al. [242] proposed HDMI (High-order Deep Multiplex Infomax) model, which is conceptually similar to the DGI model. The HDMI model could optimize the high-order mutual information to process different relation types.

Increasing the number of hidden layers to aggregate more structural information of graphs can lead to an over-smoothing problem [97,252]. Previous models have considered

the weights of messages to be the same role in aggregating information from neighbors of nodes. In recent years, various studies have focused on attention mechanisms to extract valuable information from neighborhoods of nodes [19,253,254]. Table 13 presents a summary of attentive GNN models.

Table 13. A summary of attentive convolutional GNN models. p_{ij}^{l} denotes the probability of an edge between two node v_i and v_j at layer l, $p_{ij} = \sigma\Big(W(h_i||h_j)\Big)$.

Model	Graph Type	Tasks	Loss Function
GAT [19]	Static graphs	Node classification	$-\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
GATv2 [58]	Static graphs	Link prediction, Graph prediction, Node classification	$-\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
Gaan [255]	Static graphs	Node classification	$-\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
GraphStar [256]	Static graphs	Node classification, Graph classification, Link prediction	$-\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
HAN [25]	Heterogeneous graphs	Node classification, Node clustering	$-\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
[257]	Static graphs	Label-agreement prediction, Link prediction	$-\sum_{v_i \in V} \left(y_i^{\intercal} \log(\hat{y}_i) + (1-y_i) log(1-\hat{y}_i)\right)$
SuperGAT [258]	Static graphs	Label-agreement Link prediction	$-\frac{1}{\lvert V \rvert}\sum_{v_i \in V} \left(y_i^{\intercal} \log(\hat{y}_i) + (1-y_i) log(1-\hat{y}_i)\right) + \alpha \sum_{l=1}^{L} L_E^l + \beta L_2$ where $L_E^l = \sum_{(v_i,v_j)} 1_{((v_i,v_j)\in E)} \log p_{ij}^l + 1_{((v_i,v_j)\notin E)} \log(1-p_{ij}^l)$
CGAT [259]	Static graphs	Node classification	$\sum_{v_i \in V} \sum_{\substack{v_j \in N(v_i), \\ v_k \notin N(v_i)}} \max\Big(0, \phi_{ik} + m - \phi_{ij}\Big) - \sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
[260]	Static graphs	Node classification	$-\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
[261]	Static graphs	Node classification, Object recognition	$-\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i) + \alpha \sum_{(v_i,v_j)\in E} \Big\lVert Z_i - Z_j \Big\rVert_2^2$
[25]	Heterogeneous graphs	Node classification, Node clustering	$-\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
[262]	Knowledge graphs	Relation prediction	$\sum_{\langle h,t,r\rangle} \sum_{\langle h',t',r\rangle} \max(0, \lVert \mathbf{h} + \mathbf{r} - \mathbf{t} \rVert_1 - \lVert \mathbf{h}' + \mathbf{r} - \mathbf{t}' \rVert_1 + m)$
[224]	Static graphs	Node classification	$-\frac{1}{\lvert V \rvert}\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
R-GCN [241]	Knowledge graphs	Entity classification, Link prediction	$-\frac{1}{\lvert V \rvert}\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
DMGI [243]	Attributed multiplex graphs	Node clustering, Node classification	$-\frac{1}{\lvert V \rvert}\sum_{v_i \in V} \left(y_i^{\intercal} \log(\hat{y}_i) + (1-y_i) log(1-\hat{y}_i)\right) + \alpha L_E + \beta L_2$ where $L_E = \Big[Z - \sigma\Big(H^{(r)} \lvert r \in R\Big)\Big]^2 - \Big[Z - \sigma\Big(\tilde{H}^{(r)} \lvert r \in R\Big)\Big]^2$
SHetGCN [263]	Heterogeneous graphs	Node classification	$-\frac{1}{\lvert V \rvert}\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
DualHGCN [264]	Multiplex bipartite graphs	Node classification Link prediction	$\sum_{(v_i,v_j)\in E} \left[\alpha \log \sigma\Big(Z_i^{\intercal} Z_j\Big) + (1-\alpha) \sum_{k=1}^{n} \Big(E_{v_k \sim P(v_i)} \log \sigma(Z_i^{\intercal} Z_k)\Big)\right]$
HANE [265]	Heterogeneous graphs	Node classification	$-\frac{1}{\lvert V \rvert}\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$
MHGCN [266]	Multiplex heterogeneous Graph	Link prediction Node classification	$-\frac{1}{\lvert V \rvert}\sum_{v_i \in V} y_i^{\intercal} \log(\hat{y}_i)$ $\sum_{(v_i,v_j)\in E} \log \sigma\Big(Z_i^{\intercal} Z_j\Big) - \sum_{(v_i,v_k)\notin E} \log \sigma\Big(-Z_i^{\intercal} Z_j\Big)$

Velickovi et al. [19] presented the GATs (graph attention networks) model, one of the first models in applying attention mechanism to graph representation learning. The purpose of the attention mechanism is to compute a weighted message for each neighbor node during the message-passing of GNNs. Formally, there are three steps for GATs which can be explained as follows:

- Attention score: At layer l, the model takes a set of features of a node as inputs $h = \{h_i \in \mathbb{R}^d | v_i \in V\}$ and the output $h' = \{h'_i \in \mathbb{R}^{d'} | v_i \in V\}$. An attention score

measures the importance of neighbor nodes v_i to the target node v_j could be computed as:

$$s_{ij} = \sigma\big(a^{\intercal}(Wh_i \parallel Wh_j)\big) \tag{85}$$

where $a \in \mathbb{R}^{2d'}$, and $W^{(k)} \in \mathbb{R}^{d' \times d}$ are trainable weights, $\parallel$ denotes the concatenation.

- Normalization: The score then is normalized comparable across all neighbors of node v_i using the SoftMax function:

$$\alpha_{ij} = \text{SoftMax}(s_{ij}) = \frac{\exp\big(s_{ij}\big)}{\sum_{v_k \in N(v_i)} \exp(s_{ik})} \,. \tag{86}$$

- Aggregation: After normalization, the embeddings of node v_i could be computed by aggregating states of neighbor nodes which could be computed as:

$$h_i{}' = \sigma\left(\sum_{v_j \in N(v_i)} \alpha_{ij} \cdot Wh_j\right) . \tag{87}$$

Furthermore, the GAT model used multi-head attention to enhance the model power and stabilize the learning strategy. Since the GAT model takes the attention coefficient between nodes as inputs and ranks the attention unconditionally, this results in a limited capacity to summarize the global graph structure.

In recent years, various models have been proposed based on the GAT idea. Most of them aimed to improve the ability of the self-attention mechanism to capture more global graph structures [253,254]. Zhang et al. [253] presented GaAN (Gated Attention Networks) model to control the importance of neighbor nodes by controlling the amount of attention score. The main idea of GaAN is to measure the different weights that come to different heads in target nodes. Formally, the gated attention aggregator could be defined as follows:

$$h_i{}' = \text{MLP}_{\theta}\left(x_i \oplus \Big\Vert_{m=1}^{M_{head}} \left(g_i^{(m)} \sum_{j \in N(v_i)} w_{ij}^{(m)} \,\text{MLP}_{\theta^{(m)}}(h_i)\right)\right) \tag{88}$$

$$g_i = [g_i^{(1)}, g_i^{(2)} \cdots g_i^{(M_{head})}] \tag{89}$$

where $\text{MLP}(\cdot)$ denotes a simple linear transformation, and $g_i^{(m)}$ is the gate value of m-th head of node v_i.

To capture a coarser graph structure, Kim and Oh [258] considered attention based on the importance of nodes to each other. The importance of nodes is based on whether the two nodes are directly connected. By defining the different attention from target nodes to context nodes, the model could solve the permutation equivalent and capture more global graph structure. Based on this idea, they proposed the SuperGAT model with two variants, scaled dot product (SD) and mixed GO and DP (MX), to enhance the attention span of the original model. The attention score s_{ij} between two nodes v_i and v_j can be defined as follows:

$$s_{ij,SD}^{(l+1)} = \frac{\left(W^{(l+1)}h_i^{(l)}\right)^{\intercal} \times W^{(l+1)}h_j^{(l)}}{\sqrt{d}} \tag{90}$$

$$s_{ij,MX}^{(l+1)} = \left(a^{(l+1)}\right)^{\intercal}\left[W^{(l+1)}h_i^{(l)} || W^{(l+1)}h_j^{(l)}\right] \cdot \sigma\left(\left(W^{(l+1)}h_i^{(l)}\right)^{\intercal} \times W^{(l+1)}h_j^{(l)}\right) \tag{91}$$

where d denotes the number of features at layer $l+1$. The two attention scores can softly decline the number of nodes that are not connected to the target node v_i.

Wang et al. [259] aimed to introduce a margin-based constraint to control over-fitting and over-smoothing problems. By assigning the attention weight of each neighbor to

target nodes across all nodes in graphs, the proposed model can adjust the influence of the smoothing problem and drop unimportant edges.

Extending the GAT model to capture more global structural information using attention, Haonan et al. [256] introduced the GraphStar model using a virtual node (a virtual start) to maintain global information at each hidden layer. The main difference between the GraphStar and GATs models is that they introduce three different types of relationships: node-to-node (self-attention), node-to-start (global attention), and node-to-neighbors (local attention). Using different types of relationships, GraphStar could solve the over-smoothing problem when staking more neural network layers. Formally, the attention coefficients could be defined as:

$$h_i^{(t+1)} = \Big\Vert_{m}^{M_{head}} \sigma\left(\sum_{r\in R}\sum_{j\in N_i^r} \alpha_{ijr}^m W_1^{m(t)} h_j^t + \alpha_{is,r=s}^m W_2^{m(t)} S^t + \alpha_{i0,r=0}^m W_3^{m(t)} h_i^t\right) \tag{92}$$

where $W_1^{m(t)}$, $W_2^{m(t)}$, and $W_3^{m(t)}$ denotes the node-to-node, node-to-start and node-to-neighbors relations at the m-th head of node v_i, respectively.

One of the problems with the GAT model is that the model only provides static attention which mainly focuses the high-weight attention on several neighbor nodes. As a result, GAT cannot learn universal attention for all nodes in graphs. Motivated by the limitations of the GAT model, Brody et al. [58] proposed the GATv2 model using dynamic attention which could learn graph structure more efficiently from a target node v_i to neighbor node v_j. The attention score can be computed with a slight modification:

$$s_{ij} = a^\intercal \sigma(W \cdot [h_i || h_j]) \, . \tag{93}$$

Similar to Wang et al. [259], Zhang et al. [260] presented ADSF (ADaptive Structural Fingerprint) model, which could monitor attention weights from each neighbor of the target node. However, the difference between GraphStar [259] and the ADSF model is that the ADSF model introduced two attention scores s_{ij} and e_{ij} for each node v_i which can capture the graph structure and context, respectively.

Besides the GAT-based models applied to homogeneous graphs, several models tried to apply attention mechanism to heterogeneous and knowledge graphs [25,261,262]. For example, Wang et al. [25] presented hierarchical attention to learn the importance of nodes in graphs. One of the advantages of this model is to handle heterogeneous graphs with different types of nodes and edges by deploying local and global level attention. The model proposed two levels of attention: node and semantic-level attention. The node-level attention aims to capture the attention between two nodes in meta-paths. Given a node pair (v_i, v_j) in a meta-path P, the attention score of P could be defined as:

$$s_{ij}^P = \text{Att}_{node}(h_i^{'}, h_j^{'}; P) \tag{94}$$

where h_i' and h_j' denote the original and projected features of node v_i and v_j via a projection function M_ϕ, respectively, and Att_{node} is a function which scores the node-level attention. To make the coefficients across other nodes in a meta-path P which contain a set of neighbors N_i^P of a target node v_i, the attention score α_{ij}^P, and node embedding with k multi-head attention can be defined as:

$$\alpha_{ij}^{P} = \frac{\exp\left(\sigma(\mathrm{s}_{ij}^{\mathrm{T}} \cdot [h_i' || h_j'])\right)}{\sum_{k \in N_i^P} \exp\left(\sigma(\mathrm{s}_{ik}^{\mathrm{T}} \cdot [h_i' || h_k'])\right)} \tag{95}$$

$$z_i^P = \Big\|_{k=1}^{K} \sigma\left(\sum_{j \in N_i^P} \alpha_{ij}^P h'_j\right). \tag{96}$$

The score z_i^P indicates how the importance of the set of neighbors based on meta-path P contributes to node v_i. Furthermore, the semantic-level aggregation aims to score the importance of meta-paths. Given an attention coefficient z_i^P, the importance of meta-path P and its normalization could be defined as w_P:

$$w_P = \frac{1}{|V|}\left(\sum_{i \in V} \mathrm{q}^{\intercal} \cdot \tanh(W \cdot z_i^P + b)\right) \tag{97}$$

$$\bar{w}_P = \frac{\exp(w_P)}{\sum_{p=1}^{l} \exp(w_p)}. \tag{98}$$

In addition to applying CGNNs to homogeneous graphs, several studies focused on applying CGNNs for heterogeneous and knowledge graphs [224,241,243,263,264,266]. Since heterogeneous graphs have different types of edges and nodes, the main problem when applying CGNN models is the aggregation of messages based on different edge types. Schlichtkrull et al. [241] introduced the R-GCNs model (Relational Graph Convolutional Networks) to model relational entities in knowledge graphs. R-GCNs is the first model to be applied to learn node embeddings in heterogeneous graphs to several downstream tasks, such as link prediction and node classification. In addition, they also use parameter sharing to learn the node embedding efficiently. Formally, given a node v_i under relation $r \in R$, the hidden state at layer $l+1$ could be defined as:

$$h_i^{(l+1)} = \sigma\left(\sum_{r \in R} \sum_{j \in N_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)}\right), \tag{99}$$

where $c_{i,r}$ is the normalization constant, and N_i^r denotes the set of neighbors of node v_i with relation r. Wang et al. [265] introduced HANE (Heterogeneous Attributed Network Embedding) model to learn embeddings for heterogeneous graphs. The key idea of the HANE model is to measure attention scores for different types of nodes in heterogeneous graphs. Formally, given a node v_i, the attention coefficients $s_{ij}^{(l)}$, attention score $\alpha_{ij}^{(l)}$, and the hidden state $h_i^{(l+1)}$ at layer $l+1$ could be defined as:

$$z_i^{(l)} = W_i^{(l)} x_i^{(l)} \;\; s_{ij}^{(l)} = (z_i^{(l)} || z_j^{(l)}) \;\; \alpha_{ij}^{(l)} = \frac{\exp(s_{ij}^{(l)})}{\sum_{v_k \in N(v_i)} \exp(s_{ik}^{(l)})} \tag{100}$$

$$h_i^{(l+1)} = \sigma\left(z_i^{(l)}\right) \oplus \left(\sum_{v_k \in N(v_i)} \alpha_{ik}^{(l)} z_l^{(l)}\right) \tag{101}$$

where $N(v_i)$ denotes the set of neighbors of node v_i, x_i denotes the feature of v_i, and $W_i^{(l)}$ is the weighted matrix of each node type.

Several studies focused on applying CGNNs for recommendation systems [228,267–269]. For instance, Wang et al. [267] presented KGCN (Knowledge Graph Convolutional Networks) model to extract the user preferences in the recommendation systems. Since most existing models suffer from the cold start problem and sparsity of user–item interactions, the proposed model can capture users' side information (attributes) on knowledge graphs. The users' preferences, therefore, could be captured by a multilayer receptive field in GCN.

Formally, given a user u, item v, N_v denotes the set of items connected to u, the user–item interaction score could be computed as:

$$\tilde{\pi}^u_{r_{v,e}} = \frac{\exp(\pi^u_{r_{v,e}})}{\sum_{e \in N(v)} \exp(\pi^u_{r_{v,e}})}, \quad v^u_{N(v)} = \sum_{e \in N(v)} \exp(\tilde{\pi}^u_{r_{v,e}} e) \tag{102}$$

where $\pi^u_{r_{v,e}}$ denotes an inner product where the score between user u and relation r, e is the representation of item v.

Since the power of the autoencoder architecture is to learn a low-dimensional node representation in an unsupervised manner, several studies focused on integrating the convolutional GNNs into autoencoder architecture to leverage the power of the autoencoder architecture [72,270]. Table 14 summarizes graph convolutional autoencoder models for static and dynamic graphs.

Table 14. A summary of graph convolutional autoencoder models. E is the edge attribute tensor, X is a node attribute matrix.

Algorithms	Graph Types	Tasks	Loss Function
GAE [72]	Static graphs	Link prediction	$E_{q(Z\|X,A)}[\log p(A\|Z)] - KL[q(Z\|X,A)\|\|p(Z)]$
VGAE [72]	Static graphs	Link prediction	$E_{q(Z\|X,A)}[\log p(A\|Z)] - KL[q(Z\|X,A)\|\|p(Z)]$
[271]	Static graphs	Graph generation	$-\alpha_1 \log p(A\|Z) - \alpha_2 \log p(X\|Z) - \alpha_3 \log p(E\|Z)$
[272]	Static graphs	Graph generation	$-\sum_{v_i \in V} \left(y_i^\intercal \log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i)\right)$
MGAE [270]	Static graphs	Graph clustering	$\left\|A - \hat{A}\right\|^2$
[273]	Static graphs	Graph reconstruction	$\frac{1}{\|V\|} \sum_{v_i \in V} \|y_i - \hat{y}_i\|_2^2$
LDANE [244]	Dynamic graphs	Graph reconstruction, Link prediction, Node classification	$\sum_{v_i \in V} \left\|\hat{A}_i - A_i\right\|_2^2 + \alpha \sum_{\langle i,j \rangle \in E} \left\|Z_i - Z_j\right\|_2^2 + L_1 + L_2$

Most graph autoencoder models were designed based on VAE (variational autoencoders) architecture to learn embeddings [274]. Kipf and Welling [72] introduced the GAE model, one of the first studies on applying autoencoder architecture to graph representation learning. GAE model [72] aimed to reconstruct the adjacency matrix A and feature matrix X from original graphs by adopting the CGNNs as an encoder and an inner product as the decoder part. Figure 19 presents the detail of the GAE model. Formally, the output embedding Z and the reconstruction process of the adjacency matrix input could be defined as:

$$\mathrm{Z} = \mathrm{GCN}(X, A) \quad \hat{\mathrm{A}} = \sigma(ZZ^\intercal) \tag{103}$$

where $\mathrm{GCN}(\cdot,\cdot)$ function could be defined by Equation (65), and σ is an activation function $\mathrm{ReLU}(\cdot) = max(0, \cdot)$. The model aims to reconstruct the adjacency matrix A by an inner product decoder part:

$$p(A, Z) = \prod_{i=1}^{N} \prod_{j=1}^{N} p(A_{ij}|Z_i, Z_j), \quad p(A_{ij} = 1|Z_i, Z_j) = \sigma(Z_i^\intercal Z_j) \tag{104}$$

where σ is the sigmoid function and A_{ij} is the value at row i-th and column j-th in the adjacency matrix A. In the training process, the model tries to minimize the loss function by gradient descent:

$$\mathfrak{L}(\theta) = E_{q(Z|X,A)}[logp(A|Z;\theta)] - \mathrm{KL}[q(Z|Z,A)||p(Z)] \tag{105}$$

where $\mathrm{KL}[q(Z|Z,A)||p(Z)]$ is the Kullback–Leibler divergence between two distributions p and q.

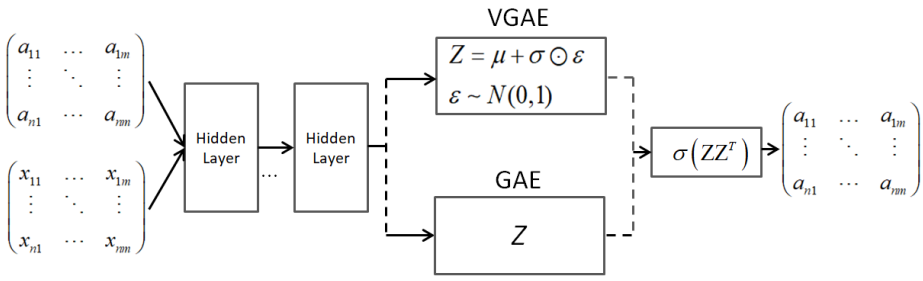


Figure 19. The architecture of GAE and VGAE model. The model adopts the adjacency matrix A and the feature matrix X as inputs. The encoder part includes two convolutional GNN layers. In the GAE model, the decoder part adopts the embedding matrix Z as input and reconstructs the adjacency matrix A using an inner product. In the VAGE model, the output of GNN could be represented as a Gaussian distribution.

Several models attempted to incorporate the autoencoder architecture into the GNN model to reconstruct graphs. For example, the MGAE model [270] combined the message-passing mechanism from GNNs and GAE architecture for graph clustering. The primary purpose of MGAE is to capture information about the features of the nodes by randomly removing several noise pieces of information from the feature matrix to train the GAE model.

The GNNs have shown outstanding performance in learning complex structural graphs that shallow models could not solve [245,275,276]. There are several main advantages of deep neural network models:

- Parameter sharing: Deep neural network models share weights during the training phase to reduce training time and training parameters while increasing the performance of the models. In addition, the parameter-sharing mechanism allows the model to learn multi-tasks.
- Inductive learning: The outstanding advantage of deep models over shallow models is that deep models can support inductive learning. This makes deep-learning models capable of generalizing to unseen nodes and having practical applicability.

However, the CGNNs are considered the most advantageous in the line of GNNs and have limitations in graph representation learning.

- Over-smoothing problem: When capturing the graph structure and entity relationships, CGNNs rely on an aggregation mechanism that captures information from neighboring nodes for target nodes. This results in stacking multiple graph convolutional layers to capture higher-order graph structure. However, increasing the depth of convolution layers could lead to over-smoothing problems [252]. To overcome this drawback, models based on transformer architecture have shown several improvements compared to CGNNs using self-attention.
- The ability on disassortative graphs: Disassortative graphs are graphs where nodes with different labels tend to be linked together. However, the aggregation mechanism in GNN samples all the features of the neighboring nodes even though they have different labels. Therefore, the aggregation mechanism is the limitation and challenge of GNNs for disassortative graphs in classification tasks.

3.4.4. Graph Transformer Models

Transformers [277] have gained tremendous success for many tasks in natural language processing [278,279] and image processing areas [280,281]. In documents, the transformer models could tokenize sentences into a set of tokens and represent them as one-hot encodings. With image processing, the transformer models could adopt image patches and use two-dimensional encoding to tokenize the image data. However, the tokenization of graph entities is non-trivial since graphs have irregular structures and disordered nodes.

Therefore, applying transformers to graphs is still an open question of whether the graph transformer models are suitable for graph representation learning.

The transformer architecture consists of two main parts: a self-attention module and a position-wise feedforward network. Mathematically, the input of the self-attention model at layer l could be formulated as $\mathrm{H} = \left[\mathrm{h}_1^l, \mathrm{h}_2^l, \cdots, \mathrm{h}_N^l\right]$ where h_i^l denotes the hidden state of position of node v_i. Then, the self-attention could be formulated as:

$$Q = HW_Q \;\; K = HW_K \;\; V = HW_V \tag{106}$$

$$S = \frac{QK^{\mathrm{T}}}{\sqrt{d_K}} \;\; \mathcal{S}(H) = \mathrm{Softmax}(S)V \tag{107}$$

where Q, K, and V depict the query matrix, key matrix, and value matrix, respectively, and d is the hidden dimension embedding. The matrix S measures the similarity between the queries and keys.

The architecture of graph transformer models differs from GNNs. GNNs use message-passing to aggregate the information from neighbor nodes to target nodes. However, graph transformer models use a self-attention mechanism to capture the context of target nodes in graphs, which usually denotes the similarity between nodes in graphs. The self-attention mechanism could help capture the amount of information aggregated between two nodes in a specific context. In addition, the models use a multi-head self-attention that allows various information channels to pass to the target nodes. Transformer models then learn the correct aggregation patterns during training without pre-defining the graph structure sampling. Table 15 lists a summary of graph transformer models.

Table 15. A summary of graph transformer models. MP is the message-passing, SPD is the shortest-path distance.

Model	Graph Type	Transformer Type	Sampling Strategy	Self-Supervised Learning
[64]	Tree-like graphs	Structural encoding	Dependency path BFS, DFS	Structure reconstruction
[65]	Tree-like graphs	Structural encoding	Dependency path	Structure reconstruction
[282]	Tree-like graphs	Structural encoding	SPD	Structure reconstruction
Graph-Bert [63]	Static graphs	Structural encoding Attention + GNN	WL and *K*-hop	Attribute reconstruction Structure reconstruction
[283]	Static graphs	Structural encoding	WL, *K*-hop	Structure reconstruction
[29]	Heterogeneous graphs	Structural encoding Edge channels	Laplacian matrix	Structure reconstruction
SAN [284]	Heterogeneous graphs	Structural encoding Edge channels	Laplacian matrix	Structure reconstruction
Grover [100]	Heterogeneous graphs	MP + Attention	*k*-hop	Feature prediction Motif prediction
Mesh Graphormer [99]	Static graphs	Attention + CGNNs	*k*-order proximity	Graph reconstruction
HGT [285]	Heterogeneous graphs	Attention+ MP	Meta-paths	Graph reconstruction
UGformer [61]	Heterogeneous graphs	Attention +GNN	1st-order proximity	Graph reconstruction
StA-PLAN [286]	Heterogeneous graphs	Attention matrix	1st-order proximity	Structure reconstruction
NI-CTR [287]	Heterogeneous graphs	Attention matrix	Subgraph sampling	Structure reconstruction
[101]	Heterogeneous graphs	MP + Attention	1-hop neighbors	Structure reconstruction
[66]	Heterogeneous graphs	Attention + MP	Subgraph	Masked label prediction
Gophormer [46]	Heterogeneous graphs	Attention matrix	Ego-graph *k*-order proximity	Node classification
Graformer [47]	Knowledge graphs	Edge channels	SPD	Structure reconstruction
Graphormer [67]	Homogeneous graphs	Edge channels	SPD	Structure reconstruction
EGT [30]	Homogeneous graphs	Edge channels	SPD	Structure reconstruction

In this section, we divide graph transformer models for graph representation learning into three main groups based on the strategy of applying graph transformer models.

- Structural encoding-based graph transformer: These models focus on various positional encoding schemes to capture absolute and relative information about entity relationships and graph structure. Structural encoding strategies are mainly suitable for tree-like graphs since the models should capture the hierarchical relations between the target nodes and their parents as well as the interaction with other nodes of the same level.
- GNNs as an auxiliary module: GNNs bring a powerful mechanism in terms of aggregating local structural information. Therefore, several studies try integrating message-passing and GNN modules with a graph transformer encoder as an auxiliary.
- Edge channel-based attention: The graph structure could be viewed as the combination of the node and edge features and the ordered/unordered connection between them. From this perspective, we do not need GNNs as an auxiliary module. Recently, several models have been proposed to capture graph structure in depth as well as apply graph transformer architecture based on the self-attention mechanism.

Several models tried to apply vanilla transformers to tree-like graphs to capture the node position [64,65,277,288]. Preserving tree structure depicts preserving a node's relative and absolute structural positions in trees. Absolute structural position describes the positional relationship of the current node to the parent nodes (root nodes). In contrast, relative structural position describes the positional relationship of the current node to its neighbors.

Shiv and Quirk [64] proposed to build a positional encoding (PE) strategy for programming language translation tasks. The significant advantage of tree-based models is that they can explore nonlinear dependencies. By custom positional encodings of nodes in the graph in a hieratical manner, the model could strengthen the transformer model's power to capture the relationship between node pairs in the tree. The key idea is to represent programming language data in the form of a binary tree and encode the target nodes based on the location of the parent nodes and the relationship with neighboring nodes at the same level. Specifically, they used binary matrices to encode the relationship of target nodes with their parents and neighbors.

Similarly, Wang et al. [65] introduced structural position representations for tree-like graphs. However, they combine sequential and structural positional encoding to enrich the contextual and structural language data. The absolute position and relative position encoding for each word w_i could be defined as:

$$\mathrm{PE}_i = f\left(\frac{\mathrm{Abs}(v_i)}{10000^{2i/d}}\right) \tag{108}$$

$$\mathrm{PE}_{ij} = \frac{x_i W^Q (x_j W^K)^\intercal + x_i W^Q (a_{ij}^K)^\intercal}{\sqrt{d}} \tag{109}$$

where Abs is the absolute position of the word in the sentence, d denotes the hidden size of K, Q matrix, $f(\cdot)$ is the sin/cos function depending on the even/old dimension, respectively, and R is the matrix presenting relative position representation.

The sentences also are represented in an independent tree which could represent the structural relations between words. For structural position encoding, the absolute and relative structural position of a node v_i could be encoded as:

$$\mathrm{PE}_i = d(v_i, root) \tag{110}$$

$$\mathrm{PE}_{ij} = \begin{cases} \mathrm{PE}_i - \mathrm{PE}_j & \text{if}(v_i, v_j) \in \mathrm{E}\,. \\ \mathrm{PE}_i + \mathrm{PE}_j & \text{if}(v_i, v_j) \notin \mathrm{E}, i > j. \\ -(\mathrm{PE}_i + \mathrm{PE}_j) & \text{if}(v_i, v_j) \notin \mathrm{E}, i < j. \\ 0 & \text{otherwise}\,. \end{cases} \tag{111}$$

where $d(\cdot\cdot)$ denotes the distance between the root node and the target nodes. They then use a linear function to combine sequential PE and structural PE as inputs to the transformer encoder.

To capture more global structural information in the tree-like graphs, Cai and Lam [282] also proposed an absolute position encoding to capture the relation between target and root nodes. Regarding the relative positional encoding, they use attention score to measure the relationship between nodes in the same shortest path sampled from graphs. The power of using the shortest path is that it can capture the hieratical proximity and the global structure of the graph. Given two nodes v_i and v_j, the attention score between two nodes can be calculated as:

$$S_{ij} = H_i W_q^{\intercal} W_k H_j \tag{112}$$

where W_q and W_k are trainable projection matrices, H_i and H_j depict the node presentation v_i and v_j, respectively. To define the relationship $r_{i \to j}$ between two nodes v_i and v_j, they adopt a bi-directional GRUs model, which could be defined as follows:

$$\vec{s}_i = \text{GRU}\left(\vec{s}_{i-1}, SPD_{i \to j}\right) \tag{113}$$

$$\overleftarrow{s}_i = \text{GRU}\left(\overleftarrow{s}_{i+1}, SPD_{i \to j}\right) \tag{114}$$

where SPD denotes the shortest path from node v_i to node v_j, $\vec{s}_i$ and $\overleftarrow{s}_i$ are the states of the forward and backward GRU, respectively.

Several models tried to encode positional information of nodes based on subgraph sampling [63,283]. Zhang et al. [63] proposed a Graph-Bert model, which samples the subgraph structure using absolute and relative positional encoding layers. In terms of subgraph sampling, they adopt a top-k intimacy sampling strategy to capture subgraphs as inputs for positional encoding layers. Four layers in the model are responsible for positional encoding. Since several strategies were implemented to capture the structural information in graphs, the advantage of Graph-Bert is that it can be trainable with various types of subgraphs. In addition, Graph-Bert could be further fine-tuned to learn various downstream tasks. For each node v_i in a subgraph $G_i = (V_i, E_i)$, they first embed raw feature x_i using a linear function. They then adopt three layers to encode the positional information of a node, including absolute role embedding, relative positional embedding, and hop-based relative distance embedding. Formally, the output of three embedding layers of the node v_i from subgraph G_i could be defined as follows:

$$\text{PE}_i^{(1)} = f(WL(v_i)), \tag{115}$$

$$\text{PE}_i^{(2)} = f(P(v_i)), \tag{116}$$

$$\text{PE}_i^{(3)} = f\big(H(v_j, v_i)\big), \tag{117}$$

$$f(x_i) = \left[sin\left(\frac{x_i}{10000^{2l/d}}\right), cos\left(\frac{x_i}{10000^{2l+1/d}}\right) \right]_{l=0}^{\lfloor \frac{d}{2} \rfloor}, \tag{118}$$

where $WL(v_i)$ denotes the WL code that labels node v_i, which can be calculated from whole graphs, l and d are the numbers of interactions throughout all nodes, and the vector dimension of nodes, $P(\cdot)$ is a position metric, $H(\cdot\cdot)$ denotes the distance metric between two nodes, and $\text{PE}_i^{(1)}$, $\text{PE}_i^{(2)}$, $\text{PE}_i^{(3)}$ denote the absolute, relative structure intimacy, and relative structure hop PE, respectively. They then aggregate all the vector embeddings together as initial embedding vectors for the graph transformer encoder. Mathematically, the transformer architecture could be explained as follows:

$$h_i^{(0)} = \mathrm{PE}_i^{(1)} + \mathrm{PE}_i^{(2)} + \mathrm{PE}_i^{(3)} + X_i \tag{119}$$

$$\mathrm{H}^{(l)} = \mathrm{Transformer}\left(\mathrm{H}^{(l-1)}\right) \tag{120}$$

$$Z_i = \mathrm{Fusion}\left(\mathrm{H}^{(l)}\right). \tag{121}$$

Similar to Graph-Bert, Jeon et al. [283] tried to present subgraphs for the paper citation network and capture the contextual citation of each paper. Each paper is considered a subgraph with nodes as reference papers. To extract the citation context, they encode the order of the referenced papers in the target paper based on the position and order of the referenced papers. In addition, they use the WL label to capture the structural role of the references. The approach by Liu et al. [289] was conceptually similar to [283]. However, there is a significant difference between them. They proposed an MCN sampling strategy to capture the contextual neighbors from a subgraph. The purpose of MCN sampling is based on the importance of the target node based on the frequency of occurrence when sampling.

