Knowledge-Based Systems 151 (2018) 78–94

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Graph embedding techniques, applications, and performance: A survey

Palash Goyal*, Emilio Ferrara

University of Southern California, Information Sciences Institute 4676 Admiralty Way, Suite 1001 Marina del Rey, CA 90292, USA

ARTICLE INFO

Article history:
Received 10 December 2017
Revised 13 March 2018
Accepted 15 March 2018
Available online 16 March 2018

Keywords:
Graph embedding techniques
Graph embedding applications
Python graph embedding methods GEM library

ABSTRACT

Graphs, such as social networks, word co-occurrence networks, and communication networks, occur naturally in various real-world applications. Analyzing them yields insight into the structure of society, language, and different patterns of communication. Many approaches have been proposed to perform the analysis. Recently, methods which use the representation of graph nodes in vector space have gained traction from the research community. In this survey, we provide a comprehensive and structured analysis of various graph embedding techniques proposed in the literature. We first introduce the embedding task and its challenges such as scalability, choice of dimensionality, and features to be preserved, and their possible solutions. We then present three categories of approaches based on factorization methods, random walks, and deep learning, with examples of representative algorithms in each category and analysis of their performance on various tasks. We evaluate these state-of-the-art methods on a few common datasets and compare their performance against one another. Our analysis concludes by suggesting some potential applications and future directions. We finally present the open-source Python library we developed, named GEM (*Graph Embedding Methods*, available at https://github.com/palash1992/GEM), which provides all presented algorithms within a unified interface to foster and facilitate research on the topic.

1. Introduction

Graph analysis has been attracting increasing attention in the recent years due the ubiquity of networks in the real world. Graphs (a.k.a. networks) have been used to denote information in various areas including biology (Protein-Protein interaction networks) [1], social sciences (friendship networks) [2] and linguistics (word co-occurrence networks) [3]. Modeling the interactions between entities as graphs has enabled researchers to understand the various network systems in a systematic manner[4]. For example, social networks have been used for applications like friendship or content recommendation, as well as for advertisement [5]. Graph analytic tasks can be broadly abstracted into the following four categories: (a) node classification [6], (b) link prediction [5], (c) clustering [7], and (d) visualization [8]. Node classification aims at determining the label of nodes (a.k.a. vertices) based on other labeled nodes and the topology of the network. Link prediction refers to the task of predicting missing links or links that are likely to occur in the future. Clustering is used to find subsets of similar nodes and group them together; finally, visualization helps in providing insights into the structure of the network.

In the past few decades, many methods have been proposed for the tasks defined above. For node classification, there are broadly two categories of approaches — methods which use random walks to propagate the labels [9,10], and methods which extract features from nodes and apply classifiers on them [11,12]. Approaches for link prediction include similarity based methods [13,14], maximum likelihood models [15,16], and probabilistic models [17,18]. Clustering methods include attribute based models [19] and methods which directly maximize (resp., minimize) the inter-cluster (resp., intra-cluster) distances [7,20]. This survey will provide a taxonomy that captures these application domains and the existing strategies.

Typically, a model defined to solve graph-based problems either operates on the original graph adjacency matrix or on a derived vector space. Recently, the methods based on representing networks in vector space, while preserving their properties, have become widely popular [21–23]. Obtaining such an embedding is useful in the tasks defined above.[1] The embeddings are input as features to a model and the parameters are learned based on the

* Coresponding author.
E-mail address: palashgo@usc.edu (P. Goyal).

https://doi.org/10.1016/j.knosys.2018.03.022
0950-7051/

[1] The term *graph embedding* has been used in the literature in two ways: to represent an entire graph in vector space, or to represent each individual node in vector space. In this paper, we use the latter definition since such representations can be used for tasks like node classification, differently from the former representation. The former definition, embedding the entire graph, has many interesting applications including pattern recognition [101], computer vision [102] and document analysis [103]. Interested readers are referred to Yan et al. [35] for further reading.

training data. This obviates the need for complex classification models which are applied directly on the graph.

1.1. Challenges

Obtaining a vector representation of each node of a graph is inherently difficult and poses several challenges which have been driving research in this field:

(i) Choice of property: A "good" vector representation of nodes should preserve the structure of the graph and the connection between individual nodes. The first challenge is choosing the property of the graph which the embedding should preserve. Given the plethora of distance metrics and properties defined for graphs, this choice can be difficult and the performance may depend on the application.

(ii) Scalability: Most real networks are large and contain millions of nodes and edges — embedding methods should be scalable and able to process large graphs. Defining a scalable model can be challenging especially when the model is aimed to preserve global properties of the network.

(iii) Dimensionality of the embedding: Finding the optimal dimensions of the representation can be hard. For example, higher number of dimensions may increase the reconstruction precision but will have high time and space complexity. The choice can also be application-specific depending on the approach: E.g., lower number of dimensions may result in better link prediction accuracy if the chosen model only captures local connections between nodes.

1.2. Our contribution

This survey provides a three-pronged contribution:

(1) We propose a taxonomy of approaches to graph embedding, and explain their differences. We define four different tasks, *i.e.,* application domains of graph embedding techniques. We illustrate the evolution of the topic, the challenges it faces, and future possible research directions.

(2) We provide a detailed and systematic analysis of various graph embedding models and discuss their performance on the various tasks. For each method, we analyze the properties preserved and its accuracy, through comprehensive comparative evaluation on a few common data sets and application scenarios. Furthermore, we perform hyperparameter sensitivity analysis on the methods evaluated to test their robustness and provide an understanding of the dependence of optimal hyperparameter on the task performed.

(3) To foster further research in this topic, we finally present *GEM*, the open-source Python library we developed that provides, under a unified interface, implementations of *all* graph embedding methods discussed in this survey. To the best of our knowledge, this is the first paper to survey graph embedding techniques and their applications.

To the best of our knowledge, this is one of the first papers to survey graph embedding techniques. Recent work reviewed prominent graph embedding methods and proposed similar taxonomies [104,105]. Our work, in addition, also focuses on graph embedding applications, implementations, and performance. We provide an extensive evaluation and experimental comparison of multiple techniques, and the analysis of methods with varying hyperparameter settings. A final major differentiator of our work is the open-source release of the Python GEM library.

1.3. Organization of the survey

The survey is organized as follows. In Section 2, we provide the definitions required to understand the problem and models discussed next. Section 3 proposes a taxonomy of graph embedding approaches and provides a description of representative algorithms in each category. The list of applications for which researchers have used the representation learning approach for graphs is provided in Section 4. We then describe our experimental setup (Section 5) and evaluate the different models (Section 6). Section 7 introduces our Python library for graph embedding methods. Finally, in Section 8 we draw our conclusions and discuss potential applications and future research direction.

2. Definitions and preliminaries

We represent the set $\{1, \cdots, n\}$ by $[n]$ in the rest of the paper. We start by formally defining several preliminaries which have been defined similar to Wang et al. [23].

Definition 1 (Graph)**.** A graph $G(V, E)$ is a collection of $V = \{v_1, \cdots, v_n\}$ vertices (a.k.a. nodes) and $E = \{e_{ij}\}_{i,j=1}^{n}$ edges. The adjacency matrix S of graph G contains non-negative weights associated with each edge: $s_{ij} \geq 0$. If v_i and v_j are not connected to each other, then $s_{ij} = 0$. For undirected weighted graphs, $s_{ij} = s_{ji} \;\; \forall i, j \in [n]$.

The edge weight s_{ij} is generally treated as a measure of similarity between the nodes v_i and v_j. The higher the edge weight, the more similar the two nodes are expected to be.

Definition 2 (First-order proximity)**.** Edge weights s_{ij} are also called first-order proximities between nodes v_i and v_j, since they are the first and foremost measures of similarity between two nodes.

We can similarly define higher-order proximities between nodes. For instance,

Definition 3 (Second-order proximity)**.** The second-order proximity between a pair of nodes describes the proximity of the pair's neighborhood structure. Let $\boldsymbol{s_i} = [s_{i1}, \cdots, s_{in}]$ denote the first-order proximity between v_i and other nodes. Then, second-order proximity between v_i and v_j is determined by the similarity of $\boldsymbol{s_i}$ and $\boldsymbol{s_j}$.

Second-order proximity compares the neighborhood of two nodes and treats them as similar if they have a similar neighborhood. It is possible to define higher-order proximities using other metrics, e.g. Katz Index, Rooted PageRank, Common Neighbors, Adamic Adar, etc. (for detailed definitions, omitted here in the interest of space, see Ou et al. [24]). Next, we define a graph embedding:

Definition 4 (Graph embedding)**.** Given a graph $G = (V, E)$, a graph embedding is a mapping $f : v_i \rightarrow \boldsymbol{y_i} \in \mathbb{R}^d \;\; \forall i \in [n]$ such that $d \ll |V|$ and the function f preserves some proximity measure defined on graph G.

An embedding therefore maps each node to a low-dimensional feature vector and tries to preserve the connection strengths between vertices. For instance, an embedding preserving first-order proximity might be obtained by minimizing $\sum_{i,j} s_{ij} \|\boldsymbol{y_i} - \boldsymbol{y_j}\|_2^2$. Let two node pairs (v_i, v_j) and (v_i, v_k) be associated with connections strengths such that $s_{ij} > s_{ik}$. In this case, v_i and v_j will be mapped to points in the embedding space that will be closer each other than the mapping of v_i and v_k.

3. Algorithmic approaches: a taxonomy

In the past decade, there has been a lot of research in the field of graph embedding, with a focus on designing new embedding algorithms. More recently, researchers pushed forward scalable embedding algorithms that can be applied on graphs with millions

Table 1
List of graph embedding approaches.

Category	Year	Published	Method	Time Complexity	Properties preserved
Factorization	2000	Science[26]	LLE	$O(\|E\|d^2)$	
	2001	NIPS[25]	Laplacian Eigenmaps	$O(\|E\|d^2)$	1st order proximity
	2013	WWW[21]	Graph Factorization	$O(\|E\|d)$	
	2015	CIKM[27]	GraRep	$O(\|V\|^3)$	
	2016	KDD[24]	HOPE	$O(\|E\|d^2)$	$1-k$th order proximities
Random Walk	2014	KDD[28]	DeepWalk	$O(\|V\|d)$	
	2016	KDD[29]	*node2vec*	$O(\|V\|d)$	$1-k$th order proximities, structural equivalence
Deep Learning	2016	KDD[23]	SDNE	$O(\|V\|\|E\|)$	1st and 2nd order proximities
	2016	AAAI[30]	DNGR	$O(\|V\|^2)$	$1-k$th order proximities
	2017	ICLR[31]	GCN	$O(\|E\|d^2)$	$1-k$th order proximities
Miscellaneous	2015	WWW[22]	LINE	$O(\|E\|d)$	1st and 2nd order proximities

of nodes and edges. In the following, we provide historical context about the research progress in this domain (Section 3.1), then propose a taxonomy of graph embedding techniques (Section 3.2) covering (i) factorization methods (Section 3.3), (ii) random walk techniques (Section 3.4), (iii) deep learning (Section 3.5), and (iv) other miscellaneous strategies (Section 3.6).

3.1. Graph embedding research context and evolution

In the early 2000s, researchers developed graph embedding algorithms as part of dimensionality reduction techniques. They would construct a similarity graph for a set of n D-dimensional points based on neighborhood and then embed the nodes of the graph in a d-dimensional vector space, where $d \ll D$. The idea for embedding was to keep connected nodes closer to each other in the vector space. Laplacian Eigenmaps [25] and Locally Linear Embedding (LLE) [26] are examples of algorithms based on this rationale. However, scalability is a major issue in this approach, whose time complexity is $O(|V|^2)$.

Since 2010, research on graph embedding has shifted to obtaining scalable graph embedding techniques which leverage the sparsity of real-world networks. For example, Graph Factorization [21] uses an approximate factorization of the adjacency matrix as the embedding. LINE [22] extends this approach and attempts to preserve both first order and second proximities. HOPE [24] extends LINE to attempt preserve high-order proximity by decomposing the similarity matrix rather than adjacency matrix using a generalized Singular Value Decomposition (SVD). SDNE [23] uses autoencoders to embed graph nodes and capture highly non-linear dependencies. The new scalable approaches have a time complexity of $O(|E|)$.

3.2. A taxonomy of graph embedding methods

We propose a taxonomy of embedding approaches. We categorize the embedding methods into three broad categories: (1) Factorization based, (2) Random Walk based, and (3) Deep Learning based. Below we explain the characteristics of each of these categories and provide a summary of a few representative approaches for each category (cf. Table 1), using the notation presented in Table 2.

3.3. Factorization based methods

Factorization based algorithms represent the connections between nodes in the form of a matrix and factorize this matrix to obtain the embedding. The matrices used to represent the connections include node adjacency matrix, Laplacian matrix, node transition probability matrix, and Katz similarity matrix, among others. Approaches to factorize the representative matrix vary based on the matrix properties. If the obtained matrix is positive semidefinite, e.g. the Laplacian matrix, one can use eigenvalue decomposition. For unstructured matrices, one can use gradient descent methods to obtain the embedding in linear time.

Table 2
Summary of notation.

