SIAM REVIEW
Vol. 63, No. 4, pp. 825–853

Understanding Graph Embedding Methods and Their Applications[*]

Mengjia Xu[†]

Abstract. Graph analytics can lead to better quantitative understanding and control of complex networks, but traditional methods suffer from the high computational cost and excessive memory requirements associated with the high-dimensionality and heterogeneous characteristics of industrial size networks. Graph embedding techniques can be effective in converting high-dimensional sparse graphs into low-dimensional, dense, and continuous vector spaces, preserving maximally the graph structure properties. Another type of emerging graph embedding employs Gaussian distribution–based graph embedding with important uncertainty estimation. The main goal of graph embedding methods is to pack every node's properties into a vector with a smaller dimension; hence, node similarity in the original complex irregular spaces can be easily quantified in the embedded vector spaces using standard metrics. The nonlinear and highly informative graph embeddings generated in the latent space can be conveniently used to address different downstream graph analytics tasks (e.g., node classification, link prediction, community detection, visualization, etc.). In this review, we present some fundamental concepts in graph analytics and graph embedding methods, focusing in particular on random walk–based and neural network–based methods. We also discuss the emerging deep learning–based dynamic graph embedding methods. We highlight the distinct advantages of graph embedding methods in four diverse applications, and we present implementation details and references to open-source software as well as available databases in the supplementary material to help interested readers start their exploration into graph analytics.

Key words. deep neural networks, high-dimensionality, latent space, similarity, uncertainty quantification, intrinsic dimension, graph embedding at scale

AMS subject classifications. 68T07, 05C62, 94A15, 68T37, 68R10, 68T30

DOI. 10.1137/20M1386062

Contents

1 Introduction **826**

2 Mathematical Formulation of the Graph Embedding Problem **828**
2.1 Preliminaries 828
2.2 Graph Embedding Problem Setting 829
2.2.1 Graph Structure Property Preservation 830
2.2.2 Node Similarity Measures in the Original Graphs 831
2.2.3 Similarity Measures in the Embedded Space 833

[*] Received by the editors December 20, 2020; accepted for publication (in revised form) February 23, 2021; published electronically November 4, 2021.
https://doi.org/10.1137/20M1386062
Funding: This work was funded by NIH grant U01 HL1163232 and a J-Clinic for Machine Learning in Health award at MIT.

[†] McGovern Institute for Brain Research, Massachusetts Institute of Technology, Cambridge, MA 02139 USA, and Division of Applied Mathematics, Brown University, Providence, RI 02912 USA (mengjia_xu1@hotmail.com).

3 Overview of Graph Embedding Methods 833
3.1 Vector Point–Based Graph Embedding Methods 833
3.2 Gaussian Distribution–Based Graph Embedding 836
3.2.1 Gaussian Embedding for Knowledge Graphs 837
3.2.2 Gaussian Embedding for Attributed Graphs 839
3.3 Dynamic Graph Embedding 841
3.3.1 Defining Dynamic Graphs 841
3.3.2 Problem Settings for Dynamic Graph Embedding 842

4 Applications 843
4.1 Social Network Applications 844
4.2 Citation Network Applications 844
4.3 Brain Network Applications 845
4.4 Genomic Network Applications 847

5 Summary and Discussion 849

References 850

1. Introduction.

Graphs are a universal language for describing and modeling complex systems. Network data are ubiquitous across diverse application fields such as brain networks [41, 57, 56] in brain imaging, molecular networks [26] in drug discovery, protein-protein interaction networks [21] in genetics, social networks [51] in social media, bank-asset networks [61] in finance, and publication networks [53] in scientific collaborations. Unlike the regular grid-like Euclidean space data (e.g., images, audio, and text), the aforementioned network data are derived from irregular non-Euclidean domains. However, as a nonlinear data structure, a graph can be used as an effective tool to describe and model the complex structure of network data. Specifically, a graph $G(V, E)$ is typically composed of a set of vertices (or entities), denoted by V, and certain edges (or relations), denoted by E, between different vertices. Modeling complex systems as graphs facilitates the characterization of very useful high-order geometric patterns for the networks, which has a great impact on improving the performance of different network data analysis tasks, e.g., gene network reconstruction, protein function prediction, and human face recognition.

Graph analytics (also known as network analysis) has become an exciting and impactful research area in recent years. Developing effective and efficient graph analytics can greatly help to better understand complex networks. However, traditional graph analytics, including path analysis, connectivity analysis, community analysis, and centrality analysis, is largely based on extracting handcrafted graph topological features directly from the adjacency matrices. When we apply these methods to large-scale network analysis in industrial systems, they may suffer from high computational cost and excessive memory requirements due to the challenging and inevitable high-dimensionality and emerging heterogeneous characteristics of the original networks [43]. Moreover, the hand-engineered features are usually task-specific and cannot achieve equivalent performance when employed for different tasks.

Recently, graph embedding techniques (mainly referred to as node embedding) have shown a remarkable capacity to convert high-dimensional sparse graphs into low-dimensional, dense, and continuous vector spaces (see Figure 1), where graph structure properties are maximally preserved [9]. The nonlinear and highly informative graph embeddings (or features) generated in the latent space can be conveniently used to address different downstream graph analytic tasks (e.g., node classification, link prediction, community detection, and visualization). The main aim of graph embedding methods is to encode nodes into a latent vector space, i.e., pack every node's properties into a vector with a smaller dimension. Hence, node similarity in the original complex irregular spaces can be easily quantified based on various similarity measures (such as dot product and cosine distance) in the embedded vector spaces. Furthermore, the learned latent embeddings can greatly support much faster and more accurate graph analytics as opposed to directly performing such tasks in the high-dimensional complex graph domain. In addition to node embedding, there are other types of embedding, e.g., edge embedding, substructure embedding, and whole-graph embedding; see the survey paper [9].

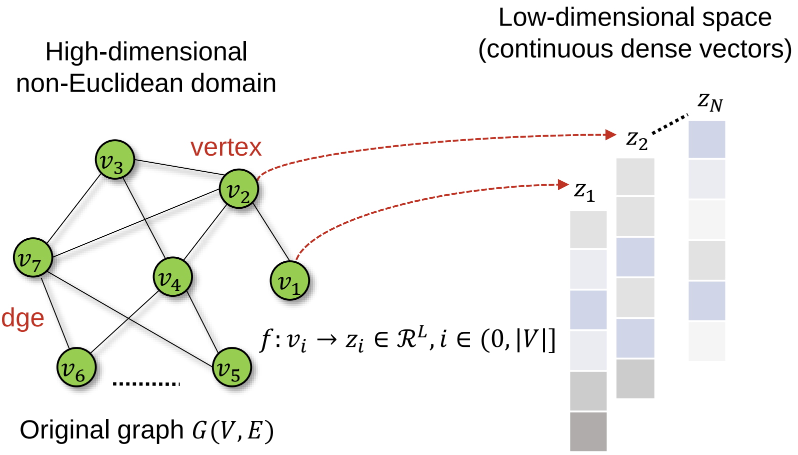


Fig. 1 ***Schematic of graph (node) embedding***. *For a simple graph $G(V, E)$ consisting of a node set V and an edge set E, using a graph embedding model f, different nodes (e.g., v_1 and v_2) from the original graph in a high-dimensional irregular domain can be mapped into a latent low-dimensional space as an L-dimensional dense and continuous vector z_i, $L \ll |V|$. The node structure property can be also preserved in the latent space, i.e., similar nodes in the original space will be close to each other in the latent space. Moreover, the latent variables $z_i, i \in V$ (i.e., features) obtained can be readily used for diverse downstream graph analytic tasks.*

A broad taxonomy of graph embedding methods used in the literature points to three major categories [12]: matrix factorization–based methods, random walk–based methods, and neural network–based methods. Matrix factorization–based methods (e.g., [10, 2, 38, 42]) construct a high-order proximity matrix based on transition probabilities and factorize it to obtain the node embeddings, but they cannot easily scale up to large network embeddings. We refer the reader also to [9, 12] for earlier surveys of graph embedding methods, which cover whole-graph embedding (graph kernel–based methods) [9] as well as task-specific graph embedding methods that incorporate domain knowledge [12]. In this paper, we focus on the more scalable random walk–based methods and neural network–based methods that incorporate uncertainty quantification. Furthermore, in addition to the static graph embedding methods, we also discuss the emerging deep learning–based *dynamic* graph embedding methods. Finally, we highlight the distinct advantages of graph embedding methods in four diverse applications and conclude with a summary including a discussion on graph embedding in non-Euclidean spaces, e.g., hyperbolic graph embedding. In the supplementary material, we include information about open-source software, available data, and implementation details.

2. Mathematical Formulation of the Graph Embedding Problem.

In this section, we first present some preliminary graph notation, definitions, and properties used in graph embedding. Then we review the node similarity measures frequently applied in graph embedding. Finally, we formulate the specific graph embedding problem setting.

2.1. Preliminaries.

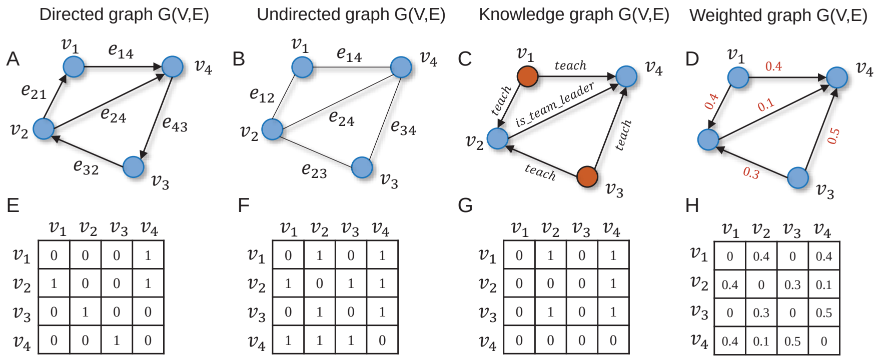
We consider a graph $G(V, E)$ as a mathematical data structure that contains a node (or vertex) set $V = \{v_1, v_2, v_3, \ldots, v_n\}$ and an edge (or link) set E. In the edge set, one edge e_{ij} describes the connection between two different nodes v_i and v_j, and hence e_{ij} can be represented as (v_i, v_j), where $v_i, v_j \in V$ and nodes v_i and v_j are adjacent nodes. From different perspectives (i.e., edge direction, diversity of nodes and edges, edge cost), graphs can be classified into different categories: directed/undirected graphs, homogeneous/heterogeneous graphs, or weighted/binary graphs. We present more details below.

(1) *Directed/undirected graphs:* Regarding the "directed graph" (also called digraph) as shown in Figure 2(A), every edge has a specific direction, e.g., flight networks, Google maps. Conversely, in an "undirected graph," edges have no directions; see an example in Figure 2(B). Since edges in undirected graphs are unordered pairs, the edge e_{12} can be also replaced with e_{21}. An undirected graph can be also viewed as a bidirectional graph.

(2) *Homogeneous/heterogeneous graphs:* For homogeneous graphs, all nodes or edges are of the same type, e.g., friendship network with nodes denoting different people and edges denoting their friendship. In contrast, heterogeneous graphs consist of multiple types of nodes and/or edges. A knowledge graph is a typical directed heterogeneous graph; see the example in Figure 2(C) showing an education network. Different node colors refer to different node types, i.e., brown nodes stand for "Teacher" type while blue ones stand for "Student" type. Moreover, there are two edge types ("teach" and "is_team_leader") in the knowledge graph.

(3) *Weighted/binary graphs:* For weighted graphs, every edge is assigned a specific numerical value (i.e., weight); see the example in Figure 2(D). By contrast, binary graphs (or unweighted graphs) do not have any weight associated with edges.

