

# 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 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 generated nonlinear and highly informative graph embeddings 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 present implementation details and references to open-source software as well as available databases in the Appendix for the interested readers to 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

**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 [42, 61, 60] in brain imaging, molecular networks [25] in drug discovery, protein-protein interaction networks [19] in genetics, social networks [55] in social media, bank-asset networks [67] in finance, publication networks [57] in scientific collaborations, etc. Unlike the regular grid-like Euclidean space data (e.g., images, audio and text), the aforementioned network data are 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, human face recognition, etc.

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, are largely based on extracting handcrafted graph topologi-

---

\*

**Funding:** This work was funded by the NIH grant U01 HL1163232 and a J-Clinic for Machine Learning in Health award at MIT.

<sup>†</sup>McGovern Institute for Brain Research, Massachusetts Institute of Technology, Cambridge, MA AND Division of Applied Mathematics, Brown University, Providence, RI ([mengjia\\_xu1@hotmail.com](mailto:mengjia_xu1@hotmail.com))

cal 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 [46]. Moreover, the hand-engineered features are usually task-specific and cannot get equivalent performance while employing them for different tasks.

Recently, graph embedding techniques have shown remarkable capacity of converting high-dimensional sparse graphs into low-dimensional, dense and continuous vector spaces (see Figure 1), where graph structure properties are maximally preserved [9]. The generated nonlinear and highly-informative graph embeddings (or features) in the latent space can be conveniently used to address different downstream graph analytic tasks (e.g., node classification, link prediction, community detection, visualization, etc.). 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 (e.g., 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.

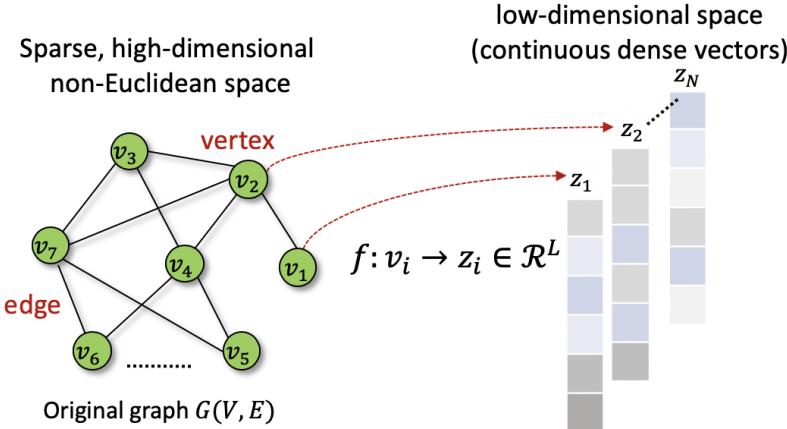


FIG. 1. *Schematic of graph (node) embedding.* For a simple graph  $G(V, E)$  consisting of a node set  $V$  and an edge set  $E$ , using a graph embedding model  $f$ , different nodes (e.g.,  $v_1$  and  $v_2$ ) from the original graph in a high-dimensional irregular domain can be mapped into a latent low-dimensional space as a  $L$ -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 obtained latent variables  $z_i, i \in V$  (i.e., features) 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 [11]: matrix factorization-based methods, random walk-based methods, and neural network-based methods. Matrix factorization-based methods (e.g., [10, 2, 37, 43]) 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. In this paper, we focus on the more scalable random walk-based methods and neural network-based methods. 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. 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 notations, 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  $V = \{v_1, v_2, v_3, \dots, v_n\}$  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  $v_j$ , 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 un-ordered 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 persons 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 an example in [Figure 2\(C\)](#) showing an education network. Different node colors refer to different node types, i.e., brown color 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 an example in [Figure 2\(D\)](#), whereas binary graphs (or unweighted graphs) do not have any weight associated with edges.

According to different graph edge density and types of operations applied on a graph, a graph can be represented in three different ways as: **i) Adjacency matrix ( $A$ ):**  $A$  is a  $|V| \times |V|$  2D square matrix with its element  $\{s_{i,j} | i, j \in (0, |V|)\}$  representing whether two nodes are connected in a graph, and  $|V|$  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,  $w$  is the edge weight), otherwise,  $s_{i,j} = 0$ . Moreover,  $A$  is a symmetric matrix for undirected graphs and asymmetric for the directed graphs, see examples in [Figure 2\(E-H\)](#). Furthermore, the adjacency matrix has a high computational cost for a traversal of every node’s adjacent nodes of the order of  $O(|V|)$ , and therefore, it is not appropriate for representing large sparse graphs due to high space cost  $O(|V|^2)$ . **ii) Adjacency list:** It makes use of a linked list to only store each node’s adjacent nodes (see a detailed example in [Figure 3](#)). Therefore, the adjacency list is convenient for node operations (i.e., insert, delete or add nodes), and the space cost is only  $O(|E|)$ , which benefits effective representations

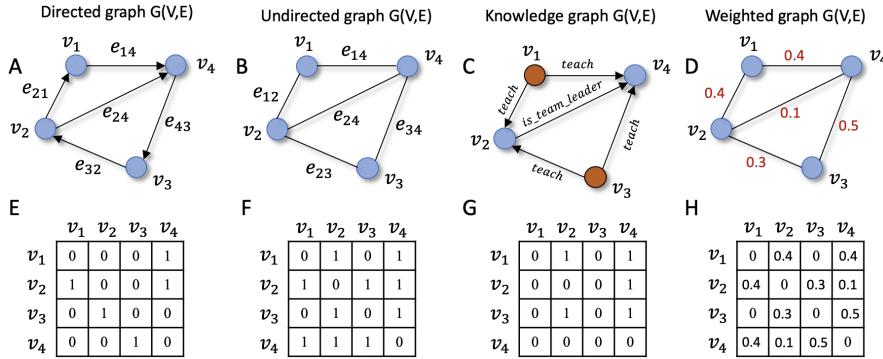


FIG. 2. *Different types of graphs and their corresponding adjacency matrix representations.* The first row from (A) to (D) are, respectively, directed, undirected, knowledge and weighted graph examples. The main difference between (A) and (B) is that edges are directed in (A) but undirected in (B). (C) is a knowledge graph consisting of two different types of nodes (in “brown” and “blue” colors) and two different types of edges (“teach” and “is\_team\_leader”). Graph (C) is an instance of 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  $4 \times 4$  adjacency matrices for graphs (A)-(D).

for large sparse graphs (i.e.,  $|E| \ll |V| \times |V|$ ). **iii) Incidence matrix:** It 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”).

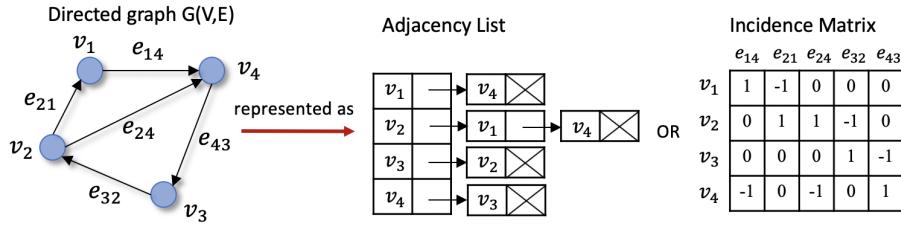


FIG. 3. *Different ways to represent a static graph.* Beyond the adjacency matrix representation as shown in Figure 2 (E-H), a static graph can be also represented as an “adjacency list” or “incidence matrix”.

**2.2. Graph embedding problem setting.** In line with the aforementioned graph notations and definitions, given a graph  $G = (V, E)$ , the task to learn its graph node embeddings (e.g.,  $L$  dimension,  $L \ll |V|$ ) can be mathematically formulated as learning a projection  $\phi$ , 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 ( $\Phi = \{z_i \in \mathbb{R}^L | i = 1, 2, \dots, |V|\}$ ) while another one is stochastic probabilistic distributions ( $\Phi = \{\mathbb{P}_i \sim \mathcal{N}(\mu_i, \Sigma_i) | i = 1, 2, \dots, |V|\}$ ), 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 pre-defined similarity function (see [Table A1](#) in the Appendix and more details in [subsection 2.2.3](#)). The specific graph node embedding formula is shown in [\(2.1\)](#), i.e.,

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

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 are 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 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*: It is used to characterize local pairwise similarity between nodes linked by edges, i.e., 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 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.

$$(2.2) \quad \mathbb{E} = D_{KL}(\hat{p}_1 || p_1) = - \sum_{i,j \in E} w_{ij} \log(p_1(v_i, v_j)),$$

where  $\hat{p}_1(v_i, v_j) = \frac{w_{ij}}{\sum_{s,t \in E} w_{st}}, \ p_1(v_i, v_j) = \frac{\exp(z_i^T z_j)}{\sum_{s,t \in E} \exp(z_s^T z_t)}$ .