In several types of graphs, such as molecular networks, the edges could bring features presenting the chemical connections between atoms. Several models adopted Laplacian eigenvectors to encode the positional node information with edge features [29,284]. Dwivedi and Bresson [29] proposed the positional encoding strategy using node position and edge channel as inputs to the transformer model. The idea of this model is to use Laplacian eigenvectors to encode the node position information from graphs and then define edge channels to capture the global graph structures. The advantage of using the Laplacian eigenvector is that it can help the transformer model learn the proximity of neighbor nodes by maximizing the dot product operator between Q and K matrix. They first pre-computed Laplacian eigenvectors from the Laplacian matrix that could be calculated as:

$$\Delta = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U^{\intercal} \Lambda U \tag{122}$$

where Δ is the Laplacian matrix, and Λ and U denote the eigenvalues and eigenvectors, respectively. The Laplacian eigenvectors λ_i then could denote the positional encoding for node v_i. Given node v_i with feature x_i and the edge feature e_{ij}, the first hidden layer and edge channel could be defined as:

$$h_i^{(0)} = A^0 x_i + \lambda_i^{(0)} + a^{(0)} \tag{123}$$

$$e_{ij}^{(0)} = B^{(0)} e_{ij} + b^{(0)}. \tag{124}$$

The hidden layers $\hat{h}_i^{(l+1)}$ of node v_i and the edge channel $\hat{e}_i^{(l+1)}$ at layer $l+1$ could be defined as follows:

$$\hat{h}_i^{(l+1)} = O_h^{(l)} \Big\|_{k=1}^{H} \left(\sum_{j \in N_i} A_{ij}^{k,l} V^{k,l} h_j^l \right), \tag{125}$$

$$\hat{e}_i^{(l+1)} = O_e^{(l)} \Big\|_{k=1}^{H} \left(A_{ij}^{k,l} \right), \tag{126}$$

$$S_{ij}^{k,l} = \left(\frac{Q^{k,l} h_i^l \cdot K^{k,l} h_j^l}{\sqrt{d_k}} \right) \cdot E^{k,l} e_{ij}^l \tag{127}$$

where Q, K, V, E are learned output projection matrices, H denotes the number of attention head.

Similar to [29], Kreuzer at al. [284] aimed to add edge channels to all pairs of nodes in an input graph. However, the critical difference between them is that they combine full-graph attention with sparse attention. One of the advantages of the model is that it could capture more global structural information since they implement self-attention to

nodes in the sparse graph. Therefore, they use two different types of similarity matrices to guide the transformer model to distinguish the local and global connections between nodes in graphs. Formally, they re-define the similarity matrix for pair of connected and disconnected nodes, which could be defined as follows:

$$\hat{S}_{ij}^{k,l} = \begin{cases} \frac{Q^{1,k,l}h_i^l \cdot K^{1,k,l}h_i^l E^{1,k,l}e_{ij}}{\sqrt{d}} & \text{if}(v_i, v_j) \in E. \\ \frac{Q^{2,k,l}h_i^l \cdot K^{2,k,l}h_i^l E^{2,k,l}e_{ij}}{\sqrt{d}} & \text{otherwise.} \end{cases} \tag{128}$$

where $\hat{S}_{ij}^{k,l}$ denotes the similarity between two nodes v_i and v_j, (Q^1, K^1, E^1) and (Q^2, K^2, E^2) are the keys, queries, and edge projections of connected and disconnected pair nodes, respectively.

In some specific cases where graphs are sparse, small, or fully connected, the self-attention mechanism could lead to the over-smoothing problem and structure loss since it cannot learn the graph structure. To overcome these limitations, several models adopt GNNs as an auxiliary model to maintain the local structure of the target nodes [99,100,285]. Rong et al. [100] proposed the Grover model, which integrates the message-passing mechanism into the transformer encoder for self-supervised tasks. They used the dynamic message-passing mechanism to capture the number of hops compatible with different graph structures. To avoid the over-smoothing problem, they used a long-range residual connection to strengthen the awareness of local structures.

Several models attempted to integrate GNNs on top of the multi-attention sublayers to preserve local structure between nodes neighbors [63,99,290]. For instance, Lin et al. [99] presented Mesh Graphormer model to capture the global and local information from 3D human mesh. Unlike the Grover model, they inserted a sublayer graph residual block with two GCN layers on top of the multi-head attention layer to capture more local connections between connected pair nodes. Hu et al. [285] integrated message-passing with a transformer model for heterogeneous graphs. Since heterogeneous graphs have different types of node and edge relations, they proposed an attention score, which could capture the importance of nodes. Given a source node v_i and a target node v_j with the edge e_{ij}, the attention score could be defined as:

$$\mathcal{S}(v_i, e_{ij}, v_j) = \text{Softmax}\left(\Big\|_{m=1}^{M_{head}} \alpha^m(v_i, e_{ij}, v_j)\right) \tag{129}$$

$$\alpha^m(v_i, e_{ij}, v_j) = K^m(v_i) W_{\tau(e_{ij})} Q^m(v_j) \frac{\mu}{\sqrt{d}} \tag{130}$$

where $\alpha^m(\cdot,\cdot,\cdot)$ denotes the m-th attention head, $W_{\tau(e_{ij})}$ is the attentive trainable weights for each edge types, K and Q are linear projection of all type of source node v_i and v_j, respectively, and μ is the importance of each relationship.

Nguyen et al. [61] introduced the UGformer model, which uses a convolution layer on top of the transformer layer to work with sparse and small graphs. Applying only self-attention could result in structure loss in several small-sized and sparse graphs. A GNN layer is stacked after the output of the transformer encoder to maintain local structures in graphs. One of the advantages of the GNN layer is that it can help the transformer model retain the local structure information since all the nodes in the input graph are fully connected.

In graphs, the nodes are arranged chaotically and non-ordered compared to sentences in documents and pixels in images. They can be in a multidimensional space and interact with each other through connection. Therefore, the structural information around a node can be extracted by the centrality of the node and its edges without the need for a positional encoding strategy. Recently, several proposed studies have shown remarkable results in understanding graph structure.

Several graph transformer models have been proposed to capture the structural relations in the natural language processing area. Zhu et al. [62] presented a transformer model to encode abstract meaning representation (AMR) graphs to word sequences. This is the first transformer model that aims to integrate structural knowledge in AMR graphs. The model aims to add a sequence of edge features to the similarity matrix and attention score to capture the graph structure. Formally, the attention score and the vector embedding could be defined as:

$$S_{ij} = \frac{\left(x_i W^Q\right)\left(x_j W^K + r_{ij} W^R\right)^\intercal}{\sqrt{d}} \tag{131}$$

$$Z_i = \sum_{j=1}^{n} \text{Softmax}(S_{ij})\left(x_j W^V + r_{ij} W^F\right) \tag{132}$$

where W^R and W^F are parameter matrices, r_{ij} is the vector representation for the relation between v_i and v_j, which could be computed by several methods, such as average values or summation. Khoo et al. [286] introduced the StA-PLAN model, which aims to detect fake news on social networking sites. Given a node v_i, the attention score and the node embedding could be defined as:

$$S_{ij} = \frac{q_i K_j^\intercal + a_{ij}^K}{\sqrt{d}} \tag{133}$$

$$Z_i = \sum_{j=1}^{n} \text{Softmax}(S_{ij})\left(Z_j + a_{ij}^V\right) \tag{134}$$

where a_{ij}^K and a_{ij}^V denotes the learned parameter vectors, which represent the relation types between v_i and v_j. The a_{ij}^K matrix aims to capture the structural information surrounding target nodes, while the purpose of a_{ij}^V matrix is to spread to other nodes.

The study from [66] aims to add the edge information between nodes to the similarity matrix. However, the difference is that they add a label information matrix combined with node features as input for the graph transformer. Formally, the feature propagation at the first layer could be defined as:

$$H^{(l+1)} = \sigma\Big(\left((1-\beta)\tilde{A} + \beta I\right)H^{(l)}\Big) \quad H^{(0)} = X + \hat{Y}W_d \tag{135}$$

where X and $\hat{Y}$ denote the input feature and partially labeled matrix, respectively. $\tilde{A} = D^{-1}A$ and β is a predefined hyper-parameter. They then put a message-passing layer on top of the multi-head attention layers to capture the local graph structures.

Schmitt et al. [47] proposed a model that adds the relative position embedding parameter to the proximity and attention score matrices in the graph-to-text problem. The main objective of this model is to define the attention score of relationships between nodes based not only on the topology of the nodes but also on their connection weights extracted from shortest paths. Specifically, the proximity matrix of a node and its attention score can be defined as:

$$S_{ij} = \frac{H_i K^Q \left(H_j W^K\right)}{\sqrt{d}} + \gamma R_{ij} \tag{136}$$

where γ denotes a scalar embedding, and R_{ij} presents a relative positional encoding between node v_i and v_j which are sampled from shortest paths P.

Ying et al. [67] introduced the Graphormer model, which aims to encode effectively graph structures. The model first captures the importance of nodes in graphs by describing the node centrality.The hidden state at the first layer of a node v_i could be defined as:

$$h_i^{(0)} = x_i + z^-_{\deg^-(v_i)} + z^+_{\deg^+(v_i)} \tag{137}$$

where $z^-_{\deg^-(v_i)}$ and $z^+_{\deg^+(v_i)}$ depict the embedding vectors of in-degree and out-degree of node v_i, respectively. To capture the global structure and the connection between nodes, they add more information about the node pairwise and edge features to the similarity matrix S. Mathematically, the similarity matrix S that captures the relation between keys and queries matrix could be defined as:

$$S_{ij} = \frac{(h_i W_Q)(h_j W_K)^\intercal}{\sqrt{d}} + b_{\varphi(v_i,v_j)} + c_{ij} \tag{138}$$

$$c_{ij} = \frac{1}{N}\sum_{n=1}^{N} x_{e_n}(w_n^E)^\intercal \tag{139}$$

where $b_{\varphi(v_i,v_j)}$ is the learnable scalar indexed by the shortest-path distance from node v_i to node v_j, (w_n^E) denotes the weight embedding of edge, and x_{e_n} denotes the n-th edge feature in the shortest path from v_i to v_j. Using the centrality encoding strategy, the Graphormer model could capture the importance of nodes in graphs that are significant in several graphs, such as the social network. Furthermore, the spatial encoding based on the shortest path could help the model capture the local and global structural information in graphs.

By contrast, Hussain et al. [30] proposed EGT (Edge-augmented Graph Transformer) model to capture the graph structure more in-depth by only using edge channels. The main idea of this model is to consider the proximity in an input graph matrix of size k-hop. In this case, the self-attention captures the edge information channels obtained using the shortest-path distance (SPD) between two nodes in an input matrix. They added edge channels to the proximity matrix of the two nodes and hidden layers for each target node. The attention matrix at layer l-th and m-th attention head could be defined as:

$$A^{m,l} = \text{Softmax}\left(H^{m,l}\right) \odot \sigma\left(G^{m,l}\right) \tag{140}$$

$$H^{m,l} = \left(\frac{Q^{m,l}\left(K^{m,l}\right)^\intercal}{\sqrt{d_k}}\right) + E^{m,l} \tag{141}$$

where m, l denote the m-th attention head and l-th hidden layer, respectively, $G^{m,l}$ and $E^{m,l}$ are the two matrices obtained from edge channels between two nodes by a linear function, $\sigma(\cdot)$ is the sigmoid function. To capture the importance of nodes, they introduced a centrality score for each node which could be obtained from a k-hop distance. The main idea is to make the model capable of distinguishing non-isomorphic subgraphs, and the model's performance is at least better than the 1-WL test. Formally, the centrality scaler matrix could be defined as:

$$s_i^{m,l} = \ln\left(1 + \sum_{j=1}^{N} \sigma\left(G^{m,l} e_{ij}\right)\right) \tag{142}$$

where N denotes the number of nodes in a matrix input and e_{ij} is the edge between two nodes v_i and v_j. In addition, they also added positional encoding, which is based on SVD. They first decompose the adjacency matrix $A \approx \hat{U}\hat{V}^\intercal$, then concatenate two matrices U and V as positional encoding. However, the experimental results show that the model's performance is not significantly improved compared to the original version.

To sum up, since the graph structure differs from the text and images mentioned above, various models have adjusted self-attention to apply the transformer to graph data. Moreover, it can also be considered that graph transformer architecture is a GAT in fully connected graphs. Therefore, in some specially structured graphs, ideal models combining GNNs as an auxiliary module for transformers also yield remarkable results. Several models [30,47] showed remarkable success in an in-depth understanding of the graph structure using edge channels based on the shortest-path distance. These results could bring a new approach to applying transformer architecture to graph representation learning.

3.5. Non-Euclidean Models

Graph representation learning models in Euclidean space have shown significant results for various applications [4,291]. In Euclidean space, graph representation learning models aim to map the graph entities to low-dimensional vector points. However, in the real world, graphs could have complex structures and various shapes, and the number of nodes could increase exponentially over time [292]. Representing such graphs in Euclidean space could lead to an incomplete representation of the graph structure and information loss [68,293]. Several recent studies have focused on representing complex structural graphs in non-Euclidean space and different metrics, which yielded some desirable results [68,70,102,293]. Each type of geometry has the advantage of describing differently shaped graph structures. For graph representation in non-Euclidean spaces, there are two typical spaces, spherical and hyperbolic, each one has its advantages. Spherical space could represent graph structures with large cycles, while hyperbolic space is suitable for hierarchical graph structures. Another method is Gaussian-based models, which could learn embeddings as a probability distribution in a latent space. This can be appropriate with the distribution in several graphs since a node could belong to different clusters based on probability density. This section covers various models in non-Euclidean space and Gaussian models.

3.5.1. Hyperbolic Embedding Models

Hyperbolic geometry has the advantage of representing hierarchical graph data, which is tree-like and mostly obeys the power law [292]. Since the Euclidean operators could not be implemented directly in hyperbolic space, most models focus on transforming the properties of models from hyperbolic space (e.g., operators, optimization) to a tangent space where we are familiar with Euclidean operators. We first briefly introduce some basic notions and definitions of hyperbolic geometry and then cover graph embedding models later.

Definition 9 (Hyperbolic space [102])**.** *A hyperbolic space (sometimes called Bolyai–Lobachevsky space) is an n-dimensional Riemannian manifold of constant negative curvature. When* $n = 2$*, it is also called the hyperbolic plane.*

Due to the complex structure of hyperbolic space, the visual representation of data and implementing operators in hyperbolic space seems complicated. Most models use a tangent space to approximate a manifold as an n-dimensional vector space. Formally, the manifold and tangent space could be defined as follows:

Definition 10 (Manifold and Tangent space [293])**.** *A manifold* $\mathcal{M}$ *of multi-dimension n is a topological space where the Euclidean space* R^n *could locally approximate its neighborhood. When* $n = 2$*, it is also called surfaces. A tangent space* $\mathcal{T}_v\mathcal{M}$ *is a Euclidean space* R^n *that approximates the manifold* $\mathcal{M}$ *at any node v in graphs.*

The hyperbolic space is a smooth Riemannian manifold, considered a locally Euclidean space where we could generate Euclidean operations. The Riemannian manifold could be defined as follows:

Definition 11 (Riemannian manifold [293]). *A Riemannian manifold is defined as a tuple* $(\mathcal{M}, g)$*, where* g *denotes a Riemannian metric which is a smooth collection of inner products on the associated tangent space:* $\langle \cdot \cdot \rangle_v : \mathcal{T}_v\mathcal{M} \times \mathcal{T}_v\mathcal{M}$*. The metric space* g *denotes curvature properties, such as the angle and the volume.*

There are several isometric models which are different metrics. However, two hyperbolic models, Poincaré and Lorentz, are widely studied in graph representation learning. Mathematically, the Poincaré and Lorentz models could be defined as:

Definition 12 (Poincaré Model [70]). *A Poincaré ball is a Riemannian manifold with a tuple* $\left(\mathcal{B}_c^n, g_x^{\mathcal{B}}\right)$*, where* c *is a negative curvature, and* $\mathcal{B}_c^n = \left\{x \in R^n : \|x\|^2 < -\frac{1}{c}\right\}$ *is a open ball with radius* $r = 1/\sqrt{|c|}$*. The matrix tensor* $g_x^B = \left(\lambda_x^2\right)^2 g^E$ *denotes a conformal factor* $\lambda_x^c = \frac{2}{1+c\|x\|_2^2}$ *and* g^E *is a Euclidean matrix. Since* R^2 *could present a single hierarchical structure sufficiently, the Poincaré disk* B_2^n *is commonly used to define hyperbolic geometry.*

Unlike the Poincaré disk, the Lorentz model is suitable for representing cyclic graphs. The Lorentz model has different characteristics from the Poincaré disk, but they are equivalent and could be transformed into each other. Mathematically, the Lorentz model is defined as follows:

Definition 13 (Lorentz/hyperboloid Model [102]). *A Lorentz or hyperboloid model is a Riemannian manifold with a tuple* $\left(\mathcal{L}_c^n, g_x^{\mathcal{L}}\right)$*, where* $L_c^n = \left\{x \in R^{n+1} : \langle x, x \rangle_L < \frac{1}{c}\right\}$ *with a negative curvature* c*, and* $g_c^n = diag([-1\,1\,1\,\ldots\,1])_n$*.*

Most studies flatten a hyperbolic manifold and then apply graph operations in tangent space, which are similar to Euclidean space. Once the results are available, they will be mapped back into the hyperbolic space. The projection of components from hyperbolic space to the manifold and back projection is handed through exponential and logarithmic mapping functions, which will be shown in the models below in detail. Table 16 summarizes hyperbolic models for graphs.

Table 16. A summary of hyperbolic models.

Models	Graph Types	Hyperbolic Models	Model Types
[70]	Homogeneous graphs	Poincaré disk	Shallow models
[102]	Homogeneous graphs	Lorentz model	Shallow models
[71]	Heterogeneous graphs	Poincaré disk	Shallow models
[293]	Homogeneous graphs	Poincaré disk	Convolutional GNNs
[294]	Homogeneous graphs	Poincaré disk, Lorentz model	GNNs
LGCN [69]	Homogeneous graphs	Lorentzian model	GNNs
[68]	Homogeneous graphs	Gyrovector model	GAT

Nickel Kiela [70] was among the first studies to learn graph embeddings in Poincaré ball based on similarities and hierarchies of nodes. They first put all nodes in graphs into the Poincaré disk and optimize the distance between pairwise nodes. Mathematically, the distance measure in Poincaré disk between two nodes v_i and v_j could be defined as:

$$d\left(Z_i, Z_j\right) = \operatorname{arccosh}\left(1 + 2\frac{\left\|Z_i - Z_j\right\|^2}{\left(1 - \left\|Z_i\right\|^2\right)\left(1 - \left\|Z_j\right\|^2\right)}\right). \tag{143}$$

They then define operators and compute the loss function on the tangent space. The loss function can be minimized using Riemannian SGD (RSGD) optimization. Formally, the loss function is defined as:

$$\mathfrak{L}(V) = \sum_{(v_i,v_j)\in E} \log \frac{e^{-d(v_i,v_j)}}{\sum_{v_k\in N_{neg}(v_i)} e^{-d(v_i,v_k)}} . \tag{144}$$

Similarly, the study of Nickel and Kiela [102] tried to improve embeddings in the Poincaré model by learning pairwise hierarchical relations in graphs. However, the difference between [70,102] is that they adopted the Lorentz model to learn embeddings. Wang et al. [71] tried to learn embeddings of heterogeneous graphs in the Poincaré disk. The meta-paths are generated using random-walk sampling strategies. They then use Equation (143) to calculate the distance between two nodes in the vector space. The Riemannian stochastic gradient descent (RSGD) is used to optimize the objective function, which minimizes the proximity between target nodes and their neighbors. Mathematically, given a node v_i and set of its neighbors $N(v_i)$, the objective function could be defined as:

$$\mathfrak{L}(V) = \sum_{(v_i,v_j)\in E} \left[\alpha \log \sigma\left(Z_i^\intercal Z_j\right) + (1-\alpha) \sum_{k=1}^{n} \left(E_{v_k\sim P(v_i)} \log \sigma\left(Z_i^\intercal Z_k\right)\right)\right] . \tag{145}$$

Since there is no definition of GNN operations in the hyperbolic space, most models tried to transform GNN operators from the hyperbolic space to the tangent manifold and performed the operators in this space. The work of Chami et al. [293] aimed to transform features from the Euclidean space to a tangent manifold and perform aggregation and activation functions on this space. The results are then projected to the H space. Exponential and logarithmic functions are used to map between T and H space. Given a vector $x^{0,E} \in R^d$ in Euclidean space, the mapping features from Euclidean space into hyperboloid manifold could be defined as:

$$x^{0,H} = \exp_{\mathbf{o}}^{C}\left(0, x^{0,E}\right) = \left(\sqrt{C}\cosh\left(\frac{||x^{0,E}||_2}{\sqrt{K}}\right), \sqrt{C}\sinh\left(\frac{||x^{0,E}||_2}{\sqrt{K}}\right)\frac{x^{0,E}}{||x^{0,E}||_2}\right) \tag{146}$$

where $\mathbf{o} := \left\{\sqrt{C}, 0, \cdots, 0\right\} \in H^{d,C}$ denotes the original pole in the hyperbolic space. The model defines trainable curves C at different layers and mapping operations between the hyperbolic space and manifold. After mapping input features into hyperbolic space, the definition operators for the message mapping mechanism can be defined as:

$$h_i^{l,H} = \left(W^l \otimes^{K_{l-1}} x_i^{l-1,H}\right) \oplus^{C_{l-1}} \mathbf{b}^l \tag{147}$$

$$m_i^{l,H} = \mathrm{AGG}^{C_{l-1}}\left(h^{l,H}\right)_i \tag{148}$$

$$Z_i^{l,H} = \sigma^{\otimes^{C_{l-1},C_l}}\left(m_i^{l,H}\right) \tag{149}$$

where $\mathrm{AGG}(\cdot)$ denotes the hyperbolic aggregation, which is based on the attention mechanism and could be calculated as:

$$\mathrm{AGG}^{C}\left(x^H\right)_i = \exp_{x_i^H}^{C}\left(\sum_{v_j\in N(v_i)} w_{ij} \log_{x_i^H}^{C} x_j^H\right) . \tag{150}$$

Similarly, Zhang et al. [68] used Gyrovector space to build GNN layers in hyperbolic space. The Gyrovector space is an open d-dimensional ball which could be defined as:

$$D_c^d := \left\{x \in R^d : c\|x\|^2 < 1\right\} \tag{151}$$

where c denotes the radius of the ball. They first put input features x from Euclidean space into the Gyrovector ball by an exponential mapping:

$$x^H = \exp_o^c(x) = \tanh\left(\sqrt{c}\|x\|\right)\frac{x}{\left(\sqrt{c}\|x\|\right)}\,. \tag{152}$$

After exponential mapping, a linear transform is used as a latent representation of each node. Formally, the hidden state of a node is obtained by applying a shared linear transformation matrix M:

$$h_i = M\otimes^c x^H = \frac{1}{\sqrt{c}}\tanh\left(\frac{\|Mx^H\|}{\|x^H\|}\tanh^{-1}\left(\sqrt{c}\|x\|\right)\right)\,. \tag{153}$$

Liu et al. [294] employed a similar approach for HGNNs. However, the main objective of this study aimed to compare which space could be suitable for graph data representation between Poincaré disk and Lorentz space in terms of implementing GNN models. Zhang et al. [69] proposed an LGCN model to learn embeddings on the Lorentzian model. They first map input features from Euclidean space to hyperbolic space and rebuild GNN operators, such as dot product and linear transformation. In addition, they aggregate information from neighborhood nodes by computing the centroid of nodes in the hyperbolic space. Given a node v_i and its feature $h_i^{d,C} \in H^{d\times C}$ and a set of its neighbors $N(v_i)$, finding a centroid of nodes could be considered as an optimization problem:

$$\bar{c}^{d,C} = \underset{c^{d,C}\in H^{d,C}}{\arg\min}\sum_{j\in N(v_i)} w_{ij}d_L^2\left(h_j^{d,C}, c^{d,C}\right) \tag{154}$$

$$c^{d,C} = \sqrt{C}\frac{\sum_{j\in N(v_i)} w_{ij}h_j^{d,C}}{\left|\left\|\sum_{j\in N(v_i)} w_{ij}h_j^{d,C}\right\|_L\right|} \tag{155}$$

where w_{ij} denotes the weights that could be normalized and computed via an attention coefficient μ as:

$$w_{ij} = \frac{\exp(\mu_{ij})}{\sum_{v_m\in N(v_i)}\exp(\mu_{im})} \tag{156}$$

$$\mu_{ij} = -d_L^2\left(M\otimes^C h_i^{d,C}, M\otimes^C h_j^{d,C}\right) \tag{157}$$

where d_L^2 denotes a squared Lorentzian distance [295], M is a matrix to transform node feature to attention-based space.

3.5.2. Spherical Embedding Models

Spherical geometry is a topological space that could represent graph structure with large cycles [296]. A spherical space is an n-dimensional Riemannian manifold of constant positive curvature ($c > 0$). The implementation of operators is similar to hyperbolic space. For each point x in the spherical space S, the connection between the spherical space S and a tangent space $T_xS_c^n$ could be computed through exponential and logarithmic mapping, which could be defined as:

$$\exp_x^c(v) = x\oplus^c\left(\tanh\left(\sqrt{c}\frac{\lambda_x\|v\|}{2}\right)\frac{v}{\sqrt{c}\|v\|}\right) \tag{158}$$

$$\log_x^c(y) = \frac{2}{\sqrt{c}\lambda_x}\tanh^{-1}\left(\sqrt{c}\|-x\oplus^c y\|\right)\frac{-x\oplus^c y}{\|-x\oplus^c y\|} \tag{159}$$

where x and y are two points in the S space and $v \in T_x S_c^n$. The distance between x and y, and the operator $\oplus_c$ is the Möbius addition for any $x, y \in S$ which could be defined as:

$$d_c(x,y) = \frac{2}{\sqrt{c}} \tanh^{-1}\left(\sqrt{c}\|-x \oplus^c y\|\right) \tag{160}$$

$$x \oplus^c y = \frac{\left(1 + 2c\langle x,y\rangle + c\|y\|^2\right)x + \left(1 - c\|x\|^2\right)y}{1 + 2c\langle x,y\rangle + c^2\|x\|^2\|y\|^2} \tag{161}$$

A few studies on spherical space have yielded promising results in recent years [103,297]. For instance, Cao et al. [103] proposed combining the representation of the knowledge graphs into three different spaces, including Euclidean, hyperbolic, and spherical spaces. Specifically, each entity e of the knowledge graph could be presented by three embeddings: Euclidean space E_e, hyperbolic space E_h, and hypersphere space E_s. For a triplet (h, r, t) denotes the head, relation, and tail, respectively, in the knowledge graph, the embedding of an entity e in the hyperbolic and hypersphere space could be defined as:

$$H_e = r \otimes^v \exp_o^v(e) \tag{162}$$

$$S_e = r \otimes^u \exp_o^u(e) \tag{163}$$

where H_e and S_e denote the embedding of entity e in the hyperbolic and hypersphere space with two negative and positive curvatures u and v, respectively. They then can obtain the embedding for each entity by combining embedding components from different spaces through the exponential function.

3.5.3. Gaussian Embedding Models

Most of the aforementioned graph embedding models represent graph entities as vector points in latent space. However, several models proposed using probability distributions to learn embeddings, considering each entity as density-based embedding. Unlike vector-point embedding models, density-based models learn embeddings as continuous density in latent space. Vector embeddings could be represented as a multivariate Gaussian distribution $P \sim \mathcal{N}(\mu, \Sigma)$. Table 17 presents a summary of Gaussian embedding models for various types of graphs.

Table 17. A summary of Gaussian embeddings models.

Model	Graph Type	Model	Structure Preservation
VGAE [72]	Homogeneous graphs	Autoencoder-based GCNs	Random-walk sampling
DVNE [298]	Homogeneous graphs	Autoencoder	1-order, 2-order proximity
[104]	Heterogeneous graphs	MLP	Meta-path
[23]	Homogeneous graphs	Autoencoder	k-order proximity
KG2E [299]	Knowledge graphs	Triplet score (h, t, r)	1-order proximity

Most Gaussian embedding models are inspired by the Word2Gauss approach [300] in natural language processing. Each word is projected into an infinite-dimensional space rather than a vector which could enable a rich geometry for better quantification of the word-type properties in the latent space. Kipf and Welling [72] introduced a VGAE (Variational Graph Autoencoder) model based on an autoencoder architecture. The encoder part includes two convolutional graph layers. The model takes an adjacency matrix A and features X as input for GCNs layers. Mathematically, the μ and $\log \Sigma^2$ parameters can be defined as:

$$\mu = \sigma_\mu\left(H^0, A\right) = \tilde{A}H^0W_1 \tag{164}$$

$$\log\Sigma^2 = \sigma_\Sigma\left(H^0, A\right) = \tilde{A}H^0W_1 \tag{165}$$

The vector embedding Z_i for each node v_i could be defined as:

$$q(Z_i|X, A) = \mathcal{N}\left(Z_i|\mu_i, diag\left(\Sigma_i^2\right)\right). \tag{166}$$

Zhu et al. [298] proposed DVNE (Deep Variational Network Embedding) model to preserve the similarity between the distributions based on autoencoder architecture. The DVNE model aims to preserve 1st-order and 2nd-order proximity in Wasserstein space. The main objective is to minimize the Wasserstein distance between distributions over the Gaussian distribution. For $p \in [0, \infty)$, the Wasserstein p-distance between two distributions P and Q could be defined as:

$$d_p(P, Q) := \inf_{x,y} \|x - y\|_p \tag{167}$$

where (x, y) is all pairs of random variables. Since Gaussian distribution is used to present the uncertainty of nodes in latent space, they aim to preserve the Wasserstein distance, which could be formulated as:

$$W_2(\mathcal{N}(\mu_1, \Sigma_1); \mathcal{N}(\mu_2, \Sigma_2))^2 = \|\mu_1 - \mu_2\|_2^2 + \|\Sigma_1 - \Sigma_2\|_F^2 \tag{168}$$

where Σ_1 and Σ_2 are diagonal covariance matrices. They use the square-exponential loss to minimize the proximity and the reconstruction loss that could be defined as:

$$\mathfrak{L}(V) = \mathfrak{L}_1(V) + \alpha\mathfrak{L}_2(V) \tag{169}$$

$$\mathfrak{L}_1(V) = \sum_{v_i, v_j, v_k \in V} E_{ij}^2 + \exp(-E_{ik}) \tag{170}$$

$$\mathfrak{L}_2(V) = \underset{X, \hat{X}}{inf} \left\|X \circ \left(X - \hat{X}\right)\right\|_2^2 \tag{171}$$

where (i, j, k) denotes a tuple (v_i, v_j, v_k) from k-hop neighborhood of v_i with constraints defined in Equation (174).

Santos et al. [104] targeted node representation associated with the uncertainty in classification tasks for heterogeneous graphs. Specifically, each node v_i is projected into latent space, which is followed by a Gaussian distribution $Z_i = \mathcal{N}(\mu_i, \Sigma_i)$. The key objective of the model is to minimize the loss function for the classification problem and regularization for structural loss using stochastic gradient descent. In terms of structure preservation, they aim to preserve the 1-hop distance for each target node in graphs. They use KL Divergence to minimize the difference between two probability distributions which could be defined as:

$$\mathfrak{L}(V) = \sum_{v_i \in V} \sum_{v_j \in N(v_i)} w_{ij} D_{KL}(Z_i||Z_j), \tag{172}$$

$$D_{KL}(Z_i||Z_j) = \frac{1}{2}\left(tr\left(\Sigma_i^{-1}\Sigma_j\right) + (\mu_i - \mu_j)^\intercal \Sigma_i^{-1}(\mu_i - \mu_j) - d - \log\frac{\det\left(\Sigma_j\right)}{\det\left(\Sigma_i\right)}\right) \tag{173}$$

where w_{ij} denotes the weight of e_{ij}.

Similar to [104], Bojchevski and Gunnemann [23] proposed a G2G (Graph2Gauss) model, an idea of learning node embeddings as uncertain. The difference between G2G and [104] is that the G2G model could preserve up to k-hop neighborhood proximity, which captures the global graph structure. Given a target node v_i and set of its neighbors within

k-hop distance N_{ik}, the objective of G2G is to build a set of constraints that the dissimilarity measure from node v_i to all nodes in N_{i1} should be smaller compared to all nodes in N_{i2} and so on, up to k-hop. Mathematically, the pairwise constraints could be defined as:

$$\mathbf{E}(Z_i, Z_j) < \mathbf{E}(Z_i, Z_m), \forall v_i \in \mathrm{V}, \forall v_m \in N_{ik}, \forall j < m \, . \tag{174}$$

Similar to [104], Equation (173) is used to measure the dissimilarity between two distributions, and they adopt square-exponential loss for optimization. Since there has been an uncertain lack of information about node embedding in latent space, the G2G model could learn node embeddings efficiently by representing nodes as Gaussian distribution. In addition, the personalized ranking could learn the order of nodes in graphs and the distance between them, eventually capturing local and global structural information.

