Augmented Network Embedding in Attributed Graphs

Daokun Zhang

Faculty of Engineering and Information Technology
University of Technology Sydney

This dissertation is submitted for the degree of
Doctor of Philosophy

August 2019

I would like to dedicate this thesis to my loving parents.

Certificate of Original Authorship

I, Daokun Zhang declare that this thesis, is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise reference or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Production Note:
Signature: Signature removed prior to publication.

Date: 30 Aug, 2019

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Doctor Jie Yin, Professor Xingquan Zhu and Professor Chengqi Zhang, for their selfless devotion to supporting and cultivating me in past three and a half years.

I would like to acknowledge my external supervisor Doctor Jie Yin, who provided me with a valuable opportunity to study at CSIRO (Commonwealth Scientific and Industrial Research Organisation) as a visiting student. As my supervisor, Doctor Yin worked responsibly and patiently to train me in all aspects of my PhD study, from research idea conceiving and technical implementation to paper writing and presentation skill developing. From her, I learned not only the essential skills for independent scientific research, but also the positive attitude as a responsible researcher, which both will constantly benefit me throughout my research career.

I would also like to express my thanks to my principal supervisor Professor Chengqi Zhang, who offered me an unique opportunity to study at the Centre for Artificial Intelligence, UTS, where I was able to network with and learn from so many talented colleagues. From the beginning scholarship application to the final thesis submission, Professor Zhang constantly gave me great support. His warming support cheered me up to behave actively in the face of stress and hardships during my PhD life. His wisdom also inspired me in how to overcome the anxiety caused by peer pressure and try my best to make progress.

I would also like to thank my external supervisor Professor Xingquan Zhu, who encouraged me to pursue PhD study. Before and during my PhD study, Professor Zhu gave me great encouragement to aim high-quality research. Even though through Skype and E-mail, I was so glad to communicate with him to discuss our research work, which was really an encouraging and eye-opening experience. His optimism, and

passion in solving challenging research problems deeply affected me, and will encourage me to make research progress endlessly.

I want to thank my academic brother Doctor Shirui Pan, who recommended me to Professor Xingquan Zhu. During my PhD study, Doctor Pan offered me a lot of generous help, not only in my study but also in my life, especially in the early stage of my PhD life. He provided many valuable suggestions to help me start up my research. He also helped me to adapt to the oversea student life.

I am delighted to thank my friends, colleagues, housemates. We shared our life, laughter and tears, which made my oversea student life much colorful. Because of them, I no longer felt lonely in Australia. They gave me great support and comfort when I was in trouble. I was so glad to know and experience all of them.

I acknowledge the financial support from CSC-UTS Scholarship, CSIRO Top-up Scholarship, CIKM Travel Award (2016), ICDM Travel Award (2016 and 2018). Without these financial supports, I couldn't have finished my PhD study.

Finally, above all, I want to show my special thanks to my family: my parents, my brother, my sister-in-law and my little niece. Because of them, my every effort became meaningful. I especially thank my parents for their self-sacrifice and endless love. They always forgave, understood, supported me unconditionally. To them, I dedicate this dissertation.

Abstract

With the widespread use of information technologies, information networks are becoming increasingly popular to capture complex relationships across various disciplines, such as social networks, citation networks, telecommunication networks, and biological networks. Analyzing these networks sheds light on different aspects of social life, such as the structure of societies, information diffusion, and communication patterns. In reality, however, the large scale of information networks often makes network analytic tasks computationally expensive or intractable. Network embedding has been recently proposed as a new learning paradigm to embed network nodes into a low-dimensional vector space. This facilitates the original network to be easily handled in the new vector space for further analysis. Existing research on network embedding mainly focuses on capturing the structure relatedness in the embedding space, while ignores the important information carried by the widely existing node attributes and labels, which limited the network embedding performance significantly. In this thesis, we dealt with the research problem of augmented network embedding in attributed graphs that aims to learn informative node vector-format representations by augmenting network topology structure with node content attributes and node labels if available. We summarized four research challenges in augmented network embedding: (1) *heterogeneity* caused by the discrepancy between network structure and node attributes/labels; (2) *data sparsity* in network structure and node attributes/labels; (3) *scalability* for handling large-scale networks; (4) *task orientation* for directly benefiting specific network analytic tasks.

To overcome the above challenges, we proposed a series of augmented network embedding algorithms in this thesis. To handle the *heterogeneity* challenge, we proposed the HSCA algorithm that effectively encodes the similarity measured by homophily, structural context and node content attributes into a unified node representation

through the regularized inductive matrix factorization. The attri2vec algorithm was then proposed to address the *heterogeneity* and *data sparsity* challenges, in which node representations are learned by discovering an attribute subspace that better respects network structure. For handling large-scale incomplete networks, we proposed the SINE algorithm that learns node representations by simultaneously modeling node-neighbor and node-attribute relations through a three-layer neural network, with an efficient Stochastic Gradient Descent based online learning strategy. The above three augmented network embedding algorithms only augment network structure with node content attributes, with the purpose to obtain more informative network representations. They are unsupervised, task-general and incapable of directly benefiting specific tasks. To seamlessly integrate network embedding with network analytic tasks, we proposed two task-orientated network embedding algorithms. For collective classification on sparsely labeled networks, we proposed the discriminative attributed network embedding algorithm DMF that integrates network embedding with an empirical loss minimization for classifying node labels, with the purpose of simultaneously exerting the discriminative power of node labels and informativeness of node representations. For searching similar nodes efficiently on large-scale networks, BinaryNE was proposed to learn binary node representations from network structure and node content attributes so that node similarity search can be efficiently done through the fast bitwise Hamming distance calculation performed on the learned binary node representations. To verify the effectiveness of the proposed algorithms, extensive experiments were carried out on nine real-world attributed networks, showing the advantage of the proposed algorithms over state-of-the-art baselines.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Background and Motivation

Information networks are becoming ubiquitous across a large spectrum of real-world applications in forms of social networks, citation networks, telecommunication networks and biological networks, *etc.* The scale of these networks ranges from hundreds to millions or even billions of nodes [76]. Analyzing information networks plays a crucial role in a variety of emerging applications across many disciplines. For example, in social networks, classifying users into meaningful social groups is useful for many important tasks, such as user search, targeted advertising and recommendations; in communication networks, detecting community structures can help better understand the rumor spreading process; in biological networks, inferring interactions between proteins can facilitate new treatments for diseases. Nevertheless, efficient analysis of these networks heavily relies on the ways how networks are represented. Often, a discrete adjacency matrix is used to represent a network, which only captures neighboring relationships between nodes. Indeed, this simple representation cannot embody more complex, higher-order structure relationships, such as paths, frequent substructure *etc.* As a result, such a traditional routine often makes many network analytic tasks computationally expensive and intractable over large-scale networks. Taking community detection as an example, most existing algorithms involve calculating the spectral decomposition of a matrix [50] with at least quadratic time complexity

with respect to the number of nodes. This computational overhead makes algorithms hard to scale to large-scale networks with millions of nodes.

Recently, network embedding has aroused a lot of research interest. Network embedding aims to learn latent, low-dimensional representations of network nodes, while preserving network topology structure, and/or node content. After new node representations are learned, network analytic tasks can be easily and efficiently carried out by applying conventional vector-based machine learning algorithms to the new representation space. This obviates the necessity for deriving complex algorithms that are applied directly on the original networks.

Earlier work related to network embedding dates back to the early 2000s, when researchers proposed graph embedding algorithms as part of dimensionality reduction techniques. Given a set of *i.i.d.* (independent and identically distributed) data points as input, graph embedding algorithms first calculate the similarity between pairwise data points to construct an affinity graph, *e.g.*, the k-nearest neighbor graph, and then embed the affinity graph into a new space having much lower dimensionality. The idea is to find a low-dimensional manifold structure hidden in the high-dimensional data geometry reflected by the constructed graph, so that connected nodes are kept closer to each other in the new embedding space. Isomap [80], Locally Linear Embedding (LLE) [67] and Laplacian Eigenmap [4] are examples of algorithms based on this rationale. However, graph embedding algorithms are designed on *i.i.d.* data mainly for dimensionality reduction purpose. Most of these algorithms usually have at least quadratic time complexity with respect to the number of nodes, so the scalability is a major issue when they are applied to large-scale networks.

Since 2008, significant research efforts have shifted to the development of effective and scalable representation learning techniques that are directly designed for complex information networks, where network nodes are usually attached with content attributes, such as user profiles in social networks, texts in Web page networks. Many network embedding algorithms, *e.g.*, [63, 96, 100, 9], have been proposed to embed the real-world natural networks, showing promising performance for various applications. These algorithms embed a network into a latent, low-dimensional space that preserves structure proximity and attribute affinity, such that the original nodes of the network

can be represented as low-dimensional vectors. The resulting compact, low-dimensional vector representations can be then taken as features to any vector-based machine learning algorithms. This paves the way for a wide range of network analytic tasks to be easily and efficiently tackled in the new vector space, such as node classification [112, 5], link prediction [47, 18], clustering [50], recommendation [99, 95], similarity search [46], and visualization [75]. Using vector representation to represent complex networks has now been gradually advanced to many other domains, such as point-of-interest recommendation in urban computing [95], and knowledge graph search [45] in knowledge engineering and database systems.

Existing network embedding algorithms main focus on leveraging network structure to learn node representations so that network nodes with strong structural relatedness will be represented closely in the new embedding space. However, in many cases, network structure is noisy, sparse and uninformative for measuring node similarity. As a result, only structure preserving network embedding algorithms are incapable of learning high-quality node representations for benefiting the downstream network analytic tasks. In addition to network structure, the widely existing node content attributes that provide direct evidence to measure node content-level similarity can be leveraged for more effective network embedding. Attributed network embedding aims to learn informative node representations for attributed networks through the collaboration between network structure and node content attributes. As more information is imported, it has been proved that attributed network embedding algorithms are able to perform better than the only structure preserving network embedding algorithms.

1.2 Research Challenges

Despite its great potential, attributed network embedding is inherently difficult and is confronted with several key challenges that we summarize as follows.

- **Heterogeneity**: According to the social influence principle, network structure often exhibits certain correlations with node attributes [65]. For example, in social networks, users who are connected are more likely to have similar attributes. However, because network structure and node attributes are two heterogeneous

information sources, inevitably, there is a discrepancy between the two different feature spaces. As has been revealed in previous studies [7, 74], social network users with similar attributes, or antithetical attributes are both likely to connect to one another.

- **Data sparsity**: For many real-world information networks, due to the privacy or legal restrictions, the problem of data sparsity exists in both network structure and node content. At the structure level, only very limited links are sometimes observed, making it difficult to discover the structure-level relatedness between nodes that are not explicitly connected. At the node content level, many values of node attributes are usually missing, which increases the difficulty of measuring content-level node similarity. Thus, it is challenging for network embedding to overcome the data sparsity problem.

- **Scalability**: Real-world networks, social networks in particular, consist of millions or billions of nodes and edges, as well as high-dimensional node content features. The large scale of the networks challenges not only the traditional network analytic tasks but also the newborn network embedding task. Without special concern, learning node representations for large-scale networks with limited computing resources may cost months of time, which is practically infeasible, especially for the case involving a large number of trails for tuning parameters. Therefore, it is necessary to design network embedding algorithms that can learn node representations efficiently and meanwhile guarantee the effectiveness for large-scale networks.

- **Task orientation**: Most attributed network embedding is conducted in an unsupervised and task-general way. The network embedding process is independent of the downstream network analytics tasks. Though the learned node representations can be generalize to various network analytic tasks, they are incapable of directly benefiting the downstream tasks. To improve the performance of network embedding based network analysis, it is necessary to make network embedding directly serve the targeted network analytic tasks. However, network embedding and the downstream network analysis are two heterogeneous tasks.

How to encode the objective of specific tasks into network embedding and learn task oriented node representations is another challenge.

1.3 Research Problems

In this thesis, we explore two groups of attributed network embedding problems: task-general attributed network embedding that purely learns node representations and does not consider the downstream network analytic tasks, and task-general attributed network embedding that learns node representations with the objective of directly benefiting specific tasks.

1.3.1 Task-general Attributed Network Embedding

Task-general attributed network embedding learns node vector-format representations in an unsupervised way. The learned node representations are able to generalize to diverse network analytic tasks. Keeping the **heterogeneity**, **data sparsity**, and **scalability** research challenge in mind, we study three research problems by handling increasing research challenges:

- **Homophily, Structure, and Content Augmented Network Embedding**: From literatures, we summarize that three are three main information sources for network embedding: (1) *homophily*, (2) *structural context* and (3) *node content*. Homophily states the social phenomenon where individuals sharing similar attributes (content) tend to be directly connected through relational ties, while structural context captures more global structure by characterizing the similarity between nodes sharing common direct or indirect neighbors. To ensure effective network embedding, this research aims to augment the three information sources into one learning objective, so that the interplay roles between three parties are enforced by requiring the learned node representations (1) being consistent with structure context and node content, and (2) following the social homophily constraints in the learned space.

- **Subspace Discovery based Attributed Network Embedding**: In reality, networks often have sparse content, incomplete node attributes, as well as the discrepancy between node attribute feature space and network structure space, which severely deteriorates the performance of existing methods. This research aims to design a unified framework for attributed network embedding that learns node embeddings by discovering a latent node attribute subspace via a network structure guided transformation performed on the original attribute space. The resultant latent subspace can respect network structure in a more consistent way towards learning high-quality node representations.

- **Scalable Incomplete Network Embedding**: In the big data era, real-world information networks are usually on a large scale. Meanwhile, real-world networks are often incomplete with missing edges and/or missing node content features, due to various reasons, such as privacy/legal restrictions and data access difficulty. However, existing attributed network embedding algorithms all operate under the assumption that networks are complete. Thus, their performance is vulnerable to missing data and suffers from poor scalability. This research aims to propose a scalable incomplete network embedding algorithm for learning informative node representations from incomplete graphs, with the expectation that the proposed algorithm is able to efficiently handle large-scale attributed networks with a large number nodes/edges, and high-dimensional node attributes.

1.3.2 Task-dependent Attributed Network Embedding

Task-dependent attributed network embedding aims to make network embedding directly benefit the targeted network analytic tasks. Keeping two network analytic tasks – collective classification and node similarity search – in mind, we study two task-dependent attributed network embedding problems:

- **Discriminative Attributed Network Embedding for Collective Classification**: Collective classification aims to classify network nodes into several predefined groups by making joint predictions on the labels of connected nodes. Collective classification can benefit many important tasks in social networks,

such as fraud detection, targeted advertising and recommendations. Existing collective classification algorithms have been shown to be effective for for jointly deriving labels of related nodes, by exploiting class label dependencies among neighboring nodes. However, when the underlying network is sparsely labeled, most nodes have too few or even no connections to labeled nodes. This makes it very difficult to leverage supervised knowledge from labeled nodes to accurately estimate label dependencies, thereby largely degrading the classification accuracy. To address this difficulty, this research aims to propose a novel collective classification algorithm based on discriminative network embedding that simultaneously learns (1) node representations by exploiting topological paths between labeled and unlabeled nodes, as well as node content attributes, and (2) a linear classifier that classifies node representations into different groups. By unleashing the discriminative power of labeled nodes through an empirical loss minimization objective, the learned node representations are expected to be directly benefit node classification with sparse labels.

- **Binary Attributed Network Embedding for Efficient Search**: Traditional network embedding primarily focuses on learning a dense vector representation for each node, which encodes network structure and/or node content information, such that off-the-shelf machine learning algorithms can be easily applied to the vector-format node representations for network analysis. However, the learned dense vector representations are inefficient for large-scale similarity search, which requires to find the nearest neighbor measured by Euclidean distance in a continuous vector space. This research aims to propose a search efficient binary network embedding algorithm to learn a sparse binary code for each node, by leveraging both network structure and node content. The learned binary encoding is expected to not only reduce memory usage to represent each node, but also allow fast bitwise comparisons to support much quicker network node search compared to Euclidean distance or other distance measures.

1.4 Thesis Contributions

This thesis proposes a series of solutions to attributed network embedding, including task-general attributed network embedding and task-dependent attributed network embedding. For task-general attributed network embedding, we address the **heterogeneity**, **data sparsity** and **scalability** challenges and design state-of-the-art algorithms. For task-dependent attributed network embedding, we endow attributed network embedding with the power to significantly boost network analysis, and provide a revolutionized way to leverage network embedding to solve real-world problems.

1.4.1 Task-general Attributed Network Embedding

- **Homophily, Structure, and Content Augmented Network Embedding**:
  - *Contribution*:
    - Through formulating a new matrix factorization problem, we propose a novel attributed network embedding algorithm – HSCA – that seamlessly integrate information from three sources (a) homophily, (b) structural context and (c) node content into learning informative node representations.
    - We design a new optimization objective function for HSCA and derive an efficient algorithm based on conjugate gradient descent for solving it.
    - We evaluate the performance of the HSCA algorithm on four real-world networks for the task of multi-class classification. Experimental results show that HSCA achieves better classification performance than the state-of-the-art network embedding algorithms.
  - *Outcome*:
    - The proposed HSCA algorithm was published in ICDM-2016 [101].
    - The Matlab implementation of HSCA is available at https://github.com/daokunzhang/HSCA.

- **Subspace Discovery based Attributed Network Embedding**:
  - *Contribution*:
    * We propose a unified framework for attributed social network embedding, attri2vec, that learns node embeddings by discovering a latent, structure aware attribute subspace and formulate an optimization problem.
    * We develop an efficient stochastic gradient descent algorithm to solve the optimization problem, which not only remedies the local minima problem, but also improves the effectiveness and efficiency, thus making attri2vec able to scale to large-scale networks.
    * We validate the effectiveness and efficiency of attri2vec on four real-world networks of various types. Extensive experiments on multi-class node classification, node clustering and link prediction tasks demonstrate that attri2vec learns node embeddings of better quality than state-of-the-art baselines, with an additional advantage to solve the out-of-sample problem.
  - *Outcome*:
    * The first version of attri2vec, UPP-SNE (User Profile Preserving Social Network Embedding), was published in IJCAI-2017 [102].
    * The C implementation of UPP-SNE is available at https://github.com/daokunzhang/UPPSNE.
    * We then extended UPP-SNE to attri2vec, a unified framework for attributed network embedding. The extended version is accepted by DMKD [106].
    * The C implementation of attri2vec is available at https://github.com/daokunzhang/attri2vec.
- **Scalable Incomplete Network Embedding**:
  - *Contribution*:
    * We advance the existing attributed network embedding learning to a realistic missing data setting, allowing embedding leaning to be highly efficient and accurate for real-world networks.

    - We propose a scalable and efficient algorithm – SINE – to combine network structure and node attributes to learn a joint embedding representation, thereby diminishing negative impacts of missing information.
    - We evaluate the effectiveness of the proposed method under different missing data settings, showing its superior performance to the state-of-the-art baselines.
  - *Outcome*:
    - The proposed SINE algorithm was published in ICDM-2018 [105].
    - The C implementation of SINE is available at https://github.com/daokunzhang/SINE.

1.4.2 Task-dependent Attributed Network Embedding

- **Discriminative Attributed Network Embedding for Collective Classification**:
  - *Contribution*:
    - We study the label sparsity problem in network settings that many real-world applications often encounter, but traditional algorithms are ineffective to address.
    - We propose a unified matrix factorization framework for collective classification with label sparsity – DMF – that seamlessly integrates attributed network embedding and discriminative learning into a new optimization problem, and derive solutions with high efficiency.
    - We compare DMF with four state-of-the-art collective classification baselines on three real-world information networks, showing the advantage of DMF in handling label sparsity.
  - *Outcome*:
    - The DMF algorithm was published in CIKM-2016 [100].
    - The Matlab implementation of DMF is available at https://github.com/daokunzhang/DMF_CC.

- **Binary Attributed Network Embedding for Efficient Search**:
  - *Contribution*:
    - We analyze the feasibility and advantage of learning binary node representations as a solution to efficient node similarity search over large-scale networks.
    - We propose the BinaryNE algorithm to effectively learn high-quality binary node representations from both network structure and node content features.
    - We conduct experiments to compare search performance of the BinaryNE algorithm and other network embedding algorithms, showing the superiority of BinaryNE in terms of memory usage and search time.
  - *Outcome*:
    - The BinaryNE algorithm is under review by TKDE [107].