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  $v_j$ ;  $w_{ij}$  denotes the weight between two nodes  $i$  and  $j$ .

2) *Second-order proximity*: It 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 condi-

tional probabilities, as follows:

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

Here,  $z'_i$  is the representation of  $v_i$  when it is treated as a specific “context” [48].

3) *High-order proximity*: There are not many works in the literature focusing on high-order proximity quantification so far, as second-order proximity works well in most graph embedding methods. Usually, high-order proximity can be defined based on some 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 effective node context (or neighbors) aiding for optimizing diverse graph embedding models. Similarity between nodes in the original graphs can be characterized based on five different aspects: 1) presence of edge, 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: multi-hop neighborhood-based [4], random walk-based [39, 16, 48], and adjacency matrix-based methods [43, 2, 37]. 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) *Multi-hop 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 [20]. Recently, in order to sample node neighbors in the original graphs and preserve graph structure property, the multi-hop neighborhood (or  $k$ -hop neighborhood) method has become an effective solution to address this problem in graph embedding. Specifically,  $k$ -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 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 G2G 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 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 [35], community detection [36], image segmentation [3, 59]). In particular, the node similarity ( $\text{Sim}(v_i, v_j)$ ) 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 ( $W_{v_i} = r_0, r_1, \dots, r_l$ ) of length  $l$  over the graph from the source node  $r_0 = v_i$ , see a random walk example in Figure 4(B). Particularly, the nodes in the random walk are generated based on the distribution described in (2.4), where  $r_i$  denotes the  $i$ -th node in the random walk starting from  $r_0 = u$ ,  $\pi_{vx}$  denotes un-normalized transition probability between nodes  $v$  and  $x$ , and  $Z$  is a normalizing constant [16].

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

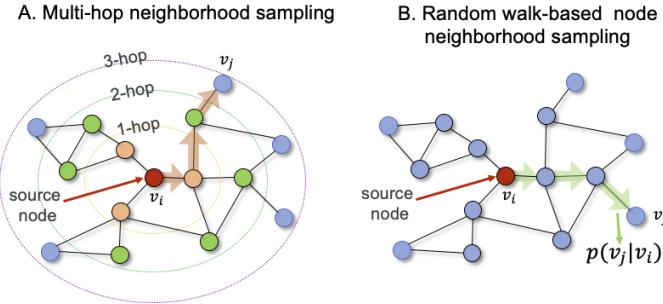


FIG. 4. **Node neighborhood sampling techniques.** A) Multi-hop neighborhood-based node neighbor sampling. The red node ( $v_i$ ) is taken as the source target node, and nodes of different colors represent the sampled nodes at different hops;  $v_j$  is one of the 3-hop neighbors of  $v_i$ . The orange 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).

In the earlier DeepWalk methods [39], a random walk is generated (similar as a sampled “sentence” in nature language processing applications) for each node (comparable as “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 co-occurred on the generated random walk using a language embedding model (i.e., SkipGram model [31]). Moreover, other methods that we discuss in subsection 3.1, such as LINE [48] and node2vec [16], also apply modified random walk strategies to sample similar neighbors (or context) for different nodes in the original graph. Compared with the multi-hop 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 it only considers the node pairs that co-occur 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 [43], Laplacian eigenmaps [2], HOPE [37]) (see section 4) and stochastic neighborhood embedding (SNE) method [18] all employ adjacency matrix-based node similarity measures for graph embedding. They have achieved good performance in graph reconstruction problems. However, using the adjacency matrix-based node similarity measure to carry out graph embedding, the obtained node embeddings usually overfit the adjacency matrix of the original graph. Hence, it does not work well on inconspicuous connection detection (such as downstream link prediction tasks) [33]. Moreover, due to the computational complexity  $O(|V|^2)$ , using adjacency matrix-based measures is 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, Gaussian distribution representations) in the latent space using different graph embedding methods, we can employ different metrics ( $M$ ) to measure the 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 A1 of Appendix A](#) in the Appendix). 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 A1](#) in the Appendix).  $P_i \sim \mathcal{N}(\mu_i, \Sigma_i)$  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  $\mu_i$  and  $\Sigma_i$  denote the learned mean vector and covariance matrix for node  $v_i$ , and  $\mu_j$  and  $\Sigma_j$  are the learned mean vector and covariance matrix for node  $v_j$ .

**3. Overview of graph embedding methods.** According to the aforementioned 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 [37]), b) random walk-based methods (e.g., DeepWalk [39], LINE [48], node2vec [16]), and c) deep learning-based methods (SDNE [52]). 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 to 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 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 random walk-based graph node embedding method for  $G$  is shown in [Figure 5](#). It mainly contains three key stages, which are described in detail, as follows.

**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 node context (similar as “sentence”), and then employed an encoder called “SkipGram” model [31] to learn graph node embeddings based on the sampled similar node pairs in the random 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 only capture the local structure information by using the truncated random walks, but the global structure information is missing. To address this problem, node2vec [16] 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 de-

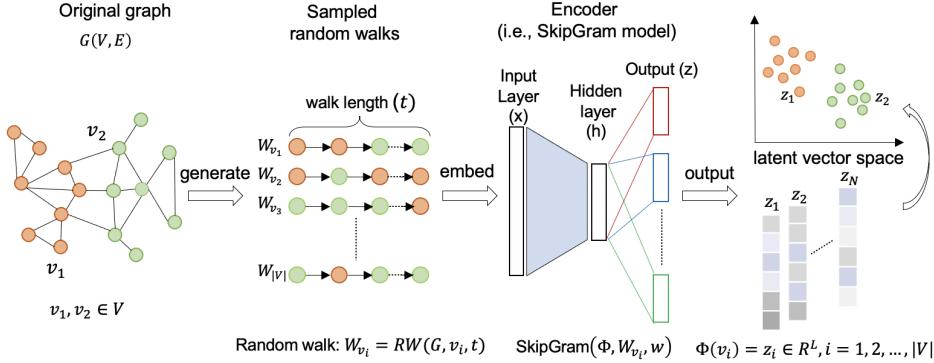


FIG. 5. Pipeline for random walk-based graph embedding methods. In order to learn node embeddings for the original graph  $G = (V, E)$  consisting of two types of nodes marked in different colors, we apply random walks methods to first generate a set of node context ( $W_{v_i}$ ) for every node ( $v_i \in V$ ); the sampled node context (i.e., random walks) are of same fixed walk length  $t$ . Second, based on the generated node contexts, a language embedding model playing 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, community detection, etc.

tailed schematic of node neighbor expansion using the biased random walk approach (or “2nd-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  $\pi_{vx}$  in (2.4) is modified using a search bias ( $\alpha$ ) in conjunction with edge weight  $k_{vx}$  ( $k_{vx} = 1$  if binary graphs) to guide node neighbor searching. Specifically, assume a random walk just traversed nodes  $t, v$ , and resides at  $v$ , then the unnormalized transition probability ( $\pi_{vx}$ ) between the node  $v$  and the next walk node  $x$  can be computed by the formula in (3.1) as follows.

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

where the search bias ( $\alpha_{pq}$ ) 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  $x$ ;  $p$  and  $q$  enables to control how fast the walk explores and leaves the starting node ( $v$ ) (see Figure 6). Particularly, the larger is  $p$  ( $> \max(q, 1)$ ), the less likely is to sample an already visited node, i.e., *moderate exploration*; inversely, when  $p$  ( $< \min(q, 1)$ ), it will lead to backtrack 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 towards a *micro-view* of the node neighborhoods. Whereas,  $q < 1$  is “DFS-like” *macro-view exploration* for the node neighborhoods. Additionally, when  $p = q$ , the node2vec method is the same as DeepWalk.

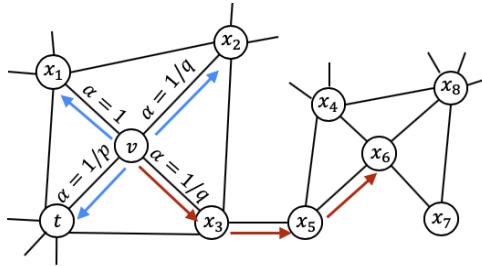


FIG. 6. *Biased random walk schema incorporating both BFS and DFS.* Starting from the source target node ( $v$ ) to generate a random walk with fixed length ( $l = 4$ ) over the graph, blue arrows show the breadth-first search (BFS) directions while red arrows indicate the depth-first search (DFS) directions. The DFS strategy helps expanding the node neighbors deeper and enables exploration of node neighbors (e.g.,  $x_6$ ) that have similar structure roles as the starting node (i.e.,  $v$ ) beyond the directly connected node neighbors (e.g.,  $x_1, x_2, x_3$ ). Hence, 2nd-order proximity can be also preserved. The biased random walk strategy proposed in node2vec [16] achieves a trade-off of BFS and DFS; the explored neighborhood search is guided by using a search bias parameter ( $\alpha$ ), 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.