Given different graph edge densities and types of operations applied on a graph, a graph can be represented in three different ways. (i) *Adjacency matrix (A):* A is a $|V| \times |V|$ 2D square matrix with element $\{s_{i,j} | i, j \in (0, |V|]\}$ representing whether or not two nodes are connected in a graph, where $|V|$ is the total number of vertices in the graph. Specifically, if nodes v_i and v_j are connected, $s_{i,j} = 1$ for a binary graph (note that $s_{i,j} = w$ for a weighted graph, where w is the edge weight); otherwise, $s_{i,j} = 0$. Moreover, A is a symmetric matrix for undirected graphs and asymmetric for directed graphs; see examples in Figure 2(E)–(H). Furthermore, in order to find the neighbors of each node, using the adjacency matrix we have to traverse every node's adjacent nodes, which has computational complexity of the order of $O(|V|)$. Therefore, performing this operation for all nodes of the adjacent matrix costs $O(|V|^2)$, which is not appropriate for representing large sparse graphs. (ii) *Adjacency list:* This makes use of a linked list to store only each node's adjacent nodes (see the detailed example in Figure 3) and so it is convenient for node operations (i.e., insert, delete, or



E

	v_1	v_2	v_3	v_4
v_1	0	0	0	1
v_2	1	0	0	1
v_3	0	1	0	0
v_4	0	0	1	0

F

	v_1	v_2	v_3	v_4
v_1	0	1	0	1
v_2	1	0	1	1
v_3	0	1	0	1
v_4	1	1	1	0

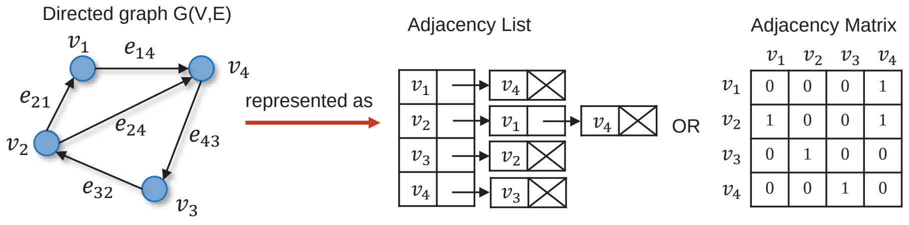
G

	v_1	v_2	v_3	v_4
v_1	0	1	0	1
v_2	0	0	0	1
v_3	0	1	0	1
v_4	0	0	0	0

H

	v_1	v_2	v_3	v_4
v_1	0	0.4	0	0.4
v_2	0.4	0	0.3	0.1
v_3	0	0.3	0	0.5
v_4	0.4	0.1	0.5	0

Fig. 2 ***Different types of graphs and their corresponding adjacency matrix representations.*** *The first row from* (A) *to* (D) *shows, respectively, directed, undirected, knowledge, and weighted graph examples. The main difference between* (A) *and* (B) *is that edges are directed in* (A) *but undirected in* (B). (C) *is a knowledge graph consisting of two different types of nodes (in brown and blue) and two different types of edges ("teach" and "is_team_leader"). Graph* (C) *is an instance of a directed and heterogeneous graph.* (D) *shows a weighted graph where every edge is weighted with a specific value. The second row from* (E) *to* (H) *shows the corresponding* 4×4 *adjacency matrices for graphs* (A)–(D).



	v_1	v_2	v_3	v_4
v_1	0	0	0	1
v_2	1	0	0	1
v_3	0	1	0	0
v_4	0	0	1	0

Fig. 3 ***Different ways to represent a static graph.*** *Beyond the adjacency matrix representation shown in Figure* 2(E)–(H), *a static graph can be also represented as an "adjacency list" or "incidence matrix."*

add nodes), and its space cost is only $O(|E|)$, which benefits effective representations for large sparse graphs (i.e., $|E| \ll |V| \times |V|$). (iii) *Incidence matrix:* This employs a $|V| \times |E|$ matrix to represent the relationship between node set and edge set in a graph. More details can be seen in Figure 3, where each column denotes different edges and each row denotes different nodes in a directed graph. Each element in the matrix can be filled with "1" (the column edge is one outgoing edge from the row node); "0" (the column edge is not connected with the row node); "−1" (the column edge is one incoming edge to the row node); for undirected graphs, elements with "−1" are all filled with "1."

2.2. Graph Embedding Problem Setting. In line with the aforementioned graph notations and definitions, given a graph $G = (V, E)$, the task of learning its graph node embeddings (e.g., L dimension, $L \ll |V|$) can be mathematically formulated as finding a projection ϕ such that all graph nodes ($V = \{v_i | i = 1, 2, \dots, |V|\}$) can be encoded as two different embedding forms from a high-dimensional space into a low-dimensional space. One node embedding form is deterministic point vectors

$(\Phi = \{z_i \in \mathbb{R}^L | i = 1, 2, \ldots, |V|\})$, while another is stochastic probabilistic distributions $(\Phi = \{\mathbb{P}_i \sim \mathcal{N}(\mu_i, \Sigma_i) | i = 1, 2, \ldots, |V|\})$, where the mean vector $\mu_i \in \mathbb{R}^{L/2}$ and the covariance matrix $\Sigma_i \in \mathbb{R}^{L/2 \times L/2}$. Moreover, graph structure properties are maximally preserved in the embedded space. To this end, node pairwise similarity (e.g., dot product $z_j^T z_i$) in the embedded latent space facilitates an approximation of the corresponding node similarity $(Sim(v_i, v_j))$ in the original space, i.e., $Sim(v_i, v_j) \approx z_j^T z_i$ for vector–based graph embedding; specifically, $Sim(v_i, v_j) \approx \mu_j^T \mu_i$ for Gaussian distribution–based graph embedding. Here, Sim is a predefined similarity function (see Table SM1 in the supplementary material and more details in subsection 2.2.3). The specific graph node embedding formula is shown in (2.1), i.e.,

$$\phi : v_i \rightarrow \begin{cases} z_i \in \mathbb{R}^L \ \ (i = 1, 2, \ldots, |V|) \ \text{ or} \\ P_i \sim \mathcal{N}(\mu_i, \Sigma_i), \mu_i \in \mathbb{R}^{L/2}, \Sigma_i \in \mathbb{R}^{L/2 \times L/2} \ \ (i = 1, 2, \ldots, |V|). \end{cases} \tag{2.1}$$

Generally, the main goal of graph embedding is to encode nodes into low-dimensional space, such that similarity in the latent embedded space approximates similarity in the original high-dimensional graph, while preserving the graph structure property. Essentially, most of the advanced graph node embedding techniques learn low-dimensional node representations by solving an optimization problem, which follows an *unsupervised* learning schema and is independent of downstream prediction tasks. Specifically, graph embedding approaches contain three major components: graph structure property preservation, node similarity measurement in both the original and the latent space, and an encoder. Next, we present more details about these three key components.

2.2.1. Graph Structure Property Preservation. Proximity measures play an important role in the quantification of graph structure property preservation for diverse graph embedding methods. Specifically, we present three different types of proximity measures.

(1) *First-order proximity*: This is used to characterize local pairwise similarity between nodes linked by edges, i.e., the local network structure property. However, it is insufficient for preserving the global network structure. First-order proximity between two nodes is equal to their edge weight ($w_{ij} = 1$ for a binary graph). Moreover, in order to develop a graph embedding model preserving the first-order proximity, the objective function $(\mathbb{E})$ can be constructed in terms of the KL-divergence (D_{KL}) as in (2.2), where $\hat{p}_1(v_i, v_j)$ and $p_1(v_i, v_j)$ represent the corresponding empirical distribution and model distribution of first-order proximity, respectively, which are modeled as joint probabilities.

$$\begin{aligned} &\mathbb{E} = D_{KL}(\hat{p_1} || p_1) = - \sum_{i,j \in E} w_{ij} \log(p_1(v_i, v_j)), \\ &\text{where } \ \hat{p}_1(v_i, v_j) = \frac{w_{ij}}{\sum\limits_{s,t \in E} w_{st}}, \ \ p_1(v_i, v_j) = \frac{exp(z_i^T z_j)}{\sum\limits_{s,t \in E} exp(z_s^T z_t)}. \end{aligned} \tag{2.2}$$

Here, v_i and v_j represent the original two nodes i and j; z_i, z_j are the corresponding embedding vectors of v_i and v_j; and w_{ij} denotes the weight between two nodes i and j.

(2) *Second-order proximity*: This is used to capture the proximity between the neighborhood structures of the nodes [9], i.e., the global network structure property. Second-order proximity can be easily estimated using the transitional probability metric between two nodes. In order to develop a graph embedding model preserving the second-order proximity, the objective function $(\mathbb{E}^{'})$ is defined as in (2.3), where

$\hat{p}_2(v_i|v_j)$ and $p_2(v_i|v_j)$ represent the corresponding empirical distribution and model distribution of the neighborhood structure that are modeled as two different conditional probabilities as follows:

$$\begin{aligned} \mathbb{E}^{'} &= \sum_{i\in V} D_{KL}(\hat{p}_2(\cdot|v_i)||p_2(\cdot|v_i)) = -\sum_{i,j\in E} w_{ij}\log(p_2(v_j|v_i)), \\ &\text{where}\quad \hat{p}_2(v_j|v_i) = \frac{w_{ij}}{\sum\limits_{k\in V} w_{ik}},\;\; p_2(v_j|v_i) = \frac{exp(z_j'^T z_i)}{\sum\limits_{k\in V} exp(z_k'^T z_i)}. \end{aligned} \tag{2.3}$$

Here, z_i' is the representation of v_i when it is treated as a specific "context" [45].

(3) *High-order proximity*: There are few works in the literature focusing on high-order proximity quantification, as second-order proximity works well in most graph embedding methods. Usually, high-order proximity can be defined based on metrics such as Rooted PageRank [6].

2.2.2. Node Similarity Measures in the Original Graphs. Selecting a proper node similarity measure is particularly important for finding the effective node context (or neighbors) helpful for optimizing diverse graph embedding models. Similarity between nodes in the original graphs can be characterized based on five different aspects: (1) presence of edges, (2) overlapped neighborhood, (3) reachable by k-hops, (4) reachable through random walks, and (5) similar node attributes. Accordingly, node similarity definitions commonly used in existing graph embedding methods mainly consist of three different types: multihop neighborhood based [4], random walk based [39, 18, 45], and adjacency matrix based [42, 2, 38]. The key idea of graph embedding is to optimize low-dimensional embeddings to approximate the node similarities in the original graph generated by the above measures.

(1) *Multihop neighborhood–based node similarity:* The "hop" terminology originally appeared in the wired computer networking field, and "hop count" can provide a measure of distance between source host and destination host [22]. Recently, in order to sample node neighbors in the original graphs and preserve the graph structure property, the multihop neighborhood (or k-hop neighborhood) method has become an effective solution to address this problem in graph embedding. Specifically, the k-hop neighborhood is defined as the set of vertices that are reachable from a source node in k hops or fewer (i.e., following a path with k edges or fewer in binary graphs); see an example of a 3-hop neighborhood in Figure 4(A). In particular, the k-hop neighborhood searching method enables us to sample similar neighbors for every sampled source node at different hops in the original graphs. In addition, higher-order proximity can be preserved by increasing the number of hops. For instance, the Graph2Gauss method [4] that we study in subsection 3.2 evaluates the 2-hop and 3-hop node neighborhood sampling strategies for binary graph embedding, and tests show that $k = 2$ is sufficient for preserving the graph structure property in high-order proximity.

(2) *Random walk–based node similarity:* Random walks have been used as stochastic similarity measures for various problems (e.g., content recommendation [36], community detection [37], and image segmentation [3, 55]). In particular, the node similarity ($Sim(v_i, v_j)$) in the complex graph can be defined as the probability $p(v_j|v_i)$ of reaching a node v_j on a random walk ($W_{v_i} = r_0, r_1, \ldots, r_l$) of length l over the graph from the source node $r_0 = v_i$; see the random walk example in Figure 4(B). In particular, the nodes in the random walk are generated based on the distribution described in (2.4), where r_i denotes the ith node in the random walk starting from $r_0 = u$, π_{vx} denotes the unnormalized transition probability between nodes v and x,

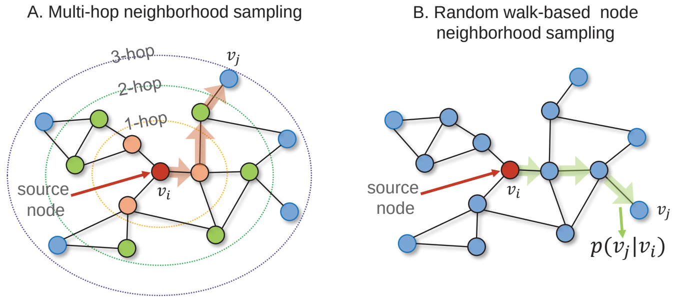

A. Multi-hop neighborhood sampling
3-hop
2-hop
1-hop
v_j
source node
v_i
B. Random walk-based node neighborhood sampling
source node
v_i
v_j
$p(v_j|v_i)$

Fig. 4 ***Node neighborhood sampling techniques.*** (A) *Multihop neighborhood–based node neighbor sampling. The red node* (v_i) *is taken as the source target node, and nodes of different colors represent the sampled nodes at different hops;* v_j *is one of the* 3*-hop neighbors of* v_i*. The orange arrows mark the shortest path ranging from* v_i *to* v_j*.* (B) *Random walk–based node neighbor sampling strategy: every node neighborhood is sampled based on the transition probability computed using* (2.4)*.*

and Z is a normalizing constant [18],

$$p(r_i = x|r_{i-1} = v) = \begin{cases} \pi_{vx}/Z & \text{if } (v,x) \in E, \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

In the earlier DeepWalk methods [39], a random walk is generated (similar to a sampled "sentence" in nature language processing applications) for each node (comparable to "word" in the language modeling problems) in the graph following the random search strategy as shown in (2.4). It then learns the node embeddings through predicting the nearby nodes cooccurring on the generated random walk using a language embedding model (i.e., the SkipGram model [31]). Moreover, other methods that we discuss in subsection 3.1, such as LINE [45] and node2vec [18], also apply modified random walk strategies to sample similar neighbors (or contexts) for different nodes in the original graph. Compared with the multihop neighborhood–based measure, the major advantage of the random walk–based similarity measure is that it does not need to train all the node pairs, but only considers the node pairs that cooccur on random walks.

