A Survey of Graph Embedding

Yi Wu
Facultad de Informática de Barcelona
Universitat Politècnica De Catalunya
yi.wu@estudiantat.upc.edu

Zhiyang Guo
Facultad de Informática de Barcelona
Universitat Politècnica De Catalunya
guoxiaoxiaoyu@gmail.com

1 Introduction

Graph embedding is a structure-preserving dimensionality reduction problem. Input the graph, output low-dimension representation in the latent space(usually in vectors format), and feed it into downstream graph analytics tasks.

1.1 Graph Embedding Inputs

Graph embedding input can be divided into four categories: homogeneous graph, heterogeneous graph, graph with auxiliary information and constructed graph. Each category can be further classified as directed/undirected graphs, or weighed/unweighted graphs, or static/dynamic graphs. Intuitively, two nodes that share a higher-weight edge should be embedded closer.

Homogeneous graph is a graph that all nodes belong to a single type, so do all edges.

Heterogeneous graph, on contrary, allows the various types of nodes and edges. For example, knowledge graph. In a film related knowledge graph constructed from Freebase (1), the types of entities can be "director", "actor", "film", etc. The types of relations can be "produce", "direct", "act in".

The third type is graphs with auxiliary information. In addition to the structural relations of node, it contains labels, affiliated attributes, information propagation, etc. One example can be Neo4J property graph.

The last type is graphs constructed from non-relational data. Imagine to convert a picture into a graph(2), where pixels are treated as nodes and the co-occurrence of spatial relations between pixels are as edges.

Insight: For heterogeneous graphs, like knowledge graph, the distributions of different types nodes/edges can be skewed, how to deal the imbalance can be a challenge. For graphs with auxiliary information, like the property graph, the embedding shall reflect both topological and unstructured auxiliary information sources.

1.2 Graph Embedding Outputs

The output of embedding can be either a deterministic low-dimension vector, or an uncertain multivariate Gaussian distribution, corresponding to Vector Point-Based embedding methods and Gaussian Distribution-Based embedding methods(3). Gaussian distribution-based graph embedding enables more effective in-corporation of useful unstructured attribute information associated with each node.

Based on the output granularity, graph embedding output can be divided into four categories, node embedding, edge embedding, hybrid embedding and whole-graph embedding. The choice to implement which level of granularity is task-driven.

Node embedding is used for node clustering, node classification.

Edge embedding is common in link prediction. For instance in knowledge graph embedding model TransE(4), $< h, r, t >$ is a triplet, with head/tail entity nodes $< h, t >$ and their relationship $< r >$ both embedded into same vector space. The subtraction of head/tail entity embedded vectors is exactly the relationship embedded vector. By given two components of a triplet, the third component can be predicted.

Hybrid embedding, like sub-graph embedding and community embedding, is represent a higher-level structure with a low dimension vector/distribution. The community embedding requires the nodes embedding inside a community is similar to its community embedding, while two communities embedding should be disparate.

Whole-graph embedding provides a straightforward and efficient solution for graph classification. It's usually used in small graphs, such as proteins, molecules, etc. Two similar graphs should be embedded closer.

1.3 From Embedding Inputs to Outputs

Several questions need to be addressed when embedding inputs into outputs. Take node embedding as an example, since it is the most common task.

Nodes that are "close" in the graph are embedded to have "similar" vector representations. The objective function is clear, that is to minimize the difference of original graph's node closeness and the embedded graph's vector similarity.

There lays three questions, 1)how to define the closeness of two nodes in original graph(see Chapter 2.4 on the following page), 2)how to define the similarity of two vectors in embedded space(see Chapter 2.3 on the next page), and 3)how to convert them. Concerning the first question, we shall consider as well to which extent to include the node context information, for example, 1-hop neighbors, 2-hop neighbors, or even the whole graph structure. Concerning the third question, the way to mapping high-dimension node context to low-dimension vector is via an Encoder.

2 Mathematics Preliminaries

In this chapter, the mathematics representation of the original graph and the embedding result, the measurement of original graph's 'closeness' and embedding space's 'similarity' are provided.