To learn embeddings in knowledge graphs, He et al. [299] proposed the KG2E model to learn the certainty of entities and relations in knowledge graphs. This first study aims to learn node embeddings based on density in knowledge graphs. Furthermore, KG2E adopted two methods to measure the scores of triplets to learn embeddings based on symmetric and asymmetric similarity. For each triplet (h, r, t) which denotes head, relation, and tail, respectively, there are three different Gaussian distributions $H \sim \mathcal{N}(\mu_h, \Sigma_h)$, $R \sim \mathcal{N}(\mu_r, \Sigma_r)$,$T \sim \mathcal{N}(\mu_t, \Sigma_t)$.

The score function of the KG2E model could be defined as:

$$s(h, r, t) = s(P_e, P_r) = D_{KL}(P_e, P_r) \tag{175}$$

where P_e denotes probability distribution $P_e \sim \mathcal{N}(\mu_h - \mu_t, \Sigma_h - \Sigma_t)$, and $D_{KL}(\cdot\,\cdot)$ is defined in Equation (173).

4. Applications

This section focuses on practical applications of graph representation learning in various fields. We first explain how a graph can be constructed in different contexts and then discuss how graph-based models could be applied in practice. In several areas, graph embedding models may not be applied directly to solve specific tasks in the real world. However, they could act as auxiliary modules to help improve the performance of specific tasks.

4.1. Computer Vision

In image processing, a graph could be constructed for image processing problems by representing each pixel as a node and each edge describing the relationship between nodes. Several CGNNs have been proposed for the task of learning convolutional filters in the frequency domain [56,96,301,302] for classification tasks. For instance, Defferrard et al. [96] transform images from the spatial domain to the spectral domain using a Fourier transform. They then learn the convolution filter on the frequency domain to produce a sparse Laplacian matrix as input for classification tasks.

Each image segment or an entire image could be considered to be nodes and edges describing the relationships between them. Several graph-based methods adopt this strategy for the clustering tasks [303,304]. For example, Yang et al. [304] first extract image features from a CNN model and then build a large face image dataset. By considering k nearest neighbors as super nodes and the relationship between them as edges, they could construct graphs and use CGNNs to learn the cluster labels.

By considering each object in images as nodes and the relations between them as edges, several GNNs are applied to learn the proximity between the objects [305,306]. The graph embedding models can help image processing algorithms understand images' semantic relationships and spatial structure more deeply. CGNNs could aid in connecting relationships between objects in images and scene graphs [307,308]. For example, Johnson et al. [308] used scene graphs to predict corresponding layouts by calculating embeddings for objects and their relationships in the image. The model is used to learn the vector embeddings for

objects. CGNNs could also help build a reasoning network for objects in images to capture the interaction between objects [309,310]. Chen et al. [309] proposed CGNNs for relation reasoning for new actions, which should be more friendly in interaction space. CGNNs with a self-attention mechanism could help enhance the object representation in images combined with text guidance [310]. This strategy could capture relations between arbitrary regions in images and the interactions between objects in images.

Graphs are also constructed by combining general knowledge of text and images with image-question-facts. Specifically, each node in the graph is an embedding processed from the word and image processing algorithms, and edges represent the relationship between them. CGNNs are used to learn embeddings to retrieve the correct fact. For instance, Cui et al. [306] build a joint model by combining the semantic and spatial scene graphs to find internal correlations across object instances in images. They first use object detection approaches to detect objects in the images. Then, a semantic graph is constructed with nodes as objects, and edges connect objects in the image.

The sequence of skeletons is treated as a dynamic graph consisting of a sequence of snapshots. Each snapshot corresponds to a skeleton frame where each node is a joint, and the edge describes the connection of bones. Several graph-based models effectively learn features containing joint and bone information and their dependencies, which can facilitate action recognition. For instance, spatial and motion information in skeleton data could be presented in graphs for pose prediction [311]. CGNNs could also help to understand and recognize action sequences in videos and object relationships [312–314]. The models could assign candidate moments by structural reasoning to model relations between moments in videos. Each moment could be considered to be a node, and the edges are relations between them.

4.2. Natural Language Processing

A graph can be built by considering each word/document as a node, and edges could describe the relationship between the nodes or their occurrence frequency in a given context. Recently, graph-based models, which are mainly based on GNNs have attracted much attention in several applications to text classification tasks [16,18,22,211]. These models can capture the rich relational structure and preserve global structure information of documents. For instance, the DGCN model [211] was proposed to classify scientific publications by considering each paper as a node and edges as reference citations. Hamilton et al. [22] build document graphs from Reddit post data and citation data to predict paper and post categories.

Each sentence could be represented as a graph, with each node being a word and an edge describing the dependency between them. Recently, the graph-based models applied in machine translation show the potential of syntax-aware feature representations of words [315,316]. For instance, CGNNs could be used to predict syntactic dependency trees of source sentences to produce representations of words [315,316]. Bastings et al. [315] first transformed sentences into syntactic dependency trees. They then use convolution layers to learn dependency relation types which could support language models to understand the meanings of words in depth. SynGCN [317] could capture the structural relation between words in sentences from a dependency graph. They consider nodes as words and edges as the co-occurrence frequencies of two words in the entire corpus. The structural semantics could then be used to improve the performance of the Elmo model. F-GCN model (fusion GCN) from [318] can help a dialog system deal with diagram questions. They then use RNNs to capture the meaning of answers by considering the answer representation obtained from F-GCN as inputs.

4.3. Computer Security

The development of technology has increased the cyber security risk that is a social concern. Researchers have proposed various solutions, such as firewalls and intrusion detection systems against network attacks. The intrusion detection system could be divided

into two main approaches: predefined rule-based and artificial intelligence-based. In recent years, several GNNs have also been applied to improve the detection of network attacks [231,319].

A graph network is constructed by nodes that are IP addresses and edges that are packet data flows exchanged between IP addresses. Hao et al. [319] proposed a Packet2Vec model to capture the proximity features based on graph representation to build an intrusion detection system. They consider each network traffic flow as a graph where nodes are packets and edges denote the similarity between two packets. They then prune the relational graph to obtain a local proximity feature for each graph and use this as input for an autoencoder that could learn embeddings for each network flow. By contrast, Lo et al. [231] built an intrusion detection system by improving the GraphSAGE model for building intrusion detection systems. They construct a computer graph by considering each IP address as a node and the edges as links between IP addresses. By constructing the computer network graph, they can train the model with packet information from clients to the server to detect anomalous information.

Since the source code can be represented as an abstract syntax tree, several graph embedding models have been proposed to help detect malware code by learning dependency graphs. The dependency graph is built with API function nodes and directed edges representing other functional queries from the current function [320,321]. For instance, Narayanan et al. [320] built rooted subgraphs that capture the connection between API functions in source code. The model learns latent representations of rooted subgraphs and detects malware code in an Android operating system.

4.4. Bioinformatics

Drug discovery is vital in finding new chemical properties to treat diseases. A graph could represent the interaction between drug–drug, drug–target, and protein–protein by considering each node as a drug or a protein, and the edges describe the interaction between them. Since searching for successful drug candidates is challenging, graph-based models can aid experiments in the chemistry area. Several models [322–324] use a matrix-factorization-based model to predict the interaction between the clinical manifestations of diseases and their molecular signatures. This could contribute to predicting potential diseases based on human genomic databases. Yoshihiro et al. [325] constructed a bipartite graph as a chemical and genomic space to capture the interaction between drug and protein nodes. The matrix factorization-based model is used to learn embeddings and detect potential drug interactions [326,327]. The matrix factorization-based model is also used to project drugs and targets into a common low-rank feature space and create new drugs and targets for predicting drug–target interactions [328–330].

For protein–protein interaction presentation, the atoms could be considered to be nodes and edges are bonds that link two atoms. CGNNs help to predict the properties of molecular and classification tasks [323,324]. The attention-based CGNN model could predict chemical stability [331]. For identifying drug targets, several CGNNs are used to present the structure of protein–protein interaction assessment and function prediction [332]. The DeepWalk model measures similarities within a miRNA-disease association network [333].

In recent years, various GNN-based models have been proposed to predict drug–drug interactions [334–337]. A knowledge graph is constructed by a set of entity-relation-entity triples that describe the interactions between drug–drug nodes. Most knowledge graphs comprise drug features gained from DrugBank or KEGG dataset. GNN-based models could then explore the topological structure of drugs in the knowledge graph to predict the potential drug–drug interactions. For example, Lin et al. [337] proposed a GNN model to learn drug features and knowledge graph structure to predict the drug–drug interaction. Su et al. [334] proposed a DDKG model based on attentive GNNs to learn the drug embedding. The key idea of this model is first to initialize the node features based on SMILE sequences gained from a random-walk sampling strategy. This could construct the node features,

bringing a global structure at the initial step. The model then learns node embeddings based on attention from the neighborhood and triple facts.

For drug–target interaction, which is a crucial area in drug discovery, several graph-based models could help to predict the drug–target interactions [325,338–340]. For example, Hao et al. [338] proposed a GNN-based model to learn drug–target interaction. A heterogeneous graph is constructed by nodes denoting a drug–target pair, and the edges describe the connection strength between the pairs. The model then applies the graph convolution filter to learn the feature of drug–protein pairs. Peng et al. [340] introduced EEG-DTI (end-to-end graph drug–target interactions) model to predict the relations between drugs and targets based on the GCN model. A heterogeneous graph represents the interactions between drugs and targets (e.g., drug–drug interaction and drug–protein interaction). Each edge type denotes the interactions between two entities in the heterogeneous graph, computed based on Jaccard similarity. GCN model then could help to learn node representation and predict the drug–target relation.

4.5. Social Media Analysis

Social networks have played an essential role in communication among users worldwide. Various graph embedding models have been applied to social media to learn embeddings [72,216]. In social networks, most graphs are initialized by defining nodes as users and edges describing user relationships (e.g., messages). Several GNNs are applied to help detect fake news shared on social networking platforms [341,342]. Nguyen et al. [343] employed GraphSAGE to classify fake news in social media.

For social interaction network representation, directed graphs can be built with nodes as users and edges describing user social relationships or action interactions [344,345]. GAT model [346] is used to predict the influence of essential users in the social network. Piao et al. [344] proposed a motif-based graph attention network to predict the social relationships between customers and companies. CGNNs [345] could classify relations between political and regular news media users.

4.6. Recommendation Systems

Bipartite graphs could be used to represent user–item interactions in recommendation systems. In the graph, nodes can be presented as users and categories, and directed edges denote interactions between users and items. Several traditional models based on matrix factorization have been applied to help the system understand the predictions of users' ratings on items or click actions [347,348].

The side information is mainly the attributes of categories and users. This information helps to represent the relationship between users and items [349,350]. Heterogeneous graphs with properties of nodes and relationship types have been proposed to represent side information. Several shallow models [351,352] and GNNs [353,354] have been proposed to capture the interaction between users and items with side information.

Knowledge graphs can represent entities and their relationships from the knowledge base. Knowledge graphs, therefore, can collect high-order proximity between items and user interactions [267,355]. Exploiting social correlations such as homophily and social influence can improve the performance of online recommendation systems. Several applications put user and item interaction into CGNNs to learn embeddings and solve collaborative filtering problems [356,357].

4.7. Smart Cities

People encounter current traffic-related issues in big cities, such as traffic jams and difficulty finding parking spaces. Addressing these issues that play an essential role in building smart cities and transportation has been studied in the literature. Traffic forecasting is one of the crucial factors in improving traffic efficiency and solving related problems.

In this context, a graph can be considered a whole city map with nodes as intersections and edges describing paths connecting the nodes [358]. For the traffic prediction problem,

nodes and edges can have properties that describe the traffic state. Besides static graphs, dynamic graphs with dynamic adjacency matrices are also used to describe the dynamic state of overtime traffic. In recent years, GNNs have been widely applied to predict traffic conditions [359–361]. CGNNs are applied to predict traffic flow conditions in big cities. For example, the attention-based GNNs are applied to predict traffic congestion status [359]. The self-attention mechanism can capture the state around the target vehicles by considering connections to its ego network.

Dynamic graphs can represent a spatial-temporal dependency. Several applications are also practical to spatial-temporal transportation networks [358,360,361] to predict traffic flow. A study from [360] applied CGNNs to capture the traffic's current state and historical conditions to predict the next state of the traffic condition. They construct a dynamic graph including a collection of snapshots, and each snapshot is the current state of the traffic. The model then uses temporal convolution layers to learn dynamic node features.

There are several applications of graph embedding for energy-related problems, such as predicting electricity consumption and predicting wind and solar energy through IoT systems [362–364]. For example, in the problem of solar irradiance forecasting, a graph can be presented with nodes being the locations of energy measurements and edges describing the correlation between them according to historical data. By contrast, with wind speed forecasting systems, nodes describe wind farms, and edges represent two nodes as neighbors. For instance, a convolutional graph autoencoder-based model is used to help predict the radiative state of solar energy [362]. Khodayar et al. [363] presented a CGNN model to predict wind speed and direction.

4.8. Computational Social Science

The analysis of social issues and human behavior has been expanded due to the increased availability of big data. The application of computational science has created new opportunities for researchers in social science to achieve more detailed information by examining the trends and patterns of social phenomena.

Graph-based models provide an improved understanding of social issues, ranging from social inequity to the spread of child maltreatment across generations, using data, theory, and diverse media sources. In existing studies, directed acyclic graphs are typically used to represent the research hypotheses about causal relationships among variables based on existing literature [365]. They encode nodes in DAG graphs using color and predicting factors affecting children's psychology.

The graph-based models have also been applied to political problems to explore the phenomena and trends of influence of political populations in social networks [366–368]. For example, the Community2Vec model [369] is used in [366] to identify political populations in a community. They measure the similarity between politically different communities and identify changes and trends in the community.

4.9. Digital Humanity

There is a growing interest in computational narrative analysis in the field of digital humanities. A character graph is one of the essential ways of expressing narratives, representing various relationships formed between characters as the story progresses. There are various methods of constructing a character graph. Typically, they use conversations in the story [370,371], consider events that make up the story [372,373], or are based on the co-occurrence of characters [374,375]. Recently, high-quality distributed representations of characters have been attempted for efficient and easy machine learning of character graphs. Lee and Jung [168] applied a subgraph-based graph embedding model to the dynamic networks of movie characters to compare similarities between stories. Inoue et al. [376] presented GNNs that could help to learn character embedding. If the characters in different works share similar properties, their connection relationships can be represented. Kounelis et al. [377] presented the movie's plot to improve the movie recommendation system's performance using the Graph2Vec model. First, a character relationship graph

containing all necessary information for plot representation was built using the movie script. Graph embedding was then generated from the character relationship graph through the embedding method.

Since the digitization of large-scale literature works enables computer analysis of narratives, character graph embedding can be used in various ways in digital humanities. First, it is easy to measure similarities between stories. Second, since the unique aesthetic characteristics of a specific writer can be identified through machine learning on character graph embedding, it can be used to compare the styles of writers or to develop a story generation system that imitates the writing style of a specific writer. Third, characters can be classified based on their roles and personalities through character graph embedding. Fourth, character graph embedding can play an essential role in improving the computer's narrative understanding in research on the narrative intelligence of computers, which has been attracting significant interest in recent years. Riedl [378] defined narrative intelligence as the ability to create and understand stories and argued that when computers are equipped with narrative intelligence, systems benefit humans, such as human-computer dialog systems can be developed.

4.10. Semiconductor Manufacturing

Recently, graph representation learning models have expanded their field of applications to semiconductor research and development, including semiconductor material screening [379], circuit design [380,381], chip design [382], and semiconductor manufacturing and supply chain management [383,384]. A graph could be constructed from crystal networks with nodes being atoms and edges describing the relation between them. GNNs could help to predict material properties for the fast screening of candidate materials. A tuples graph neural network exhibits an improved generalization capability for unseen data for bandgap prediction in perovskite crystals, 2D material, materials for solar cells, and binary and ternary inorganic compound semiconductors [379].

For circuit [380,381] (or chip [382]) design tasks, a graph could be constructed with nodes being transistors (or macro-cells/blocks) and edges being wires (or routings). A computer chip could be considered to be a hypergraph of circuit components as a netlist graph. Chip designers adopted GNNs to unleash themselves from extensive design space exploration, i.e., running many parallel physical design implementations to achieve the best timing closure [385]. It can be significantly fast and efficient by combining the GNN and LSTM, responsible for netlist encoding and sequential flow modeling [382].

For semiconductor manufacturing tasks, a graph could be constructed as nodes representing an operation of a job on a device and directed edges representing a relation between nodes (e.g., process flow). Graph2Vec model was adopted to learn fab states, which are the processing of lots on machines and transfer between machines and setup and maintenance activities [386].

4.11. Weather Forecasting

Graph-based models have shown great effectiveness in learning correlations of spatial and temporal features for weather prediction tasks. Typically, a graph is built with nodes that describe stations that collect information in different geographical locations, edges that describe the spatial neighbors of the stations, and attributes that describe meteorological variables. Meteorological variables include measurements over a specified time period, such as temperature, humidity, soil moisture, seismic source, etc. Several CGNNs have been proposed to capture spatial relations between different geographical locations [363,387,388]. The models could help to combine with an LSTM model to process temporal time series in solar radiation prediction.

Since the interactions of meteorological variables at different locations could show dynamic behaviors and mutual influence, several graph-based models could help to capture these dynamic influences. For example, Lira et al. [389] proposed spatio-temporal attention-based GNNs to predict frost by capturing the influences between round envi-

ronmental sensors (nodes). GNNs could help to capture the spatial dependency patterns for predicting several weather tasks (e.g., temperature and humidity prediction) [390]. Jeon et al. [391] proposed the MST-GCN model (Multi-attributed Spatio-Temporal GCN) to predict hourly solar irradiance using GCNs to learn the spatio-temporal correlations between meteorological variables (e.g., temperature, wind speed, relative humidity, etc.). A graph could be constructed by considering each station as a node, and edges were defined in two ways: distances between stations and correlations between historical meteorological variables of stations.

For air quality prediction, several graph-based models could help to predict air quality by learning the correlations between air pollution variables (e.g., CO_2, O_3, etc.) and meteorological variables. Since the diffusion of air pollutants is affected by multiple factors (e.g., meteorological conditions, vehicle emissions, and industrial sources), Xiao et al. [392] used CGNNs to help predict the diffusion of $PM_{2.5}$ concentration. A dynamic-directed graph could be constructed by considering nodes as stations, and edges denote the distance of stations that denotes the edges' strength. Several studies [393,394] used a heterogeneous graph to represent the type of each station as a node type and the connection between them as an edge. They then adopt RGNNs to learn spatial and temporal correlations to predict air quality.

Graph-based models could also help to predict surface-related tasks, such as seismic source characterization, seismic wave analysis, and earthquakes [395–397]. A graph could be constructed by nodes as stations and edges are the relationships of nodes if seismic events can occur simultaneously. For example, GNNs could help to estimate earthquake location by leveraging waveform information from multiple stations [397].

Several graph-based models could help predict sea surface temperature (SST), which plays an important role in various ocean-related predictions (e.g., global warming, oceanic environmental protection, and disaster reduction) [398–400]. A graph could be constructed as longitude and latitude grids where nodes are coordinates and edges represent the relationship between nodes. For example, GCNs [401] could help to learn temporal shifts to predict the sea surface temperature.

A graph can be constructed as a hierarchical tree representing different variables' influences on global-scale weather forecasting. Lam et al. [402] transformed the 3D data into a multi-resolution icosahedral network as a mesh hierarchy. GNNs could help to capture long-range spatial interactions for modeling global forecasting systems. Shi et al. [403] designed an adaptive mesh grid based on Voronoi polygons for ocean simulations and used GNNs to investigate environmental parameters for arbitrary visual mapping.

For El Niño-Southern Oscillation (ENSO) prediction and global ocean-atmosphere interaction, graph-based models could help to improve climate prediction tasks. For example, Cachay et al. [404] constructed the climate graph that defines each grid cell as a node, and the edge denotes the similarity between nodes. GNNs could help to capture correlations between spatio-temporal samples to improve the El Niño forecasting task. CGNNs is used to capture interactions of different air-sea coupling strengths in various period of time [405].

5. Evaluation Methods

Since we cannot evaluate the performance of learned graph embedding models, numerous benchmarks have been used to investigate the performance of various models to solve specific downstream tasks. A good graph embedding model should provide vector representations of graph entities that preserve the graph structure and entity relationship. In this section, we first discuss benchmark datasets and then examine typical downstream tasks such as classification, ranking, and regression tasks.

5.1. Benchmark Datasets

The goal of benchmark datasets is the standard for developing, evaluating, and comparing graph representation learning models. Table 18 presents a summary of bench-

mark datasets for graph embedding models. Typically, the benchmark datasets are categorized into four main groups: citation networks, social networks, webpages, and biochemical networks.

Table 18. A summary of benchmark datasets for graph embedding models. # Nodes and # Edges indicate the number of nodes and edges in graphs, respectively.

Dataset	Graph Type	Category	# Nodes	# Edges
Cora [406]	Homogeneous graph	Citation network	2808	5429
Citeseer [407]	Homogeneous graph	Citation network	3312	4732
Reddit [22]	Homogeneous graph	Social network	232,965	114,615,892
PubMed [406]	Homogeneous graph	Citation network	19,717	44,338
Wikipedia [406]	Homogeneous graph	Webpage	2405	17,981
DBLP [408]	Homogeneous graph	Citation network	781,109	4,191,677
BlogCatalog [408]	Homogeneous graph	Social network	10,312	333,983
Flickr[1]	Homogeneous graph	Social network	80,513	5,899,882
Facebook[409]	Homogeneous graph	Social network	4039	88,234
PPI [22]	Homogeneous graph	Biochemical network	56,944	818,716
MUTAG [410]	Homogeneous graph	Biochemical network	27,163	148,100
PROTEIN [411]	Homogeneous graph	Biochemical network	43,500	162,100
Wiki	Homogeneous graph	Webpage	4,780	184,81 K
YouTube	Homogeneous graph	Video streaming	1,130,000	2,99 M
DBLP [412]	Heterogeneous graph	Citation network	Author (A): 4057 Paper (P): 14,328 Term (T): 7723 Venue (V): 20	A-P: 19,645 P-T: 85,810 P-V: 14,328
ACM [412]	Heterogeneous graph	Citation network	Paper (P): 4019 Author (A): 7167 Subject (S): 60	P-P: 9615 P-A: 13,407 P-S: 4019
IMDB [412]	Heterogeneous graph	Movie reviews	Movie (M): 4278 Director (D): 2081 Actor (A): 5257	M-D: 4278 M-A: 12,828
DBIS [413]	Heterogeneous graph	Citation network	Venues (V): 464 Authors (A): 5000 Publication (P): 72,902	-
BlogCatalog3 [414]	Heterogeneous graph	Social network	User: 10,312 Group: 39	348,459
Yelp [415]	Heterogeneous graph	Social media	User: 630,639 Business: 86,810 City: 10 Category: 807	-
U.S. Patents [180]	Heterogeneous graph	Patent, Trademark Office	Patent: 295,145 Inventor: 293,848 Assignee: 31,805 Class: 14	-
UCI [416]	Dynamic graph	Social network	1899	59,835
DNC [416]	Dynamic graph	Social network	2029	39,264
Epinions [417]	Dynamic graph	Social media	6224	19496
Hep-th [418]	Dynamic graph	Citation network	34,000	421,000
Auto Systems [419]	Dynamic graph	BGP logs	6000	13,000
Enron	Dynamic graph	Email network	87,000	1,100,000
StackOverflow [420]	Dynamic graph	Question&Answer	14,000	195,000
dblp [408]	Dynamic graph	Citation network	90,000	749,000
Darpa [421]	Dynamic graph	Computer network	12,000	22,000

Citation networks depict a network of documents linked together in a particular manner. The citation graph could be constructed by considering each node as a document, and each edge of two nodes describes the citation. Since citations are directed from a source document to a destination document, citation graphs usually are directed graphs. Since the labels of the citation network could represent document topics, there

are several downstream tasks for citation network analysis, such as link prediction and node classification.

The social networking datasets describe the connections between users on social networking sites such as Facebook [409], Twitter, or blog forums [422]. An online social network describes the links between users or groups, usually through the link of adding friends. In addition, the user properties could also be included in the graphs. Due to privacy policies, several user information could be hidden in social networks. Therefore, there are several downstream tasks for social network analysis, such as missing node classification and link prediction.

Webpage datasets are a term used to refer to a collection of webpages of information organized and linked together to represent information such as text and images. A webpage can be an article, a category, or any information page. For instance, Wikipedia dataset [406] in Table 18 is a directed network with 2405 nodes and 17,981 edges linking nodes. There are several downstream tasks for webpage analysis, such as node classification and link prediction.

Biochemical networks are data sources containing information in the field of biochemistry area. Several downstream tasks are used for the biochemical networks, such as predicting the composition of cancer classification proteins [423] or drug–drug interaction prediction. Protein dataset [411], for example, are biochemical graph sets with 1113 graphs. The protein dataset includes more than 435,000 nodes and 1,621,000 links between nodes.

5.2. Downstream Tasks and Evaluation Metrics

After the models learn vector embeddings, various downstream tasks can benefit from such embeddings, such as classification tasks, regression tasks, and prediction tasks. Therefore, we first discuss the downstream tasks and then examine the standard evaluation metrics for each task.

The classification problem denotes the graph entities classification tasks, including node classification, edge classification, subgraph classification, and graph classification. There are also link prediction tasks that can be considered to be classification problems where the output is discrete. The goal of classification tasks is to predict the classes of unlabeled graph entities given a set of labeled entities. For example, in the Cora citation network, the task of node classification is to classify publications grouped into seven main classes that correspond to the research area. Several evaluation metrics could be used for classification tasks, such as Accuracy (A), Precision (P), Recall (R), and F_β score.

Consider a dataset consisting n multi-label examples $\mathbb{D} = \{x_i, Y_i\}$ where $1 \leq i \leq n$ and $Y_i = \{0,1\}^m$ with a labelset $\mathbb{L}$: $|\mathbb{L}| = m$. Let C be a multi-label classifier and $\hat{Y}_i = C(x_i) = \{0,1\}^m$ denotes the set of the label for the classification of the sample x_i. Accuracy measures the number of correct classifications over all the number (predicted and actual) of labels for that instance. The higher the accuracy, the more accurate the models. The precision metric P is measured as the ratio of predicted correct labels to the total number of actual labels. The Recall metric R is measured as the ratio of predicted correct labels to the total number of predicted labels. In several classification tasks, where both Precision and Recall metrics are important in the model evaluation, a common metric that combines both Recall and Precision is called F_β-score. Mathematically, the Accuracy (A), Precision (P), Recall (R), and F_β score for all instances could be computed as:

$$A = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap \hat{Y}_i|}{|Y_i \cup \hat{Y}_i|}, P = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap \hat{Y}_i|}{|\hat{Y}_i|}, R = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap \hat{Y}_i|}{|Y_i|}, F_\beta = (1+\beta)^2\frac{PR}{\beta^2 P + R}, \quad (176)$$

where β denotes a positive factor to change the impact between P score and R score. Besides measuring based on samples, we could measure the performance based on label evaluation. This could be beneficial when the number of labels is large, and it is challenging to compute a performance snapshot. Therefore, we can compute the score in each class label first

and then average over all classes (macro averaging) or across all the classes and samples (micro averaging).

Several regression metrics could be used for rating prediction in recommendation systems to evaluate the user–item interaction pairs [424,425]. Similar to graph classification, graph regression problems aim to predict the labels of entities in a graph by learning neighbor node labels. However, the difference between classification and regression problems is that the metrics for regression problems are explained in error, which measures the difference between predicted and actual labels. Another metric that is widely used for measuring the performance of regression models is the Coefficient of Discrimination (R^2). R^2 measures the ratio between the unexplained variations over total variations. The standard metrics, which are Mean Square Error (*MSE*), Root Mean Square Error (*RMSE*), and Mean Absolute Error (*MAE*), and R^2 could be computed as:

$$MSE = \frac{\left\|Y_i - \hat{Y}_i\right\|_2^2}{N}, MAE = \sum_{i=1}^{n} \left|Y_i - \hat{Y}_i\right|, \tag{177}$$

$$RMSE = \sqrt{\frac{\left\|Y_i - \hat{Y}_i\right\|_2^2}{N}}, R^2 = 1 - \frac{\left\|Y_i - \hat{Y}_i\right\|_2^2}{\left\|Y_i - \bar{Y}\right\|_2^2}, \tag{178}$$

where $\bar{Y}$ denotes the mean of the dependent variable in the dataset.

In graph ranking tasks, the models try to predict the rank (or relevance index) of a list of items for a particular task. The models can learn the order of the predicted labels for multi-label classification problems where each sample has more than one label. For example, in the case of most recommendation systems, a user could have more than one preference. Several commonly used metrics evaluate model performance for the raking problems, including Mean Reciprocal Rank (*MRR*), *P@k*, *MAP@k*, and *R@k*.

The Mean Reciprocal Rank (*MRR*) metric is one of the simplest metrics in evaluating ranking models. The MRR metric calculates the average of the corresponding terms of the first related item for a set of queries Q, which can be defined as:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{Q} \frac{1}{rank_i}. \tag{179}$$

One of the limitations of the MRR metric is that it only counts from the first item to the rank of actual labels in the query list. Precision at k (*P@k*) is a metric that could compute the proportion of the number of the first k predicted labels in the actual labelset over the k. The predicted label order is not taken into account in the *P@k* metric. Similar to the *P@k* metric, *Recall@k* is a metric that computes the proportion of the number of the first k predicted labels in the actual labelset over all relevant items.

$$P@k = \frac{\left|\{Y_i\} \cap \{\hat{Y}_i[:k]\}\right|}{\left|\{\hat{Y}_i[:k]\}\right|} \; R@k = \frac{\left|\{Y_i\} \cap \{\hat{Y}_i[:k]\}\right|}{\left|\{\hat{Y}_i\}\right|}. \tag{180}$$

Mean Average Precision (*MAP@k*) can be applied to the entire dataset because of the stability in ranking the labels. Compared to *P@k*, MAP focuses more on how many predicted labels are in the actual labelset, where the order of predicted labels is taken into account. Mathematically, *MAP@k* is the average across all instances, which could be calculated as:

$$MAP = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{K} \sum_{k=1}^{K} P@k \times rel(k) \tag{181}$$

where $rel(k)$ denotes the relevance at k for each sample.

5.3. Libraries for Graph Representation Learning

Several libraries provide state-of-the-art graph representation learning models which have a variety of sampling strategies and downstream tasks. To ease researchers to develop graph representation learning models, this section introduces a collection of libraries, which are summarized in Table 19.

Table 19. A summary of libraries of graph embeddings models. The accessibility of URLs for open-source repositories of the libraries have been checked on 16 April 2023.

Library	URL	Platform	Model
PyTorch Geometric (PyG) [426]	https://github.com/pyg-team/pytorch_geometric	PyTorch	Various GNN models and basic graph deep-learning operations
Deep Graph Library (DGL) [427]	https://github.com/dmlc/dgl	TensorFlow, PyTorch	Various GNN models and basic graph deep-learning operations
OpenNE	https://github.com/thunlp/OpenNE/tree/pytorch	TensorFlow, PyTorch	Shallow models: DeepWalk, Node2Vec, GAE, VGAE, LINE, TADW, SDNE, HOPE, GraRep, GCN
CogDL [428]	https://github.com/THUDM/cogdl	TensorFlow, PyTorch	Various GNN models
Dive into Graphs (DIG) [429]	https://github.com/divelab/DIG	PyTorch	Various GNN models and research-oriented studies (Graph generation, Self-supervised learning (SSL), explainability, 3D graphs, and graph out-of-distribution).
Graphvite [430]	https://github.com/deepgraphlearning/graphvite	Python	DeepWalk, LINE, Node2Vec, TransE, RotatE, and LargeVis.
GraphLearn [431]	https://github.com/alibaba/graph-learn	Python	Various GNN models, the framework can support the sampling batch graphs or offline training process.
Connector	https://github.com/NSLab-CUK/connector	Pytorch	Various shallow models and GNN models.

PyTorch Geometric (PyG) [426] is a graph neural network framework based on PyTorch. PyG can handle and process large-scale graph data, multi-GPU training, multiple classic graph neural network models, and multiple commonly used graph neural network training datasets. PyG already contains numerous benchmark datasets, including Cora, Citeseer, etc. It is also effortless to initialize such a dataset, which will automatically download the corresponding dataset and process it into the required format for various GNNs. Furthermore, many real-world datasets are stored as heterogeneous graphs, which prompted the introduction of specialized functions in PyG.

Deep Graph Library (DGL) [427] is an easy-to-use, high-performance, scalable Python package for building graph representation learning models. DGL has better memory management for GNNs that can be expressed as sparse matrix multiplication. Therefore, the DGL library provides flexible, efficient strategies for building new GNN layers. Furthermore, DGL has a programming interface for flexible applications, which helps researchers understand the process of designing GNNs for large graphs.

OpenNE is a standard Network Representation Learning framework that enables graph embedding models with multi-GPU training. Most of the graph embedding models in OpenNE framework are matrix factorization-based and shallow models, including DeepWalk, LINE, Node2Vec, GraRep, TADW, GCN, HOPE, GF, and SDNE. Furthermore, the framework could also provide dimension-reduction techniques, such as t-SNE and PCA, for visualization.