(3) *Adjacency matrix–based node similarity:* Node similarity can be also characterized by the presence of edges between nodes in the adjacency matrices of original graphs. For example, the classic matrix factorization–based graph embedding methods (e.g., locally linear embedding [42], Laplacian eigenmaps [2], and HOPE [38]) (see section 4) and the stochastic neighborhood embedding (SNE) method [20] all employ adjacency matrix–based node similarity measures for graph embedding, and have achieved good performance in graph reconstruction problems. However, if we use the node similarity measure based on the adjacency matrix to carry out graph embedding, the obtained node embeddings will usually overfit the adjacency matrix of the original graph. Hence, the method does not work well on inconspicuous connection detection (such as downstream link prediction tasks) [33]. Moreover, due to the computational complexity $O(|V|^2)$, adjacency matrix–based measures are hard to scale up to large-scale networks. Beyond these limitations, adjacency matrix–based similarity measures only consider local connections.

2.2.3. Similarity Measures in the Embedded Space. According to the specific learned node representation/embedding form (e.g., vector point representations and Gaussian distribution representations) in the latent space using different graph embedding methods, we can employ different metrics (M) to measure node similarity in the low-dimensional embedded space. Specifically, given two randomly selected nodes v_i, v_j from an original graph G, where z_i, z_j are the corresponding point vector node embeddings in the latent space, the similarity measures (M) for the point vector node embeddings can be computed by measures such as dot product, cosine similarity, and Euclidean distance (see detailed formulas in Table SM1 in the supplementary material). However, for node embeddings as Gaussian distributions in the latent space, the similarity measures frequently used in recent works involve expected likelihood (EL), KL-divergence (D_{KL}), and the Wasserstein distance (W_2) (see Table SM1). $P_i \sim \mathcal{N}(\mu_i, \Sigma_i)$ and $P_j \sim \mathcal{N}(\mu_j, \Sigma_j)$ are the learned stochastic node representations (i.e., multivariate Gaussian distributions) in the latent space for nodes v_i and v_j, where μ_i and Σ_i denote the learned mean vector and covariance matrix for node v_i, and μ_j and Σ_j are the learned mean vector and covariance matrix for node v_j.

3. Overview of Graph Embedding Methods.

According to the mathematical problem settings of graph embedding in section 2, we further subdivide the prevalent graph embedding methods into three main categories: (1) vector point–based graph embedding, (2) Gaussian distribution–based graph embedding, and (3) dynamic graph embedding. More details about these three types of methods are presented below.

3.1. Vector Point–Based Graph Embedding Methods. Vector point–based graph embedding methods can be further divided into three main types: (a) matrix factorization–based methods (GraRep [10], HOPE [38]), (b) random walk–based methods (e.g., DeepWalk [39], LINE [45], node2vec [18]), and (c) deep learning–based methods (SDNE [48]). The main purpose of vector point–based graph embedding is to project high-dimensional graph nodes into low-dimensional vectors in a latent space, while preserving the original graph structure properties; see a specific definition in (2.1). Nodes that are "close" in the original graph are embedded in a latent space with similar "vector representations." The "closeness" between variant nodes in the original space can be modeled in different ways by using the measures presented in the subsection 2.2.2. Specifically, matrix factorization–based graph embedding methods measure the "closeness" degree between nodes via the adjacency matrix–based measures, while random walk–based graph embedding methods apply random walk techniques to extract close node neighbors.

Since we have already reviewed the advantages and limitations of adjacent matrix–based methods in (3) of subsection 2.2.2, here we mainly focus on summarizing the random walk–based graph embedding methods including the main pipeline and key techniques. Given a graph $G(V, E)$, the main pipeline of a random walk–based graph node embedding method for G is shown in Figure 5. It involves three key stages, described in detail as follows.

(1) Stage 1: Node context generation based on random walks. Motivated by the "word2vec" method [31] developed for learning word representations based on sentences, the authors of DeepWalk [39] first adopted the random walk–based node neighbors sampling technique (see subsection 2.2.2) to extract the node context (similar to "sentence"), and then employed an encoder called the "SkipGram" model [31] to learn graph node embeddings based on the sampled similar node pairs in the random

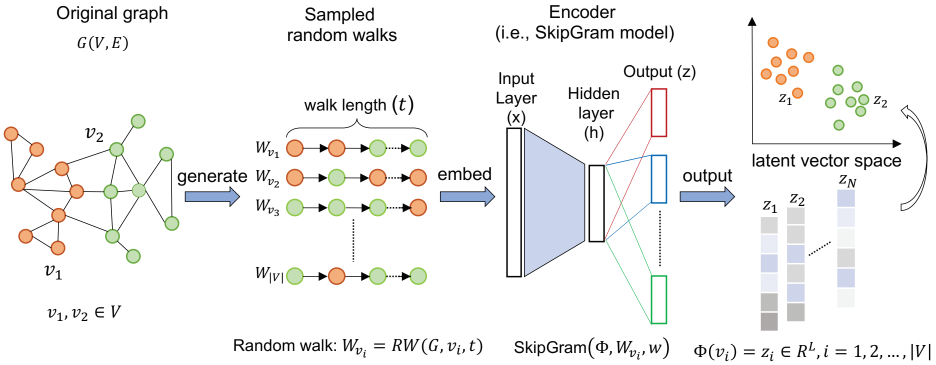


Fig. 5 ***Pipeline for random walk–based graph embedding methods.*** *In order to learn node embeddings for the original graph $G = (V, E)$ consisting of two types of nodes marked in different colors, we apply random walk methods to first generate a set of node context (W_{v_i}) for every node ($v_i \in V$); the sampled node contexts (i.e., random walks) are of the same fixed walk length t. Second, based on the generated node contexts, a language embedding model plays the role of an encoder such that every node is represented as a low-dimensional, continuous vector in the latent space. The distance (e.g., dot product, cosine similarity, or Euclidean distance) between vectors (or "node embeddings") in the latent vector space approximates the similarity in the original graph. Additionally, the learned vectors can be simply mapped to 2D space as points using dimension reduction techniques (e.g., t-SNE, MDS, PCA). The learned node embedding features ($\Phi \in \mathbb{R}^{|V| \times L}$) for all nodes can be readily and efficiently used for different downstream tasks, such as link prediction, node classification, and community detection.*

walks. In this way, it can effectively encode the structure and topological information from the original graph into the latent space. However, DeepWalk [39] can capture only the local structure information by using the truncated random walks, and the global structure information is missing. To address this problem, node2vec [18] employs an improved *biased random walk* method to sample node context by considering both local and global structure information from the original graph. Figure 6 shows a detailed schematic of node neighbor expansion using the biased random walk approach (or "second-order random walk"), which incorporates both BFS (breadth-first search) and DFS (depth-first search) searching strategies with two ratio parameters (p and q). In general, nodes in a random walk are generated by using the formula defined in (2.4); however, in the "biased random walk," the unnormalized transition probability π_{vx} in (2.4) is modified using a search bias (α) in conjunction with edge weight k_{vx} ($k_{vx} = 1$ if binary graphs) to guide node neighbor searching. Specifically, assume a random walk has just traversed nodes t, v and resides at v; then the unnormalized transition probability (π_{vx}) between the node v and the next walk node x can be computed as follows:

$$\pi_{vx} = \alpha_{pq}(t,x) \cdot k_{vx}, \quad \text{where} \quad \alpha_{pq}(t,x) = \begin{cases} 1/p & \text{if} \quad d_{tx} = 0, \\ 1 & \text{if} \quad d_{tx} = 1, \\ 1/q & \text{if} \quad d_{tx} = 2, \end{cases} \tag{3.1}$$

where the search bias (α_{pq}) is defined by a return rate parameter (p) and an "in-out" exploration rate parameter (q); d_{tx} represents the shortest distance between the previous visited node t and the next visiting nodes x; and p and q control how fast the

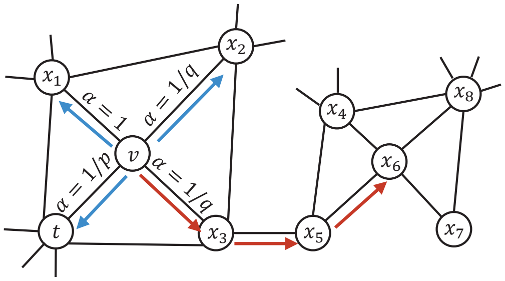


Fig. 6 ***Biased random walk schema incorporating both BFS and DFS.*** *Starting from the source target node* (v) *to generate a random walk with fixed length* ($l = 4$) *over the graph, blue arrows show the BFS directions, while red arrows indicate the DFS directions. The DFS strategy helps in expanding the node neighbors deeper and enables exploration of node neighbors (e.g.,* x_6*) that have similar structure roles to the starting node (i.e.,* v*) beyond the directly connected node neighbors (e.g.,* x_1, x_2, x_3*). Hence, second-order proximity can be also preserved. The biased random walk strategy proposed in node2vec* [18] *achieves a trade-off of BFS and DFS; the explored neighborhood search is guided by using a search bias parameter* (α)*, which is defined by the return rate* (p) *and exploration rate* (q) *as shown in* (3.1)*. We provide more details about these two parameters in* (1) *in subsection* 3.1*.*

walk explores and leaves the starting node (v) (see Figure 6). In particular, the larger p ($> \max(q, 1)$) becomes, the less likely it is to sample an already visited node, i.e., *moderate exploration*; inversely, when p ($< \min(q, 1)$), it will lead to backtracking of the step, which results in *local neighborhood sampling*. Moreover, q is used to control the exploration of "inward" and "outward," i.e., when $q > 1$, it approximates the BFS behavior and leads the random walks toward a *microview* of the node neighborhoods, whereas $q < 1$ leads to a "DFS-like" *macroview exploration* of the node neighborhoods. Additionally, when $p = q$, the node2vec method is the same as DeepWalk.

Moreover, the selection of window size (w) and walk length (t) also plays an important role in generating an effective node context, because it can implicitly influence the number of node neighbors covered in the random walks and affects the constraints of node similarity. The larger the walk length t used, the more noisy node cooccurrences introduced in the graph embedding.

(2) Stage 2: Learn node embedding with an encoder. Aiming to transform the graph nodes to low-dimensional vectors, the encoder is a key and necessary component in graph embedding techniques. The SkipGram model [31] is a typical language embedding model, and it can be also employed as an effective encoder for prevalent graph node embedding problems by feeding the generated node context from Stage 1 and outputting the low-dimensional node embeddings. Specifically, learning a graph node embedding ($\Phi = \{z_i\}_{i=1}^{|V|}, z_i \in \mathbb{R}^L$) corresponds to minimizing the negative log-likelihood of observing its neighborhood nodes ($v \in N_R(u)$) conditioned on the predicted source node's embedding (z_u), which is the output of the encoder (i.e., the SkipGram model). The parameters of the embedding model can be learned by minimizing the cross-entropy loss function ($\mathcal{L}$) as shown in (3.2):

$$\mathcal{L} = -\log \sum_{u \in V} \sum_{v \in N_R(u)} p(v|z_u) = -\log \sum_{u \in V} \sum_{v \in N_R(u)} \frac{\exp(z_u^T z_v)}{\sum\limits_{n \in V} \exp(z_u^T z_n)}, \tag{3.2}$$

where V refers to the entire node set of the original graph, $N_R(u)$ denotes the sampled neighbor set for the source node u using some neighborhood sampling strategy R (here, we refer to "random walks"), and v refers to the nearby nodes of node u. In practice, DeepWalk [39] and node2vec [18] have the same objective function as shown in the first part of (3.2), but they adopt different optimization policies. In particular, DeepWalk [39] is optimized based on the hierarchical softmax [39]; the simplified loss function with the hierarchical softmax is shown in the last part of (3.2), which we need to normalize with respect to all nodes with high computational complexity ($O(|V|)^2$). However, node2vec [18] applies stochastic gradient descent (SGD) with "negative sampling" for more efficient model optimization. According to (3.2), the simplified objective function ($\mathcal{L}^{'}$) for node2vec using "negative sampling" is given as follows:

$$\mathcal{L}^{'} = -\sum_{u\in V}\sum_{v\in N_R(u)}\left(\log(\sigma(z_u^T z_v)) - \sum_{i=1}^{k}\log(\sigma(z_u^T z_{n_i}))\right), \quad n_i \sim P_v, \tag{3.3}$$

where σ denotes the sigmoid function and n_i denotes the random sampled negative node neighbors, which follow a random distribution over all nodes. To find graph node embeddings with the objective function in (3.3), we just need to normalize against "k" random negative samples, which can effectively speed up the training process and greatly reduce the computational complexity.