2.1 Original Graph

We consider a graph $G(V, E)$ as a mathematical data structure that contains a node (or vertex) set $V = \{v_1, v_2, v_3, ..., v_n\}$ and an edge (or link) set E. The edge set, one edge e_{ij} describes the connection between two adjacent nodes vi and v_j, and hence e_{ij} can be represented as (v_i, v_j).

2.2 Embedding Space

The task of learning its node/edge/hybrid/graph embedding can be mathematically formulated as finding a projection ϕ such that, take node embedding(e.g., from $|V|$ to d dimensions) as an example,

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

is either Point Vector-based representation or Gaussian Distribution-based representation. Edge/hybrid/graph embeddings are in similar format.

2.3 Similarity Measurement in Embedding Space

Representations	Similarity Measures
Point Vector	• Dot Product : $z_j^T z_i$ • Cosine similarity: $\frac{z_i \cdot z_j}{\|\|z_i\|\| \times \|\|z_j\|\|}$ • Euclidean distance: $\|\|z_i - z_j\|\|$
Gaussian Distribution	• Expected likelihood: $EL(P_i, P_j)$ • KL-divergence: $D_{KL}(P_j\|\|P_i))$ • Wasserstein distance: $W(P_i, P_j)^2$

When embedded as vectors, the similarity can be measured by dot product, cosine, or Euclidean distance. When embedded as Gaussian distributions, the similarity can be measured by expected likelihood, KL-divergence and Wasserstein distance.

2.4 Closeness Measurement in Original Graph

2.4.1 Adjacency Matrix

Adjacency Matrix(5) is the most intuitive one. If an edge exists between two nodes v_i and v_j, the edge weight is the value on the corresponding position in the adjacency matrix w_{ij}.

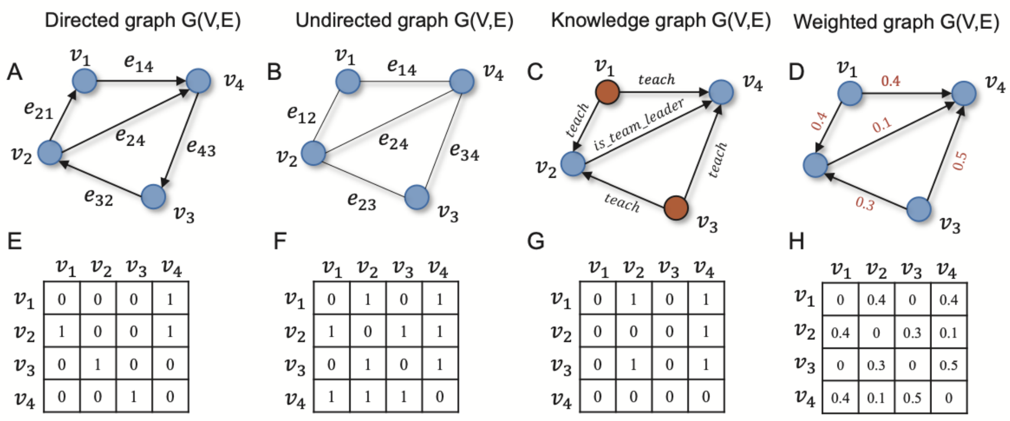

Figure 1: Adjacency Matrix

2.4.2 Proximity Graph and Proximity Matrix

Compared to Adjacency Matrix, which is a matrix derived from the original graph, a proximity graph is a new graph generated from the original graph, that only contains similar nodes and the edges between them.

The way to define the similar nodes can be very flexible. For example, we can say in Figure.2, s_1 and a_1 are similar because they are linked by one edge. And we can also define that s_2 and t_1 are similar because they share common neighbors, as illustrated in Shared Nearest Neighbor algorithm(SNN)(14), which defines proximity between two vertices in terms of k common neighbors (i.e., directly connected vertices). Similarly, algorithm kNN preserves only the nearest k neighbors in proximity graph.

Further, proximity measures take into account not only the adjacent, but also long distance connections. The Neumann Kernel uses a tunable decay parameter γ to control how much weight is given to the more distant connections, which can produce results based entirely on the immediate neighbors or results that take the full structure of the graph into consideration(14).