G	Graphical representation of the data
V	Set of vertices in the graph
E	Set of edges in the graph
d	Number of dimensions
Y	Embedding of the graph, $\|V\| \times d$
Y_i	Embedding of node v_i, $1 \times d$ (also ith row of Y)
Y_s	Source embedding of a directed graph, $\|V\| \times d$
Y_t	Target embedding of a directed graph, $\|V\| \times d$
W	Adjacency matrix of the graph, $\|V\| \times \|V\|$
D	Diagonal matrix of the degree of each vertex, $\|V\| \times \|V\|$
L	Graph Laplacian ($L = D - W$), $\|V\| \times \|V\|$
$< Y_i, Y_j >$	Inner product of Y_i and Y_j i.e. $Y_i Y_j^T$
S	Similarity matrix of the graph, $\|V\| \times \|V\|$

3.3.1. Locally Linear Embedding (LLE)

LLE [26] assumes that every node is a linear combination of its neighbors in the embedding space. If we assume that the adjacency matrix element W_{ij} of graph G represents the weight of node j in the representation of node i, we define

$$Y_i \approx \sum_j W_{ij} Y_j \qquad \forall i \in V.$$

Hence, we can obtain the embedding $Y^{N \times d}$ by minimizing

$$\phi(Y) = \sum_i |Y_i - \sum_j W_{ij} Y_j|^2,$$

To remove degenerate solutions, the variance of the embedding is constrained as $\frac{1}{N} Y^T Y = I$. To further remove translational invariance, the embedding is centered around zero: $\sum_i Y_i = 0$. The above constrained optimization problem can be reduced to an eigenvalue problem, whose solution is to take the bottom $d+1$ eigenvectors of the sparse matrix $(I-W)^T(I-W)$ and discarding the eigenvector corresponding to the smallest eigenvalue.

3.3.2. Laplacian Eigenmaps

Laplacian Eigenmaps [25] aims to keep the embedding of two nodes close when the weight W_{ij} is high. Specifically, they minimize the following objective function

$$\begin{aligned}\phi(Y) &= \frac{1}{2} \sum_{i,j} |Y_i - Y_j|^2 W_{ij} \\ &= tr(Y^T L Y),\end{aligned}$$

where L is the Laplacian of graph G. The objective function is subjected to the constraint $Y^T D Y = I$ to eliminate trivial solution. The

solution to this can be obtained by taking the eigenvectors corresponding to the d smallest eigenvalues of the normalized Laplacian, $L_{norm} = D^{-1/2}LD^{-1/2}$.

3.3.3. Cauchy graph embedding

Laplacian Eigenmaps uses a quadratic penalty function on the distance between embeddings. The objective function thus emphasizes preservation of dissimilarity between nodes more than their similarity. This may yield embeddings which do not preserve local topology, which can be defined as the equality between relative order of edge weights (W_{ij}) and inverse order of distances in the embedded space ($|Y_i - Y_j|^2$). Cauchy Graph Embedding [32] tackles this problem by replacing the quadratic function $|Y_i - Y_j|^2$ with $\frac{|Y_i - Y_j|^2}{|Y_i - Y_j|^2 + \sigma^2}$. Upon rearrangement, the objective function to be maximized becomes

$$\phi(Y) = \sum_{i,j} \frac{W_{ij}}{|Y_i - Y_j|^2 + \sigma^2},$$

with constraints $Y^TY = I$ and $\sum_i Y_i = 0$ for each i. The new objective is an inverse function of distance and thus puts emphasis on similar nodes rather than dissimilar nodes. The authors propose several variants including Gaussian, Exponential and Linear embeddings with varying relative emphasis on the distance between nodes.

3.3.4. Structure Preserving Embedding (SPE)

Structure Preserving Embedding [33] is another approach which extends Laplacian Eigenmaps. SPE aims to reconstruct the input graph exactly. The embedding is stored as a positive semidefinite kernel matrix K and a connectivity algorithm $\mathcal{G}$ is defined which reconstructs the graph from K. The kernel K is chosen such that it maximizes $tr(KW)$ which attempts to recover rank-1 spectral embedding. Choice of the connectivity algorithm $\mathcal{G}$ induces constraints on this objective function. For e.g., if the connectivity scheme is to connect each node to neighbors which lie within a ball of radius ϵ, the constraint $(K_{ii} + K_{jj} - 2K_{ij})(W_{ij} - 1/2) \leq \epsilon(W_{ij} - 1/2)$ produces a kernel which can perfectly reconstruct the original graph. To handle noise in the graph, a slack variable is added. For ξ-connectivity, the optimization thus becomes $max\ tr(KA) - C\xi$ s.t. $(K_{ii} + K_{jj} - 2K_{ij})(W_{ij} - 1/2) \leq \epsilon(W_{ij} - 1/2) - \xi$, where ξ is the slack variable and C controls slackness.

3.3.5. Graph Factorization (GF)

To the best of our knowledge, Graph Factorization [21] was the first method to obtain a graph embedding in $O(|E|)$ time. To obtain the embedding, GF factorizes the adjacency matrix of the graph, minimizing the following loss function

$$\phi(Y, \lambda) = \frac{1}{2} \sum_{(i,j) \in E} (W_{ij} - < Y_i, Y_j >)^2 + \frac{\lambda}{2} \sum_i \|Y_i\|^2,$$

where λ is a regularization coefficient. Note that the summation is over the observed edges as opposed to all possible edges. This is an approximation in the interest of scalability, and as such it may introduce noise in the solution. Note that as the adjacency matrix is often not positive semidefinite, the minimum of the loss function is greater than 0 even if the dimensionality of embedding is $|V|$.

3.3.6. GraRep

GraRep [27] defines the node transition probability as $T = D^{-1}W$ and preserves k-order proximity by minimizing $\|X^k - Y_s^k Y_t^{kT}\|_F^2$ where X^k is derived from T^k (refer to [27] for a detailed derivation). It then concatenates Y_s^k for all k to form Y_s. Note that this is similar to HOPE [24] which minimizes $\|S - Y_s Y_t^T\|_F^2$ where S is an appropriate similarity matrix. The drawback of GraRep is scalability, since T^k can have $O(|V|^2)$ non-zero entries.

3.3.7. HOPE

HOPE [24] preserves higher order proximity by minimizing $\|S - Y_s Y_t^T\|_F^2$, where S is the similarity matrix. The authors experimented with different similarity measures, including Katz Index, Rooted Page Rank, Common Neighbors, and Adamic-Adar score. They represented each similarity measure as $S = M_g^{-1} M_l$, where both M_g and M_l are sparse. This enables HOPE to use generalized Singular Value Decomposition (SVD) [34] to obtain the embedding efficiently.

3.3.8. Additional variants

For the purpose of dimensionality reduction of high dimensional data, there are several other methods developed capable of performing graph embedding. Yan et al. [35] survey a list of such methods including Principal Component Analysis (PCA) [36], Linear Discrimant Analysis (LDA) [37], ISOMAP [38], Multidimesional Scaling (MDS) [39], Locality Preserving Properties (LPP) [40] and Kernel Eigenmaps [41]. Matrinex et al. [42]) proposed a general framework, non-negative graph embedding, which yields non-negative embeddings for these algorithms.

A number of recent techniques have focused on jointly learning network structure and additional node attribute information available for the network., Augmented Relation Embedding (ARE) [43] augments network with content based features for images and modifies the graph-Laplacian to capture such information. Text-associated DeepWalk (TADW) [44] performs matrix factorization on node similarity matrix disentangling the representation using text feature matrix. Heterogeneous Network Embedding (HNE) [45] learns representation for each modality of the network and then unifies them into a common space using linear transformations. Other works [46–48] perform similar transformations between various node attributes and learn joint embedding.

3.4. Random walk based methods

Random walks have been used to approximate many properties in the graph including node centrality [49] and similarity [50]. They are especially useful when one can either only partially observe the graph, or the graph is too large to measure in its entirety. Embedding techniques using random walks on graphs to obtain node representations have been proposed: *DeepWalk* and *node2vec* are two examples.

3.4.1. DeepWalk

DeepWalk [28] preserves higher-order proximity between nodes by maximizing the probability of observing the last k nodes and the next k nodes in the random walk centered at v_i, i.e. maximizing $\log Pr(v_{i-k}, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{i+k} | Y_i)$, where $2k+1$ is the length of the random walk. The model generates multiple random walks each of length $2k+1$ and performs the optimization over sum of log-likelihoods for each random walk. A dot-product based decoder is used to reconstruct the edges from the node embeddings.

3.4.2. node2vec

Similar to DeepWalk [28], *node2vec* [29] preserves higher-order proximity between nodes by maximizing the probability of occurrence of subsequent nodes in fixed length random walks. The crucial difference from DeepWalk is that *node2vec* employs biased-random walks that provide a trade-off between breadth-first (BFS) and depth-first (DFS) graph searches, and hence produces higher-quality and more informative embeddings than DeepWalk. Choosing the right balance enables *node2vec* to preserve community structure as well as structural equivalence between nodes.

3.4.3. Hierarchical representation learning for networks (HARP)

DeepWalk and *node2vec* initialize the node embeddings randomly for training the models. As their objective function is non-convex, such initializations can be stuck in local optima. HARP [51] introduces a strategy to improve the solution and avoid local optima by better weight initialization. To this purpose, HARP creates hierarchy of nodes by aggregating nodes in the previous layer of hierarchy using graph coarsening. It then generates embedding of the coarsest graph and initializes the node embeddings of the refined graph (one up in the hierarchy) with the learned embedding. It propagates such embeddings through the hierarchy to obtain the embeddings of the original graph. Thus HARP can be used in conjunction with random walk based methods like DeepWalk and *node2vec* to obtain better solutions to the optimization function.

3.4.4. Walklets

DeepWalk and *node2vec* implicitly preserve higher order proximity between nodes by generating multiple random walks which connect nodes at various distances due to its stochastic nature. On the other hand, factorization based approaches like GF and HOPE explicitly preserve distances between nodes by modeling it in their objective function. Walklets [52] combine this idea of explicit modeling with random walks. The model modifies the random walk strategy used in DeepWalk by skipping over some nodes in the graph. This is performed for multiple skip lengths, analogous to factorizing A^k in GraRep, and the resulting set of random walks are used for training the model similar to DeepWalk.

3.4.5. Additional variants

There have been several variations of the above methods proposed recently. Similar to augmenting graph structure with node attributes for factorization based methods, GenVector [53], Discriminative Deep Random Walk (DDRW) [54], Tri-party Deep Network Representation (TriDNR) [55] and [56] extend random walks to jointly learn network structure and node attributes.

3.5. Deep learning based methods

The growing research on deep learning has led to a deluge of deep neural networks based methods applied to graphs [23,30,57]. Deep autoencoders have been used for dimensionality reduction [58] due to their ability to model non-linear structure in the data. Recently, SDNE [23], DNGR [30] utilized this ability of deep autoencoder to generate an embedding model that can capture non-linearity in graphs.

3.5.1. Structural deep network embedding (SDNE)

Wang et al. [23] proposed to use deep autoencoders to preserve the first and second order network proximities. They achieve this by jointly optimizing the two proximities. The approach uses highly non-linear functions to obtain the embedding. The model consists of two parts: unsupervised and supervised. The former consists of an autoencoder aiming at finding an embedding for a node which can reconstruct its neighborhood. The latter is based on Laplacian Eigenmaps [25] which apply a penalty when similar vertices are mapped far from each other in the embedding space.

3.5.2. Deep neural networks for learning graph representations (DNGR)

DNGR combines random surfing with deep autoencoder. The model consists of 3 components: random surfing, *positive pointwise mutual information* (PPMI) calculation and stacked denoising autoencoders. Random surfing model is used on the input graph to generate a probabilistic co-occurence matrix, analogous to similarity matrix in HOPE. The matrix is transformed to a PPMI matrix and input into a stacked denoising autoencoder to obtain the embedding. Inputting PPMI matrix ensures that the autoencoder model can capture higher order proximity. Furthermore, using stacked denoising autoencoders aids robustness of the model in presence of noise in the graph as well as in capturing underlying structure required for tasks such as link prediction and node classification.

3.5.3. Graph convolutional networks (GCN)

Deep neural network based methods discussed above, namely SDNE and DNGR, take as input the global neighborhood of each node (a row of PPMI for DNGR and adjacency matrix for SDNE). This can be computationally expensive and inoptimal for large sparse graphs. Graph Convolutional Networks (GCNs) [31] tackle this problem by defining a convolution operator on graph. The model iteratively aggregates the embeddings of neighbors for a node and uses a function of the obtained embedding and its embedding at previous iteration to obtain the new embedding. Aggregating embedding of only local neighborhood makes it scalable and multiple iterations allows the learned embedding of a node to characterize global neighborhood.

Several recent papers [59–64] have proposed methods using convolution on graphs to obtain semi-supervised embedding, which can be used to obtain unsupervised embedding by defining unique labels for each node. The approaches vary in the construction of convolutional filters which can broadly be categorized into spatial and spectral filters. Spatial filters operate directly on the original graph and adjacency matrix whereas spectral filters operate on the spectrum of graph-Laplacian.

3.5.4. Variational graph auto-encoders (VGAE)

Kipf et al. [65] evaluate the performance of variational autoencoders [66] on the task of graph embedding. The model uses a graph convolutional network (GCN) encoder and an inner product decoder. The input is adjacency matrix and they rely on GCN to learn the higher order dependencies between nodes. They empirically show that using variational autoencoders can improve performance compared to non-probabilistic autoencoders.