(3) Stage 3: Output low-dimensional node embeddings for downstream tasks. With the node embedding learning with an encoder described above, all nodes in the original graph are transformed into low-dimensional continuous vectors ($\Phi \in \mathbb{R}^{|V|\times L}$) in the latent space, which are shown in the last column of Figure 5. Each vector can be easily projected and visualized in 2D space as a point based on t-SNE [20] or PCA [40] techniques, where similar nodes in the original space will be close to each other in the latent space. Hence, similar nodes will be clustered together in the 2D space, and different types of nodes can be characterized by different clusters. The obtained node representations can provide effective task-independent node features for solving different downstream tasks (e.g., link prediction, node classification, and node importance) with only simple traditional machine learning classifiers (e.g., SVM, random forest, and k-means). Moreover, the low-dimensional vectors can also be further computed in different forms to feed into classifiers, such as concatenation, average, or Hadamard product.

"Vector point–based graph embedding" methods (summarized in Table SM3 of the supplementary material) show great effectiveness in learning graph representations for diverse downstream tasks. However, the main drawback of such methods is that the most important network *uncertainty information* is not captured. Predicting the *uncertainty* for node embedding is particularly important for quantitative analysis of large and complex network systems with diverse nodes and heterogeneous relations.

3.2. Gaussian Distribution–Based Graph Embedding.

Beyond the deterministic vector point–based graph embedding method in subsection 3.1, another type of emerging graph embedding method (e.g., G2G [4], KG2E [19], DVNE [62]) named "Gaussian distribution–based graph embedding" or "probabilistic graph embedding" holds great promise for stochastic graph embedding with important uncertainty estimation. Unlike the *deterministic* vector point–based graph embedding method introduced in subsection 3.1, Gaussian distribution–based graph embedding enables the learning of node embeddings as "potential functions" or "continuous densities" in latent space, which leads to two major advantages: (a) it enables the effective incorporation of useful *unstructured attribute information* associated with each node, (b) every node from the original graph is encoded as a low-dimensional *multivariate Gaussian distribution* ($P_i \sim \mathcal{N}(\mu_i, \Sigma_i)$) in terms of the mean vector ($\mu_i \in \mathbb{R}^{L/2}$) and covariance matrix ($\Sigma_i \in \mathbb{R}^{L/2 \times L/2}$); the learned covariance term can provide extra free but important *uncertainty information.* The mathematical problem definition for the Gaussian distribution–based graph embedding is shown in (2.1). We summarize some recent works based on graph Gaussian embedding in Table SM2 of the supplementary material.

The graph Gaussian embedding technique is inspired by the word2Gauss approach [47], which embeds words as Gaussian distributional potential functions in an infinite-dimensional functional space. Therefore, each word is mapped into a "soft region" in latent space rather than a single point, which allows us to explicitly model the uncertainty, entailment, and inclusion, as well as providing a rich geometry for better quantification of the word-type properties in the latent space. The aforementioned distinct properties using the Gaussian embedding technique facilitate better representation of the word properties in latent space with uncertainty quantification. In addition, it is applicable for solving more complex graph representation learning tasks. In the following, we summarize the current graph Gaussian embedding methods from two graph types: (1) Gaussian embedding for knowledge graphs, and (2) Gaussian embedding for attributed graphs. Generally, there are three main components for learning a graph Gaussian embedding model: *triplet pair generation, energy function*, and *loss function.*

3.2.1. Gaussian Embedding for Knowledge Graphs. He et al. [19] first proposed a Gaussian embedding model (KG2E) for learning stochastic knowledge graph (KG) representations in terms of Gaussian distributions, enabling the modeling of uncertainty of relations and entities in the KG using two benchmarks (WordNet [32] and Freebase [5]). Figure 7 presents a density-based KG Gaussian embedding example for both "entities" (i.e., nodes) and "relations" (i.e., links) in the KG. Different entities and relations are transformed into a latent space of Gaussian distributions. The KG2E model is learned based on an energy-based learning framework [23]. In particular, it learns graph Gaussian representations (or embeddings) by using a *margin-based triplet ranking loss* ($\mathcal{L}$) as shown in (3.4) with stochastic gradient descent (SGD) and the negative sampling technique. Hence, it can push the scores of positive triplets (E_{pos}) greater than the scores of negative triplets (E_{neg}) by a predefined margin (γ),

$$\mathcal{L} = \sum_{(h,r,t)\in\tau} \sum_{(h',r',t')\in\tau'} \max(0, E_{pos}(h,r,t) - \gamma + E_{neg}(h',r',t')). \tag{3.4}$$

In (3.4), $(h,r,t) \in \tau$ denotes a *positive triplet* sample from the KG consisting of a relation (r) between the head entity (h) and tail entity (t), and $(h',r',t') \in \tau'$ denotes the corresponding *negative triplet* sample built by replacing h or t randomly (e.g., (h',r,t) or (h,r,t')) using "unif" and "bern" techniques [52]. In order to measure the similarity between the learned entity (node) and relation (link) embeddings in each positive and negative triplet sampled from the original KG, there are two different ways to define the energy function: (a) Expected likelihood ("EL") of inner product–based symmetric similarity, with the specific formula for denoting a positive triplet EL energy function $E_{pos}(h,r,t)$ shown in (3.5); (b) KL-divergence (asymmetric similarity), with the specific formula for denoting the positive triplet KL energy function

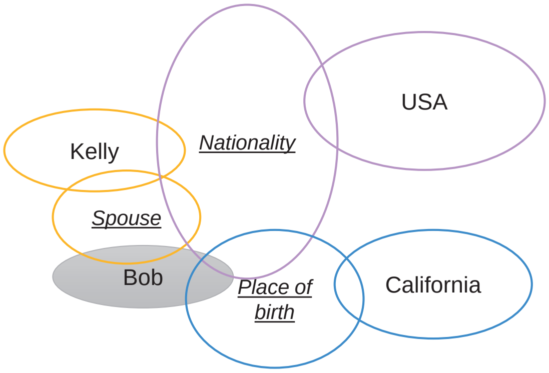


Fig. 7 ***Example of density-based KG Gaussian embedding in latent space.*** *Four entities ("Bob," "Kelly," "USA," "California") and three types of relations ("Nationality," "Spouse," "Place of birth") are shown in the diagram. Different colors indicate different facts/triplets, e.g., (Bob, spouse, Kelly) about the entity ("Bob"). Every fact (or triplet) consists of two nodes (head node and tail node) and a relation between these two nodes. Since Gaussian embedding is a density-based embedding approach, every entity (i.e., node) can be embedded into a soft region (i.e., ellipse) represented as a multivariate Gaussian distribution with mean and diagonal variances. The variances provide the corresponding uncertainty. Based on the density-based embeddings, we can infer that "Bob was born in California"; on the other hand, the relation ("Spouse") has lower uncertainty (i.e., smaller variances) than the relation ("Nationality") while inferring a person (e.g., "Bob").*

$E_{pos}(h, r, t)$ shown in (3.6):

$$\begin{aligned} E_{pos}(h, r, t) &= E_{pos}(P_e, P_r) \\ &= \int_{x \in \mathbb{R}^{k_e}} \mathcal{N}(x; \mu_e, \Sigma_e)\mathcal{N}(x; \mu_r, \Sigma_r)dx \\ &= \mathcal{N}(x; \mu_e - \mu_r, \Sigma_e + \Sigma_r), \end{aligned} \tag{3.5}$$

$$\begin{aligned} &E_{pos}(h, r, t) = E_{pos}(P_e, P_r) = D_{KL}(P_e || P_r) \\ &= \int_{x \in \mathbb{R}^{k_e}} \mathcal{N}(x; \mu_r, \Sigma_r) \log \frac{\mathcal{N}(x; \mu_r, \Sigma_r)}{\mathcal{N}(x; \mu_e, \Sigma_e)} dx \\ &= \frac{1}{2}\left\{ \mathrm{tr}(\Sigma_r^{-1}\Sigma_e) + (\mu_r - \mu_e)^T \Sigma_r^{-1} (\mu_r - \mu_e)^T - \log \frac{\det(\Sigma_e)}{\det(\Sigma_r)} - k_e \right\}, \end{aligned} \tag{3.6}$$

where $P_e \sim \mathcal{N}(\mu_h - \mu_t, \Sigma_h + \Sigma_t)$ denotes the transformation from head entity to tail entity of a triplet, with $\mu_h \in \mathbb{R}^{L/2}$ (L is the embedding size) and $\mu_t \in \mathbb{R}^{L/2}$ denoting the mean vectors of head entity and tail entity embeddings; $\Sigma_h \in \mathbb{R}^{L/2 \times L/2}$ and $\Sigma_t \in \mathbb{R}^{L/2 \times L/2}$ denote the covariance matrices of the Gaussian embeddings for head entity and tail entity. The relation embedding in latent space is denoted by $P_r \sim \mathcal{N}(\mu_r, \Sigma_r)$. In (3.6), $\mathrm{tr}(\cdot)$ indicates the trace and Σ_r^{-1} refers to the inverse of the covariance matrix. Similarly, $E_{neg}(h', r', t')$ represents the negative triplet energy function that measures the similarity between entity and relation Gaussian embeddings (i.e., $P_e^{'}$ and $P_r^{'}$) for the corresponding negative triplets. It follows the same definition principles as in (3.5) and (3.6).

3.2.2. Gaussian Embedding for Attributed Graphs. Zhu et al. [62] proposed a deep variational autoencoder (AE)-based graph embedding model (DVNE) to learn Gaussian embeddings using one publication network benchmark (i.e., Cora [28]) and three other social network benchmarks (i.e., Facebook [25], Flickr [27]). The DVNE model [62] is learned by optimizing a hybrid loss function combining a ranking loss (preserving first-order proximity) and a reconstruction loss (preserving second-order proximity). DVNE employs the "second Wasserstein (W2) distance"

$$E(P_i, P_j)^2 = W2(\mathcal{N}(\mu_i, \Sigma_i), \mathcal{N}(\mu_j, \Sigma_j))^2 = ||\mu_i - \mu_j||_2^2 + ||\Sigma_i^{1/2} - \Sigma_j^{1/2}||_F^2 \tag{3.7}$$

to measure the similarity between the learned probabilistic Gaussian distributions for positive node pairs and negative node pairs, where μ, Σ stand for the predicted mean vector and diagonal covariance matrices corresponding to each positive or negative node pair (v_i, v_j). However, the DVNE model is only evaluated for undirected, non-attributed (plain) graph embedding problems.

Aiming to incorporate the important node attribute features for graph Gaussian embedding, Bojchevski and Günnemann [4] proposed an inductive and unsupervised graph Gaussian embedding learning model, named Graph2Gauss (or G2G). It applies the "multihop neighborhood sampling" technique combined with a deep encoder (yet without decoder) to learn probabilistic graph embeddings (i.e., Gaussian distributions) in latent space for *attributed* and *directed/undirected* graphs. Specifically, let us consider a directed/undirected graph $G = (A, X)$ with V and E the corresponding vertex and edge sets, where A denotes the (symmetric or asymmetric) adjacency matrix of size $|V| \times |V|$ and X is the attribute matrix of size $|V| \times D$. The main goal of the G2G model is to project every node from the high-dimensional space into a latent space of multivariate Gaussian distributions with node attributes. For instance, the embedding of node i ($P_i \sim \mathcal{N}(\mu_i, \Sigma_i)$) is an L-dimensional joint normal distribution with a mean vector ($\mu_i \in \mathbb{R}^{L/2}$) and a covariance matrix ($\Sigma_i \in \mathbb{R}^{L/2 \times L/2}$) of diagonal shape, where $L \ll D$. The main architecture of G2G contains four main elements: (1) unsupervised node representation learning based on a deep encoder; (2) node embedding modeling as Gaussian distributions; (3) energy estimation for pairs of nodes in the embedding space; (4) Gaussian embedding learning by minimizing the energy-based loss, i.e., employing the square-exponential loss for the optimization of hyperparameters of the deep encoder. The complete workflow is illustrated in Figure 8, and we present details for the key components of the Gaussian embedding learning process below.

