GRAPH EMBEDDING METHODS AND THEIR APPLICATIONS

A Brief Introduction to Graph Embedding

Nam Le[abc]

[a]Department of Computer Science
[b]Faculty of Information Technology, University of Science, Ho Chi Minh City, Vietnam
[c]Vietnam National University, Ho Chi Minh City, Vietnam

October 25, 2024

Introduction

A powerful language

"Everything is a Graph"

Examples

- Excel
- Package Management -> Directed acyclic (no cycles) graph (DAG)
- Graph Databases -> Neo4j
- Software Pipelines -> Continuous Integration and Continuous Deployment (CI/CD)
- Machine Learning -> Computation graphs with trillions of edges
- Compilers -> Control-flow graph (CFG)
- Others: Social graphs, Knowledge graphs, GPS or map routing, Computer networks

Euclidean vs. Non-Euclidean data structures

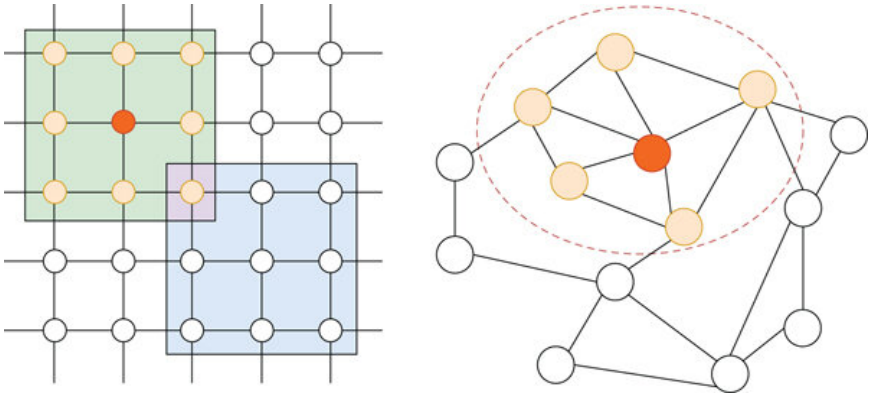


Figure 1: **Euclidean vs. Non-Euclidean data structures**.

Intuition of graph embedding techniques

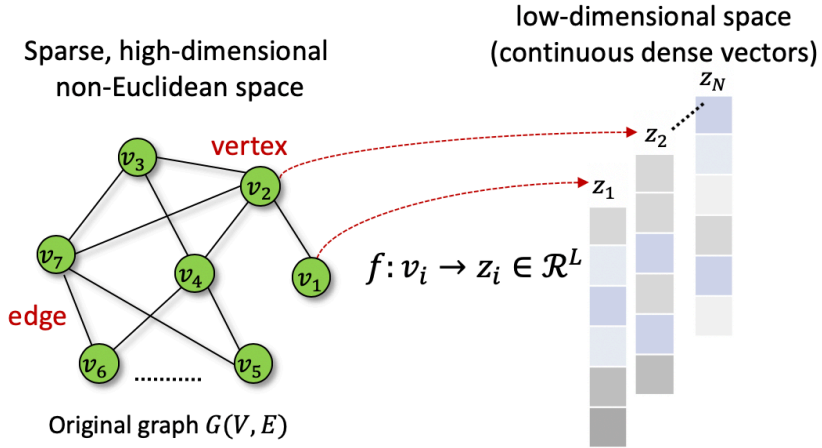


Figure 2: **Schematic of graph (node) embedding**.

Three major categories of graph embedding techniques

- Matrix factorization-based methods -> high-order proximity matrix based on probabilities -> hard to scale!
- Random walk-based methods (Interested)
- Neural network-based methods (Interested)

Mathematical Formulation

Basic notations

Given a graph $G(V, E)$

- A node (or vertex) set $V = \{v_1, v_2, v_3, ..., v_n\}$
- An edge (or link) set E
- $e_{ij} = (v_i, v_j) =$ connection between two different nodes v_i and v_j