3.6. Other methods

3.6.1. LINE

LINE [22] explicitly defines two functions, one each for first- and second-order proximities, and minimizes the combination of the two. The function for first-order proximity is similar to that of Graph Factorization (GF) [21] in that they both aim to keep the adjacency matrix and dot product of embeddings close. The difference is that GF does this by directly minimizing the difference of the two. Instead, LINE defines two joint probability distributions for each pair of vertices, one using adjacency matrix and the other using the embedding. Then, LINE minimizes the Kullback–Leibler (KL) divergence of these two distributions. The two distributions and the objective function are as follows

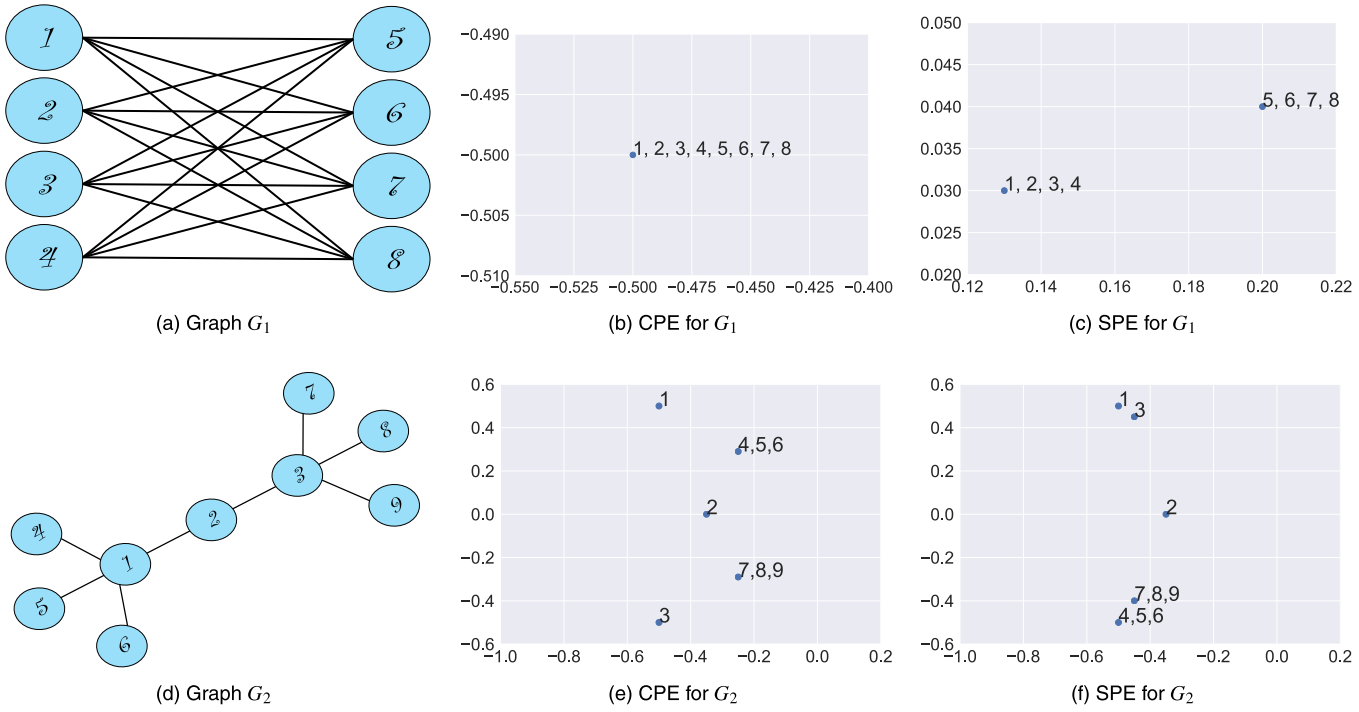
$$p_1(v_i, v_j) = \frac{1}{1 + exp(- < Y_i, Y_j >)}$$

$$\hat{p_1}(v_i, v_j) = \frac{W_{ij}}{\sum_{(i,j)\in E} W_{ij}}$$

$$O_1 = KL(\hat{p_1}, p_1)$$

$$O_1 = - \sum_{(i,j)\in E} W_{ij} \log p_1(v_i, v_j).$$

The authors similarly define probability distributions and objective function for the second-order proximity.



(a) Graph G_1 (b) CPE for G_1 (c) SPE for G_1

(d) Graph G_2 (e) CPE for G_2 (f) SPE for G_2

Fig. 1. Examples illustrating the effect of type of similarity preserved. Here, CPE and SPE stand for *Community Preserving Embedding* and *Structural-equivalence Preserving Embedding*, respectively.

3.7. Discussion

We can interpret embeddings as representations which describe graph data. Thus, embeddings can yield insights into the properties of a network. We illustrate this in Fig. 1. Consider a complete bipartite graph G. An embedding algorithm which attempts to keep two connected nodes close (i.e., preserve the community structure), would fail to capture the structure of the graph — as shown in Fig. 1(b). However, an algorithm which embeds structurally-equivalent nodes together learns an interpretable embedding — as shown in Fig. 1(c). Similarly, in 1(d) we consider a graph with two star components connected through a hub. Nodes 1 and 3 are structurally equivalent (they link to the same nodes) and are clustered together in Fig. 1(f), whereas in Fig. 1(e) they are far apart. The classes of algorithms above can be described in terms of their ability to explain the properties of graphs.

Factorization-based methods are not capable of learning an arbitrary function, e.g., to explain network connectivity. Thus, unless explicitly included in their objective function, they cannot learn structural equivalence. In random walk based methods, the mixture of equivalences can be controlled to a certain extent by varying the random walk parameters. Deep learning methods can model a wide range of functions following the universal approximation theorem [67]: given enough parameters, they can learn the mix of community and structural equivalence, to embed the nodes such that the reconstruction error is minimized. We can interpret the weights of the autoencoder as a representation of the structure of the graph. For example, Fig. 1(c) plots the embedding learned by SDNE for the complete bipartite graph G_1. The autoencoder stored the bipartite structure in weights and achieved perfect reconstruction. Given the variety of properties of real-world graphs, using general non-linear models that span a large class of functions is a promising direction that warrants further exploration.

4. Applications

As graph representations, embeddings can be used in a variety of tasks. These applications can be broadly classified as: network compression (Section 4.1), visualization (Section 4.2), clustering (Section 4.3), link prediction (Section 4.4), and node classification (Section 4.5).

4.1. Network compression

Feder et al. [68] introduced the concept of network compression (a.k.a. graph simplification). For a graph G, they defined a compression G^* which has smaller number of edges. The goal was to store the network more efficiently and run graph analysis algorithms faster. They obtained the compression graph by partitioning the original graph into bipartite cliques and replacing them by trees, thus reducing the number of edges. Over the years, many researchers have used aggregation based methods [69–71] to compress graphs. The main idea in this line of work is to exploit the link structure of the graph to group nodes and edges. Navlakha et al. [72] used Minimum Description Length (MDL) [73] from information theory to summarize a graph into a graph summary and edge correction.

Similar to these representations, graph embedding can also be interpreted as a summarization of graph. Wang et al. [23] and Ou et al. [24] tested this hypothesis explicitly by reconstructing the original graph from the embedding and evaluating the reconstruction error. They show that a low dimensional representation for each node (in the order of 100s) suffices to reconstruct the graph with high precision.

4.2. Visualization

Application of visualizing graphs can be dated back to 1736 when Euler used it to solve “Konigsberger Bruckenproblem” [74]. In the recent years, graph visualization has found applications in soft-

ware engineering [75], electrical circuits [76], biology [1] and sociology [2]. Battista et al. [76] and Eades et al. [77] survey a range of methods used to draw graphs and define aesthetic criteria for this purpose. Herman et al. [78] generalize this and view it from an information visualization perspective. They study and compare various traditional layouts used to draw graphs including tree-, 3D- and hyperbolic-based layouts.

As embedding represents a graph in a vector space, dimensionality reduction techniques like Principal Component Analysis (PCA) [79] and *t-distributed stochastic neighbor embedding* (t-SNE) [8] can be applied on it to visualize the graph. The authors of DeepWalk [28] illustrated the goodness of their embedding approach by visualizing the Zachary's Karate Club network. The authors of LINE [22] visualized the DBLP co-authorship network, and showed that LINE is able to cluster together authors in the same field. The authors of SDNE [23] applied it on 20-Newsgroup document similarity network to obtain clusters of documents based on topics.

4.3. Clustering

Graph clustering (a.k.a., network partitioning) can be of two types: (a) structure based, and (b) attribute based clustering. The former can be further divided into two categories, namely community based, and structurally equivalent clustering. Structure-based methods [7,20,80], aim to find dense subgraphs with high number of intra-cluster edges, and low number of inter-cluster edges. Structural equivalence clustering [81], on the contrary, is designed to identify nodes with similar roles (like bridges and outliers). Attribute based methods [19] utilize node labels, in addition to observed links, to cluster nodes.

White et al. [82] used k-means on the embedding to cluster the nodes and visualize the clusters obtained on Wordnet and NCAA data sets verifying that the clusters obtained have intuitive interpretation. Recent methods on embedding haven't explicitly evaluated their models on this task and thus it is a promising field of research in the graph embedding community.

4.4. Link prediction

Networks are constructed from the observed interactions between entities, which may be incomplete or inaccurate. The challenge often lies in identifying spurious interactions and predicting missing information. Link prediction refers to the task of predicting either missing interactions or links that may appear in the future in an evolving network. Link prediction is pervasive in biological network analysis, where verifying the existence of links between nodes requires costly experimental tests. Limiting the experiments to links ordered by presence likelihood has been shown to be very cost effective. In social networks, link prediction is used to predict probable friendships, which can be used for recommendation and lead to a more satisfactory user experience. Liben–Nowell et al. [5], Lu et al. [83] and Hasan et al. [84] survey the recent progress in this field and categorize the algorithms into (a) similarity based (local and global) [13,14,85], (b) maximum likelihood based [15,16] and (c) probabilistic methods [17,18,86].

Embeddings capture inherent dynamics of the network either explicitly or implicitly thus enabling application to link prediction. Wang et al. [23] and Ou et al. [24] predict links from the learned node representations on publicly available collaboration and social networks. In addition, Grover et al. [29] apply it to biology networks. They show that on these data sets links predicted using embeddings are more accurate than traditional similarity based link prediction methods described above.

4.5. Node classification

Often in networks, a fraction of nodes are labeled. In social networks, labels may indicate interests, beliefs, or demographics. In language networks, a document may be labeled with topics or keywords, whereas the labels of entities in biology networks may be based on functionality. Due to various factors, labels may be unknown for large fractions of nodes. For example, in social networks many users do not provide their demographic information due to privacy concerns. Missing labels can be inferred using the labeled nodes and the links in the network. The task of predicting these missing labels is also known as node classification. Bhagat et al.[6] survey the methods used in the literature for this task. They classify the approaches into two categories, i.e., feature extraction based and random walk based. Feature-based models [11,12,87] generate features for nodes based on their neighborhood and local network statistics and then apply a classifier like Logistic Regression [88] and Naive Bayes [89] to predict the labels. Random walk based models [9,10] propagate the labels with random walks.

Embeddings can be interpreted as automatically extracted node features based on network structure and thus falls into the first category. Recent work [22–24,28,29] has evaluated the predictive power of embedding on various information networks including language, social, biology and collaboration graphs. They show that embeddings can predict missing labels with high precision.

5. Experimental setup

Our experiments evaluate the feature representations obtained using the methods reviewed before on the previous four application domains. Next, we specify the datasets and evaluation metrics we used. The experiments were performed on a Ubuntu 14.04.4 LTS system with 32 cores, 128GB RAM and a clock speed of 2.6 GHz. The GPU used for deep network based models was Nvidia Tesla K40C.

5.1. Datasets

We evaluate the embedding approaches on a synthetic and 6 real datasets. The datasets are summarized in Table 3.

SYN-SBM: We generate synthetic graph using Stochastic Block Model [90] with 1024 nodes and 3 communities. We set the in-block and cross-block probabilities as 0.1 and 0.01 respectively. As we know the community structure in this graph, we use it to visualize the embeddings learnt by various approaches.

KARATE [91]: Zachary's karate network is a well-known social network of a university karate club. It has been widely studied in social network analysis. The network has 34 nodes, 78 edges and 2 communities.

BLOGCATALOG [92]: This is a network of social relationships of the bloggers listed on the BlogCatalog website. The labels represent blogger interests inferred through the metadata provided by the bloggers. The network has 10,312 nodes, 333,983 edges and 39 different labels.

YOUTUBE [93]: This is a social network of Youtube users. This is a large network containing 1,157,827 nodes and 4,945,382 edges. The labels represent groups of users who enjoy common video genres.

HEP-TH [94]: The original dataset contains abstracts of papers in *High Energy Physics Theory* for the period from January 1993 to April 2003. We create a collaboration network for the papers published in this period. The network has 7980 nodes and 21,036 edges.

ASTRO-PH [95]: This is a collaboration network of authors of papers submitted to e-print arXiv during the period from January

Table 3
Dataset Statistics.

Name	Synthetic	Social network			Collaboration network		Biology network
	SYN-SBM	KARATE	BLOGCATALOG	YOUTUBE	HEP-TH	ASTRO-PH	PPI
$\|V\|$	1024	34	10,312	1,157,827	7980	18,772	3,890
$\|E\|$	29,833	78	333,983	4,945,382	21,036	396,160	38,739
Avg. degree	58.27	4.59	64.78	8.54	5.27	31.55	19.91
No. of labels	3	4	39	47	–	–	50

1993 to April 2003. The network has 18,772 nodes and 396,160 edges.

PROTEIN-PROTEIN INTERACTIONS (PPI) [96]: This is a network of biological interactions between proteins in humans. This network has 3890 nodes and 38,739 edges.

5.2. Evaluation metrics

To evaluate the performance of embedding methods on graph reconstruction and link prediction, we use Precision at k ($Pr@k$) and *MeanAveragePrecision*(*MAP*) as our metrics. For node classification, we use *micro-F1* and *macro-F1*. These metrics are defined as follows:

Pr@k is the fraction of correct predictions in top k predictions. It is defined as $Pr@k = \frac{|E_{pred}(1{:}k) \cap E_{obs}|}{k}$, where $E_{pred}(1{:}\,k)$ are the top k predictions and E_{obs} are the observed edges. For the task of graph reconstruction, $E_{obs} = E$ and for link prediction, E_{obs} is the set of hidden edges.