Moreover, the selection of window size ( $w$ ) and walk length ( $t$ ) also plays an important role in generating 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 walk length  $t$  is used, the more noisy node co-occurrences would be introduced in the graph embedding.

**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, which can be also employed as an effective encoder for prevalent graph node embedding problems by feeding the generated node context from the step 1) and output 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).

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

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 [16] 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 [16] 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 shown in

(3.3) as follows:

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

where  $\sigma$  denotes the sigmoid function;  $n_i$  denotes the random sampled negative node neighbors, which follows a random distribution over all nodes. To learn 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) 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 [18] or PCA [41] 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, node importance) with only simple traditional machine learning classifiers (e.g., SVM, random forest, k-means). Moreover, the low-dimensional vectors can be also further computed in different forms to feed into classifiers, such as concatenation, average or Hadamard product, etc.

“Vector point-based graph embedding” methods (summarized in Table C1 of Appendix C in the Appendix) show great effectiveness for learning graph representation 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 aforementioned deterministic vector point-based graph embedding methods in subsection 3.1, another type of emerging graph embedding methods (e.g., G2G [4], KG2E [17], DVNE [68]) 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 learning of node embeddings as “potential functions” or “continuous densities” in latent space, which leads to two major advantages: a) it enables to effectively incorporate useful *unstructured attribute information* associated with each node, b) every node from the original graph is encoded as low-dimensional *multivariate Gaussian distributions* ( $P_i \sim \mathcal{N}(\mu_i, \Sigma_i)$ ) 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 Gaussian distribution-based graph embedding is shown in (2.1). We summarize some recent works based on graph Gaussian embedding in Table B1 of Appendix B in the Appendix.

The graph Gaussian embedding technique is inspired by the word2Gauss approach [51], 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 to explicitly model the uncertainty, entailment, 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 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, 2) Gaussian embedding for attributed graphs. Generally, there are three main components for learning a graph Gaussian embedding model, including *triplet pair generation*, *energy function* and *loss function*.

**3.2.1. Gaussian embedding for knowledge graphs.** He et al. [17] first proposed a Gaussian embedding model (KG2E) for learning stochastic knowledge graph (KG) representations in terms of Gaussian distributions, enabling modeling of uncertainty of relations and entities in the knowledge graph using two benchmarks (i.e., WordNet [32] and Freebase [5]). Figure 7 presents a density-based knowledge graph Gaussian embedding example for both “entities” (i.e., nodes) and “relations” (i.e., links) in the knowledge graph. 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 [21]. 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 pre-defined margin ( $\gamma$ ).

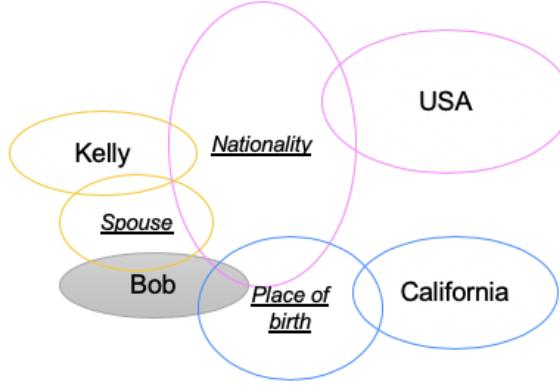


FIG. 7. *Example of density-based knowledge graph Gaussian embedding in latent space.* Four entities (“Bob”, “Kelly”, “USA”, “California”) and three types of relations (“Nationality”, “Spouse”, “Place of birth”) are shown in the diagram. Different colors indicate different facts/triplets, e.g., (Bob, 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”).

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

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 [56]. 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  $E_{pos}(h, r, t)$  shown in (3.6).

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

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

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 embeddings in latent space is denoted by  $P_r \sim \mathcal{N}(\mu_r, \Sigma_r)$ . In (3.6),  $\text{tr}(\cdot)$  indicates the trace and  $\Sigma_r^{-1}$  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 shown in (3.5) and (3.6).

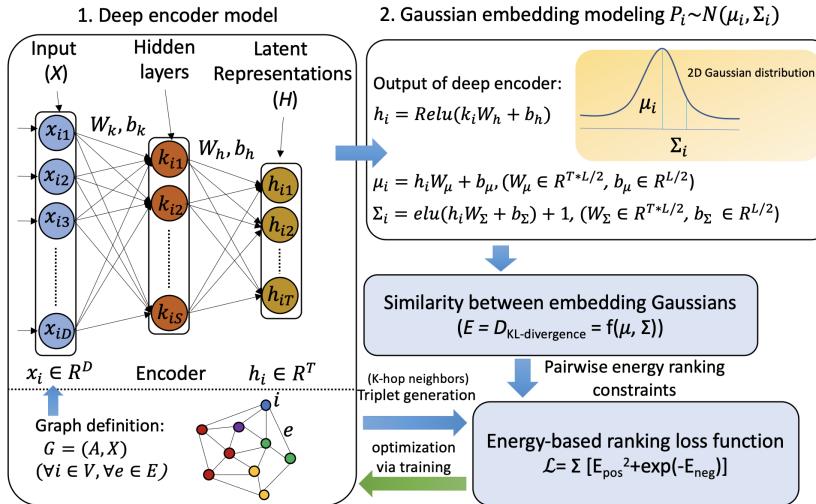
**3.2.2. Gaussian embedding for attributed graphs.** Zhu et al. [68] proposed a deep variational autoencoder (AE)-based graph embedding model (DVNE) to learn Gaussian embeddings using one publication network benchmark (i.e., Cora [29]) and three other social network benchmarks (i.e., Facebook [23], Flickr [28]). The DVNE model [68] is learned by optimizing a hybrid loss function combining a ranking loss (preserving 1st-order proximity) and a reconstruction loss (preserving 2nd-order proximity). DVNE employs the “2nd-Wasserstein (W2) distance” as shown in (3.7)

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

to measure the similarity between the learned probabilistic Gaussian distributions for positive node pairs and negative node pairs, where  $\mu, \Sigma$  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 et al. [4] proposed an inductive and unsupervised graph Gaussian embedding learning model, named Graph2Gauss (or G2G). It applies the “multi-hop 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,  $A$  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 *Graph2Gauss* 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 a  $L$ -dimensional joint normal distribution with a mean vector ( $\mu_i \in \mathbb{R}^{L/2}$ ) and a covariance matrix ( $\Sigma_i \in \mathbb{R}^{L/2 \times L/2}$ ) in diagonal shape, where  $L \ll D$ . The main architecture of *Graph2Gauss* 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 hyper-parameters 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.



**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 deep encoder in 1) to learn node encodings ( $p_\theta(h|x)$ ) with bottleneck of  $T$ . Finally, we use an “uncertainty module” to output the mean and variance vectors for Gaussian embeddings of each node.

**1) Node triplet generation:** Given a node  $i$ , its  $k$ -hop neighbors can be represented as  $N_{ik}$  in (3.8),

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

where  $sp(i, j)$  denotes the shortest path between node  $i$  and node  $j$  (if  $i$  and  $j$  are not reachable it returns  $\infty$ );  $K$  is the maximum considered distance, usually  $K \geq 2$  enables capturing high-order proximity. A triplet sample usually consists of anchor, positive, and negative nodes [1]; a set of valid triplets can be represented as in (3.9),

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

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 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 way 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 multi-scale level for graph embedding, personalized ranking of energy (similarity) constraints in (3.10)

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

are imposed to 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 the one 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:** *Graph2Gauss* employs the asymmetric KL-based energy function (same as *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 expected likelihood (EL), the Jensen-Shannon divergence (JS), the asymmetric KL-divergence (*KL*), and the  $p$ -th Wasserstein distance (*Wp*). Different from the first two metrics, *KL* and *Wp* can be also used to handle *directed graph* embeddings while at the same time preserving the transitivity of nodes. Obtaining smaller energy ( $E$ ) represents 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 [21] is used. The *Graph2Gauss* 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  $(E_{iji})$ . The ranking-based loss ( $\mathcal{L}$ ) usually consists of two parts: positive node pair energy ( $E_{ij_k}$ ) and negative node pair energy ( $E_{iji}$ ). “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 given in (3.11).