1.5 Thesis Overview

According to the category of attributed network embedding, the body of this thesis is structured in two parts: task-general attributed network embedding and task-dependent attributed network embedding. Table 1.1 summarizes the overall structure of this thesis. The detailed roadmap of the thesis is summarized as follows:

- **Chapter 2**: Before formally addressing research problems, in this Chapter, we conduct a comprehensive literature review on related research work, including network embedding, collective classification and node similarity search.
- **Chapter 3**: As preliminaries, this Chapter provides the definitions of important terminologies and the research problem of network embedding, as well as the summarization of benchmark datasets used in this thesis.
- **Part I**: In this Part, we address the research problem of task-general attributed network embedding. Chapters 4-6 respectively detail the solutions to the research problems (1) Homophily, Structure, and Content Augmented Network Embedding,

Table 1.1 Thesis structure

Part	Research Task	Chapter	Publication
—	Literature Review	Chapter 2	[104]
	Preliminaries	Chapter 3	—
Part I Task-general Attributed Network Embedding	Homophily, Structure, and Content Augmented Network Embedding	Chapter 4	[101]
	Subspace Discovery Based Attributed Network Embedding	Chapter 5	[102, 106]
	Scalable Incomplete Network Embedding	Chapter 6	[105]
Part II Task-dependent Attributed Network Embedding	Discriminative Network Embedding for Collective Classification	Chapter 7	[100]
	Binary Network Embedding for Efficient Search	Chapter 8	[107]
—	Conclusion and Future Work	Chapter 9	—

(2) Subspace Discovery based Attributed Network Embedding, and (3) Scalable Incomplete Network Embedding.

- **Part II**: In this Part, we study how to tailor network embedding to directly benefit specific network analytic tasks. In Chapter 7 and 8, we respectively study two task-dependent attributed network embedding problems: (1) Discriminative Attributed Network Embedding for Collective Classification and (2) Binary Attributed Network Embedding for Efficient Search.
- **Chapter 9**: Finally, we conclude this thesis and point out future research directions.

1.6 Publications

Below are the papers finished during my PhD study, which have been submitted, accepted and published:

- **Journal Publications**:
  1. **Daokun Zhang**, Jie Yin, Xingquan Zhu, and Chengqi Zhang, Network Representation Learning: A Survey, *IEEE Transactions on Big Data (**TBD**)*, **accepted**, 2018. [104]

2. **Daokun Zhang**, Jie Yin, Xingquan Zhu, and Chengqi Zhang, Attributed Network Embedding via Subspace Discovery, *Data Mining and Knowledge Discovery (**DMKD**)*, **accepted**. [106]
3. **Daokun Zhang**, Jie Yin, Xingquan Zhu, and Chengqi Zhang, Search Efficient Binary Network Embedding, *IEEE Transactions on Knowledge and Data Engineering (**TKDE**)*, **under review**. [107]

- **Conference Publications**:

1. Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang, Collective Classification via Discriminative Matrix Factorization on Sparsely Labeled Networks, *ACM International Conference on Information and Knowledge Management (**CIKM**)*, pp. 1563-1572, 2016. [100]
2. Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang, Homophily, Structure, and Content Augmented Network Representation Learning, *IEEE International Conference on Data Mining (**ICDM**)*, pp. 609-618, 2016. [101]
3. Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang, User Profile Preserving Social Network Embedding, *International Joint Conference on Artificial Intelligence (**IJCAI**)*, pp. 3378-3384, 2017. [102]
4. Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang, MetaGraph2Vec: Complex Semantic Path Augmented Heterogeneous Network Embedding, *Pacific-Asia Conference on Knowledge Discovery and Data Mining (**PAKDD**)*, pp. 196-208, 2018. [103]
5. Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang, SINE: Scalable Incomplete Network Embedding, *IEEE International Conference on Data Mining (**ICDM**)*, pp. 737-746, 2018. [101]

Chapter 2

Literature Review

In this Chapter, we conduct a comprehensive review on network embedding. In addition, we review the research work related to the network analytic tasks, collective classification and node similarity search, which are concerned by this thesis for designing task-dependent attributed network embedding algorithms.

2.1 Network Embedding

Depending on whether node content features are leveraged, network embedding techniques can be divided into two groups [104]: *structure preserving network embedding* that leverages only network structure to learn node representations, and *attributed network embedding* that couples node content features with network structure to learn more informative node representations.

2.1.1 Structure Preserving Network Embedding

Early work on network embedding has attempted to partition a graph into different communities for creating social dimensions [77, 79, 78]. These methods usually cluster nodes into different affiliations. The representation for each node is determined by its membership with respect to different affiliations. Clustering algorithms used for this purpose include spectral clustering [86], modularity maximization [59], and

EdgeCluster [78]. These studies provide a cluster-centric view of the network, but ignore important local structure information of individual nodes.

Several network embedding algorithms have been proposed to capture node local neighborhood structure. DeepWalk [63] borrows the idea of the Skip-Gram model [55], originally designed for learning word representations using word context in sentences, to learn the representations of nodes in a network. In DeepWalk, short node sequences generated by random walks are taken as sentences, and node representations are learned using contextual information in random walk sequences. To leverage neighborhood structure more properly, node2vec [19] employs biased random walk to generate node sequences, which smoothly interpolates between two extreme neighborhood sampling strategies, Breadth-first Sampling (BFS) and Depth-first Sampling (DFS).

Different from the random walk based methods, LINE [76] explicitly defines a loss function to capture two types of node relationships, *i.e.* the first-order proximity that nodes linked to each other tend to be represented closely in the new embedding space, and the second-order proximity that nodes sharing direct common neighbors tend to have similar latent representations. LINE introduces two different models to capture 1-step and 2-step node relationships, respectively. The node representations learned by the two models are combined as the final node representations. The GraRep model [8] steps forward to consider different k-step proximity (in particular for $k > 2$), for learning network representations. However, both LINE and GraRep learn representations separately for k-step proximity and simply concatenate the learned vector features to form the final network representation. Such a simple combination inherently neglects the interplay between different types of proximity, including their relative importance, possibly leading to unsatisfactory performance.

In contrast to GraRep that preserves the symmetric node representations, HOPE [60] and APP [111] learn node representations that capture the asymmetric high-order proximity. In undirected networks, the transitivity is symmetric, but it is asymmetric in directed networks. To preserve the asymmetric transitivity, HOPE and APP learn two sets of node representations, *i.e.*, source and target representations. HOPE first constructs the high-order proximity matrix from one of the four proximity measures, Katz Index [30], Rooted PageRank [71], Common Neighbors and Adamic-Adar, then

factorizes it into source and target node representations. APP leverages random walks to capture the high-order proximity. Considering the direction of random walk sequences, APP learns asymmetric transitivity preserving node representations by using the source representations of starting nodes to predict the target representations of ending nodes.

Deep learning has also been imported to learn node representations from network structure. DNGR [9] first obtains high dimensional structure preserving node representations through the proposed random surfing method, and then utilizes the *stacked denoising autoencoder* (SDAE) [85] to learn low-dimensional representations. SDNE [87] employs deep autoencoder to learn deep nonlinear node representations, by reconstructing node adjacent matrix representations for preserving the second-order proximity and penalizing the representation difference of connected nodes for preserving the first-order proximity. GraphGAN [88] learns node representations by modeling node connectivity behavior through an adversarial learning framework, with two components: (i) generator $G(v|v_c)$, which fits the distribution of the nodes connected to v_c and generates the likely connected nodes, and (ii) discriminator $D(v, v_c)$, which outputs a connecting probability for the node pair (v, v_c), to differentiate the node pairs generated by $G(v|v_c)$ from the ground truth. Through the competition between the generator and discriminator, GraphGAN learns informative node representations that well respect node connectivity behavior.

The above network embedding methods only take advantage of local neighborhood structure. To learn more informative node representations, M-NMF [91] complements the local neighborhood structure with the community structure, by making the local structure proximity preserving representations consistent with the community indicative representations learned by modularity maximization [59].

Besides local connectivity patterns, nodes often share similar structural roles at a mesoscopic level, such as centers of stars or members of cliques. Structural role proximity preserving network embedding aims to embed network nodes that are far away from each other but share similar structural roles closely. This not only facilitates the downstream structural role dependent tasks, but also enhances local structure preserving network embedding. struct2vec [66] first encodes node structural

role similarity into a multilayer graph, where the weights of edges at each layer are determined by the structural role difference at the corresponding scale. DeepWalk [63] is then performed on the multilayer graph to learn node representations, such that nodes close to each other in the multilayer graph (with high structural role similarity) are embedded closely in the new representation space. By making use of the spectral graph wavelet diffusion patterns, GraphWave [14] embeds node neighborhood structure into a low-dimensional space and preserves the structural role proximity through the assumption that, if two nodes residing distantly in the network share similar structural roles, the graph wavelets starting at them will diffuse similarly across their neighbors. SNS [48] enhances a random walk based local structure proximity preserving method with structural role proximity. To measure node structural role similarity, SNS represents each node as a *Graphlet Degree Vector* with each element being the number of times the given node is touched by the corresponding orbit of graphlets.

The above structure preserving network embedding algorithms learn task-general node representations in an unsupervised way. Some research efforts have been made to enhance structure preserving network embedding with node labels, with the expectation that the learned node representations are discriminative for benefiting the targeted node classification task. DDRW [43] proposes to learn discriminative network representations through jointly optimizing the objective of DeepWalk [63] together with the L2-loss Support Vector Classification objective. Similarly, MMDW [82] couples the objective of the matrix factorization version DeepWalk [96] with the multi-class Support Vector Machine (SVM) objective. Along similar lines, TLINE [109] is proposed as a semi-supervised extension of LINE [76] that simultaneously learns LINE's node representations and an SVM classifier. GENE [11] integrates group (label) information with network structure in a probabilistic manner, with the assumption that nodes should be embedded closely in low-dimensional space, if they share similar neighbors or join similar groups. SemiNE [41] learns semi-supervised node representations in two stages. In the first stage, SemiNE exploits the DeepWalk [63] framework to learn node representations in an unsupervised manner. In the second stage, SemiNE learns a neural network that tunes the learned unsupervised node representations to fit node labels.

The *structure preserving network embedding* algorithms only leverage network structure to learn node representations, while neglects the important information in node content features. As a result, the learned node representations tend to be less informative.

2.1.2 Attributed Network Embedding

TADW [96] is the first attempt to incorporate rich node textual features into network embedding. TADW proves the equivalence between DeepWalk and a matrix factorization formulation, and then encodes node text features into the matrix factorization process for learning more informative node representations. pRBM [90] leverages the power of Restricted Boltzmann Machine (RBM) [24] to do attributed network embedding. pRMB learns node representations though applying RBM to node content attributes and simultaneously forcing the learned node representations of connected nodes to be similar to each other. The SNE algorithm [44] learns node embeddings for attributed networks through a neural network, in which, for each node, its embedding is aggregated from its ID embedding that captures the structural proximity and attribute embedding that captures the attribute proximity. MVC-DNE [97] applies deep autoencoder to node adjacent matrix representations and node content attributes respectively, and adopts cross-view learning to capture the interplay between network structure and node content and fuses information from the two sources into a unified embedding. GraphSAGE [22] first takes node content features as node representations, and then iteratively updates node representations by aggregating representations of neighboring nodes. PPNE [42] learns node representations for attributed networks through jointly optimizing two objectives: the *structure-driven objective* and the *attribute-driven objective*. The *structure-driven objective* inherits DeepWalk objective, aiming to make nodes sharing similar structural context nodes represented closely, while the *attribute-driven objective* aims to force node distributions in the new representations space to respect the similarity estimated from node content attributes. AANE [26] learns node representations by performing symetric matrix factorization [35] on attribute affinity matrix, and simultaneously minimizing the representation difference between connected nodes. However, the performance of existing attributed network embedding would be largely

degraded in real-world scenarios where node attributes are sparse and incomplete, and in particular, when there exists a discrepancy between network structure and node attributes.

In addition to the above unsupervised network embedding algorithms, there are also some research works on supervised network embedding. TriDNR [62] learns node representations for attributed networks with text features through a coupled neural network. To capture node content and label information, TriDNR couples DeepWalk with the Paragraph Vector model [37] that describes the node word relation and label-word correspondence. LDE [89] is proposed to learn representations for linked documents, which are actually the nodes of citation or webpage networks. Similar to TriDNR, LDE learns node representations by modeling three kinds of relations, *i.e.*, word-word-document relations, document-document relations, and document-label relations. LANE [27] learns node representations for attributed networks by embedding the network structure proximity, attribute affinity, and label proximity into a unified latent representation. Planetoid [98] maps network embedding vectors and node attribute vectors into a hidden layer space via deep neural network, and uses the mapped vectors to predict node labels, with network embedding vectors obtained through minimizing the loss for structural context prediction. Graph Convolutional Network (GCN) [33] takes node attributes as input and constructs node representations through the convolution of neighboring node representations in each hidden layer, and uses the final layer to predict node labels by minimizing cross-entropy loss. Graph Attention Network (GAT) [84] then imports the self-attention mechanism into GCN to make the graph convolution performed in a more intelligent way. The supervised network embedding algorithms encode label information into the network embedding process. The learned embeddings not only well respect the network structure and node content attributes, but also possess the discriminative power for more accurate node classification. Depending on the existence of node labels, the supervised network embedding is only tailored for node classification task and is hard to generalized to other tasks.

2.2 Collective Classification

For networked data classification, collective classification (CC) has received considerable attention in recent years [53, 68]. Its key idea is to construct a new set of features that summarize label dependencies to improve classification accuracy. Exact inference solutions to CC are NP-hard and prohibitively expensive. Therefore, many research studies have focused on deriving approximate inference algorithms.

Two representative approximate inference algorithms for CC are Iterative Classification Algorithm (ICA) and Gibbs Sampling (GS). They are both local classifier-based methods that can be implemented in two steps: (1) the local classifier learning step, in which a local classifier is learned to predict labels for each node using node content features together with relational features derived from the label information in the neighborhood; and (2) the collective inference step, in which the two processes of computing relational features for unlabeled nodes using predicted labels and classifying unlabeled nodes using the learned local classifier are iteratively repeated until convergence.

When the network is sparsely labeled, traditional CC algorithms are ineffective to provide satisfactory accuracy. This is mainly because, at low label density, class labels of many neighboring nodes are unknown during inference, and it thus becomes difficult to learn accurate parameters related to label-based features. Therefore, semi-supervised CC algorithms have been proposed to leverage unlabeled data to enhance classification performance. These algorithms typically first adopt a bootstrap step to predict labels for unlabeled nodes, through a classifier trained on labeled nodes using only content features. These predicted labels are taken as true labels, and together with known labels, are used to compute relational features. A classifier is then trained on labeled nodes or all nodes to perform collective inference. The variants of semi-supervised algorithms mainly differ in how they train a best classifier from all available features. [51] provided a comprehensive comparison of different semi-supervised CC algorithms and found that learning a hybrid classifier from content features and relational features can best boost the performance of semi-supervised CC algorithms. In a follow-up work, [52] pointed out that training discriminative classifiers additionally with neighbors' attributes can yield further accuracy gains for semi-supervised collective classification.

Another line of research has focused more on increasing network connectivity between unlabeled nodes and labeled nodes [17, 49, 69]. [49] enriched the linkage of the original network by adding extra links based on node attribute similarity, and a weighted-vote relational neighbor with relational labeling (wwRN+RL) classifier was used to classify unlabeled nodes. [17] proposed to add "ghost edges" created via random walk with restart, to a network, which enables information propagation from labeled nodes to unlabeled ones. In [69], a Latent Network Propagation model (LNP) was proposed that constructs a latent graph from different information sources and uses label propagation to predict labels of unlabeled nodes. However, most of these methods consider either local dependencies between network structure and labels, or correlations between nodes' content features and labels, but ignore the important role of their interplay for jointly classifying labels of related nodes.

2.3 Node Similarity Search

To enable similarity search over networks, various metrics have been proposed to measure the structural relatedness between nodes. Bibliographic Coupling [32] and Co-citation [70] measure node similarity by counting the number of common neighbors. Other common neighbor based metrics include Jaccard's coefficient, Salton's coefficient, the Adamic/Acar coefficient [1], *etc.* This kind of metrics are incapable of capturing the similarity between nodes sharing no common neighbors. SimRank [28] estimates node similarity recursively with the principle that two nodes are similar if they have connections with similar nodes. Because calculating SimRank similarity is computationally expensive, other algorithms, like TopSim [38] and [36], are proposed to reduce its time complexity. P-Rank [110] enhances SimRank by jointly modeling both in- and out-link relationships for node structural similarity estimation. VertexSim [81] represents each node as a convex combination of anchor nodes by optimizing a geometric objective, and then measures node similarity with the new representations. The above metrics only capture the similarity relying on the connectivity among the local neighborhood, but neglect the *structural equivalence* between nodes sharing similar structural roles while distantly located. [29] justifies a series of axiomatic properties that should be satisfied by a role similarity measure, and proposes RoleSim, a role similarity measure, which

is calculated in an iterative way and is proved to satisfy all the justified properties. Panther [108] estimates local structural similarity between pairwise nodes through their co-occurrence frequencies in randomly sampled paths. Panther++ [108] augments Panther with structural role similarity by measuring the difference in neighbor node co-occurrence distributions.

Calculating the aforementioned structure similarity metrics between all pairwise nodes, which is necessary for exact node similarity search, is usually time-consuming, with a time complexity at least quadratic to the number of nodes. Moreover, the above structure similarity metrics fail to capture the similarity measured by node content features. The two limitations make the existing structural similarity estimation based search methods unsuitable for large-scale networks with rich node content features.

Chapter 3

Preliminaries

In this chapter, as preliminaries, we give the formal definitions of the important terminologies, and summarize the benchmark datasets used in our experiments.

3.1 Notations

For ease of presentation, we first define a list of common notations that will be used throughout the thesis, as shown in Table 1.

Table 3.1 A summary of common notations

G	The given information network
$\mathcal{V}$	Set of nodes in the given information network
$\mathcal{E}$	Set of edges in the given information network
$\|\mathcal{V}\|$	Number of nodes
$\|\mathcal{E}\|$	Number of edges
X, T	The node attribute matrix
d	Dimension of learned node representations

3.2 Definitions

In this section, we first define important terminologies that are used to discuss the proposed algorithms next, following by a formal definition of network embedding.

Definition 1 (Information Network) *An information network is defined as* $G = (\mathcal{V}, \mathcal{E}, X)$*, where* $\mathcal{V}$ *denotes a set of nodes, and* $|\mathcal{V}|$ *denotes the number of nodes in network* G*.* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ *denotes a set of edges connecting the nodes, and* $|\mathcal{E}|$ *denotes the number of edges.* X *is the node attribute matrix, for networks with text attributes, we also use* T *to denote node text attribute matrix. Due to privacy concern or information access difficulty, node attribute matrix* X *is often sparse. For each* $(v_i, v_j) \in \mathcal{E}$*, if information network* G *is undirected, we have* $(v_j, v_i) \in \mathcal{E}$*; if* G *is directed,* (v_j, v_i) *unnecessarily belongs to* $\mathcal{E}$*.*[1] *Each edge* $(v_i, v_j) \in \mathcal{E}$ *is also associated to a weight* w_{ij}*, which is equal to* 1*, if the information network is binary (unweighted).*

Intuitively, the generation of information networks is not groundless, but guided or dominated by certain latent mechanisms. Although the latent mechanisms are hardly known, they can be reflected by some network properties that widely exist in information networks. Hence, the common network properties are essential for the learning of node representations that are informative enough to accurately interpret information networks. Below, we introduce several common network properties.

Definition 2 (First-order Proximity) *The first-order proximity is the local pairwise proximity between two connected nodes [76]. For each node pair* (v_i, v_j)*, if* $(v_i, v_j) \in \mathcal{E}$*, the first-order proximity between* v_i *and* v_j *is* w_{ij}*; otherwise, the first-order proximity between* v_i *and* v_j *is 0. The first-order proximity captures the direct neighbor relationships between nodes.*

Definition 3 (Second-order Proximity and High-order Proximity) *The second-order proximity captures the* 2*-step relations between each pair of node [76]. For each node pair* (v_i, v_j)*, the second order proximity is determined by the number of common*

[1] Without any specific declaration, the networks discussed in this thesis are assumed to be undirected.

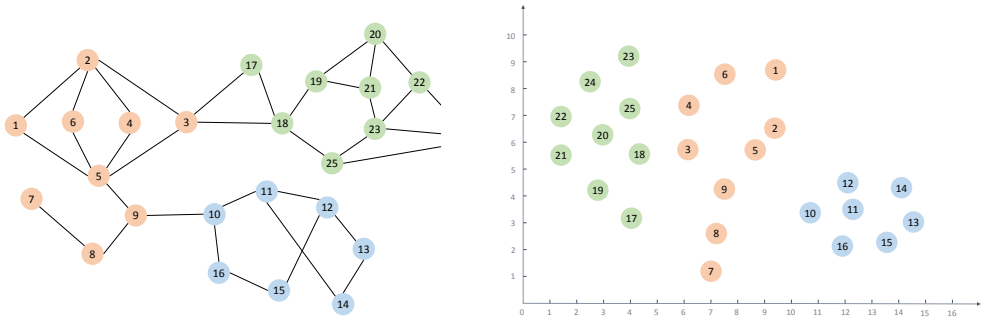
neighbors shared by the two nodes, which can also be measured by the 2-step transition probability from v_i to v_j equivalently. Compared with the second-order proximity, the high-order proximity [8] captures more global structure, which explores k-step ($k \geq 3$) relations between each pair of nodes. For each node pair (v_i, v_j), the higher-order proximity is measured by the k-step ($k \geq 3$) transition probability from node v_i to node v_j, which can also be reflected by the number of k-step ($k \geq 3$) paths from v_i to v_j. The second-order and high-order proximity capture the similarity between a pair of, indirectly connected, node with similar structural contexts.