Developed by Tsinghua University, CogDL [428] framework could integrate various downstream tasks and match evaluation methods. Therefore, the framework could help researchers efficiently run the results of various baseline models and develop new graph embedding models. Furthermore, the framework could integrate algorithms task-oriented and assigns each algorithm to one or more tasks. In addition, CogDL also supports

researchers in customizing models and datasets and is embedded in the overall framework of CogDL to help them improve efficiency.

For complex downstream tasks, such as graph generation and graph neural network interpretability, DIG [429] provides APIs for data interfaces, commonly used algorithms and evaluation standards. DIG is designed to make it easy for researchers to develop algorithms and conduct experimental comparisons with benchmark models. The framework could help researchers solve tasks, including graph generation, graph self-supervised learning, graph neural network interpretability, and 3D graph deep-learning tasks.

GraphVite [430] is a general-purpose graph embedding framework to help researchers learn embeddings with high speed and large scale. One of the advantages of the framework is that GraphVite can support multi-GPU parallelism. Therefore, the framework could quickly handle large-scale graphs with millions of nodes and learn the node representation. GraphVite provides complete training and evaluation for various types of graphs, including homogeneous and knowledge graphs.

GraphLearn [431] is a graph learning framework designed to develop and apply large-scale GNN models in practical situations. The framework could help researchers parallel negative sampling from industrial application scenarios to speed up training. Therefore, the framework could implement sampling optimization, sparse scene model optimization, and GPU acceleration for PyTorch. As a result, GraphLearn has been successfully applied in Alibaba and several scenarios, such as recommendation systems and security risks.

Another library for graph representation learning is Connector which can help researchers develop new graph embedding models efficiently. The framework provides various widespread graph representation learning models, such as matrix factorization-based, shallow, and GNN models. Furthermore, Connector can analyze various types of graphs, ranging from homogeneous and heterogeneous graphs to knowledge graphs with different sampling processes. Therefore, Connector could help researchers efficiently construct various baseline models and design new graph embedding models.

6. Challenges and Future Research Directions

Graph representation learning models have gained significant results recently, showing the model's power and practical applications in the real world. However, there are still several challenges for existing models since graph data are complicated (e.g., nodes are disordered and have a complex relationship). Therefore, this section presents challenges and promising directions for future research. The main challenges and future research directions of graph embedding models are summarized as follows:

- Graph representation in a suitable geometric space: Euclidean space may not capture the graph structure sufficiently and lead to structural information loss.
- The trade-off between the graph structure and node features: Most graph embedding models suffer from noise from non-useful neighbor node features. This could lead to a trade-off between structure preservation and node feature representation, which can be the future research direction.
- Dynamic graphs: Many real-world graphs show dynamic behaviors representing entities' dynamic structure and properties, bringing a potential research direction.
- Over-smoothing problem: Most GNN models suffer from this problem. The graph transformer model could only handle the over-smoothing problem in several cases.
- Disassortative graphs: Most graph representation learning models suffer from this problem. Several solutions have been proposed but have yet to fully solve to the whole extent.
- Pre-trained models: Pre-trained models could be beneficial to handle the little availability of node labels. However, a few graph embedding models have been pre-trained on specific tasks and small domains.

The performance of graph embedding models is determined by how well the geometric space for graph representation matches the graph structure [292]. Therefore, choosing a suitable geometric space to represent the graph structure is a crucial step in building

efficient graph representation learning models. Most existing graph embedding models represent the graph structure in Euclidean space, which defines the similarity between entities by the inner product, Euclidean distance, and so on. However, representing the graph structure in Euclidean space may not capture the graph structure sufficiently and lead to structural information loss [432]. For example, models in Euclidean space fail to represent adequate tree-like graph data where the nodes grow exponentially and follow the power law. In the case of webpage networks with millions of nodes, there are a few important websites that are hubs and dominate the network, while most other websites have few connections, which leads to most existing models in the Euclidean space failing to learn embeddings. Recently, several studies have been trying to represent graph data in the non-Euclidean space, and the results are relatively promising [69,103,432]. Nevertheless, it still needs to be resolved whether representing graph data in non-Euclidean space is more efficient and significantly improves accuracy. One major issue is the choice of suitable isometric models, and the reasons why and when to use the models are still an open question that existing models have yet to analyze to a whole extent [294]. Another problem is that developing operators and optimization in the non-Euclidean space for deep neural networks is challenging. Most existing models aim to approximate graph data in a tangent space where we are familiar with Euclidean operators. However, several studies presented that tangent space approximation could negatively influence the training phase [293,433]. Therefore, developing operators, manifold space, and optimization for various embedding models are significant problems for implementing models in non-Euclidean space.

A good graph representation learning model should preserve the graph structure and represent appropriate features for nodes in graphs. This inspires many shallow models to explore various substructures of graph data (e.g., random walk [4,14], *k*-hop distance [16], motifs [87,89–91], subgraphs [145], graphlets [88], and roles [21]). Several of these sampling strategies ignore the substructures surrounding target nodes [4,14,16], while others omit the node features which could also carry significant information [145]. Recently, models based on message-passing mechanisms effectively capture graph structures and represent node feature embeddings. The message-passing could suffer from noise coming from non-useful neighbor node features, which cause a barrier to the downstream tasks and eventually reduce the performance of models. There are several studies have been proposed to overcome weaknesses of message-passing, such as structural identity [60], and dropout [434, 435]. However, collecting sufficient structural topology and a trade-off between structure preservation and node feature representation still needs to be explored to a full extent.

Most existing graph embedding models work with static graphs where the graph structure and entity properties do not change over time [4,14]. However, in the real world, graphs are dynamic, consisting of both graph structure and properties that evolve over time [10,82]. There are several dynamic behaviors of graph evolution, including topological evolution (the set of nodes and edges change over time), feature evolution (the node and edge feature or its label changes over time), degree distribution, and the node role changes over time. However, most existing models only aim to find out which patterns of evolution should be captured and represented that do not represent fully dynamic behaviors in general [10,112]. For example, in the case of social networks, users could change personal attributes such as hometown, occupation, and their role in a specific small group over time. This leads to how models can represent the dynamic structure and properties of entities bringing a potential research direction.

Graph neural networks have shown significant advantages in working with large-scale graphs for specific tasks. However, these existing models still have limitations regarding the over-smoothing problem when stacking more GNN layers. Recently, several works have attempted to handle the over-smoothing problem, such as adding initial residual connection [28], using dropout [436], and PageRank [437]. However, most of them need to be effectively adaptable to a wide and diverse scope of various graph structures. Several graph transformer models have been proposed in recent years to overcome the limitation of the message-passing mechanism by self-attention [63,438]. However, the self-attention

mechanism considers input graphs as fully connected graphs that have yet to entirely solve the over-smoothing problems, especially in small and sparse graphs [61]. Therefore, building a deep-learning model to address the over-smoothing problem is still an open question and a promising research direction.

Another challenge for graph embedding models is the problem of working with disassortative graphs for various downstream tasks, especially classification tasks. Disassortative graphs are graphs where pairs of nodes with different labels tend to be connected. For example, in the case of amino acid networks, amino acids with different labels tend to be connected by peptide bonds [439]. Looking back at the sampling mechanism of GNNs and graph transformer models, the target nodes update the vector embeddings based on the k-hop neighbor features [24,310]. This is a problem for classification tasks where the aggregation mechanisms assume that interconnected nodes should have the same label, which is completely different from the disassortative graph structure. Several methods have been proposed in recent years to overcome classification problems for disassortative graphs [58,440]. However, the message-passing-based mechanisms are still a problem and challenge when working with disassortative graphs.

Another problem in challenging deep-learning models is to pre-train the graph embedding models and then fine-tune the models on various downstream tasks. Most current models are designed independently to be suitable for some specific tasks that have yet to be generalized, even with graphs in the same domain [8]. Although several graph transformer models have been pre-trained on related tasks, the transfer of the models across other tasks is still limited in a few specific graph data [30,63]. This leads to the problem that the models must train from scratch when we have new graph data and other tasks, which is time-consuming and limits practical applicability. The pre-trained models are also beneficial to handle the little availability of node labels. Therefore, if the graph embedding models are pre-trained, they could be transferred and used to handle new tasks.

7. Conclusions

This paper has presented a comprehensive view of graph representation learning. Specifically, most models have been discussed, ranging from traditional models, such as graph kernels and matrix factorization models, to deep-learning models with various graphs. One of the most thriving models is the GNN with the power of an aggregation mechanism in learning the local and global structures of the graph. The achievements of GNN-based models have been seen in various real-world tasks with large-scale graphs. Recently, graph transformer models have shown promising results in applying self-attention to learn embeddings. However, the self-attention mechanism need also be improved to solve the over-smoothing problem to a whole extent.

Practical applications in various fields are also presented, showing the contribution of graph representation learning to society and related areas. Our paper not only shows the applications of graph embedding models but also describes how a graph is initialized in each specific domain and the application of the graph embedding model to each application. In addition, evaluation metrics and downstream tasks were also discussed to understand more about graph embedding models. Although deep graph embedding models have shown great success in recent years, they still have several limitations. The balance between the graph structure and the node features is still challenging for deep graph embedding models in various downstream tasks. Our paper also points out the current challenges and future directions of promising research.

Author Contributions: Conceptualization, V.T.H. and O.-J.L.; Methodology, V.T.H., H.-J.J., E.-S.Y., Y.Y., S.J., and O.-J.L.; Writing—original draft, V.T.H., H.-J.J., E.-S.Y., Y.Y., and S.J.; Writing—review and editing, V.T.H. and O.-J.L.; Project administration, O.-J.L.; Supervision, O.-J.L.; Funding acquisition, H.-J.J., S.J., and O.-J.L.; All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2022R1F1A1065516) (O.-J.L.), in part by the

Research Fund, 2022 of The Catholic University of Korea (M-2022-B0008-00153) (O.-J.L.), in part by the R&D project "Development of a Next-Generation Data Assimilation System by the Korea Institute of Atmospheric Prediction System (KIAPS)," funded by the Korea Meteorological Administration (KMA2020-02211) (H.-J.J.), in part by the National Research Foundation of Korea (NRF) under grant NRF-2022M3F3A2A01076569 (S.J.), and in part by the Advanced Institute of Convergence Technology under grant (AICT-2022-0015) (S.J.).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Open-Source Implementations

We deliver a summary of open-source implementations of graph embedding models described in Section 3. Table A1 provides open-source implementations of graph kernels (Section 3.1), matrix factorization-based (Section 3.2), and shallow models (Section 3.3). Table A2 provides open-source implementations of deep-learning-empowered models (Section 3.4). Table A3 presents open-source implementations of non-Euclidean models (Section 3.5).

Table A1. A summary of open-source implementations of graph kernels, matrix factorization-based, and shallow models, which are introduced in Section 3.1, Section 3.2, and Section 3.3, respectively. The accessibility of URLs for the open-source implementations have been checked on 16 April 2023.

Model	Category	URL
[84]	Graph kernels	https://github.com/BorgwardtLab/WWL
[111]	Graph kernels	https://github.com/hasanmdal/GraTFEL-Source
[113]	Graph kernels	https://github.com/ferencberes/online-node2vec
[36]	Graph kernels	https://github.com/yeweiysh/MSPG
[35]	Graph kernels	https://github.com/haidnguyen0909/weightedWWL
[34]	Graph kernels	https://github.com/chrsmrrs/glocalwl
[7]	Matrix factorization-based models	https://github.com/andompesta/ComE.git
GLEE [122]	Matrix factorization-based models	https://github.com/DefuLian/lightne
GraRep [15]	Matrix factorization-based models	https://github.com/ShelsonCao/GraRep
HOPE [5]	Matrix factorization-based models	https://github.com/ZW-ZHANG/HOPE
ProNE [43]	Matrix factorization-based models	https://github.com/THUDM/ProNE
TADW [131]	matrix factorization-based models	https://github.com/thunlp/tadw
PME [138]	matrix factorization-based models	https://github.com/TimDettmers/ConvE
DeepWalk [14]	Shallow models	https://github.com/phanein/deepwalk
Node2vec [4]	Shallow models	https://github.com/aditya-grover/node2vec
Node2Vec+ [148]	Shallow models	https://github.com/krishnanlab/node2vecplus_benchmarks
Struct2Vec [21]	Shallow models	https://github.com/leoribeiro/struc2vec
Gat2Vec [154]	Shallow models	https://github.com/snash4/GAT2VEC
NME [160]	Shallow models	https://github.com/HKUST-KnowComp/MNE
[162]	Shallow models	http://www3.ntu.edu.sg/home/aspatra/research/Yongjin_BI2010.zip
[166]	Shallow models	https://github.com/higd963/Multi-resolution-Network-Embedding
EvoNRL [170]	Shallow models	https://github.com/farzana0/EvoNRL
STWalk [172]	Shallow models	https://github.com/supriya-pandhre/STWalk
[173]	Shallow models	https://github.com/urielsinger/tNodeEmbed
LINE [16]	Shallow models	https://github.com/tangjianpku/LINE
DNGR [196]	Shallow models	https://github.com/ShelsonCao/DNGR
TriDNR [441]	Shallow models	https://github.com/shiruipan/TriDNR
[188]	Shallow models	https://github.com/fuguoji/event2vec

Table A2. A summary of open-source implementations of deep-learning-empowered graph embedding models discussed in Section 3.4. The accessibility of URLs for the open-source implementations have been checked on 16 April 2023.

Model	Category	URL
SDNE [50]	Graph autoencoder	http://nrl.thumedia.org/structural-deep-network-embedding
[57]	Graph autoencoder	https://github.com/minkky/Graph-Embedding
Topo-LSTM [49]	Graph autoencoder	https://github.com/vwz/topolstm
GCN [18]	Spectral GNNs	https://github.com/tkipf/gcn
[211]	Spectral GNNs	https://github.com/ZhuangCY/Coding-NN
[214]	Spectral GNNs	https://github.com/Tiiiger/SGC
FastGCN [52]	Spatial GNNs	https://github.com/matenure/FastGCN
GraphSAINT [53]	Spatial GNNs	https://github.com/GraphSAINT/GraphSAINT
Hi-GCN [54]	Spatial GNNs	https://github.com/haojiang1/hi-GCN
GIN [24]	Spatial GNNs	https://github.com/weihua916/powerful-gnns
ST-GDN [220]	Spatial GNNs	https://github.com/jill001/ST-GDN
SACNNs [223]	Spatial GNNs	https://github.com/vector-1127/SACNNs
[227]	Spatial GNNs	https://github.com/rdevon/DIM
[229]	Spatial GNNs	https://github.com/dinhinfotech/PGC-DGCNN
PHC-GNNs [233]	Spatial GNNs	https://github.com/bayer-science-for-a-better-life/phc-gnn
Dyn-GRCNN [236]	Spatial GNNs	https://github.com/RingBDStack/GCNN-In-Traffic
DMGI [243]	Spatial GNNs	https://github.com/pcy1302/DMGI
EvolveGCN [245]	Spatial GNNs	https://github.com/IBM/EvolveGCN
GAT [19]	Attentive GNNs	https://github.com/PetarV-/GAT
GATv2 [58]	Attentive GNNs	https://github.com/tech-srl/how_attentive_are_gats
SuperGAT [258]	Attentive GNNs	https://github.com/dongkwan-kim/SuperGAT
GraphStar [256]	Attentive GNNs	https://github.com/graph-star-team/graph_star
HAN [25]	Attentive GNNs	https://github.com/Jhy1993/HAN
[260]	Attentive GNNs	https://github.com/AvigdorZ/ADaptive-Structural-Fingerprint
DualHGCN [264]	Attentive GNNs	https://github.com/xuehansheng/DualHGCN
MHGCN [266]	Attentive GNNs	https://github.com/NSSSJSS/MHGCN
[334]	Attentive GNNs	https://github.com/Blair1213/DDKG
Graformer [47]	Graph transformer	https://github.com/mnschmit/graformer
Graph-Bert [63]	Graph transformer	https://github.com/jwzhanggy/Graph-Bert
EGT [30]	Graph transformer	https://github.com/shamim-hussain/egt
UGformer [61]	Graph transformer	https://github.com/daiquocnguyen/Graph-Transformer
Graphormer [67]	Graph transformer	https://github.com/microsoft/MeshGraphormer
Yao *et al.* [101]	Graph transformer	https://github.com/QAQ-v/HetGT
[282]	Graph transformer	https://github.com/jcyk/gtos
[29]	Graph transformer	https://github.com/graphdeeplearning/graphtransformer
SAN [284]	Graph transformer	https://github.com/DevinKreuzer/SAN
HGT [285]	Graph transformer	https://github.com/UCLA-DM/pyHGT
NI-CTR [287]	Graph transformer	https://github.com/qwerfdsaplking/F2R-HMT

Table A3. A summary of open-source implementations of non-Euclidean graph embedding models, which are described in Section 3.5. The accessibility of URLs for the open-source implementations have been checked on 16 April 2023.

Model	Category	URL
[70]	Hyperbolic space	https://github.com/facebookresearch/poincare-embeddings
[293]	Hyperbolic space	http://snap.stanford.edu/hgcn/
[294]	Hyperbolic space	https://github.com/facebookresearch/hgnn
Graph2Gauss [23]	Gaussian embedding	https://github.com/abojchevski/graph2gauss

References

1. Rossi, R.A.; Ahmed, N.K. The Network Data Repository with Interactive Graph Analytics and Visualization. In Proceedings of the 29th Conference on Artificial Intelligence (AAAI 2015), Austin, TX, USA, 25–30 January 2015; AAAI Press: Austin, TX, USA, 2015; pp. 4292–4293.
2. Ahmed, Z.; Zeeshan, S.; Dandekar, T. Mining biomedical images towards valuable information retrieval in biomedical and life sciences. *Database J. Biol. Databases Curation* **2016**, *2016*, baw118. https://doi.org/10.1093/database/baw118.
3. Hamilton, W.L. Graph Representation Learning. In *Synthesis Lectures on Artificial Intelligence and Machine Learning*; Springer: Berlin/Heidelberg, Germany, 2020. https://doi.org/10.2200/S01045ED1V01Y202009AIM046.
4. Grover, A.; Leskovec, J. node2vec: Scalable Feature Learning for Networks. In Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining (SIGKDD 2016), San Francisco, CA, USA, 13–17 August 2016; ACM: San Francisco, CA, USA, 2016; pp. 855–864. https://doi.org/10.1145/2939672.2939754.
5. Ou, M.; Cui, P.; Pei, J.; Zhang, Z.; Zhu, W. Asymmetric Transitivity Preserving Graph Embedding. In Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD 2016), San Francisco, CA, USA, 13–17 August 2016; ACM: San Francisco, CA, USA, 2016; pp. 1105–1114. https://doi.org/10.1145/2939672.2939751.
6. Sun, J.; Bandyopadhyay, B.; Bashizade, A.; Liang, J.; Sadayappan, P.; Parthasarathy, S. ATP: Directed Graph Embedding with Asymmetric Transitivity Preservation. In Proceedings of the 33th Conference on Artificial Intelligence (AAAI 2019), Honolulu, Hawaii, USA, 27 January–1 February 2019; AAAI Press: Honolulu, HI, USA, 2019; https://doi.org/10.1609/aaai.v33i01.3301265.
7. Cavallari, S.; Zheng, V.W.; Cai, H.; Chang, K.C.; Cambria, E. Learning Community Embedding with Community Detection and Node Embedding on Graphs. In Proceedings of the 26th Conference on Information and Knowledge Management (CIKM 2017), Singapore, 6–10 November 2017; ACM: Singapore, 2017; pp. 377–386. https://doi.org/10.1145/3132847.3132925.
8. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Yu, P.S. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Networks Learn. Syst.* **2021**, *32*, 4–24. https://doi.org/10.1109/TNNLS.2020.2978386.
9. Zhou, Y.; Liu, W.; Pei, Y.; Wang, L.; Zha, D.; Fu, T. Dynamic Network Embedding by Semantic Evolution. In Proceedings of the International Joint Conference on Neural Networks (IJCNN 2019), Budapest, Hungary, 14–19 July 2019; IEEE: Budapest, Hungary, 2019; pp. 1–8. https://doi.org/10.1109/IJCNN.2019.8852247.
10. Mitrovic, S.; De Weerdt, J. Dyn2Vec: Exploiting dynamic behaviour using difference networks-based node embeddings for classification. In Proceedings of the 15th International Conference on Data Science (ICDATA 2019), Las Vegas, NV, USA, 29 July–1 August 2019; CSREA Press: Las Vegas, NV, USA, 2019; pp. 194–200.
11. Lee, J.B.; Rossi, R.A.; Kim, S.; Ahmed, N.K.; Koh, E. Attention Models in Graphs: A Survey. *ACM Trans. Knowl. Discov. Data* **2019**, *13*, 62.1–62.25. https://doi.org/10.1145/3363574.
12. Xia, F.; Sun, K.; Yu, S.; Aziz, A.; Wan, L.; Pan, S.; Liu, H. Graph Learning: A Survey. *IEEE Trans. Artif. Intell.* **2021**, *2*, 109–127. https://doi.org/10.1109/TAI.2021.3076021.
13. Chen, F.; Wang, Y.C.; Wang, B.; Kuo, C.C. Graph representation learning: A survey. *APSIPA Trans. Signal Inf. Process.* **2020**, *9*, e15. https://doi.org/10.1017/ATSIP.2020.13.
14. Perozzi, B.; Al-Rfou, R.; Skiena, S. DeepWalk: online learning of social representations. In Proceedings of the 20th International Conference on Knowledge Discovery and Data Mining (KDD 2014), New York, NY, USA, 25–30 January 2015; ACM: New York, NY, USA, 2015; pp. 701–710. https://doi.org/10.1145/2623330.2623732.
15. Cao, S.; Lu, W.; Xu, Q. GraRep: Learning Graph Representations with Global Structural Information. In Proceedings of the 24th International Conference on Information and Knowledge Management (CIKM 2015), Melbourne, VIC, Australia, 19–23 October 2015; ACM: Melbourne, VIC, Australia, 2015; pp. 891–900. https://doi.org/10.1145/2806416.2806512.
16. Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; Mei, Q. LINE: Large-scale Information Network Embedding. In Proceedings of the 24th International Conference on World Wide Web (WWW 2015), Florence, Italy, 18–22 May 2015; ACM: Florence, Italy, 2015; pp. 1067–1077. https://doi.org/10.1145/2736277.2741093.
17. Li, Y.; Tarlow, D.; Zemel, R.S. Gated Graph Sequence Neural Networks. In Proceedings of the 4th International Conference on Learning Representations (ICLR 2016), San Juan, Puerto Rico, USA, 2–4 May 2016; ICLR: San Juan, Puerto Rico, USA, 2016.
18. Kipf, T.N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In Proceedings of the 5th International Conference on Learning Representations (ICLR 2017), Toulon, France, 24–26 April 2017; OpenReview.net: Toulon, France, 2017.
19. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; Bengio, Y. Graph Attention Networks. *arXiv* **2017**, arXiv:1710.10903. https://doi.org/10.48550/ARXIV.1710.10903.
20. Dong, Y.; Chawla, N.V.; Swami, A. metapath2vec: Scalable Representation Learning for Heterogeneous Networks. In Proceedings of the 23rd International Conference on Knowledge Discovery and Data Mining (SIGKDD 2017), Halifax, NS, Canada, 13–17 August 2017; ACM: Halifax, NS, Canada, 2017; pp. 135–144. https://doi.org/10.1145/3097983.3098036.
21. Ribeiro, L.F.R.; Saverese, P.H.P.; Figueiredo, D.R. *struc2vec*: Learning Node Representations from Structural Identity. In Proceedings of the 23rd International Conference on Knowledge Discovery and Data Mining (KDD 2017), Halifax, NS, Canada, 13–17 August 2017; ACM: Halifax, NS, Canada, 2017; pp. 385–394. https://doi.org/10.1145/3097983.3098061.

22. Hamilton, W.L.; Ying, Z.; Leskovec, J. Inductive Representation Learning on Large Graphs. In Proceedings of the 30th Annual Conference on Neural Information Processing Systems (NeurIPS 2017), Long Beach, CA, USA, 4–9 December 2017; Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R., Eds.; NeurIPS: Long Beach, CA, USA, 2017; pp. 1024–1034.
23. Bojchevski, A.; Günnemann, S. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In Proceedings of the 6th International Conference on Learning Representations (ICLR 2018), Vancouver, BC, Canada, 30 April–3 May 2018; OpenReview.net: Vancouver, BC, Canada, 2018.
24. Xu, K.; Hu, W.; Leskovec, J.; Jegelka, S. How Powerful are Graph Neural Networks? In Proceedings of the 7th International Conference on Learning Representations (ICLR 2019), New Orleans, LA, USA, 6–9 May 2019; OpenReview.net: New Orleans, LA, USA, 2019.
25. Wang, X.; Ji, H.; Shi, C.; Wang, B.; Ye, Y.; Cui, P.; Yu, P.S. Heterogeneous Graph Attention Network. In Proceedings of the World Wide Web Conference (WWW 2019), San Francisco, CA, USA, 13–17 May 2019; ACM: San Francisco, CA, USA, 2019; pp. 2022–2032. https://doi.org/10.1145/3308558.3313562.
26. Velickovic, P.; Fedus, W.; Hamilton, W.L.; Liò, P.; Bengio, Y.; Hjelm, R.D. Deep Graph Infomax. In Proceedings of the 7th International Conference on Learning Representations (ICLR 2019), New Orleans, LA, USA, 6–9 May 2019; OpenReview.net: New Orleans, LA, USA, 2019.
27. Feng, Y.; You, H.; Zhang, Z.; Ji, R.; Gao, Y. Hypergraph Neural Networks. In Proceedings of The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI 2019), Honolulu, HI, USA, 27 January–1 February 2019; AAAI Press: Honolulu, HI, USA, 2019; pp. 3558–3565. https://doi.org/10.1609/aaai.v33i01.33013558.
28. Chen, M.; Wei, Z.; Huang, Z.; Ding, B.; Li, Y. Simple and Deep Graph Convolutional Networks. In Proceedings of the 37th International Conference on Machine Learning (ICML 2020), Virtual Event, 13–18 July 2020; PMLR: Virtual Event, 2020; Volume 119; pp. 1725–1735.
29. Dwivedi, V.P.; Bresson, X. A Generalization of Transformer Networks to Graphs. *CoRR* **2020**, arXiv:2012.09699. http://arxiv.org/abs/2012.09699.
30. Hussain, M.S.; Zaki, M.J.; Subramanian, D. Global Self-Attention as a Replacement for Graph Convolution. In Proceedings of the 28th Conference on Knowledge Discovery and Data Mining (KDD 2022), Washington, DC, USA, 14–18 August 2022; ACM: Washington, DC, USA, 2022; pp. 655–665. https://doi.org/10.1145/3534678.3539296.
31. Weisfeiler, B.; Leman, A. The reduction of a graph to canonical form and the algebra which appears therein. *NTI Ser.* **1968**, *2*, 12–16.
32. Nikolentzos, G.; Siglidis, G.; Vazirgiannis, M. Graph Kernels: A Survey. *J. Artif. Intell. Res.* **2021**, *72*, 943–1027. https://doi.org/10.1613/jair.1.13225.
33. Shervashidze, N.; Schweitzer, P.; van Leeuwen, E.J.; Mehlhorn, K.; Borgwardt, K.M. Weisfeiler-Lehman Graph Kernels. *J. Mach. Learn. Res.* **2011**, *12*, 2539–2561. https://doi.org/10.5555/1953048.2078187.
34. Morris, C.; Kersting, K.; Mutzel, P. Glocalized Weisfeiler-Lehman Graph Kernels: Global-Local Feature Maps of Graphs. In Proceedings of the International Conference on Data Mining (ICDM 2017), New Orleans, LA, USA, 18–21 November 2017; IEEE Computer Society: New Orleans, LA, USA, 2017; pp. 327–336. https://doi.org/10.1109/ICDM.2017.42.
35. Nguyen, D.H.; Nguyen, C.H.; Mamitsuka, H. Learning Subtree Pattern Importance for Weisfeiler-Lehman Based Graph Kernels. *Mach. Learn.* **2021**, *110*, 1585–1607. https://doi.org/10.1007/s10994-021-05991-y.
36. Ye, W.; Tian, H.; Chen, Q. Graph Kernels Based on Multi-scale Graph Embeddings. *arXiv* **2022**, arXiv:2206.00979. https://doi.org/10.48550/ARXIV.2206.00979.
37. Orsini, F.; Frasconi, P.; Raedt, L.D. Graph Invariant Kernels. In Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015), Buenos Aires, Argentina, 25–31 July 2015; AAAI Press: Buenos Aires, Argentina, 2015; pp. 3756–3762.
38. Belkin, M.; Niyogi, P. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In Proceedings of the 14th Annual Conference on Neural Information Processing Systems (NIPS 2001), Vancouver, BC, Canada, 3–8 December 2001; Dietterich, T.G., Becker, S., Ghahramani, Z., Eds.; MIT Press: Vancouver, BC, Canada, 2001; pp. 585–591.
39. Gong, C.; Tao, D.; Yang, J.; Fu, K. Signed Laplacian Embedding for Supervised Dimension Reduction. In Proceedings of the 28th Conference on Artificial Intelligence (AAAI 2014), Québec City, QC, Canada, 27–31 July 2014; AAAI Press: Québec City, QC, Canada, 2014; pp. 1847–1853.
40. Allab, K.; Labiod, L.; Nadif, M. A Semi-NMF-PCA Unified Framework for Data Clustering. *IEEE Trans. Knowl. Data Eng.* **2017**, *29*, 2–16. https://doi.org/10.1109/TKDE.2016.2606098.
41. Belkin, M.; Niyogi, P. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Comput.* **2003**, *15*, 1373–1396. https://doi.org/10.1162/089976603321780317.
42. He, X.; Niyogi, P. Locality Preserving Projections. In Proceedings of the 16th Neural Information Processing Systems (NIPS 2003), Vancouver and Whistler, British Columbia, Canada, 8–13 December 2003; MIT Press: Vancouver/Whistler, BC, Canada, 2003; pp. 153–160.
43. Zhang, J.; Dong, Y.; Wang, Y.; Tang, J.; Ding, M. ProNE: Fast and Scalable Network Representation Learning. In Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019), Macao, China, 10–16 August 2019; ijcai.org: Macao, China, 2019; pp. 4278–4284. https://doi.org/10.24963/ijcai.2019/594.