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

The uncertainty-aware Graph2Gauss model can learn a high-dimensional graph in a probabilistic latent space incorporating both graph structure and auxiliary attributes information. Moreover, multi-scale 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 an example in [section 4](#). In particular, when the dimensions with high average uncertainty are removed, the link prediction performance remains stable and without a decrease. More importantly, the G2G model is robust to the number of training edges, the number of labeled nodes, as well as the hyper-parameters 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 forecast of the future connections between nodes [[14](#)].

**3.3.1. Definitions of 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  $t$ -th 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 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  $\mathcal{T} : E \rightarrow \mathbb{R}^+$  ( $\mathbb{R}^+$  is the time domain) for every edge, and every edge is labeled by time (see an 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., “ $v_1 \rightarrow v_6 \rightarrow v_3$ ” is not a valid walk in the [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 learning of more meaningful and accurate graph representations.

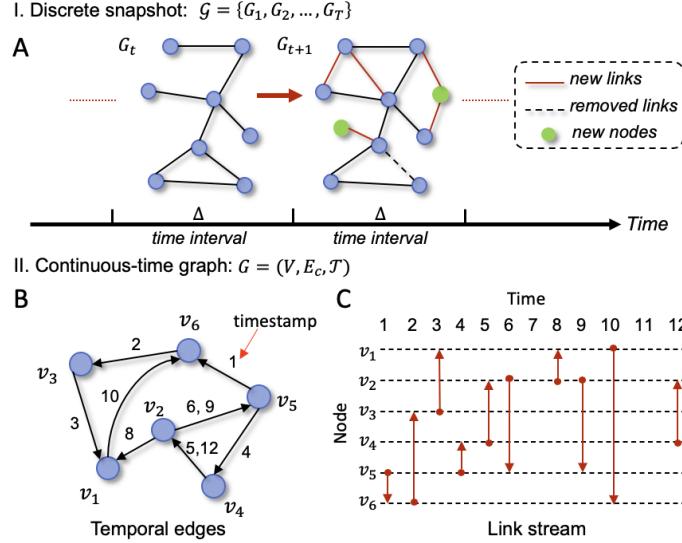


FIG. 9. *Different representations of a dynamic graph.* (A) Modeling a discrete-time dynamic graph  $\mathcal{G}$  as a series of discrete snapshots at different time intervals ( $\Delta$ ), i.e.,  $\mathcal{G} = \{G_1, G_2, \dots, G_T\}$ ), which contains  $T$  static snapshots;  $G_{t+1}$  is the next 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 (CTDN)  $\mathcal{G}$  through assigning each edge with multiple timestamps based on a edge-time mapping function ( $\mathcal{T}$ ), i.e., representing a continuous time dynamic graph as  $\mathcal{G} = (V, E_c, \mathcal{T})$ , where  $V$  is the node set,  $E_c$  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 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 ( $\mathcal{F} = \{f_1, f_2, \dots, f_T\}$ ) corresponding to different snapshots [15], 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 learning of a mapping function  $f : V \rightarrow \mathbb{R}^L$  s.t.  $L \ll V$ , such that every node is embedded into low dimensional space as a  $L$ -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 ( $\mathcal{K}_s(\mathcal{F})$  in (3.12))

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

by computing the maximum “relative stability” difference between different neighboring snapshot pairs [15]. 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,  $f_t(V_t) \in \mathbb{R}^{|V_t| \times d}$  denotes

the node embedding matrix for all nodes ( $V_t$ ) in the  $t - th$  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) *How to effectively and efficiently update the node embedding* with topological graph evolution over time. iii) *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 D1](#) in the Appendix, 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 the interested readers to [Appendix D](#) in the Appendix for references to these methods.

**4. Applications.** An increasing number of diverse applications have employed the aforementioned deep learning-based graph embedding methods across different fields due to their inductive and unsupervised non-linear 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. For example, detecting an anomaly in social networks [47, 12] or financial networks [7], analyzing the non-pharmacological cognitive training effects using functional brain network embedding patterns [61] and to predict early stages of Alzheimer’s disease [60], mining biochemical multi-scale structures in biochemical graphs [54], finding functional modules or sub-compartments in genomic networks [1], etc. In the following, we highlight some representative applications based on four graph types: 1) social networks, 2) citation networks, 3) brain networks, 4) genomic networks. Moreover, all the public benchmark datasets introduced in the following can be obtained using the easy-to-use APIs provided in public libraries (e.g., PyTorch Geometric library [13], Deep Graph Library (DGL) package [53]).

**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’s karate club network [64]) consists of social friendships (links) between 34 members (nodes) of a university karate club. In order to predict the membership for each member, Perozzi et al. 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 can be captured by the distance (i.e., the metrics in [Table A1](#)) between node embedding vectors in the latent space, see example in [Figure 10](#). This is a relatively simple benchmark for the interested reader to practice so we present implementation details in [Appendix E](#).

In addition, studies also demonstrated the emerging graph embedding methods performed very robust and in particular computationally efficient for analyzing large-scale attributed social networks (e.g., Facebook, Github, Friendster and Twitter network datasets [22]). For example, the authors in [65] 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., multi-class classification, regression, etc.

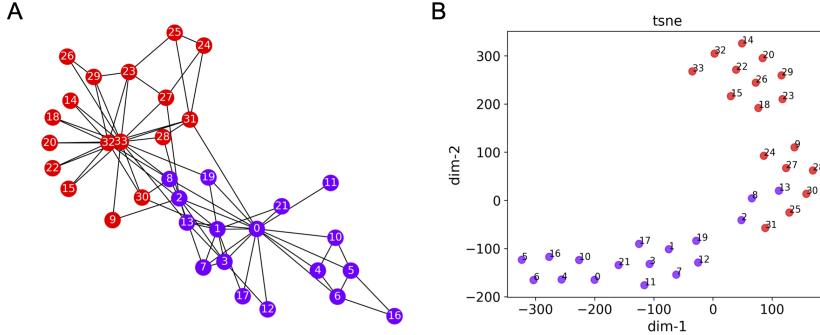


FIG. 10. *Example of applying graph embedding method for community detection in the karate club dataset [64].* (A) Visualization of the Zachary’s karate club network [64]; it contains 34 nodes and 154 undirected and unweighted edges with nodes of the same color belonging to the same ground-truth community. (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 2D euclidean space via the t-SNE method [18]. For the interested readers, the detailed implementation is described in Appendix E.1 in the Appendix.

**4.2. Citation network applications.** Given the vast amount 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 paper [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. Moreover, the advantages of quantifying uncertainty using the variance output has been demonstrated in [4] by analyzing two subset datasets (CORA and CORA-ML) extracted from the original “Cora” citation network dataset [29]. 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 a member 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 describes two sets of lines (red and blue, 64 in total) with each line representing the average variance for a give 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 di-

dimensions. The authors of [4] also observed 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.

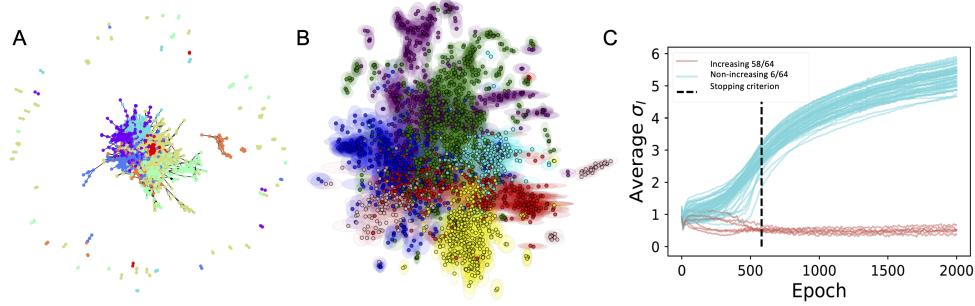


FIG. 11. *Example of applying stochastic graph embedding method for node classification in CORA-ML dataset [4].* (A) Visualization of the CORA-ML network [4]; it consists of 2995 nodes, each node has 2879 attributes, and all nodes are divided into 7 classes, which are plotted using different colors. (B) Scatter plot for the stochastic graph embedding results generated by the graph2gauss 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 Appendix E.2 in the Appendix.

**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, however, few studies have focused on latent node representation learning for brain networks. Moreover, less attention has been paid to conducting node-wise 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 multi-modality brain network analysis and bears great potential in characterizing both latent link-level and node-level features in the brain networks, which can be further used for identifying vital multi-scale network markers for a wide variety of brain disorders.