(1) Node triplet generation. Given a node i, its k-hop neighbors can be represented as

$$N_{ik} = \{j \in V | i \neq j, \min(sp(i,j), K) = k\}, \tag{3.8}$$

where $sp(i,j)$ denotes the shortest path between node i and node j (if i and j are not reachable, it returns ∞); K is the maximum considered distance, and usually $K \geq 2$ enables the capturing of high-order proximity. A triplet sample usually consists of anchor, positive, and negative nodes [1]; a set of valid triplets can be represented as

$$D_t = \{(i, j_k, j_l) | sp(i, j_k) < sp(i, j_l)\}, \tag{3.9}$$

with $j_k \in N_{ik}$, $j_l \in N_{il}$, and $k < l$. Thus, for the triplet (i, j_k, j_l), node i is more similar to node j_k than node j_l, and the node pair (i, j_k) denotes one positive pair, while the

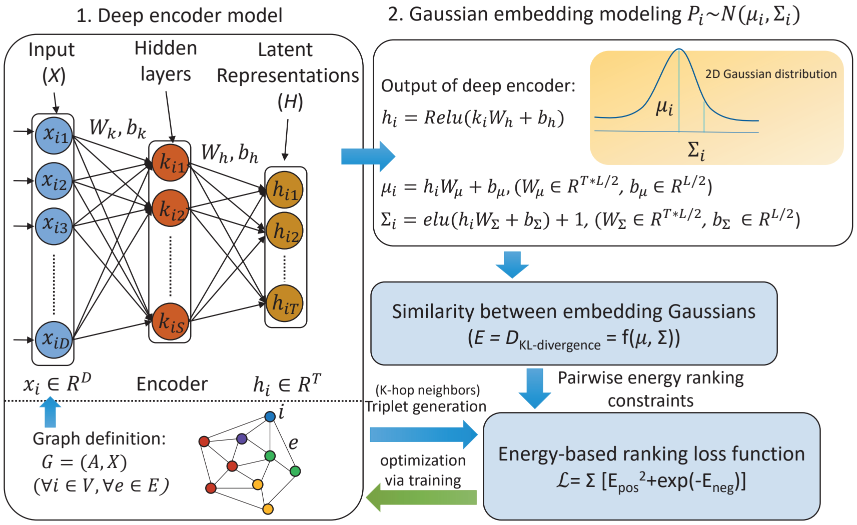

1. Deep encoder model
2. Gaussian embedding modeling $P_i \sim N(\mu_i, \Sigma_i)$
Input (X)
Hidden layers
Latent Representations (H)
W_k, b_k
W_h, b_h
x_{i1}
x_{i2}
x_{i3}
x_{iD}
k_{i1}
k_{i2}
k_{iS}
h_{i1}
h_{i2}
h_{iT}
$x_i \in R^D$
Encoder
$h_i \in R^T$
Graph definition:
$G = (A, X)$
$(\forall i \in V, \forall e \in E)$
i
e
Output of deep encoder:
$h_i = Relu(k_i W_h + b_h)$
2D Gaussian distribution
μ_i
Σ_i
$\mu_i = h_i W_\mu + b_\mu, (W_\mu \in R^{T*L/2}, b_\mu \in R^{L/2})$
$\Sigma_i = elu(h_i W_\Sigma + b_\Sigma) + 1, (W_\Sigma \in R^{T*L/2}, b_\Sigma \in R^{L/2})$
Similarity between embedding Gaussians
($E = D_{\text{KL-divergence}}$ = f(μ, Σ))
Pairwise energy ranking constraints
(K-hop neighbors) Triplet generation
optimization via training
Energy-based ranking loss function
$\mathcal{L}$= Σ [$\text{E}_{\text{pos}}{}^2$+exp(-$\text{E}_{\text{neg}}$)]

Fig. 8 ***Illustration of Gaussian distribution–based graph embedding framework.*** *First, node triplets are generated using the k-hop neighborhood method based on the adjacency matrix (A) of original graph G. Then we input all node pairs and the corresponding attributes to the deep encoder in panel 1 to learn node encodings ($p_\theta(h|x)$) with bottleneck T. Finally, we use an "uncertainty module" to output the mean and variance vectors for the Gaussian embeddings of each node.*

node pair (i, j_l) denotes one negative pair, which are generated for the energy-based ranking loss construction. Moreover, a "node-anchored sampling" strategy [4] provides an effective method for triplet generation that can help reduce the computational complexity in a large graph.

(2) Network structure preservation via personalized ranking. In order to capture the network structure properties at a multiscale level for graph embedding, personalized ranking of energy (similarity) constraints

$$\begin{aligned} &E(P_i, P_{k_1}) < E(P_i, P_{k_2}) < \cdots < E(P_i, P_{k_K}) \\ &\forall k_1 \in N_{i1}, \forall k_2 \in N_{i2}, , \ldots, \forall k_1 \in N_{iK} \end{aligned} \tag{3.10}$$

is imposed on the latent node embeddings. That is, the respective energy (or distance) between embeddings of node i and each node in its k-hop neighborhood ("positive energy") should be lower than that between embeddings of node i and its $(k+1)$-hop neighbors ("negative energy"). Here, $E(P_i, P_j)$ denotes the energy function between learned Gaussian distributions (P_i, P_j) for nodes i and j.

(3) Similarity measure for embedding Gaussians. G2G employs the asymmetric KL-based energy function (same as the use of $KG2E$ in (3.6)) to measure the distance between the learned nodes' Gaussian distributions (P_i, P_j) and score the positive and negative triplets. Here, (i, j) can be either positive node pairs or negative node pairs. Currently, the most commonly used similarity metrics include the symmetric EL, the Jensen–Shannon divergence (JS), the asymmetric KL-divergence (KL), and the pth Wasserstein distance (Wp). Unlike the first two metrics, KL and Wp can also be used to handle *directed graph* embeddings, while at the same time preserving the transitivity of nodes. Obtaining smaller energy (E) means that the nodes' embedding Gaussians are more similar or closer to each other.

(4) Energy-based ranking loss function. Based on the aforementioned constraints on the respective energy between latent embeddings of adjacent k-hop neighbors of each node (e.g., 2-hop vs. 3-hop of each node), in order to learn a graph embedding that satisfies the constraints, the energy-based learning approach [23] is used. The G2G model is trained with an energy-based ranking loss ($\mathcal{L}$) over the sampled triplets (D_t) for penalizing the ranking errors, such that positive energy E_{ij_k} terms are always lower than negative energy terms (E_{ij_l}). The ranking-based loss ($\mathcal{L}$) usually consists of two parts: positive node pair energy (E_{ij_k}) and negative node pair energy (E_{ij_l}). "Margin-based ranking loss" is a frequently used loss function for graph embedding learning; however, the margin has to be manually selected before training. Thus, the "square-exponential loss" representing negative pair energy as an exponential term has better performance in penalizing the ranking error automatically; the specific formula is

$$\mathcal{L} = \sum_{(i,j_k,j_l)\in D_t} (E_{ij_k}^2 + \exp^{-E_{ij_l}}), \quad k < l. \tag{3.11}$$

The uncertainty-aware G2G model can learn a high-dimensional graph in a probabilistic latent space incorporating both graph structure and auxiliary attributes information. Moreover, multiscale graph structure properties can be preserved during graph embedding by performing k-hop neighborhood sampling and energy-based personalized ranking. Additionally, the learned important uncertainty term can be used for multiple aspects: (1) quantification of the neighborhood diversity for every node (i.e., the number of distinct classes in a node's neighborhood), and (2) discovering the intrinsic latent dimensionality of the graph (i.e., $\approx L$); see the example in section 4. In particular, when the dimensions with high average uncertainty are removed, the link prediction performance remains stable and does not drop. More importantly, the G2G model is robust to the number of training edges and the number of labeled nodes, as well as the hyperparameters of the neural networks.

3.3. Dynamic Graph Embedding.

In real applications, many networks exhibit dynamics and evolve over time in terms of growing or shrinking nodes (or links), altering relations or edge weights. Dynamic graphs present a lot of benefits for representing the ubiquitous interactive networks, e.g., evolving gene-protein networks, financial transaction graphs, traffic-flow networks, longitudinal citation networks, and dynamic social communication networks. Recently, studies on dynamic graph embedding have attracted considerable attention due to its capability of capturing the underlying evolving patterns (or embeddings) for each node at different time steps, hence facilitating more accurate forecasts of the future connections between nodes [16].

3.3.1. Defining Dynamic Graphs.

Generally, a dynamic graph ($\mathcal{G}$) can be mathematically defined in two different ways: (1) *Discrete snapshots:* A dynamic graph can be approximated as a sequence of discrete static graph snapshots with equal time intervals ($\mathcal{G} = \{G_1, G_2, \dots, G_T\}$); see the snapshot-based example in Figure 9(A), where $G_t = (V_t, E_t), t \in [1, T]$, denotes the tth snapshot with the node set V_t and the edge set E_t, while T refers to the number of snapshots. Notably, the problem of using the snapshot definition is that it assumes coarse-grained global evolution changes. Hence, the important continuous time-varying graph evolution properties cannot be captured. (2) *Continuous-time graphs:* There are two ways to model a graph as a continuous-time changing graph. (i) *Temporal edge based*: a continuous-time changing graph $G = (V, E_c, \mathcal{T})$ can be denoted by a node set V, a temporal edge set $E_c \subseteq V \times V \times \mathbb{R}^+$, and a time-mapping function $\mathcal{T} : E \to \mathbb{R}^+$ ($\mathbb{R}^+$ is the time domain) for every edge, and every edge is labeled by time (see the example in Figure 9(B)). Hence, the important temporal sequence order and flow information in the temporal graphs can be captured in a fine-grained manner. Notably, sampling valid walks from a dynamic graph has to follow the increasing time order, e.g., "$v_1 \to v_6 \to v_3$" is not a valid walk in Figure 9(B) since edge (v_6, v_3) is in the past of edge (v_1, v_6). (ii) *Link stream based*: another alternative continuous-time graph representation is "link stream," and Figure 9(C) shows the corresponding link stream representation for the dynamic graph in Figure 9(B). The vertical axis represents all the nodes of the dynamic graph and the horizontal axis shows different time instants, while the red arrows correspond to the link streams at different time points. As such, the continuous-time dynamics can be fully represented in a single graph. Additionally, continuous-time dynamic graph representations (temporal edges and link streams) facilitate sampling temporally valid walks from dynamic graphs, hence enabling the learning of more meaningful and accurate graph representations.

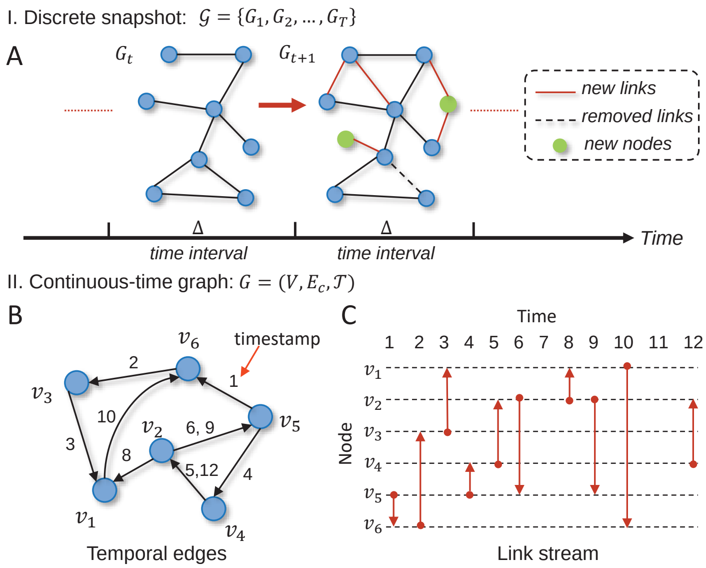


Fig. 9 ***Different representations of a dynamic graph.*** (A) *Modeling a discrete-time dynamic graph $\mathcal{G}$ as a series of discrete snapshots at different time intervals ($\triangle$), i.e., $\mathcal{G} = \{G_1, G_2, \ldots, G_T\}$), which contains T static snapshots; G_{t+1} is the next snapshot of snapshot G_t with five new added links (red solid lines), two new added nodes (green dots), and a removed edge (black dashed line).* (B) *Modeling a continuous-time dynamic graph $\mathcal{G}$ through assigning to each edge multiple timestamps based on an edge-time mapping function ($\mathcal{T}$), i.e., representing a continuous-time dynamic graph as $G = (V, E_c, \mathcal{T})$, where V is the node set and E_c indicates the temporal edge set with each element as a triplet; e.g., $(v_2, v_5, 6), (v_2, v_5, 9)$ represent two temporal edges starting from node v_2 to v_5 at time $t = 6, 9 \in \mathcal{T}(v_2, v_5)$.* (C) *Link stream is another way to represent a continuous-time dynamic graph; the x-axis indicates the time instants while the y-axis represents all the nodes in the dynamic graph.*

3.3.2. Problem Settings for Dynamic Graph Embedding. Based on the snapshot definition above, dynamic graph embedding can be defined as a time series of mappings ($\mathcal{F} = \{f_1, f_2, \ldots, f_T\}$) corresponding to different snapshots [17], where f_T is the embedding of the snapshot (G_T). Moreover, the graph structure properties and temporal dynamics are preserved in the latent space. For continuous-time dynamic networks (CTDNs), the embedding problem can be formulated as the learning of a mapping function $f : V \to \mathbb{R}^L$ such that $L \ll V$, such that every node is embedded into low-dimensional space as an L-dimensional time-dependent vector (see related works in [34]).