44. Gori, M.; Monfardini, G.; Scarselli, F. A new model for learning in graph domains. In Proceedings of the International Joint Conference on Neural Networks (IJCNN 2005), Montreal, QC, Canada, 31 July–04 August 2005; IEEE: Montreal, QC, Canada, 2005; Volume 2; pp. 729–734. https://doi.org/10.1109/IJCNN.2005.1555942.
45. Scarselli, F.; Gori, M.; Tsoi, A.C.; Hagenbuchner, M.; Monfardini, G. The Graph Neural Network Model. *IEEE Trans. Neural Netw.* **2009**, *20*, 61–80. https://doi.org/10.1109/TNN.2008.2005605.
46. Zhao, J.; Li, C.; Wen, Q.; Wang, Y.; Liu, Y.; Sun, H.; Xie, X.; Ye, Y. Gophormer: Ego-Graph Transformer for Node Classification. *CoRR* **2021**, arXiv:2110.13094. http://arxiv.org/abs/2110.13094.
47. Schmitt, M.; Ribeiro, L.F.R.; Dufter, P.; Gurevych, I.; Schütze, H. Modeling Graph Structure via Relative Position for Better Text Generation from Knowledge Graphs. *CoRR* **2020**, arXiv:2006.09242. http://arxiv.org/abs/2006.09242.
48. Zhang, C.; Swami, A.; Chawla, N.V. SHNE: Representation Learning for Semantic-Associated Heterogeneous Networks. In Proceedings of the 12th International Conference on Web Search and Data Mining (WSDM 2019), Melbourne, VIC, Australia, 11–15 February 2019; ACM: Melbourne, VIC, Australia, 2019; pp. 690–698. https://doi.org/10.1145/3289600.3291001.
49. Wang, J.; Zheng, V.W.; Liu, Z.; Chang, K.C. Topological Recurrent Neural Network for Diffusion Prediction. In Proceedings of the International Conference on Data Mining (ICDM 2017), New Orleans, LA, USA, 18–21 November 2017; Raghavan, V., Aluru, S., Karypis, G., Miele, L., Wu, X., Eds.; IEEE Computer Society: New Orleans, LA, USA, 2017; pp. 475–484. https://doi.org/10.1109/ICDM.2017.57.
50. Wang, D.; Cui, P.; Zhu, W. Structural Deep Network Embedding. In Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining (SIGKDD 2016), San Francisco, CA, USA, 13–17 August 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 1225–1234. https://doi.org/10.1145/2939672.2939753.
51. Tu, K.; Cui, P.; Wang, X.; Wang, F.; Zhu, W. Structural Deep Embedding for Hyper-Networks. In Proceedings of the 32nd Conference on Artificial Intelligence (AAAI 2018), New Orleans, LA, USA, 2–7 February 2018; AAAI Press: New Orleans, LA, USA, 2018; pp. 426–433.
52. Chen, J.; Ma, T.; Xiao, C. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In Proceedings of the 6th International Conference on Learning Representations (ICLR 2018), Vancouver, BC, Canada, 30 April–3 May 2018; OpenReview.net: Vancouver, BC, Canada, 2018.
53. Zeng, H.; Zhou, H.; Srivastava, A.; Kannan, R.; Prasanna, V.K. GraphSAINT: Graph Sampling Based Inductive Learning Method. In Proceedings of the 8th International Conference on Learning Representations (ICLR 2020, Addis Ababa, Ethiopia, 26–30 April 2020; OpenReview.net: Addis Ababa, Ethiopia, 2020.
54. Jiang, H.; Cao, P.; Xu, M.; Yang, J.; Zaïane, O.R. Hi-GCN: A hierarchical graph convolution network for graph embedding learning of brain network and brain disorders prediction. *Comput. Biol. Med.* **2020**, *127*, 104096. https://doi.org/10.1016/j.compbiomed.2020.104096.
55. Li, R.; Huang, J. Learning Graph While Training: An Evolving Graph Convolutional Neural Network. *CoRR* **2017**, arXiv:1708.04675. http://arxiv.org/abs/1708.04675.
56. Bruna, J.; Zaremba, W.; Szlam, A.; LeCun, Y. Spectral Networks and Locally Connected Networks on Graphs. In Proceedings of the 2nd International Conference on Learning Representations (ICLR 2014), Banff, AB, Canada, 14–16 April 2014; ICLR: Banff, AB, Canada, 2014.
57. Seo, M.; Lee, K.Y. A Graph Embedding Technique for Weighted Graphs Based on LSTM Autoencoders. *J. Inf. Process. Syst.* **2020**, *16*, 1407–1423. https://doi.org/10.3745/JIPS.04.0197.
58. Brody, S.; Alon, U.; Yahav, E. How Attentive are Graph Attention Networks? In Proceedings of the 10th International Conference on Learning Representations (ICLR 2022), Virtual Event, 25–29 April 2022; OpenReview.net: Virtual Event, 2022.
59. Zhu, J.; Yan, Y.; Zhao, L.; Heimann, M.; Akoglu, L.; Koutra, D. Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs. In Proceedings of the 33rd Annual Conference on Neural Information Processing Systems (NeurIPS 2020), Virtual Event, 6–12 December 2020; Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; Lin, H., Eds.; NeurIPS: Virtual Event, 2020.
60. Suresh, S.; Budde, V.; Neville, J.; Li, P.; Ma, J. Breaking the Limit of Graph Neural Networks by Improving the Assortativity of Graphs with Local Mixing Patterns. In Proceedings of the 27th Conference on Knowledge Discovery and Data Mining (KDD 2021), Singapore, 14–18 August 2021; Zhu, F.; Ooi, B.C.; Miao, C., Eds.; ACM: Singapore, 2021; pp. 1541–1551. https://doi.org/10.1145/3447548.3467373.
61. Nguyen, D.Q.; Nguyen, T.D.; Phung, D. Universal Graph Transformer Self-Attention Networks. In Proceedings of the Companion Proceedings of the Web Conference 2022 (WWW 2022), Lyon, France, 25–29 April 2022; Association for Computing Machinery: New York, NY, USA, 2022; pp. 193–196. https://doi.org/10.1145/3487553.3524258.
62. Zhu, J.; Li, J.; Zhu, M.; Qian, L.; Zhang, M.; Zhou, G. Modeling Graph Structure in Transformer for Better AMR-to-Text Generation. In Proceedings of the Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP 2019), Hong Kong, China, 3–7 November 2019; Association for Computational Linguistics: Hong Kong, China, 2019; pp. 5458–5467. https://doi.org/10.18653/v1/D19-1548.
63. Zhang, J.; Zhang, H.; Xia, C.; Sun, L. Graph-Bert: Only Attention is Needed for Learning Graph Representations. *CoRR* **2020**, arXiv:2001.05140. http://arxiv.org/abs/2001.05140.

64. Shiv, V.L.; Quirk, C. Novel positional encodings to enable tree-based transformers. In Proceedings of the 32nd Annual Conference on Neural Information Processing Systems 2019 (NeurIPS 2019), Vancouver, BC, Canada, 8–14 December 2019; NeurIPS 2019, Vancouver, BC, Canada, 2019; pp. 12058–12068.
65. Wang, X.; Tu, Z.; Wang, L.; Shi, S. Self-Attention with Structural Position Representations. In Proceedings of the Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP 2019), Hong Kong, China, 3–7 November 2019; Association for Computational Linguistics: Hong Kong, China, 2019; pp. 1403–1409. https://doi.org/10.18653/v1/D19-1145.
66. Shi, Y.; Huang, Z.; Feng, S.; Zhong, H.; Wang, W.; Sun, Y. Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification. In Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI 2021), Montreal, QC, Canada, 19–27 August 2021; Zhou, Z., Ed.; ijcai.org: Montreal, QC, Canada, 2021; pp. 1548–1554. https://doi.org/10.24963/ijcai.2021/214.
67. Ying, C.; Cai, T.; Luo, S.; Zheng, S.; Ke, G.; He, D.; Shen, Y.; Liu, T. Do Transformers Really Perform Badly for Graph Representation? In Proceedings of the 34th Annual Conference on Neural Information Processing Systems 2021 (NeurIPS 2021), Virtual Event, 6–14 December 2021; Ranzato, M., Beygelzimer, A., Dauphin, Y.N., Liang, P., Vaughan, J.W., Eds.; nips.cc: Virtual Event, 2021; pp. 28877–28888.
68. Zhang, Y.; Wang, X.; Shi, C.; Jiang, X.; Ye, Y. Hyperbolic Graph Attention Network. *IEEE Trans. Big Data* **2022**, *8*, 1690–1701. https://doi.org/10.1109/TBDATA.2021.3081431.
69. Zhang, Y.; Wang, X.; Shi, C.; Liu, N.; Song, G. Lorentzian Graph Convolutional Networks. In Proceedings of the The Web Conference (WWW 2021), Ljubljana, Slovenia, 19–23 April 2021; ACM/IW3C2: Ljubljana, Slovenia, 2021; pp. 1249–1261. https://doi.org/10.1145/3442381.3449872.
70. Nickel, M.; Kiela, D. Poincaré Embeddings for Learning Hierarchical Representations. In Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 4–9 December 2017; nips.cc: Long Beach, CA, USA, 2017; pp. 6338–6347.
71. Wang, X.; Zhang, Y.; Shi, C. Hyperbolic Heterogeneous Information Network Embedding. In Proceedings of the 33rd Conference on Artificial Intelligence (AAAI 2019), Honolulu, Hawaii, USA, 27 January–1 February 2019; AAAI Press: Honolulu, HI, USA, 2019; pp. 5337–5344. https://doi.org/10.1609/aaai.v33i01.33015337.
72. Kipf, T.N.; Welling, M. Variational Graph Auto-Encoders. *CoRR* **2016**, arXiv:1611.07308. http://arxiv.org/abs/1611.07308.
73. Zhang, D.; Yin, J.; Zhu, X.; Zhang, C. Network Representation Learning: A Survey. *IEEE Trans. Big Data* **2020**, *6*, 3–28. https://doi.org/10.1109/TBDATA.2018.2850013.
74. Hamilton, W.L.; Ying, R.; Leskovec, J. Representation Learning on Graphs: Methods and Applications. *CoRR* **2017**, arXiv:1709.05584. http://arxiv.org/abs/1709.05584.
75. Cai, H.; Zheng, V.W.; Chang, K.C. A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications. *IEEE Trans. Knowl. Data Eng.* **2018**, *30*, 1616–1637. https://doi.org/10.1109/TKDE.2018.2807452.
76. Zhou, C.; Liu, Y.; Liu, X.; Liu, Z.; Gao, J. Scalable Graph Embedding for Asymmetric Proximity. In Proceedings of the 31th Conference on Artificial Intelligence (AAAI 2017), San Francisco, CA, USA, 4–9 February 2017; AAAI Press: San Francisco, CA, USA, 2017; pp. 2942–2948.
77. Man, T.; Shen, H.; Liu, S.; Jin, X.; Cheng, X. Predict Anchor Links across Social Networks via an Embedding Approach. In Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016), New York, NY, USA, 9–15 July 2016; IJCAI/AAAI Press: New York, NY, USA, 2016; pp. 1823–1829.
78. Gui, H.; Liu, J.; Tao, F.; Jiang, M.; Norick, B.; Han, J. Large-Scale Embedding Learning in Heterogeneous Event Data. In Proceedings of the 16th IEEE International Conference on Data Mining (ICDM 2016), Barcelona, Spain, 12–15 December 2016; IEEE Computer Society: Barcelona, Spain, 2016; pp. 907–912. https://doi.org/10.1109/ICDM.2016.0111.
79. Goyal, P.; Ferrara, E. Graph embedding techniques, applications, and performance: A survey. *Knowl.-Based Syst.* **2018**; Volume 151, pp. 78–94. https://doi.org/10.1016/j.knosys.2018.03.022.
80. Casteigts, A.; Flocchini, P.; Quattrociocchi, W.; Santoro, N. Time-varying graphs and dynamic networks. *Int. J. Parallel Emergent Distrib. Syst.* **2012**, *27*, 387–408. https://doi.org/10.1080/17445760.2012.668546.
81. Li, J.; Dani, H.; Hu, X.; Tang, J.; Chang, Y.; Liu, H. Attributed Network Embedding for Learning in a Dynamic Environment. In Proceedings of the Conference on Information and Knowledge Management (CIKM 2017), Singapore, 6–10 November 2017; ACM: Singapore, 2017; pp. 387–396. https://doi.org/10.1145/3132847.3132919.
82. Chen, C.; Tao, Y.; Lin, H. Dynamic Network Embeddings for Network Evolution Analysis. *CoRR* **2019**, arXiv:1906.09860. http://arxiv.org/abs/1906.09860.
83. Shervashidze, N.; Vishwanathan, S.V.N.; Petri, T.; Mehlhorn, K.; Borgwardt, K.M. Efficient graphlet kernels for large graph comparison. In Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS 2009), Clearwater Beach, FL, USA, 16–18 April 2009; JMLR.org: Clearwater Beach, FL, USA, 2009; Volume 5; pp. 488–495.
84. Togninalli, M.; Ghisu, M.E.; Llinares-López, F.; Rieck, B.; Borgwardt, K.M. Wasserstein Weisfeiler-Lehman Graph Kernels. In Proceedings of the 32nd Annual Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, BC, Canada, 8–14 December 2019; NeurIPS: Vancouver, BC, Canada, 2019; pp. 6436–6446.

85. Kang, U.; Tong, H.; Sun, J. Fast Random Walk Graph Kernel. In Proceedings of the 12th International Conference on Data Mining (ICDM 2012), Anaheim, CA, USA, 26–28 April 2012; SIAM/Omnipress: Anaheim, CA, USA, 2012; pp. 828–838. https://doi.org/10.1137/1.9781611972825.71.
86. Zhang, Z.; Cui, P.; Wang, X.; Pei, J.; Yao, X.; Zhu, W. Arbitrary-Order Proximity Preserved Network Embedding. In Proceedings of the 24th International Conference on Knowledge Discovery & Data Mining (KDD 2018), London, UK, 19–23 August 2018; ACM: London, UK, 2018; pp. 2778–2786. https://doi.org/10.1145/3219819.3219969.
87. Dareddy, M.R.; Das, M.; Yang, H. motif2vec: Motif Aware Node Representation Learning for Heterogeneous Networks. In Proceedings of the International Conference on Big Data (IEEE BigData), Los Angeles, CA, USA, 9–12 December 2019; IEEE: Los Angeles, CA, USA, 2019; pp. 1052–1059. https://doi.org/10.1109/BigData47090.2019.9005670.
88. Tu, K.; Li, J.; Towsley, D.; Braines, D.; Turner, L.D. gl2vec: learning feature representation using graphlets for directed networks. In Proceedings of the 19th International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2019), Vancouver, BC, Canada, 27–30 August 2019; ACM: Vancouver, BC, Canada, 2019; pp. 216–221. https://doi.org/10.1145/3341161.3342908.
89. Yu, Y.; Lu, Z.; Liu, J.; Zhao, G.; Wen, J. RUM: Network Representation Learning Using Motifs. In Proceedings of the 35th International Conference on Data Engineering (ICDE 2019), Macao, China, 8–11 April 2019; IEEE: Macao, China, April 2019; pp. 1382–1393. https://doi.org/10.1109/ICDE.2019.00125.
90. Hu, Q.; Lin, F.; Wang, B.; Li, C. MBRep: Motif-Based Representation Learning in Heterogeneous Networks. *Expert Syst. Appl.* **2022**, *190*, 116031. https://doi.org/10.1016/j.eswa.2021.116031.
91. Shao, P.; Yang, Y.; Xu, S.; Wang, C. Network Embedding via Motifs. *ACM Trans. Knowl. Discov. Data* **2021**, *16*, 44. https://doi.org/10.1145/3473911.
92. De Winter, S.; Decuypere, T.; Mitrović, S.; Baesens, B.; De Weerdt, J. Combining Temporal Aspects of Dynamic Networks with Node2Vec for a More Efficient Dynamic Link Prediction. In Proceedings of the International Conference on Advances in Social Networks Analysis and Mining (IEEE/ACM 2018), Barcelona, Spain, 28–31 August 2018; IEEE Press: Barcelona, Spain, 2018; pp. 1234–1241.
93. Huang, X.; Song, Q.; Li, Y.; Hu, X. Graph Recurrent Networks With Attributed Random Walks. In Proceedings of the 25th International Conference on Knowledge Discovery & Data Mining (KDD 2019), Anchorage, AK, USA, 4–8 August 2019; ACM: Anchorage, AK, USA, 2019; pp. 732–740. https://doi.org/10.1145/3292500.3330941.
94. Sperduti, A.; Starita, A. Supervised neural networks for the classification of structures. *IEEE Trans. Neural Netw.* **1997**, *8*, 714–735. https://doi.org/10.1109/72.572108.
95. Chiang, W.; Liu, X.; Si, S.; Li, Y.; Bengio, S.; Hsieh, C. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In Proceedings of the 25th International Conference on Knowledge Discovery & Data Mining (KDD 2019), Anchorage, AK, USA, 4–8 August 2019; ACM: Anchorage, AK, USA, 2019; pp. 257–266. https://doi.org/10.1145/3292500.3330925.
96. Defferrard, M.; Bresson, X.; Vandergheynst, P. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NeurIPS 2016), Barcelona, Spain, 5–10 December 2016; NeurIPS: Barcelona, Spain, 2016;pp. 3837–3845.
97. Li, Q.; Han, Z.; Wu, X. Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning. In Proceedings of the 32nd Conference on Artificial Intelligence (AAAI-18), New Orleans, LA, USA, 2–7 February 2018; McIlraith, S.A.; Weinberger, K.Q., Eds.; AAAI Press: New Orleans, LA, USA, 2018; pp. 3538–3545.
98. Li, G.; Xiong, C.; Thabet, A.K.; Ghanem, B. DeeperGCN: All You Need to Train Deeper GCNs. *CoRR* **2020**, arXiv:2006.07739. http://arxiv.org/abs/2006.07739.
99. Lin, K.; Wang, L.; Liu, Z. Mesh Graphormer. In Proceedings of the International Conference on Computer Vision (ICCV 2021), Montreal, QC, Canada, 10–17 October 2021; IEEE: Montreal, QC, Canada, 2021; pp. 12919–12928. https://doi.org/10.1109/ICCV48922.2021.01270.
100. Rong, Y.; Bian, Y.; Xu, T.; Xie, W.; Wei, Y.; Huang, W.; Huang, J. Self-Supervised Graph Transformer on Large-Scale Molecular Data. In Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS 2020), Virtual Event, 6–12 December 2020; NeurIPS: Virtual event, 2020.
101. Yao, S.; Wang, T.; Wan, X. Heterogeneous Graph Transformer for Graph-to-Sequence Learning. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020), Virtual Event, 5–10 July 2020; ACL: Virtual Event, 2020; pp. 7145–7154. https://doi.org/10.18653/v1/2020.acl-main.640.
102. Nickel, M.; Kiela, D. Learning Continuous Hierarchies in the Lorentz Model of Hyperbolic Geometry. In Proceedings of the 35th International Conference on Machine Learning (ICML 2018), Stockholm, Sweden, 10–15 July 2018; mlr.press: Stockholm, Sweden, 2018; Volume 80; pp. 3776–3785.
103. Cao, Z.; Xu, Q.; Yang, Z.; Cao, X.; Huang, Q. Geometry Interaction Knowledge Graph Embeddings. In Proceedings of the 36th Conference on Artificial Intelligence (AAAI 2022), Virtual Event, 22 February–1 March 2022; AAAI Press: Virtual Event, 2022; pp. 5521–5529.

104. Santos, L.D.; Piwowarski, B.; Gallinari, P. Multilabel Classification on Heterogeneous Graphs with Gaussian Embeddings. In Proceedings of the Machine Learning and Knowledge Discovery in Databases—European Conference (ECML PKDD 2016), Riva del Garda, Italy, 19–23 September 2016; Springer: Riva del Garda, Italy, 2016; Volume 9852; pp. 606–622. https://doi.org/10.1007/978-3-319-46227-1_38.
105. Kriege, N.M. Comparing Graphs: Algorithms & Applications. Ph.D. Thesis, Technische Universität, Dortmund, Germany, 2015.
106. Kondor, R.; Shervashidze, N.; Borgwardt, K.M. The graphlet spectrum. In Proceedings of the 26th Annual International Conference on Machine Learning (ICML 2009), Montreal, QC, Canada, 14–18 June 2009; ACM: Montreal, QC, Canada, 2009; Volume 382; pp. 529–536. https://doi.org/10.1145/1553374.1553443.
107. Feragen, A.; Kasenburg, N.; Petersen, J.; de Bruijne, M.; Borgwardt, K.M. Scalable kernels for graphs with continuous attributes. In Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NeurIPS 2013), Lake Tahoe, NV, USA, 5–8 December 2013; Burges, C.J.C., Bottou, L., Ghahramani, Z., Weinberger, K.Q., Eds.; NeurIPS: Lake Tahoe, NV, USA, 2013; pp. 216–224.
108. Morris, C.; Kriege, N.M.; Kersting, K.; Mutzel, P. Faster Kernels for Graphs with Continuous Attributes via Hashing. In Proceedings of the 16th International Conference on Data Mining (ICDM 2016), Barcelona, Spain, 12–15 December 2016; IEEE Computer Society: Barcelona, Spain, 2016; pp. 1095–1100. https://doi.org/10.1109/ICDM.2016.0142.
109. Borgwardt, K.M.; Kriegel, H. Shortest-Path Kernels on Graphs. In Proceedings of the 5th International Conference on Data Mining (ICDM 2005), Houston, TX, USA, 27–30 November 2005; IEEE Computer Society: Houston, TX, USA, 2005; pp. 74–81. https://doi.org/10.1109/ICDM.2005.132.
110. Kashima, H.; Tsuda, K.; Inokuchi, A. Marginalized Kernels Between Labeled Graphs. In Proceedings of the 20th International Conference Machine Learning (ICML 2003), Washington, DC, USA, 21–24 August 2003; AAAI Press: Washington, DC, USA, 2003; pp. 321–328.
111. Rahman, M.; Hasan, M.A. Link Prediction in Dynamic Networks Using Graphlet. In Proceedings of the Machine Learning and Knowledge Discovery in Databases—European Conference (ECML PKDD 2016), Riva del Garda, Italy, 19–23 September 2016; Springer: Riva del Garda, Italy, 2016; Volume 9851; pp. 394–409. https://doi.org/10.1007/978-3-319-46128-1_25.
112. Rahman, M.; Saha, T.K.; Hasan, M.A.; Xu, K.S.; Reddy, C.K. DyLink2Vec: Effective Feature Representation for Link Prediction in Dynamic Networks. *CoRR* **2018**, arXiv:1804.05755. http://arxiv.org/abs/1804.05755.
113. Béres, F.; Kelen, D.M.; Pálovics, R.; Benczúr, A.A. Node embeddings in dynamic graphs. *Appl. Netw. Sci.* **2019**, *4*, 64.1–64.25. https://doi.org/10.1007/s41109-019-0169-5.
114. Cuturi, M. Sinkhorn Distances: Lightspeed Computation of Optimal Transport. In Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NeurIPS 2013), Lake Tahoe, NV, USA, 5–8 December 2013; NeurIPS: Lake Tahoe, NV, USA, 2013; pp. 2292–2300.
115. Floyd, R.W. Algorithm 97: Shortest path. *Commun. ACM* **1962**, *5*, 345. https://doi.org/10.1145/367766.368168.
116. Kriege, N.M.; Johansson, F.D.; Morris, C. A survey on graph kernels. *Appl. Netw. Sci.* **2020**, *5*, 6. https://doi.org/10.1007/s41109-019-0195-3.
117. Urry, M.; Sollich, P. Random walk kernels and learning curves for Gaussian process regression on random graphs. *J. Mach. Learn. Res.* **2013**, *14*, 1801–1835. https://doi.org/10.5555/2567709.2567721.
118. Bunke, H.; Irniger, C.; Neuhaus, M. Graph Matching—Challenges and Potential Solutions. In Proceedings of the 13th Image Analysis and Processing— International Conference (ICIAP 2005), Cagliari, Italy, 6–8 September 2005; Springer: Cagliari, Italy, 2005; Volume 3617; pp. 1–10. https://doi.org/10.1007/11553595_1.
119. Hofmann, T.; Buhmann, J.M. Multidimensional Scaling and Data Clustering. In Proceedings of the 7th Advances in Neural Information Processing Systems (NIPS 1994), Denver, CO, USA, 1994; Tesauro, G., Touretzky, D.S., Leen, T.K., Eds.; MIT Press: Denver, CO, USA, 1994; pp. 459–466.
120. Han, Y.; Shen, Y. Partially Supervised Graph Embedding for Positive Unlabelled Feature Selection. In Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016), New York, NY, USA, 9–15 July 2016; IJCAI/AAAI Press: New York, NY, USA, 2016; pp. 1548–1554.
121. Cai, D.; He, X.; Han, J. Spectral regression: A unified subspace learning framework for content-based image retrieval. In Proceedings of the 15th International Conference on Multimedia 2007 (ICM 2007), Augsburg, Germany, 24–29 September 2007; ACM: Augsburg, Germany, 2007; pp. 403–412. https://doi.org/10.1145/1291233.1291329.
122. Torres, L.; Chan, K.S.; Eliassi-Rad, T. GLEE: Geometric Laplacian Eigenmap Embedding. *J. Complex Netw.* **2020**, *8*, cnaa007. https://doi.org/10.1093/comnet/cnaa007.
123. Li, Y.; Wang, Y.; Zhang, T.; Zhang, J.; Chang, Y. Learning Network Embedding with Community Structural Information. In Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019), Macao, China, 10–16 August 2019; ijcai.org: Macao, China, 2019; pp. 2937–2943. https://doi.org/10.24963/ijcai.2019/407.
124. Coskun, M. A high order proximity measure for linear network embedding. *NiğDe öMer Halisdemir üNiversitesi MüHendislik Bilim. Derg.* **2022**, *11*, 477–483. https://doi.org/10.28948/ngumuh.957488.
125. Ahmed, A.; Shervashidze, N.; Narayanamurthy, S.M.; Josifovski, V.; Smola, A.J. Distributed large-scale natural graph factorization. In Proceedings of the 22nd International World Wide Web Conference (WWW 2013), Rio de Janeiro, Brazil, 13–17 May 2013; International World Wide Web Conferences Steering Committee/ACM: Rio de Janeiro, Brazil, 2013; pp. 37–48. https://doi.org/10.1145/2488388.2488393.

126. Lian, D.; Zhu, Z.; Zheng, K.; Ge, Y.; Xie, X.; Chen, E. Network Representation Lightening from Hashing to Quantization. *IEEE Trans. Knowl. Data Eng.* **2022**, 35, 5119–5131. https://doi.org/10.1109/TKDE.2022.3151474.
127. Katz, L. A new status index derived from sociometric analysis. *Psychometrika* **1953**, *18*, 39–43. https://doi.org/https://doi.org/10.1007/BF02289026.
128. Page, L.; Brin, S.; Motwani, R.; Winograd, T. *The PageRank Citation Ranking: Bringing Order to the Web*; Technical Report 1999-66; Stanford InfoLab: Stanford, CA, USA, 1999; Previous number = SIDL-WP-1999-0120.
129. Adamic, L.A.; Adar, E. Friends and neighbors on the Web. *Soc. Netw.* **2003**, *25*, 211–230. https://doi.org/10.1016/S0378-8733(03)00009-1.
130. Zhao, Y.; Liu, Z.; Sun, M. Representation Learning for Measuring Entity Relatedness with Rich Information. In Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015), Buenos Aires, Argentina, 25–31 July 2015 ; AAAI Press: Buenos Aires, Argentina, 2015; pp. 1412–1418.
131. Yang, C.; Liu, Z.; Zhao, D.; Sun, M.; Chang, E.Y. Network Representation Learning with Rich Text Information. In Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015), Buenos Aires, Argentina, 25–31 July 2015; AAAI Press: Buenos Aires, Argentina, 2015; pp. 2111–2117.
132. Yang, R.; Shi, J.; Xiao, X.; Yang, Y.; Bhowmick, S.S. Homogeneous Network Embedding for Massive Graphs via Reweighted Personalized PageRank. *Proc. Vldb Endow.* **2020**, *13*, 670–683. https://doi.org/10.14778/3377369.3377376.
133. Qiu, J.; Dong, Y.; Ma, H.; Li, J.; Wang, C.; Wang, K.; Tang, J. NetSMF: Large-Scale Network Embedding as Sparse Matrix Factorization. In Proceedings of The World Wide Web Conference (WWW 2019), San Francisco, CA, USA, 13–17 May 2019; ACM: New York, NY, USA, 2019; pp. 1509–1520. https://doi.org/10.1145/3308558.3313446.
134. Rossi, R.A.; Gallagher, B.; Neville, J.; Henderson, K. Modeling dynamic behavior in large evolving graphs. In Proceedings of the 6th International Conference on Web Search and Data Mining (WSDM 2013), Rome, Italy, 4–8 February 2013; ACM: Rome, Italy, 2013; pp. 667–676. https://doi.org/10.1145/2433396.2433479.
135. Yu, W.; Aggarwal, C.C.; Wang, W. Temporally Factorized Network Modeling for Evolutionary Network Analysis. In Proceedings of the 10th International Conference on Web Search and Data Mining (WSDM 2017), Cambridge, United Kingdom, 6–10 February 2017; ACM: Cambridge, UK, 2017; pp. 455–464. https://doi.org/10.1145/3018661.3018669.
136. Zhu, L.; Guo, D.; Yin, J.; Steeg, G.V.; Galstyan, A. Scalable Temporal Latent Space Inference for Link Prediction in Dynamic Social Networks. *IEEE Trans. Knowl. Data Eng.* **2016**, *28*, 2765–2777. https://doi.org/10.1109/TKDE.2016.2591009.
137. Yu, W.; Cheng, W.; Aggarwal, C.C.; Chen, H.; Wang, W. Link Prediction with Spatial and Temporal Consistency in Dynamic Networks. In Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI 2017), Melbourne, Australia, 19–25 August 2017; ijcai.org: Melbourne, Australia, 2017; pp. 3343–3349. https://doi.org/10.24963/ijcai.2017/467.
138. Chen, H.; Yin, H.; Wang, W.; Wang, H.; Nguyen, Q.V.H.; Li, X. PME: Projected Metric Embedding on Heterogeneous Networks for Link Prediction. In Proceedings of the 24th International Conference on Knowledge Discovery & Data Mining (KDD 2018), London, UK, 19–23 August 2018; ACM: London, UK, 2018; pp. 1177–1186. https://doi.org/10.1145/3219819.3219986.
139. Xu, L.; Wei, X.; Cao, J.; Yu, P.S. Embedding of Embedding (EOE): Joint Embedding for Coupled Heterogeneous Networks. In Proceedings of the 10th International Conference on Web Search and Data Mining (WSDM 2017), Cambridge, UK, 6–10 February 2017; ACM: Cambridge, UK, 2017; pp. 741–749. https://doi.org/10.1145/3018661.3018723.
140. Shi, Y.; Gui, H.; Zhu, Q.; Kaplan, L.M.; Han, J. AspEm: Embedding Learning by Aspects in Heterogeneous Information Networks. In Proceedings of the International Conference on Data Mining (SDM 2018), San Diego, CA, USA, 3–5 May 2018; SIAM: San Diego, CA, USA, 2018; pp. 144–152. https://doi.org/10.1137/1.9781611975321.16.
141. Matsuno, R.; Murata, T. MELL: Effective Embedding Method for Multiplex Networks. In Proceedings of the Companion Proceedings of The Web Conference 2018 (WWW 2018), Lyon, France, 23–27 April 2018; International World Wide Web Conferences Steering Committee: Geneva, Switzerland, 2018; ACM: Lyon, France, 2018; pp. 1261–1268. https://doi.org/10.1145/3184558.3191565.
142. Ren, X.; He, W.; Qu, M.; Voss, C.R.; Ji, H.; Han, J. Label Noise Reduction in Entity Typing by Heterogeneous Partial-Label Embedding. In Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining (KDD 2016), San Francisco, CA, USA, 13–17 August 2016 ; ACM: San Francisco, CA, USA, 2016; pp. 1825–1834. https://doi.org/10.1145/2939672.2939822.
143. Dettmers, T.; Minervini, P.; Stenetorp, P.; Riedel, S. Convolutional 2D Knowledge Graph Embeddings. In Proceedings of the 32nd Conference on Artificial Intelligence (AAAI 2018), New Orleans, LA, USA, 2–7 February 2018; AAAI Press: New Orleans, LA, USA, 2018; pp. 1811–1818.
144. Safavi, T.; Koutra, D. CoDEx: A Comprehensive Knowledge Graph Completion Benchmark. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2020), Virtual Event, 16–20 November 2020; Webber, B., Cohn, T., He, Y., Liu, Y., Eds.; ACL: Virtual Event, 2020; pp. 8328–8350. https://doi.org/10.18653/v1/2020.emnlp-main.669.
145. Narayanan, A.; Chandramohan, M.; Chen, L.; Liu, Y.; Saminathan, S. subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs. *CoRR* **2016**, arXiv:1606.08928. http://arxiv.org/abs/1606.08928.
146. Wang, C.; Wang, C.; Wang, Z.; Ye, X.; Yu, P.S. Edge2vec: Edge-based Social Network Embedding. *ACM Trans. Knowl. Discov. Data* **2020**, *14*, 45.1–45.24. https://doi.org/10.1145/3391298.
147. Perozzi, B.; Kulkarni, V.; Chen, H.; Skiena, S. Don't Walk, Skip!: Online Learning of Multi-scale Network Embeddings. In Proceedings of the International Conference on Advances in Social Networks Analysis and Mining 2017 (ASONAM 2017), Sydney, Australia, 31 July–3 August 2017; ACM: Sydney, Australia, 2017; pp. 258–265. https://doi.org/10.1145/3110025.3110086.