For example, Rosenthal et al. [42] proposed a predictive model that employed the node2vec [16] 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 the latent brain network embeddings, 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 multi-domain cognitive intervention effects for amnestic mild cognitive impairment (aMCI) patients using fMRI data, the authors of [62] presented a functional brain network analysis method based on the multi-graph 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 employs the Wasserstein

distance for quantitative evaluation of brain network changes (at region-level, system-level and subject-level) after 3-month of multi-domain cognitive training (MDCT), i.e., a non-pharmacological intervention. The embedding variances provide extra uncertainty quantification for the latent node embeddings, which lead to better patient-specific evaluation for the 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 [42]. [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 measures 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 the 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, and 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 [62].

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 [60]. 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 learned high-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, it can also tackle the noise and incomplete problems existing in large molecular network datasets. Here, we present two different applications, where 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 three-dimensional hierarchical sub-compartments as these sub-regions exhibit specific epigenomic signatures. Chromatin plays a key role in defining this spatial organization, and hence studying chromatin interactions (Hi-C) can shed light into this fundamental question. The authors of [1] employ the LINE graph embedding method [48] and unsupervised learning to predict the sub-compartments 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 pre-processing 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-

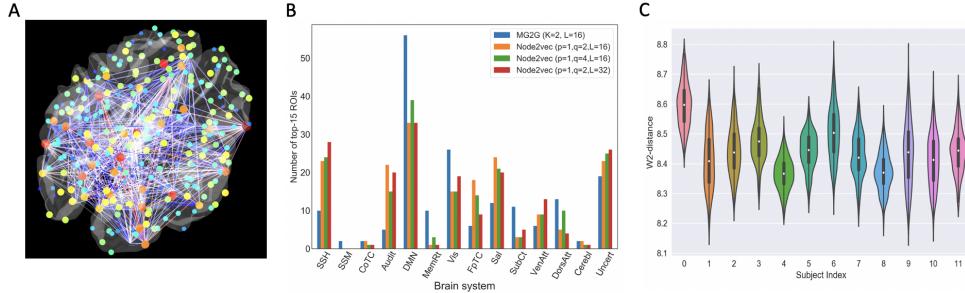


FIG. 12. *Example of applying stochastic (MG2G) and deterministic (node2vec) graph embedding methods for cognitive training effects using fMRI brain networks.* (A) Visualization of the top 50 affected brain regions with the corresponding brain connectivity due to multi-domain cognitive training (MDCT) detected by fMRI brain network embedding for one of the aMCI patients. (B) and (C) are from [62]. (B) illustrates the quantification of functional and system-level changes for 12 patients before and after MDCT intervention based on MG2G (blue) and node2vec [16] (yellow, green, and red correspond to different node2vec parameters); (C) MG2G result: Violin plot for the  $W_2$ -distance distributions and probability densities of all 264 regions w.r.t. different patients (the embedding size  $L = 16$ ).

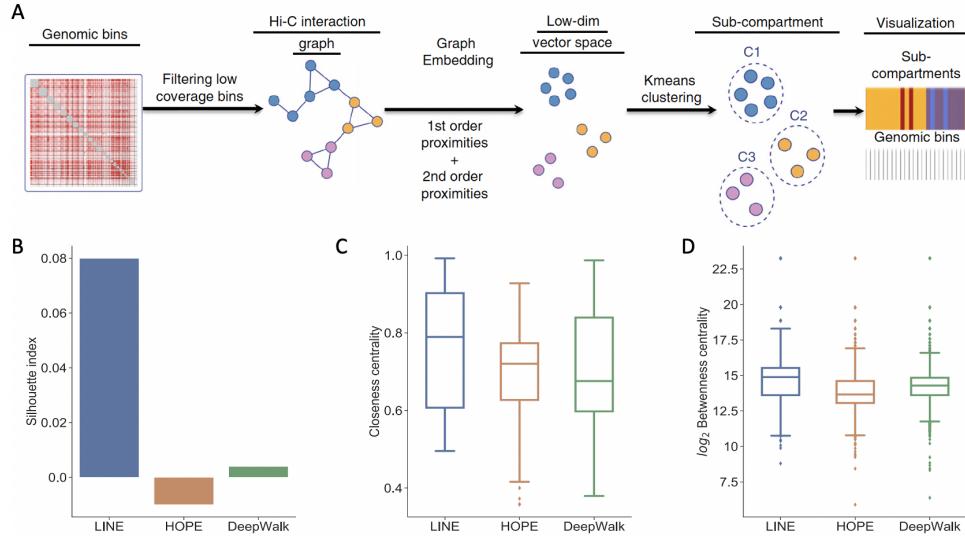


FIG. 13. *Example of applying the deterministic LINE graph embedding method for sub-compartment identification using Hi-C chromatin interaction data.* (A) Workflow from left to right proposed by [1], showing a normalized Hi-C interchromosome matrix used to construct the interaction graph. The nodes with 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 75th and 25th percentiles. Both figures adopted from [1].

means clustering to predict the various hierarchical sub-compartments. The method proposed in [1] based on LINE embedding outperforms the hidden Markov model (HMM) for sub-compartment prediction, and in fact it predicts a larger number (9 versus 5) of sub-compartments. It also outperforms two other graph embedding methods that we described in this review paper, namely HOPE [37] and DeepWalk [39], as shown in Figure 13 (B-D). Genomic regions that map to the same sub-compartment based on the graph embedding method are spatially close to each other. In the study of [1] this is measured quantitatively using the 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 [48] 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 them useful for downstream tasks. The authors of [54] 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, set2Gaussian 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 the 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 [54], Wang et al. demonstrate 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 [54] 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-dimension original graph to a more compatible representation for efficient downstream processing tasks, e.g., node classification, link prediction, community detection, etc. A key

milestone towards this goal in the modern era was the seminal paper on Isomap [49], a nonlinear dimensionality reduction method, which introduced the geodesic distance instead of the Euclidean distance used in the 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 we demonstrated in the previous section 4. For example, the computational complexity of Isomap is  $O(L^2M)$  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.,  $O(N)$ . Hence, one can work with very large graphs with  $N = 20,000$  nodes as in the CORA and PUBMBED 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 aforementioned example in genetic network analysis in section 4.

Broadly, we can classify the graph embedding methods in 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  $O(LM)$ ,  $O(LN)$  or  $O(N)$ . Moreover, in all these methods only unsupervised learning is required. In the Appendix Appendix C, 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 karate club as well as the CORA-ML (see section 4) so that the interested readers can have a head start on the exploration of graph analytics.

In addition to the static graph embedding methods, we also discuss the emerging deep learning-based *dynamic* graph embedding methods, which can be particularly computational 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 D1 in the Appendix.

Finally, for application highlights we present results from four different domains. In the first one, we consider a social network represented by the karate club benchmark for which we employ the DeepWalk for community detection. In the second one, we consider a directed and attributed citation network for the CORA-ML dataset, and we employ the stochastic embedding method G2G for node classification. In the third application, we analyze multi-modality functional brain networks (fMRI and MEG) using MG2G, which is an extension of G2G, for cognitive training effect evaluation and for predicting AD progression. Finally, in the last example we present two different applications of deterministic (LINE method) and stochastic (set2Gauss) 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 aforementioned embedding algorithm SkipGram [31] on the massive open-source Friendster graph [22]

with 68 million nodes and 2.5 billion edges.

**Appendix A. Similarity measures used in the latent space.** We summarize some basic similarity measures commonly used in the vector point-based graph embedding approaches and Gaussian distribution-based graph embedding approaches in [Table A1](#).

TABLE A1  
*Common similarity measures for the graph node embeddings in the latent space.*

| Node representations   | Similarity measures ( $M$ )   |
|------------------------|---|
| Point vectors          | 1) Dot product: $z_j^T z_i$<br>2) Cosine similarity: $\frac{z_i \cdot z_j}{\ z_i\  \times \ z_j\ }$<br>3) Euclidean distance: $\ z_i - z_j\ $   |
| Gaussian distributions | 1) Expected likelihood: $EL(P_i, P_j)$ (see equation (3.5))<br>2) KL-divergence: $D_{KL}(P_j    P_i)$ (see equation (3.6))<br>3) Wasserstein distance: $W_2(P_i; P_j)$ (see equation (3.7)) |

**Appendix B. Summary of Graph Gaussian embedding methods.** We summarize some Graph Gaussian embedding methods in [Table B1](#). The complexity of G2G method and DVNE method is  $O(N)$  and  $O(T \times N \times (d \times S + S \times L + L))$ , respectively, where  $T$  is the number of iterations,  $N$  is the number of nodes in the graph,  $d$  is the average degree of all nodes,  $S$  is the size of hidden layer of the encoder and decoder, and  $L$  is embedding size.