There are three key challenges in dynamic graph embedding: (i) *How to learn a stable graph embedding ($\mathcal{F}$)*, such that the mappings are also similar when the adjacent snapshots only exhibit subtle dynamics/changes. Specifically, the stability of $\mathcal{F}$ can be evaluated by a "stability constant" measure

$$\begin{aligned} &\mathcal{K}_s(\mathcal{F}) = \max_{\tau,\tau'} |S_r(F;\tau) - S_r(F;\tau')|, \\ &\text{where} \quad S_r(F;t) = \tfrac{||f_{t+1}(V_t) - f_t(V_t)||_F}{||f_t(V_t)||_F} / \tfrac{||S_{t+1}(V_t) - S_t(V_t)||_F}{||S_t(V_t)||_F}, \end{aligned} \tag{3.12}$$

by computing the maximum "relative stability" difference between different neighboring snapshot pairs [17]. The *relative stability* ($S_r(\mathcal{F};t)$) is denoted by the ratio of the relative difference between embeddings to that of the relative difference between adjacency matrices of two neighboring snapshots. In particular, $f_t(V_t) \in \mathbb{R}^{|V_t|*d}$ denotes the node embedding matrix for all nodes (V_t) in the tth snapshot, where d refers to the embedding size. Similarly, $S_t(V_t), S_{t+1}(V_t)$ denote the adjacency matrices for two neighboring snapshots (see the corresponding definitions in (3.12)). A small stability constant indicates that the embedding is more stable.

(ii) *The second challenge is how to effectively and efficiently update the node embedding* with topological graph evolution over time. (iii) *The third is how to scale it* to large-scale dynamic network embedding.

To tackle these problems, some dynamic graph embedding models have been developed and are listed in Table SM4 in the supplementary material, which contains five different categories: (1) matrix factorization based, (2) SkipGram based, (3) autoencoder (AE) based, (4) graph convolutional networks (GCN) based, and (5) generative adversarial network (GAN) based. We refer interested readers to section SM4 in the supplementary material for references to these methods.

4. Applications.

An increasing number of diverse applications have employed deep learning–based graph embedding methods across different fields due to their inductive and unsupervised nonlinear graph mapping function learning property. High-dimensional and large-scale graphs can be encoded as continuous, low-dimensional graph embeddings in the latent space, while the latent space graph patterns (i.e., embeddings) can be readily used for solving diverse downstream graph analytic problems, such as detecting an anomaly in social networks [44, 13] or financial networks [7], analyzing the nonpharmacological cognitive training effects using functional brain network embedding patterns [57] to predict the early stages of Alzheimer's disease [56], mining biochemical multiscale structures in biochemical graphs [50], and finding functional modules or subcompartments in genomic networks [1]. In what follows, we highlight some representative applications based on four graph types: (1) social networks, (2) citation networks, (3) brain networks, and (4) genomic networks. Moreover, all the public benchmark datasets introduced next can be obtained using the easy-to-use APIs provided in public libraries (e.g., the PyTorch Geometric library [14] and the Deep Graph Library (DGL) package [49]).

4.1. Social Network Applications. With the increasing user-generated content collected from public social media sites and mobile apps, there are a lot of challenging social network mining tasks (e.g., forming conclusions about users, acting upon the information, advertising to users, etc.). Graph embedding techniques can provide a very useful and effective graph representation learning tool for solving these challenging problems. For example, the Zachary karate club network [59] consists of social friendships (links) between 34 members (nodes) of a university karate club. In order to predict the membership for each member, Perozzi, Al-Rfou, and Skiena adopted a deterministic graph node embedding method (called "DeepWalk" [39]) to learn a low-dimensional vector representation for each node, and then the relationships between different nodes could be captured by the distance (i.e., the metrics in Table SM1) between node embedding vectors in the latent space; see the example in Figure 10. This is a relatively simple benchmark for the interested reader to work with so we present implementation details in section SM5.

In addition, studies have also demonstrated the emerging graph embedding methods to be very robust and, in particular, computationally efficient for analyzing large-scale attributed social networks (e.g., Facebook, Github, Friendster, and Twitter network datasets [24]). For example, the authors in [60] leveraged a transfer learning technique in the traditional SkipGram-based graph embedding model for learning node embeddings of attributed large-scale networks; the embeddings were subsequently used as latent features for different machine learning tasks, e.g., multiclass classification, regression, etc.

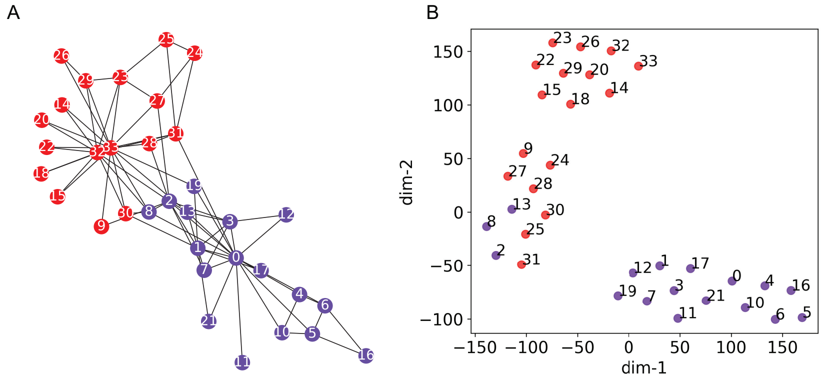


Fig. 10 ***Example of applying graph embedding method for community detection in the karate club dataset*** [59]. (A) *Visualization of the Zachary karate club network* [59]*; this contains* 34 *nodes and* 154 *undirected and unweighted edges with nodes of the same color belonging to the same ground-truth community.* (B) *Scatter plot for the low-dimensional node embedding vectors obtained by the DeepWalk method* [39]*; each node embedding vector with a dimension of* 128 *was projected into* 2*D Euclidean space via the t-SNE method* [20]. *For the interested reader, the detailed implementation is described in subsection* SM5.1 *in the supplementary material.*

4.2. Citation Network Applications. Given the vast number of publications on multidisciplinary fields, a citation network can be used as an effective way to represent the complex citation relationships ("directed links") among the publications ("nodes") from different scientific research domains. In addition, citation network nodes are usually assigned with attributes (e.g., publication date, paper version, etc.), which provide useful auxiliary node features for better characterizing the heterogeneous relationships among network nodes. Advanced graph embedding methods enable us to solve diverse citation network analytics problems more quantitatively and efficiently.

For example, in [4], the authors adopted a generative network-based model to encode every node of the directed and attributed citation networks into a multivariate Gaussian distribution with mean and variance vectors, which can effectively capture uncertainty information. The advantages of quantifying uncertainty using the variance output was demonstrated in [4] by analyzing two subset datasets (CORA and CORA-ML) extracted from the original "Cora" citation network dataset [28]. In addition to the physical interpretation associated with possible conflicts in link prediction or node classification, uncertainty quantification (UQ) can also be used to estimate neighborhood diversity and most importantly to infer the intrinsic latent dimensionality of a graph. In particular, more homogeneous nodes have small variance compared to more heterogeneous nodes whose adjacent neighbors may be members of multiple classes, leading to a more uncertain embedding. On the issue of dimensionality, we show in Figure 11 an important result from [4] for the Coral-ML dataset. The plot gives two sets of lines (red and blue, 64 in total) with each line representing the average variance for a given dimension l over all nodes as a function of the epoch number for a link prediction validation test. It is interesting to see that the lines split clearly into two distinct sets fairly quickly as training continues: a small subset of lines with low uncertainty that asymptotes a small variance constant value, and another one with increasing variance, which obviously denotes the unstable dimensions. The authors of [4] also pointed out that the effective dimensionality ($L = 6$) observed using this approach is very close to the number of ground-truth communities (7); this is also something we observed in our own work in the two neuroscience tasks that we discuss below.

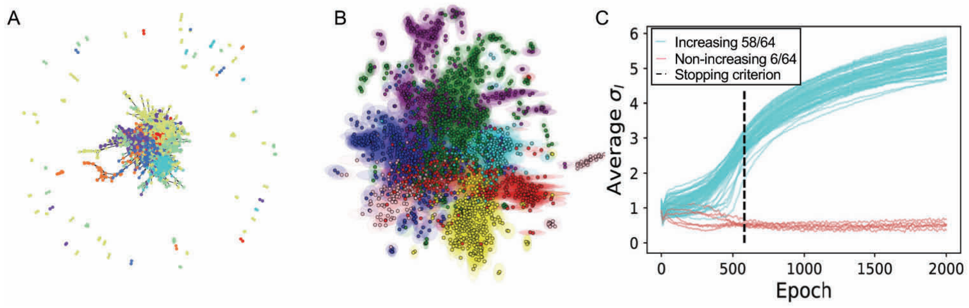


Fig. 11 ***Example of applying stochastic graph embedding method for node classification in the CORA-ML dataset*** [4]. (A) *Visualization of the CORA-ML network* [4]*; this consists of* 2995 *nodes, where each node has* 2879 *attributes and all nodes are divided into* 7 *classes, which are plotted using different colors.* (B) *Scatter plot for the stochastic graph embedding results generated by the G2G method* [4]. *The oval shapes around the points denote the uncertainty.* (C) *Average uncertainty per dimension versus the number of epochs during training. The embedding size is $L = 2 + 2$; implementation details can be found in subsection* SM5.2 *in the supplementary material.*

4.3. Brain Network Applications. The brain system consists of spatially distributed but functionally linked specialized brain elements (neurons, neural assemblies, or brain regions), i.e., "nodes," which continuously share information with each other and form a complex "brain network" (or connectome). Graph theory–based brain network analysis has received extensive attention in the past few decades, but few studies have focused on latent node representation learning for brain networks. Moreover, little attention has been paid to conducting nodewise quantitative comparisons among different brain networks [30]. Graph embedding methods can provide a powerful network representation learning tool and can effectively tackle the aforementioned key issues with hierarchical and quantitative brain network analysis. In particular, graph embedding learning is applicable to multimodality brain network analysis and offers great potential in characterizing both latent link-level and node-level features in the brain networks, which can be further used for identifying vital multiscale network markers for a wide variety of brain disorders.

For example, Rosenthal et al. [41] proposed a predictive model that employed the node2vec [18] method to learn vector-based embeddings for cortical regions in the brain networks, which were then used for link inference between brain network structure and function. Their experimental results demonstrated an important advantage of latent brain network embedding, i.e., that it can simulate the effects of localized network lesions on the global pattern of functional connectivity.

In another application, in order to analyze the subtle multidomain cognitive intervention effects for amnestic mild cognitive impairment (aMCI) patients using fMRI data, the authors of [58] presented a functional brain network analysis method based on the multigraph unsupervised Gaussian embedding method (MG2G), which applied graph weights to sample node neighbors. The model can encode network nodes as multivariate Gaussian distributions ("embeddings"), and it employs the Wasserstein distance for quantitative evaluation of brain network changes (at the region level, system level, and subject level) after three months of multidomain cognitive training (MDCT), i.e., a nonpharmacological intervention. The embedding variances provide extra uncertainty quantification for the latent node embeddings, which lead to better patient-specific evaluation for cognitive training effects as well as the extra benefit of identifying the intrinsic dimensionality of the brain network, which is approximately equal to the number of communities (14) in the brain. These advantages are apparently superior to the deterministic embeddings applied in the previous study [41]. Figure 12 presents quantitative results for evaluating the MDCT effects for aMCI patients. In Figure 12(A), different colors of the "dots" (i.e., 264 brain regions in the fMRI brain network) correspond to the computed Wasserstein-2 (W2) distance values, which measure the learned low-dimensional stochastic resting state fMRI brain network embedding distance before and after cognitive intervention in the latent probabilistic space. "Links" represent the sampled brain connectivity among the detected top 50 affected brain regions (mostly concentrated in the *default mode,* visual, and *memory retrieval* functional brain systems). In Figure 12(B) we compare a stochastic (MG2G) and a deterministic (node2vec) graph embedding method; the plot shows the top 15 most affected brain regions for all patients measured by the W2 distance, identified with the brain systems they belong to. Figure 12(C) shows a violin plot of the W2-distance distributions and probability densities of all 264 regions for all 12 different patients included in this aMCI study [58].

Beyond the aforementioned cognitive training effect assessment application, using graph embedding for brain network representation learning can also provide a promising tool for characterization of Alzheimer's disease (AD) progression [56]. Specifically, the obtained brain network embeddings can be used as latent features to feed into classic classifiers (e.g., random forest, SVM, SVD, etc.) for identifying different early stages in AD progression using magnetoencephalography (MEG) brain networks. The

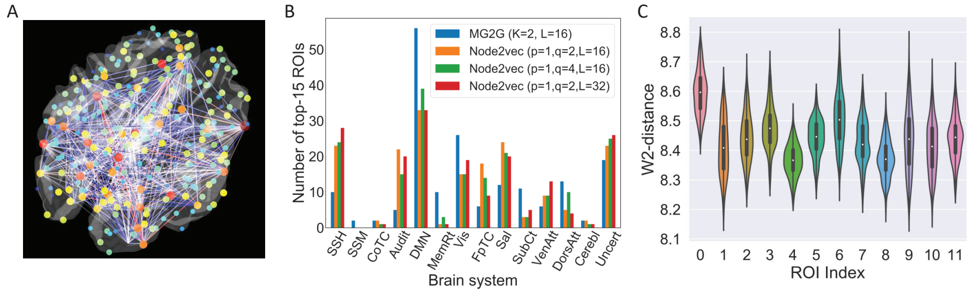


Fig. 12 ***Example of applying stochastic (MG2G) and deterministic (node2vec) graph embedding methods for cognitive training effects using fMRI brain networks.*** (A) *Visualization of the top* 50 *affected brain regions with the corresponding brain connectivity due to MDCT detected by fMRI brain network embedding for one of the aMCI patients.* (B) *and* (C) *are from* [58]. (B) *illustrates the quantification of functional- and system-level changes for* 12 *patients before and after MDCT intervention based on MG2G (blue) and* node2vec [18] *(yellow, green, and red correspond to different node2vec parameters);* (C) *MG2G result: Violin plot for the W2-distance distributions and probability densities of all* 264 *regions with respect to different patients (embedding size $L = 16$).*

learned highly informative latent brain network embeddings can also be effectively used for quantitatively identifying specific brain regions with network alterations related to mild cognitive impairment (MCI).