148. Liu, R.; Hirn, M.J.; Krishnan, A. Accurately Modeling Biased Random Walks on Weighted Graphs Using Node2vec+. *CoRR* **2021**, arXiv:2109.08031. http://arxiv.org/abs/2109.08031.
149. Jeong, J.; Yun, J.; Keam, H.; Park, Y.; Park, Z.; Cho, J. div2vec: Diversity-Emphasized Node Embedding. In Proceedings of the 14th Workshops on Recommendation in Complex Scenarios and the Impact of Recommender Systems co-located with 14th ACM Conference on Recommender Systems (RecSys 2020), Virtual Event, 25 September 2020; CEUR-WS.org: Virtual Event, 2020; Volume 2697.
150. Zhang, Y.; Shi, Z.; Feng, D.; Zhan, X.X. Degree-Biased Random Walk for Large-Scale Network Embedding. *Future Gener. Comput. Syst.* **2019**, *100*, 198–209. https://doi.org/10.1016/j.future.2019.05.033.
151. Ahmed, N.K.; Rossi, R.A.; Lee, J.B.; Willke, T.L.; Zhou, R.; Kong, X.; Eldardiry, H. Role-Based Graph Embeddings. *IEEE Trans. Knowl. Data Eng.* **2022**, *34*, 2401–2415. https://doi.org/10.1109/TKDE.2020.3006475.
152. Khosla, M.; Leonhardt, J.; Nejdl, W.; Anand, A. Node Representation Learning for Directed Graphs. In Proceedings of the Machine Learning and Knowledge Discovery in Databases—European Conference (ECML PKDD 2019), Dublin, Ireland, 10–14 September 2018; Springer: Würzburg, Germany, 2019; Volume 11906; pp. 395–411. https://doi.org/10.1007/978-3-030-46150-8_24.
153. Adhikari, B.; Zhang, Y.; Ramakrishnan, N.; Prakash, B.A. Sub2Vec: Feature Learning for Subgraphs. In Proceedings of the 22nd Advances in Knowledge Discovery and Data Mining - Pacific-Asia Conference (PAKDD 2018), Melbourne, VIC, Australia, 3–6 June 2018; Springer: Melbourne, VIC, Australia, 2018; Volume 10938; pp. 170–182. https://doi.org/10.1007/978-3-319-93037-4_14.
154. Sheikh, N.; Kefato, Z.; Montresor, A. Gat2vec: Representation Learning for Attributed Graphs. *Computing* **2019**, *101*, 187–209. https://doi.org/10.1007/s00607-018-0622-9.
155. Dou, W.; Zhang, W.; Weng, Z. An Attributed Network Representation Learning Method Based on Biased Random Walk. *Procedia Comput. Sci.* **2020**, *174*, 291–298. International Conference on Identification, Information and Knowledge in the Internet of Things 2019, https://doi.org/https://doi.org/10.1016/j.procs.2020.06.088.
156. Narayanan, A.; Chandramohan, M.; Venkatesan, R.; Chen, L.; Liu, Y.; Jaiswal, S. graph2vec: Learning Distributed Representations of Graphs. *CoRR* **2017**, arXiv:1707.05005. http://arxiv.org/abs/1707.05005.
157. Yang, D.; Qu, B.; Yang, J.; Cudré-Mauroux, P. Revisiting User Mobility and Social Relationships in LBSNs: A Hypergraph Embedding Approach. In Proceedings of the The World Wide Web Conference (WWW 2019), San Francisco, CA, USA, 13–17 May 2019; ACM: San Francisco, CA, USA, 2019; pp. 2147–2157. https://doi.org/10.1145/3308558.3313635.
158. Zhu, Y.; Guan, Z.; Tan, S.; Liu, H.; Cai, D.; He, X. Heterogeneous hypergraph embedding for document recommendation. *Neurocomputing* **2016**, *216*, 150–162. https://doi.org/10.1016/j.neucom.2016.07.030.
159. Hussein, R.; Yang, D.; Cudré-Mauroux, P. Are Meta-Paths Necessary?: Revisiting Heterogeneous Graph Embeddings. In Proceedings of the 27th International Conference on Information and Knowledge Management (CIKM 2018), Torino, Italy, 22–26 October 2018; ACM: Torino, Italy, 2018; pp. 437–446. https://doi.org/10.1145/3269206.3271777.
160. Zhang, H.; Qiu, L.; Yi, L.; Song, Y. Scalable Multiplex Network Embedding. In Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018), Stockholm, Sweden, 13–19 July 2018; ijcai.org: Stockholm, Sweden, 2018; pp. 3082–3088. https://doi.org/10.24963/ijcai.2018/428.
161. Lee, S.; Park, C.; Yu, H. BHIN2vec: Balancing the Type of Relation in Heterogeneous Information Network. In Proceedings of the 28th International Conference on Information and Knowledge Management (CIKM 2019), Beijing, China, 3–7 November 2019; ACM: Beijing, China, 2019; pp. 619–628. https://doi.org/10.1145/3357384.3357893.
162. Li, Y.; Patra, J.C. Genome-wide inferring gene-phenotype relationship by walking on the heterogeneous network. *Bioinformatics* **2010**, *26*, 1219–1224. https://doi.org/10.1093/bioinformatics/btq108.
163. Joodaki, M.; Ghadiri, N.; Maleki, Z.; Shahreza, M.L. A scalable random walk with restart on heterogeneous networks with Apache Spark for ranking disease-related genes through type-II fuzzy data fusion. *J. Biomed. Informatics* **2021**, *115*, 103688. https://doi.org/10.1016/j.jbi.2021.103688.
164. Tian, Z.; Guo, M.; Wang, C.; Xing, L.; Wang, L.; Zhang, Y. Constructing an integrated gene similarity network for the identification of disease genes. In Proceedings of the International Conference on Bioinformatics and Biomedicine (BIBM 2016), Shenzhen, China, 15–18 December 2016; IEEE Computer Society: Shenzhen, China, 2016; pp. 1663–1668. https://doi.org/10.1109/BIBM.2016.7822768.
165. Luo, J.; Liang, S. Prioritization of potential candidate disease genes by topological similarity of protein-protein interaction network and phenotype data. *J. Biomed. Sci.* **2015**, *53*, 229–236. https://doi.org/10.1016/j.jbi.2014.11.004.
166. Lee, O.; Jeon, H.; Jung, J.J. Learning multi-resolution representations of research patterns in bibliographic networks. *J. Inf.* **2021**, *15*, 101126. https://doi.org/10.1016/j.joi.2020.101126.
167. Du, B.; Tong, H. MrMine: Multi-resolution Multi-network Embedding. In Proceedings of the 28th International Conference on Information and Knowledge Management (CIKM 2019), Beijing, China, 3–7 November 2019; ACM: Beijing, China, 2019; pp. 479–488. https://doi.org/10.1145/3357384.3357944.
168. Lee, O.; Jung, J.J. Story Embedding:Learning Distributed Representations of Stories based on Character Networks. In Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020), Yokohama, Japan, 7–15 January 2021; Bessiere, C., Ed.; ijcai.org: Yokohama, Japan, 2020; pp. 5070–5074. https://doi.org/10.1016/j.artint.2020.103235.
169. Sajjad, H.P.; Docherty, A.; Tyshetskiy, Y. Efficient Representation Learning Using Random Walks for Dynamic Graphs. *CoRR* **2019**, arXiv:1901.01346. http://arxiv.org/abs/1901.01346.

170. Heidari, F.; Papagelis, M. EvoNRL: Evolving Network Representation Learning Based on Random Walks. In Proceedings of the 7th International Conference on Complex Networks and Their Applications (COMPLEX NETWORKS 2018), Cambridge, UK, 11–13 December 2018; Springer: Cambridge, UK, 2018; Volume 812; pp. 457–469. https://doi.org/10.1007/978-3-030-05411-3_37.
171. Nguyen, G.H.; Lee, J.B.; Rossi, R.A.; Ahmed, N.K.; Koh, E.; Kim, S. Continuous-Time Dynamic Network Embeddings. In Proceedings of the Companion of The Web Conference (WWW 2018), Lyon, France, 23–27 April 2018; ACM: Lyon, France, 2018; pp. 969–976. https://doi.org/10.1145/3184558.3191526.
172. Pandhre, S.; Mittal, H.; Gupta, M.; Balasubramanian, V.N. STwalk: learning trajectory representations in temporal graphs. In Proceedings of the India Joint International Conference on Data Science and Management of Data (COMAD/CODS 2018), Goa, India, 11–13 January 2018; ACM: Goa, India, 2018; pp. 210–219. https://doi.org/10.1145/3152494.3152512.
173. Singer, U.; Guy, I.; Radinsky, K. Node Embedding over Temporal Graphs. In Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019), Macao, China, 10–16 August 2019; ijcai.org: Macao, China, 2019; pp. 4605–4612. https://doi.org/10.24963/ijcai.2019/640.
174. Mahdavi, S.; Khoshraftar, S.; An, A. dynnode2vec: Scalable Dynamic Network Embedding. In Proceedings of the International Conference on Big Data (IEEE BigData 2018), Seattle, WA, USA, 10–13 December 2018; IEEE: Seattle, WA, USA, 2018; pp. 3762–3765. https://doi.org/10.1109/BigData.2018.8621910.
175. Lin, D.; Wu, J.; Yuan, Q.; Zheng, Z. T-EDGE: Temporal WEighted MultiDiGraph Embedding for Ethereum Transaction Network Analysis. *Front. Phys.* **2020**, *8*, 204. https://doi.org/10.3389/fphy.2020.00204.
176. Lee, O.; Jung, J.J.; Kim, J. Learning Hierarchical Representations of Stories by Using Multi-Layered Structures in Narrative Multimedia. *Sensors* **2020**, *20*, 1978. https://doi.org/10.3390/s20071978.
177. Lee, O.; Jung, J.J. Story Embedding: Learning Distributed Representations of Stories based on Character Networks (Extended Abstract). In Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020), Virtual Event, 7–15 January 2021; ijcai.org: Yokohama, Japan, 2020; pp. 5070–5074. https://doi.org/10.24963/ijcai.2020/709.
178. Chen, H.; Perozzi, B.; Hu, Y.; Skiena, S. HARP: Hierarchical Representation Learning for Networks. In Proceedings of the 32nd Conference on Artificial Intelligence (AAAI-18), New Orleans, Louisiana, USA, 2–7 February 2018; AAAI Press: New Orleans, LA, USA, 2018; pp. 2127–2134.
179. Tang, J.; Qu, M.; Mei, Q. PTE: Predictive Text Embedding through Large-scale Heterogeneous Text Networks. In Proceedings of the 21th International Conference on Knowledge Discovery and Data Mining (SIGKDD 2015), Sydney, Australia, 10–13 August 2015; ACM: Sydney, Australia, 2015; pp. 1165–1174. https://doi.org/10.1145/2783258.2783307.
180. Fu, T.; Lee, W.; Lei, Z. HIN2Vec: Explore Meta-paths in Heterogeneous Information Networks for Representation Learning. In Proceedings of the Conference on Information and Knowledge Management (CIKM 2017), Singapore, 6–10 November 2017; ACM: Singapore, 2017; pp. 1797–1806. https://doi.org/10.1145/3132847.3132953.
181. Kullback, S.; Leibler, R.A. On Information and Sufficiency. *Ann. Math. Stat.* **1951**, *22*, 79–86. https://doi.org/10.1214/aoms/1177729694.
182. Wei, X.; Xu, L.; Cao, B.; Yu, P.S. Cross View Link Prediction by Learning Noise-resilient Representation Consensus. In Proceedings of the 26th International Conference on World Wide Web (WWW 2017), Perth, Australia, 3–7 April 2017; ACM: Perth, Australia, 2017; pp. 1611–1619. https://doi.org/10.1145/3038912.3052575.
183. Liu, L.; Cheung, W.K.; Li, X.; Liao, L. Aligning Users across Social Networks Using Network Embedding. In Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016), New York, NY, USA, 9–15 July 2016; IJCAI/AAAI Press: New York, NY, USA, 2016; pp. 1774–1780.
184. Wang, S.; Tang, J.; Aggarwal, C.C.; Chang, Y.; Liu, H. Signed Network Embedding in Social Media. In Proceedings of the International Conference on Data Mining (SIAM 2017), Houston, Texas, USA, 27–29 April 2017; SIAM: Houston, TX, USA, 2017; pp. 327–335. https://doi.org/10.1137/1.9781611974973.37.
185. Huang, Z.; Mamoulis, N. Heterogeneous Information Network Embedding for Meta Path based Proximity. *CoRR* **2017**, arXiv:1701.05291. http://arxiv.org/abs/1701.05291.
186. Zhang, Q.; Wang, H. Not All Links Are Created Equal: An Adaptive Embedding Approach for Social Personalized Ranking. In Proceedings of the 39th International Conference on Research and Development in Information Retrieval (SIGIR 2016), Pisa, Italy, 17–21 July 2016; ACM: Pisa, Italy, 2016; pp. 917–920. https://doi.org/10.1145/2911451.2914740.
187. Wang, J.; Lu, Z.; Song, G.; Fan, Y.; Du, L.; Lin, W. Tag2Vec: Learning Tag Representations in Tag Networks. In Proceedings of The World Wide Web Conference (WWW 2019), San Francisco, CA, USA, 13–17 May 2019; Association for Computing Machinery: New York, NY, USA, 2019; p. 3314–3320. https://doi.org/10.1145/3308558.3313622.
188. Fu, G.; Yuan, B.; Duan, Q.; Yao, X. Representation Learning for Heterogeneous Information Networks via Embedding Events. In Proceedings of the 26th International Conference (ICONIP 2019), Sydney, Australia, 12–15 December 2019; Springer: Sydney, Australia, 2019; Volume 11953; pp. 327–339. https://doi.org/10.1007/978-3-030-36708-4_27.
189. Wu, X.; Pang, H.; Fan, Y.; Linghu, Y.; Luo, Y. ProbWalk: A random walk approach in weighted graph embedding. In Proceedings of the 10th International Conference of Information and Communication Technology (ICICT 2020), Wuhan, China, 13–15 November 2020; Elsevier Procedia: Wuhan, China, 2021; Volume 183; pp. 683–689. https://doi.org/https://doi.org/10.1016/j.procs.2021.02.115.
190. Li, Q.; Cao, Z.; Zhong, J.; Li, Q. Graph Representation Learning with Encoding Edges. *Neurocomputing* **2019**, *361*, 29–39. https://doi.org/10.1016/j.neucom.2019.07.076.

191. Li, Q.; Zhong, J.; Li, Q.; Cao, Z.; Wang, C. Enhancing Network Embedding with Implicit Clustering. In Proceedings of the 24th Database Systems for Advanced Applications (DASFAA 2019), Chiang Mai, Thailand, 22–25 April 2019; Springer: Chiang Mai, Thailand, 2019; Volume 11446; pp. 452–467. https://doi.org/10.1007/978-3-030-18576-3_27.
192. Gao, H.; Huang, H. Deep Attributed Network Embedding. In Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018), Stockholm, Sweden, 13–19 July 2018; Lang, J., Ed.; ijcai.org: Stockholm, Sweden, 2018; pp. 3364–3370. https://doi.org/10.24963/ijcai.2018/467.
193. Sun, X.; Guo, J.; Ding, X.; Liu, T. A General Framework for Content-enhanced Network Representation Learning. *CoRR* **2016**, arXiv:1610.02906. http://arxiv.org/abs/1610.02906.
194. Zhang, D.; Yin, J.; Zhu, X.; Zhang, C. Homophily, Structure, and Content Augmented Network Representation Learning. In Proceedings of the 16th International Conference on Data Mining (ICDM 2016), Barcelona, Spain, 12–15 December 2016; Bonchi, F., Domingo-Ferrer, J., Baeza-Yates, R., Zhou, Z., Wu, X., Eds.; IEEE Computer Society: Barcelona, Spain, 2016; pp. 609–618. https://doi.org/10.1109/ICDM.2016.0072.
195. Chen, H.; Anantharam, A.R.; Skiena, S. DeepBrowse: Similarity-Based Browsing Through Large Lists (Extended Abstract). In Proceedings of the 10th International Conference on Similarity Search and Applications (SISAP 2017), Munich, Germany, 4–6 October 2017; Beecks, C., Borutta, F., Kröger, P., Seidl, T., Eds.; Springer: Munich, Germany, 2017; Volume 10609, pp. 300–314. https://doi.org/10.1007/978-3-319-68474-1_21.
196. Cao, S.; Lu, W.; Xu, Q. Deep Neural Networks for Learning Graph Representations. In Proceedings of the 30th Conference on Artificial Intelligence (AAAI 2016), Phoenix, Arizona, USA, 12–17 February 2016; Schuurmans, D., Wellman, M.P., Eds.; AAAI Press: Phoenix, AZ, USA, 2016; pp. 1145–1152.
197. Shen, X.; Chung, F. Deep Network Embedding for Graph Representation Learning in Signed Networks. *Trans. Cybern.* **2020**, *50*, 1556–1568. https://doi.org/10.1109/TCYB.2018.2871503.
198. Goyal, P.; Kamra, N.; He, X.; Liu, Y. DynGEM: Deep Embedding Method for Dynamic Graphs. *CoRR* **2018**, arXiv:1805.11273. http://arxiv.org/abs/1805.11273.
199. Yu, W.; Cheng, W.; Aggarwal, C.C.; Zhang, K.; Chen, H.; Wang, W. NetWalk: A Flexible Deep Embedding Approach for Anomaly Detection in Dynamic Networks. In Proceedings of the 24th International Conference on Knowledge Discovery & Data Mining (KDD 2018), London, UK, 19–23 August 2018; ACM: London, UK, 2018; pp. 2672–2681. https://doi.org/10.1145/3219819.3220024.
200. Yu, Z.; Kuang, Z.; Liu, J.; Chen, H.; Zhang, J.; You, J.; Wong, H.; Han, G. Adaptive Ensembling of Semi-Supervised Clustering Solutions. *IEEE Trans. Knowl. Data Eng.* **2017**, *29*, 1577–1590. https://doi.org/10.1109/TKDE.2017.2695615.
201. Goyal, P.; Chhetri, S.R.; Canedo, A. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowl.-Based Syst.* **2020**, *187*, 104816. https://doi.org/10.1016/j.knosys.2019.06.024.
202. Taheri, A.; Gimpel, K.; Berger-Wolf, T. Learning graph representations with recurrent neural network autoencoders. *KDD Deep Learn. Day*. Available online: https://www.kdd.org/kdd2018/files/deep-learning-day/DLDay18_paper_27.pdf (accessed on 16 April 2023).
203. Khoshraftar, S.; Mahdavi, S.; An, A.; Hu, Y.; Liu, J. Dynamic Graph Embedding via LSTM History Tracking. In Proceedings of the International Conference on Data Science and Advanced Analytics (DSAA 2019), Washington, DC, USA, 5–8 October 2019; IEEE: Washington, DC, USA, 2019; pp. 119–127. https://doi.org/10.1109/DSAA.2019.00026.
204. Chen, J.; Zhang, J.; Xu, X.; Fu, C.; Zhang, D.; Zhang, Q.; Xuan, Q. E-LSTM-D: A Deep Learning Framework for Dynamic Network Link Prediction. *IEEE Trans. Syst. Man Cybern.* **2021**, *51*, 3699–3712. https://doi.org/10.1109/TSMC.2019.2932913.
205. Taheri, A.; Gimpel, K.; Berger-Wolf, T.Y. Sequence-to-sequence modeling for graph representation learning. *Appl. Netw. Sci.* **2019**, *4*, 68.1–68.26. https://doi.org/10.1007/s41109-019-0174-8.
206. Taheri, A.; Gimpel, K.; Berger-Wolf, T.Y. Learning to Represent the Evolution of Dynamic Graphs with Recurrent Models. In Proceedings of the Companion of The World Wide Web Conference (WWW 2019), San Francisco, CA, USA, 13–17 May 2019; ACM: San Francisco, CA, USA, 2019; pp. 301–307. https://doi.org/10.1145/3308560.3316581.
207. Zhang, C.; Huang, C.; Yu, L.; Zhang, X.; Chawla, N.V. Camel: Content-Aware and Meta-path Augmented Metric Learning for Author Identification. In Proceedings of the World Wide Web Conference on World Wide Web (WWW 2018), Lyon, France, 23–27 April 2018; Champin, P.; Gandon, F.; Lalmas, M.; Ipeirotis, P.G., Eds.; ACM: Lyon, France, 2018; pp. 709–718. https://doi.org/10.1145/3178876.3186152.
208. Park, C.; Kim, D.; Zhu, Q.; Han, J.; Yu, H. Task-Guided Pair Embedding in Heterogeneous Network. In Proceedings of the 28th International Conference on Information and Knowledge Management (CIKM 2019), Beijing, China, 3–7 November 2019; ACM: Beijing, China, 2019; pp. 489–498. https://doi.org/10.1145/3357384.3357982.
209. Hu, W.; Liu, B.; Gomes, J.; Zitnik, M.; Liang, P.; Pande, V.S.; Leskovec, J. Strategies for Pre-training Graph Neural Networks. In Proceedings of the 8th International Conference on Learning Representations (ICLR 2020), Addis Ababa, Ethiopia, 26–30 April 2020; OpenReview.net: Addis Ababa, Ethiopia, 2020.
210. Li, R.; Wang, S.; Zhu, F.; Huang, J. Adaptive Graph Convolutional Neural Networks. In Proceedings of the 32nd Conference on Artificial Intelligence (AAAI-18), New Orleans, Louisiana, USA, 2–7 February 2018; AAAI Press: New Orleans, LA, USA, 2018; pp. 3546–3553.
211. Zhuang, C.; Ma, Q. Dual Graph Convolutional Networks for Graph-Based Semi-Supervised Classification. In Proceedings of the Conference on World Wide Web (WWW 2018), Lyon, France, 23–27 April 2018; ACM: Lyon, France, 2018; pp. 499–508. https://doi.org/10.1145/3178876.3186116.

212. Wang, Y.; Wu, X.; Wu, L. Differential Privacy Preserving Spectral Graph Analysis. In Proceedings of the 17th Advances in Knowledge Discovery and Data Mining, Pacific-Asia Conference (PAKDD 2013), Gold Coast, Australia, 14–17 April 2013; Springer: Gold Coast, Australia, 2013; Volume 7819, pp. 329–340. https://doi.org/10.1007/978-3-642-37456-2_28.
213. NT, H.; Maehara, T. Revisiting Graph Neural Networks: All We Have is Low-Pass Filters. *CoRR* **2019**, arXiv:1905.09550. http://arxiv.org/abs/1905.09550.
214. Wu, F.; Zhang, T.; de Souza, A.H., Jr.; Fifty, C.; Yu, T.; Weinberger, K.Q. Simplifying Graph Convolutional Networks. In Proceedings of the 36th International Conference on Machine Learning (ICML 2019), Long Beach, CA, USA, 9–15 June 2019; Chaudhuri, K., Salakhutdinov, R., Eds.; PMLR: Long Beach, CA, USA, 2019; Volume 97; pp. 6861–6871.
215. Yang, Z.; Cohen, W.W.; Salakhutdinov, R. Revisiting Semi-Supervised Learning with Graph Embeddings. In Proceedings of the 33nd International Conference on Machine Learning (ICML 2016), New York City, NY, USA, 19–24 June 2016; JMLR.org: New York City, NY, USA, 2016; Volume 48; pp. 40–48.
216. Levie, R.; Monti, F.; Bresson, X.; Bronstein, M.M. CayleyNets: Graph Convolutional Neural Networks with Complex Rational Spectral Filters. *IEEE Trans. Signal Process.* **2019**, *67*, 97–109. https://doi.org/10.1109/TSP.2018.2879624.
217. Liu, X.; Xia, G.; Lei, F.; Zhang, Y.; Chang, S. Higher-Order Graph Convolutional Networks With Multi-Scale Neighborhood Pooling for Semi-Supervised Node Classification. *IEEE Access* **2021**, *9*, 31268–31275. https://doi.org/10.1109/ACCESS.2021.3060173.
218. Yuan, S.; Wang, C.; Jiang, Q.; Ma, J. Community Detection with Graph Neural Network using Markov Stability. In Proceedings of the International Conference on Artificial Intelligence in Information and Communication (ICAIIC 2022), Jeju Island, Republic of Korea, 21–24 February 2022; IEEE: Jeju Island, Republic of Korea, 2022; pp. 437–442. https://doi.org/10.1109/ICAIIC54071.2022.9722614.
219. Zhou, X.; Shen, Y.; Zhu, Y.; Huang, L. Predicting Multi-Step Citywide Passenger Demands Using Attention-Based Neural Networks. In Proceedings of the 11th International Conference on Web Search and Data Mining (WSDM 2018), Marina Del Rey, CA, USA, 5–9 February 2018; ACM: New York, NY, USA, 2018; pp. 736–744. https://doi.org/10.1145/3159652.3159682.
220. Zhang, X.; Huang, C.; Xu, Y.; Xia, L.; Dai, P.; Bo, L.; Zhang, J.; Zheng, Y. Traffic Flow Forecasting with Spatial-Temporal Graph Diffusion Network. In Proceedings of the 35th Conference on Artificial Intelligence (AAAI 2021), Virtual Event, 2–9 February 2021; AAAI Press: Virtual Event, 2021; pp. 15008–15015.
221. Zhao, Q.; Ye, Z.; Chen, C.; Wang, Y. Persistence Enhanced Graph Neural Network. In Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS 2020), Palermo, Italy, 26–28 August 2020; PMLR: Palermo, Italy, 2020; pp. 2896–2906.
222. Gilmer, J.; Schoenholz, S.S.; Riley, P.F.; Vinyals, O.; Dahl, G.E. Neural Message Passing for Quantum Chemistry. In Proceedings of the 34th International Conference on Machine Learning (ICML 2017), Sydney, Australia, 6–11 August 2017; PMLR: Sydney, Australia, 2017; Volume 70; pp. 1263–1272.
223. Chang, J.; Gu, J.; Wang, L.; Meng, G.; Xiang, S.; Pan, C. Structure-Aware Convolutional Neural Networks. In Proceedings of the 31st Annual Conference on Neural Information Processing Systems 2018 (NeurIPS 2018), Montréal, QC, Canada, 3–8 December 2018; NeurIPS: Montréal, QC, Canada, 2018; pp. 11–20.
224. Chen, J.; Zhu, J.; Song, L. Stochastic Training of Graph Convolutional Networks with Variance Reduction. In Proceedings of the 35th International Conference on Machine Learning (ICML 2018), Stockholm, Sweden, 10–15 July 2018; PMLR: Stockholm, Sweden, 2018; Volume 80; pp. 941–949.
225. Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W.L.; Leskovec, J. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In Proceedings of the 24th International Conference on Knowledge Discovery & Data Mining (KDD 2018), London, UK, 19–23 August 2018; Guo, Y., Farooq, F., Eds.; ACM: London, UK, 2018; pp. 974–983. https://doi.org/10.1145/3219819.3219890.
226. Wang, H.; Lian, D.; Ge, Y. Binarized Collaborative Filtering with Distilling Graph Convolutional Networks. In Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019), Macao, China, 10–16 August 2019; AAAI Press: Macao, China, 2019; pp. 4802–4808.
227. Hjelm, R.D.; Fedorov, A.; Lavoie-Marchildon, S.; Grewal, K.; Bachman, P.; Trischler, A.; Bengio, Y. Learning deep representations by mutual information estimation and maximization. In Proceedings of the 7th International Conference on Learning Representations (ICLR 2019), New Orleans, LA, USA, 6–9 May 2019; OpenReview.net: New Orleans, LA, USA, 2019.
228. Qu, M.; Tang, J.; Shang, J.; Ren, X.; Zhang, M.; Han, J. An Attention-based Collaboration Framework for Multi-View Network Representation Learning. In Proceedings of the 26th Conference on Information and Knowledge Management (CIKM 2017), Singapore, 6–10 November 2017; ACM: Singapore, 2017; pp. 1767–1776. https://doi.org/10.1145/3132847.3133021.
229. Tran, D.V.; Navarin, N.; Sperduti, A. On Filter Size in Graph Convolutional Networks. In Proceedings of the Symposium Series on Computational Intelligence (SSCI 2018), Bangalore, India, 18–21 November 2018; IEEE: Bangalore, India, 2018; pp. 1534–1541. https://doi.org/10.1109/SSCI.2018.8628758.
230. Li, Y.; Yu, R.; Shahabi, C.; Liu, Y. Graph Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. *CoRR* **2017**, arXiv:1707.01926. http://arxiv.org/abs/1707.01926.
231. Lo, W.W.; Layeghy, S.; Sarhan, M.; Gallagher, M.; Portmann, M. E-GraphSAGE: A Graph Neural Network based Intrusion Detection System for IoT. In Proceedings of the Network Operations and Management Symposium (NOMS 2022), Budapest, Hungary, 25–29 April 2022; IEEE: Budapest, Hungary, 2022; pp. 1–9. https://doi.org/10.1109/NOMS54207.2022.9789878.

232. Cai, T.; Luo, S.; Xu, K.; He, D.; Liu, T.; Wang, L. GraphNorm: A Principled Approach to Accelerating Graph Neural Network Training. In Proceedings of the 38th International Conference on Machine Learning (ICML 2021), Virtual Event, 18–24 July 2021; PMLR: Virtual Event, 2021; Volume 139, pp. 1204–1215.
233. Le, T.; Bertolini, M.; Noé, F.; Clevert, D. Parameterized Hypercomplex Graph Neural Networks for Graph Classification. In Proceedings of the 30th Artificial Neural Networks and Machine Learning (ICANN 2021), Bratislava, Slovakia, 14–17 September 2021; Farkas, I., Masulli, P., Otte, S., Wermter, S., Eds.; Springer: Bratislava, Slovakia, 2021; Volume 12893, pp. 204–216. https://doi.org/10.1007/978-3-030-86365-4_17.
234. Yadati, N.; Nimishakavi, M.; Yadav, P.; Nitin, V.; Louis, A.; Talukdar, P.P. HyperGCN: A New Method For Training Graph Convolutional Networks on Hypergraphs. In Proceedings of the 32nd Annual Conference on Neural Information Processing Systems 2019 (NeurIPS 2019), Vancouver, BC, Canada, 8–14 December 2019; Wallach, H.M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.B., Garnett, R., Eds.; NeurIPS: Vancouver, BC, Canada, 2019; pp. 1509–1520.
235. Zhang, W.; Liu, H.; Liu, Y.; Zhou, J.; Xiong, H. Semi-Supervised Hierarchical Recurrent Graph Neural Network for City-Wide Parking Availability Prediction. In Proceedings of the 34th Conference on Artificial Intelligence (AAAI 2020), New York, NY, USA, 7–12 February 2020; AAAI Press: New York, NY, USA, 2020; pp. 1186–1193.
236. Peng, H.; Wang, H.; Du, B.; Bhuiyan, M.Z.A.; Ma, H.; Liu, J.; Wang, L.; Yang, Z.; Du, L.; Wang, S.; et al. Spatial temporal incidence dynamic graph neural networks for traffic flow forecasting. *Inf. Sci.* **2020**, *521*, 277–290. https://doi.org/https://doi.org/10.1016/j.ins.2020.01.043.
237. Cheng, D.; Xiang, S.; Shang, C.; Zhang, Y.; Yang, F.; Zhang, L. Spatio-Temporal Attention-Based Neural Network for Credit Card Fraud Detection. In Proceedings of the 34th Conference on Artificial Intelligence (AAAI 2020), New York, NY, USA, 7–12 February 2020; AAAI Press: New York, NY, USA, 2020; pp. 362–369.
238. Xie, Z.; Lv, W.; Huang, S.; Lu, Z.; Du, B.; Huang, R. Sequential Graph Neural Network for Urban Road Traffic Speed Prediction. *IEEE Access* **2020**, *8*, 63349–63358. https://doi.org/10.1109/ACCESS.2019.2915364.
239. Yao, H.; Wu, F.; Ke, J.; Tang, X.; Jia, Y.; Lu, S.; Gong, P.; Ye, J.; Li, Z. Deep Multi-View Spatial-Temporal Network for Taxi Demand Prediction. In Proceedings of the 32nd Conference on Artificial Intelligence (AAAI-18), New Orleans, LA, USA, 2–7 February 2018; AAAI Press: New Orleans, LA, USA, 2018; pp. 2588–2595.
240. Zhang, J.; Zheng, Y.; Qi, D. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. In Proceedings of the 31st Conference on Artificial Intelligence (AAAI 2017), San Francisco, CA, USA, 4–9 February 2017; Singh, S., Markovitch, S., Eds.; AAAI Press: San Francisco, CA, USA, 2017; pp. 1655–1661.
241. Schlichtkrull, M.S.; Kipf, T.N.; Bloem, P.; van den Berg, R.; Titov, I.; Welling, M. Modeling Relational Data with Graph Convolutional Networks. In Proceedings of the 15th The Semantic Web International Conference (ESWC 2018), Heraklion, Greece, 3–7 June 2018; Springer: Heraklion, Greece, 2018; Volume 10843, pp. 593–607. https://doi.org/10.1007/978-3-319-93417-4_38.
242. Jing, B.; Park, C.; Tong, H. HDMI: High-order Deep Multiplex Infomax. In Proceedings of the The Web Conference (WWW '21), Ljubljana, Slovenia, 19–23 April 2021; Leskovec, J., Grobelnik, M., Najork, M., Tang, J., Zia, L., Eds.; ACM/IW3C2: Ljubljana, Slovenia, 2021; pp. 2414–2424. https://doi.org/10.1145/3442381.3449971.
243. Park, C.; Kim, D.; Han, J.; Yu, H. Unsupervised Attributed Multiplex Network Embedding. In Proceedings of the 34th Conference on Artificial Intelligence (AAAI 2020), New York, NY, USA, 7–12 February 2020; AAAI Press: New York, NY, USA, 2020; pp. 5371–5378.
244. Wei, H.; Hu, G.; Bai, W.; Xia, S.; Pan, Z. Lifelong representation learning in dynamic attributed networks. *Neurocomputing* **2019**, *358*, 1–9. https://doi.org/10.1016/j.neucom.2019.05.038.
245. Pareja, A.; Domeniconi, G.; Chen, J.; Ma, T.; Suzumura, T.; Kanezashi, H.; Kaler, T.; Schardl, T.B.; Leiserson, C.E. EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. In Proceedings of the 34th Conference on Artificial Intelligence (AAAI 2020), New York, NY, USA, 7–12 February 2020; AAAI Press: New York, NY, USA, 2020; pp. 5363–5370.
246. Sun, F.; Hoffmann, J.; Verma, V.; Tang, J. InfoGraph: Unsupervised and Semi-supervised Graph-Level Representation Learning via Mutual Information Maximization. In Proceedings of the 8th International Conference on Learning Representations (ICLR 2020), Addis Ababa, Ethiopia, 26–30 April 2020; OpenReview.net: Addis Ababa, Ethiopia, 2020.
247. Hu, Z.; Dong, Y.; Wang, K.; Chang, K.; Sun, Y. GPT-GNN: Generative Pre-Training of Graph Neural Networks. In Proceedings of the 26th SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2020), Virtual Event, CA, USA, 6–10 July 2020; ACM: Virtual Event, CA, USA, 2020; pp. 1857–1867. https://doi.org/10.1145/3394486.3403237.
248. Karypis, G.; Kumar, V. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.* **1998**, *20*, 359–392. https://doi.org/10.1137/S1064827595287997.
249. Zeng, H.; Zhou, H.; Srivastava, A.; Kannan, R.; Prasanna, V.K. Accurate, Efficient and Scalable Graph Embedding. In Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2019), Rio de Janeiro, Brazil, 20–24 May 2019; IEEE: Rio de Janeiro, Brazil, 2019; pp. 462–471. https://doi.org/10.1109/IPDPS.2019.00056.
250. Huang, W.; Zhang, T.; Rong, Y.; Huang, J. Adaptive Sampling Towards Fast Graph Representation Learning. In Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, QC, Canada, 3–8 December 2018; NeurIPS: Montréal, QC, Canada, 2018; pp. 4563–4572.
251. Bahdanau, D.; Cho, K.; Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. In Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015), San Diego, CA, USA, 7–9 May 2015; ICLR: San Diego, CA, USA,2015.