MAP estimates precision for every node and computes the average over all nodes, as follows:

$$MAP = \frac{\sum_i AP(i)}{|V|},$$

where $AP(i) = \frac{\sum_k Pr@k(i) \cdot \mathbb{I}\{E_{pred_i}(k) \in E_{obs_i}\}}{|\{k : E_{pred_i}(k) \in E_{obs_i}\}|}$, $Pr@k(i) = \frac{|E_{pred_i}(1{:}k) \cap E_{obs_i}|}{k}$, and E_{pred_i} and E_{obs_i} are the predicted and observed edges for node i respectively.

macro-F1 , in a multi-label classification task, is defined as the average $F1$ of all the labels, i.e.,

$$macro - F1 = \frac{\sum_{l \in \mathcal{L}} F1(l)}{|\mathcal{L}|},$$

where $F1(l)$ is the $F1$-score for label l.

micro-F1 calculates $F1$ globally by counting the total true positives, false negatives and false positives, giving equal weight to each instance. It is defined as follows:

$$micro - F1 = \frac{2 * P * R}{P + R},$$

where $P = \frac{\sum_{l \in \mathcal{L}} TP(l)}{\sum_{l \in \mathcal{L}} (TP(l) + FP(l))}$, and $R = \frac{\sum_{l \in \mathcal{L}} TP(l)}{\sum_{l \in \mathcal{L}} (TP(l) + FN(l))}$, are precision (P) and recall (R) respectively, and $TP(l)$, $FP(l)$ and $FN(l)$ denote the number of true positives, false positives and false negatives respectively among the instances which are associated with the label l either in the ground truth or the predictions.

6. Experiments and analysis

In this section, we evaluate and compare embedding methods on the for tasks presented above. For each task, we show the effect of number of embedding dimensions on the performance and compare hyper parameter sensitivity of the methods. Furthermore, we correlate the performance of embedding techniques on various tasks varying hyper parameters to test the notion of an "all-good" embedding which can perform well on all tasks.

6.1. Graph reconstruction

Embeddings as a low-dimensional representation of the graph are expected to accurately reconstruct the graph. Note that reconstruction differs for different embedding techniques (refer to Section 3). For each method, we reconstruct the proximity of nodes and rank pair of nodes according to their proximity. Then we calculate the ratio of real links in top k predictions as the reconstruction precision. As the number of possible node pairs ($N(N-1)$) can be very large for networks with a large number of nodes, we randomly sample 1024 nodes for evaluation. We obtain 5 such samples for each dataset and calculate the mean and standard deviation of precision and MAP values for subgraph reconstruction. To obtain optimal hyperparameters for each embedding method, we compare the mean MAP values for each hyperparameter. We then re-run the experiments with the optimal hyperparameter on 5 random samples of 1024 nodes and report the results.

Fig. 2 illustrates the reconstruction precision obtained by 128-dimensional embeddings. We observe that although performance of methods is dataset dependent, embedding approaches which preserve higher order proximities in general outperform others. Exceptional performance of Laplacian Eigenmaps on SBM can be attributed to the lack of higher order structure in the data set. We also observe that SDNE consistently performs well on all data sets. This can be attributed to its capability of learning complex structure from the network. Embeddings learned by *node2vec* have low reconstruction precision. This may be due to the highly non-linear dimensionality reduction yielding a non-linear manifold. However, HOPE, which learns linear embeddings but preserves higher order proximity reconstructs the graph well without any additional parameters.

Effect of dimension. Fig. 3 illustrates the effect of dimension on the reconstruction error. With a couple of exceptions, as the number of dimensions increase, the MAP value increases. This is intuitive as higher number of dimensions are capable of storing more information. We also observe that SDNE is able to embed the graphs in 16-dimensional vector space with high precision although decoder parameters are required to obtain such precision.

6.2. Visualization

Since embedding is a low-dimensional vector representation of nodes in the graph, it allows us to visualize the nodes to understand the network topology. As different embedding methods preserve different structures in the network, their ability and interpretation of node visualization differ. For instance, embeddings learned by *node2vec* with parameters set to prefer BFS random walk would cluster structurally equivalent nodes together. On the other hand, methods which directly preserve k-hop distances between nodes (GF, LE and LLE with $k = 1$ and HOPE and SDNE with $k > 1$) cluster neighboring nodes together. We compare the ability of different methods to visualize nodes on SBM and Karate graph. For SBM, following [23], we learn a 128-dimensional embedding for each method and input it to t-SNE [8] to reduce the dimensionality to 2 and visualize nodes in a 2-dimensional space.

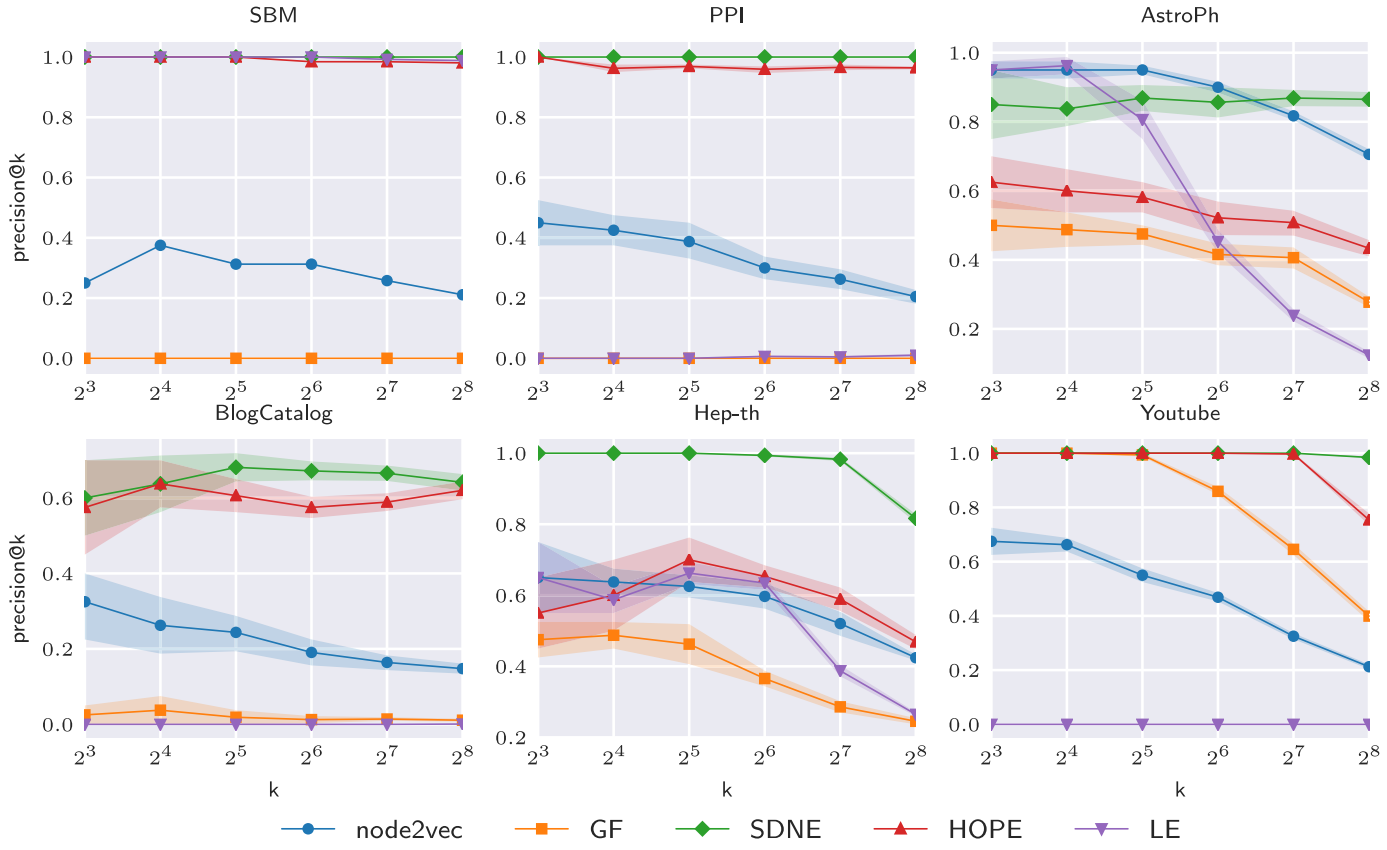


Fig. 2. Precision@k of graph reconstruction for different data sets (dimension of embedding is 128).

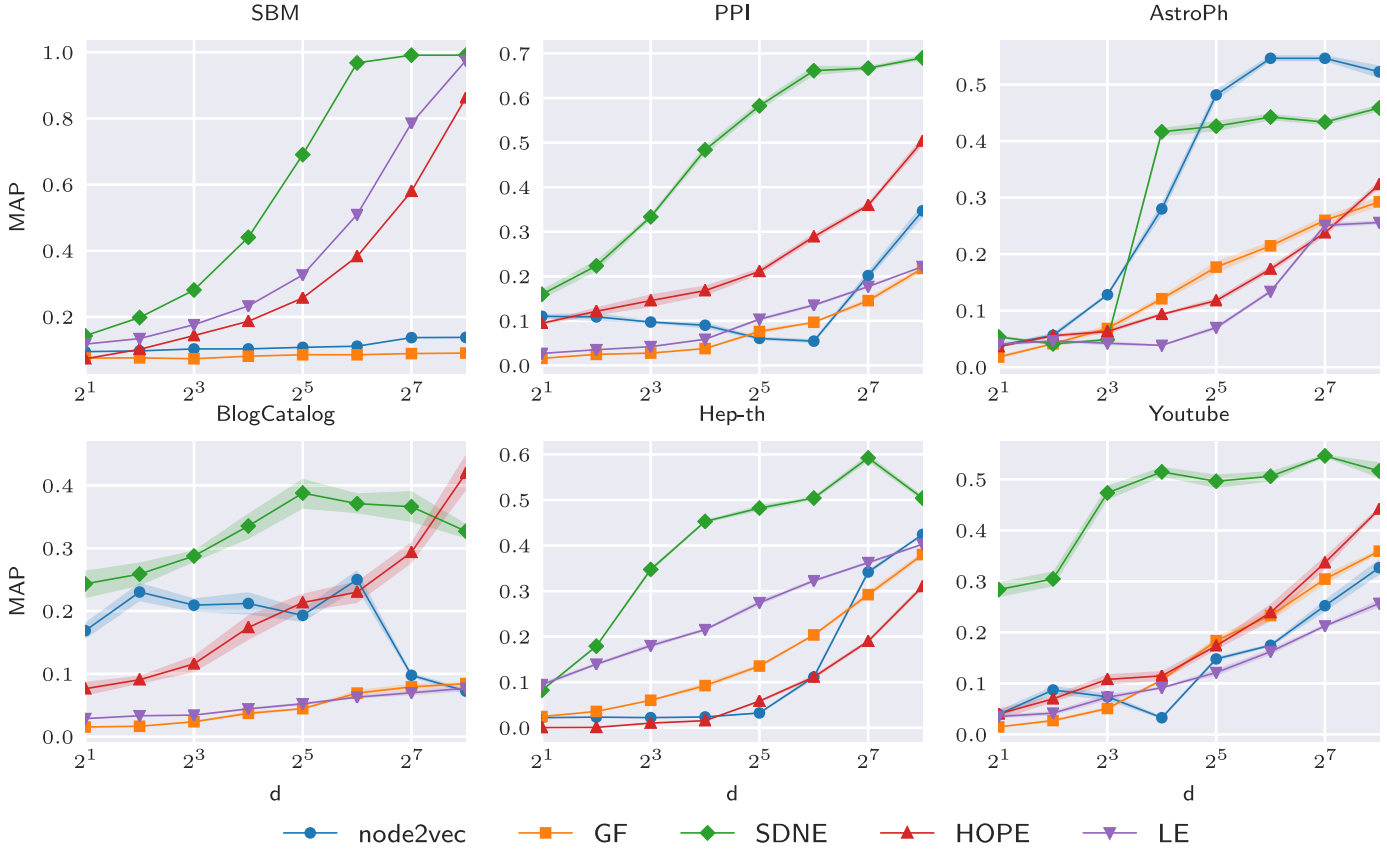


Fig. 3. MAP of graph reconstruction for different data sets with varying dimensions.

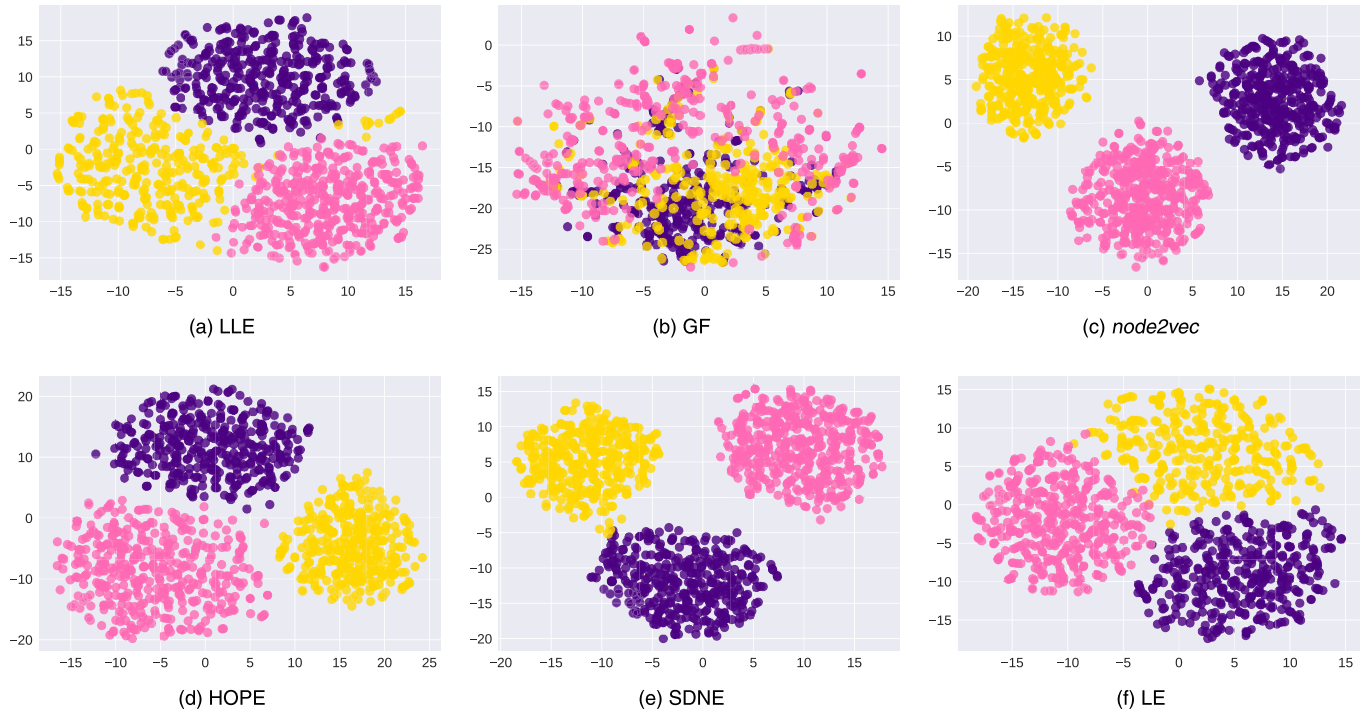


Fig. 4. Visualization of SBM using t-SNE (original dimension of embedding is 128). Each point corresponds to a node in the graph. Color of a node denotes its community.