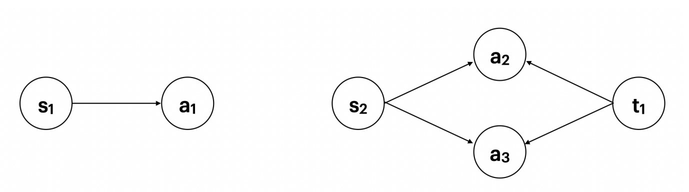

Figure 2: Smilar Nodes

Based on Proximity Graph, the nodes' Proximity Matrix can be generated as an input of the embedding problem.

3 Graph Embedding Methods

We focus on vector-based graph embedding methods. And in this chapter, we talk about three mainstream embedding methods, that is, matrix factorization method, deep learning method, and random walk method.

3.1 Matrix Factorization Method

There are two types of matrix factorization based graph embedding. One is to factorize graph Laplacian eigenmaps, and the other is to directly factorize the node proximity matrix.

3.1.1 Graph Laplacian Eigenmaps

Theory: Let $z = (z_1, ..., z_n)$ represent the d dimension embedding of the n nodes. We then formulate the following problem,

$$\min_{z \in \mathbb{R}^d} \sum_{i,j} \frac{1}{2} w_{ij} (z_i - z_j)^2$$

where w_{ij} is the similarity weights between node v_i and v_j in original graph. The higher the w_{ij} is, the smaller the $(z_i - z_j)^2$ should be. Further scale the embedding vector as $\sum_i (z_i)^2 = 1$.

Considers the Laplacia matrix of G, defined as,

$$L = D - W$$

where D is the diagonal matrix whose entries are the sum weights of each node. Then the objective function can be written as,

$$\min_{z \in \mathbb{R}^d} \sum_{i,j} \frac{1}{2} w_{ij} (z_i - z_j)^2 = \min_{z \in \mathbb{R}^d} z^T L z$$

Finally the optimal z is given by the second smallest eigenvector of L.

Insight: The differences of existing studies mainly lie in how they calculate the pairwise node similarity w_{ij}.

The initial study MDS(21) adopted Euclidean distance of two nodes v_i and v_j as w_{ij}. Later, it's proposed to construct a proximity graph first, which considers mainly local structure. As in Epsilon-ball approach, only nodes whose distance $||v_i - v_j|| <= \epsilon$ are considered as nearby nodes. In the kNN approach, only k nearest neighbors are reserved in the proximity graph(6)(7)(8)(9), where nodes distance is the value of w_{ij}.

Recent Works: Some more advanced models are designed recently. For example, AgLPP(10) introduces an anchor graph to significantly improve the efficiency of earlier matrix factorization model LPP. LGRM(11) learns a local regression model to grasp the graph structure and a global regression term for out-of- sample data extrapolation. LSE(12) uses local spline regression to preserve global geometry.

3.1.2 Node Proximity Matrix Factorization

Theory: Proximity measures are applicable to weighted structures and could take into account not only the shortest, but also long-distance connections(14). Given the node proximity matrix W(as mentioned in Chapter 2.4.2 on page 3), the objective is,

$$min||W - ZZ^{cT}||$$

where $Z \in R^{|V| \times d}$ is the node embedding, and $Z^c \in R^{|V| \times d}$is the embedding for the context nodes. One popular solution is to apply SVD (Singular Value Decomposition) on W. Formally,

$$W = \sum_{i=1}^{|V|} \sigma_i \mu_i \mu_i^{cT} \approx \sum_{i=1}^{d} \sigma_i \mu_i \mu_i^{cT}$$

where $\{\sigma_1, \sigma_2, ..., \sigma_{|V|}\}$ are the singular values sorted in descending order, μ_i and μ_i^c are singular vectors of σ_i. The optimal embedding is obtained using the largest d singular values and corresponding singular vectors as follows,

$$Z = [\sqrt{\sigma_1}\mu_1, ..., \sqrt{\sigma_d}\mu_d],$$
$$Z^c = [\sqrt{\sigma_1}\mu_1^c, ..., \sqrt{\sigma_d}\mu_d^c].$$

Depending on whether the asymmetric property is preserved or not, the embedding of node v_i is either $z_i = Z_i$ or the concatenation, i.e., $z_i = [Z_i, Z_i^c]$.