TABLE B1  
*Graph Gaussian embedding methods surveyed in this study. KG2E is for directed knowledge graphs; DVNE is for undirected, non-attributed Graphs; Graph2Gauss is for attributed/non-attributed/directed/undirected graphs; MG2G is for multiple undirected/attributed graphs. T refers to the number of iterations.*

| Method               | Energy function | Loss function                      | Sampling                                   | Optimization                            |
|----------------------|-----------------|------------------------------------|--|---|
| KG2E [17]<br>G2G [4] | KL/EL<br>KL     | margin-based<br>square exponential | Unif/bern<br>Anchor-based<br>sampling<br>– | SGD<br>Adam                             |
| DVNE [68]            | W2              | hybrid loss                        | –  | SGD( $T \leq 50$ ,<br><i>converge</i> ) |
| MG2G [61]            | KL              | square exponential                 | Edge weight-based<br>k-hop<br>neighborhood | –                                       |

**Appendix C. Comparisons of random walk-based graph embedding methods.** In [Table C1](#), we compared three random walk-based graph node embedding approaches (DeepWalk, LINE and node2vec) based on five key properties: node neighborhood sampling techniques, proximity, learning mode, optimization and labeled data.

#### Appendix D. Overview of recent dynamic graph embedding methods.

In the following [Table D1](#), we present references to five main classes of dynamic graph embedding models.

**Appendix E. Implementation Details.** In this section, we introduce specific implementations for two graph embedding based on node2vec [16] and G2G [4] methods. We use two different experimental datasets, which are, respectively, an undirected graph (karate club dataset [64]) without attributes and a directed graph

TABLE C1

**Comparison of random walk-based graph embedding methods.** Among the three graph embedding approaches, “DeepWalk” supports “online” embedding learning since it only preserves local network structure information without considering global network structure preservation during the entire embedding procedure. Moreover, all of these three methods use “shallow networks”. BFS stands for breadth-first sampling, DFS stands for depth-first sampling.

| Method        | Neighborhood sampling | Proximity               | Mode   | Optimization         | labeled data |
|---------------|-----------------------|-------------------------|--------|----------------------|--------------|
| DeepWalk [39] | Truncated random walk | 1st-order               | online | hierarchical softmax | No           |
| LINE [48]     | BFS                   | 1st-order/<br>2nd-order | -      | negative sampling    | No           |
| node2vec [16] | BFS+DFS               | 2nd-order               | -      | negative sampling    | Yes          |

TABLE D1  
*Dynamic graph embedding methods surveyed in this study.*

| Type                        | Model  | code   |
|-----------------------------|--|--|
| Matrix factorization -based | DANE[24]<br>TIMERS [66]<br>DHPE [69]                     | <a href="https://github.com/yyyouc/DANE">https://github.com/yyyouc/DANE</a><br><a href="https://github.com/ZW-ZHANG/TIMERS">https://github.com/ZW-ZHANG/TIMERS</a><br><a href="https://github.com/BingSu12/HDPE">https://github.com/BingSu12/HDPE</a>                                      |
| SkipGram-based              | DNE [45]<br>CTDNE [34]<br>dynnode2vec [26]               | -<br><a href="https://github.com/LogicJake/CTDNE">https://github.com/LogicJake/CTDNE</a><br>-  |
| Autoencoder -based, RNN     | DynGEM [15]<br>Dyngraph2vec [14]<br>NetWalk [63]         | <a href="https://github.com/paulpjoby/DynGEM">https://github.com/paulpjoby/DynGEM</a><br><a href="https://github.com/palash1992/DynamicGEM">https://github.com/palash1992/DynamicGEM</a><br><a href="https://github.com/chengw07/NetWalk">https://github.com/chengw07/NetWalk</a>          |
| GCN-based                   | DyRep [50]<br>TDGNN [40]<br>DySAT [44]<br>EvolveGCN [38] | -<br><a href="https://github.com/stefanosantaris/TDGNN">https://github.com/stefanosantaris/TDGNN</a><br><a href="https://github.com/aravindsankar28/DySAT">https://github.com/aravindsankar28/DySAT</a><br><a href="https://github.com/IBM/EvolveGCN">https://github.com/IBM/EvolveGCN</a> |
| GAN-based                   | DynGraphGAN [58]<br>DynGAN [27]                          | -<br>-   |

(CORA-ML dataset [4]) with extra attributes. The code for these two experiments is written in Python language, incorporating *NetworkX* (network analysis library), *Gensim* (natural language processing library), *scikit-learn* (machine learning library) and *TensorFlow* (deep learning library).

**E.1. Karate club network embedding based on *node2vec*.** To perform graph embedding using the *node2vec* method for the undirected and non-attributed karate club network ( $G1 = (V, E)$ ), we proceed in six steps:

1) Read the original graph ( $G1$ ) from the edge list file saved in *input\_dir*, see Line 2 in [Algorithm E.1](#). Additionally, the karate network dataset (34 nodes, 77 undirected edges, 2 communities) can also be imported using other packages, e.g., DGL package [53] and TORCH\_GEOMETRIC library [13]. 2) Create a graph instance object ( $G$ ) for the *Graph* operation class, initialize the graph, and node neighborhood sampling parameters (return factor  $p$  and in-out factor  $q$ ) with preset default values ( $p = q = 1$ ), respectively, for the BFS and DFS strategies. 3) Compute the transition probability for simulating random walks for nodes in the graph (see Line 4 in [Algorithm E.1](#)); the detailed preprocessing procedure can be found in [16]. 4) Simulate random walks for all nodes in the input graph in terms of the preset random walk hyper-parameters (*num\_walks* and *walk\_length*) and the computed transition probabilities obtained in 3), see Line 5 in [Algorithm E.1](#). 5) Train the neural network-based “word2vec” (SkipGram) model on the training dataset (i.e., generated walks or called “node se-

quences") obtained from step 4) with a stochastic gradient descent (SGD) optimizer in conjunction with *negative sampling* policy in an efficient and unsupervised manner (see Line 6 in [Algorithm E.1](#)), where the context size  $c$  is a preset factor for sampling node context for training. 6) Visualize graph embedding vectors in 2D space by the t-SNE tool in the *scikit-learn* library (see Line 7 in [Algorithm E.1](#)); all nodes can be projected into 2D space as *points*.

The detailed python implementation procedure and functions for the karate club graph embedding using *node2vec* approach are shown in [Algorithm E.1](#).

**Algorithm E.1** Karate club network embedding based on the *node2vec*


---

```

1: Procedure: KARATE_EMBEDDING(input_dir, num_walks, walk_length, p, q,
   L, c, num_iter)
2: G1 = read_graph(input_dir)                                // read undirected karate graph
3: if G1 is not None then
4:   G2 = preprocess_transition_probs(G1, p, q)          // preprocessing the graph
5:   walks = simulate_walks(G2, num_walks, walk_length)    // generate walks
6:   embeddings = learn_embeddings(walks, L, c, num_iter)
7:   vis_vecEmb(embeddings)                                     // visualize embeddings
8: end if

9: Function: SIMULATE_WALKS (G2, num_walks, walk_length)
10: Initialize walks to Empty
11: for iter  $\leftarrow$  0 to num_walks do
12:   for all nodes in G2 do
13:     walk = node2vec_Walk(G2, u, walk_length)           // u is the starting node
14:     Append walk to walks
15:   end for
16: end for
17: return walks
18: End Function

19: Function: LEARN_EMBEDDINGS(walks, L, c, num_iter)
20: if walks is not Empty then
21:   Map node indices as string type in walks
22:   model = Word2Vec(walks, L, num_iter, c)          // c is the context size
23:   embeddings = (model.wv.vectors)                         // karate graph embeddings
24:   model.wv.save_word2vec_format(output)                      // save embeddings
25:   return embeddings
26: end if
27: End Function

28: Function: VIS_VCEMB(embeddings, L)
29: if L is larger than 2 then
30:   t_emb = tsne_project(embeddings, dim = 2) // transform embeddings into 2D
   space
31: else if L equals to 2 then
32:   t_emb = embeddings
33: end if
34: Plot the embedded nodes' positions t_emb                  // see Figure 10 (B)
35: End Function