Node attribute: In addition to network structure, node attributes can provide direct evidence to measure content-level similarity between nodes. As shown in [96, 90], node attributes and network structure can help each other filter out noisy information and compensate each other to jointly learn informative node representations.

Node label: Node labels provide direct information about the semantic categorization of each network node to certain classes or groups. Node labels are strongly influenced by and inherently correlated to both network structure and node attributes [27]. Though node labels are usually partially observed, when coupled with network structure and node attributes, they encourage a network structure and node attribute consistent labeling, and help learn informative and discriminative node representations.

Definition 4 (Network Embedding) *Given an information network $G = (\mathcal{V}, \mathcal{E}, X)$, by integrating network structure in $\mathcal{E}$, node attributes in X and node labels, the task of network embedding is to learn a mapping function $f : v \longmapsto r_v \in \mathbb{R}^d$, where r_v is the learned vector representation of node v, and d is the dimension of the learned representation. The transformation f preserves the original network information, such that two nodes similar in the original network should also be represented similarly in the learned vector space.*

In general, the learned node representations should satisfy the following conditions: (1) *low-dimensional,i.e.*, $d \ll |\mathcal{V}|$, in other words, the dimension of learned node representations should be much smaller than the dimension of the original adjacency matrix representation for memory efficiency and the scalability of subsequent network analytic tasks; (2) *informative*, *i.e.*, the learned node representations should preserve the



(a) Input: Information Network (b) Output: node Representations

Fig. 3.1 A conceptual view of network embedding. Nodes in (a) are indexed using their ID and color coded based on their community information. The network representation learning in (b) transforms all nodes into a two-dimensional vector space, such that nodes with structural proximity are close to each other in the new embedding space.

proximity reflected by network structure, node attributes and node labels (if available); (3) *continuous*, *i.e.*, the learned node representations are supposed to have continuous real values to support subsequent network analytic tasks, like node classification, node clustering, or anomaly detection, and have smooth decision boundaries to ensure the robustness of these tasks.

Fig. 3.1 demonstrates a conceptual view of network embedding, using a toy network. In this case, only network structure is considered to learn node representations. Given an information network shown in Fig. 3.1(a), the objective of network embedding is to embed all network nodes into a low-dimensional space, as depicted in Fig. 3.1(b). In the embedding space, nodes with structural proximity are represented closely to each other. For example, as node 7 and node 8 are directly connected, the first-order proximity enforces them close to each other in the embedding space. Though node 2 and node 5 are not directly connected, they are also embedded closely to each other because they have high second-order proximity, which is reflected by 4 common neighbors shared by these two nodes. Node 20 and node 25 are not directly connected, nor do they share common direct neighbors. However, they are connected by many k-step paths ($k \geq 3$), which proves that they have high-order proximity. Thus, node 20 and node 25 also have close embeddings.

Table 3.2 Summary of nine real-world networks

	$\|V\|$	$\|E\|$	# of Attribute	# of Class
Cora	2,708	5,278	1,433	7
Citeseer	3,312	4,732	3,703	6
Wikipedia	2,405	17,981	4,973	19
PubMed	19,717	44,338	500	3
BlogCatalog	5,196	171,743	8,189	6
Flickr	7,575	239,738	12,047	9
Facebook	4,039	88,234	1,403	4
DBLP(Subgraph)	18,448	45,611	5,959	4
DBLP(Full)	1,632,442	2,327,450	154,309	N/A

3.3 Benchmark Datasets

In this thesis, nine real-world information networks are used to conduct experiments. Table 3.2 summarizes the statistics of these networks. The detailed information about the networks are as follows:

- **Cora**[2]. The Cora network is composed of 2,708 machine learning publications and their citation relationships. Theses publications are categorized into seven groups. Each publication is represented by a 1,433-dimensional binary vector, with each dimension denoting the presence/absence of the corresponding word.
- **Citeseer**[2]. Citeseer is another citation network with 3,312 papers and 4,732 citation relations. There are 6 classes among papers. According to the occurrence of the corresponding word, each paper is described by a 3,703-dimensional binary vector.
- **PubMed**[2]. The PubMed network contains 19,717 scientific publications related to diabetes in three classes. Each paper is represented by a TF-IDF weighted word vector from a dictionary composed by 500 unique words. There are 44,338 links among them.

[2]https://linqs.soe.ucsc.edu/data

- **Wikipedia**[2]. The Wikipedia network contains 2,405 Wikipedia articles from 19 classes. Each article is represented by a 4,973-dimensional TF-IDF vector. There are 17,981 hyperlinks among these articles.

- **BlogCatalog**[3]. The BlogCatalog network is an online social network formed by BlogCatalog, a blogger community. The BlogCatalog network contains 5,196 users and 171,743 follower-followee relations. Users' groups are defined as the categories of their blogs. The keywords of users' blogs are used to construct users' feature vectors. Here, binary feature vectors are constructed, with only the keyword occurrence state concerned.

- **Flickr**[3]. Flickr is an online photo sharing platform. The Flickr network includes 7,575 users and 239,738 follower-followee relations. These users join in 9 predefined groups. Users' features are described by the tags of their images. Each user is represented by 12,047-dimensional binary vector, according to the occurrence/absence of the corresponding tag.

- **Facebook**. The Facebook network is merged from 10 Facebook ego-networks from SNAP[4]. There are 4,039 users and 88,234 friendship relations. Each user is described by a 1403-dimensional bag-of-words vector from tree-structured user profiles [39]. Users' education types are used as labels.

- **DBLP(Subgraph)** and **DBLP(Full)**. The DBLP(Full) network is formed by the papers, paper titles, and paper citations of the DBLP bibliographic network[5]. In DBLP(Full), there are in total 1,632,442 papers and 2,327,450 citations. The DBLP(Subgraph) is a subgraph of the DBLP(Full) network, constructed by papers from four research areas: *Database, Data Mining, Artificial Intelligence, and Computer Version*, which also act as paper labels. DBLP(Subgraph) contains 18,448 papers and 45,611 citation relations. For DBLP(Full) and DBLP(Subgraph), papers' titles are used to construct binary bag-of-words feature vectors.

[3]http://people.tamu.edu/~xhuang/Code.html
[4]https://snap.stanford.edu/data/
[5]https://aminer.org/citation (Version 3 is used)

Part I

Task-general Attributed Network Embedding

Overview

As data preparation for general network analysis, task-general attributed network embedding aims to learn node representations for attributed networks in an unsupervised and task-independent way. As the node representations are not tailored for specific network analytic tasks, they can be applied to a variety of real-world applications.

In this part, we propose a series of task-general attributed network embedding, aiming to overcome the following research challenges:

- **Heterogeneity**: How to effectively fuse information from network structure and node content attributes into a unified node representations, making them complement rather than deteriorate each other.
- **Data Sparsity**: How to overcome the incompleteness existing in both network structure and node content attributes, *i.e.*, make the best of available data and minimize the negative impacts caused by missing data.
- **Scalability**: How to make attributed network embedding able to scale to large-scale networks, with a large number of nodes and edges, as well as high-dimensional node content features.

In Chapter 4, to overcome the **heterogeneity** challenge, we propose the Homophily, Structure, and Content Augmented network embedding (HSCA) algorithm that seamlessly integrate information from three sources (1) *homophily (the first-order proximity)* (2) *structural context* and (3) *node content* into a unified node representation through regularized inductive matrix factorization. The learned node representations are expected to be consistent with structural context and node content, and follow the social homophily constraints in the learned space.

In Chapter 5, to handle the **heterogeneity** and **data sparsity** challenge, we propose the attri2vec algorithm that learns node embeddings by discovering a latent node attribute subspace via a network structure guided transformation performed on the original attribute space. The resultant subspace can respect network in a more consistent way towards learning high-quality node representations.

In Chapter 6, to address the **data sparsity** and **scalability** challenge, we propose the Scalable Incomplete Network Embedding (SINE) algorithm that learns node representations for incomplete graphs, by separately modeling pairs of node-context and node-attribute relationships through a three-layer neural network. A stochastic gradient descent based online algorithm is derived to train the algorithm, allowing the algorithm to scale up to large-scale networks with high efficiency.

Chapter 4

Homophily, Structure, and Content Augmented Network Embedding

4.1 Introduction

Recent years have witnessed an enormous growth of information networks, such as social networks, communication networks, and citation networks, generated from all aspects of social applications. Unlike conventional vector-based data, instances in information networks are neither independent nor identically distributed, but are connected to each other with complex dependencies. To process such networked data, one critical problem is network embedding, which aims at embedding a network into a low-dimensional space, *i.e.*, learning vector representations for each node, with the objective of preserving network structure in the embedded space. As a result, off-the-shelf vector-based machine learning techniques can be easily applied. Existing studies on network embedding [63, 76] have shown that network embedding can effectively capture network structures to facilitate subsequent analytic tasks, such as node classification [68], anomaly detection [6], and link prediction [47].

From literatures up to date, we summarize that three information sources are leveraged to learn node representations:

- **Homophily** (or the first-order proximity): Homophily phenomenon [54] is a known fact in social networks where nodes with similar attributes (content) tend to be connected through relational ties, which contribute to local structures of individual nodes [2]. To preserve such local connectivity in the network, nodes directly connected to each other should be embedded closely together in the learned representation space.

- **Structural Context:** In addition to direct neighbors, nodes in a network also have dependencies with indirect neighbors. To respect the global network structure, nodes sharing common neighbors (*i.e.*, context nodes) should be close to each other in the learned space. Structural context includes the second-order proximity and high-order proximity. Compared with the connectivity based homophily, structural context employs broader structure to measure node similarity.

- **Node Content:** Rich information on node attributes provides the most direct evidence to measure content-based similarity of nodes. Thus, nodes having similar content features should also be represented closely to each other in the new space.

Fig. 4.1 gives an illustrative example. As node 1 and node 3 are directly connected, they should be represented closely to each other in the embedded space, to reflect the homophily principle. On the other hand, though there is no direct link between node 1 and node 2, they share three common neighbors, *i.e.*, they have the same structural context and thus should be also represented closely to each other. In addition, though node 1 and node 4 are not directly connected nor share common neighbors, they are likely to have similar representations because they share similar content on node attributes. We expect that the consideration of all three information sources (*i.e.*, homophily, contextual structure, and node content), and more importantly, the interplay between the three parties can maximally contribute to learning effective network representations.

To date, most of existing work on network embedding (*e.g.*, [77, 63, 76]) learns representations from network structure only. For example, DeepWalk [63] generalizes the Skip-Gram model [55], which learns word representations using word context in

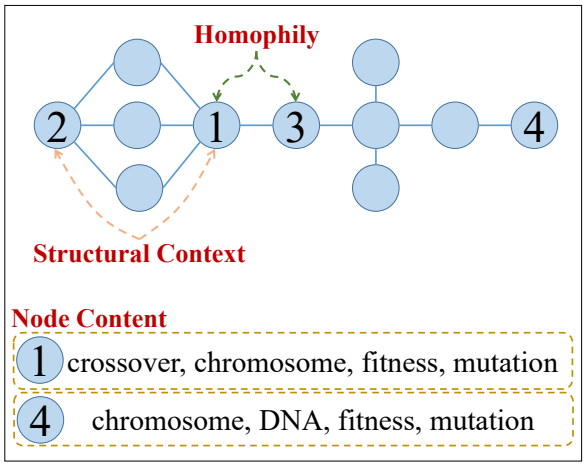


Fig. 4.1 A toy example of information network. Our proposed HSCA algorithm utilizes all three information sources (*i.e.*, homophily, structural context, and node content), for learning useful network representations.

sentences, by taking random walk sequences in graphs as the equivalence of sentences. As an analogy to word context, nodes with similar structural context in random walk sequences tend to have similar representations. Hence, only structural context is utilized by DeepWalk to learn node representations. LINE [76] learns node representations by explicitly modeling two kinds of pairwise node relationships, *i.e.* the first-order proximity and the second-order proximity. The GraRep model [8] moves a further step to consider different k-step proximities (in particular when $k > 2$), for forming a global representation. Thus, homophily and structural context are both utilized by LINE and GraRep. Only very recently, Text-Associated DeepWalk (TADW) [96] attempts to import node features into matrix factorization based DeepWalk. TADW inherits the information source of structural context from DeepWalk and exploits node content for improving representation learning. Table 4.1 summarizes the information sources used by all these methods. Clearly, none of these methods have utilized all above three information sources to learn informative node representations.

In this chapter, to fill this gap, we propose a new network embedding algorithm, called HSCA (Homophily, Structure, and Content Augmented network embedding), that effectively learns a latent network representation using all three information sources. Inspired by TADW [96], we formulate the problem of network embedding as a matrix factorization problem that 1) considers broader structural context in random walk

Table 4.1 A summary of network embedding algorithms based on the information sources used.

Algorithm	**Homophily**	**Structural Context**	**Node Content**
DeepWalk [63]		✓	
LINE [76]	✓	✓	
GraRep [8]	✓	✓	
TADW [96]		✓	✓
HSCA (our algorithm)	✓	✓	✓

sequences to discover hidden relatedness between nodes; 2) introduces a regularization term to enforce homophily between directly connected nodes; and 3) imports nodes' content features into the matrix factorization process. To this end, we design a new matrix factorization objective function and derive an efficient optimization algorithm based on conjugate gradient methods to solve it. By simultaneously integrating homophily, structural context, and node content, the HSCA algorithm effectively embeds a network into a single latent representation space that captures the interplay between network structures and node content. The representation learned by HSCA has the advantage of preserving both network structure and node content information, as well as following the homophily constraints in the new space. We evaluate the HSCA algorithm on four real-world networks for the task of multi-class node classification. Experimental results show that, compared with the state-of-the-art network embedding algorithms, HSCA's representation achieves better classification performance, especially for sparsely labeled networks.

4.2 Problem Definition and Preliminaries

In this section, we formally define our network embedding problem and review preliminaries of Text-Associated DeepWalk (TADW).

4.2.1 Problem Definition

Given an undirected graph $G = (\mathcal{V}, \mathcal{E}, T)$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. T is the node text feature matrix with the i-th column of T, $\boldsymbol{t}_i$, characterizing node $\boldsymbol{v}_i \in \mathcal{V}$. Our problem **aims** to embed graph G into a low-dimensional space, *i.e.*, learning a vector representation $\boldsymbol{x} \in \mathbb{R}^k$ for each node $\boldsymbol{v} \in \mathcal{V}$ in G.

The learned representation $\boldsymbol{x}$ should satisfy two properties: (1) low-dimensionality, the dimension k of the embedded space is expected to be much smaller than $|\mathcal{V}|$, and (2) informativeness, the learned representation $\boldsymbol{x}$ should preserve the graph structure information reflected in $\mathcal{E}$ and node content information in node text features T. With these two properties, the learned representation can be taken as input to network mining tasks, like node classification.

4.2.2 Preliminaries: Text-Associated DeepWalk

Inspired by DeepWalk [63], Text-Associated DeepWalk (TADW) [96] incorporates node text features into network embedding. In DeepWalk, nodes in a network are taken as words in natural language and a set of short random walk sequences are generated from the given network to provide context for nodes. Given a node v_i and its context nodes within t window size, $\{v_{i-t}, \cdots, v_{i+t}\} \setminus v_i$, the representation for v_i, $\Phi(v_i)$, is learned by maximizing the conditional probability:

$$\mathrm{P}(\{v_{i-t}, \cdots, v_{i+t}\} \setminus v_i | \Phi(v_i)) = \prod_{j=i-t, j\neq i}^{i+t} \mathrm{P}(v_j | \Phi(v_i)). \tag{4.1}$$

In [63], the probability $\mathrm{P}(v_j|\Phi(v_i))$ is approximated using Hierarchical Softmax [56] to speed up training.

We denote the PageRank matrix of graph G as $P \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$. The degree of node v_i is denoted by d_i. We then have $P_{ij} = 1/d_i$, if $(v_i, v_j) \in \mathcal{E}$; otherwise, $P_{ij} = 0$. In [96], DeepWalk is proved to be equivalent to factorizing a matrix $M \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ given by

$$M_{ij} = \log \frac{[P + P^2 + \cdots + P^t]_{ij}}{t}. \tag{4.2}$$

Matrix M can be factorized as two low-rank matrices $W \in \mathbb{R}^{k\times|\mathcal{V}|}$ and $H \in \mathbb{R}^{k\times|\mathcal{V}|}$ by solving the following problem

$$\min_{W,H} \frac{1}{2}\|M - W^{\mathrm{T}}H\|_F^2 + \frac{\lambda}{2}(\|W\|_F^2 + \|H\|_F^2). \tag{4.3}$$

The first minimization term guarantees precise factorization. The minimization of $\|W\|_F^2$ and $\|H\|_F^2$ imposes low-rank constraint on W and H, and $\frac{\lambda}{2}$ controls the trade-off between these two terms. After this matrix factorization problem is solved, W and H can be concatenated together to form a $2k$-dimensional representation for each node.

To make the representation learned by matrix factorization based DeepWalk more informative, TADW imports node text features $T \in \mathbb{R}^{f_t\times|\mathcal{V}|}$ into the matrix factorization of (4.3), with f_t being the number of text features. The matrix factorization formulation is changed into

$$\min_{W,H} \frac{1}{2}\|M - W^{\mathrm{T}}HT\|_F^2 + \frac{\lambda}{2}(\|W\|_F^2 + \|H\|_F^2), \tag{4.4}$$

where $H \in \mathbb{R}^{k\times f_t}$. For computational efficiency, in (4.4), M is approximated by $(P + P^2)/2$. After W and H are solved, node representations are generated by concatenating W and HT as feature vectors.

4.3 HSCA: Homophily, Structure, and Content Augmented Network Embedding

In this section, we formulate the optimization problem of our proposed algorithm and elaborate solutions to this problem.

4.3.1 The Optimization Problem

TADW can learn more informative representations than DeepWalk, by importing text features into the matrix factorization formulation of DeepWalk. However, TADW's representations lack the ability to preserve the homophily property of the network in the embedded space. Therefore, in this work, we propose to enforce homophily between connected nodes in our matrix factorization formulation.