4.4. Genomic Network Applications. The dimension reduction property of graph embedding techniques, in addition to computational expediency, can also tackle the noise and incomplete problems that exist in large molecular network datasets. Here, we present two different applications, in which two types of graph embedding techniques (deterministic and stochastic) are used for genomic network analysis.

One of the fundamental open problems in computational biology is how the genome is organized in the nucleus of a cell in 3D hierarchical subcompartment, as these subregions exhibit specific epigenomic signatures. Chromatin plays a key role in defining this spatial organization, and hence studying chromatin interactions (Hi-C) can shed light on this fundamental question. The authors of [1] employ the LINE graph embedding method [45] and unsupervised learning to predict the subcompartments in the nucleus based on data describing Hi-C chromatin interactions obtained by a profiling technique combining massively parallel sequencing and proximity-based ligation. The workflow consists of multiple steps and begins by preprocessing the Hi-C interaction data to output a normalized intercromosome matrix, based on which the interaction graph is constructed. A schematic of this workflow is shown in Figure 13(A). Areas (genomic bins) with the same chromosome are represented by the same color nodes. Based on the constructed interaction graph, the graph embedding algorithm projects the graph into a lower-dimensional space for k-means clustering to predict the various hierarchical subcompartments. The method proposed in [1] based on LINE embedding outperforms the hidden Markov model (HMM) for subcompartment prediction, and in fact it predicts a larger number (9 versus 5) of subcompartments. It also outperforms two other graph embedding methods that we have described in this paper, namely, HOPE [38] and DeepWalk [39], as shown in Figure 13(B)–(D). Genomic regions that map to the same subcompartment based on the graph embedding method are spatially close to each other. In the study of [1] this

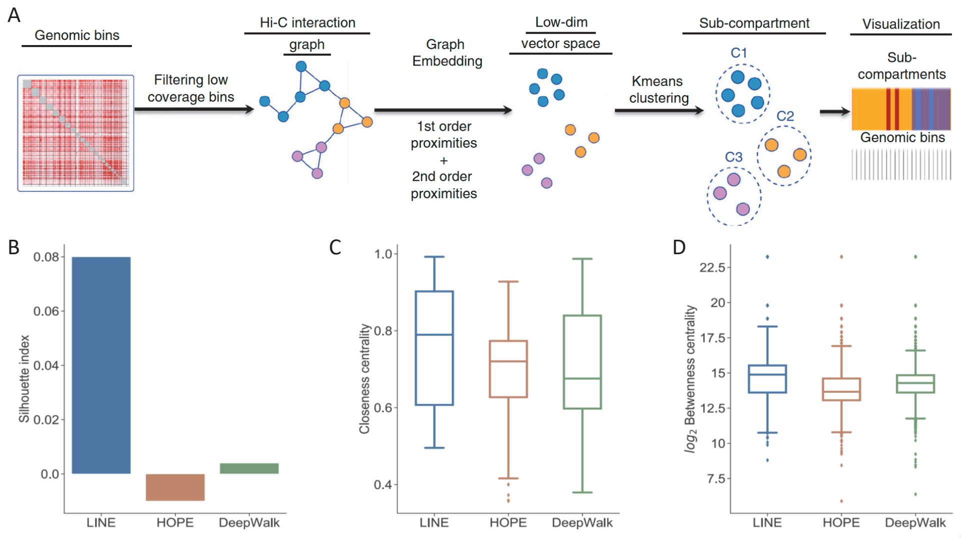


Fig. 13 ***Example of applying the deterministic LINE graph embedding method for sub-compartment identification using Hi-C chromatin interaction data.*** (A) *Workflow from left to right proposed by* [1], *showing a normalized Hi-C interchromosome matrix used to construct the interaction graph. The nodes with the same color denote genomic bins from the same chromosome. LINE projects the graph into a lower-dimensional space for a subsequent k-means clustering step. Comparative performance of LINE against two other graph embedding methods, HOPE and DeepWalk. The plots show the Silhouette index* (B), *the closeness centrality* (C), *and the betweenness centrality* (D). *The line in each box represents the median of the samples while the other lines show their* 75*th and* 25*th percentiles. Both figures are adopted from* [1].

is measured quantitatively using network topology features, including the network centrality (closeness and betweenness) as well as the Silhouette indices, which reflect the consistency within each data cluster. The Silhouette index ranges from -1 to 1, with a higher value indicating better clustering performance. The LINE method [45] improves the Silhouette index compared to HMM and is also superior to HOPE and DeepWalk with respect to all three metrics, as shown in Figure 13(B)–(D).

In the second application, we describe a novel use of Gaussian embedding for gene set member identification. Gene sets give rise to big data given the recent advances in high-throughput biological data; however, processing such data to gain biological insights has been hindered by the lack of appropriate methods to make it useful for downstream tasks. The authors of [50] employed Gaussian embedding to formulate a new method (called “set2Gauss”) to encode each gene set as a multivariate Gaussian distribution in the latent space. Specifically, set2Gauss requires as an input a biological network (e.g., a protein-protein network) as well as a group of gene sets. Each gene is represented as a single point while each gene set is modeled via a multivariate Gaussian distribution. The projection from the graph to the low-dimensional vector space is based on the proximity of the genes in the protein-protein interaction network. Genes in the same set may have different functions, but such differences have been ignored in standard average embedding methods. This functional diversity in each gene set is naturally modeled in set2Gauss by the uncertainty of each dimension in the projected vector. In [50], Wang, Flynn, and Altman demonstrated for three different large-scale gene set groups that set2Gauss can capture differences between gene sets that standard approaches (e.g., mean pooling) would ignore. Overall, the detailed analysis of [50] demonstrates that by using the graph Gaussian embedding approach for projecting the gene sets into latent space as multivariate Gaussian distributions, it can effectively model the *uncertainty* and capture the functional diversity within a gene set, i.e., inherently capture the properties of redundancy, size, and coverage of gene sets.

5. Summary and Discussion. Graph analytics has become an indispensable tool in research as well as in the industrial sector in recent years as it can deal effectively with large and noisy datasets across different application domains. Industrial systems may employ graphs with over 50 million nodes and over a billion edges; hence it is very important to employ and further develop low computational complexity algorithms. Graph embedding is an effective approach for transforming graphs into low-dimensional vector representations for individual vertices in networks. In this review, we have provided introductory material for the mathematical formulation of the graph embedding problem, including preliminaries (i.e., basic graph notations and definitions) as well as specific graph embedding settings for deterministic and stochastic graph embedding problems.

An important high-level task in graph analytics is to transform a high-dimensional original graph to a more compatible representation for efficient downstream processing tasks, e.g., node classification, link prediction, community detection, etc. A key milestone toward this goal in the modern era was the seminal paper on Isomap [46], a nonlinear dimensionality reduction method, which introduced the geodesic distance instead of the Euclidean distance used in classical principal component analysis (PCA). Isomap can be applied to a wide range of datasets, but in the last twenty years since the first publication of Isomap there have been great developments that have reduced the computational complexity and have increased the accuracy in diverse network applications, as demonstrated in section 4. For example, the computational complexity of Isomap is $\mathrm{O}\big(L^2M\big)$ for a graph with N nodes and M $(M \geq N)$ edges with embedding size L. In contrast, the stochastic G2G method we highlighted in the previous section has linear computational complexity, i.e., $\mathrm{O}\big(N\big)$. Hence, one can work with very large graphs with $N = 20{,}000$ nodes as in the CORA and PubMed datasets at reasonable computational cost on a single GPU. Moreover, in modern graph embedding methods we can quantify uncertainty, which can have useful interpretations as in the example in genetic network analysis in section 4.

Broadly, we can classify the graph embedding methods into three major categories: matrix factorization–based methods, random walk–based methods, and neural network–based methods. Matrix factorization–based methods, e.g., HOPE, cannot easily scale up to large network embeddings; their computational complexity is similar to that of Isomap. Hence, the focus of this review has been on random walk–based methods and neural network–based methods, e.g., LINE, DeepWalk, node2vec, G2G, etc., which are characterized by computational complexity of $\mathrm{O}\big(LM\big)$, $\mathrm{O}\big(LN\big)$, or $\mathrm{O}\big(N\big)$. Moreover, in all these methods only unsupervised learning is required. In the supplementary material, section SM3, we summarize the main characteristics of some of these methods along with their specific computational complexity. We also provide implementation details for both deterministic and stochastic graph embedding methods using the relatively simple datasets of the karate club as well as the CORA-ML (see section 4) so that interested readers can have a head start on their exploration of graph analytics.

In addition to static graph embedding methods, we have also discussed the emerging deep learning–based *dynamic* graph embedding methods, which can be particularly computationally expensive due to the evolution of the network (e.g., describing a biological system). Hence, careful consideration is required to select the proper method from a host of different methods, which we summarize in Table SM4 in the supplementary material. The methods we have covered here embed the networks into the Euclidean space. However, other spaces can also be utilized, for example, the hyperbolic space [54, 11, 29]. Hyperbolic n-space is an n-dimensional Riemannian manifold with a constant negative sectional curvature. Hyperbolic embedding methods can be effective for hierarchical data, e.g., complex symbolic datasets [35], and have shown great promise for high-fidelity and parsimonious representations. An example of using hyperbolic embedding for visualization of high-dimensional data is presented in [15]. Another example is presented in [35], where the authors introduced an algorithm to learn the embeddings based on Riemannian optimization and demonstrated that hyperbolic embeddings outperform Euclidean embeddings on data with latent hierarchies.

Finally, as application highlights we have presented results from four different domains. In the first, we considered a social network represented by the karate club benchmark for which we employed the DeepWalk for community detection. In the second, we considered a directed and attributed citation network for the CORA-ML dataset, and we employed the stochastic embedding method G2G for node classification. In the third application, we analyzed multimodality functional brain networks (fMRI and MEG) using MG2G, an extension of G2G, for cognitive training effect evaluation and for predicting AD progression. Finally, in the last example we presented two different applications of deterministic (LINE method) and stochastic (set2Gauss) techniques for genomic networks. Taken together, the results presented in section 4 demonstrate the effectiveness of the graph embedding methods in transforming high-dimensional heterogeneous networks into low-dimensional compact vectors or probability density functions that can lead to great efficiencies in processing downstream tasks depending on the particular application. In addition, using a distributed parallel architecture for training embeddings on very large-scale graphs, graph embedding at scale can be achieved for industrial systems, as demonstrated in [8] for the embedding algorithm SkipGram on the massive open-source Friendster graph [24] with 65 million nodes and 1.8 billion edges. Similarly, for the same dataset, good scalability is achieved with the GraphVite visualization system [63] that implements various models, including DeepWalk, node2vec, and LINE, for node embedding.