Directed/un-directed graphs

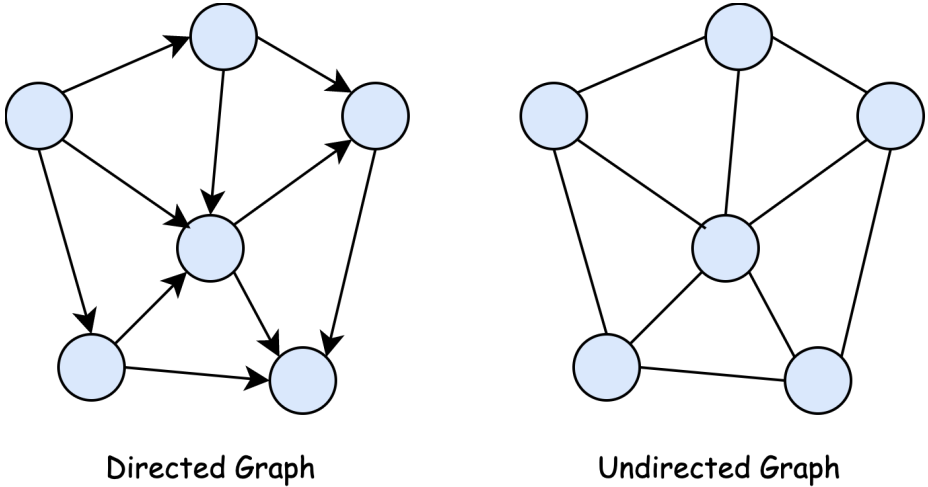


Figure 3: **Directed vs. un-directed graphs.**

Homogeneous/heterogeneous graphs

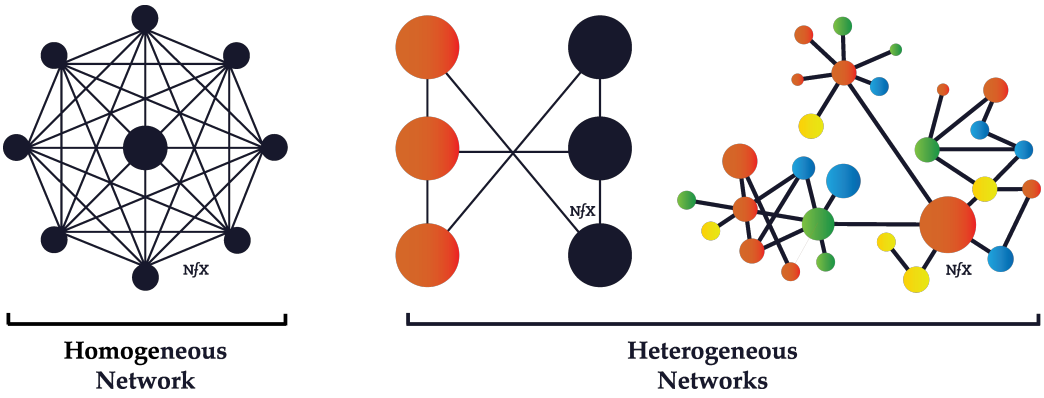


Figure 4: **Homogeneous vs. heterogeneous graphs.**

Weighted/binary graphs

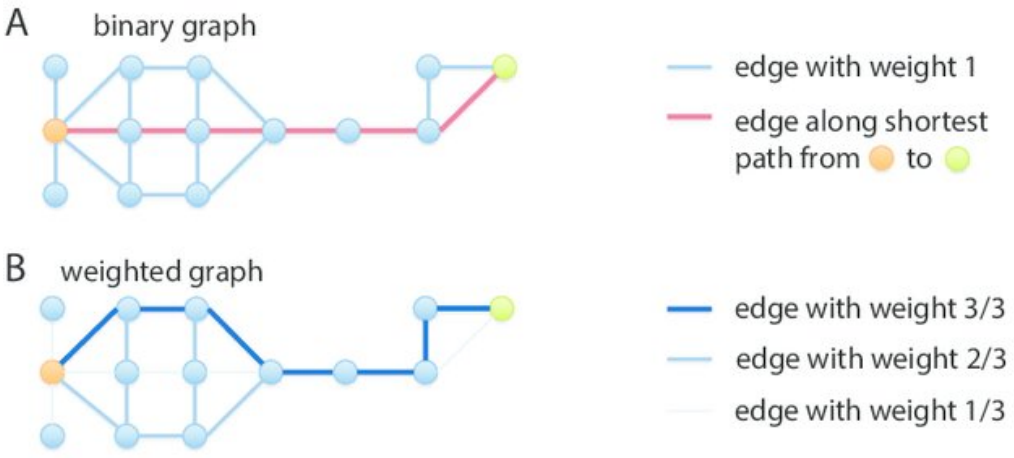


Figure 5: **Weighted vs. binary graphs.**

Adjacency matrix representations

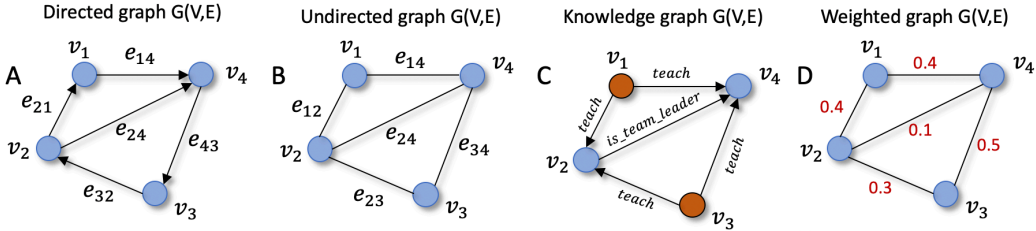


Figure 6: **Different types of graphs and their corresponding adjacency matrix representations**.

Types of static graph representation

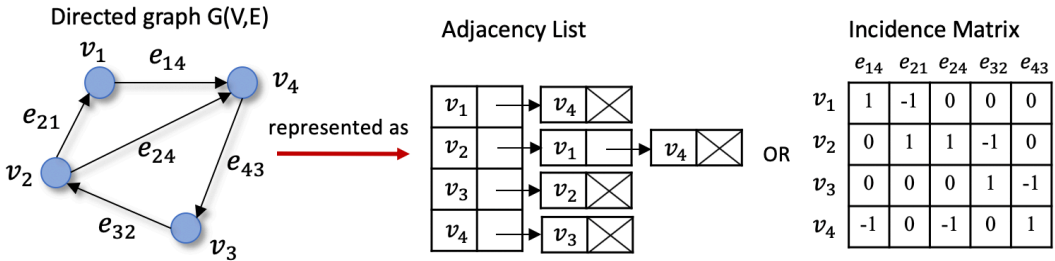


Figure 7: **Different ways to represent a static graph.**

Input - Output

- Input: given a graph $G = (V, E)$
- Output: graph node embeddings (L dimension, $L \ll |V|$)

The learning task

The task is learning a projection ϕ, encode graph nodes ($V = \{v_i | i = 1, 2, ..., |V|\}$) from a high-dimensional space into a low-dimensional space.