By solving the problem of (4.4), the i-th node is then represented by $\left[\boldsymbol{w}_i^{\mathrm{T}}, (H\boldsymbol{t}_i)^{\mathrm{T}}\right]^{\mathrm{T}}$, with $\boldsymbol{w}_i$ and $\boldsymbol{t}_i$ being the i-th column of W and T respectively. To make connected nodes close enough to each other in the learned $2k$-dimensional space, the distance between their feature vectors should be constrained in the representation space. To achieve this, we import a regularization term $\mathcal{R}(W, H)$ into the minimization problem of (4.4). The regularization term is defined as

$$\begin{aligned}\mathcal{R}(W,H) &= \frac{1}{4}\sum_{i,j=1}^{|\mathcal{V}|} A_{ij}\left\|\begin{bmatrix}\boldsymbol{w}_i\\ H\boldsymbol{t}_i\end{bmatrix} - \begin{bmatrix}\boldsymbol{w}_j\\ H\boldsymbol{t}_j\end{bmatrix}\right\|_2^2 \\ &= \frac{1}{4}\sum_{i,j=1}^{|\mathcal{V}|} A_{ij}\|\boldsymbol{w}_i - \boldsymbol{w}_j\|_2^2 + \frac{1}{4}\sum_{i,j=1}^{|\mathcal{V}|} A_{ij}\|H\boldsymbol{t}_i - H\boldsymbol{t}_j\|_2^2,\end{aligned} \tag{4.5}$$

where A is the adjacent matrix of network G. $\mathcal{R}(W, H)$ can be further decomposed as $\mathcal{R}(W, H) = \mathcal{R}_1(W) + \mathcal{R}_2(H)$, where $\mathcal{R}_1(W)$ is the regularization term on W, and $\mathcal{R}_2(H)$ is the regularization term on H. $\mathcal{R}_1(W)$ is formulated as

$$\begin{aligned}\mathcal{R}_1(W) &= \frac{1}{4}\sum_{i,j=1}^{|\mathcal{V}|} A_{ij}\|\boldsymbol{w}_i - \boldsymbol{w}_j\|_2^2 \\ &= \frac{1}{4}\sum_{i,j=1}^{|\mathcal{V}|} A_{ij}(\boldsymbol{w}_i - \boldsymbol{w}_j)^{\mathrm{T}}(\boldsymbol{w}_i - \boldsymbol{w}_j) \\ &= \frac{1}{2}\sum_{i=1}^{|\mathcal{V}|} D_{ii}\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{w}_i - \frac{1}{2}\sum_{i,j=1}^{|\mathcal{V}|} A_{ij}\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{w}_j \\ &= \frac{1}{2}\sum_{i,j=1}^{|\mathcal{V}|} L_{ij}\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{w}_j,\end{aligned} \tag{4.6}$$

where $D_{ii} = \sum_{j=1}^{|\mathcal{V}|} A_{ij}, D_{ij} = 0$, for $i \neq j$ and $L = D - A$. $\mathcal{R}_2(H)$ is formulated as

$$\begin{aligned}
\mathcal{R}_2(H) &= \frac{1}{4}\sum_{i,j=1}^{|\mathcal{V}|} A_{ij}\|H\boldsymbol{t}_i - H\boldsymbol{t}_j\|_2^2 \\
&= \frac{1}{4}\sum_{i,j=1}^{|\mathcal{V}|} A_{ij}\|H(\boldsymbol{t}_i - \boldsymbol{t}_j)\|_2^2 \\
&= \frac{1}{4}\sum_{i,j=1}^{|\mathcal{V}|} A_{ij}(\boldsymbol{t}_i - \boldsymbol{t}_j)^{\mathrm{T}} H^{\mathrm{T}} H(\boldsymbol{t}_i - \boldsymbol{t}_j) \\
&= \frac{1}{2}\sum_{i=1}^{|\mathcal{V}|} D_{ii}\boldsymbol{t}_i^{\mathrm{T}} H^{\mathrm{T}} H\boldsymbol{t}_i - \frac{1}{2}\sum_{i,j=1}^{|\mathcal{V}|} A_{ij}\boldsymbol{t}_i^{\mathrm{T}} H^{\mathrm{T}} H\boldsymbol{t}_j \\
&= \frac{1}{2}\sum_{i,j=1}^{|\mathcal{V}|} L_{ij}\boldsymbol{t}_i^{\mathrm{T}} H^{\mathrm{T}} H\boldsymbol{t}_j.
\end{aligned} \tag{4.7}$$

Here, we define the new object function as

$$\mathcal{F}(W, H) = \frac{1}{2}\|M - W^{\mathrm{T}} HT\|_F^2 + \frac{\lambda}{2}(\|W\|_F^2 + \|H\|_F^2) + \mu(\mathcal{R}_1(W) + \mathcal{R}_2(H)). \tag{4.8}$$

Then, a new optimization problem is formulated as

$$\min_{W,H} \mathcal{F}(W, H). \tag{4.9}$$

4.3.2 Solving the Problem

We now discuss how to derive the solution to the optimization problem (4.9). Though $\mathcal{F}(W, H)$ is not convex with respect to W and H, it can be easily proved that when one of W and H is fixed, $\mathcal{F}(W, H)$ is convex with respect to the other variable. After W and H are initialized properly, we can solve problem (4.9) with the following alternative update procedure:

$$W^{(\tau)} = \arg\min_{W} \mathcal{F}(W, H^{(\tau-1)}), \tag{4.10}$$

$$H^{(\tau)} = \arg\min_{H} \mathcal{F}(W^{(\tau)}, H). \tag{4.11}$$

To optimize W with H fixed, we define $\boldsymbol{\omega} = \boldsymbol{vec}(W^{\mathrm{T}})$ and $\tilde{\boldsymbol{x}}_{ij} = (H\boldsymbol{t}_j) \otimes \boldsymbol{e}_i$, where $\boldsymbol{e}_i$ is the i-th column of the $|\mathcal{V}| \times |\mathcal{V}|$ identity matrix. Then W can be updated by

solving the following problem:

$$\arg\min_{\boldsymbol{\omega}\in\mathbb{R}^{|\mathcal{V}|k}} f(\boldsymbol{\omega}), \tag{4.12}$$

where

$$f(\boldsymbol{\omega}) = \frac{1}{2}\sum_{i,j=1}^{|\mathcal{V}|}(M_{ij} - \boldsymbol{\omega}^{\mathrm{T}}\tilde{\boldsymbol{x}}_{ij})^2 + \frac{\mu}{2}\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{\Lambda}\boldsymbol{\omega} + \frac{\lambda}{2}\|\boldsymbol{\omega}\|_2^2, \tag{4.13}$$

where $\boldsymbol{\Lambda} = \mathbf{I}_k \otimes L$ with $\mathbf{I}_k$ being the $k \times k$ identity matrix.

To optimize H with W fixed, we define $\boldsymbol{\eta} = \boldsymbol{vec}(H)$ and $\tilde{\boldsymbol{y}}_{ij} = \boldsymbol{t}_j \otimes \boldsymbol{w}_i$. Then H can be updated by solving the following problem:

$$\arg\min_{\boldsymbol{\eta}\in\mathbb{R}^{kf_t}} g(\boldsymbol{\eta}), \tag{4.14}$$

where

$$g(\boldsymbol{\eta}) = \frac{1}{2}\sum_{i,j=1}^{|\mathcal{V}|}(M_{ij} - \boldsymbol{\eta}^{\mathrm{T}}\tilde{\boldsymbol{y}}_{ij})^2 + \frac{\mu}{2}\boldsymbol{\eta}^{\mathrm{T}}\boldsymbol{\Omega}\boldsymbol{\eta} + \frac{\lambda}{2}\|\boldsymbol{\eta}\|_2^2, \tag{4.15}$$

where $\boldsymbol{\Omega} = (TLT^{\mathrm{T}}) \otimes \mathbf{I}_k$.

To solve the optimization problem (4.12) and (4.14), we utilize Conjugate Gradient (CG) algorithm. For CG, the gradient of $f(\boldsymbol{\omega})$ and $g(\boldsymbol{\eta})$, and the multiplication of the Hessian matrix of $f(\boldsymbol{\omega})$ and $g(\boldsymbol{\eta})$ with vector $\boldsymbol{s}$ are calculated as follows:

$$\nabla f(\boldsymbol{\omega}) = \sum_{i,j=1}^{|\mathcal{V}|}(\boldsymbol{\omega}^{\mathrm{T}}\tilde{\boldsymbol{x}}_{ij} - M_{ij})\tilde{\boldsymbol{x}}_{ij} + \mu\Lambda\boldsymbol{\omega} + \lambda\boldsymbol{\omega}, \tag{4.16}$$

$$\nabla g(\boldsymbol{\eta}) = \sum_{i,j=1}^{|\mathcal{V}|}(\boldsymbol{\eta}^{\mathrm{T}}\tilde{\boldsymbol{y}}_{ij} - M_{ij})\tilde{\boldsymbol{y}}_{ij} + \mu\Omega\boldsymbol{\eta} + \lambda\boldsymbol{\eta}, \tag{4.17}$$

$$\nabla^2 f(\boldsymbol{\omega})\boldsymbol{s} = \sum_{i,j=1}^{|\mathcal{V}|}\tilde{\boldsymbol{x}}_{ij}\tilde{\boldsymbol{x}}_{ij}^{\mathrm{T}}\boldsymbol{s} + \mu\Lambda\boldsymbol{s} + \lambda\boldsymbol{s}, \tag{4.18}$$

$$\nabla^2 g(\boldsymbol{\eta})\boldsymbol{s} = \sum_{i,j=1}^{|\mathcal{V}|}\tilde{\boldsymbol{y}}_{ij}\tilde{\boldsymbol{y}}_{ij}^{\mathrm{T}}\boldsymbol{s} + \mu\Omega\boldsymbol{s} + \lambda\boldsymbol{s}. \tag{4.19}$$

In this way, the calculation of $\nabla f(\boldsymbol{\omega})$ and $\nabla^2 f(\boldsymbol{\omega})\boldsymbol{s}$ needs at least $O(|\mathcal{V}|^2 k)$ time, and the calculation of $\nabla g(\boldsymbol{\eta})$ and $\nabla^2 g(\boldsymbol{\eta})\boldsymbol{s}$ needs at least $O(|\mathcal{V}|^2 k f_t)$ time. Below, we give

more efficient solutions to calculating these four variables, with the detailed derivations given in the Appendix.

A. Calculation for $\nabla f(\boldsymbol{\omega})$

As shown in the Appendix, we have proven that

$$\nabla f(\boldsymbol{w}) = \boldsymbol{vec}(W^{\mathrm{T}}HTT^{\mathrm{T}}H^{\mathrm{T}} - MT^{\mathrm{T}}H^{\mathrm{T}}) + \mu\boldsymbol{vec}(LW^{\mathrm{T}}) + \lambda\boldsymbol{\omega}. \tag{4.20}$$

By changing the order of matrix multiplication in (4.20) carefully as

$$\nabla f(\boldsymbol{w}) = \boldsymbol{vec}(W^{\mathrm{T}}((HT)(HT)^{\mathrm{T}}) - M(HT)^{\mathrm{T}}) + \mu\boldsymbol{vec}(LW^{\mathrm{T}}) + \lambda\boldsymbol{\omega}, \tag{4.21}$$

the calculation of $\nabla f(\boldsymbol{w})$ requires only $O(|\mathcal{V}|k^2 + |\mathcal{V}|kf_t + nnz(L)k + nnz(M)k)$ time. For most real-world networks, edges are sparse, i.e., $O(|\mathcal{E}|) = O(|\mathcal{V}|)$, which indicates that $nnz(L), nnz(M) \ll |\mathcal{V}|^2$. Thus, calculating $\nabla f(\boldsymbol{w})$ with (4.20) is more efficient.

B. Calculation for $\nabla^2 f(\boldsymbol{\omega})\boldsymbol{s}$

In Appendix, we have shown that

$$\nabla^2 f(\boldsymbol{\omega})\boldsymbol{s} = \boldsymbol{vec}(SHTT^{\mathrm{T}}H^{\mathrm{T}}) + \mu\boldsymbol{vec}(LS) + \lambda\boldsymbol{s}, \tag{4.22}$$

where $S \in \mathbb{R}^{|\mathcal{V}|\times k}$ satisfies $\boldsymbol{s} = \boldsymbol{vec}(S)$. Similarly, in implementation, $f(\boldsymbol{\omega})\boldsymbol{s}$ is calculated as

$$\nabla^2 f(\boldsymbol{\omega})\boldsymbol{s} = \boldsymbol{vec}(S((HT)(HT)^{\mathrm{T}})) + \mu\boldsymbol{vec}(LS) + \lambda\boldsymbol{s}. \tag{4.23}$$

The above calculation of $f(\boldsymbol{\omega})\boldsymbol{s}$ requires only $O(|\mathcal{V}|k^2 + |\mathcal{V}|kf_t + nnz(L)k)$ time.

C. Calculation for $\nabla g(\boldsymbol{\eta})$

As is proved in Appendix,

$$\nabla g(\boldsymbol{\eta}) = \boldsymbol{vec}(WW^{\mathrm{T}}HTT^{\mathrm{T}} - WMT^{\mathrm{T}}) + \mu\boldsymbol{vec}(H(TLT^{\mathrm{T}})^{\mathrm{T}}) + \lambda\boldsymbol{\eta}. \tag{4.24}$$

For implementation efficiency, $\nabla g(\boldsymbol{\eta})$ is calculated as

$$\nabla g(\boldsymbol{\eta}) = \boldsymbol{vec}(W((W^{\mathrm{T}}H)(TT^{\mathrm{T}})) - W(MT^{\mathrm{T}})) + \mu\boldsymbol{vec}(H(TLT^{\mathrm{T}})^{\mathrm{T}}) + \lambda\boldsymbol{\eta}, \tag{4.25}$$

which requires only $O(|\mathcal{V}|kf_t + |\mathcal{V}|f_t^2 + nnz(L)f_t + nnz(M)f_t)$ time.

D. Calculation for $\nabla^2 g(\boldsymbol{\eta})\boldsymbol{s}$

It can also be proved that

$$\nabla^2 g(\boldsymbol{\eta})\boldsymbol{s} = \boldsymbol{vec}(WW^{\mathrm{T}}STT^{\mathrm{T}}) + \mu\boldsymbol{vec}(S(TLT^{\mathrm{T}})^{\mathrm{T}}) + \lambda\boldsymbol{s}. \tag{4.26}$$

For implementation efficiency, $\nabla^2 g(\boldsymbol{\eta})\boldsymbol{s}$ is calculated as

$$\nabla^2 g(\boldsymbol{\eta})\boldsymbol{s} = \boldsymbol{vec}(W((W^{\mathrm{T}}S)(TT^{\mathrm{T}}))) + \mu\boldsymbol{vec}(S(TLT^{\mathrm{T}})^{\mathrm{T}}) + \lambda\boldsymbol{s}, \tag{4.27}$$

which needs only $O(|\mathcal{V}|kf_t + |\mathcal{V}|f_t^2 + nnz(L)f_t)$ time.

4.4 Experiments

To evaluate the quality of learned representations, we conduct multi-class node classification on four real-world networks.

Table 4.2 Summary of Four Real-world Networks

Data Set	Cora	Citeseer	PubMed	Wikipedia
# of Nodes	2,708	3,312	19,717	2,405
# of Links	5,429	4,732	44,338	17,981
# of Features	1,433	3,703	500	4,973
# of Class	7	6	3	19

4.4.1 Benchmark Networks

Four information networks, Cora, Citeseer, Wikipedia, PubMed[1], are used in our experiments. A summary of the four networks is given in Table 4.2. For each of them, the direction of links and node's self-links are not considered. Two nodes have a link, if either of them is directly linked to the other. A detailed introduction to the four networks is given as follows:

Cora: It contains 2,708 machine learning publications from seven classes, and 5,249 citation links among these papers. Each paper is represented by a 0/1 vector of 1,433 dimensions indicating the presence/absence of the corresponding words.

Citeseer: It contains 3,312 scientific publications from six classes. Among these papers, there are 4,732 links. Each paper is represented by a 0/1 vector of 3,703 dimensions.

PubMed: It contains 19,717 scientific publications related to diabetes in three classes. Each paper is represented by a TF-IDF weighted word vector from a dictionary composed by 500 unique words. There are 44,338 links among them.

Wikipedia: It contains 2,405 Wikipedia articles from 19 classes. Each article is represented by a 4,973-dimensional TF-IDF vector. There are 17,981 hyperlinks among these articles.

[1]https://linqs.soe.ucsc.edu/data

4.4.2 Experimental Settings

In our experiments, we focus on multi-class node classification in sparsely labeled networks, in which we vary the training ratio from 1% to 10% by an increment of 1%. Because node classification with a small amount of training data is a difficult task, the effectiveness of different node representations can be clearly compared under this setting. For each training ratio, we randomly select a small number of labeled nodes for training, and the rest of unlabeled nodes are held out for testing. We repeat this process ten times and report the averaged Micro-F_1 and Macro-F_1 values as our evaluation criteria. Let TP_i, FP_i and FN_i denote true positive, false positive and false negative for class i, Micro-F_1 and Macro-F_1 for m-class node classification are defined as follows.

$$\text{Micro-}F_1 = \frac{2\sum_{i=1}^{m} TP_i}{2\sum_{i=1}^{m} TP_i + \sum_{i=1}^{m} FP_i + \sum_{i=1}^{m} FN_i}, \tag{4.28}$$

$$\text{Macro-}F_1 = \frac{1}{m}\sum_{i=1}^{m} \frac{2TP_i}{2TP_i + FP_i + FN_i}. \tag{4.29}$$

For all network embedding algorithms, we use the linear SVM implemented by Liblinear [15] to perform multi-class classification. We reduce the dimension of node text features to 200 by applying Singular Value Decomposition (SVD) on the TF-IDF matrix. For our proposed HSCA algorithm, k is set to 120, λ is set to 0.2, μ is set to 0.1 for Cora, Citeseer and PubMed, and for Wikipedia, μ is set to 0.04. W and H are initialized by random numbers and the iterative update for W and H stops when $\|W^{(\tau)} - W^{(\tau-1)}\|_F^2 + \|H^{(\tau)} - H^{(\tau-1)}\|_F^2 \leq 0.01$.

4.4.3 Baseline Methods

Our proposed HSCA algorithm is compared against the following baseline methods:

Text: This baseline simply uses 200-dimensional text features as node representations, which exploit only node content.

DeepWalk [63]: DeepWalk is a popularly studied network embedding algorithm that utilizes structural context for learning network representations. Default parameters are set for this algorithm.

DeepWalk+Text: The representations learned by DeepWalk and text features are simply concatenated together to serve as new node representations, which consider structural context information and node content information.

LINE [76]: LINE is a recently proposed network embedding algorithm that considers the first-order proximity and second-order proximity. Two types of representations are separately learned and concatenated together to form the final representation. Default parameters are also used.

LINE+Text: This baseline concatenates LINE's representations and node text features to form new node representations.

GraRep [8]: GraRep is one state-of-the-art network embedding algorithm that learns node representations by exploiting more global structural context than LINE. Parameters for this algorithm are also set as default values.

GraRep+Text: This method combines the representations learned by GraRep and node content features to generate new node representations.

TADW [96]: TADW utilizes both structural context and node content information to learn node representations, which is shown to outperform the naive combination of DeepWalk's representations and node content text features. For fair comparison, the parameter k is set to 120 as done in our proposed algorithm, and other parameters are set as default values.

4.4.4 Comparison of Network Representations

The Micro-F_1 and Macro-F_1 values for multi-class classification with different network representations on different networks are reported in Tables 4.3-4.6, respectively. For each training ratio, the largest Micro-F_1 and Macro-F_1 values are highlighted in **bold**. As the PubMed network is too large for GraRep to run, Micro-F_1 and Macro-F_1 values for GraRep and GraRep+Text are not available in Table 4.5.