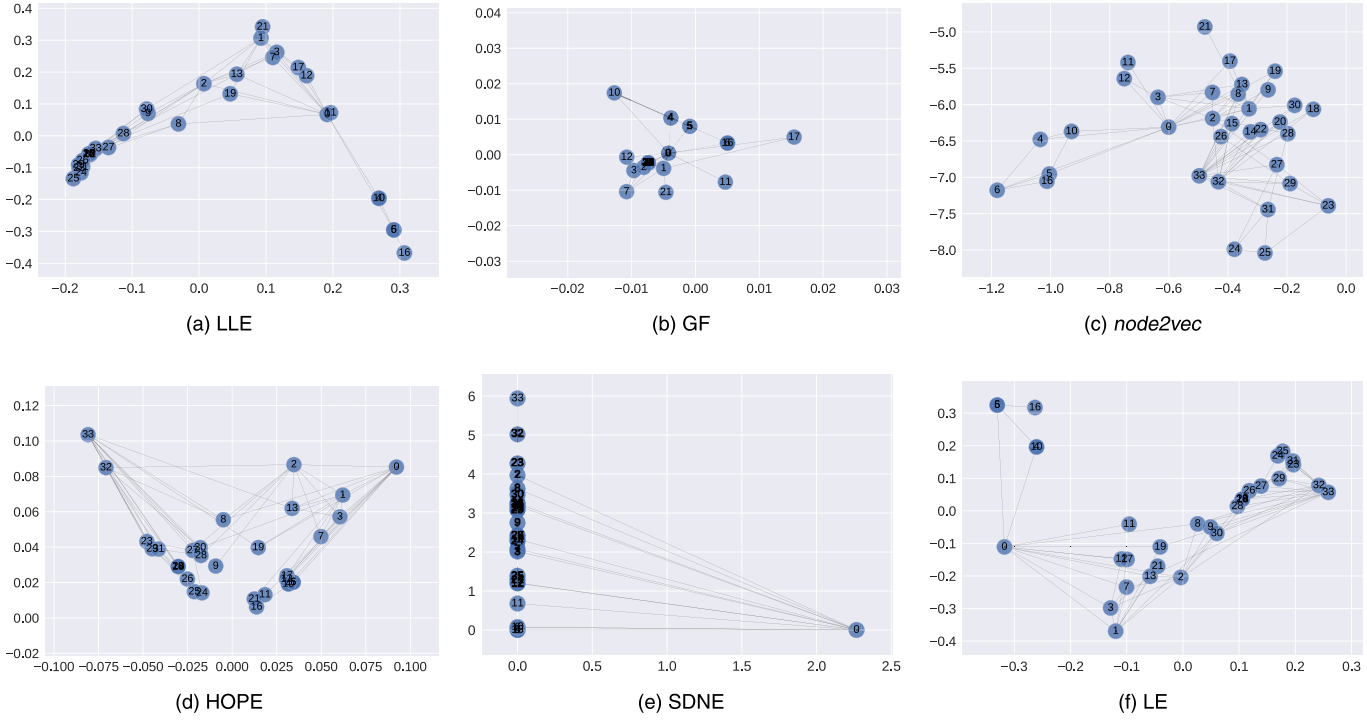


Fig. 5. Visualization of Karate club graph. Each point corresponds to a node in the graph.

Visualization of SBM is show in Fig. 4. As we know the underlying community structure, we use the community label to color the nodes. We observe that embeddings generated by HOPE and SDNE which preserve higher order proximities well separate the communities although as the data is well structured LE, GF and LLE are able to capture community structure to some extent.

We visualize Karate graph (see Fig. 5) to illustrate the properties preserved by embedding methods. LLE and LE ((a) and (f)) attempt to preserve the community structure of the graph and cluster nodes with high intra-cluster edges together. GF ((b)) embeds communities very closely and keeps leaf nodes far away from other nodes. In (d), we observe that HOPE embeds nodes 16 and 21, whose Katz similarity in the original graph is very low (0.0006), farthest apart (considering dot product similarity). *node2vec* and SDNE ((c) and (e)) preserve a mix of community structure and structural property of the nodes. Nodes 32 and 33, which are both

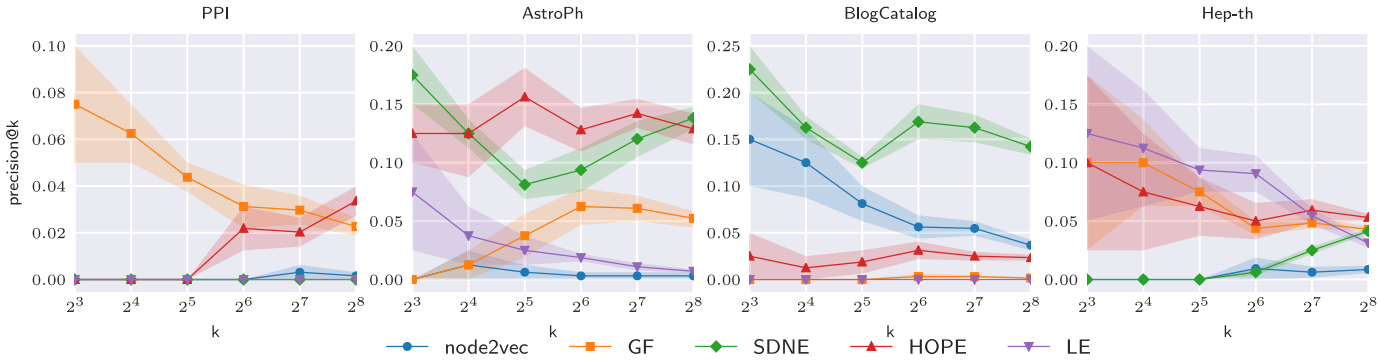


Fig. 6. Precision@k of link prediction for different data sets (dimension of embedding is 128).

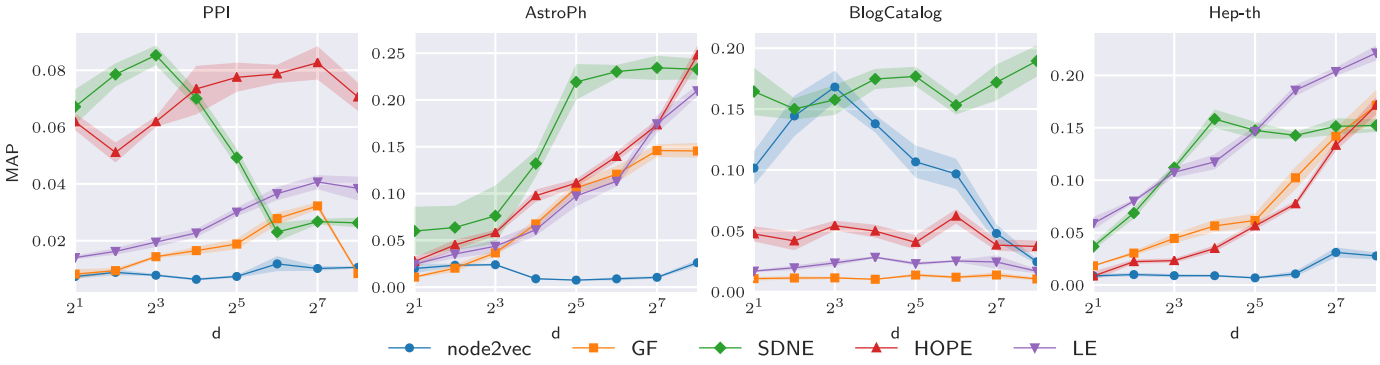


Fig. 7. MAP of link prediction for different data sets with varying dimensions.

high degree hubs and central in their communities, are embedded together and away from low degree nodes. Also, they are closer to nodes which belong to their communities. SDNE embeds node 0, which acts a bridge between communities, far away from other nodes. Note that, unlike for other methods, it does not imply that node 0 is disconnected from the rest of the nodes. The implication here is that SDNE identifies node 0 as a separate type of node and encodes its connection to other nodes in encoder and decoder. The ability of deep autoencoders to identify important nodes in the network has not been studied but given this observation we believe this direction can be promising.

6.3. Link prediction

Another important application of graph embedding is predicting unobserved links in the graph. A good network representation should be able to capture the inherent structure of graph well enough to predict the likely but unobserved links. To test the performance of different embedding methods on this task, for each data set we randomly hide 20% of the network edges. We learn the embedding using the rest of the 80% edges and predict the most likely edges which are not observed in the training data from the learned embedding. As with graph reconstruction, we generate 5 random subgraphs with 1024 nodes and test the predicted links against the held-out links in the subgraphs. We perform this experiment for each hyperparameter and re-run it for optimal hyperparameters on a new random 80-20 link split.

Figs. 6 and 7 show the *precision@k* results for link prediction with 128-dimensional embeddings and MAP for each dimension respectively. Here we can see that the performance of methods is highly data set dependent. *node2vec* achieves good performance on BlogCatalog but performs poorly on other data sets. HOPE achieves good performance on all data sets which implies that preserving higher order proximities is conducive to predicting unobserved links. Similarly, SDNE outperforms other methods with the exception on PPI for which the performance degrades drastically as embedding dimension increases above 8.

Effect of dimension. Fig. 7 illustrates the effect of embedding dimension on link prediction. We make two observations. Firstly, in PPI and BlogCatalog, unlike graph reconstruction performance does not improve as the number of dimensions increase. This may be because with more parameters the models overfit on the observed links and are unable to predict unobserved links. Secondly, even on the same data set, relative performance of methods depends on the embedding dimension. In PPI, HOPE outperforms other methods for higher dimensions,whereas embedding generated by SDNE achieves higher link prediction MAP for low dimensions.

6.4. Node classification

Predicting node labels using network topology is widely popular in network analysis and has variety of applications, including document classification and interest prediction. A good network embedding should capture the network structure and hence be useful for node classification. We compare the effectiveness of embedding methods on this task by using the generated embedding as node features to classify the nodes. The node features are input to a one-vs-rest logistic regression using the LIBLINEAR library. For each data set, we randomly sample 10% to 90% of nodes as training data and evaluate the performance on the remaining nodes. We perform this split 5 times and report the mean with confidence interval. For data sets with multiple labels per node, we assume that we know how many labels to predict.

Fig. 8 shows the results of our experiments. We can see that *node2vec* outperforms other methods on the task of node classification. As mentioned earlier (Section 3), *node2vec* preserves homophily as well as structural equivalence between nodes. Results suggest this can be useful in node classification: e.g., in BlogCatalog users may have similar interests, yet connect to others based on social ties rather than interests overlap. Similarly, proteins in

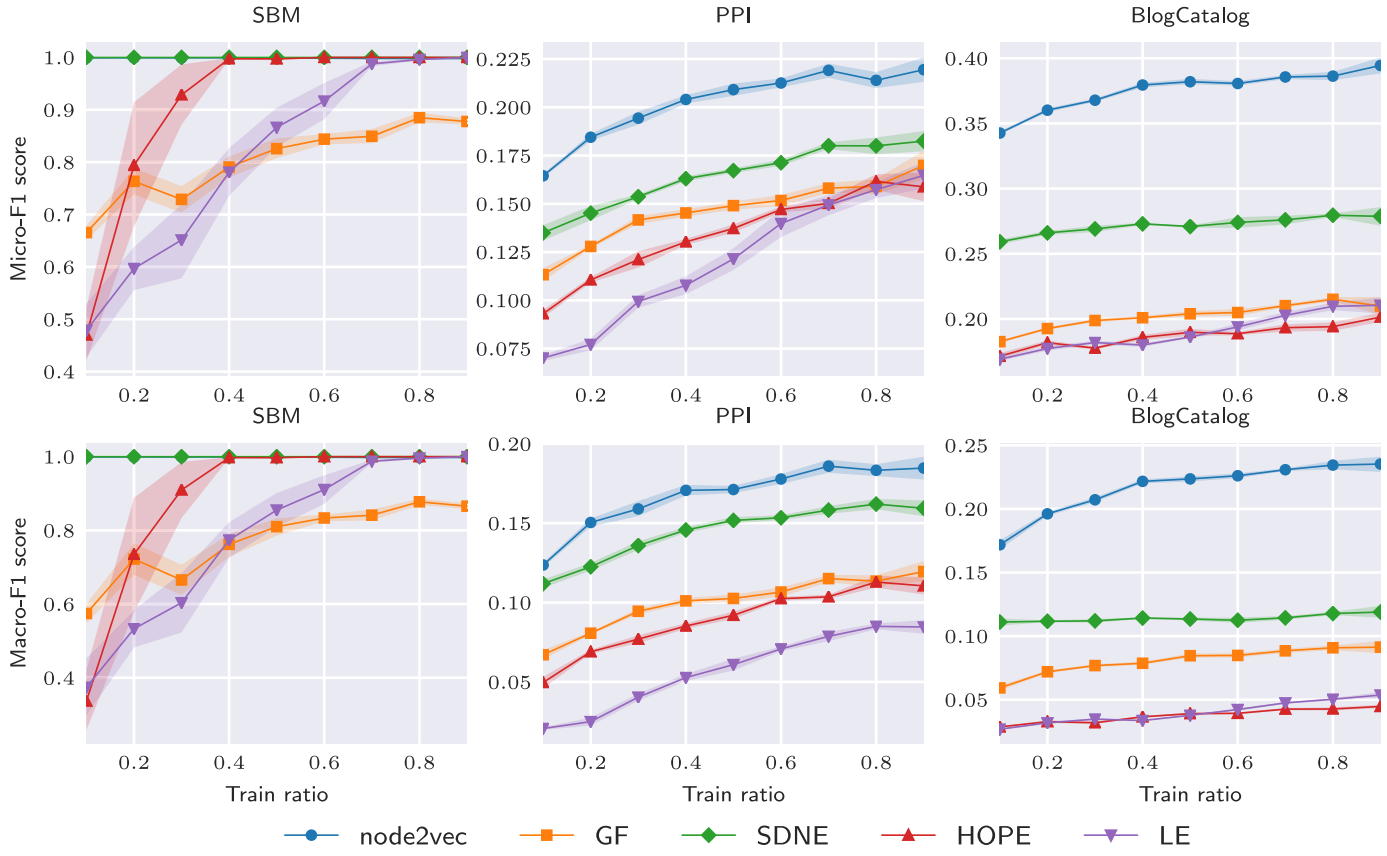


Fig. 8. Micro-F1 and Macro-F1 of node classification for different data sets varying the train-test split ratio (dimension of embedding is 128).