How many form of graph node embedding?

Actually, two. Deterministic point vectors; or Stochastic probabilistic distributions.

Input - Output: Deterministic point vectors

How many form of graph node embedding?

Actually, two. Deterministic point vectors; or Stochastic probabilistic distributions.

Deterministic point vectors:

$$\Phi = \{z_i \in \mathbb{R}^L | i = 1, 2, ..., |V|\} \tag{1}$$

Input - Output: Stochastic probabilistic distributions

How many form of graph node embedding?

Actually, two. Deterministic point vectors; or Stochastic probabilistic distributions.

Stochastic probabilistic distributions:

$$\Phi = \{\mathbb{P}_i \sim \mathcal{N}(\mu_i, \Sigma_i) | i = 1, 2, ..., |V|\} \tag{2}$$

where the mean vector $\mu_i \in \mathbb{R}^{L/2}$, and the covariance matrix $\Sigma_i \in \mathbb{R}^{L/2 \times L/2}$

Input - Output

How to preserve maximally graph structure properties in embedded space?

To this end, we need to defined node pairwise similarity (e.g., dot product-$z_j^T z_i$)

- $Sim(v_i, v_j) \approx z_j^T z_i$ for vector-based graph embedding;
- $Sim(v_i, v_j) \approx \mu_j^T \mu_i$ for Gaussian distribution-based graph embedding.

Summary

The specific graph node embedding formula is

$$\phi : v_i \rightarrow \begin{cases} z_i \in \mathbb{R}^L \ \ (i = 1, 2, ..., |V|) \ \ or \\ P_i \sim \mathcal{N}(\mu_i, \Sigma_i), \mu_i \in \mathbb{R}^{L/2}, \Sigma_i \in \mathbb{R}^{L/2 \times L/2} \ \ (i = 1, 2, ..., |V|). \end{cases} \tag{3}$$

The goal of graph embedding

$$\begin{aligned} \min \quad & (\text{similarity in embedded space}, \text{similarity in origial space}) \\ \text{s.t.} \quad & \text{preserving the graph structure property} \end{aligned}$$

Key components

- Graph structure property preservation
- Node similarity measurement in both the original and latent space
- An encoder

First-order proximity

First-order proximity between two nodes is equal to their edge weight.

Advantages

It is sufficient for preserving local network structure property.

Question?

Why?

First-order proximity

First-order proximity between two nodes is equal to their edge weight.

Disadvantages

It is insufficient for preserving the global network structure.

Question?

Why?

First-order proximity

Objective function $\mathbb{E}(.)$ for first-order proximity methods:

$$\mathbb{E} = D_{KL}(\hat{p_1}||p_1) = -\sum_{i,j \in E} w_{ij} \log(p_1(v_i, v_j)) \tag{4}$$

with

$$\hat{p}_1(v_i, v_j) = \frac{w_{ij}}{\sum_{s,t \in E} w_{st}} \qquad p_1(v_i, v_j) = \frac{exp(z_i^T z_j)}{\sum_{s,t \in E} exp(z_s^T z_t)}. \tag{5}$$

where 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.

Second-order proximity

Question

Based on Disadvantages of First-order proximity, we need to develop more expressive proximity to capture global network structure.

Hint

The neighborhoods of nodes?

Second-order proximity

Ideas

We can estimate the transitional probability metric between two nodes. Thus allow us to measure proximity between the neighborhood structures of the nodes, i.e., the global network structure property.

Second-order proximity

Objective function $\mathbb{E}'(.)$ for second-order proximity methods:

$$\mathbb{E}' = \sum_{i \in V} D_{KL}(\hat{p}_2(\cdot|v_i)||p_2(\cdot|v_i)) = -\sum_{i,j \in E} w_{ij} \log(p_2(v_j|v_i)) \tag{6}$$

with

$$\hat{p}_2(v_j|v_i) = \frac{w_{ij}}{\sum_{k \in V} w_{ik}}, \qquad p_2(v_j|v_i) = \frac{exp(z_j'^T z_i)}{\sum_{k \in V} exp(z_k'^T z_i)}. \tag{7}$$

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

Note that, z_i' is the representation of v_i when it is treated as a specific "context"

High-order proximity

Question

Are First and Second-order proximity enough? :D Maybe

Usually, high-order proximity can be defined based on some metrics, such as **Rooted PageRank**.

High-order proximity

The rooted PageRank for node x calculates the stationary distribution of a random walker starting at x, who iteratively moves to a random neighbor of its current position with probability α or returns to x with probability $1 - \alpha$.
Let

- π_x denote the stationary distribution vector.
- $[\pi_x]_i$ denote the probability that the random walker is at node i under the stationary distribution.