Insight: To explicitly derive the proximity matrix, the cost and is too expensive for large graphs. In the case in NetMF(13), it typically takes $\Theta(n^2)$.

Recent Works: To avoid the $\Theta(n^2)$ running cost, HOPE(15), AROPE(16), and NRP(17) are proposed to derive the embedding without explicitly computing the proximity matrix.

For instance, instead of computing all-pair proximity scores and then decomposing the proximity matrix, NRP turns to do SVD on the adjacency matrix, which is sparse for most real-life graphs, reducing the embedding computational cost. Another solution to avoid the $\Theta(n^2)$ cost is to calculate a sparsified proximity matrix S. The representative is STRAP(18), which imposes a threshold δ and returns at most $\Theta(\frac{n}{\delta})$ proximity.

Since the second solution explicitly derives the proximity matrix, it allows to take non-linear operations on the proximity matrix, improving the representation powers.

DANE(21) captures the changes of dynamic structures with the adjacency matrix and models the changes of attributes with the attribute matrix, which only considers the first-order proximity. DHPE(22) preserves high-order proximity between nodes

Sum Up: Matrix Factorization method suffers from high computation cost($\Theta(n^2)$). Adjacency matrix-based algorithms only consider pair-wise local relationships, neglecting the global structures. The downstream analysis tasks which are built on Adjacency matrix embedding method usually has overfitting problem.

3.2 Deep Learning Method

So far we have talked about the shallow encoder, which maps a node to a low-dimension vector only through one encoding layer. It means that the number of encoding parameters is proportional to the number of nodes. And it's transductive. Graph neural network has allowed the multiple layers of non-linear transformations of graph structure, so called deep encoders. The parameters are shared. Also it's inductive.

Graphs are challenging in deep learning, since it's shape is not fixed and it's limitless, compared with image input which is a fixed-size matrix and text input which is a line graph with fixed-length sliding window. One way to cope with graphs is through Graph Convolutional Network(GCN)(23), which aggregate the neighbor nodes information like a convolution kernel, as shown in Figure.3. Node A aggregates both itself and its neighbors B,C,E's information. Node B contains as well both itself and neighbors' information. Such multi-layer aggregations are propagating, and each node contains both local and global information.

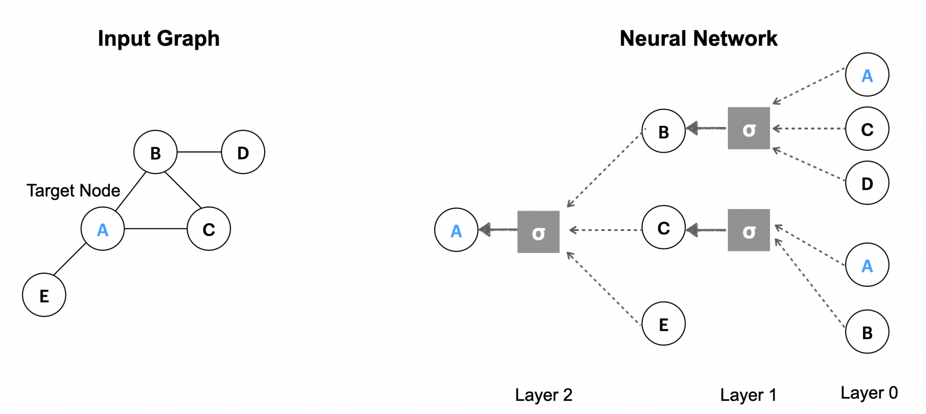

Figure 3: Graph Convolutional Network

Theory: In GCN algorithm, let's suppose in initial layer h^0, we have node v features x_v as input, $h_v^0 = x_v$. And in last layer, we get node embedding $z_v = h_v^L$. At the middle layer l, the embedding is like,

$$h_v^{l+1} = \sigma(W_l \sum_{\mu \in N(v)} \frac{h_\mu^l}{|N(v)|} + B_l h_v^l), \forall l \in \{0, ..., L-1\}$$

where σ is activation function(usually ReLU, Sigmoid or Parametric ReLU); and inside it, the first part is the aggregation of all neighbors $N(v)$ information from last iteration, the second part is the node itself's information from last iteration. W_l and B_l are corresponding trainable weight matrix to learn.