Table 4.3 Classification Results on Cora

	Training Ratio	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Micro-F_1 (%)	Text	28.59	34.18	38.37	42.77	46.33	49.32	51.78	52.88	56.75	57.03
	DeepWalk	46.18	58.49	62.75	67.00	70.32	71.24	72.49	73.13	73.51	74.43
	DeepWalk+Text	44.03	56.67	61.66	67.62	71.22	72.57	74.07	74.48	75.85	76.85
	LINE	31.45	37.17	40.34	43.15	45.44	46.81	47.52	47.58	49.72	50.11
	LINE+Text	32.49	39.09	43.71	48.19	51.67	53.84	55.71	56.87	59.37	60.45
	GraRep	40.70	51.80	58.79	62.20	67.85	69.61	71.24	72.04	73.40	73.91
	GraRep+Text	40.25	51.78	58.88	63.66	69.34	71.13	72.95	73.50	75.67	76.16
	TADW	43.00	56.78	64.10	70.21	75.54	76.73	78.93	79.79	80.95	82.06
	HSCA	**49.09**	**64.85**	**69.79**	**75.27**	**79.10**	**79.80**	**81.27**	**81.99**	**82.53**	**83.69**
Macro-F_1 (%)	Text	14.21	20.18	26.52	32.17	38.57	41.38	45.70	47.29	51.95	52.59
	DeepWalk	33.22	53.49	56.85	62.32	67.79	68.00	70.22	71.10	71.55	72.94
	DeepWalk+Text	30.38	50.93	55.53	62.85	68.51	69.23	72.09	72.52	73.92	75.48
	LINE	20.51	29.26	33.68	37.44	42.81	43.65	45.15	45.28	47.46	48.17
	LINE+Text	20.30	29.67	35.68	41.81	47.99	49.72	52.69	54.11	57.04	58.41
	GraRep	32.64	48.37	55.57	59.21	66.36	67.59	69.80	70.76	72.19	72.62
	GraRep+Text	30.14	47.42	54.83	60.26	67.83	69.04	71.37	72.16	74.27	74.97
	TADW	30.48	51.01	58.38	65.98	73.64	74.27	77.25	78.10	79.28	80.82
	HSCA	**37.31**	**60.70**	**65.38**	**71.94**	**77.48**	**77.89**	**79.92**	**80.53**	**81.05**	**82.45**

Table 4.4 Classification Results on Citeseer

	Training Ratio	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Micro-F_1 (%)	Text	27.09	35.26	41.09	43.65	46.11	49.75	51.91	55.37	56.53	58.02
	DeepWalk	30.71	38.60	42.49	43.29	44.05	45.75	45.92	46.87	47.96	47.72
	DeepWalk+Text	32.51	42.35	48.02	49.61	52.07	54.10	55.53	57.98	58.99	59.71
	LINE	22.36	26.09	29.08	30.19	31.05	33.03	33.65	34.30	35.35	36.24
	LINE+Text	26.55	33.28	38.35	40.20	42.12	45.74	47.70	50.38	51.28	52.70
	GraRep	19.34	19.65	20.04	20.34	19.79	19.98	19.49	19.99	19.69	20.41
	GraRep+Text	27.09	35.23	41.14	43.66	46.13	49.75	51.86	55.41	56.54	58.09
	TADW	36.10	49.39	56.97	60.14	63.11	64.17	65.97	68.01	68.74	68.99
	HSCA	**37.66**	**51.46**	**59.03**	**61.90**	**64.33**	**65.59**	**67.03**	**68.78**	**69.42**	**69.63**
Macro-F_1 (%)	Text	19.02	29.34	36.05	38.10	40.94	44.11	46.77	50.16	51.56	52.73
	DeepWalk	23.09	32.12	36.22	36.46	37.50	38.87	39.06	39.63	41.16	40.79
	DeepWalk+Text	24.74	36.51	42.24	43.56	46.61	48.04	49.94	52.45	53.80	54.36
	LINE	16.00	21.71	25.70	26.53	27.73	29.35	30.29	31.06	32.26	32.93
	LINE+Text	19.83	28.33	34.08	35.53	38.06	41.07	43.50	46.12	47.21	48.37
	GraRep	5.40	5.47	5.56	5.63	5.51	5.55	5.43	5.55	5.48	5.65
	GraRep+Text	19.01	29.33	36.08	38.09	40.94	44.11	46.70	50.20	51.57	52.80
	TADW	28.67	43.60	51.46	53.94	57.24	57.83	60.20	62.28	63.44	63.61
	HSCA	**30.04**	**45.66**	**53.28**	**55.65**	**58.39**	**59.08**	**61.21**	**62.91**	**64.20**	**64.03**

From the four tables, we can clearly see that our HSCA algorithm outperforms all other baselines in terms of both Micro-F_1 and Macro-F_1 in most cases, with an exception of Macro-F_1 on Wikipedia with 8% and 9% training ratios (Table 4.6),

Table 4.5 Classification Results on PubMed

	Training Ratio	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Micro-F_1(%)	Text	52.81	62.16	67.43	71.69	74.53	76.29	77.99	78.55	79.60	80.24
	DeepWalk	67.98	73.82	75.90	76.80	77.23	77.38	77.93	77.82	78.19	78.23
	DeepWalk+Text	71.20	78.07	80.27	81.59	82.32	82.71	83.21	83.32	83.72	83.92
	LINE	52.01	56.88	60.54	62.32	63.51	64.46	65.48	65.71	66.67	66.93
	LINE+Text	59.29	67.90	72.75	75.44	77.33	78.57	79.60	80.16	81.25	81.57
	GraRep	–	–	–	–	–	–	–	–	–	–
	GraRep+Text	–	–	–	–	–	–	–	–	–	–
	TADW	69.71	78.93	81.48	82.75	83.35	83.74	84.16	84.21	84.33	84.59
	HSCA	**75.31**	**81.55**	**82.93**	**83.69**	**84.10**	**84.29**	**84.73**	**84.73**	**84.79**	**85.05**
Macro-F_1(%)	Text	39.75	53.10	62.58	68.56	72.46	74.72	77.07	77.66	79.09	79.86
	DeepWalk	55.80	68.10	72.35	73.67	74.45	74.65	75.50	75.25	75.87	75.89
	DeepWalk+Text	62.70	74.87	78.49	80.10	81.12	81.56	82.21	82.29	82.87	83.04
	LINE	39.39	46.68	52.91	55.85	58.13	59.59	61.31	61.32	63.06	63.44
	LINE+Text	49.25	62.52	70.01	73.56	76.03	77.51	78.78	79.40	80.74	81.11
	GraRep	–	–	–	–	–	–	–	–	–	–
	GraRep+Text	–	–	–	–	–	–	–	–	–	–
	TADW	62.45	76.99	80.68	82.07	82.87	83.28	83.79	83.81	83.98	84.25
	HSCA	**71.39**	**80.61**	**82.45**	**83.24**	**83.77**	**83.94**	**84.43**	**84.41**	**84.51**	**84.75**

Table 4.6 Classification Results on Wikipedia

	Training Ratio	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Micro-F_1(%)	Text	29.80	41.79	46.90	51.75	55.86	60.44	61.14	64.67	65.80	66.38
	DeepWalk	36.07	44.32	47.92	51.62	52.78	54.74	55.08	56.65	58.00	58.43
	DeepWalk+Text	37.41	48.36	52.36	56.35	59.08	62.65	62.99	65.38	66.99	67.28
	LINE	25.11	30.92	34.57	38.67	39.88	43.01	43.77	46.65	47.49	49.28
	LINE+Text	27.84	36.01	40.42	45.50	48.21	52.50	53.36	56.66	58.87	60.09
	GraRep	14.85	15.40	14.41	15.43	16.16	16.46	15.97	15.62	15.62	16.77
	GraRep+Text	29.75	41.81	46.86	51.66	55.95	60.46	61.18	64.67	65.85	66.33
	TADW	36.72	50.10	52.60	58.25	60.18	63.63	64.22	65.44	67.00	66.89
	HSCA	**39.92**	**53.85**	**55.78**	**60.29**	**62.70**	**65.49**	**65.45**	**67.04**	**68.44**	**68.79**
Macro-F_1(%)	Text	14.98	26.96	30.09	34.43	39.87	41.51	43.34	47.60	**48.50**	48.14
	DeepWalk	19.09	28.33	28.85	32.55	34.16	35.81	36.42	37.26	38.47	39.17
	DeepWalk+Text	19.99	31.25	32.95	36.83	40.52	42.36	43.35	46.41	47.54	47.73
	LINE	13.46	20.85	22.70	26.57	27.33	29.98	30.71	32.18	33.21	34.90
	LINE+Text	14.67	23.97	26.34	30.87	33.82	36.60	38.15	40.78	42.44	43.28
	GraRep	1.35	1.40	1.32	1.40	1.46	1.49	1.45	1.42	1.42	1.51
	GraRep+Text	14.97	26.91	30.04	34.35	39.92	41.56	43.28	**47.63**	**48.49**	48.11
	TADW	19.83	32.95	33.56	38.11	41.25	42.69	44.37	45.59	46.92	46.55
	HSCA	**22.07**	**35.05**	**35.41**	**39.22**	**43.72**	**44.52**	**45.03**	46.73	48.37	**48.44**

where HSCA performs slightly worse than GraRep+Text. Among other baselines, TADW is the second best performer because it captures the interactions between structural context and node content. However, as TADW does not consider homophily,

its representations are less effective as those learned by HSCA. This suggests that homophily is one important property to preserve, especially when node content is exploited for network representation learning.

By comparing HSCA with other baselines except LINE+text and GraRep+text, we can infer that with more information sources imported, the representations learned by HSCA can achieve better classification performance. The comparison between HSCA, LINE+Text, and GraRep+Text indicates that, although they all utilize three information sources (homophily, contextual structure, and node content), LINE+Text and GraRep+Text just naively combines different information sources, while HSCA can better capture their interplay by embedding the network into a single latent representation space. This leads to more informative network representations.

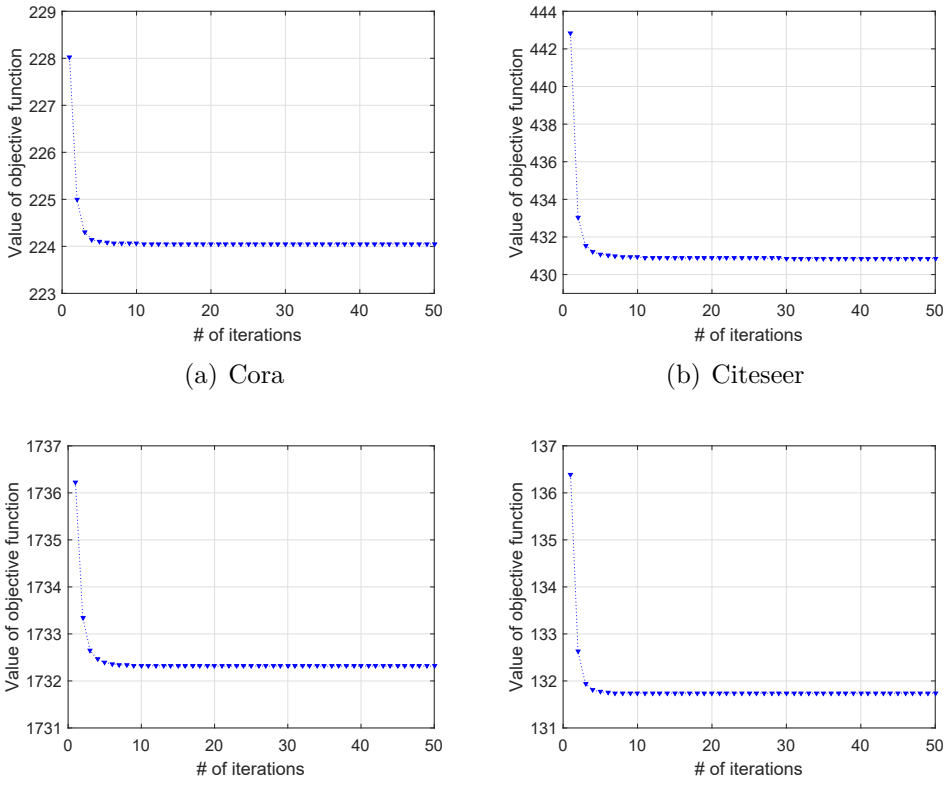
For DeepWalk, LINE and GraRep, when naively combined with text features, their learned representations can achieve better performance. This proves that node content information is helpful for learning informative node representations. On Cora, GraRep can be observed to remarkably outperform LINE, which demonstrates the effectiveness of more global structural context for learning node representations. However, the performance of GraRep on Citeseer and Wikipedia is very poor. This possibly because, by naively combing representations learned from different k-step proximity, GraRep might "wash out" the effectiveness of the most important component, leading to unsatisfactory performance.

4.4.5 Convergence Analysis

We also carry out experiments to examine the convergence of solving the problem (4.9). The values of the objective function (4.8) from iteration 1 to iteration 50 on all four networks are given in Fig. 4.2. Fig. 4.2 shows that the algorithm can quickly convergence within only 5 iterations on all networks.

4.4.6 Parameter Sensitivity

In our algorithm, there are three parameters, k, μ and λ. We fix any two parameters and investigate the sensitivity of the third one in turns. Fig. 4.3(a)-4.3(c) show the



(a) Cora (b) Citeseer

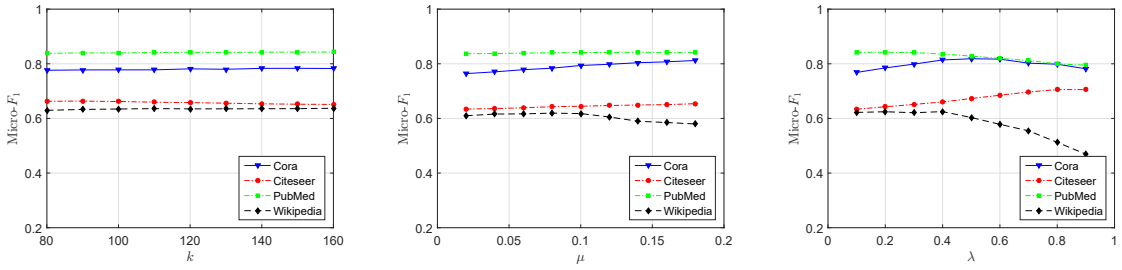
(c) PubMed (d) Wikipedia

Fig. 4.2 Convergence of the objective function

averaged Micro-F_1 values on all four networks by varying k, μ and λ, respectively. Here, we set the training ratio as 5%. From Fig. 4.3(a) and 4.3(b), we can see that, our HSCA algorithm is relatively robust to the change of parameters k and μ. As the value of λ increases, the change of Micro-F_1 follows different trends on different networks. Yet, overall, the best performance can be achieved when λ is set to be 0.4 or 0.5.

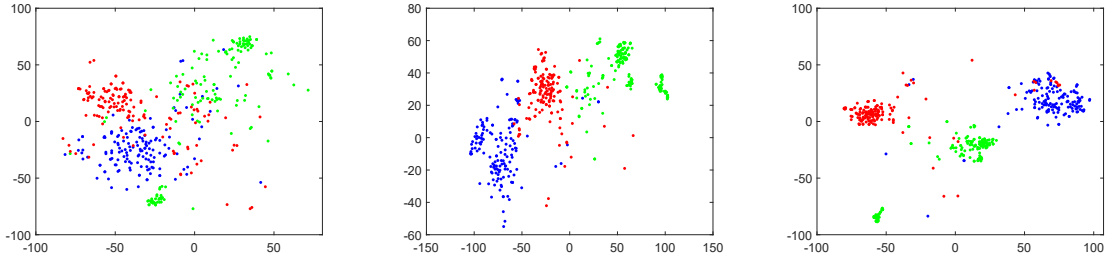
4.4.7 Visualization of Learned Representations

To validate the representations learned by our HSCA algorithm are more informative than those learned by TADW and DeepWalk, we project the representations learned by the three algorithms into 2-dimensional space using the t-SNE package [83]. We randomly select 150 nodes from three subcategories of Cora, including Genetic Algorithms, Rule Learning and Reinforcement Learning, and plot the visualization results



(a) k (b) μ (c) λ

Fig. 4.3 Micro-F_1 values with respect to different values of k, μ and λ



(a) DeepWalk (b) TADW (c) HSCA

Fig. 4.4 Visualization of network representations learned by different algorithms

in Fig. 4.4. By comparing Fig. 4.4(a) and Fig. 4.4(b), we can observe that different classes of data points in Fig. 4.4(b) are better separated from each other than those in Fig. 4.4(a). This indicates that with node content imported, TADW's representations are more informative than those learned by DeepWalk. Compared to Fig. 4.4(b), data points in Fig. 4.4(c) tend to have a cluster structure; data points belonging to the same class are closer to each other, while the ones falling into different classes are far apart separated. This observation confirms that, by preserving homophily, HSCA's representations have more discriminative power for node classification.

4.5 Conclusion

In this chapter, we proposed a new network embedding algorithm called HSCA that effectively augments information from (1) network homophily, (2) structural context, and (3) node content into a new matrix factorization objective function for learning

network representations. An efficient optimization algorithm is also derived to solve the objective function. Our HSCA algorithm effectively embeds a network into a single latent representation space that captures the interplay between the three information sources. The learned representations preserve both network structure and node content information, as well as follow the social homophily constraints in the new space. We validated the effectiveness of our HSCA algorithm on real-world networks. Experiments and visualization results demonstrate that the node representations learned by our HSCA algorithm can be more effectively used as features for tasks such as multi-class node classification, compared to those learned by the state-of-the-art network embedding algorithms.

4.6 Appendix

A. Calculation for $\nabla f(\boldsymbol{\omega})$

As $\tilde{\boldsymbol{x}}_{ij} = (H\boldsymbol{t}_j) \otimes \boldsymbol{e}_i = \boldsymbol{vec}(\boldsymbol{e}_i \boldsymbol{t}_j^{\mathrm{T}} H^{\mathrm{T}})$, we calculate

$$\sum_{i,j=1}^{|\mathcal{V}|} (\boldsymbol{\omega}^{\mathrm{T}} \tilde{\boldsymbol{x}}_{ij} - M_{ij}) \boldsymbol{e}_i \boldsymbol{t}_j^{\mathrm{T}} H^{\mathrm{T}} = F T^{\mathrm{T}} H^{\mathrm{T}}, \tag{4.30}$$

where $F_{ij} = \boldsymbol{\omega}^{\mathrm{T}} \tilde{\boldsymbol{x}}_{ij} - M_{ij}$, i.e., $F = W^{\mathrm{T}} H T - M$. Therefore,

$$\sum_{i,j=1}^{|\mathcal{V}|} (\boldsymbol{\omega}^{\mathrm{T}} \tilde{\boldsymbol{x}}_{ij} - M_{ij}) \boldsymbol{e}_i \boldsymbol{t}_j^{\mathrm{T}} H^{\mathrm{T}} = W^{\mathrm{T}} H T T^{\mathrm{T}} H^{\mathrm{T}} - M T^{\mathrm{T}} H^{\mathrm{T}}. \tag{4.31}$$

Using the identity $(B^{\mathrm{T}} \otimes A)\boldsymbol{vec}(X) = \boldsymbol{vec}(AXB)$, $\Lambda\boldsymbol{\omega}$ can be expanded as

$$\Lambda\boldsymbol{\omega} = (\mathrm{I}_k \otimes L)\boldsymbol{vec}(W^{\mathrm{T}}) = \boldsymbol{vec}(LW^{\mathrm{T}}). \tag{4.32}$$

Thus,

$$\nabla f(\boldsymbol{w}) = \boldsymbol{vec}(W^{\mathrm{T}} H T T^{\mathrm{T}} H^{\mathrm{T}} - M T^{\mathrm{T}} H^{\mathrm{T}}) + \mu \boldsymbol{vec}(LW^{\mathrm{T}}) + \lambda\boldsymbol{\omega}. \tag{4.33}$$

B. Calculation for $\nabla^2 f(\boldsymbol{\omega})\boldsymbol{s}$

After substituting $\tilde{\boldsymbol{x}}_{ij} = (H\boldsymbol{t}_j) \otimes \boldsymbol{e}_i$, we have

$$\sum_{i,j=1}^{|\mathcal{V}|} \tilde{\boldsymbol{x}}_{ij}\tilde{\boldsymbol{x}}_{ij}^{\mathrm{T}}\boldsymbol{s} = \sum_{i,j=1}^{|\mathcal{V}|} \left[(H\boldsymbol{t}_j\boldsymbol{t}_j^{\mathrm{T}}H^{\mathrm{T}}) \otimes (\boldsymbol{e}_i\boldsymbol{e}_i^{\mathrm{T}})\right]\boldsymbol{s}. \tag{4.34}$$

Let $S \in \mathbb{R}^{|\mathcal{V}|\times k}$ such that $\boldsymbol{s} = \boldsymbol{vec}(S)$. Similar to (4.32),

$$\left[(H\boldsymbol{t}_j\boldsymbol{t}_j^{\mathrm{T}}H^{\mathrm{T}}) \otimes (\boldsymbol{e}_i\boldsymbol{e}_i^{\mathrm{T}})\right]\boldsymbol{s} = \boldsymbol{vec}(\boldsymbol{e}_i\boldsymbol{e}_i^{\mathrm{T}}SH\boldsymbol{t}_j\boldsymbol{t}_j^{\mathrm{T}}H^{\mathrm{T}}). \tag{4.35}$$

We can calculate

$$\begin{aligned}\sum_{i,j=1}^{|\mathcal{V}|} \boldsymbol{e}_i\boldsymbol{e}_i^{\mathrm{T}}SH\boldsymbol{t}_j\boldsymbol{t}_j^{\mathrm{T}}H^{\mathrm{T}} &= \sum_{i=1}^{|\mathcal{V}|} \boldsymbol{e}_i \left(\sum_{j=1}^{|\mathcal{V}|} \boldsymbol{e}_i^{\mathrm{T}}SH\boldsymbol{t}_j\boldsymbol{t}_j^{\mathrm{T}}H^{\mathrm{T}}\right) \\ &= \sum_{i=1}^{|\mathcal{V}|} \boldsymbol{e}_i \left(\sum_{j=1}^{|\mathcal{V}|} U_{ij}\boldsymbol{t}_j^{\mathrm{T}}H^{\mathrm{T}}\right) \\ &= UT^{\mathrm{T}}H^{\mathrm{T}},\end{aligned} \tag{4.36}$$

where $U_{ij} = \boldsymbol{e}_i^{\mathrm{T}}SH\boldsymbol{t}_j$, i.e., $U = SHT$. Similar to (4.32),

$$\Lambda\boldsymbol{s} = (\mathrm{I}_k \otimes L)\boldsymbol{vec}(S) = \boldsymbol{vec}(LS). \tag{4.37}$$

Hence, (4.18) can be finally rewritten as

$$\begin{aligned}\nabla^2 f(\boldsymbol{\omega})\boldsymbol{s} &= \boldsymbol{vec}(UT^{\mathrm{T}}H^{\mathrm{T}}) + \mu\boldsymbol{vec}(LS) + \lambda\boldsymbol{s} \\ &= \boldsymbol{vec}(SHTT^{\mathrm{T}}H^{\mathrm{T}}) + \mu\boldsymbol{vec}(LS) + \lambda\boldsymbol{s}.\end{aligned} \tag{4.38}$$