```

---

**E.2. CORA-ML network embedding based on stochastic graph embedding.** The CORA-ML dataset from the G2G paper [4] represents a citation graph ( $G = (A, X)$ ), where  $A \in \mathbb{R}^{N \times N}$  is the adjacency matrix, and  $X \in \mathbb{R}^{N \times D}$  is the attributed matrix.  $G$  contains  $N = 2995$  nodes,  $E = 8416$  directed edges and  $D = 2879$  dimensional *attribute vector* for every node. In order to learn stochastic graph embeddings for this directed and attributed CORA-ML graph, each node is projected into a multivariate Gaussian distribution ( $\mathbb{N} = (\mu_i, \Sigma_i)$ ) with mean ( $\mu_i \in \mathbb{R}^{L/2}$ ) and

covariance matrix ( $\Sigma_i \in \mathbb{R}^{L/2 \times L/2}$ ). This embedding process mainly follows five steps (see detailed pseudo code description in [Algorithm E.2](#)):

1) Load the input graph from the file path (“filepath”), and obtain the adjacency matrix ( $A$ ), attribute matrix ( $X$ ) and node labels ( $z$ ) for the subsequent stochastic graph embedding, see Line 2 in [Algorithm E.2](#). 2) Sample node triplets based on “k-hop neighborhoods” ( $K$ ); each node triplet comprises an anchor node and the corresponding *positive* and *negative* nodes w.r.t. the anchor node. 3) In order to evaluate the G2G model and demonstrate the effectiveness of the learned embeddings through performing “link prediction” task on a “validation set”, graph edges in  $A$  are split into three subsets (training, validation and test sets) based on the input ratios of validation and test dataset ( $p_{val} = 0.10, p_{test} = 0.05$ ). Moreover, equal number of non-edges are sampled for validation and test. 4) Train the deep encoder-based G2G model with unsupervised manner, and minimize the square-exponential loss function (see equation in [Equation \(3.11\)](#)) with the Adam optimizer (learning rate = 1e-3) provided in TensorFlow ( $\geq 1.4$ ); the output Gaussian embeddings consist of mean and variance vectors (see Line 6 of [Algorithm E.2](#)) 5) Visualize the stochastic graph embeddings into 2D space using the t-SNE tool in *scikit-learn*; the *vis.GaussEmb* function (described in Line 35 in [Algorithm E.2](#)) was implemented to project every node’s Gaussian embeddings (both *mean* and *variance* vectors) into 2D Euclidean space as a “probability ellipse” (see [Figure 11\(B\)](#)). Specifically, the centroid of each ellipse represents the *mean* projection result, and the contour represents the predicted uncertainty of the node embedding by *sigma* projection using t-SNE. 6) Apply the learned graph embeddings (“mean” or “variance” vectors) for downstream tasks, e.g., node classification (see the example of implementation in Line 19 in [Algorithm E.2](#)).

---

**Algorithm E.2** CORA-ML stochastic embedding based on *G2G*

---

```

1: Procedure: GAUSSIAN_EMBEDDING (data path filepath, hop k, embedding
   size L)
2: [A, X, z] = load_dataset(filepath)                                // load graph data
3: while A is not Empty do
4:   g2g = Graph2Gauss(A, X, L, k, p_val, p_test, p_nodes)
5:   sess = g2g.train()                                              // train the G2G model
6:   mu, sigma = sess.run([g2g.mu, g2g.sigma]) // output Gaussian embeddings
7:   test_auc, test_ap = link_pred(test_label, test.edges)
8:   f1_micro, f1_macro = node_clf(mu, z, n_repeat = 10, norm = True)
9:   vis_GaussEmb(mu, sigma, L)
10: end while

11: Function: LINK_PRED(test_labels, test_edges)
12: while test_edges is not Null do
13:   scores = energy_kl(test.edges)
14:   auc = roc_auc_score(test_labels, scores)
15:   ap = average_precision_score(test_labels, scores)
16: end while
17: return auc, ap
18: End Function

19: Function: NODE_CLF(mu, z, n_repeat = 10, norm = True)
20: lrcv = LogisticRegressionCV()
21: if norm is TRUE then
22:   Normalize the feature matrix mu
23: end if
24: Initialize F1_score to Empty
25: for seed  $\leftarrow$  0 to n_repeat do
26:   Shuffle and split feature matrix (mu) and labels (z)
27:   lrcv.fit(x_train, y_train)
28:   predict = lrcv.predict(x_test)
29:   f1_micro = f1_score(y_test, predict, average = 'micro')
30:   f1_macro = f1_score(y_test, predict, average = 'macro')
31:   Append (f1_micro, f1_macro) to F1_score
32: end for
33: return Mean F1_score
34: End Function

35: Function: VIS_GAUSSSEMB(mu, sigma, L)
36: if L is larger than 2 then
37:   t_mu, t_sigma = tsne_project(mu, sigma)                  // transform into 2D space
38: else if L equals to 2 then
39:   t_mu = mu, t_sigma = sigma
40: end if
41: Plot the embedded positions (t_mu) for all vertices
42: Plot the embedded uncertainty (t_sigma) for all vertices
43: End Function