Let matrix A be the adjacency matrix, matrix D be the degrees matrix, and $H^l = [h_1^l, ..., h_{|V|}^l]^T$, the formula can be rewritten as,

$$H^{l+1} = \sigma(\hat{A} H^l W_l^T + H^l B_l^T)$$

where $\hat{A} = D^{-1}A$. Matrix format is computation effective via Gradient Descent. We want to minimize the loss in supervised learning of,

$$\mathcal{L}(y, f(z_v))$$

where $\mathcal{L}$ can be L_2 norm if y is numerical and cross entropy if y is categorical; and in unsupervised learning of,

$$\mathcal{L} = \sum_{z_u, z_v} CE(y_{u,v}, DEC(z_u, z_v))$$

where $y_{u,v} = 1$ if nodes u, v are similar, and CE is the cross entropy, and DEC is the decoder, which can be any embedding vector similarity measurement(see Chapter 2.3 on page 3).

Recent Works: GraphSage(24), compared with GCN, rather than sum the node vector with neighbors' inside the activation function, it concatenates them. Also, for the aggregation of all neighbors' information, it offers choices of mean, pool, even LSTM. Note that LSTM is not order invariant, a random selection of neighbors is applied.

Graph Attention Network(GAT)(25) introduced the attention mechanism. It assigns the different weights of different neighbors. Multi-head attention(26) further replicates the attention operations at a given layer for R times, and output the concatenating or adding result, which stabilize the learning process. TemporalGAT(31) has applied attention algorithm on dynamic graphs.

Other than homogeneous graph algorithms, some deep learning methods(e.g., HNE(28) TransE(29), ProxEmbed(30)) are applied in heterogeneous graphs as well.

3.3 Random Walk Method

The general concept of Random Walk Method is that, starting from a node v_i, as shown in Figure.4, within a k-step random walk, the sum of the probability that v_j co-occurs $(\sum p(v_j|v_i))$, should be maximized.

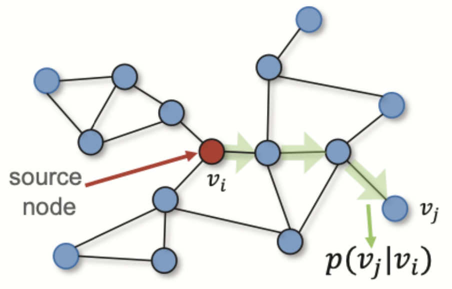

Figure 4: Graph Random Walk

Random Walk Method is efficient, because it only considers the nodes that show up in walk paths, drastically reducing the computation cost in traditional deep learning embedding method.

Theory: Now let u be the starting point, and $N_R(u)$ notates the multiset of nodes visited on random walks. Then the loss function is,

$$\mathcal{L} = \sum_{\mu \in V} \sum_{v \in N_R(\mu)} -log(P(v|z_u))$$

where

$$P(v|z_u) = \frac{exp(z_u^T z_v)}{\sum_{n \in V} exp(z_u^T z_n)}$$

It means that, iterate all starting points u; for each given u, maximize the log possibility of visited nodes. And the probability could be alternatively represented as their embedding vectors' doc product $z_u \cdot z_v^T$, normalized by the Softmax function, in order to learn the best embedding representation vectors.

However the denominator of the normalization $P(v|z_u)$ shows to sum all the nodes. In order to simplify it, two methods are introduced to approximate the problem.

First is the Negative Sampling method(19),

$$log(\frac{exp(z_u^T z_v)}{\sum_{n \in V} exp(z_u^T z_n)}) \approx log(\sigma(z_u^T z_v)) - \sum_{i=1}^{k} log(\sigma(z_u^T z_{n_i})), n_i \sim P_v$$

where k negative samples are selected from a noise distribution P_v, reducing the computation cost from $\Theta(|V|^2)$ to $\Theta(k|V|)$. According to experience, parameter k is usually set in range 15 to 20.

Second way is the Hierarchical Softmax(19). A binary tree is constructed, and only the path from the root to the corresponding leaf needs to be evaluated,reducing the computation cost from $\Theta(|V|^2)$ to $\Theta(|V|log(|V|)$.