C. Calculation for $\nabla g(\boldsymbol{\eta})$

As $\tilde{\boldsymbol{y}}_{ij} = \boldsymbol{t}_j \otimes \boldsymbol{w}_i = \boldsymbol{vec}(\boldsymbol{w}_i\boldsymbol{t}_j^{\mathrm{T}})$, we calculate

$$\sum_{i,j=1}^{|\mathcal{V}|} (\boldsymbol{\eta}^{\mathrm{T}}\tilde{\boldsymbol{y}}_{ij} - M_{ij})\boldsymbol{w}_i\boldsymbol{t}_j^{\mathrm{T}} = WFT^{\mathrm{T}} = WW^{\mathrm{T}}HTT^T - WMT^T. \tag{4.39}$$

where $F_{ij} = \boldsymbol{\eta}^{\mathrm{T}}\tilde{\boldsymbol{y}}_{ij} - M_{ij}$ i.e., $F = W^{\mathrm{T}}HT - M$. Then,

$$\Omega\boldsymbol{\eta} = ((TLT^{\mathrm{T}}) \otimes \mathrm{I}_k)\boldsymbol{vec}(H) = \boldsymbol{vec}(H(TLT^{\mathrm{T}})^{\mathrm{T}}). \tag{4.40}$$

Thus,

$$\nabla g(\boldsymbol{\eta}) = \boldsymbol{vec}(WW^{\mathrm{T}}HTT^{\mathrm{T}} - WMT^{\mathrm{T}}) + \mu\boldsymbol{vec}(H(TLT^{\mathrm{T}})^{\mathrm{T}}) + \lambda\boldsymbol{\eta}. \tag{4.41}$$

D. Calculation for $\nabla^2 g(\boldsymbol{\eta})\boldsymbol{s}$

After substituting $\tilde{\boldsymbol{y}}_{ij} = \boldsymbol{t}_j \otimes \boldsymbol{w}_i$, we have

$$\sum_{i,j=1}^{|\mathcal{V}|} \tilde{\boldsymbol{y}}_{ij}\tilde{\boldsymbol{y}}_{ij}^{\mathrm{T}}\boldsymbol{s} = \sum_{i,j=1}^{|\mathcal{V}|} \left[(\boldsymbol{t}_j\boldsymbol{t}_j^{\mathrm{T}}) \otimes (\boldsymbol{w}_i\boldsymbol{w}_i^{\mathrm{T}})\right]\boldsymbol{s}. \tag{4.42}$$

Let $S \in \mathbb{R}^{k \times f_t}$, such that $\boldsymbol{s} = \boldsymbol{vec}(S)$. Similar to (4.32),

$$\left[(\boldsymbol{t}_j\boldsymbol{t}_j^{\mathrm{T}}) \otimes (\boldsymbol{w}_i\boldsymbol{w}_i^{\mathrm{T}})\right]\boldsymbol{s} = \boldsymbol{vec}(\boldsymbol{w}_i\boldsymbol{w}_i^{\mathrm{T}}S\boldsymbol{t}_j\boldsymbol{t}_j^{\mathrm{T}}). \tag{4.43}$$

Similar to (4.36),

$$\sum_{i,j=1}^{|\mathcal{V}|} \boldsymbol{w}_i\boldsymbol{w}_i^{\mathrm{T}}S\boldsymbol{t}_j\boldsymbol{t}_j^{\mathrm{T}} = WVT^{\mathrm{T}}, \tag{4.44}$$

where $V_{ij} = \boldsymbol{w}_i^{\mathrm{T}}S\boldsymbol{t}_j$, i.e., $V = W^{\mathrm{T}}ST$. Similar to (4.32),

$$\Omega\boldsymbol{s} = ((TLT^{\mathrm{T}}) \otimes \mathrm{I}_k)\boldsymbol{vec}(S) = \boldsymbol{vec}(S(TLT^{\mathrm{T}})^{\mathrm{T}}). \tag{4.45}$$

Hence, (4.19) can be rewritten as

$$\begin{aligned}\nabla^2 g(\boldsymbol{\eta})\boldsymbol{s} &= \boldsymbol{vec}(WVT^{\mathrm{T}}) + \mu\boldsymbol{vec}(S(TLT^{\mathrm{T}})^{\mathrm{T}}) + \lambda\boldsymbol{s} \\ &= \boldsymbol{vec}(WW^{\mathrm{T}}STT^{\mathrm{T}}) + \mu\boldsymbol{vec}(S(TLT^{\mathrm{T}})^{\mathrm{T}}) + \lambda\boldsymbol{s}.\end{aligned} \tag{4.46}$$

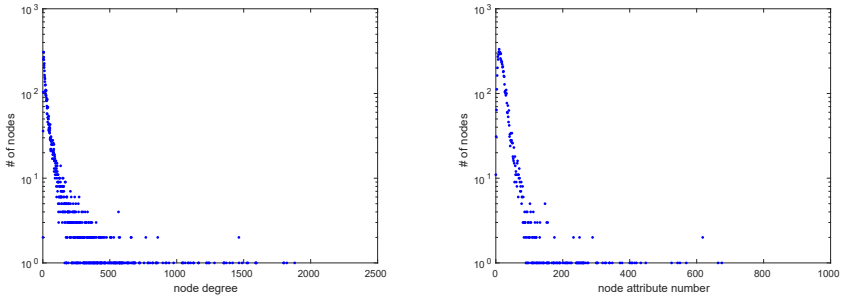
Chapter 5

Attributed Network Embedding via Subspace Discovery

5.1 Introduction

With the ubiquity of network data across a diverse set of fields, network embedding has aroused a surge of research interests because it provides a revolutionized solution to network data analysis. Network embedding aims at learning lower-dimensional vector representations of network nodes, which allows downstream tasks, such as node classification, link prediction, or community detection, to be efficiently solved in the new vector space by simply applying traditional vector-based machine learning algorithms. The essential principle is to learn low-dimensional node representations that are able to preserve the proximity in the original network space, such that nodes with larger proximity would have similar vector representations.

Traditionally, network embedding techniques (e.g., DeepWalk [63], LINE [76], node2vec [19], GraRep [8]) mainly focus on preserving local or global network structure, making structurally similar nodes to be represented close to each other in the new vector space. In real-world scenarios, however, apart from structural information, network nodes are often associated with content, such as text features in citation networks, or user profiles in social networks. Node attributes not only provide direct evidence about content-level proximity but also influence the forming of interactions between nodes.



(a) degree (b) attribute number

Fig. 5.1 Node distributions with respect to node degree and the number of node attributes on the Flickr network.

It has been shown that, by jointly learning node embeddings with network structure and node attributes, attributed network embedding can achieve better performance. For example, TADW [96] exploits inductive matrix factorization [57] to incorporates textual features of nodes into the learning of node representations, showing better performance than DeepWalk. HSCA [101] learns informative node representations through simultaneously integrating structural context with node content while enforcing the homophily property in the representation space.

Despite existing methods' demonstrated effectiveness in improving embedding performance, attributed network embedding is confronted with two major challenges. The first challenge lies in the fact that real-world networks are often very sparse; observable links connecting network nodes are rather limited, and node content information is sparse or incomplete. For example, in Figs. 5.1(a) and 5.1(b) we plot the node distributions with respect to node degree and attribute number of a Flickr network, which contains 7,575 users and 12,047 user tag attributes. The vertical axis of Figs. 5.1(a) and 5.1(b) indicates the number of nodes, and the horizontal axis indicates node degree, and the number of node attributes, respectively. Figs. 5.1(a) and 5.1(b) show that the two plots both follow power-law distributions [20], where only a small number of nodes have a high degree or a large number of node attributes. In other words, a significant number of nodes tend to have very few or missing links as well as incomplete attribute information. The sparsity in network structure and node attributes can negatively impact the estimation of structural-level and content-level proximity, thus further deteriorating the performance of network embedding.

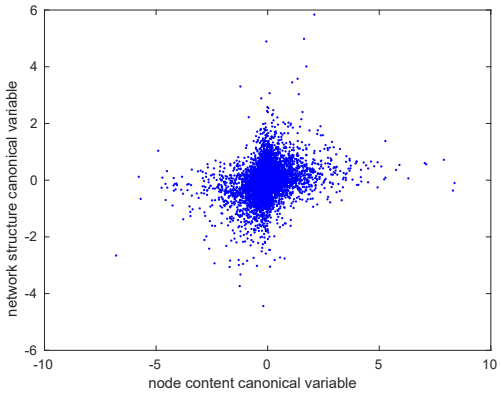


Fig. 5.2 The scatter plot of node content canonical variable and network structure canonical variable.

The second challenge stems from the heterogeneity between network structure and node attributes. According to the social influence principle, network structure often exhibits certain correlations with node attributes [65]. For example, in social networks, users who are connected are more likely to have similar attributes. However, because network structure and node attributes are two heterogeneous information sources, inevitably, there is a discrepancy between the two different feature spaces. As has been revealed in previous studies [7, 74], social network users with similar attributes, or antithetical attributes are both likely to connect to one another. To illustrate this point, let us consider a case study on the Flickr network, where we conduct correlation analysis between node representations constructed from network structure and those from node attributes. Specifically, we obtain two sets of 100-dimensional node representations by performing Singular Value Decomposition (SVD) on network adjacent matrix and node content features, and then conduct Canonical Correlation Analysis (CCA) [25] on the two sets of generated representations. Fig. 5.2 shows the scatter plot about node content canonical variable and network structure canonical variable, which are the linear combination of 100-dimensional structure representation and attribute representation, respectively, with maximum correlation between them. The correlation between the two canonical variables is 0.5544. As shown in the figure, the two canonical variables are not well correlated. The CCA results show that network structure and node content features do not always exhibit strong linear correlations. Such a discrepancy adds extra difficulties in fusing node attributes with network structure in a proper way that they

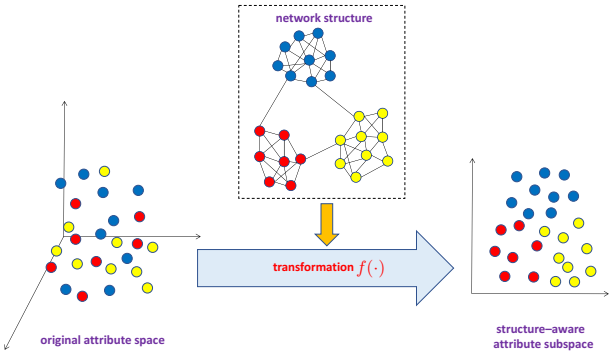


Fig. 5.3 The working mechanism of the proposed attri2vec algorithm. A transformation $f(\cdot)$ guided by network structure is performed from the original node attribute space to seek a structure-aware attribute subspace, where node attributes and network structure can better compliment each other in a more consistent way towards learning high-quality node representations.

complement rather than deteriorate each other towards learning high-quality node representations.

Motivated by aforementioned observations, we propose a unified attributed network embedding framework, called attri2vec, which learns node representations by discovering the *latent subspace* of node attributes that well respects network structure. The working mechanism of attri2vec is illustrated in Fig. 5.3. The core idea is to perform a transformation $f(\cdot)$ guided by network structure from the original node attribute space, where node attribute distributions are not well aligned with network structure, to a structure-aware attribute subspace, where structurally similar nodes are located closely to each other. The latent attribute subspace can better respect network structure in a consistent way towards learning high-quality node representations. Following [63], attri2vec generates random walks to capture structural context. After random walks are generated, attri2vec uses the image $f(\boldsymbol{x}_i)$ of node v_i with attribute $\boldsymbol{x}_i$ in the new attribute subspace to predict its context nodes collected from random walk sequences. In this way, network structure is seamlessly encoded into the new attribute subspace by allowing nodes sharing similar neighbors to be located closely to each other. As the attribute subspace also preserves node attribute proximity, the learned node representations are informative enough to capture both the structure-level and node content-level proximity. As attri2vec infers attribute subspace that well respects

network structure through directly learning the transformation $f(\cdot)$ from data, the weights for the respective contributions made by attributes and network structure for embedding learning is automatically learned from the data itself, which avoids the laborious weight parameter turning. In our prior work [102], we have proposed the UPP-SNE (User Profile Preserving Social Network Embedding) algorithm that learns user embeddings in social networks by leveraging both the network structure and user profile features. It constructs user embeddings by performing a non-linear kernel mapping [64] on user profile features, in which noisy information in user profiles is filtered out. Initial experimental results on social networks have shown that user embeddings learned by UPP-SNE can achieve substantial performance gains in node classification and clustering tasks.

In this work, we generalize the idea of UPP-SNE to tackle generic network embedding problems and also to cope with large-scale networks. First, we investigate four different types of linear or non-linear transformation functions for discovering the latent, structure-aware attribute subspace, and empirically validate their effectiveness on various types of networks. Second, we develop an efficient stochastic gradient descent algorithm to solve the optimization problem, which makes attri2vec able to be efficiently trained in an online mode. At each iteration, after sampling a node context pair, we only update the parameters related to the corresponding partial objective with gradient descent. Compared with the gradient descent strategy used by UPP-SNE, the stochastic gradient descent is not only more efficient, but also able to effectively mitigate the local minima problem, thus discovering more accurate solutions for attri2vec. Third, given that attri2vec learns a network structure aware mapping on node attributes, it provides an alternative way to solve the out-of-sample problem. In other words, on the dynamically evolving networks, when new nodes join in, rather than learning representations for all nodes from the scratch, attri2vec can construct representations for new coming nodes from their available node attributes via the mapping function learned from previous network snapshots. We also conduct node classification and link prediction experiments to study the effectiveness of attri2vec in solving the out-of-sample problem.

The contribution of this chapter can be summarized as follows:

(1) We propose a unified framework for attributed network embedding, attri2vec, that learns node embeddings by discovering a latent, structure aware attribute subspace and formulate an optimization problem.

(2) We develop an efficient stochastic gradient descent algorithm to solve the optimization problem, which not only remedies the local minima problem, but also improves the effectiveness and efficiency, thus making attri2vec able to scale to large-scale networks.

(3) We validate the effectiveness and efficiency of attri2vec on four real-world networks of various types. Extensive experiments on multi-class node classification, node clustering and link prediction tasks demonstrate that attri2vec learns node embeddings of better quality than state-of-the-art baselines, with an additional advantage to solve the out-of-sample problem.

5.2 Problem Definition and Preliminaries

In this section, we give a formal problem definition of attributed network embedding and the preliminary background about DeepWalk.

5.2.1 Problem Definition

We assume that an attributed information network is given as $G = (\mathcal{V}, \mathcal{E}, X)$, with $\mathcal{V}$ denoting the set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denoting the set of edges. In G, $X \in \mathbb{R}^{m \times |\mathcal{V}|}$ is the node attribute matrix, with the i-th column of X, $\boldsymbol{x}_i \in \mathbb{R}^m$, denoting the m-dimensional content feature vector for node $v_i \in \mathcal{V}$. By taking advantage of network structure and node attributes, the task of attributed network embedding is to effectively embed nodes into a low-dimensional vector space so as to obtain the node images $\Phi(v_i) \in \mathbb{R}^d$ in the latent space as node vector-format representations.

The learned vector-format node representations $\Phi(v_i)$ are expected to embody the following properties: (1) *low-dimensional*, for the efficiency of the subsequent analytic tasks, the dimension of $\Phi(v_i)$, d, should be much smaller than the dimension

of the original adjacency matrix representation, $|\mathcal{V}|$; (2) *structure preserving*, network structure is well encoded into node representations, i.e., two nodes with structural proximity should be embedded closely in the embedding space, (3) *attribute preserving*, node attribute proximity should be well captured so that it complements rather than deteriorates network structure.

5.2.2 Preliminaries: DeepWalk

For learning node embeddings, DeepWalk [63] exploits random walk to explore node neighborhood structure. By taking an analogy between truncated random walk sequences and sentences in natural language, DeepWalk extends the Skip-Gram model [55] from word representation learning to network embedding, with the expectation that two nodes sharing similar neighborhood structure have similar low-dimensional representations. Given a random walk node sequence $\mathcal{S} = \{v_{r_1}, v_{r_2}, \cdots, v_{r_{|\mathcal{S}|}}\}$ with length $|\mathcal{S}|$, DeepWalk learns embedding $\Phi(v_{r_i})$ for $v_{r_i} (1 \leq i \leq L)$, by using $\Phi(v_{r_i})$ to predict v_{r_i}'s context in t-window size:

$$\max_{\Phi} \log \Pr(\{v_{r_{i-t}}, \cdots, v_{r_{i+t}}\} \setminus v_i | \Phi(v_{r_i})). \tag{5.1}$$

By exploiting the conditional independence assumption, the probability $\Pr(\{v_{r_{i-t}}, \cdots, v_{r_{i+t}}\} \setminus v_{r_i} | \Phi(v_{r_i}))$ can be calculated as

$$\Pr(\{v_{r_{i-t}}, \cdots, v_{r_{i+t}}\} \setminus v_{r_i} | \Phi(v_{r_i})) = \prod_{j=i-t, j \neq i}^{i+t} \Pr(v_{r_j} | \Phi(v_{r_i})). \tag{5.2}$$

To model the probability $\Pr(v_{r_j} | \Phi(v_{r_i}))$, an one-hidden-layer neural network is utilized:

$$\Pr(v_{r_j} | \Phi(v_{r_i})) = \frac{\exp(\Phi(v_{r_i}) \cdot \boldsymbol{w}_{r_j}^{out})}{\sum_{k=1}^{|V|} \exp(\Phi(v_{r_i}) \cdot \boldsymbol{w}_k^{out})}, \tag{5.3}$$

where node embedding $\Phi(v_{r_i})$ is the representation in the hidden layer, and $\boldsymbol{w}_{r_j}^{out}$ is the r_j-th column of $W^{out} \in \mathbb{R}^{d \times |\mathcal{V}|}$ (the weight matrix from the hidden layer to the output

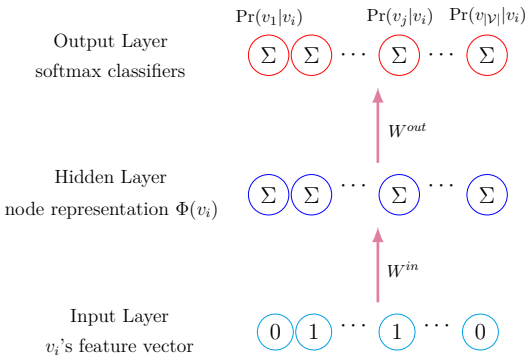


Fig. 5.4 The architecture of attri2vec. For each node context pair (v_i, v_j), attri2vec learns node representations by modeling $\Pr(v_j|v_i)$. attri2vec firstly constructs node representations in the hidden layer by performing a linear or non-linear transformation on v_i's content attributes, then uses the hidden layer representation to predict the probability $\Pr(v_j|v_i)$ with softmax.

layer). The node embedding $\Phi(v_i)$ is constructed as

$$\Phi(v_{r_i}) = {W^{in}}^{\mathrm{T}} \boldsymbol{p}_{r_i} = \boldsymbol{w}^{in}_{r_i}, \tag{5.4}$$

where $\boldsymbol{p}_{r_i} \in \mathbb{R}^{|\mathcal{V}|}$ is the one-hot representation of v_{r_i} with the value of r_i-th dimension being 1 and the values of other dimensions being 0, and $\boldsymbol{w}^{in}_{r_i}$ is the transpose of the r_i-th row of $W^{in} \in \mathbb{R}^{|\mathcal{V}| \times d}$ (the weight matrix from the input layer to hidden layer).

5.3 The attri2vec Framework

In DeepWalk, the node embeddings $\Phi(v_i)$ are learned from scratch, which is independent of node attributes. However, as proved in previous work [96, 101], node content features are crucially useful for enhancing the only structure preserving network embedding. To obtain more informative node embeddings, here, we adapt the DeepWalk's architecture to capture both network structure and node content features.