High-order proximity: Rooted PageRank

Let P be the transition matrix with $P_{i,j} = \frac{1}{|\Gamma(v_j)|}$ if $(i,j) \in E$ and $P_{i,j} = 0$ otherwise. Let $\mathbf{e}_x$ be a vector with the x^{th} element being 1 and others being 0. The stationary distribution satisfies

$$\pi_x = \alpha P \pi_x + (1-\alpha)\mathbf{e}_x. \tag{8}$$

High-order proximity: Rooted PageRank

For link prediction, the score for (x, y) is given by $[\pi_x]_y$ (or $[\pi_x]_y + [\pi_y]_x$ for symmetry)

Motivation

Selecting a proper node similarity measure is particularly important for finding effective node context (or neighbors) aiding for optimizing diverse graph embedding models.

Five different aspects of Similarity between nodes

- Presence of edge
- Overlapped neighborhood
- Reachable by k-hops
- Reachable through random walks
- Similar node attributes

Type of node similarity definitions

- Multi-hop neighborhood-based
- Random walk-based
- Adjacency matrix-based methods

Key idea of graph embedding

Optimizing low-dimensional embeddings to approximate the node similarities in the original graph generated by the above measures.

Multi-hop neighborhood-based node similarity

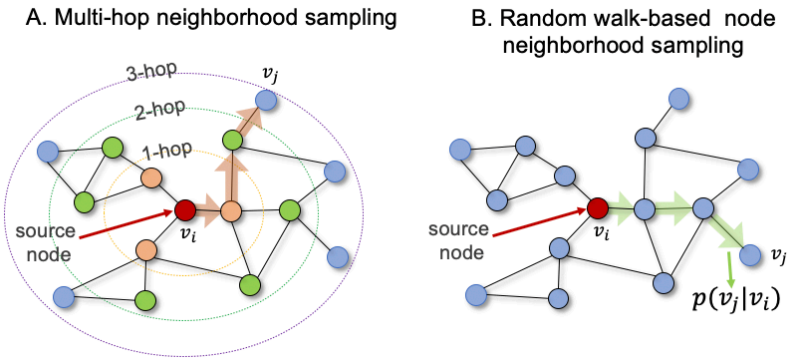


Figure 8: **Node neighborhood sampling techniques.**

Multi-hop neighborhood-based node similarity

- 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.

Multi-hop neighborhood-based node similarity: "hop"

Question

What is "hop" terminology? Where it come from?

Multi-hop neighborhood-based node similarity: k-hop neighborhood

Definition

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)

What k-hop can do for us?

- enables us to sample similar neighbors for every sampled source node at different hops in the original graphs.
- higher-order proximity can be preserved by increasing the number of hops.

Random walk-based node similarity

Random walks = stochastic similarity measures for various problems such as

- content recommendation
- community detection
- image segmentation

Random walk-based node similarity

Definition

Node similarity ($Sim(v_i, v_j)$) = probability $p(v_j|v_i)$ of reaching a node v_j on a random walk ($W_{v_i} = r_0, r_1, ..., r_l$) of length l over the graph from the source node $r_0 = v_i$. (See, example B in Fig. 8)

Random walk-based node similarity

Nodes in the random walk are generated based on the distribution

$$p(r_i = x | r_{i-1} = v) = \begin{cases} \pi_{vx}/Z & if (v, x) \in E, \\ 0 & otherwise. \end{cases} \quad (9)$$

where r_i denotes the $i - th$ node in the random walk starting from $r_0 = u$, π_{vx} denotes un-normalized transition probability between nodes v and x, and Z is a normalizing constant

Random walk-based node similarity: Techniques

- DeepWalk methods [**?**]
- LINE [**?**]
- node2vec [**?**]

Random walk-based node similarity: Comparison

- DOES NOT need to train all the node pairs but it only considers the node pairs that co-occur on random walks.

Adjacency matrix-based node similarity

Example

How about presencing of edges between nodes in the adjacency matrices of original graphs?
=> Adjacency matrix-based node similarity.

Adjacency matrix-based node similarity

Classic matrix factorization-based

- Locally linear embedding [**?**]
- Laplacian eigenmaps [**?**]
- HOPE [**?**]

Stochastic neighborhood embedding (SNE) [**?**]

Point vector node embeddings

Table 1: **Common similarity measures for the graph node embeddings in the latent space (Point vector).**

Node representations	Similarity measures (M)
Point vectors	1) Dot product: $z_j^T z_i$ 2) Cosine similarity: $\frac{z_i \cdot z_j}{\|\|z_i\|\| \times \|\|z_j\|\|}$ 3) Euclidean distance: $\|\|z_i - z_j\|\|$

Gaussian distribution node embeddings

Table 2: **Common similarity measures for the graph node embeddings in the latent space (Point vector).**

Node representations	Similarity measures (M)
Gaussian distributions	1) Expected likelihood 2) KL-divergence: $D_{KL}(P_j \| \| P_i))$ 3) Wasserstein distance: $W_2(P_i; P_j)$

Graph Embedding Methods

Main purpose of vector point-based graph embedding

The main purpose of vector point-based graph embedding

Projecting high-dimensional graph nodes into low-dimensional vectors in a latent space, while preserving the original graph structure properties, see a specific definition in Eq. 3.

Main purpose of vector point-based graph embedding

The “closeness” between variant nodes in the original space can be modeled in different ways by using the measures.