PPI may be related in functionality and interact with similar proteins but may not assist each other. However, in SBM, other methods outperform *node2vec* as labels reflect communities yet there is no structural equivalence between nodes.

Effect of dimension. Fig. 9 illustrates the effect of embedding dimensions on node classification. As with link prediction, we observe that performance often saturates or deteriorates after certain number of dimensions. This may suggest overfitting on the training data. As SBM exhibits very structured communities, an 8-dimensional embedding suffices to predict the communities. *node2vec* achieves best performance on PPI and BlogCat with 128 dimensions.

6.5. Hyperparameter sensitivity

In this section we plan to address the following questions:

- How robust are the embedding methods with respect to hyperparameters?
- Do optimal hyperparameters depend on the downstream tasks the embeddings are used for?
- What insights does performance variance with hyperparameter provide about a data set?

We answer these questions by analyzing performance of the embedding methods with various hyperparameters. We present results on SBM, PPI and Hep-th. Of the methods we evaluated in this survey, Laplacian Eigenmaps has no hyperparameters, thus is not included in the following analysis.

Graph Factorization (GF). The objective function of GF contains a weight regularization term with a coefficient. Intuitively, this coefficient controls the generalizability of the embedding. A low regularization coefficient facilitates better reconstruction but may overfit to the observed graph leading to poor prediction performance. On the other side, a high regularization may underrepresent the data and hence perform poorly on all tasks. We observe this effect in Fig. 10. We see that performance on prediction tasks, namely link prediction (Fig. 10b) and node classification (Fig. 10c), improves as we increase the regularization coefficient, reaches a peak and then starts deteriorating. However, graph reconstruction performance (Fig. 10a) may deteriorate with increasing regularization. We also note that the performance change is considerable and thus the coefficient should be carefully tuned to the given data set.

HOPE. As HOPE factorizes a similarity matrix between nodes to obtain the embedding, the hyperparameters depend on the method used to obtain the similarity matrix. Since in our experiments we used Katz index for this purpose, we evaluate the effect of the *attenuation factor* (β), which can be interpreted as the higher order proximity coefficient on performance. Graph structure affects the optimal value of this parameter. For well-structured graphs with tightly knit communities, high values of *beta* would erroneously assign dissimilar nodes closer in the embedding space. On the contrary, for graphs with weak community structure it is important to capture higher order distances and thus high values of *beta* may yield better embeddings. We validate our hypothesis in Fig. 11. As our synthetic data SBM consists of tight communities, increasing β does not improve the performance on any task. However, gain in performance with increasing *beta* can be observed in PPI and HEP-TH. This is expected as collaboration and protein networks tend to evolve via higher order connections. We observe that the optimal *beta* for these data sets is lowest for the task of graph reconstruction (Fig. 11(a)) and highest for link prediction (Fig. 11b), 2^{-6} and 2^{-4} respectively. This also follows intuition as higher order information is not necessary for reconstructing observed edges but is useful for prediction tasks. Nodes farther from each other are more

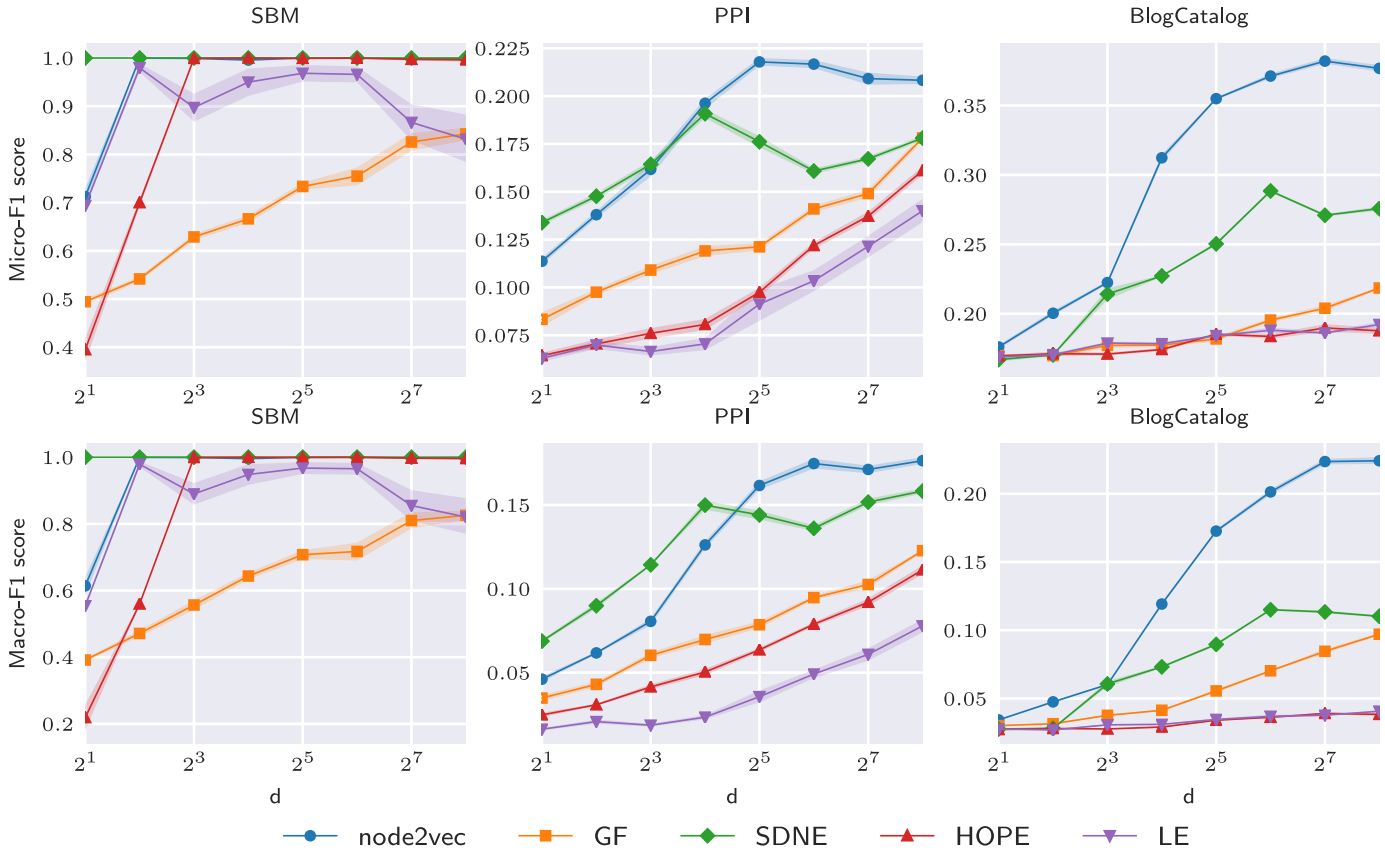


Fig. 9. Micro-F1 and Macro-F1 of node classification for different data sets varying the number of dimensions. The train-test split is 50%.

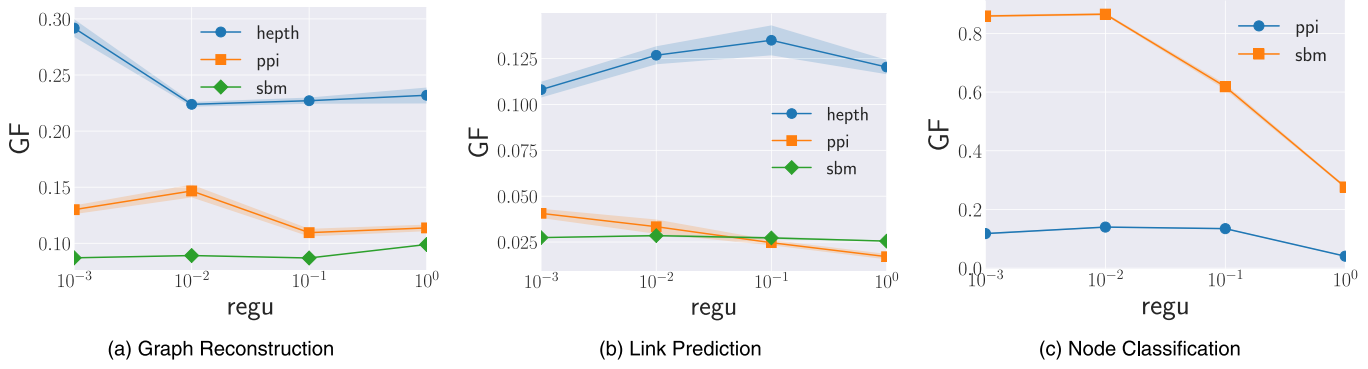


Fig. 10. Effect of regularization coefficient (α) in Graph Factorization on various tasks.

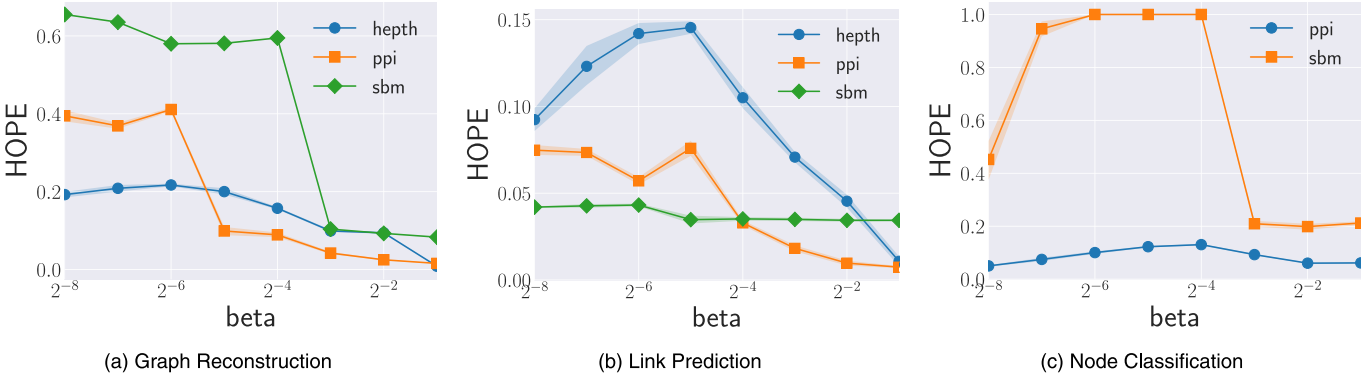


Fig. 11. Effect of attenuation factor (β) in HOPE on various tasks.

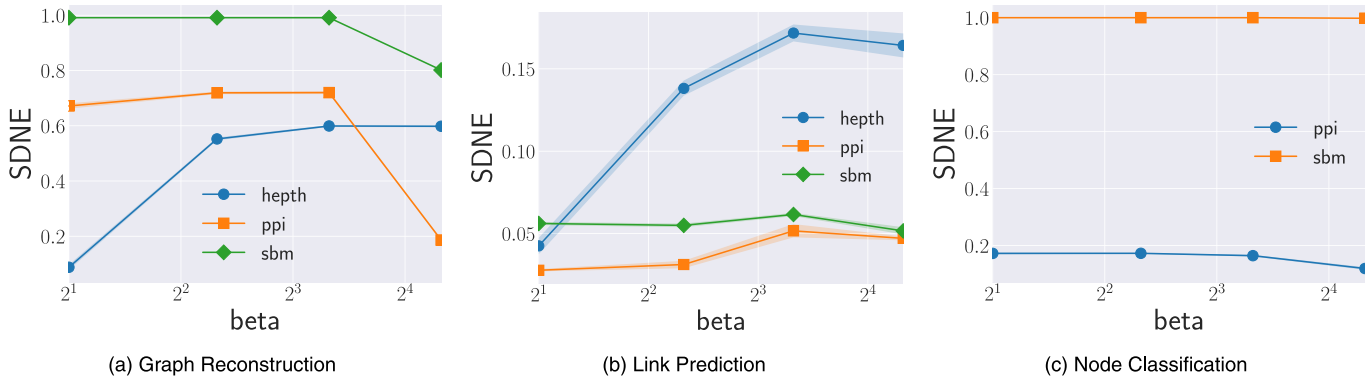


Fig. 12. Effect of observed link reconstruction weight in SDNE on various tasks.

likely to have common labels than links between them. For example, in a collaboration network, authors in a field will form a weak community but many of them need not be connected to each other (in other words, most researchers of a same community won't be co-authors of one another). Thus, if we were to predict field for an author, having information about other authors inside the weak community can help improve the accuracy of our model.