REFERENCES

[1] H. Ashoor, X. Chen, W. Rosikiewicz, J. Wang, A. Cheng, P. Wang, Y. Ruan, and S. Li, *Graph embedding and unsupervised learning predict genomic sub-compartments from HiC chromatin interaction data*, Nat. Comm., 11 (2020), pp. 1–11. (Cited on pp. 843, 847, 848)

[2] M. Belkin and P. Niyogi, *Laplacian eigenmaps and spectral techniques for embedding and clustering*, in Proc. of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic, NeurIPS, MIT Press, 2002, pp. 585–591. (Cited on pp. 827, 831, 832)

[3] G. Bertasius, L. Torresani, S. X. Yu, and J. Shi, *Convolutional random walk networks for semantic image segmentation*, in Proc. of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2017, pp. 6137–6145. (Cited on p. 831)

[4] A. Bojchevski and S. Günnemann, *Deep Gaussian embedding of graphs: Unsupervised inductive learning via ranking*, in Proc. of the 6th International Conference on Learning Representations (ICLR), 2018. (Cited on pp. 831, 836, 839, 840, 845)

[5] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, *Freebase: A collaboratively created graph database for structuring human knowledge*, in Proc. of the ACM SIGMOD International Conference on Management of Data, 2008, pp. 1247–1250. (Cited on p. 837)

[6] S. Brin and L. Page, *The anatomy of a large-scale hypertextual Web search engine*, in Proc. of the 7th International World-Wide Web Conference, 1998. (Cited on p. 831)

[7] C. B. Bruss, A. Khazane, J. Rider, R. Serpe, A. Gogoglou, and K. E. Hines, *DeepTrax: Embedding Graphs of Financial Transactions*, preprint, https://arxiv.org/abs/1907.07225, 2019. (Cited on p. 843)

[8] C. B. Bruss, A. Khazane, J. Rider, R. Serpe, S. Nagrecha, and K. E. Hines, *Graph embeddings at scale*, in Proc. of the 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2019, pp. 1–9. (Cited on p. 850)

[9] H. Cai, V. W. Zheng, and K. C.-C. Chang, *A comprehensive survey of graph embedding: Problems, techniques, and applications*, IEEE Trans. Knowl. Data Eng., 30 (2018), pp. 1616–1637. (Cited on pp. 827, 830)

[10] S. Cao, W. Lu, and Q. Xu, *GraRep: Learning graph representations with global structural information*, in Proc. of the 24th ACM International Conference on Information and Knowledge Management, 2015, pp. 891–900. (Cited on pp. 827, 833)

[11] I. Chami, A. Wolf, D.-C. Juan, F. Sala, S. Ravi, and C. Ré, *Low-dimensional hyperbolic knowledge graph embeddings*, in Proc. of the 58th Annual Meeting of the Association for Computational Linguistics, 2020, pp. 6901–6914. (Cited on p. 850)

[12] P. Cui, X. Wang, J. Pei, and W. Zhu, *A survey on network embedding*, IEEE Trans. Knowl. Data Eng., 31 (2018), pp. 833–852. (Cited on p. 827)

[13] K. Ding, J. Li, R. Bhanushali, and H. Liu, *Deep anomaly detection on attributed networks*, in Proc. of the 2019 SIAM International Conference on Data Mining (SDM), SIAM, 2019, pp. 594–602, https://doi.org/10.1137/1.9781611975673.67. (Cited on p. 843)

[14] M. Fey and J. E. Lenssen, *Fast graph representation learning with PyTorch Geometric*, in ICLR Workshop on Representation Learning on Graphs and Manifolds, 2019. (Cited on p. 843)

[15] D. Filonik, T. Feng, K. Sun, R. Nock, A. Collins, and T. Bednarz, *Non-Euclidean embeddings for graph analytics and visualisation*, in SIGGRAPH Asia 2019, 2019, art. 47. (Cited on p. 850)

[16] P. Goyal, S. R. Chhetri, and A. Canedo, *dyngraph2vec: Capturing network dynamics using dynamic graph representation learning*, Knowledge-Based Syst., 187 (2020), art. 104816. (Cited on p. 841)

[17] P. Goyal, N. Kamra, X. He, and Y. Liu, *DynGEM: Deep embedding method for dynamic graphs*, in the 3rd International Workshop on Representation Learning for Graphs (ReLiG), IJCAI, 2017. (Cited on p. 843)

[18] A. Grover and J. Leskovec, *node2vec: Scalable feature learning for networks*, in Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 855–864. (Cited on pp. 831, 832, 833, 834, 835, 836, 846, 847)

[19] S. He, K. Liu, G. Ji, and J. Zhao, *Learning to represent knowledge graphs with Gaussian embedding*, in Proc. of the 24th ACM Internationaln Conference on Information and Knowledge Management, 2015, pp. 623–632. (Cited on pp. 836, 837)

[20] G. E. Hinton and S. T. Roweis, *Stochastic neighbor embedding*, in Proc. of the 15th International Conference on Neural Information Processing Systems, 2003, pp. 857–864. (Cited on pp. 832, 836, 844)

[21] S. Kashyap, S. Kumar, V. Agarwal, D. P. Misra, S. R. Phadke, and A. Kapoor, *Protein-protein interaction network analysis of differentially expressed genes to understand involved biological processes in coronary artery disease and its different severity*, Gene Rep., 12 (2018), pp. 50–60. (Cited on p. 826)

[22] J. Kurose, *On computing per-session performance bounds in high-speed multi-hop computer networks*, ACM SIGMETRICS Perform. Eval. Rev., 20 (1992), pp. 128–139. (Cited on p. 831)

[23] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang, *A tutorial on energy-based learning*, in Predicting Structured Data, MIT Press, 2006. (Cited on pp. 837, 841)

[24] J. Leskovec and A. Krevl, *SNAP Datasets: Stanford Large Network Dataset Collection*, http://snap.stanford.edu/data, 2019. (Cited on pp. 844, 850)

[25] J. Leskovec and J. J. Mcauley, *Learning to discover social circles in ego networks*, in Proc. of the 25th International Conference on Neural Information Processing Systems, 2012, pp. 539–547. (Cited on p. 839)

[26] K. Liu, X. Sun, L. Jia, J. Ma, H. Xing, J. Wu, H. Gao, Y. Sun, F. Boulnois, and J. Fan, *Chemi-Net: A molecular graph convolutional network for accurate drug property prediction*, Int. J. Mol. Sci., 20 (2019), art. 3389. (Cited on p. 826)

[27] J. McAuley and J. Leskovec, *Image labeling on a network: Using social-network metadata for image classification*, in European Conference on Computer Vision (ECCV 2012), Springer, 2012, pp. 828–841. (Cited on p. 839)

[28] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, *Automating the construction of internet portals with machine learning*, Inf. Retr., 3 (2000), pp. 127–163. (Cited on pp. 839, 845)

[29] D. McDonald and S. He, *Heat: Hyperbolic embedding of attributed networks*, in Intelligent Data Engineering and Automated Learning (IDEAL 2020), Springer, 2020, pp. 28–40. (Cited on p. 850)

[30] A. Mheich, F. Wendling, and M. Hassan, *Brain network similarity: Methods and applications*, Netw. Neurosci., 4 (2020), pp. 507–527. (Cited on p. 846)

[31] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient Estimation of Word Representations in Vector Space*, preprint, https://arxiv.org/abs/1301.3781, 2013. (Cited on pp. 832, 833, 835)

[32] G. A. Miller, *WordNet: An Electronic Lexical Database*, MIT Press, 1998. (Cited on p. 837)

[33] P. Nerurkar, M. Chandane, and S. Bhirud, *Survey of network embedding techniques for social networks*, Turkish J. Elec. Eng. Comput. Sci., 27 (2019), pp. 4768–4782. (Cited on p. 832)

[34] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim, *Dynamic network embeddings: From random walks to temporal random walks*, in the 2018 IEEE International Conference on Big Data, IEEE, 2018, pp. 1085–1092. (Cited on p. 843)

[35] M. Nickel and D. Kiela, *Poincaré embeddings for learning hierarchical representations*, in Proc. of the 31st Conference on Neural Information Processing Systems, 2017, pp. 6338–6347. (Cited on p. 850)

[36] A. N. Nikolakopoulos and G. Karypis, *RecWalk: Nearly uncoupled random walks for top-n recommendation*, in Proc. of the 12th ACM International Conference on Web Search and Data Mining, 2019, pp. 150–158. (Cited on p. 831)

[37] M. Okuda, S. Satoh, Y. Sato, and Y. Kidawara, *Community detection using restrained random-walk similarity*, IEEE Trans. Pattern Anal. Mach. Intell., 43 (2021), pp. 89–103. (Cited on p. 831)

[38] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, *Asymmetric transitivity preserving graph embedding*, in Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1105–1114. (Cited on pp. 827, 831, 832, 833, 847)

[39] B. Perozzi, R. Al-Rfou, and S. Skiena, *DeepWalk: Online learning of social representations*, in Proc. of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2014, pp. 701–710. (Cited on pp. 831, 832, 833, 834, 836, 844, 847)

[40] M. Ringnér, *What is principal component analysis?*, Nat. Biotechnol., 26 (2008), pp. 303–304. (Cited on p. 836)

[41] G. Rosenthal, F. Váša, A. Griffa, P. Hagmann, E. Amico, J. Goñi, G. Avidan, and O. Sporns, *Mapping higher-order relations between brain structure and function with embedded vector representations of connectomes*, Nat. Comm., 9 (2018), pp. 1–12. (Cited on pp. 826, 846)

[42] S. T. Roweis and L. K. Saul, *Nonlinear dimensionality reduction by locally linear embedding*, Science, 290 (2000), pp. 2323–2326. (Cited on pp. 827, 831, 832)

[43] C. Su, J. Tong, Y. Zhu, P. Cui, and F. Wang, *Network embedding in biomedical data science*, Brief. Bioinform., 21 (2020), pp. 182–197. (Cited on p. 826)

[44] Q. Tan, N. Liu, and X. Hu, *Deep representation learning for social network analysis*, Front. Big Data, 2 (2019), art. 2. (Cited on p. 843)

[45] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, *LINE: Large-scale information network embedding*, in Proc. of the 24th International Conference on World Wide Web, 2015, pp. 1067–1077. (Cited on pp. 831, 832, 833, 847, 848)

[46] J. B. Tenenbaum, V. De Silva, and J. C. Langford, *A global geometric framework for nonlinear dimensionality reduction*, Science, 290 (2000), pp. 2319–2323. (Cited on p. 849)

[47] L. Vilnis and A. McCallum, *Word representations via Gaussian embedding*, in Proc. of the 3rd International Conference on Learning Representations (ICLR 2015), 2015. (Cited on p. 837)

[48] D. Wang, P. Cui, and W. Zhu, *Structural deep network embedding*, in Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1225–1234. (Cited on p. 833)

[49] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma, et al., *Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs*, preprint, https://arxiv.org/abs/1909.01315, 2019. (Cited on p. 843)

[50] S. Wang, E. R. Flynn, and R. B. Altman, *Gaussian embedding for large-scale gene set analysis*, Nat. Mach. Intell., 2 (2020), pp. 387–395. (Cited on pp. 843, 848, 849)

[51] Y. Wang, C. Feng, L. Chen, H. Yin, C. Guo, and Y. Chu, *User identity linkage across social networks via linked heterogeneous network embedding*, World Wide Web, 22 (2019), pp. 2611–2632. (Cited on p. 826)

[52] Z. Wang, J. Zhang, J. Feng, and Z. Chen, *Knowledge graph embedding by translating on hyperplanes*, in Proc. of the 28th AAAI Conference on Artificial Intelligence, 2014, pp. 1112–1119. (Cited on p. 837)

[53] J. D. West, I. Wesley-Smith, and C. T. Bergstrom, *A recommendation system based on hierarchical clustering of an article-level citation network*, IEEE Trans. Big Data, 2 (2016), pp. 113–123. (Cited on p. 826)

[54] R. C. Wilson, E. R. Hancock, E. Pekalska, and R. P. Duin, *Spherical and hyperbolic embeddings of data*, IEEE Trans. Pattern Anal. Mach. Intell., 36 (2014), pp. 2255–2269. (Cited on p. 850)

[55] M. Xu, D. P. Papageorgiou, S. Z. Abidi, M. Dao, H. Zhao, and G. E. Karniadakis, *A deep convolutional neural network for classification of red blood cells in sickle cell anemia*, PLoS Comput. Biol., 13 (2017), art. e1005746. (Cited on p. 831)

[56] M. Xu, D. L. Sanz, P. Garces, F. Maestu, Q. Li, and D. Pantazis, *A graph Gaussian embedding method for predicting Alzheimer's disease progression with MEG brain networks*, IEEE Trans. Biomed. Eng., 68 (2021), pp. 1579–1588. (Cited on pp. 826, 843, 846)

[57] M. Xu, Z. Wang, H. Zhang, D. Pantazis, H. Wang, and Q. Li, *Gaussian Embedding-Based Functional Brain Connectomic Analysis for Amnestic Mild Cognitive Impairment Patients with Cognitive Training*, preprint, https://doi.org/10.1101/779744, 2019. (Cited on pp. 826, 843)

[58] M. Xu, Z. Wang, H. Zhang, D. Pantazis, H. Wang, and Q. Li, *A new graph Gaussian embedding method for analyzing the effects of cognitive training*, PLoS Comput. Biol., 16 (2020), art. e1008186. (Cited on pp. 846, 847)

[59] W. W. Zachary, *An information flow model for conflict and fission in small groups*, J. Anthropol. Res., 33 (1977), pp. 452–473. (Cited on p. 844)

[60] D. Zhang, J. Yin, X. Zhu, and C. Zhang, *User profile preserving social network embedding*, in Proc. of the 26th International Joint Conference on Artificial Intelligence (IJCAI-17), 2017, pp. 3378–3384. (Cited on p. 844)

[61] Y. Zhou and H. Li, *Asset diversification and systemic risk in the financial system*, J. Econ. Interact. Coord., 14 (2019), pp. 247–272. (Cited on p. 826)

[62] D. Zhu, P. Cui, D. Wang, and W. Zhu, *Deep variational network embedding in Wasserstein space*, in Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2018, pp. 2827–2836. (Cited on pp. 836, 839)

[63] Z. Zhu, S. Xu, J. Tang, and M. Qu, *GraphVite: A high-performance CPU-GPU hybrid system for node embedding*, in WWW '19: The World Wide Web Conference, 2019, pp. 2494–2504. (Cited on p. 850)