To learn node embeddings $\Phi(\cdot)$, DeepWalk solves the following joint optimization problem:

$$\min_{W^{in}, W^{out}} \mathcal{O}(W^{in}, W^{out}), \tag{5.5}$$

where

$$\begin{aligned}
&\mathcal{O}(W^{in}, W^{out}) \\
&= -\sum_{\mathcal{S}}\sum_{i=1}^{|\mathcal{S}|}\log \Pr(\{v_{r_{i-t}}, \cdots, v_{r_{i+t}}\} \setminus v_{r_i} | \Phi(v_i)), \\
&= -\sum_{\mathcal{S}}\sum_{i=1}^{|\mathcal{S}|}\sum_{j=i-t, j\neq i}^{i+t} \Pr(v_{r_j} | \Phi(v_{r_i})).
\end{aligned} \tag{5.6}$$

For convenience, we rewrite the objective function $\mathcal{O}(W^{in}, W^{out})$ as

$$\mathcal{O}(W^{in}, W^{out}) = -\sum_{i=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{V}|} n(v_i, v_j) \log \Pr(v_j | \Phi(v_i)), \tag{5.7}$$

where $n(v_i, v_j)$ is the number of times that v_j occurs in v_i context within t-window size in the generated set of random walks. Using Eq. (5.3), we model the probability $\Pr(v_j|\Phi(v_i))$ as

$$\Pr(v_j | \Phi(v_i)) = \frac{\exp(\Phi(v_i) \cdot \boldsymbol{w}_j^{out})}{\sum_{k=1}^{|\mathcal{V}|} \exp(\Phi(v_i) \cdot \boldsymbol{w}_k^{out})}. \tag{5.8}$$

Here, different from DeepWalk that constructs $\Phi(v_i)$ with a linear transformation from node v_i's one-hot representation $\boldsymbol{p}_i$, to incorporate node content features, we construct $\Phi(v_i)$ from node v_i's content features $\boldsymbol{x}_i \in \mathbb{R}^m$ with a linear or non-linear transformation:

$$\Phi(v_i) = f(W^{in^{\mathrm{T}}} \boldsymbol{x}_i), \tag{5.9}$$

where $W^{in} \in \mathbb{R}^{m \times d}$ is the input-hidden weight matrix and we use $\boldsymbol{w}_j^{in}$ to denote the j-th column of W^{in}. The architecture of attri2vec is given in Fig 5.4. To obtain the embedding $\Phi(v_i)$ for each node $v_i \in \mathcal{V}$, a series of mapping function $f(\cdot)$ are investigated:

(1) **linear mapping**:

$$\begin{aligned}
f(W^{in^{\mathrm{T}}} \boldsymbol{x}_i) &= W^{in^{\mathrm{T}}} \boldsymbol{x}_i \\
&= \left[\boldsymbol{w}_1^{in^{\mathrm{T}}} \boldsymbol{x}_i, \cdots, \boldsymbol{w}_d^{in^{\mathrm{T}}} \boldsymbol{x}_i\right]^{\mathrm{T}};
\end{aligned} \tag{5.10}$$

(2) **rectified linear unit (ReLU) mapping**:

$$f(W^{in^{\mathrm{T}}} \boldsymbol{x}_i) = \left[\max(0, \boldsymbol{w}_1^{in^{\mathrm{T}}} \boldsymbol{x}_i), \cdots, \max(0, \boldsymbol{w}_d^{in^{\mathrm{T}}} \boldsymbol{x}_i)\right]^{\mathrm{T}}; \tag{5.11}$$

(3) **approximated kernel mapping** used in UPP-SNE [102]:

$$f(W^{in^{\mathrm{T}}}\boldsymbol{x}_i) = \frac{1}{\sqrt{m}}\Big[\cos(\boldsymbol{w}_1^{in^{\mathrm{T}}}\boldsymbol{x}_i), \cdots, \cos(\boldsymbol{w}_{d/2}^{in\ \mathrm{T}}\boldsymbol{x}_i), \sin(\boldsymbol{w}_1^{in^{\mathrm{T}}}\boldsymbol{x}_i), \cdots, \sin(\boldsymbol{w}_{d/2}^{in\ \mathrm{T}}\boldsymbol{x}_i)\Big]^{\mathrm{T}}; \tag{5.12}$$

(4) **sigmoid mapping**:

$$f(W^{in^{\mathrm{T}}}\boldsymbol{x}_i) = \Big[1/(1+\exp(-\boldsymbol{w}_1^{in^{\mathrm{T}}}\boldsymbol{x}_i)), \cdots, 1/(1+\exp(-\boldsymbol{w}_d^{in^{\mathrm{T}}}\boldsymbol{x}_i))\Big]^{\mathrm{T}}. \tag{5.13}$$

We denote the attri2vec algorithm with the four mapping functions as attri2vec-linear, attri2vec-ReLU, attri2vec-kernel and attri2vec-sigmoid respectively.

We adopt the stochastic gradient descent to solve the optimization problem in Eq. (5.7). After randomly sampling a node context pair (v_i, v_j) according to the frequency distribution in $n(v_i, v_j)$, we try to update parameters for reducing the value the following partial objective:

$$\begin{aligned}\mathcal{O}_{ij}(W^{in}, W^{out}) &= -\log \Pr(v_j|\Phi(v_i)) \\ &= -\Phi(v_i)\cdot \boldsymbol{w}_j^{out} + \log \sum_{k=1}^{|\mathcal{V}|} \exp(\Phi(v_i)\cdot \boldsymbol{w}_k^{out}).\end{aligned} \tag{5.14}$$

In Eq. (5.14), the calculation for the partial objective $\mathcal{O}_{ij}$ involves the summation over all nodes in the given network, which is prohibitively time-consuming. To make a speed-up, we adopt negative sampling [21] to approximate the partial objective:

$$\mathcal{O}_{ij}(W^{in}, W^{out}) = -\log \sigma(\Phi(v_i)\cdot \boldsymbol{w}_j^{out}) - \sum_{k=1}^{K} \log \sigma(-\Phi(v_i)\cdot \boldsymbol{w}_{N_{i,k}}^{out}), \tag{5.15}$$

where $\sigma(\cdot)$ is the sigmoid function with $\sigma(x) = 1/(1+\exp(-x))$, $N_{i,k}$ is the index for the k-th negative node sampled for node v_i and K is the number of sampled negative nodes. Exploiting the approximated partial objective $\mathcal{O}_{ij}(W^{in}, W^{out})$ in Eq. (5.15),

the parameters are updated by:

$$\begin{aligned}\boldsymbol{w}_p^{in} &= \boldsymbol{w}_p^{in} - \eta\frac{\partial \mathcal{O}_{ij}}{\partial \boldsymbol{w}_p^{in}}, \quad p \in \{1, 2, \cdots, d\};\\ \boldsymbol{w}_q^{out} &= \boldsymbol{w}_q^{out} - \eta\frac{\partial \mathcal{O}_{ij}}{\partial \boldsymbol{w}_q^{out}}, \quad q \in \{j\} \cup \{N_{i,1}, N_{i,2}, \cdots, N_{i,K}\}.\end{aligned} \tag{5.16}$$

The gradients are calculated as

$$\begin{aligned}\frac{\partial \mathcal{O}_{ij}}{\boldsymbol{w}_p^{in}} &= -\sigma(-\Phi(v_i)\cdot \boldsymbol{w}_j^{out})\frac{\partial \Phi(v_i)}{\partial \boldsymbol{w}_p^{in}}\boldsymbol{w}_j^{out}\\ &\quad + \sum_{k=1}^{K}\sigma(\Phi(v_i)\cdot \boldsymbol{w}_{N_{i,k}}^{out})\frac{\partial \Phi(v_i)}{\partial \boldsymbol{w}_p^{in}}\boldsymbol{w}_{N_{i,k}}^{out};\end{aligned} \tag{5.17}$$

$$\begin{aligned}\frac{\partial \mathcal{O}_{ij}}{\boldsymbol{w}_q^{out}} &= -\mathbf{1}_q(j)\sigma(-\Phi(v_i)\cdot \boldsymbol{w}_j^{out})\Phi(v_i)\\ &\quad + \sum_{k=1}^{K}\mathbf{1}_q(N_{i,k})\sigma(\Phi(v_i)\cdot \boldsymbol{w}_{N_{i,k}}^{out})\Phi(v_i).\end{aligned} \tag{5.18}$$

where $\frac{\partial \Phi(v_i)}{\partial \boldsymbol{w}_p^{in}}$ is the $m \times d$ Jacobian matrix under different mappings for constructing $\Phi(v_i)$. $\mathbf{1}_q(\cdot)$ is an indicator function, which is defined as

$$\mathbf{1}_q(x) = \begin{cases}1 & \text{if } x = q,\\ 0 & \text{if } x \neq q.\end{cases} \tag{5.19}$$

The workflow of the attri2vec algorithm is given in Algorithm 1. Firstly, in line 1, a set of random walk sequences are generated on the given attributed network G, by starting random walk with length l from each node for γ times. Then, in line 2, the statistics $n(v_i, v_j)$ is calculated from the generated random walks with a fixed window size t. In line 3, we initialize W^{in} with random numbers, and initialize W^{out} with 0. After that, in line 4-8, the parameters are updated iteratively with stochastic gradient descent. At each iteration, a node context pair (v_i, v_j) is sampled from the distribution of $n(v_i, v_j)$ with the alias table method [40], which takes only $O(1)$ time; parameters are then updated for reducing the value of the partial objective O_{ij} in Eq. (5.15).

By leveraging the sparsity of node attributes, the time complexity of line 7 is only $O(\bar{m}d)$ with $\bar{m}$ being the averaged number of non-zero elements of $\boldsymbol{x}_i$. The number

Algorithm 1 attri2vec: a Unified Framework for Attributed Network Embedding

Input:
An attributed network $G = (\mathcal{V}, \mathcal{E}, X)$;
Output:
Node embedding $\Phi(\cdot)$ for each $v_i \in \mathcal{V}$;
$\mathbb{S} \leftarrow$ generate a set of random walks on G;
$n(v_i, v_j) \leftarrow$ count frequency of node context pairs (v_i, v_j) in $\mathbb{S}$;
$\{W^{in}, W^{out}\} \leftarrow$ initialize parameters;
repeat
  $(v_i, v_j) \leftarrow$ sample a node context pair according to the distribution of $n(v_i, v_j)$;
  $\{v_{N_{i,1}}, \cdots, v_{N_{i,K}}\} \leftarrow$ draw K negative nodes;
  $\{W^{in}, W^{out}\} \leftarrow$ update parameters with Eq. (5.16);
until maximum number of iterations expire;
construction node embedding $\Phi(\cdot)$ with W^{in} and the selected mapping function;
return $\Phi(\cdot)$;

Table 5.1 Summary of Four Real-world Networks

	Citeseer	DBLP	Facebook	Flickr
$\lvert\mathcal{V}\rvert$	3,312	18,448	4,039	7,575
$\lvert\mathcal{E}\rvert$	4,732	45,611	88,234	239,738
m	3,703	2,476	1,403	12,047
$nnz(X)$	105,165	103,130	31,656	182,517
# of Class	6	4	4	9

of iterations is at the scale of the number of non-zero $n(v_i, v_j)$, which is up bounded by $2\gamma lt|\mathcal{V}|$. Taking γ, l and t as constants, attri2vec has an overall time complexity of $O(\bar{m}d|\mathcal{V}|)$, which is linear to the number of nodes. This guarantees its efficiency and scalability over large-scale networks.

5.4 Experiments

In this section, we report experimental results on various types of real-world networks to evaluate the effectiveness and efficiency of the proposed attri2vec algorithm.

5.4.1 Benchmark Networks

In our experiments, four real-world attributed networks are used. Their details are as following:

- **Citeseer**. The Citeseer network[1] includes 3,312 scientific publications from six categories. There exists 4,732 citations among these papers. Each paper is represented by a 3,703-dimensional binary vector, with each entry representing the presence/absence of the corresponding word.
- **DBLP**. The DBLP network is a subgraph of the DBLP bibliographic network[2]. To construct the DBLP network, we extract papers from the four research areas: *Database*, *Data Mining*, *Artificial Intelligence*, *Computer Vision*, according to papers' venue information and remove papers with no citations. The DBLP network contains 18,448 papers and 45,661 citations. From paper titles, we construct 2,476-dimensional binary node feature vectors, with each element indicating the presence/absence of the corresponding word.
- **Facebook**. The Facebook network is merged from 10 Facebook ego-networks from SNAP[3]. There are 4,039 users and 88,234 friendship relations. Each user is described by a 1403-dimensional bag-of-words vector from tree-structured user profiles [39]. Users' education types are used as labels.
- **Flickr**. The Flickr network[4] is extracted from the Flick online photo sharing platform, which includes 7,575 users and 239,738 follower-followee relations. These users join in nine predefined interest groups. Users' features are described by the tags of their images. Each user is represented by a 12,047-dimensional binary vector, according to the occurrence/absence of the corresponding tag.

For the above networks, the direction of links is ignored. Their statistics are summarized in Table 5.1.

[1]https://linqs.soe.ucsc.edu/data
[2]https://aminer.org/citation (Version 3 is used)
[3]https://snap.stanford.edu/data/
[4]http://people.tamu.edu/~xhuang/Code.html

5.4.2 Baseline Methods

We compare the attri2vec algorithm (attri2vec-linear, attri2vec-ReLU, attri2vec-kernel and attri2vec-sigmoid) with the following baseline methods:

- **DeepWalk** [63]/**node2vec** [19]. They both learn node representations by preserving the similarity between nodes sharing similar contexts in random walks. node2vec is equivalent to DeepWalk with the default parameter setting $p = q = 1$.
- **LINE-1** [76]. LINE-1 denotes the version of LINE that learns node representations by modeling the first-order proximity.
- **LINE-2** [76]. LINE-2 denotes the version of LINE that learns node representations by preserving the second-order proximity.
- **SDNE** [87]. SDNE learns deep node representations with a semi-supervised autoencoder, which captures both the first-order and the second-order proximity.
- **TADW** [96]. TADW imports node text features into network embedding though inductive matrix factorization [57].
- **SNE** [44]. SNE constructs node representations by aggregating structure preserving ID embedding and attribute embedding.
- **MVC-DNE** [97]. MVC-DNE fuses network structure and node content into node embeddings through deep autoencoder cross-view learning.

Among the above baseline methods, DeepWalk, LINE-1, LINE-2 and SDNE only leverage network structure to learn node representations, while TADW, SNE and MVC-DNE integrate network structure with node content features.

5.4.3 Experimental Settings

For DeepWalk and the proposed attri2vec algorithm, we set the length of random walks l as 100, the number of walks starting at per node γ as 40, and the window size t as 10.

For fair comparisons, DeepWalk is trained in a same way as attri2vec, where a set of node context pairs are first collected from random walks, and parameters are then updated with stochastic gradient descent, with a node context pair sampled at each iteration. Negative sampling is adopted by DeepWalk, LINE-1, LINE-2, and attri2vec, where the number of negative samples K is set to 5 uniformly. For the four stochastic gradient descent based algorithms, we set the maximum number of iterations as 100 million, and gradually decrease the learning rate η from 0.025 to 2.5×10^{-6}. When we run attri2vec-linear and attri2vec-ReLU on Flickr, to avoid gradient explosion, we set the initial learning rate to 0.005, and gradually decrease the learning rate to 5×10^{-7}. Parameters of TADW are set to their default values. For SDNE, its hyper parameter α and ν are both set to 0.01, and β is set to 10. For SDNE, we set the number of neurons at each layer as 3312-128, and 18,448-512-128, 4039-128, 7575-128 for Citeseer, DBLP, Facebook and Flickr respectively. For MVC-DNE, on Citeseer, DBLP, Facebook and Flickr, in structure view, we respectively set the number of neurons at each layer as 3312-64, 18,448-256-64, 4039-64, and 7575-64; in the node attribute view, the number of neurons at each layer is respectively set to 3,703-64, 2,476-64, 1,403-64, and 12,047-256-64. For SDNE and MVC-DNE, 500 epochs are respectively run for pre-training and parameter fine-tuning. Other parameters of SDNE and MVC-DNE are set according to [97]. For SNE, default parameters are used. For attri2vec and all baseline methods, the dimension of learned node representations is set to 128.

5.4.4 Node Classification Experiments

To evaluate the quality of node representations learned by different network embedding algorithms, we carry out node classification experiments by taking node representations as features. We randomly select a ratio of training samples and use them to train an SVM classifier (implemented by LIBLINEAR [15]), and then test its performance on the held-out samples. We vary the training ratio from 10% to 90% with a step of 10%. For each training ratio, we repeat the random training and test set split for 10 times and report the averaged Micro-F_1 and Macro-F_1 values as final results. Tables 5.2-5.5 report the node classification results on Citeseer, DBLP, Facebook and Flickr. For

Table 5.2 Node Classification Results on Citeseer

	Training Ratio	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F_1(%)	DeepWalk	51.05	54.72	57.89	58.52	59.48	59.59	59.49	59.71	61.48
	LINE-1	43.73	48.51	50.17	52.10	53.22	54.03	54.67	54.24	56.77
	LINE-2	30.64	35.80	39.28	41.66	42.57	43.72	44.78	43.85	46.80
	SDNE	34.96	38.97	42.20	44.13	44.46	45.48	45.69	44.29	45.95
	TADW	61.21	64.05	64.88	65.20	65.72	65.65	65.84	65.94	67.22
	SNE	32.31	37.61	43.46	46.33	47.23	48.81	48.68	49.91	50.73
	MVC-DNE	50.73	54.95	59.81	63.09	64.57	65.92	66.90	67.07	68.07
	attri2vec-linear	52.69	54.47	62.74	65.72	66.97	67.28	67.62	68.35	69.94
	attri2vec-ReLU	53.47	55.82	61.94	65.71	67.25	67.69	68.04	68.56	71.06
	attri2vec-kernel	**62.04**	**65.87**	**67.56**	69.03	70.21	69.63	70.16	70.74	72.24
	attri2vec-sigmoid	59.47	60.51	65.51	**69.34**	**70.27**	**70.72**	**71.63**	**71.53**	**73.17**
Macro-F_1(%)	DeepWalk	47.74	51.23	53.52	53.73	54.08	53.87	53.44	53.54	55.41
	LINE-1	41.29	45.27	46.55	47.77	48.85	49.63	49.94	49.38	51.30
	LINE-2	28.64	33.29	36.06	37.73	38.20	38.88	39.83	38.78	41.78
	SDNE	32.38	36.23	38.82	39.88	40.23	40.84	41.17	39.26	40.36
	TADW	55.41	57.95	58.75	59.14	59.60	60.16	60.04	59.95	60.76
	SNE	30.30	35.40	39.69	42.11	42.30	44.06	43.51	44.39	44.81
	MVC-DNE	47.36	51.18	55.26	57.76	58.52	60.10	60.20	60.79	60.97
	attri2vec-linear	49.15	51.09	58.28	61.11	62.09	62.35	62.59	62.78	64.92
	attri2vec-ReLU	49.92	52.48	57.79	61.53	62.40	63.04	63.08	63.55	65.58
	attri2vec-kernel	**58.12**	**61.81**	**63.36**	64.96	**66.00**	65.50	65.80	66.27	**68.39**
	attri2vec-sigmoid	55.64	57.11	61.41	**65.05**	65.69	**66.24**	**67.04**	**66.68**	68.24

Table 5.3 Node Classification Results on DBLP

	Training Ratio	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F_1(%)	DeepWalk	78.58	79.92	80.08	80.30	80.42	80.34	80.46	80.44	80.30
	LINE-1	75.39	76.63	77.07	77.24	77.43	77.39	77.49	77.67	77.57
	LINE-2	68.08	69.76	70.23	70.47	70.81	70.53	70.89	70.65	70.81
	SDNE	63.59	64.96	65.23	65.46	65.60	65.51	65.43	65.98	64.78
	TADW	75.18	76.03	76.79	77.23	77.22	77.31	77.37	77.72	77.77
	SNE	68.37	70.19	70.73	70.81	71.29	71.08	71.20	71.10	71.01
	MVC-DNE	73.37	75.17	75.74	75.94	76.18	76.17	76.35	76.12	76.23
	attri2vec-linear	76.42	78.36	78.68	78.97	79.21	79.05	79.32	79.11	79.23
	attri2vec-ReLU	**80.86**	**82.91**	**83.31**	**83.55**	**83.80**	**83.57**	**84.00**	**83.95**	**83.95**
	attri2vec-lernel	79.56	80.58	81.11	81.45	81.54	81.30	81.63	81.57	81.74
	attri2vec-sigmoid	79.89	81.66	82.16	82.32	82.52	82.47	82.72	82.47	82.78
Macro-F_1(%)	DeepWalk	71.12	72.88	72.96	73.07	73.26	73.20	73.18	73.19	73.63
	LINE-1	66.91	68.41	68.95	69.18	69.36	69.38	69.41	69.59	70.03
	LINE-2	59.95	61.63	62.01	62.39	62.63	62.27	62.60	62.24	62.91
	SDNE	49.27	49.90	49.52	49.74	50.07	49.39	49.19	49.99	48.91
	TADW	65.17	67.14	68.03	68.47	68.61	68.65	68.50	68.91	69.61
	SNE	57.78	59.80	60.17	59.87	60.73	60.37	60.40	60.09	60.93
	MVC-DNE	65.04	67.26	67.76	67.68	67.96	68.00	68.23	67.97	67.98
	attri2vec-linear	69.19	71.79	71.99	72.16	72.59	72.51	72.66	72.15	73.09
	attri2vec-ReLU	**73.99**	**77.03**	**77.27**	**77.62**	**77.85**	**77.71**	**77.94**	**78.07**	**78.61**
	attri2vec-kernel	73.04	74.51	75.13	75.51	75.62	75.38	75.60	75.65	76.28
	attri2vec-sigmoid	73.33	75.97	76.51	76.64	76.88	77.05	77.10	76.89	77.67