Main pipeline of random walk-based graph node embedding

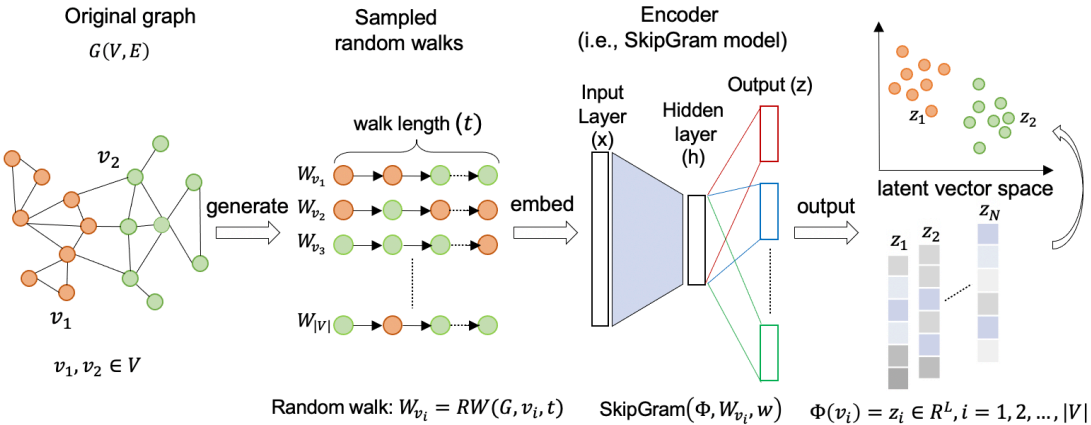


Figure 9: Pipeline for random walk-based graph embedding methods.

Node context generation based on random walks

Deepwalk

- Step 1: adopted the random walk-based node neighbors sampling technique
- Step 2: employed an encoder called "SkipGram" model to learn graph node embedding

Advantages:

- Effectively encode the structure and topological information from the original graph into the latent space.

Limitations:

- Only capture the local structure information by using the truncated random walks, but the global structure information is missing.

Node context generation based on random walks: node2vec

Question

How to capture both local and global structure information?

Solutions: incorporates both

- BFS (breadth-first-search) (p)
- DFS (depth-first-search) (q)

Node context generation based on random walks: node2vec

Biased random walk

We upgrade the formula of random walk in Eq. 9. Assume a random walk just traversed nodes t, v, and resides at v, then the unnormalized transition probability (π_{vx}) between the node v and the next walk node x can be computed

$$\pi_{vx} = \alpha_{pq}(t,x) \cdot k_{vx}, \quad where \;\; \alpha_{pq}(t,x) = \begin{cases} 1/p & if \quad d_{tx} = 0, \\ 1 & if \quad d_{tx} = 1, \\ 1/q & if \quad d_{tx} = 2, \end{cases} \tag{10}$$

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

Biased random walk schema

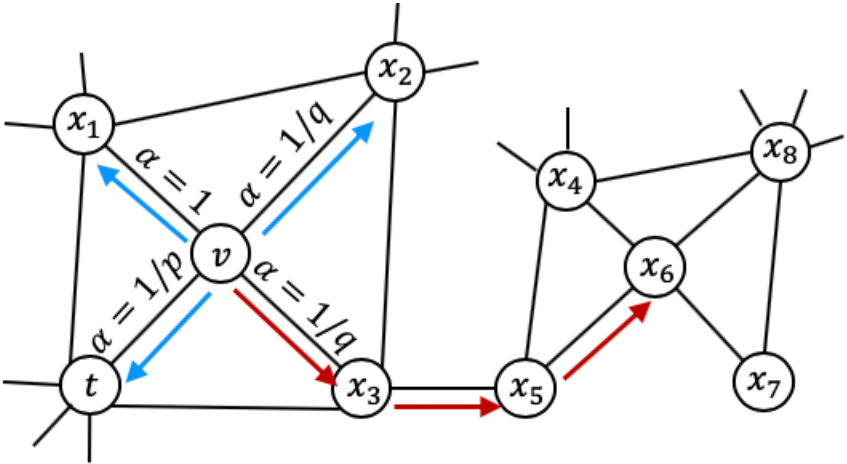


Figure 10: **Biased random walk schema incorporating both BFS and DFS.** Blue arrows show the breadth-first search (BFS) directions while red arrows indicate the depth-first search (DFS) directions.

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.

Learn node embedding with an encoder

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. The parameters of the embedding model can be learned by minimizing the cross-entropy loss function ($\mathcal{L}$) as:

$$\mathcal{L} = -\log \sum_{u \in V} \sum_{v \in N_R(u)} p(v|z_u) = -\log \sum_{u \in V} \sum_{v \in N_R(u)} \frac{exp(z_u^T z_v)}{\sum_{n \in V} exp(z_u^T z_n)}, \tag{11}$$

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.

Learn node embedding with an encoder: node2vec

node2vec applies stochastic gradient descent (SGD) with "negative sampling" for more efficient model optimization. According to Eq. 11, the simplified objective function ($\mathcal{L}^{'}$) for node2vec using "negative sampling" is shown:

$$\mathcal{L}^{'} = -\sum_{u \in V} \sum_{v \in N_R(u)} (\log(\sigma(z_u^T z_v)) - \sum_{i=1}^{k} \log(\sigma(z_u^T z_{n_i}))), n_i \sim P_v, \tag{12}$$

where σ 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 Eq. 12, we just need to normalize against "k" random negative samples, which can effectively speed up the training process and greatly reduce the computational complexity.