252. Zhao, L.; Akoglu, L. PairNorm: Tackling Oversmoothing in GNNs. In Proceedings of the 8th International Conference on Learning Representations (ICLR 2020), Addis Ababa, Ethiopia, 26–30 April 2020; OpenReview.net: Addis Ababa, Ethiopia, 2020.
253. Ma, L.; Rabbany, R.; Romero-Soriano, A. Graph Attention Networks with Positional Embeddings. In Proceedings of the 25th Advances in Knowledge Discovery and Data Mining Pacific-Asia Conference (PAKDD 2021), Virtual Event, 11–14 May 2021; Karlapalem, K., Cheng, H., Ramakrishnan, N., Agrawal, R.K., Reddy, P.K., Srivastava, J., Chakraborty, T., Eds.; Springer: Virtual Event, 2021; Volume 12712, pp. 514–527. https://doi.org/10.1007/978-3-030-75762-5_41.
254. Huang, B.; Carley, K.M. Syntax-Aware Aspect Level Sentiment Classification with Graph Attention Networks. In Proceedings of the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP 2019), Hong Kong, China, 3–7 November 2019; Inui, K., Jiang, J., Ng, V., Wan, X., Eds.; Association for Computational Linguistics: Hong Kong, China, 2019; pp. 5468–5476. https://doi.org/10.18653/v1/D19-1549.
255. Zhang, J.; Shi, X.; Xie, J.; Ma, H.; King, I.; Yeung, D. GaAN: Gated Attention Networks for Learning on Large and Spatiotemporal Graphs. In Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence (UAI 2018), Monterey, CA, USA, 6–10 August 2018; AUAI Press: Monterey, CA, USA, 2018; pp. 339–349.
256. Haonan, L.; Huang, S.H.; Ye, T.; Xiuyan, G. Graph Star Net for Generalized Multi-Task Learning. *arXiv* **2019**, arXiv:1906.12330, http://arxiv.org/abs/1906.12330.
257. Abu-El-Haija, S.; Perozzi, B.; Al-Rfou, R.; Alemi, A.A. Watch Your Step: Learning Node Embeddings via Graph Attention. In Proceedings of the 31st Advances in Neural Information Processing Systems (NeurIPS 2018), Montréal, QC, Canada, 3–8 December 2018; NeurIPS: Montréal, QC, Canada, 2018; pp. 9198–9208.
258. Kim, D.; Oh, A. How to Find Your Friendly Neighborhood: Graph Attention Design with Self-Supervision. In Proceedings of the 9th International Conference on Learning Representations (ICLR 2021), Vienna, Austria, 3–7 May 2021; OpenReview.net: Virtual Event, 2021.
259. Wang, G.; Ying, R.; Huang, J.; Leskovec, J. Improving Graph Attention Networks with Large Margin-based Constraints. *CoRR* **2019**, arXiv:1910.11945. http://arxiv.org/abs/1910.11945.
260. Zhu, Y.; Wang, J.; Zhang, J.; Zhang, K. Node Embedding and Classification with Adaptive Structural Fingerprint. *Neurocomputing* **2022**, *502*, 196–208. https://doi.org/10.1016/j.neucom.2022.05.073.
261. Yang, Y.; Qiu, J.; Song, M.; Tao, D.; Wang, X. Distilling Knowledge From Graph Convolutional Networks. In Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR 2020), Seattle, WA, USA, 13–19 June 2020; Computer Vision Foundation/IEEE: Seattle, WA, USA, 2020; pp. 7072–7081. https://doi.org/10.1109/CVPR42600.2020.00710.
262. Nathani, D.; Chauhan, J.; Sharma, C.; Kaul, M. Learning Attention-based Embeddings for Relation Prediction in Knowledge Graphs. In Proceedings of the 57th Conference of the Association for Computational Linguistics (ACL 2019), Florence, Italy, 28 July–2 August 2019; ACL: Florence, Italy, 2019; pp. 4710–4723. https://doi.org/10.18653/v1/p19-1466.
263. Sun, X.; Wang, Z.; Yang, J.; Liu, X. Deepdom: Malicious domain detection with scalable and heterogeneous graph convolutional networks. *Comput. Secur.* **2020**, *99*, 102057. https://doi.org/10.1016/j.cose.2020.102057.
264. Xue, H.; Yang, L.; Rajan, V.; Jiang, W.; Wei, Y.; Lin, Y. Multiplex Bipartite Network Embedding Using Dual Hypergraph Convolutional Networks. In Proceedings of the Web Conference 2021 (WWW 2021), Ljubljana, Slovenia, 19–23 April 2021; ACM: New York, NY, USA, 2021; pp. 1649–1660. https://doi.org/10.1145/3442381.3449954.
265. Wang, Y.; Duan, Z.; Liao, B.; Wu, F.; Zhuang, Y. Heterogeneous Attributed Network Embedding with Graph Convolutional Networks. In Proceedings of the 33rd Conference on Artificial Intelligence (AAAI 2019), Honolulu, HI, USA, 27 January–1 February 2019; AAAI Press: Honolulu, HI, USA, 2019; pp. 10061–10062. https://doi.org/10.1609/aaai.v33i01.330110061.
266. Yu, P.; Fu, C.; Yu, Y.; Huang, C.; Zhao, Z.; Dong, J. Multiplex Heterogeneous Graph Convolutional Network. In Proceedings of the 28th Conference on Knowledge Discovery and Data Mining (KDD 2022), Washington, DC, USA, 14–18 August 2022; Zhang, A., Rangwala, H., Eds.; ACM: Washington, DC, USA, 2022; pp. 2377–2387. https://doi.org/10.1145/3534678.3539482.
267. Wang, H.; Zhao, M.; Xie, X.; Li, W.; Guo, M. Knowledge Graph Convolutional Networks for Recommender Systems. In Proceedings of the World Wide Web Conference (WWW 2019), San Francisco, CA, USA, 13–17 May 2019; Liu, L., White, R.W., Mantrach, A., Silvestri, F., McAuley, J.J., Baeza-Yates, R., Zia, L., Eds.; ACM: San Francisco, CA, USA, 2019; pp. 3307–3313. https://doi.org/10.1145/3308558.3313417.
268. Mao, C.; Yao, L.; Luo, Y. MedGCN: Medication recommendation and lab test imputation via graph convolutional networks. *J. Biomed. Inform.* **2022**, *127*, 104000. https://doi.org/10.1016/j.jbi.2022.104000.
269. Xu, F.; Lian, J.; Han, Z.; Li, Y.; Xu, Y.; Xie, X. Relation-Aware Graph Convolutional Networks for Agent-Initiated Social E-Commerce Recommendation. In Proceedings of the 28th International Conference on Information and Knowledge Management (CIKM 2019), Beijing, China, 3–7 November 2019; Zhu, W., Tao, D., Cheng, X., Cui, P., Rundensteiner, E.A., Carmel, D., He, Q., Yu, J.X., Eds.; ACM: Beijing, China, 2019; pp. 529–538. https://doi.org/10.1145/3357384.3357924.
270. Wang, C.; Pan, S.; Long, G.; Zhu, X.; Jiang, J. MGAE: Marginalized Graph Autoencoder for Graph Clustering. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, 6–10 November 2017; ACM: Singapore, 2017; pp. 889–898. https://doi.org/10.1145/3132847.3132967.
271. Simonovsky, M.; Komodakis, N. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. In Proceedings of the 27th International Conference on Artificial Neural Networks (ICANN 2018), Rhodes, Greece, 4–7 October 2018; Springer: Rhodes, Greece, 2018. http://arxiv.org/abs/1802.03480.

272. Cao, N.D.; Kipf, T. MolGAN: An implicit generative model for small molecular graphs. *CoRR* **2018**, arXiv:1805.11973. http://arxiv.org/abs/1805.11973.
273. Suganuma, M.; Ozay, M.; Okatani, T. Exploiting the Potential of Standard Convolutional Autoencoders for Image Restoration by Evolutionary Search. In Proceedings of the 35th International Conference on Machine Learning (ICML 2018), Stockholmsmässan, Stockholm, Sweden, 10–15 July 2018; PMLR: Stockholm, Sweden, 2018; Volume 80, pp. 4778–4787.
274. Kingma, D.P.; Welling, M. Auto-Encoding Variational Bayes. In Proceedings of the 2nd International Conference on Learning Representations (ICLR 2014), Banff, AB, Canada, 14–16 April 2014;ICLR: Banff, AB, Canada, 2014.
275. Wu, G.; Lin, S.; Shao, X.; Zhang, P.; Qiao, J. QPGCN: Graph convolutional network with a quadratic polynomial filter for overcoming over-smoothing. *Appl. Intell.* **2022**, *53*, 7216–7231.
276. Shi, L.; Wu, W.; Hu, W.; Zhou, J.; Chen, J.; Zheng, W.; He, L. DualGCN: An Aspect-Aware Dual Graph Convolutional Network for review-based recommender. *Knowl.-Based Syst.* **2022**, *242*, 108359. https://doi.org/10.1016/j.knosys.2022.108359.
277. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention is All you Need. In Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NeurIPS 2017), Long Beach, CA, USA, 4–9 December 2017; NeurIPS: Long Beach, CA, USA, 2017; pp. 5998–6008.
278. Devlin, J.; Chang, M.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019), Minneapolis, MN, USA, 2–7 June 2019; ACL: Minneapolis, MN, USA, 2019; pp. 4171–4186. https://doi.org/10.18653/v1/n19-1423.
279. Joshi, M.; Chen, D.; Liu, Y.; Weld, D.S.; Zettlemoyer, L.; Levy, O. SpanBERT: Improving Pre-training by Representing and Predicting Spans. *Trans. Assoc. Comput. Linguist.* **2020**, *8*, 64–77. https://doi.org/10.1162/tacl_a_00300.
280. Chen, H.; Wang, Y.; Guo, T.; Xu, C.; Deng, Y.; Liu, Z.; Ma, S.; Xu, C.; Xu, C.; Gao, W. Pre-Trained Image Processing Transformer. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2021), Virtual Event, 19–25 June 2021; IEEE: Virtual Event, 2021; pp. 12299–12310. https://doi.org/10.1109/CVPR46437.2021.01212.
281. Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An Image is Worth 16 $\times$ 16 Words: Transformers for Image Recognition at Scale. In Proceedings of the 9th International Conference on Learning Representations (ICLR 2021), Vienna, Austria, 3–7 May 2021; OpenReview.net: Vienna, Austria, 2021.
282. Cai, D.; Lam, W. Graph Transformer for Graph-to-Sequence Learning. In Proceedings of the 34th Conference on Artificial Intelligence (AAAI 2020), New York, NY, USA, 7–12 February 2020; AAAI Press: New York, NY, USA, 2020; pp. 7464–7471.
283. Jeon, H.J.; Choi, G.S.; Cho, S.Y.; Lee, H.; Ko, H.Y.; Jung, J.J.; Lee, O.J.; Yi, M.Y. Learning contextual representations of citations via graph transformer. In Proceedings of the 2nd International Conference on Human-centered Artificial Intelligence (Computing4Human 2021), Da Nang, Vietnam, 28–29 October 2021; ceur-ws: Da Nang, Vietnam, 2021.
284. Kreuzer, D.; Beaini, D.; Hamilton, W.L.; Létourneau, V.; Tossou, P. Rethinking Graph Transformers with Spectral Attention. In Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS 2021), Virtual Event, 6–14 December 2021; NeurIPS: Virtual Event, 2021; pp. 21618–21629.
285. Hu, Z.; Dong, Y.; Wang, K.; Sun, Y. Heterogeneous Graph Transformer. In Proceedings of the Web Conference (WWW 2020), Taipei, Taiwan, 20–24 April 2020; ACM/IW3C2: Taipei, Taiwan, 2020; pp. 2704–2710. https://doi.org/10.1145/3366423.3380027.
286. Khoo, L.M.S.; Chieu, H.L.; Qian, Z.; Jiang, J. Interpretable Rumor Detection in Microblogs by Attending to User Interactions. In Proceedings of the 34th Conference on Artificial Intelligence(AAAI 2020), New York, NY, USA, 7–12 February 2020; AAAI Press: New York, NY, USA, 2020; pp. 8783–8790.
287. Min, E.; Rong, Y.; Xu, T.; Bian, Y.; Luo, D.; Lin, K.; Huang, J.; Ananiadou, S.; Zhao, P. Neighbour Interaction based Click-Through Rate Prediction via Graph-masked Transformer. In Proceedings of the 45th International Conference on Research and Development in Information Retrieval (SIGIR 2022), Madrid, Spain, 11–15 July 2022; ACM: Madrid, Spain, 2022; pp. 353–362. https://doi.org/10.1145/3477495.3532031.
288. Shaw, P.; Uszkoreit, J.; Vaswani, A. Self-Attention with Relative Position Representations. In Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2018), New Orleans, Louisiana, USA, 1–6 June 2018; ACL: New Orleans, LA, USA, 2018; pp. 464–468. https://doi.org/10.18653/v1/n18-2074.
289. Liu, Y.; Yang, S.; Lei, C.; Wang, G.; Tang, H.; Zhang, J.; Sun, A.; Miao, C. Pre-training Graph Transformer with Multimodal Side Information for Recommendation. In Proceedings of the Multimedia Conference (MM'21), Virtual Event, China, 20–24 October 2021; ACM: Virtual Event, China, 2021; pp. 2853–2861. https://doi.org/10.1145/3474085.3475709.
290. Lin, K.; Wang, L.; Liu, Z. End-to-End Human Pose and Mesh Reconstruction with Transformers. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2021), Virtual Event, 19–25 June 2021; Computer Vision Foundation/IEEE: Virtual Event, 2021; pp. 1954–1963. https://doi.org/10.1109/CVPR46437.2021.00199.
291. Makarov, I.; Kiselev, D.; Nikitinsky, N.; Subelj, L. Survey on graph embeddings and their applications to machine learning problems on graphs. *PeerJ Comput. Sci.* **2021**, *7*, e357. https://doi.org/10.7717/peerj-cs.357.
292. Yang, M.; Zhou, M.; Li, Z.; Liu, J.; Pan, L.; Xiong, H.; King, I. Hyperbolic Graph Neural Networks: A Review of Methods and Applications. *arXiv* **2022**, arXiv:2202.13852. https://doi.org/10.48550/ARXIV.2202.13852.

293. Chami, I.; Ying, Z.; Ré, C.; Leskovec, J. Hyperbolic Graph Convolutional Neural Networks. In Proceedings of the 32nd Annual Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, BC, Canada, 8–14 December 2019; NeurIPS: Vancouver, BC, Canada, 2019; pp. 4869–4880.
294. Liu, Q.; Nickel, M.; Kiela, D. Hyperbolic Graph Neural Networks. In Proceedings of the 32nd Annual Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, BC, Canada, 8–14 December 2019; NeurIPS: Vancouver, BC, Canada, 2019; pp. 8228–8239.
295. Ratcliffe, J.G.; Axler, S.; Ribet, K. *Foundations of Hyperbolic Manifolds*; Springer: Berlin/Heidelberg, Germany, 1994; Volume 149.
296. Krioukov, D.V.; Papadopoulos, F.; Kitsak, M.; Vahdat, A.; Boguñá, M. Hyperbolic Geometry of Complex Networks. *CoRR* **2010**, arXiv:1006.5169. http://arxiv.org/abs/1006.5169.
297. Defferrard, M.; Milani, M.; Gusset, F.; Perraudin, N. DeepSphere: A graph-based spherical CNN. In Proceedings of the 8th International Conference on Learning Representations (ICLR 2020), Addis Ababa, Ethiopia, 26–30 April 2020; OpenReview.net: Addis Ababa, Ethiopia, 2020.
298. Zhu, D.; Cui, P.; Wang, D.; Zhu, W. Deep Variational Network Embedding in Wasserstein Space. In Proceedings of the 24th International Conference on Knowledge Discovery & Data Mining (KDD 2018), London, UK, 19–23 August 2018; Guo, Y., Farooq, F., Eds.; ACM: London, UK, 2018; pp. 2827–2836. https://doi.org/10.1145/3219819.3220052.
299. He, S.; Liu, K.; Ji, G.; Zhao, J. Learning to Represent Knowledge Graphs with Gaussian Embedding. In Proceedings of the 24th International Conference on Information and Knowledge Management (CIKM 2015), Melbourne, VIC, Australia, 19–23 October 2015; ACM: Melbourne, VIC, Australia, 2015; pp. 623–632. https://doi.org/10.1145/2806416.2806502.
300. Vilnis, L.; McCallum, A. Word Representations via Gaussian Embedding. In Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015), San Diego, CA, USA, 7–9 May 2015; Bengio, Y., LeCun, Y., Eds.; ICLR: San Diego, CA, USA, 2015.
301. Kampffmeyer, M.; Chen, Y.; Liang, X.; Wang, H.; Zhang, Y.; Xing, E.P. Rethinking knowledge graph propagation for zero-shot learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2019), Long Beach, CA, USA, 16–20 June 2019; Computer Vision Foundation/IEEE: Long Beach, CA, USA, 2019; pp. 11487–11496.
302. Zhang, T.; Zheng, W.; Cui, Z.; Li, Y. Tensor graph convolutional neural network. *CoRR* **2018**, arXiv:1803.10071. http://arxiv.org/abs/1803.10071.
303. Yang, L.; Zhan, X.; Chen, D.; Yan, J.; Loy, C.C.; Lin, D. Learning to Cluster Faces on an Affinity Graph. In Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR 2019), Long Beach, CA, USA, 16–20 June 2019; Computer Vision Foundation/IEEE: Long Beach, CA, USA, 2019; pp. 2298–2306. https://doi.org/10.1109/CVPR.2019.00240.
304. Wang, Z.; Zheng, L.; Li, Y.; Wang, S. Linkage Based Face Clustering via Graph Convolution Network. In Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR 2019), Long Beach, CA, USA, 16–20 June 2019; Computer Vision Foundation/IEEE: Long Beach, CA, USA, 2019; pp. 1117–1125. https://doi.org/10.1109/CVPR.2019.00121.
305. Narasimhan, M.; Lazebnik, S.; Schwing, A.G. Out of the Box: Reasoning with Graph Convolution Nets for Factual Visual Question Answering. In Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, QC, Canada, 3–8 December 2018; Bengio, S., Wallach, H.M., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R., Eds.; NeurIPS: Montréal, QC, Canada, 2018; pp. 2659–2670.
306. Cui, Z.; Xu, C.; Zheng, W.; Yang, J. Context-Dependent Diffusion Network for Visual Relationship Detection. In Proceedings of the Multimedia Conference on Multimedia Conference (MM 2018), Seoul, Republic of Korea, 22–26 October 2018; Boll, S., Lee, K.M., Luo, J., Zhu, W., Byun, H., Chen, C.W., Lienhart, R., Mei, T., Eds.; ACM: Seoul, Republic of Korea, 2018; pp. 1475–1482. https://doi.org/10.1145/3240508.3240668.
307. Dai, B.; Zhang, Y.; Lin, D. Detecting Visual Relationships with Deep Relational Networks. In Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR 2017), Honolulu, HI, USA, 21–26 July 2017; IEEE Computer Society: Honolulu, HI, USA, 2017; pp. 3298–3308. https://doi.org/10.1109/CVPR.2017.352.
308. Johnson, J.; Gupta, A.; Fei-Fei, L. Image Generation From Scene Graphs. In Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR 2018), Salt Lake City, UT, USA, 18–22 June 2018; Computer Vision Foundation/IEEE Computer Society: Salt Lake City, UT, USA, 2018; pp. 1219–1228. https://doi.org/10.1109/CVPR.2018.00133.
309. Chen, Y.; Rohrbach, M.; Yan, Z.; Yan, S.; Feng, J.; Kalantidis, Y. Graph-Based Global Reasoning Networks. In Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR 2019), Long Beach, CA, USA, 16–20 June 2019; Computer Vision Foundation/IEEE: Long Beach, CA, USA, 2019; pp. 433–442. https://doi.org/10.1109/CVPR.2019.00052.
310. Wang, P.; Wu, Q.; Cao, J.; Shen, C.; Gao, L.; van den Hengel, A. Neighbourhood Watch: Referring Expression Comprehension via Language-Guided Graph Attention Networks. In Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR 2019), Long Beach, CA, USA, 16–20 June 2019; Computer Vision Foundation/IEEE: Long Beach, CA, USA, 2019; pp. 1960–1968. https://doi.org/10.1109/CVPR.2019.00206.
311. Li, M.; Chen, S.; Chen, X.; Zhang, Y.; Wang, Y.; Tian, Q. Actional-Structural Graph Convolutional Networks for Skeleton-Based Action Recognition. In Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR 2019), Long Beach, CA, USA, 16–20 June 2019; Computer Vision Foundation/IEEE: Long Beach, CA, USA, 2019; pp. 3595–3603. https://doi.org/10.1109/CVPR.2019.00371.

312. Wang, X.; Gupta, A. Videos as Space-Time Region Graphs. In Proceedings of the 15th Computer Vision European Conference (ECCV 2018), Munich, Germany, 8–14 September 2018; Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y., Eds.; Springer: Munich, Germany, 2018; Volume 11209, pp. 413–431. https://doi.org/10.1007/978-3-030-01228-1_25.
313. Gao, J.; Zhang, T.; Xu, C. Graph Convolutional Tracking. In Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR 2019), Long Beach, CA, USA, 16–20 June 2019; Computer Vision Foundation/IEEE: Long Beach, CA, USA, 2019; pp. 4649–4659. https://doi.org/10.1109/CVPR.2019.00478.
314. Zhong, J.; Li, N.; Kong, W.; Liu, S.; Li, T.H.; Li, G. Graph Convolutional Label Noise Cleaner: Train a Plug-And-Play Action Classifier for Anomaly Detection. In Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR 2019), Long Beach, CA, USA, 16–20 June 2019; Computer Vision Foundation/IEEE: Long Beach, CA, USA, 2019; pp. 1237–1246. https://doi.org/10.1109/CVPR.2019.00133.
315. Bastings, J.; Titov, I.; Aziz, W.; Marcheggiani, D.; Sima'an, K. Graph Convolutional Encoders for Syntax-aware Neural Machine Translation. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2017), Copenhagen, Denmark, 9–11 September 2017; Palmer, M., Hwa, R., Riedel, S., Eds.; ACL: Copenhagen, Denmark, 2017; pp. 1957–1967. https://doi.org/10.18653/v1/d17-1209.
316. Strubell, E.; McCallum, A. Dependency Parsing with Dilated Iterated Graph CNNs. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2017), Copenhagen, Denmark, 9–11 September 2017; Chang, K., Chang, M., Srikumar, V., Rush, A.M., Eds.; ACL: Copenhagen, Denmark, 2017; pp. 1–6. https://doi.org/10.18653/v1/w17-4301.
317. Vashishth, S.; Bhandari, M.; Yadav, P.; Rai, P.; Bhattacharyya, C.; Talukdar, P.P. Incorporating Syntactic and Semantic Information in Word Embeddings using Graph Convolutional Networks. In Proceedings of the 57th Conference of the Association for Computational Linguistics (ACL 2019), Florence, Italy, 28 July–2 August 2019; Korhonen, A., Traum, D.R., Màrquez, L., Eds.; ACL: Florence, Italy, 2019; pp. 3308–3318. https://doi.org/10.18653/v1/p19-1320.
318. Kim, D.; Kim, S.; Kwak, N. Textbook Question Answering with Multi-modal Context Graph Understanding and Self-supervised Open-set Comprehension. In Proceedings of the 57th Conference of the Association for Computational Linguistics (ACL 2019), Florence, Italy, 28 July–2 August 2019; Long Papers; Korhonen, A., Traum, D.R., Màrquez, L., Eds.; ACL: Florence, Italy, 2019; Volume 1, pp. 3568–3584. https://doi.org/10.18653/v1/p19-1347.
319. Hao, Y.; Sheng, Y.; Wang, J. A Graph Representation Learning Algorithm for Low-Order Proximity Feature Extraction to Enhance Unsupervised IDS Preprocessing. *Appl. Sci.* **2019**, *9*, 4473. https://doi.org/10.3390/app9204473.
320. Narayanan, A.; Chandramohan, M.; Chen, L.; Liu, Y. A multi-view context-aware approach to Android malware detection and malicious code localization. *Empir. Softw. Eng.* **2018**, *23*, 1222–1274. https://doi.org/10.1007/s10664-017-9539-8.
321. Nguyen, H.; Nguyen, D.; Ngo, Q.; Tran, V.; Le, V. Towards a rooted subgraph classifier for IoT botnet detection. In Proceedings of the 7th International Conference on Computer and Communications Management (ICCCM 2019), Bangkok, Thailand, 27–29 July 2019; ACM: Bangkok, Thailand, 2019; pp. 247–251. https://doi.org/10.1145/3348445.3348474.
322. Zitnik, M.; Agrawal, M.; Leskovec, J. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics* **2018**, *34*, i457–i466. https://doi.org/10.1093/bioinformatics/bty294.
323. Li, S.; Zhou, J.; Xu, T.; Dou, D.; Xiong, H. GeomGCL: Geometric Graph Contrastive Learning for Molecular Property Prediction. In Proceedings of the 36th Conference on Artificial Intelligence (AAAI 2022), Virtual Event, 22 February–1 March 2022; AAAI Press: 2022; pp. 4541–4549.
324. Duvenaud, D.; Maclaurin, D.; Hirzel, T.; Aspuru-Guzik, A.; Adams, R.P. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In Proceedings of the 28th Annual Conference on Neural Information Processing Systems (NeurIPS 2015), Montreal, QC, Canada, 7–12 December 2015; Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R., Eds.; NeurIPS: Montreal, QC, Canada, 2015; pp. 2224–2232.
325. Yamanishi, Y.; Araki, M.; Gutteridge, A.; Honda, W.; Kanehisa, M. Prediction of drug-target interaction networks from the integration of chemical and genomic spaces. In Proceedings of the 16th International Conference on Intelligent Systems for Molecular Biology (ISMB 2008), Toronto, ON, Canada, 19–23 July 2008; Oxford Academic: Toronto, ON, Canada, 2008; pp. 232–240. https://doi.org/10.1093/bioinformatics/btn162.
326. Cobanoglu, M.C.; Liu, C.; Hu, F.; Oltvai, Z.N.; Bahar, I. Predicting Drug-Target Interactions Using Probabilistic Matrix Factorization. *J. Chem. Inf. Model.* **2013**, *53*, 3399–3409. https://doi.org/10.1021/ci400219z.
327. Sharma, A.; Rani, R. BE-DTI': Ensemble framework for drug target interaction prediction using dimensionality reduction and active learning. *Comput. Methods Prog. Biomed.* **2018**, *165*, 151–162. https://doi.org/10.1016/j.cmpb.2018.08.011.
328. Zheng, X.; Ding, H.; Mamitsuka, H.; Zhu, S. Collaborative matrix factorization with multiple similarities for predicting drug-target interactions. In Proceedings of the the 19th International Conference on Knowledge Discovery and Data Mining (KDD 2013), Chicago, IL, USA, 11–14 August 2013; Dhillon, I.S., Koren, Y., Ghani, R., Senator, T.E., Bradley, P., Parekh, R., He, J., Grossman, R.L., Uthurusamy, R., Eds.; ACM: Chicago, IL, USA, 2013; pp. 1025–1033. https://doi.org/10.1145/2487575.2487670.
329. Yamanishi, Y.; Kotera, M.; Moriya, Y.; Sawada, R.; Kanehisa, M.; Goto, S. DINIES: drug-target interaction network inference engine based on supervised analysis. *Nucleic Acids Res.* **2014**, *42*, 39–45. https://doi.org/10.1093/nar/gku337.
330. Ezzat, A.; Zhao, P.; Wu, M.; Li, X.L.; Kwoh, C.K. Drug-Target Interaction Prediction with Graph Regularized Matrix Factorization. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **2017**, *14*, 646–656. https://doi.org/10.1109/TCBB.2016.2530062.
331. Li, X.; Yan, X.; Gu, Q.; Zhou, H.; Wu, D.; Xu, J. DeepChemStable: Chemical Stability Prediction with an Attention-Based Graph Convolution Network. *J. Chem. Inf. Model.* **2019**, *59*, 1044–1049. https://doi.org/10.1021/acs.jcim.8b00672.

332. Kulmanov, M.; Khan, M.A.; Hoehndorf, R. DeepGO: predicting protein functions from sequence and interactions using a deep ontology-aware classifier. *Bioinformatics* **2018**, *34*, 660–668. https://doi.org/10.1093/bioinformatics/btx624.
333. Li, G.; Luo, J.; Xiao, Q.; Liang, C.; Ding, P.; Cao, B. Predicting MicroRNA-Disease Associations Using Network Topological Similarity Based on DeepWalk. *IEEE Access* **2017**, *5*, 24032–24039. https://doi.org/10.1109/ACCESS.2017.2766758.
334. Su, X.; Hu, L.; You, Z.; Hu, P.; Zhao, B. Attention-based Knowledge Graph Representation Learning for Predicting Drug-drug Interactions. *Briefings Bioinform.* **2022**, *23*, bbac140. https://doi.org/10.1093/bib/bbac140.
335. Çelebi, R.; Yasar, E.; Uyar, H.; Gümüs, Ö.; Dikenelli, O.; Dumontier, M. Evaluation of knowledge graph embedding approaches for drug-drug interaction prediction using Linked Open Data. In Proceedings of the 11th International Conference Semantic Web Applications and Tools for Life Sciences (SWAT4LS 2018), Antwerp, Belgium, 3–6 December 2018; Waagmeester, A., Baker, C.J.O., Splendiani, A., Beyan, O.D., Marshall, M.S., Eds.; CEUR-WS.org: Antwerp, Belgium, 2018; Volume 2275.
336. Karim, M.R.; Cochez, M.; Jares, J.B.; Uddin, M.; Beyan, O.D.; Decker, S. Drug-Drug Interaction Prediction Based on Knowledge Graph Embeddings and Convolutional-LSTM Network. In Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics (BCB 2019), Niagara Falls, NY, USA, 7–10 September 2019; Shi, X.M., Buck, M., Ma, J., Veltri, P., Eds.; ACM: Niagara Falls, NY, USA, 2019; pp. 113–123. https://doi.org/10.1145/3307339.3342161.
337. Lin, X.; Quan, Z.; Wang, Z.; Ma, T.; Zeng, X. KGNN: Knowledge Graph Neural Network for Drug-Drug Interaction Prediction. In Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020), Yokohama, Japan, 7–15 January 2021; Bessiere, C., Ed.; ijcai.org: Yokohama, Japan, 2020; pp. 2739–2745. https://doi.org/10.24963/ijcai.2020/380.
338. Zhao, T.; Hu, Y.; Valsdottir, L.R.; Zang, T.; Peng, J. Identifying drug-target interactions based on graph convolutional network and deep neural network. *Briefings Bioinform.* **2021**, *22*, 2141–2150. https://doi.org/10.1093/bib/bbaa044.
339. An, Q.; Yu, L. A heterogeneous network embedding framework for predicting similarity-based drug-target interactions. *Briefings Bioinform.* **2021**, *22*, bbab275. https://doi.org/10.1093/bib/bbab275.
340. Peng, J.; Wang, Y.; Guan, J.; Li, J.; Han, R.; Hao, J.; Wei, Z.; Shang, X. An end-to-end heterogeneous graph representation learning-based framework for drug-target interaction prediction. *Briefings Bioinform.* **2021**, *22*, bbaa430. https://doi.org/10.1093/bib/bbaa430.
341. Monti, F.; Frasca, F.; Eynard, D.; Mannion, D.; Bronstein, M.M. Fake News Detection on Social Media using Geometric Deep Learning. *CoRR* **2019**, arXiv:1902.06673. http://arxiv.org/abs/1902.06673.
342. Benamira, A.; Devillers, B.; Lesot, E.; Ray, A.K.; Saadi, M.; Malliaros, F.D. Semi-supervised learning and graph neural networks for fake news detection. In Proceedings of the International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2019), Vancouver, British Columbia, Canada, 27–30 August 2019; ACM: Vancouver, BC, Canada, 2019; pp. 568–569. https://doi.org/10.1145/3341161.3342958.
343. Nguyen, V.; Sugiyama, K.; Nakov, P.; Kan, M. FANG: Leveraging Social Context for Fake News Detection Using Graph Representation. In Proceedings of the 29th International Conference on Information and Knowledge Management (CIKM 2020), Virtual Event, Ireland, 19–23 October 2020; d'Aquin, M., Dietze, S., Hauff, C., Curry, E., Cudré-Mauroux, P., Eds.; ACM: Virtual Event, Ireland, 2020; pp. 1165–1174. https://doi.org/10.1145/3340531.3412046.
344. Piao, J.; Zhang, G.; Xu, F.; Chen, Z.; Li, Y. Predicting Customer Value with Social Relationships via Motif-based Graph Attention Networks. In Proceedings of the The Web Conference 2021 (WWW 2021), Ljubljana, Slovenia, 19–23 April 2021; Leskovec, J., Grobelnik, M., Najork, M., Tang, J., Zia, L., Eds.; ACM/IW3C2: Ljubljana, Slovenia, 2021; pp. 3146–3157. https://doi.org/10.1145/3442381.3449849.
345. Li, C.; Goldwasser, D. Encoding social information with graph convolutional networks for political perspective detection in news media. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy, 28 July– 2 August 2019; ACL: Florence, Italy, 2019; pp. 2594–2604.
346. Qiu, J.; Tang, J.; Ma, H.; Dong, Y.; Wang, K.; Tang, J. DeepInf: Social Influence Prediction with Deep Learning. In Proceedings of the 24th International Conference on Knowledge Discovery & Data Mining (KDD 2018), London, UK, 19–23 August 2018; Guo, Y., Farooq, F., Eds.; ACM: London, UK, 2018; ACM: London, UK, 2018; pp. 2110–2119. https://doi.org/10.1145/3219819.3220077.
347. Koren, Y.; Bell, R.M.; Volinsky, C. Matrix Factorization Techniques for Recommender Systems. *Computer* **2009**, *42*, 30–37. https://doi.org/10.1109/MC.2009.263.
348. Gu, Q.; Zhou, J.; Ding, C.H.Q. Collaborative Filtering: Weighted Nonnegative Matrix Factorization Incorporating User and Item Graphs. In Proceedings of the International Conference on Data Mining (SDM 2010), Columbus, Ohio, USA, 29 April–1 May 2010; SIAM: Columbus, OH, USA, 2010; pp. 199–210. https://doi.org/10.1137/1.9781611972801.18.
349. Guo, Q.; Sun, Z.; Theng, Y. Exploiting Side Information for Recommendation. In Proceedings of the 19th International Conference (ICWE 2019), Daejeon, South Korea, 11–14 June 2019; Bakaev, M., Frasincar, F., Ko, I., Eds.; Springer: Daejeon, Republic of Korea, 2019; Volume 11496, pp. 569–573. https://doi.org/10.1007/978-3-030-19274-7_46.
350. Sun, Z.; Guo, Q.; Yang, J.; Fang, H.; Guo, G.; Zhang, J.; Burke, R. Research commentary on recommendations with side information: A survey and research directions. *Electron. Commer. Res. Appl.* **2019**, *37*. https://doi.org/10.1016/j.elerap.2019.100879.
351. He, R.; Lin, C.; Wang, J.; McAuley, J.J. Sherlock: Sparse Hierarchical Embeddings for Visually-Aware One-Class Collaborative Filtering. In Proceedings of the 35th International Joint Conference on Artificial Intelligence (IJCAI 2016), New York, NY, USA, 9–15 July 2016; Kambhampati, S., Ed.; IJCAI/AAAI Press: New York, NY, USA, 2016; pp. 3740–3746.