```

---

## 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*, Nature communications, 11 (2020), pp. 1–11.
- [2] M. BELKIN AND P. NIYOGI, *Laplacian eigenmaps and spectral techniques for embedding and clustering*, in Advances in neural information processing systems, 2002, pp. 585–591.
- [3] G. BERTASIU, L. TORRESANI, S. X. YU, AND J. SHI, *Convolutional random walk networks for semantic image segmentation*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 858–866.
- [4] A. BOJCHEVSKI AND S. GÜNNEMANN, *Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking*, in ICLR, 2018.
- [5] K. BOLLACKER, C. EVANS, P. PARITOSH, T. STURGE, AND J. TAYLOR, *Freebase: a collaboratively created graph database for structuring human knowledge*, in Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008, pp. 1247–1250.
- [6] S. BRIN AND L. PAGE, *The anatomy of a large-scale hypertextual web search engine. 1998*, in Proceedings of the Seventh World Wide Web Conference, 2017.
- [7] C. BRUSS, A. KHAZANE, J. RIDER, R. SERPE, A. GOGOGLOU, AND K. E. HINES, *Deeptrax: Embedding graphs of financial transactions*, arXiv preprint arXiv:1907.07225, (2019).
- [8] C. BRUSS, A. KHAZANE, J. RIDER, R. SERPE, S. NAGRECHA, AND K. E. HINES, *Graph embeddings at scale*, arXiv preprint arXiv:1907.01705, (2019).
- [9] H. CAI, V. W. ZHENG, AND K. C.-C. CHANG, *A comprehensive survey of graph embedding: Problems, techniques, and applications*, IEEE Transactions on Knowledge and Data Engineering, 30 (2018), pp. 1616–1637.
- [10] S. CAO, W. LU, AND Q. XU, *Grarep: Learning graph representations with global structural information*, in Proceedings of the 24th ACM international on conference on information and knowledge management, 2015, pp. 891–900.
- [11] P. CUI, X. WANG, J. PEI, AND W. ZHU, *A survey on network embedding*, IEEE Transactions on Knowledge and Data Engineering, 31 (2018), pp. 833–852.
- [12] K. DING, J. LI, R. BHANUSHALI, AND H. LIU, *Deep anomaly detection on attributed networks*, in Proceedings of the 2019 SIAM International Conference on Data Mining, SIAM, 2019, pp. 594–602.
- [13] M. FEY AND J. E. LENSSSEN, *Fast graph representation learning with PyTorch Geometric*, in ICLR Workshop on Representation Learning on Graphs and Manifolds, 2019.
- [14] P. GOYAL, S. R. CHHETRI, AND A. CANEDO, *dyngraph2vec: Capturing network dynamics using dynamic graph representation learning*, Knowledge-Based Systems, 187 (2020), p. 104816.
- [15] P. GOYAL, N. KAMRA, X. HE, AND Y. LIU, *Dyngem: Deep embedding method for dynamic graphs*, arXiv preprint arXiv:1805.11273, (2018).
- [16] A. GROVER AND J. LESKOVEC, *node2vec: Scalable feature learning for networks*, in Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp. 855–864.
- [17] S. HE, K. LIU, G. JI, AND J. ZHAO, *Learning to represent knowledge graphs with gaussian embedding*, in Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, 2015, pp. 623–632.
- [18] G. E. HINTON AND S. T. ROWEIS, *Stochastic neighbor embedding*, in Advances in neural information processing systems, 2003, pp. 857–864.
- [19] 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 Reports, 12 (2018), pp. 50–60.
- [20] J. KUROSE, *On computing per-session performance bounds in high-speed multi-hop computer networks*, ACM SIGMETRICS Performance Evaluation Review, 20 (1992), pp. 128–139.
- [21] Y. LEUNCUN, S. CHOPRA, R. HADSELL, M. RANZATO, AND F. HUANG, *A tutorial on energy-based learning*, Predicting structured data, 1 (2006).
- [22] J. LESKOVEC AND A. KREVL, *SNAP Datasets: Stanford large network dataset collection*. <http://snap.stanford.edu/data>, June 2014.
- [23] J. LESKOVEC AND J. J. MCAULEY, *Learning to discover social circles in ego networks*, in Advances in neural information processing systems, 2012, pp. 539–547.
- [24] J. LI, H. DANI, X. HU, J. TANG, Y. CHANG, AND H. LIU, *Attributed network embedding for learning in a dynamic environment*, in Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, 2017, pp. 387–396.
- [25] 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*, International journal of molecular sciences, 20 (2019), p. 3389.
- [26] S. MAHDAVI, S. KHOSHRAFTAR, AND A. AN, *dynnode2vec: Scalable dynamic network embedding*, in 2018 IEEE International Conference on Big Data (Big Data), IEEE, 2018, pp. 3762–3765.
  - [27] A. MAHESHWARI, A. GOYAL, M. K. HANAWAL, AND G. RAMAKRISHNAN, *Dyngan: Generative adversarial networks for dynamic network embedding*, in Graph Representation Learning Workshop at NeurIPS, 2019.
  - [28] J. McAULEY AND J. LESKOVEC, *Image labeling on a network: using social-network metadata for image classification*, in European conference on computer vision, Springer, 2012, pp. 828–841.
  - [29] A. K. McCALLUM, K. NIGAM, J. RENNIE, AND K. SEYMORE, *Automating the construction of internet portals with machine learning*, Information Retrieval, 3 (2000), pp. 127–163.
  - [30] A. MHEICH, F. WENDLING, AND M. HASSAN, *Brain network similarity: methods and applications*, Network Neuroscience, 4 (2020), pp. 507–527.
  - [31] T. MIKOLOV, K. CHEN, G. CORRADO, AND J. DEAN, *Efficient estimation of word representations in vector space*, arXiv preprint arXiv:1301.3781, (2013).
  - [32] G. A. MILLER, *WordNet: An electronic lexical database*, MIT press, 1998.
  - [33] P. NERURKAR, M. CHANDANE, AND S. BHIRUD, *Survey of network embedding techniques for social networks*, Turkish Journal of Electrical Engineering & Computer Sciences, 27 (2019), pp. 4768–4782.
  - [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 2018 IEEE International Conference on Big Data (Big Data), IEEE, 2018, pp. 1085–1092.
  - [35] A. N. NIKOLAKOPOULOS AND G. KARYPIS, *Recwalk: Nearly uncoupled random walks for top-n recommendation*, in Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, 2019, pp. 150–158.
  - [36] M. OKUDA, S. SATOH, Y. SATO, AND Y. KIDAWARA, *Community detection using restrained random-walk similarity*, IEEE Transactions on Pattern Analysis and Machine Intelligence, (2019).
  - [37] M. OU, P. CUI, J. PEI, Z. ZHANG, AND W. ZHU, *Asymmetric transitivity preserving graph embedding*, in Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp. 1105–1114.
  - [38] A. PAREJA, G. DOMENICONI, J. CHEN, T. MA, T. SUZUMURA, H. KANEZASHI, T. KALER, T. B. SCHARDL, AND C. E. LEISERSON, *Evolvecn: Evolving graph convolutional networks for dynamic graphs*, in AAAI, 2020, pp. 5363–5370.
  - [39] B. PEROZZI, R. AL-RFOU, AND S. SKIENA, *Deepwalk: Online learning of social representations*, in Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, pp. 701–710.
  - [40] L. QU, H. ZHU, Q. DUAN, AND Y. SHI, *Continuous-time link prediction via temporal dependent graph neural network*, in Proceedings of The Web Conference 2020, 2020, pp. 3026–3032.
  - [41] M. RINGNÉR, *What is principal component analysis?*, Nature biotechnology, 26 (2008), pp. 303–304.
  - [42] 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*, Nature communications, 9 (2018), pp. 1–12.
  - [43] S. T. ROWEIS AND L. K. SAUL, *Nonlinear dimensionality reduction by locally linear embedding*, science, 290 (2000), pp. 2323–2326.
  - [44] A. SANKAR, Y. WU, L. GOU, W. ZHANG, AND H. YANG, *Dysat: Deep neural representation learning on dynamic graphs via self-attention networks*, in Proceedings of the 13th International Conference on Web Search and Data Mining, 2020, pp. 519–527.
  - [45] X. SHEN, S. PAN, W. LIU, Y.-S. ONG, AND Q.-S. SUN, *Discrete network embedding*, in Proceedings of the 27th International Joint Conference on Artificial Intelligence, 2018, pp. 3549–3555.
  - [46] C. SU, J. TONG, Y. ZHU, P. CUI, AND F. WANG, *Network embedding in biomedical data science*, Briefings in bioinformatics, 21 (2020), pp. 182–197.
  - [47] Q. TAN, N. LIU, AND X. HU, *Deep representation learning for social network analysis*, Frontiers in Big Data, 2 (2019), p. 2.
  - [48] J. TANG, M. QU, M. WANG, M. ZHANG, J. YAN, AND Q. MEI, *Line: Large-scale information network embedding*, in Proceedings of the 24th international conference on world wide web, 2015, pp. 1067–1077.
  - [49] J. B. TENENBAUM, V. DE SILVA, AND J. C. LANGFORD, *A global geometric framework for nonlinear dimensionality reduction*, science, 290 (2000), pp. 2319–2323.

- [50] R. TRIVEDI, M. FARAJTABAR, P. BISWAL, AND H. ZHA, *Dyrep: Learning representations over dynamic graphs*, in International Conference on Learning Representations, 2018.
- [51] L. VILNIS AND A. MCCALLUM, *Word representations via gaussian embedding*, arXiv preprint arXiv:1412.6623, (2014).
- [52] D. WANG, P. CUI, AND W. ZHU, *Structural deep network embedding*, in Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp. 1225–1234.
- [53] 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*, arXiv preprint arXiv:1909.01315, (2019).
- [54] S. WANG, E. R. FLYNN, AND R. B. ALTMAN, *Gaussian embedding for large-scale gene set analysis*, Nature Machine Intelligence, 2 (2020), pp. 387–395.
- [55] 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.
- [56] Z. WANG, J. ZHANG, J. FENG, AND Z. CHEN, *Knowledge graph embedding by translating on hyperplanes*, in AAAI, vol. 14, 2014, pp. 1112–1119.
- [57] J. D. WEST, I. WESLEY-SMITH, AND C. T. BERGSTROM, *A recommendation system based on hierarchical clustering of an article-level citation network*, IEEE Transactions on Big Data, 2 (2016), pp. 113–123.
- [58] Y. XIONG, Y. ZHANG, H. FU, W. WANG, Y. ZHU, AND S. Y. PHILIP, *Dyngraphgan: Dynamic graph embedding via generative adversarial networks*, in International Conference on Database Systems for Advanced Applications, Springer, 2019, pp. 536–552.
- [59] 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 computational biology, 13 (2017), p. e1005746.
- [60] 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*, arXiv preprint arXiv:2005.05784, (2020).
- [61] 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*, bioRxiv, (2019), p. 779744.
- [62] 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 computational biology, 16 (2020), p. e1008186.
- [63] W. YU, W. CHENG, C. C. AGGARWAL, K. ZHANG, H. CHEN, AND W. WANG, *Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks*, in Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 2672–2681.
- [64] W. W. ZACHARY, *An information flow model for conflict and fission in small groups*, Journal of anthropological research, 33 (1977), pp. 452–473.
- [65] D. ZHANG, J. YIN, X. ZHU, AND C. ZHANG, *User profile preserving social network embedding*, in IJCAI International Joint Conference on Artificial Intelligence, 2017.
- [66] Z. ZHANG, P. CUI, J. PEI, X. WANG, AND W. ZHU, *Timers: Error-bounded svd restart on dynamic networks*, in Thirty-Second AAAI Conference on Artificial Intelligence, 2018.
- [67] Y. ZHOU AND H. LI, *Asset diversification and systemic risk in the financial system*, Journal of Economic Interaction and Coordination, 14 (2019), pp. 247–272.
- [68] D. ZHU, P. CUI, D. WANG, AND W. ZHU, *Deep variational network embedding in wasserstein space*, in Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 2827–2836.
- [69] D. ZHU, P. CUI, Z. ZHANG, J. PEI, AND W. ZHU, *High-order proximity preserved embedding for dynamic networks*, IEEE Transactions on Knowledge and Data Engineering, 30 (2018), pp. 2134–2144.