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 ($\Phi \in \mathbb{R}^{|\mathcal{V}| \times L}$) in the latent space, which are shown in the last column of Fig. 9.

Overview

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:

- it enables to effectively incorporate useful unstructured attribute information associated with each node,
- 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 mathematical problem definition for Gaussian distribution-based graph embedding is shown in Eq. 3.

word2Gauss

Summary:

- embeds words as Gaussian distributional potential functions in an infinite dimensional functional space.
- 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 current graph Gaussian embedding methods

The current graph Gaussian embedding methods:

- Gaussian embedding for knowledge graphs
- Gaussian embedding for attributed graphs

with three main components:

- Triplet pair generation
- Energy function
- Loss function

Gaussian embedding for knowledge graphs

An energy-based learning framework learns graph Gaussian representations (or embeddings) by using a *margin-based triplet ranking loss* ($\mathcal{L}$) as shown in Eq. 13 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 (γ).

$$\mathcal{L} = \sum_{(h,r,t)\in\tau} \sum_{(h',r',t')\in\tau'} max(0, E_{pos}(h,r,t) - \gamma + E_{neg}(h',r',t')). \tag{13}$$

Gaussian embedding for knowledge graphs: Energy function

There are two different ways to define the energy function:

- Expected likelihood ("EL") of inner product-based symmetric similarity, with the specific formula for denoting a positive triplet EL energy function $E_{pos}(h, r, t)$

$$\begin{aligned} E_{pos}(h, r, t) &= E_{pos}(P_e, P_r) \\ &= \int_{x \in \mathbb{R}^{k_e}} \mathcal{N}(x; \mu_e, \Sigma_e)\mathcal{N}(x; \mu_r, \Sigma_r)dx \\ &= \mathcal{N}(x; \mu_e - \mu_r, \Sigma_e + \Sigma_r) \end{aligned} \tag{14}$$

Gaussian embedding for knowledge graphs: Energy function

There are two different ways to define the energy function:

- KL-divergence (asymmetric similarity), with the specific formula for denoting the positive triplet KL energy function $E_{pos}(h, r, t)$

$$
\begin{aligned}
E_{pos}(h, r, t) &= E_{pos}(P_e, P_r) = D_{KL}(P_e || P_r) \\
&= \int_{x \in \mathbb{R}^{k_e}} \mathcal{N}(x; \mu_r, \Sigma_r) \log \frac{\mathcal{N}(x; \mu_r, \Sigma_r)}{\mathcal{N}(x; \mu_e, \Sigma_e)} dx \\
&= \frac{1}{2} \{ tr(\Sigma_r^{-1} \Sigma_e) + (\mu_r - \mu_e)^T \Sigma_r^{-1} (\mu_r - \mu_e)^T - \log \frac{det(\Sigma_e)}{det(\Sigma_r)} - k_e \},
\end{aligned}
\tag{15}
$$

Gaussian embedding for knowledge graphs: Energy function

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 Eq. 15, $tr(\cdot)$ indicates the trace and Σ_r^{-1} refers to the inverse of the covariance matrix. Similarly, $E_{neg}(h', r', t')$ represents the negative triplet energy function that measures the similarity between entity and relation Gaussian embeddings (i.e., $P_e^{'}$ and $P_r^{'}$) for the corresponding negative triplets. It follows the same definition principles as shown in Eq. 14 and Eq. 15.

Gaussian embedding for attributed graphs

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 $L \ll D$.

Gaussian embedding for attributed graphs

The main architecture of *Graph2Gauss* contains four main elements:

- Unsupervised node representation learning based on a deep encoder
- Node embedding modeling as Gaussian distributions
- Energy estimation for pairs of nodes in the embedding space
- 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.

Gaussian distribution-based graph embedding framework

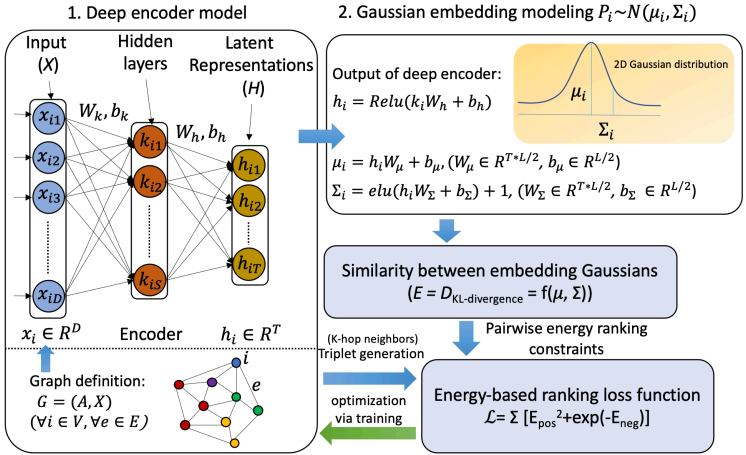


Figure 11: **Illustration of Gaussian distribution-based graph embedding framework.**

Node triplet generation

Given a node i, its k-hop neighbors can be represented as N_{ik} in Eq. 16,

$$N_{ik} = \{j \in V | i \neq j, min(sp(i,j), K) = k\}, \tag{16}$$

where $sp(i,j)$ denotes the shortest path between node i and node j (if i and j are not reachable it returns ∞); K is the maximum considered distance, 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 Eq. 17,

Node triplet generation

$$D_t = \{(i, j_k, j_l) | sp(i, j_k) < sp(i, j_l)\} \tag{17}$$

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 provides an effective way for triplet generation that can help reduce the computational complexity in a large graph.