Recent Works: DeepWalk(31) applies the above algorithm via deep learning encoding model SkipGram. But its random walk path is fixed-length and unbiased. Node2vec(32) improves it and enables biased parameters to explore either breath-first or depth-first random walks to bias towards local or global view, as shown in Figure.5.

Also the choices over encoders can vary from SkipGram with hierarchical softmax(e.g., DeepWalk), to SkipGram with negative sampling(e.g., Node2vec), to LSTM(e.g., ProxEmbed(30) HSNL(33), RMNL(34)), and to GRU(e.g., DeepCas(35)).

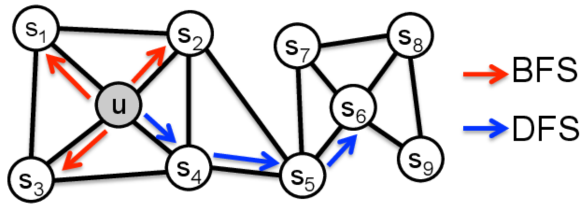

Figure 5: Node2vec

4 Applications

Graph embedding is beneficial for various graph analysis applications because it can handle vector representations efficiently in both time and space. This section classifies applications that support graph embeddings into node related, edge related and graph related.

4.1 Node Related Applications

4.1.1 Node Classification

Node classification is the assignment of class labels to each node in the graph based on the rules learned from the labeled nodes. Intuitively, "similar" nodes have the same label. This is one of the most common applications described in the Graph Embedding literature. Generally, each node is embedded as a low-dimensional vector. Node classification is performed by applying a classifier to a set of labeled node embeddings for training.Examples of classifiers include SVM logistic regression and k-nearest neighbor classification. Second, if an unlabeled node is embedded, the trained classifier can predict its class label. Relative to the above sequential processing of node embedding and then node classification, some other works have designed a unified framework that jointly optimizes graph embedding and node classification, classifies each node, and learns a classification-specific expression.

4.1.2 Node Clustering

Node clustering aims to group similar nodes together, so that nodes in the same group are more similar than nodes in other groups. It can be used as an unsupervised algorithm when node labels are not available. After representing nodes as vectors, traditional clustering algorithms can be applied to node embeddings.

4.1.3 Node Recommendation

The task of node recommendation is to recommend the top K nodes of interest to a specific node according to certain criteria such as similarity. In real-world scenarios, recommendation nodes are of different types, such as research topics, items for customers, curated images of web users, friends of social network users, and queried documents. It is also popular in community-based QA. Given a question, it predicts the relative ranking of users or answers. Proximity search sorts certain types of nodes (e.g., "users") into specific query nodes (e.g., "bob") and proximity categories (e.g., "graduate"). For example, the ranking of graduate users. Bob's. Some works focus on cross-modal search, such as keyword-based image/video search.

4.2 Edge Related Applications

The following is an edge-related application that contains edges or pairs of nodes.

4.2.1 Link Prediction

Graph embeddings are designed to represent graphs with low-dimensional vectors, but interestingly, their output vectors are also useful for inferring the graph structure. In practice, graphs are often incomplete. For example, a social network may lack a friendship link between two users who actually know each other. In graph embedding, low-dimensional vectors are expected to maintain network proximity in different orders (e.g., DeepWalk(31), LINE(36), etc.) and structural similarity at different scales (e.g., GCN, struc2vec(37), etc.). Therefore, these vectors can be used to encode a wealth of information about the network structure and predict missing links in incomplete graphs. Most attempts at graph-embedded link prediction are made with homogeneous graphs. For example, to predict friendship between two users. Relatively little work has been done on graph embeddings dealing with heterogeneous graph link prediction. For example, in a heterogeneous social graph, ProxEmbed(30) attempts to predict missing links for a specific semantic type (e.g. schoolmate) between two users based on embedding connection paths into the graph. D2AGE(38) solves the same problem by embedding two users connected by a directed acyclic graph structure.

4.2.2 Triple Classification

Triplet classification is a specific application of the knowledge graph. This is intended to classify whether the invisible triple <h, r, t> is correct, that is, whether the relationship between h and t is r.