SDNE. SDNE uses a coupled deep autoencoder to embed graphs. It leverages a parameter to control the weight of observed and unobserved link reconstruction in the objective function. A high parameter value would focus on reconstructing observed links disregarding the absence of unobserved links. This parameter can be crucial as a low value could hinder predicting hidden links. Fig. 12 illustrates the results of our analysis. We observe that performance on link prediction (Fig. 12b) greatly varies depending on the weight, with a more than 3-fold increase in MAP for Hep-th with the optimal parameter value, and about 2-fold increase for PPI. The performance on node classification (Fig. 12c), on the other hand, is less affected by the parameter. Overall, we see that the maximum performance is achieved for an intermediate value of the weight above which the performance drops.

node2vec. *node2vec* performs biased random walks on the graph and embeds nodes commonly appearing together in them close in the embedding space. Of the various hyperparameters of the method (which include walk length, context size and bias weights), we analyze the effect of bias weights on performance and adopt commonly-used values for the remaining hyperparameter [29], namely, context size of 10 and walk length of 80. *node2vec* has two bias weights: (a) *inout* parameter (q), which controls the likelihood of random walk to go further away from the incoming node (higher values favor closer nodes), and (b) return parameter (p), which weighs the return probability (lower values favor return). Lower values of q would help capture longer distances between nodes and aims towards preserving structural equivalence. Fig. 13 illustrates the effect on PPI and HEPTH. In node classification (Fig. 13c) we observe that low q values help achieve higher accuracy suggesting that capturing structural equivalence is required to accurately embed the structure of the graph. On the contrary, high q values favor link prediction (Fig. 13b) following intuition that capturing community structure is crucial for the task. We make similar observations for SBM in Fig. 14 for the task of link prediction. *MAP* increases with increasing q until it reaches an optimal. We also note that the optimal values of q in SBM are much higher as the graph has strong community structure.

6.6. Discussion

We summarize the strengths and weaknesses of the evaluated methods in Table 4.

Table 4
Summary of strengths and weaknesses of evaluated methods.

	SDNE	HOPE	*node2vec*	GF	LE
Robust to Graph Structure	√	✗	√	✗	✗
Tunable for Downstream Tasks	√	✗	√	✗	✗
Few hyper-parameters	✗	√	✗	√	√
Low hyper-parameter Sensitivity	✗	✗	✗	✗	√
Scalable	✗	√	√	√	√

In our experiments, we observed that SDNE performs well on most of the data sets for all downstream tasks. Since SDNE does not make assumptions on the graph structure and uses the information provided in the graph to learn a suitable autoencoder, it can be tuned for almost any graph and task. Moreover, for different downstream tasks, hyper-parameters can be set separately to perform well on the task. However, SDNE has many hyper-parameters and the performance depends significantly on their settings. The large number of parameters makes it unsuitable for large graphs.

We also observed that *node2vec* performs well on most tasks, with the best performance in node classification. The values of random walk bias weights can be set for the task at hand. The method is linear in the number of nodes and the dimension of the embedding and is thus scalable. Moreover, it can be parallelized over a cluster by performing random walks on separate machines.

HOPE and GF have very few hyper-parameters to tune but the performance of the models depend largely on the graph structure. Although regularization parameter can act as a tradeoff between graph reconstruction and link prediction, the methods do not capture structural equivalence which can be crucial for node classification and visualization.

Laplacian Eigenmaps is a hyper-parameter-free method which can be quickly used to assess the performance on the tasks. However, the performance is not comparable to the rest of the methods and it cannot be tuned for a task.

7. A Python library for graph embedding

We released an open-source Python library, GEM (*Graph Embedding Methods*, https://github.com/palash1992/GEM), which provides a unified interface to the implementations of all the methods presented here, and their evaluation metrics. The library supports both weighted and unweighted graphs. GEM's hierarchical design and modular implementation should help the users to test the implemented methods on new datasets as well as serve as a platform to develop new approaches with ease.

GEM (https://github.com/palash1992/GEM) provides implementations of Locally Linear Embedding [26], Laplacian Eigenmaps [25], Graph Factorization [21], HOPE [24], SDNE [23] and node2vec [29].

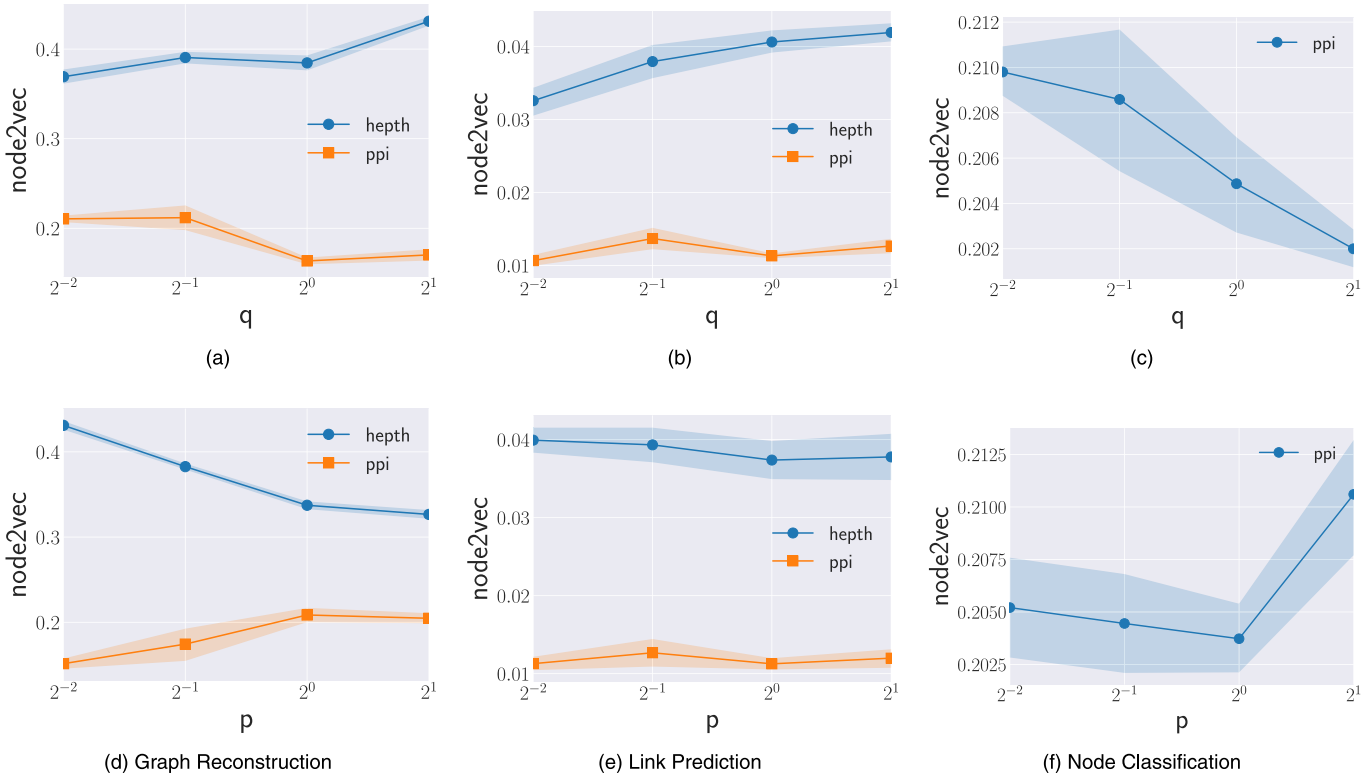


Fig. 13. Effect of random walk bias weights in *node2vec* on various tasks.

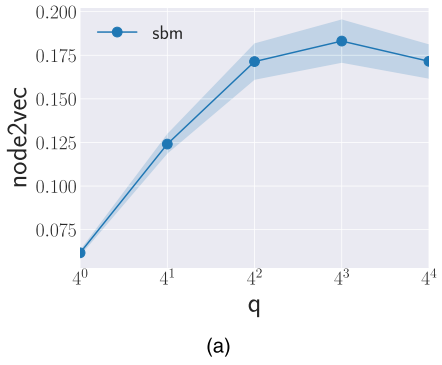


Fig. 14. Effect of random walk bias weights in *node2vec* on SBM.

For *node2vec*, we use the C++ implementation provided by the authors [95] and yield a Python interface. In addition, GEM provides an interface to evaluate the learned embedding on the four tasks presented above. The interface is flexible and supports multiple edge reconstruction metrics including cosine similarity, euclidean distance and decoder based (for autoencoder-based models). For multi-labeled node classification, the library uses one-vs-rest logistic regression classifiers and supports the use of other ad hoc classifiers.

8. Conclusion and future work

This review of graph embedding techniques covered three broad categories of approaches: factorization based, random walk based and deep learning based. We studied the structure and properties preserved by various embedding approaches and characterized the challenges faced by graph embedding techniques in general as well as each category of approaches. We reported various applications of embedding and their respective evaluation metrics. We empirically evaluated the surveyed methods on these applications using several publicly available real networks and compared their strengths and weaknesses. Finally, we presented an open-source Python library, named GEM, we developed with implementation of the embedding methods surveyed and evaluation tasks including graph reconstruction, link prediction, node classification and visualization.

We believe there are three promising research directions in the field of graph embedding: (1) exploring non-linear models, (2) studying evolution of networks, and (3) generate synthetic networks with real-world characteristics. As shown in the survey, general non-linear models (e.g. deep learning based) have shown great promise in capturing the inherent dynamics of the graph. They have the ability to approximate an arbitrary function which best explains the network edges and this can result in highly compressed representations of the network. One drawback of such approaches is the limited interpretability. Further research focusing on interpreting the embedding learned by these models can be very fruitful. Utilizing embedding to study graph evolution is a new research area which needs further exploration. Recent work by Dai et al. [97], Goyal et al. [98] and Zhu et al. [99] pursued this line of thought and illustrate how embeddings can be used for dynamic graphs. Generating synthetic networks with real-world characteristics has been a popular field of research [100] primarily for ease of simulations. Low dimensional vector representation of real graphs can help understand their structure and thus be useful to generate synthetic graphs with real world characteristics. Learning embedding with a generative model can help us in this regard.

References

[1] A. Theocharidis, S. Van Dongen, A. Enright, T. Freeman, Network visualization and analysis of gene expression data using biolayout express3d, Nat. Protoc. 4 (2009) 1535–1550.
[2] L.C. Freeman, Visualizing social networks, J. Social Struct. 1 (1) (2000) 4.