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 Eq. 18

$$E(P_i, P_{k_1}) < E(P_i, P_{k_2}) < ... < E(P_i, P_{k_K}), \quad \forall k_1 \in N_{i1}, \forall k_2 \in N_{i2}, , ..., \forall k_1 \in N_{iK} \tag{18}$$

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.

Similarity measure for embedding Gaussians

- symmetric expected likelihood (EL)
- the Jensen-Shannon divergence (JS)
- asymmetric KL-divergence (KL)
- the $p-th$ Wasserstein distance (Wp)

Energy-based ranking loss function

"Margin-based ranking loss" is a frequently used loss function for graph embedding learning, however, the margin has to be manually selected before training. Thus, the "square-exponential loss" representing negative pair energy as an exponential term has better performance in penalizing the ranking error automatically; the specific formula is

$$\mathcal{L} = \sum_{(i,j_k,j_l)\in D_t} (E_{ij_k}^2 + exp^{-E_{ij_l}}), k < l. \tag{19}$$

Definitions of dynamic graphs

Generally, a dynamic graph ($\mathcal{G}$) can be mathematically defined in two different ways:

Definitions of dynamic graphs: Discrete snapshots

A dynamic graph can be approximated as a sequence of discrete static graph snapshots with equal time intervals ($\mathcal{G} = \{G_1, G_2, ..., G_T\}$); see the snapshot-based example in Fig. 12(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.

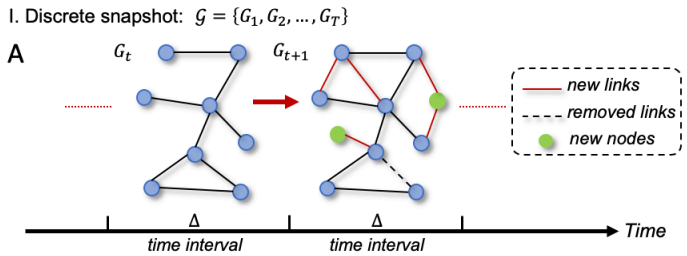
Definitions of dynamic graphs: Continuous-time graphs - Temporal edge-based

a continuous-time changing graph $G = (V, E_c, \mathcal{T})$ can be denoted by a node set V, a temporal edge set $E_c \subseteq V \times V \times \mathbb{R}^+$ and a time mapping function $\mathcal{T} : E \to \mathbb{R}^+$ ($\mathbb{R}^+$ is the time domain) for every edge, and every edge is labeled by time (see an example in Fig. 12(B)). Hence, the important temporal sequence order and flow information in the temporal graphs can be captured in a fine-grained manner. Notably, sampling valid walks from a dynamic graph has to follow the increasing time order, e.g., "$v_1 \to v_6 \to v_3$" is not a valid walk in the Fig. 12(B) since edge (v_6, v_3) is in the past of edge (v_1, v_6).

Definitions of dynamic graphs: Continuous-time graphs - Temporal edge-based

another alternative continuous time graph representation is “link stream”, and Fig. 12(C) shows the corresponding link stream representation for the dynamic graph in Fig. 12(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.

Definitions of dynamic graphs



Problem settings for dynamic graph embedding

Based on the snapshot definition above, dynamic graph embedding can be defined as a time-series of mappings ($\mathcal{F} = \{f_1, f_2, ..., f_T\}$) corresponding to different snapshots [**?**], 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 [**?**]).

Key challenges

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 Eq. 20)

$$\begin{aligned} &\mathcal{K}_s(\mathcal{F}) = \max_{\tau,\tau'} |S_r(F;\tau) - S_r(F;\tau')| \\ &\text{where} \quad S_r(F;t) = \frac{||f_{t+1}(V_t) - f_t(V_t)||_F}{||f_t(V_t)||_F} \Big/ \frac{||S_{t+1}(V_t) - S_t(V_t)||_F}{||S_t(V_t)||_F}, \end{aligned} \tag{20}$$

Key challenges

How to effectively and efficiently update the node embedding with topological graph evolution over time.

Key challenges

How to scale it to large-scale dynamic network embedding.