4.3 Graph Related Applications

4.3.1 Graph Classification

Graph classification assigns a class label to the entire graph. This is important when the graph is a unit of data. For example, each diagram is a chemical compound, organic molecule, or protein structure. In most cases, the whole graph embedding is used to compute graph-level similarity. Recently, some work has started to match node embeddings for the similarity of graphs. Each graph is represented as a set of nodes with embedded vectors. Graphs are compared based on two sets of node embeddings, decompose the graph into a set of substructures, then embed each substructure as a vector, and compare the graphs by substructure similarity.

4.3.2 Visualization

Graph visualization produces graph visualization in low-dimensional space. Usually for visualization purposes, all nodes are embedded as 2D vectors and then different graphs are drawn in 2D space. The color indicates the node category. It clearly shows whether nodes belonging to the same category are embedded close to each other.

5 Summary and Discussion

This study provides a comprehensive review of the graph embedding literature. We formally define the problem of embedding graphs and introduce some basic mathematics foundations. Then we introduce three main stream graph embedding methods, which are matrix factorization, deep learning without random walks, and random walks. The method review is conducted in aspects of theory, key problems, insights and a comparison of recent works. Finally we represent applications of the algorithms along with concrete examples. However, such topic is too broad, and we look forward to further investigate in dynamic graph embedding methods and knowledge graph embedding methods in the future.

References

[1] Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J. (2008, June). Freebase: a collaboratively created graph database for structuring human knowledge. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (pp. 1247-1250). 1.1

[2] Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., Bronstein, M. M. (2017). Geometric deep learning on graphs and manifolds using mixture model cnns. In Proceed-

ings of the IEEE conference on computer vision and pattern recognition (pp. 5115-5124). 1.1

[3] Xu, M. (2021). Understanding graph embedding methods and their applications. SIAM Review, 63(4), 825-853. 1.2

[4] Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. Advances in neural information processing systems, 26. 1.2

[5] G. Nikolentzos, P. Meladianos, and M. Vazirgiannis, "Matching node embeddings for graph similarity," in AAAI, 2017, pp. 2429– 2435. 2.4.1

[6] W. N. A. Jr. and T. D. Morley, "Eigenvalues of the laplacian of a graph," Linear and Multilinear Algebra, vol. 18, no. 2, pp. 141–145, 1985. 3.1.1

[7] X. He and P. Niyogi, "Locality preserving projections," in NIPS, 2003, pp. 153–160. 3.1.1

[8] S. T. Roweis and L. K. Saul, "Nonlinear Dimensionality Reduc- tion by Locally Linear Embedding," Science, vol. 290, no. 5500, pp. 2323–2326, 2000. 3.1.1

[9] M. Balasubramanian and E. L. Schwartz, "The isomap algorithm and topological stability," Science, vol. 295, no. 5552, pp. 7–7, 2002. 3.1.1

[10] R. Jiang, W. Fu, L. Wen, S. Hao, and R. Hong, "Dimensionality reduction on anchorgraph with an efficient locality preserving projection," Neurocomputing, vol. 187, pp. 109–118, 2016. 3.1.1

[11] 3.1.1
Y. Yang, F. Nie, S. Xiang, Y. Zhuang, and W. Wang, "Local and global regressive mapping for manifold learning with out-of- sample extrapolation," in AAAI, 2010.

[12] 3.1.1
S. Xiang, F. Nie, C. Zhang, and C. Zhang, "Nonlinear dimension- ality reduction with local spline embedding," IEEE Trans. Knowl. Data Eng., vol. 21, no. 9, pp. 1285–1298, 2009.

[13] 3.1.2
JiezhongQiu,YuxiaoDong,HaoMa,JianLi,KuansanWang,andJieTang.2018. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and Node2vec. In WSDM. 459–467.

[14] Chebotarev, P., Shamis, E. (2006). On proximity measures for graph vertices. arXiv preprint math/0602073. 2.4.2, 2.4.2, 3.1.2

[15] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asym- metric Transitivity Preserving Graph Embedding. In SIGKDD. 1105–1114. 3.1.2

[16] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. 2018. Arbitrary-Order Proximity Preserved Network Embedding. In SIGKDD. 3.1.2
2778–2786.