352. Zhang, Y.; Lai, G.; Zhang, M.; Zhang, Y.; Liu, Y.; Ma, S. Explicit factor models for explainable recommendation based on phrase-level sentiment analysis. In Proceedings of The 37th International Conference on Research and Development in Information Retrieval (SIGIR'14), Gold Coast, QLD, Australia, 6–11 July 2014; Geva, S., Trotman, A., Bruza, P., Clarke, C.L.A., Järvelin, K., Eds.; ACM: Gold Coast, QLD, Australia, 2014; pp. 83–92. https://doi.org/10.1145/2600428.2609579.
353. Fan, W.; Ma, Y.; Li, Q.; He, Y.; Zhao, Y.E.; Tang, J.; Yin, D. Graph Neural Networks for Social Recommendation. In Proceedings of the World Wide Web Conference (WWW 2019), San Francisco, CA, USA, 13–17 May 2019; Liu, L., White, R.W., Mantrach, A., Silvestri, F., McAuley, J.J., Baeza-Yates, R., Zia, L., Eds.; ACM: San Francisco, CA, USA, 2019; pp. 417–426. https://doi.org/10.1145/3308558.3313488.
354. Dong, X.; Yu, L.; Wu, Z.; Sun, Y.; Yuan, L.; Zhang, F. A Hybrid Collaborative Filtering Model with Deep Structure for Recommender Systems. In Proceedings of the 31st Conference on Artificial Intelligence, (AAAI 2017), San Francisco, CA, USA, 4–9 February 2017; Singh, S., Markovitch, S., Eds.; AAAI Press: San Francisco, CA, USA, 2017; pp. 1309–1315.
355. Wang, X.; He, X.; Cao, Y.; Liu, M.; Chua, T. KGAT: Knowledge Graph Attention Network for Recommendation. In Proceedings of the 25th International Conference on Knowledge Discovery & Data Mining (KDD 2019), Anchorage, AK, USA, 4–8 August 2019; ACM: Anchorage, AK, USA, 2019; pp. 950–958. https://doi.org/10.1145/3292500.3330989.
356. Wang, X.; He, X.; Wang, M.; Feng, F.; Chua, T. Neural Graph Collaborative Filtering. In Proceedings of the 42nd International Conference on Research and Development in Information Retrieval (SIGIR 2019), Paris, France, 21–25 July 2019; Piwowarski, B., Chevalier, M., Gaussier, É., Maarek, Y., Nie, J., Scholer, F., Eds.; ACM: Paris, France, 2019; pp. 165–174. https://doi.org/10.1145/3331184.3331267.
357. van den Berg, R.; Kipf, T.N.; Welling, M. Graph Convolutional Matrix Completion. *CoRR* **2017**, arXiv:1706.02263. http://arxiv.org/abs/1706.02263.
358. Chai, D.; Wang, L.; Yang, Q. Bike flow prediction with multi-graph convolutional networks. In Proceedings of the 26th International Conference on Advances in Geographic Information Systems (SIGSPATIAL 2018), Seattle, WA, USA, 6–9 November 2018; Kashani, F.B.; Hoel, E.G., Güting, R.H., Tamassia, R., Xiong, L., Eds.; ACM: Seattle, WA, USA, 2018; pp. 397–400. https://doi.org/10.1145/3274895.3274896.
359. Diehl, F.; Brunner, T.; Truong-Le, M.; Knoll, A.C. Graph Neural Networks for Modelling Traffic Participant Interaction. In Proceedings of the IEEE Intelligent Vehicles Symposium (IV 2019), Paris, France, 9–12 June 2019; IEEE: Paris, France, 2019; pp. 695–701. https://doi.org/10.1109/IVS.2019.8814066.
360. Wu, Z.; Pan, S.; Long, G.; Jiang, J.; Zhang, C. Graph WaveNet for Deep Spatial-Temporal Graph Modeling. In Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019), Macao, China, 10–16 August 2019; Kraus, S., Ed.; ijcai.org: Macao, China, 2019; pp. 1907–1913. https://doi.org/10.24963/ijcai.2019/264.
361. Guo, S.; Lin, Y.; Feng, N.; Song, C.; Wan, H. Attention Based Spatial-Temporal Graph Convolutional Networks for Traffic Flow Forecasting. In Proceedings of the 33rd Conference on Artificial Intelligence (AAAI 2019), Honolulu, Hawaii, USA, 27 January–1 February 2019; AAAI Press: Honolulu, HI, USA, 2019; pp. 922–929. https://doi.org/10.1609/aaai.v33i01.3301922.
362. Khodayar, M.; Mohammadi, S.; Khodayar, M.E.; Wang, J.; Liu, G. Convolutional Graph Autoencoder: A Generative Deep Neural Network for Probabilistic Spatio-Temporal Solar Irradiance Forecasting. *IEEE Trans. Sustain. Energy* **2020**, *11*, 571–583. https://doi.org/10.1109/TSTE.2019.2897688.
363. Khodayar, M.; Wang, J. Spatio-temporal graph deep neural network for short-term wind speed forecasting. *IEEE Trans. Sustain. Energy* **2019**, *10*, 670–681.
364. Owerko, D.; Gama, F.; Ribeiro, A. Predicting Power Outages Using Graph Neural Networks. In Proceedings of the 6th IEEE Global Conference on Signal and Information Processing (GlobalSIP 2018), Anaheim, CA, USA, 26–29 November 2018; IEEE: Anaheim, CA, USA, 2018; pp. 743–747. https://doi.org/10.1109/GlobalSIP.2018.8646486.
365. Austin, A.E.; Desrosiers, T.A.; Shanahan, M.E. Directed acyclic graphs: An under-utilized tool for child maltreatment research. *Child Abus. Negl.* **2019**, *91*, 78–87. https://doi.org/https://doi.org/10.1016/j.chiabu.2019.02.011.
366. Waller, I.; Anderson, A. Quantifying social organization and political polarization in online platforms. *Nature* **2021**, *600*, 264–268. https://doi.org/10.1038/s41586-021-04167-x.
367. Kubin, E.; von Sikorski, C. The role of (social) media in political polarization: A systematic review. *Ann. Int. Commun. Assoc.* **2021**, *45*, 188–206.
368. Tokita, C.K.; Guess, A.M.; Tarnita, C.E. Polarized information ecosystems can reorganize social networks via information cascades. *Proc. Natl. Acad. Sci. USA* **2021**, *118*, e2102147118. https://doi.org/10.1073/pnas.2102147118.
369. Martin, T. community2vec: Vector representations of online communities encode semantic relationships. In Proceedings of the 2nd Workshop on NLP and Computational Social Science (NLP+CSS@ACL 2017), Vancouver, BC, Canada, 3 August 2017; Hovy, D., Volkova, S., Bamman, D., Jurgens, D., O'Connor, B., Tsur, O., Dogruöz, A.S., Eds.; Association for Computational Linguistics: Vancouver, BC, Canada, 2017; pp. 27–31. https://doi.org/10.18653/v1/w17-2904.
370. Elson, D.K.; Dames, N.; McKeown, K.R. Extracting Social Networks from Literary Fiction. In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL 2010), Uppsala, Sweden, 11–16 July 2010; ACL: Uppsala, Sweden, 2010; pp. 138–147.
371. Moretti, F. Network theory, plot analysis. *New Left Rev.* **2011**; Volume 2, pp. 80–102.

372. Agarwal, A.; Kotalwar, A.; Rambow, O. Automatic Extraction of Social Networks from Literary Text: A Case Study on Alice in Wonderland. In Proceedings of the 6th International Joint Conference on Natural Language Processing (IJCNLP 2013), Nagoya, Japan, 14–18 October 2013; Asian Federation of Natural Language Processing/ACL: Nagoya, Japan, 2013; pp. 1202–1208.
373. Jayannavar, P.; Agarwal, A.; Ju, M.; Rambow, O. Validating Literary Theories Using Automatic Social Network Extraction. In Proceedings of the Workshop on Computational Linguistics for Literature (CLfL@NAACL-HLT 2015), Denver, Colorado, USA, 4 June 2015; Feldman, A., Kazantseva, A., Szpakowicz, S., Koolen, C., Eds.; The Association for Computer Linguistics: Denver, CO, USA, 2015; pp. 32–41. https://doi.org/10.3115/v1/w15-0704.
374. Grayson, S.; Wade, K.; Meaney, G.; Greene, D. The Sense and Sensibility of Different Sliding Windows in Constructing Co-occurrence Networks from Literature. In Proceedings of the Computational History and Data-Driven Humanities International Workshop (CHDDH 2016), Dublin, Ireland, 25 May 2016; Bozic, B., Mendel-Gleason, G., Debruyne, C., O'Sullivan, D., Eds.; Springer: Dublin, Ireland, 2016; Volume 482, pp. 65–77. https://doi.org/10.1007/978-3-319-46224-0_7.
375. Dekker, N.; Kuhn, T.; van Erp, M. Evaluating social network extraction for classic and modern fiction literature. *PeerJ Prepr.* **2018**, *6*, e27263. https://doi.org/10.7287/peerj.preprints.27263v1.
376. Inoue, N.; Pethe, C.; Kim, A.; Skiena, S. Learning and Evaluating Character Representations in Novels. In Proceedings of the Findings of the Association for Computational Linguistics (ACL 2022), Dublin, Ireland, 22–27 May 2022; Muresan, S., Nakov, P., Villavicencio, A., Eds.; ACL: Dublin, Ireland, 2022; pp. 1008–1019. https://doi.org/10.18653/v1/2022.findings-acl.81.
377. Kounelis, A.; Vikatos, P.; Makris, C. Movie Recommendation System Based on Character Graph Embeddings. In Proceedings of the Artificial Intelligence Applications and Innovations (AIAI 2021), Crete, Greece, 25–27 June 2021; Maglogiannis, I., MacIntyre, J., Iliadis, L., Eds.; Springer: Crete, Greece, 2021; Volume 628, pp. 418–430. https://doi.org/10.1007/978-3-030-79157-5_34.
378. Riedl, M.O. Computational Narrative Intelligence: A Human-Centered Goal for Artificial Intelligence. *CoRR* **2016**, arXiv:1602.06484. http://arxiv.org/abs/1602.06484.
379. Na, G.S.; Jang, S.; Lee, Y.L.; Chang, H. Tuplewise material representation based machine learning for accurate band gap prediction. *J. Phys. Chem.* **2020**, *124*, 10616–10623.
380. Wang, H.; Wang, K.; Yang, J.; Shen, L.; Sun, N.; Lee, H.; Han, S. GCN-RL Circuit Designer: Transferable Transistor Sizing with Graph Neural Networks and Reinforcement Learning. In Proceedings of the 57th Design Automation Conference (DAC 2020), San Francisco, CA, USA, 20–24 July 2020; IEEE: San Francisco, CA, USA, 2020; pp. 1–6. https://doi.org/10.1109/DAC18072.2020.9218757.
381. Li, Y.; Lin, Y.; Madhusudan, M.; Sharma, A.K.; Xu, W.; Sapatnekar, S.S.; Harjani, R.; Hu, J. A Customized Graph Neural Network Model for Guiding Analog IC Placement. In Proceedings of the 39th International Conference On Computer Aided Design (ICCAD 2020), San Diego, CA, USA, 2–5 November 2020; IEEE: San Diego, CA, USA, 2020; pp. 135.1–135.9. https://doi.org/10.1145/3400302.3415624.
382. Lu, Y.; Nath, S.; Khandelwal, V.; Lim, S.K. Doomed Run Prediction in Physical Design by Exploiting Sequential Flow and Graph Learning. In Proceedings of the 40th International Conference On Computer Aided Design (ICCAD 2021), Munich, Germany, 1–4 November 2021; IEEE: Munich, Germany, 2021; pp. 1–9. https://doi.org/10.1109/ICCAD51958.2021.9643435.
383. Gallo, C.; Capozzi, V. A Wafer Bin Map "Relaxed" Clustering Algorithm for Improving Semiconductor Production Yield. *Open Comput. Sci.* **2020**, *10*, 231–245. https://doi.org/10.1515/comp-2020-0175.
384. Ko, Y.; Fujita, H. An evidential analytics for buried information in big data samples: Case study of semiconductor manufacturing. *Inf. Sci.* **2019**, *486*, 190–203. https://doi.org/10.1016/j.ins.2019.01.079.
385. Mirhoseini, A.; Goldie, A.; Yazgan, M.; Jiang, J.W.; Songhori, E.M.; Wang, S.; Lee, Y.; Johnson, E.; Pathak, O.; Nazi, A.; et al. A graph placement methodology for fast chip design. *Nature* **2021**, *594*, 207–212. https://doi.org/10.1038/s41586-021-03544-w.
386. Schulz, B.; Jacobi, C.; Gisbrecht, A.; Evangelos, A.; Chan, C.W.; Gan, B.P. Graph Representation and Embedding for Semiconductor Manufacturing Fab States. In Proceedings of the Winter Simulation Conference (WSC 2022), Singapore, 11–14 December 2022; IEEE: Singapore, 2022; pp. 3382–3393. https://doi.org/10.1109/WSC57314.2022.10015297.
387. Jiao, X.; Li, X.; Lin, D.; Xiao, W. A Graph Neural Network based Deep Learning Predictor for Spatio-Temporal Group Solar Irradiance Forecasting. *IEEE Trans. Ind. Inform.* **2021**, *18*, 6142–6149.
388. Wang, K.; Qi, X.; Liu, H. Photovoltaic power forecasting based LSTM-Convolutional Network. *Energy* **2019**, *189*, 116225.
389. Lira, H.; Martí, L.; Sanchez-Pi, N. A Graph Neural Network with Spatio-Temporal Attention for Multi-Sources Time Series Data: An Application to Frost Forecast. *Sensors* **2022**, *22*, 1486. https://doi.org/10.3390/s22041486.
390. Lin, H.; Gao, Z.; Xu, Y.; Wu, L.; Li, L.; Li, S.Z. Conditional Local Convolution for Spatio-Temporal Meteorological Forecasting. In Proceedings of the 36th Conference on Artificial Intelligence (AAAI 2022), Virtual Event, 22 February–1 March 2022; AAAI Press: Virtual Event, 2022; pp. 7470–7478.
391. Jeon, H.J.; Choi, M.W.; Lee, O.J. Day-Ahead Hourly Solar Irradiance Forecasting Based on Multi-Attributed Spatio-Temporal Graph Convolutional Network. *Sensors* **2022**, *22*, 7179. https://doi.org/10.3390/s22197179.
392. Xiao, X.; Jin, Z.; Wang, S.; Xu, J.; Peng, Z.; Wang, R.; Shao, W.; Hui, Y. A dual-path dynamic directed graph convolutional network for air quality prediction. *Sci. Total. Environ.* **2022**, *827*, 154298. https://doi.org/https://doi.org/10.1016/j.scitotenv.2022.154298.
393. Han, J.; Liu, H.; Zhu, H.; Xiong, H.; Dou, D. Joint Air Quality and Weather Prediction Based on Multi-Adversarial Spatiotemporal Networks. In Proceedings of the 35th Conference on Artificial Intelligence (AAAI 2021), Virtual Event, 2–9 February 2021; AAAI Press: Virtual Event, 2021; pp. 4081–4089. https://doi.org/10.1609/aaai.v35i5.16529.

394. Han, J.; Liu, H.; Zhu, H.; Xiong, H. Kill Two Birds with One Stone: A Multi-View Multi-Adversarial Learning Approach for Joint Air Quality and Weather Prediction. *IEEE Trans. Knowl. Data Eng.* **2023**; pp. 1–14. https://doi.org/10.1109/tkde.2023.3236423.
395. Yano, K.; Shiina, T.; Kurata, S.; Kato, A.; Komaki, F.; Sakai, S.; Hirata, N. Graph-Partitioning Based Convolutional Neural Network for Earthquake Detection Using a Seismic Array. *J. Geophys. Res. Solid Earth* **2021**, *126*, e2020JB020269. https://doi.org/10.1029/2020jb020269.
396. van den Ende, M.P.A.; Ampuero, J.P. Automated Seismic Source Characterization Using Deep Graph Neural Networks. *Geophys. Res. Lett.* **2020**, *47*, e2020GL088690. https://doi.org/10.1029/2020gl088690.
397. Zhang, X.; Reichard-Flynn, W.; Zhang, M.; Hirn, M.; Lin, Y. Spatiotemporal Graph Convolutional Networks for Earthquake Source Characterization. *J. Geophys. Res. Solid Earth* **2022**, *127*, e2022JB024401. https://doi.org/10.1029/2022jb024401.
398. Yang, Y.; Dong, J.; Sun, X.; Lima, E.; Mu, Q.; Wang, X. A CFCC-LSTM Model for Sea Surface Temperature Prediction. *IEEE Geosci. Remote. Sens. Lett.* **2018**, *15*, 207–211. https://doi.org/10.1109/lgrs.2017.2780843.
399. Jia, X.; Ji, Q.; Han, L.; Liu, Y.; Han, G.; Lin, X. Prediction of Sea Surface Temperature in the East China Sea Based on LSTM Neural Network. *Remote. Sens.* **2022**, *14*, 3300. https://doi.org/10.3390/rs14143300.
400. Sun, Y.; Yao, X.; Bi, X.; Huang, X.; Zhao, X.; Qiao, B. Time-Series Graph Network for Sea Surface Temperature Prediction. *Big Data Res.* **2021**, *25*, 100237. https://doi.org/10.1016/j.bdr.2021.100237.
401. Zhang, X.; Li, Y.; Frery, A.C.; Ren, P. Sea Surface Temperature Prediction With Memory Graph Convolutional Networks. *IEEE Geosci. Remote. Sens. Lett.* **2022**, *19*, 1–5. https://doi.org/10.1109/lgrs.2021.3097329.
402. Lam, R.; Sanchez-Gonzalez, A.; Willson, M.; Wirnsberger, P.; Fortunato, M.; Pritzel, A.; Ravuri, S.V.; Ewalds, T.; Alet, F.; Eaton-Rosen, Z.; et al. GraphCast: Learning skillful medium-range global weather forecasting. *CoRR* **2022**, arXiv:2212.12794 https://doi.org/10.48550/arXiv.2212.12794.
403. Shi, N.; Xu, J.; Wurster, S.W.; Guo, H.; Woodring, J.; Van Roekel, L.P.; Shen, H.W. GNN-Surrogate: A Hierarchical and Adaptive Graph Neural Network for Parameter Space Exploration of Unstructured-Mesh Ocean Simulations. *IEEE Trans. Vis. Comput. Graph.* **2022**, *28*, 2301–2313. https://doi.org/10.1109/TVCG.2022.3165345.
404. Cachay, S.R.; Erickson, E.; Bucker, A.F.C.; Pokropek, E.; Potosnak, W.; Bire, S.; Osei, S.; Lütjens, B. The World as a Graph: Improving El Niño Forecasts with Graph Neural Networks. *CoRR* **2021**, arXiv:2104.05089 http://arxiv.org/abs/2104.05089.
405. Mu, B.; Qin, B.; Yuan, S. ENSO-ASC 1.0.0: ENSO deep learning forecast model with a multivariate air–sea coupler. *Geosci. Model Dev.* **2021**, *14*, 6977–6999. https://doi.org/10.5194/gmd-14-6977-2021.
406. Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Gallagher, B.; Eliassi-Rad, T. Collective Classification in Network Data. *AI Mag.* **2008**, *29*, 93–106. https://doi.org/10.1609/aimag.v29i3.2157.
407. Lim, K.W.; Buntine, W.L. Bibliographic Analysis with the Citation Network Topic Model. In Proceedings of the 6th Asian Conference on Machine Learning (ACML 2014), Nha Trang City, Vietnam, 26–28 November 2014; JMLR.org: Nha Trang City, Vietnam, 2014; Volume 39.
408. Tang, J.; Zhang, J.; Yao, L.; Li, J.; Zhang, L.; Su, Z. ArnetMiner: Extraction and mining of academic social networks. In Proceedings of the 14th International Conference on Knowledge Discovery and Data Mining (SIGKDD 2008), Las Vegas, Nevada, USA, 24–27 August 2008; ACM: Las Vegas, NV, USA, 2008; pp. 990–998. https://doi.org/10.1145/1401890.1402008.
409. McAuley, J.J.; Leskovec, J. Learning to Discover Social Circles in Ego Networks. In Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NeurIPS 2012), Lake Tahoe, Nevada, USA, 3–6 December 2012; NeurIPS: Lake Tahoe, Nevada, USA, 2012; pp. 548–556.
410. Debnath, A.K.; Lopez de Compadre, R.L.; Debnath, G.; Shusterman, A.J.; Hansch, C. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *J. Med. Chem.* **1991**, *34*, 786–797. https://doi.org/10.1021/jm00106a046.
411. Borgwardt, K.M.; Ong, C.S.; Schönauer, S.; Vishwanathan, S.V.N.; Smola, A.J.; Kriegel, H. Protein function prediction via graph kernels. In Proceedings of the 13th International Conference on Intelligent Systems for Molecular Biology (ISMB 2005), Detroit, MI, USA, 25–29 June 2005; Oxford University Press: Detroit, MI, USA, 2005; pp. 47–56. https://doi.org/10.1093/bioinformatics/bti1007.
412. Jin, D.; Huo, C.; Liang, C.; Yang, L. Heterogeneous Graph Neural Network via Attribute Completion. In Proceedings of the Web Conference (WWW 2021), Ljubljana, Slovenia, 19–23 April 2021; Leskovec, J., Grobelnik, M., Najork, M., Tang, J., Zia, L., Eds.; ACM/IW3C2: Virtual Event, 2021; pp. 391–400. https://doi.org/10.1145/3442381.3449914.
413. Sun, Y.; Han, J.; Yan, X.; Yu, P.S.; Wu, T. PathSim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks. *Proc. VLDB Endow.* **2011**, *4*, 992–1003.
414. Zafarani, R.; Liu, H. Social Computing Data Repository at ASU, 2009. Available online: http://socialcomputing.asu.edu (accessed on 16 April 2023).
415. Zhang, X.; Zhao, J.; LeCun, Y. Character-Level Convolutional Networks for Text Classification. *arXiv* **2015**, arXiv:1509.01626. http://arxiv.org/abs/1509.01626.
416. Kunegis, J. KONECT: the Koblenz network collection. In Proceedings of the 22nd International World Wide Web Conference (WWW 2013), Rio de Janeiro, Brazil, 13–17 May 2013; Carr, L., Laender, A.H.F., Lóscio, B.F., King, I., Fontoura, M., Vrandecic, D., Aroyo, L., de Oliveira, J.P.M., Lima, F., Wilde, E., Eds.; International World Wide Web Conferences Steering Committee/ACM: Rio de Janeiro, Brazil, 2013; pp. 1343–1350. https://doi.org/10.1145/2487788.2488173.

417. Tang, J.; Gao, H.; Liu, H. mTrust: discerning multi-faceted trust in a connected world. In Proceedings of the 5th International Conference on Web Search and Web Data Mining (WSDM 2012), Seattle, WA, USA, 8–12 February 2012; Adar, E., Teevan, J., Agichtein, E., Maarek, Y., Eds.; ACM: Seattle, WA, USA, 2012; pp. 93–102. https://doi.org/10.1145/2124295.2124309.
418. Gehrke, J.; Ginsparg, P.; Kleinberg, J.M. Overview of the 2003 KDD Cup. *SIGKDD Explor.* **2003**, *5*, 149–151. https://doi.org/10.1145/980972.980992.
419. Leskovec, J.; Krevl, A. SNAP Datasets: Stanford Large Network Dataset Collection. 2014. Available online: http://snap.stanford.edu/data (accessed on 16 April 2023).
420. Leskovec, J.; Lang, K.J.; Dasgupta, A.; Mahoney, M.W. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. *Internet Math.* **2009**, *6*, 29–123. https://doi.org/10.1080/15427951.2009.10129177.
421. Lippmann, R.; Cunningham, R.K.; Fried, D.J.; Graf, I.; Kendall, K.R.; Webster, S.E.; Zissman, M.A. Results of the DARPA 1998 Offline Intrusion Detection Evaluation. In Proceedings of the Recent Advances in Intrusion Detection, Second International Workshop, (RAID 1999), West Lafayette, IN, USA, 7–9 September 1999; Springer: West Lafayette, IN, USA, 1999.
422. Zafarani, R.; Liu, H. Users Joining Multiple Sites: Distributions and Patterns. In Proceedings of the 8th International Conference on Weblogs and Social Media (ICWSM 2014), Ann Arbor, MI, USA, 1–4 June 2014; Adar, E., Resnick, P., Choudhury, M.D., Hogan, B., Oh, A., Eds.; AAAI Press: Ann Arbor, MI, USA, 2014.
423. Furey, T.S.; Cristianini, N.; Duffy, N.; Bednarski, D.W.; Schummer, M.; Haussler, D. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics* **2000**, *16*, 906–914. https://doi.org/10.1093/bioinformatics/16.10.906.
424. Zheng, J.; Liu, J.; Shi, C.; Zhuang, F.; Li, J.; Wu, B. Recommendation in heterogeneous information network via dual similarity regularization. *Int. J. Data Sci. Anal.* **2017**, *3*, 35–48. https://doi.org/10.1007/s41060-016-0031-0.
425. Ali, Z.; Kefalas, P.; Muhammad, K.; Ali, B.; Imran, M. Deep learning in citation recommendation models survey. *Expert Syst. Appl.* **2020**, *162*, 113790. https://doi.org/10.1016/j.eswa.2020.113790.
426. Fey, M.; Lenssen, J.E. Fast Graph Representation Learning with PyTorch Geometric. *CoRR* **2019**, arXiv:1903.02428. http://arxiv.org/abs/1903.02428.
427. Wang, M.; Zheng, D.; Ye, Z.; Gan, Q.; Li, M.; Song, X.; Zhou, J.; Ma, C.; Yu, L.; Gai, Y.; et al. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv* **2019**, arXiv:1909.01315.
428. Cen, Y.; Hou, Z.; Wang, Y.; Chen, Q.; Luo, Y.; Yao, X.; Zeng, A.; Guo, S.; Zhang, P.; Dai, G.; et al. CogDL: An Extensive Toolkit for Deep Learning on Graphs. *CoRR* **2021**, arXiv:2103.00959. http://arxiv.org/abs/2103.00959.
429. Liu, M.; Luo, Y.; Wang, L.; Xie, Y.; Yuan, H.; Gui, S.; Yu, H.; Xu, Z.; Zhang, J.; Liu, Y.; et al. DIG: A Turnkey Library for Diving into Graph Deep Learning Research. *J. Mach. Learn. Res.* **2021**, *22*, 240.1–240.9.
430. Zhu, Z.; Xu, S.; Tang, J.; Qu, M. GraphVite: A High-Performance CPU-GPU Hybrid System for Node Embedding. In Proceedings of the World Wide Web Conference (WWW 2019), San Francisco, CA, USA, 13–17 May 2019; Liu, L., White, R.W., Mantrach, A., Silvestri, F., McAuley, J.J., Baeza-Yates, R., Zia, L., Eds.; ACM: San Francisco, CA, USA, 2019; pp. 2494–2504. https://doi.org/10.1145/3308558.3313508.
431. Zhu, R.; Zhao, K.; Yang, H.; Lin, W.; Zhou, C.; Ai, B.; Li, Y.; Zhou, J. AliGraph: A comprehensive graph neural network platform. *Proc. Vldb Endow.* **2019**, *12*, 2094–2105.
432. Sonthalia, R.; Gilbert, A.C. Tree! I am no Tree! I am a low dimensional Hyperbolic Embedding. In Proceedings of the 33rd Annual Conference on Neural Information Processing Systems (NeurIPS 2020), Virtual Event, 6–12 December 2020; Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H., Eds.; NeurIPS: Virtual Event, 2020.
433. Lou, A.; Katsman, I.; Jiang, Q.; Belongie, S.; Lim, S.N.; De Sa, C. Differentiating through the FréChet Mean. In Proceedings of the 37th International Conference on Machine Learning (ICML 2020), Vienna, Austria, 13–18 July 2020; JMLR.org: Vienna, Austria, 2020.
434. Shu, J.; Xi, B.; Li, Y.; Wu, F.; Kamhoua, C.A.; Ma, J. Understanding Dropout for Graph Neural Networks. In Proceedings of the Companion of The Web Conference 2022 (WWW 2022), Lyon, France, 25–29 April 2022; Laforest, F., Troncy, R., Simperl, E., Agarwal, D., Gionis, A., Herman, I., Médini, L., Eds.; ACM: Lyon, France, 2022; pp. 1128–1138. https://doi.org/10.1145/3487553.3524725.
435. Papp, P.A.; Martinkus, K.; Faber, L.; Wattenhofer, R. DropGNN: Random Dropouts Increase the Expressiveness of Graph Neural Networks. In Proceedings of the 34th Annual Conference on Neural Information Processing Systems 2021 (NeurIPS 2021), Virtual Event, 6–14 December 2021; Ranzato, M., Beygelzimer, A., Dauphin, Y.N., Liang, P., Vaughan, J.W., Eds.; NeurIPS: Virtual Event, 2021; pp. 21997–22009.
436. Rong, Y.; Huang, W.; Xu, T.; Huang, J. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In Proceedings of the 8th International Conference on Learning Representations (ICLR 2020), Addis Ababa, Ethiopia, 26–30 April 2020; OpenReview.net: Addis Ababa, Ethiopia, 2020.
437. Chien, E.; Peng, J.; Li, P.; Milenkovic, O. Adaptive Universal Generalized PageRank Graph Neural Network. In Proceedings of the 9th International Conference on Learning Representations (ICLR 2021), Vienna, Austria, 3–7 May 2021; OpenReview.net: Vienna, Austria, 2021.
438. Yun, S.; Jeong, M.; Kim, R.; Kang, J.; Kim, H.J. Graph Transformer Networks. In Proceedings of the 32nd Annual Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, BC, Canada, 8–14 December 2019; NeurIPS: Vancouver, BC, Canada, 2019; pp. 11960–11970.

439. Yang, F.; Fan, K.; Song, D.; Lin, H. Graph-based prediction of Protein-protein interactions with attributed signed graph embedding. *BMC Bioinform.* **2020**, *21*, 1–16.
440. Xie, Y.; Li, S.; Yang, C.; Wong, R.C.; Han, J. When Do GNNs Work: Understanding and Improving Neighborhood Aggregation. In Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020), Yokohama, Japan, 7–15 January 2021; Bessiere, C., Ed.; ijcai.org: Yokohama, Japan, 2020; pp. 1303–1309. https://doi.org/10.24963/ijcai.2020/181.
441. Pan, S.; Wu, J.; Zhu, X.; Zhang, C.; Wang, Y. Tri-Party Deep Network Representation. In Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016), New York, NY, USA, 9–15 July 2016; IJCAI Press: New York, NY, USA, 2016; pp. 1895–1901.