[3] R.F. i Cancho, R.V. Solé, The small world of human language, Proc. R. Soc. Lond. B 268 (1482) (2001) 2261–2265.
[4] J. Leskovec, J. Kleinberg, C. Faloutsos, Graph evolution: densification and shrinking diameters, ACM Trans. Knowl. Disc. Data (TKDD) 1 (1) (2007) 2.
[5] D. Liben-Nowell, J. Kleinberg, The link-prediction problem for social networks, J. Assoc. Inf. Sci. Technol. 58 (7) (2007) 1019–1031.
[6] S. Bhagat, G. Cormode, S. Muthukrishnan, Node classification in social networks, in: Social network data analytics, Springer, 2011, pp. 115–148.
[7] C.H. Ding, X. He, H. Zha, M. Gu, H.D. Simon, A min-max cut algorithm for graph partitioning and data clustering, in: International Conference on Data Mining, IEEE, 2001, pp. 107–114.
[8] L.v.d. Maaten, G. Hinton, Visualizing data using t-sne, J. Mach. Learn. Res. 9 (2008) 2579–2605.
[9] A. Azran, The rendezvous algorithm: Multiclass semi-supervised learning with markov random walks, in: Proceedings of the 24th international conference on Machine learning, 2007, pp. 49–56.
[10] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, M. Aly, Video suggestion and discovery for youtube: taking random walks through the view graph, in: Proc. 17th int. conference on World Wide Web, 2008, pp. 895–904.
[11] S. Bhagat, I. Rozenbaum, G. Cormode, Applying link-based classification to label blogs, in: Proceedings of WebKDD: workshop on Web mining and social network analysis, ACM, 2007, pp. 92–101.
[12] Q. Lu, L. Getoor, Link-based classification, in: ICML, 3, 2003, pp. 496–503.
[13] P. Jaccard, Etude comparative de la distribution florale dans une portion des Alpes et du Jura, Impr. Corbaz, 1901.
[14] L.A. Adamic, E. Adar, Friends and neighbors on the web, Soc. Netw. 25 (3) (2003) 211–230.
[15] A. Clauset, C. Moore, M.E. Newman, Hierarchical structure and the prediction of missing links in networks, Nature 453 (7191) (2008) 98–101.
[16] H.C. White, S.A. Boorman, R.L. Breiger, Social structure from multiple networks. I. Blockmodels of roles and positions, Am. J. Sociol. 81 (4) (1976) 730–780.
[17] N. Friedman, L. Getoor, D. Koller, A. Pfeffer, Learning probabilistic relational models, in: IJCAI, 1999, pp. 1300–1309.
[18] D. Heckerman, C. Meek, D. Koller, Probabilistic entity-relationship models, prms, and plate models, Intro. Stat. Relational Learn. (2007) 201–238.
[19] Y. Zhou, H. Cheng, J.X. Yu, Graph clustering based on structural/attribute similarities, Proc. VLDB Endow. 2 (1) (2009) 718–729.
[20] J. Shi, J. Malik, Normalized cuts and image segmentation, IEEE Trans. Pattern Anal. Mach. Intell. 22 (8) (2000) 888–905.
[21] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, A.J. Smola, Distributed large-scale natural graph factorization, in: Proceedings of the 22nd international conference on World Wide Web, ACM, 2013, pp. 37–48.
[22] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, Line: large-scale information network embedding, in: Proceedings 24th International Conference on World Wide Web, 2015, pp. 1067–1077.
[23] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 1225–1234.
[24] M. Ou, P. Cui, J. Pei, Z. Zhang, W. Zhu, Asymmetric transitivity preserving graph embedding, in: Proc. of ACM SIGKDD, 2016, pp. 1105–1114.
[25] M. Belkin, P. Niyogi, Laplacian eigenmaps and spectral techniques for embedding and clustering, in: NIPS, 14, 2001, pp. 585–591.
[26] S.T. Roweis, L.K. Saul, Nonlinear dimensionality reduction by locally linear embedding, Science 290 (5500) (2000) 2323–2326.
[27] S. Cao, W. Lu, Q. Xu, Grarep: learning graph representations with global structural information, in: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, ACM, 2015, pp. 891–900.
[28] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: online learning of social representations, in: Proceedings 20th international conference on Knowledge discovery and data mining, 2014, pp. 701–710.
[29] A. Grover, J. Leskovec, node2vec: scalable feature learning for networks, in: Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 855–864.
[30] S. Cao, W. Lu, Q. Xu, Deep neural networks for learning graph representations, in: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI Press, 2016, pp. 1145–1152.
[31] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv:1609.02907(2016).
[32] D. Luo, F. Nie, H. Huang, C.H. Ding, Cauchy graph embedding, in: Proceedings of the 28th International Conference on Machine Learning (ICML-11), 2011, pp. 553–560.
[33] B. Shaw, T. Jebara, Structure preserving embedding, in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 937–944.
[34] C.F. Van Loan, Generalizing the singular value decomposition, SIAM J. Numer. Anal. 13 (1) (1976) 76–83.
[35] S. Yan, D. Xu, B. Zhang, H.-J. Zhang, Q. Yang, S. Lin, Graph embedding and extensions: a general framework for dimensionality reduction, IEEE Trans. Pattern Anal. Mach. Intell. 29 (1) (2007) 40–51.
[36] I.T. Jolliffe, Principal component analysis and factor analysis, in: Principal component analysis, Springer, 1986, pp. 115–128.
[37] A.M. Martínez, A.C. Kak, Pca versus lda, IEEE Trans. Pattern Anal. Mach. Intell. 23 (2) (2001) 228–233.
[38] J.B. Tenenbaum, V. De Silva, J.C. Langford, A global geometric framework for nonlinear dimensionality reduction, Science 290 (5500) (2000) 2319–2323.
[39] J.B. Kruskal, M. Wish, Multidimensional scaling, 11, Sage, 1978.
[40] X. He, P. Niyogi, Locality preserving projections, in: Advances in neural information processing systems, 2004, pp. 153–160.
[41] M. Brand, Continuous nonlinear dimensionality reduction by kernel eigenmaps, in: IJCAI, 2003, pp. 547–554.
[42] A.M. Martínez, A.C. Kak, Non-negative graph embedding, IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR) 23 (2) (2008) 1–8.
[43] Y.-Y. Lin, T.-L. Liu, H.-T. Chen, Semantic manifold learning for image retrieval, in: Proceedings of the 13th annual ACM international conference on Multimedia, ACM, 2005, pp. 249–258.
[44] C. Yang, Z. Liu, D. Zhao, M. Sun, E.Y. Chang, Network representation learning with rich text information., in: IJCAI, 2015, pp. 2111–2117.
[45] S. Chang, W. Han, J. Tang, G.-J. Qi, C.C. Aggarwal, T.S. Huang, Heterogeneous network embedding via deep architectures, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2015, pp. 119–128.
[46] C. Tu, W. Zhang, Z. Liu, M. Sun, Max-margin deepwalk: discriminative learning of network representation., in: IJCAI, 2016, pp. 3889–3895.
[47] D. Zhang, J. Yin, X. Zhu, C. Zhang, Homophily, structure, and content augmented network representation learning, in: Data Mining (ICDM), 2016 IEEE 16th International Conference on, IEEE, 2016, pp. 609–618.
[48] X. Huang, J. Li, X. Hu, Label informed attributed network embedding, in: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, ACM, 2017, pp. 731–739.
[49] M.E. Newman, A measure of betweenness centrality based on random walks, Soc. Netw. 27 (1) (2005) 39–54.
[50] F. Fouss, A. Pirotte, J.-M. Renders, M. Saerens, Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation, IEEE Trans. Knowl. Data Eng. 19 (3) (2007).
[51] H. Chen, B. Perozzi, Y. Hu, S. Skiena, Harp: hierarchical representation learning for networks, arXiv:1706.07845(2017).
[52] B. Perozzi, V. Kulkarni, S. Skiena, Walklets: multiscale graph embeddings for interpretable network classification, arXiv:1605.02115(2016).
[53] Z. Yang, J. Tang, W.W. Cohen, Multi-modal bayesian embeddings for learning social knowledge graphs., in: IJCAI, 2016, pp. 2287–2293.
[54] J. Li, J. Zhu, B. Zhang, Discriminative deep random walk for network classification., ACL (1), 2016.
[55] S. Pan, J. Wu, X. Zhu, C. Zhang, Y. Wang, Tri-party deep network representation, Network 11 (9) (2016) 12.
[56] Z. Yang, W.W. Cohen, R. Salakhutdinov, Revisiting semi-supervised learning with graph embeddings, arXiv:1603.08861(2016).
[57] M. Niepert, M. Ahmed, K. Kutzkov, Learning convolutional neural networks for graphs, in: Proceedings of the 33rd annual international conference on machine learning. ACM, 2016.
[58] Y. Bengio, A. Courville, P. Vincent, Representation learning: a review and new perspectives, IEEE Trans. Pattern Anal. Mach. Intell. 35 (8) (2013) 1798–1828.
[59] J. Bruna, W. Zaremba, A. Szlam, Y. LeCun, Spectral networks and locally connected networks on graphs, arXiv:1312.6203(2013).
[60] M. Henaff, J. Bruna, Y. LeCun, Deep convolutional networks on graph-structured data, arXiv:1506.05163(2015).
[61] D.K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, R.P. Adams, Convolutional networks on graphs for learning molecular fingerprints, in: Advances in neural information processing systems, 2015, pp. 2224–2232.
[62] Y. Li, D. Tarlow, M. Brockschmidt, R. Zemel, Gated graph sequence neural networks, arXiv:1511.05493(2015).
[63] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, in: Advances in Neural Information Processing Systems, 2016, pp. 3844–3852.
[64] W.L. Hamilton, R. Ying, J. Leskovec, Inductive representation learning on large graphs, arXiv:1706.02216(2017).
[65] T.N. Kipf, M. Welling, Variational graph auto-encoders, arXiv:1611.07308(2016).
[66] D.P. Kingma, M. Welling, Auto-encoding variational bayes, arXiv:1312.6114(2013).
[67] K. Hornik, M. Stinchcombe, H. White, Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, Neural Netw. 3 (1990) 551–560.
[68] T. Feder, R. Motwani, Clique partitions, graph compression and speeding-up algorithms, in: Proceedings of the twenty-third annual ACM symposium on Theory of computing, 1991, pp. 123–133.
[69] P.M. Pardalos, J. Xue, The maximum clique problem, J. Global Optim. 4 (3) (1994) 301–328.
[70] Y. Tian, R.A. Hankins, J.M. Patel, Efficient aggregation for graph summarization, in: Proceedings of the SIGMOD international conference on Management of data, ACM, 2008, pp. 567–580.
[71] H. Toivonen, F. Zhou, A. Hartikainen, A. Hinkka, Compression of weighted graphs, in: Proc. 17th international conference on Knowledge discovery and data mining, 2011, pp. 965–973.
[72] S. Navlakha, R. Rastogi, N. Shrivastava, Graph summarization with bounded error, in: Proceedings of the international conference on Management of data, ACM, 2008, pp. 419–432.
[73] J. Rissanen, Modeling by shortest data description, Automatica 14 (5) (1978) 465–471.

[74] D. Jungnickel, T. Schade, Graphs, networks and algorithms, Springer, 2005.
[75] E.R. Gansner, S.C. North, An open graph visualization system and its applications to software engineering, Softw. Pract. Exp. 30 (11) (2000) 1203–1233.
[76] G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, Algorithms for drawing graphs: an annotated bibliography, Comput. Geom. 4 (5) (1994) 235–282.
[77] P. Eades, L. Xuemin, How to draw a directed graph, in: Visual Languages, 1989., IEEE Workshop on, IEEE, 1989, pp. 13–17.
[78] I. Herman, G. Melançon, M.S. Marshall, Graph visualization and navigation in information visualization: a survey, IEEE Trans. Visual. Comput. Graph. 6 (1) (2000) 24–43.
[79] K. Pearson, Liii. on lines and planes of closest fit to systems of points in space, Lond., Edinburgh, Dublin Philos. Mag. J. Sci. 2 (11) (1901) 559–572.
[80] M.E. Newman, M. Girvan, Finding and evaluating community structure in networks, Phys. Rev. E 69 (2) (2004) 026113.
[81] X. Xu, N. Yuruk, Z. Feng, T.A. Schweiger, Scan: a structural clustering algorithm for networks, in: Proceedings 13th international conference on Knowledge discovery and data mining, 2007, pp. 824–833.
[82] S. White, P. Smyth, A spectral clustering approach to finding communities in graphs, in: Proceedings of the 2005 SIAM international conference on data mining, SIAM, 2005, pp. 274–285.
[83] L. Lü, T. Zhou, Link prediction in complex networks: a survey, Physica A 390 (6) (2011) 1150–1170.
[84] M. Al Hasan, M.J. Zaki, A survey of link prediction in social networks, in: Social network data analytics, 2011, pp. 243–275.
[85] L. Katz, A new status index derived from sociometric analysis, Psychometrika 18 (1) (1953) 39–43.
[86] K. Yu, W. Chu, S. Yu, V. Tresp, Z. Xu, Stochastic relational models for discriminative link prediction, in: NIPS, 2006, pp. 1553–1560.
[87] J. Neville, D. Jensen, Iterative classification in relational data, in: Proc. Workshop on Learning Statistical Models from Relational Data, 2000, pp. 13–20.
[88] D.W. Hosmer Jr, S. Lemeshow, R.X. Sturdivant, Applied logistic regression, 398, John Wiley & Sons, 2013.
[89] A. McCallum, K. Nigam, et al., A comparison of event models for naive bayes text classification, in: AAAI-98 workshop on learning for text categorization, 752, Citeseer, 1998, pp. 41–48.
[90] Y.J. Wang, G.Y. Wong, Stochastic blockmodels for directed graphs, J. Am. Stat. Assoc. 82 (397) (1987) 8–19.
[91] W.W. Zachary, An information flow model for conflict and fission in small groups, J. Anthropol. Res. 33 (4) (1977) 452–473.
[92] L. Tang, H. Liu, Relational learning via latent social dimensions, in: Proceedings of the 15th international conference on Knowledge discovery and data mining, ACM, 2009, pp. 817–826.
[93] L. Tang, H. Liu, Scalable learning of collective behavior based on sparse social dimensions, in: Proceedings of the 18th ACM conference on Information and knowledge management, ACM, 2009, pp. 1107–1116.
[94] J. Gehrke, P. Ginsparg, J. Kleinberg, Overview of the 2003 kdd cup, ACM SIGKDD Expl. 5 (2) (2003).
[95] J. Leskovec, A. Krevl, SNAP datasets: Stanford large network dataset collection, 2014, (http://snap.stanford.edu/data).
[96] B.-J. Breitkreutz, C. Stark, T. Reguly, L. Boucher, A. Breitkreutz, M. Livstone, R. Oughtred, D.H. Lackner, J. Bähler, V. Wood, et al., The biogrid interaction database: 2008 update, Nucleic Acids Res. 36 (suppl 1) (2008) D637–D640.
[97] H. Dai, Y. Wang, R. Trivedi, L. Song, Deep coevolutionary network: embedding user and item features for recommendation (2017).
[98] P. Goyal, N. Kamra, X. He, Y. Liu, Dyngem: deep embedding method for dynamic graphs.
[99] L. Zhu, D. Guo, J. Yin, G. Ver Steeg, A. Galstyan, Scalable temporal latent space inference for link prediction in dynamic social networks, IEEE Trans. Knowl. Data Eng. 28 (10) (2016) 2765–2777.
[100] P.W. Holland, K.B. Laskey, S. Leinhardt, Stochastic blockmodels: first steps, Soc. Netw. 5 (2) (1983) 109–137.
[101] K. Riesen, M. Neuhaus, H. Bunke, Graph embedding in vector spaces by means of prototype selection, in: International Workshop on Graph-Based Representations in Pattern Recognition, Springer, 2007, pp. 383–393.
[102] J. Wright, Y. Ma, J. Mairal, G. Sapiro, T.S. Huang, S. Yan, Sparse representation for computer vision and pattern recognition, Proceedings of the IEEE 98 (6) (2010) 1031–1044.
[103] H. Bunke, K. Riesen, Recent advances in graph-based pattern recognition with applications in document analysis, Pattern Recognition. 44 (5) (2011) 1057–1067.
[104] W.L. Hamilton, R. Ying, J. Leskovec, Representation learning on graphs: methods and applications, arXiv preprint arXiv:1709.05584 (2017).
[105] H. Cai, V.W. Zheng, K.C.-C. Chang, A comprehensive survey of graph embedding: problems, techniques and applications, arXiv preprint arXiv:1709.07604 (2017).