[17] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S. Bhowmick. 2020. Homogeneous Network Embedding for Massive Graphs via Reweighted Personalized PageRank. PVLDB 13, 5 (2020), 670–683. 3.1.2

[18] Yuan Yin and Zhewei Wei. 2019. Scalable Graph Embeddings via Sparse Trans- pose Proximities. In SIGKDD. 1429–1437. 3.1.2

[19] 3.3
Samatova, N. F., Hendrix, W., Jenkins, J., Padmanabhan, K., Chakraborty, A. (Eds.). (2013). Practical graph mining with R. CRC Press.

[20] T. Hofmann and J. M. Buhmann, “Multidimensional scaling and data clustering,” in NIPS, 1994, pp. 459–466.3.1.1, 3.1.2

[21] 3.1.1, 3.1.2
LI J D, DANI H, HU X, et al. Attributed network embe- dding for learning in a dynamic environment[C]//Proceedings of the 2017 ACM on Conference on Information and Know- ledge Management, Singapore, Nov 6-10, 2017. New York: ACM, 2017: 387-396.

[22] 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, 2018.3.1.2

[23] KIPF T N, WELLING M. Semi- supervised classification with graph convolutional networks[J]. arXiv:1609.02907, 2016.3.2

[24] HAMILTON W L, YING R, LESKOVEC J. Inductive representation learning on large graphs[J]. arXiv:1706.02216, 2017.3.2

[25] VELIČKOVIĆ P, CUCURULL G, CASANOVA A, et al. Graph attention networks[J]. arXiv:1710.10903, 2017.3.2

[26] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.3.2

[27] FATHY A, LI K. TemporalGAT: attention-based dynamic graph representation learning[C]//LNCS 12084: Proceedings of the 24th Pacific-Asia Conference on Knowledge Dis- covery and Data Mining, Singapore, May 11-14, 2020. Cham: Springer, 2020: 413-423.

[28] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang, “Heterogeneous network embedding via deep architec- tures,” in KDD, 2015, pp. 119–128.3.2, 3.3, 4.2.1

[29] A. Bordes, N. Usunier, A. Garc ıa-Dura n, J. Weston, and O. Yakhnenko, “Translating embeddings for modeling multi- relational data,” in NIPS, 2013, pp. 2787–2795.3.2

[30] Z. Liu, V. W. Zheng, Z. Zhao, F. Zhu, K. C. Chang, M. Wu, and J. Ying, “Semantic proximity search on heterogeneous graph by proximity embedding,” in AAAI, 2017, pp. 154–160.3.2

[31] B. Perozzi, R. Al-Rfou, and S. Skiena, DeepWalk: Online learning of social representations, in Proc. of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2014, pp. 701-710.3.2, 3.3, 4.2.1

[32] A. Grover and J. Leskovec, node2vec: Scalable feature learning for networks, in Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 855-864.3.2, 3.3, 4.2.1

[33] H. Fang, F. Wu, Z. Zhao, X. Duan, Y. Zhuang, and M. Ester, “Community-based question answering via heterogeneous social network learning,” in AAAI, 2016, pp. 122–128.3.3

[34] Z. Zhao, Q. Yang, D. Cai, X. He, and Y. Zhuang, “Expert finding for community-based question answering via ranking metric network learning,” in IJCAI, 2016, pp. 3000–3006. 3.3

[35] C. Li, J. Ma, X. Guo, and Q. Mei, “Deepcas: An end-to-end predictor of information cascades,” in WWW, 2017, pp. 577–586.3.3

[36] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q. (2015, May). Line: Large-scale information network embedding. In Proceedings of the 24th international conference on world wide web (pp. 1067-1077).3.3

[37] Ribeiro, L. F., Saverese, P. H., Figueiredo, D. R. (2017, August). struc2vec: Learning node representations from structural identity. In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 385-394). 4.2.1

[38] Liu, Z., Zheng, V., Zhao, Z., Zhu, F., Chang, K., Wu, M., Ying, J. (2018, April). Distance-aware dag embedding for proximity search on heterogeneous graphs. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 32, No. 1). 4.2.1

4.2.1