Approaches

- Matrix factorization-based
- SkipGram-based
- Autoencoder (AE)-based
- Graph convolutional networks (GCN)-based
- Generative adversarial network (GAN)-based

Applications

Social network applications

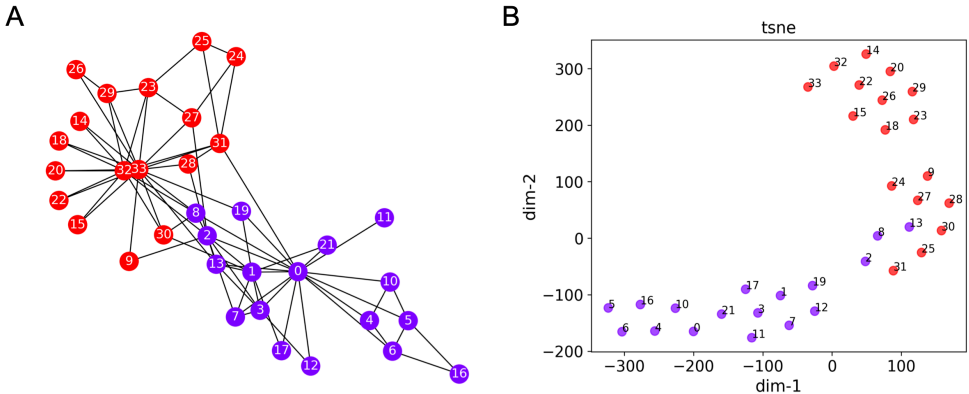


Figure 13: **Example of applying graph embedding method for community detection in the karate club dataset.**

Citation network applications

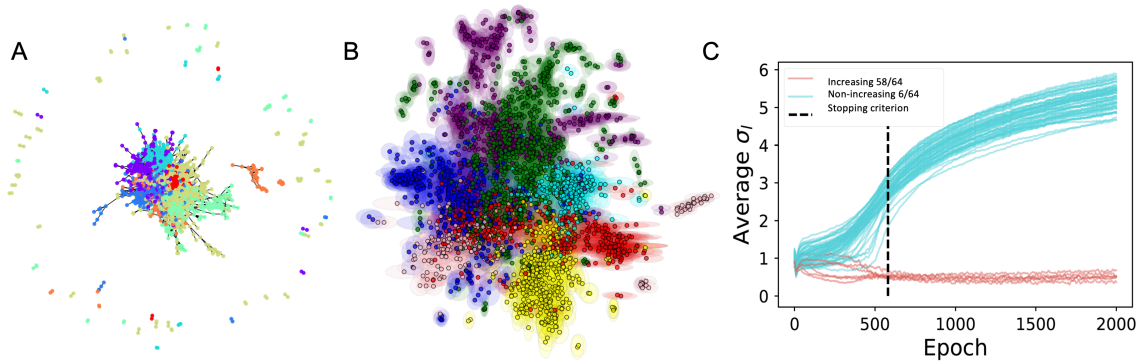


Figure 14: **Example of applying stochastic graph embedding method for node classification in CORA-ML dataset.**

Brain network applications

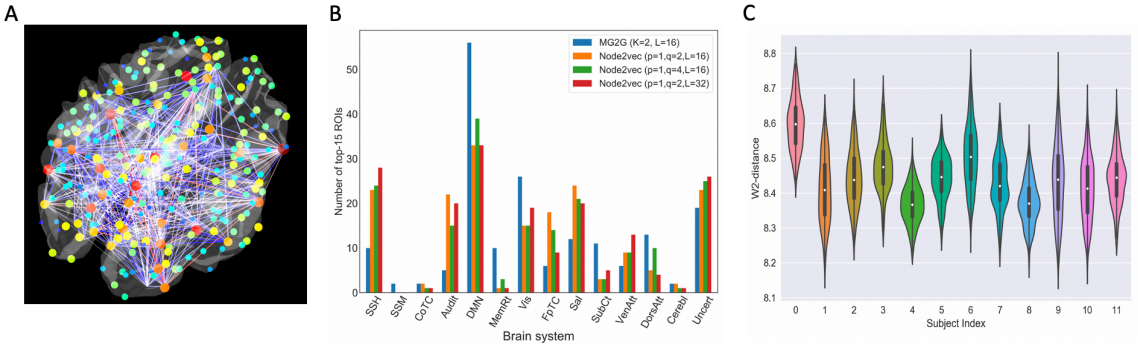


Figure 15: **Example of applying stochastic (MG2G) and deterministic (node2vec) graph embedding methods for cognitive training effects using fMRI brain networks**.

Genomic network applications

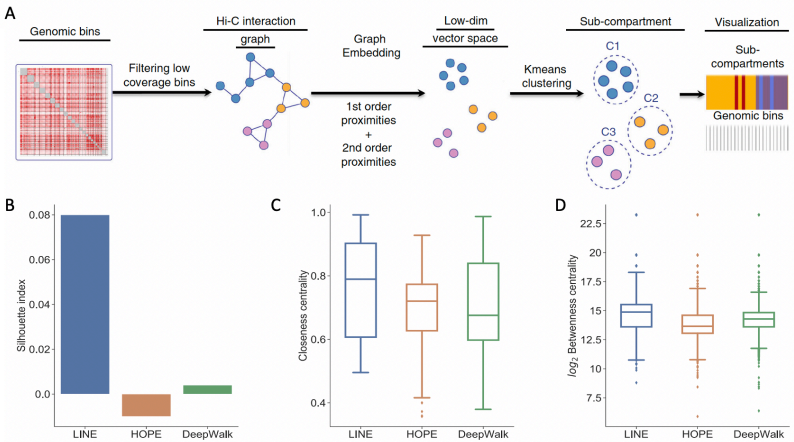


Figure 16: **Example of applying the deterministic LINE graph embedding method for sub-compartment identification using Hi-C chromatin interaction data**.