Table 5.4 Node Classification Results on Facebook

	Training Ratio	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F_1(%)	DeepWalk	46.75	49.21	51.27	51.58	52.21	52.06	52.35	52.84	50.82
	LINE-1	42.43	46.62	49.13	50.38	51.28	51.41	52.11	51.98	51.32
	LINE-2	37.04	43.27	46.94	48.62	49.45	49.29	49.90	49.79	50.82
	SDNE	38.87	45.57	48.72	50.59	51.38	51.36	51.51	52.26	51.14
	TADW	53.95	56.66	57.56	58.04	58.66	58.06	58.09	59.26	57.84
	SNE	51.32	60.15	64.01	65.26	65.95	66.93	66.16	66.85	66.13
	MVC-DNE	58.12	64.80	68.17	69.87	70.51	70.37	71.04	71.34	70.92
	attri2vec-linear	59.36	64.12	66.79	68.23	69.00	69.42	69.79	69.80	69.21
	attri2vec-ReLU	59.69	64.25	67.34	68.70	69.42	69.43	70.07	70.15	69.98
	attri2vec-kernel	**63.04**	65.86	67.57	68.48	68.59	69.24	69.32	69.68	69.45
	attri2vec-sigmoid	**63.06**	**67.35**	**69.86**	**71.38**	**71.30**	**72.19**	**72.19**	**72.49**	**72.11**
Macro-F_1(%)	DeepWalk	29.85	31.04	30.55	30.83	30.76	30.31	30.15	29.76	29.29
	LINE-1	29.84	31.09	31.56	31.54	31.56	31.09	31.41	30.63	30.26
	LINE-2	26.80	28.18	27.46	27.60	27.22	26.22	26.26	26.02	27.12
	SDNE	28.36	30.24	29.69	30.12	29.97	29.25	29.22	29.44	28.62
	TADW	32.17	35.80	36.72	37.29	37.94	37.23	36.85	38.29	36.97
	SNE	40.15	43.51	43.74	43.61	43.76	43.72	42.35	42.51	41.89
	MVC-DNE	45.15	47.26	48.06	47.67	47.73	47.26	47.34	47.50	47.16
	attri2vec-linear	45.31	47.07	47.72	47.04	47.25	46.66	46.72	46.50	45.50
	attri2vec-ReLU	45.47	47.00	48.27	48.26	48.13	47.88	48.17	48.08	47.39
	attri2vec-kernel	44.91	46.82	46.73	46.61	46.31	46.39	45.61	45.92	45.51
	attri2vec-sigmoid	**48.60**	**50.91**	**50.99**	**51.43**	**50.81**	**51.43**	**50.29**	**50.66**	**50.73**

Table 5.5 Node Classification Results on Flickr

	Training Ratio	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F_1(%)	DeepWalk	40.25	45.60	47.67	49.05	49.74	50.30	50.88	51.76	51.72
	LINE-1	42.97	51.43	55.07	57.23	58.30	58.88	59.69	60.67	59.91
	LINE-2	39.01	48.51	52.31	54.01	55.14	56.40	56.11	56.44	56.53
	SDNE	38.80	47.63	50.50	51.92	52.93	53.48	54.21	54.82	54.32
	TADW	62.17	65.17	66.32	67.11	67.17	67.97	68.12	68.16	67.79
	SNE	40.70	49.73	53.42	55.53	56.12	57.66	57.38	58.46	58.34
	MVC-DNE	56.33	64.98	68.54	70.22	70.77	72.18	71.90	72.69	72.56
	attri2vec-linear	62.04	68.95	71.99	73.28	74.31	75.04	75.04	75.77	75.85
	attri2vec-ReLU	61.27	69.12	72.14	73.95	74.71	75.38	75.66	75.99	76.39
	attri2vec-kernel	75.20	77.05	77.33	78.04	78.31	78.77	78.78	79.65	78.82
	attri2vec-sigmoid	**75.40**	**78.25**	**79.38**	**79.98**	**80.41**	**80.96**	**80.74**	**81.66**	**80.18**
Macro-F_1(%)	DeepWalk	39.67	44.65	46.48	47.80	48.43	48.87	49.48	50.13	49.99
	LINE-1	42.89	51.00	54.41	56.50	57.63	57.99	58.87	59.65	58.92
	LINE-2	38.99	47.94	51.55	53.15	54.19	55.22	55.03	55.13	55.01
	SDNE	38.62	47.12	49.79	51.14	52.09	52.55	53.29	53.73	53.17
	TADW	61.54	64.52	65.90	66.71	66.69	67.36	67.60	67.59	67.20
	SNE	40.55	49.11	52.57	54.53	55.12	56.43	56.23	57.19	57.10
	MVC-DNE	56.20	64.78	68.32	69.97	70.57	71.88	71.65	72.38	72.19
	attri2vec-linear	61.93	68.77	71.77	73.00	74.10	74.70	74.74	75.43	75.38
	attri2vec-ReLU	61.27	68.97	71.94	73.69	74.50	75.09	75.40	75.65	75.92
	attri2vec-kernel	74.87	76.81	77.12	77.77	78.11	78.53	78.57	79.36	78.48
	attri2vec-sigmoid	**75.14**	**78.02**	**79.15**	**79.68**	**80.21**	**80.69**	**80.48**	**81.33**	**79.81**

Micro-F_1 and Macro-F_1, the best and second performers are highlighted by **bold** and underline, respectively.

From Tables 5.2-5.5, we can see that attri2vec achieves the best classification performance on all networks with all training ratios, with attri2vec-sigmoid performing best on Citeseer, Facebook and Flick and attri2vec-ReLU performing best on DBLP. This suggests that, by discovering a latent node attribute subspace, attri2vec provides the best way to complement network structure with node attributes towards learning node representations. In most cases, under the attri2vec framework, the non-linear mappings (approximated kernel and sigmoid mapping) outperform the linear mappings (linear and ReLU mapping), which proves that network structure and node content usually exhibit non-linear correlations, and node attribute transformation with a non-linear mapping provides a better way to fuse network structure into a node attribute subspace.

We can also observe that, in most cases, attributed network embedding algorithms (TADW, SNE, MVC-DNE and attri2vec) significantly outperform the only structure preserving network embedding algorithms (DeepWalk, LINE-1, LINE-2 and SDNE). This proves that node content features are able to provide an essential complement to network structure to learn better node representations. Among the structure preserving network embedding algorithms, DeepWalk achieves the best classification performance. Compared with LINE-1, LINE-2 and SDNE that only capture the local first-order and second-order proximity, DeepWalk takes advantage of random walks to preserve the higher-order proximity, which contributes to DeepWalk's superior performance over LINE-1, LINE-2 and SDNE.

5.4.5 Node Clustering Experiments

To make further comparisons between the proposed attri2vec algorithm and baseline methods, on DBLP and Facebook, we also conduct node clustering experiments. We apply K-means algorithm to the learned node representations, and partition network nodes into 4 groups for DBLP and Facebook and use node labels as clustering ground truth. To reduce the variance caused by random initialization, we repeat K-means clustering for 20 times and report the averaged Accuracy, Fvalue and NMI (normalized

Table 5.6 Node Clustering Results on DBLP

	Accuracy(%)	Fvalue(%)	NMI(%)
DeepWalk	70.71	64.50	36.92
LINE-1	58.02	54.57	18.88
LINE-2	61.23	47.44	18.33
SDNE	52.33	42.61	11.73
TADW	59.63	58.16	17.81
SNE	61.37	49.96	16.19
MVC-DNE	66.25	60.42	28.30
attri2vec-linear	71.05	63.39	34.31
attri2vec-ReLU	**76.69**	**69.67**	**44.34**
attri2vec-kernel	73.18	67.96	39.87
attri2vec-sigmoid	72.07	66.08	38.26

Table 5.7 Node Clustering Results on Facebook

	Accuracy(%)	Fvalue(%)	NMI(%)
DeepWalk	52.65	39.10	2.03
LINE-1	51.91	46.27	1.34
LINE-2	51.84	38.97	1.21
SDNE	51.88	42.75	2.11
TADW	51.84	46.23	7.76
SNE	54.12	39.28	9.10
MVC-DNE	52.39	38.05	5.56
attri2vec-linear	53.89	41.62	6.52
attri2vec-ReLU	53.66	41.74	5.70
attri2vec-kernel	**66.77**	**50.32**	**17.15**
attri2vec-sigmoid	53.26	41.71	5.55

mutual information [73]). Tables 5.6-5.7 give the clustering results on DBLP and Facebook. As is shown in Tables 5.6-5.7, the proposed attri2vec algorithm achieves the best clustering performance, which further proves its advantage in learning informative node representations for attributed networks over the state-of-the-art baselines.

Table 5.8 Performance Comparison between GD and SGD on Facebook

Method	Micro-F_1	Macro-F_1	Training Time (s)
UPP-SNE (GD)	0.6192	0.3535	24515.89
attri2vec-kernel (SGD)	0.6910	0.4607	10944.35
Gain	11.60% ↑	30.33% ↑	55.36% ↓

5.4.6 Comparison of Gradient Descent *vs.* Stochastic Gradient Descent

In this section, we choose the Facebook network as a case study to compare the efficacy and efficiency of two difference optimization strategies: stochastic gradient descent (SGD) used by attri2vec and Gradient Descent (GD) used by UPP-SNE [102], for solving the optimization problem Eq. (5.5). We compare UPP-SNE with attri2vec-kernel, as they both use the approximated kernel mapping to construct node representations. To make a fair comparison, for UPP-SNE, we set the number of iterations for gradient descent to 40, and for attri2vec, we set the number of iteration as $nnz(n(v_i, v_j)) \times 40$, where $nnz(n(v_i, v_j))$ denotes the number of non-zero values of $n(v_i, v_j)$.

Table 5.8 compares both node classification performance and training time of UPP-SNE and attri2vec-kernel. Here, we report the classification results with a training ratio of 50%. As can be seen, attri2vec-kernel with SGD offers performance gains with an increase of 11.60% in Micro-F1, and 30.33% in Macro-F1, respectively, as SGD alleviates the local minima problem. On the other hand, SGD significantly reduces the training time of GD, with a decrease of 55.36%. This demonstrates the advantage of attri2vec over UPP-SNE in terms of both effectiveness and efficiency.

5.4.7 Comparison of Shallow *vs.* Deep Mapping

In this section, we select Citeseer and DBLP to study the performance change of attri2vec when using a deep neural network rather than a shallow one-layer non-linear mapping to construct node representations from node features. Here, we construct node representations through a neural network with 2 hidden layers, with the number of neurons at each layer set to 3,703-256-128 and 2,476-256-128 on Citeseer and DBLP

Table 5.9 Performance Comparison of Shallow and Deep Mapping on Citeseer

Method	Micro-F_1	Macro-F_1	Training Time (s)
attri2vec-sigmoid	0.7027	0.6569	29405.51
attri2vec-deep	0.6630	0.6076	380331.50
Gain	5.65% ↓	7.50% ↓	1193.40% ↑

Table 5.10 Performance Comparison of Shallow and Deep Mapping on DBLP

Method	Micro-F_1(%)	Macro-F_1(%)	Training Time (s)
attri2vec-sigmoid	82.52	76.88	5310.19
attri2vec-deep	83.72	78.29	319881.87
Gain	1.45% ↑	1.83% ↑	5923.96% ↑

respectively, and denote this method as attri2vec-deep. We compare attri2vec-deep with attri2vec-sigmoid. For both methods, we set the number of iterations as 100 million.

Tables 5.9-5.10 compare the classification performance and the training time of attri2vec-linear and attri2vec-deep on Citeseer and DBLP. Again, results with a training ratio of 50% are reported. As can be seen, attri2vec-sigmoid is significantly more efficient than attri2vec-deep for training. From Table 5.10, attri2vec-deep can be seen to perform slightly better than attri2vec-sigmoid on DBLP, which demonstrates the advantage of deep neural network in characterizing the complex relations between network structure and node content. However, on Citeseer, attri2vec-deep achieves even worse classification performance than attri2vec-sigmoid. This might be attributed to two reasons: (1) attri2vec-deep has a lot more parameters, which requires more iterations to properly fitting the parameters to obtain the best models; (2) compared with attri2vec-sigmoid, attri2vec-deep is prone to the local minima, which requires more advanced SGD techniques to obtain better solutions. In summary, compared with the deep-neural network, the one-layer non-linear mapping is good enough to learn reasonably high-quality node representations with much less computational cost.

Table 5.11 A Summary of Time Stamped DBLP Subgraphs

Subgraph	# of nodes	# of edges	# of out-of-sample nodes
DBLP2006	12,820	28,809	5,628
DBLP2007	14,286	32,336	4,162
DBLP2008	15,745	36,907	2,703
DBLP2009	16,960	40,460	1,488

5.4.8 Experiments on Out-of-sample Extension

In this section, we study the performance of attri2vec in solving the out-of-sample problem, which is achieved by constructing representations for new coming nodes from their content attributes through the learned mapping function. We compare attri2vec with two baselines (1) MVC-DNE [97], which is also able to infer representations for new coming nodes using node attributes, and (2) node content features that construct node representations without mapping functions. For the three methods, we consistently set the dimension of out-of-sample node representations to 128. Singular Value Decomposition (SVD) is used to do dimension reduction on node content features.

We select the DBLP network to conduct the experiments, where each node has a time stamp. From the DBLP network, we construct four subgraphs using papers published before 2006, 2007, 2008 and 2009, and denote the four subgraphs as DBLP2006, DBLP2007, DBLP2008, and DBLP2009. For each subgraph, the remaining papers are taken as out-of-sample nodes. The statistics of the four DBLP subgraphs and their corresponding out-of-sample nodes are given in Table 5.11. To evaluate the quality of the learned out-of-sample node representations, we conduct node classification and link prediction experiments.

For node classification, we first train an SVM classifier (with the LIBLINEAR implementation [15]) on the randomly selected 50% learned in-sample node representations, and then apply the learned SVM classifier to the out-of-sample node representations. To reduce the variance caused by random training sample selection, we repeat the training and test process for 10 times and report the averaged Micro-F_1 and Macro-F_1. Table 5.12 gives the node classification results for out-of-sample nodes. We can see that

Table 5.12 Node Classification Results for Out-of-sample nodes on DBLP

	method	DBLP2006	DBLP2007	DBLP2008	DBLP2009
Micro-F_1(%)	feature	65.38	64.01	61.75	62.12
	MVC-DNE	64.96	62.74	61.45	61.68
	attri2vec-sigmoid	**67.66**	**66.79**	**64.87**	**67.05**
Macro-F_1(%)	feature	60.49	59.77	58.22	57.14
	MVC-DNE	59.53	57.34	56.96	55.12
	attri2vec-sigmoid	**63.94**	**63.69**	**62.35**	**62.50**

Table 5.13 Operators to Construct Edge Features

Operator	Symbol	Definition
Average	$\boxplus$	$[\Phi(v_i) \boxplus \Phi(v_j)]_k = \frac{\Phi_k(v_i)+\Phi_k(v_j)}{2}$
Hadamard	$\boxdot$	$[\Phi(v_i) \boxdot \Phi(v_j)]_k = \Phi_k(v_i) \cdot \Phi_k(v_j)$
Weighted-L1	$\|\cdot\|_{\bar{1}}$	$\|\Phi(v_i) \cdot \Phi(v_j)\|_{\bar{1}k} = \|\Phi_k(v_i) - \Phi_k(v_j)\|$
Weighted-L2	$\|\cdot\|_{\bar{2}}$	$\|\Phi(v_i) \cdot \Phi(v_j)\|_{\bar{2}k} = (\Phi_k(v_i) - \Phi_k(v_j))^2$

attri2vec yields the best classification performance. This demonstrates the effectiveness of attri2vec in solving the out-of-sample problem.

To carry out link prediction experiments, following [19], we construct edge features from node representations with the operators given in Table 5.13. To generate the training set, on the in-sample subgraphs, for each connected node pair (v_i, v_j), we randomly sample a negative node pair (v_i, v_k) with no edges observed between them in the current time stamp. Similarly, to construct test set, for each edge connecting to the out-of-sample nodes, we randomly sample a negative node pair, with no ground-truth edges between them. On the generated edge features, we adopt SVM to perform training and testing. Table 5.14 gives the AUC values of different methods for predicting the links of out-sample-nodes. As shown in the table, attri2vec-sigmoid with the Weighted-L2 operator performs best in predicting links for new coming out-of-sample nodes. This proves the potential of attri2vec in accurately recommending links for new coming out-of-sample nodes through the inferred node representations.

Table 5.14 AUC Values (%) for Predicting the Links of Out-of-sample Nodes on DBLP

method	operator	DBLP2006	DBLP2007	DBLP2008	DBLP2009
feature	Average	53.31	53.52	54.39	54.44
	Hadamard	76.07	76.43	76.17	76.38
	Weighted-L1	64.84	65.84	65.60	65.42
	Weighted-L2	65.20	66.17	65.92	65.74
MVC-DNE	Average	51.78	52.74	54.18	55.05
	Hadamard	51.07	51.12	51.58	51.92
	Weighted-L1	49.70	50.94	50.69	49.19
	Weighted-L2	49.95	51.11	50.80	49.30
attri2vec-sigmoid	Average	53.41	54.22	56.00	57.01
	Hadamard	80.82	81.98	82.56	83.88
	Weighted-L1	87.99	89.16	90.43	91.38
	Weighted-L2	**88.11**	**89.33**	**90.66**	**91.70**

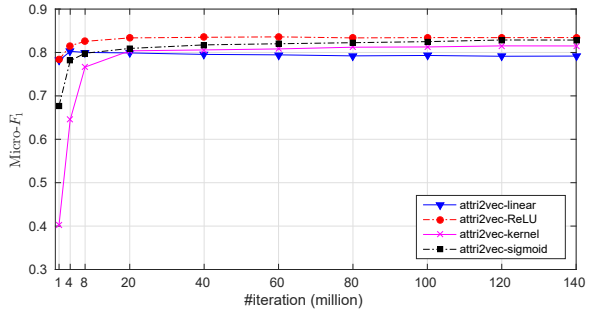
5.4.9 Parameter Sensitivity Study

In this section, we report the sensitivity study of the proposed attri2vec algorithm by analyzing its detailed performance on the DBLP network. In the experiments, we consider three important parameters: the maximum number of iterations, random walk window size t, and embedding dimension d. In order to draw conclusive observations, we fix any two of the three parameters and investigate the performance change of attri2vec when the remaining one varies, respectively. Micro-F_1 of node classification with 50% training ratio is used to evaluate the performance.

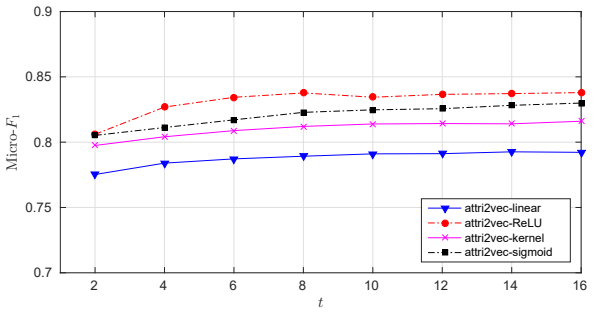
Fig. 5.5 shows the parameter sensitivity experimental results. We can find that as the parameters increase, classification accuracy of attri2vec gradually increases and stabilizes after reaching a threshold.

5.5 Conclusion

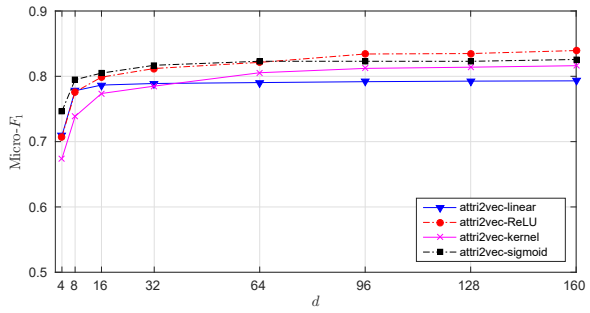
In this chapter, we proposed a unified framework for attributed network embedding, attri2vec, that learns network node representations by discovering a latent node attribute subspace via a network structure guided transformation on the original



(a) *#iterations*



(b) t



(c) d

Fig. 5.5 Parameter sensitivity study of the algorithm performance (Micro-F_1) in terms of (a): the maximum number of iterations, (b): the window size of the random walks t, and (c): embedding dimension d.

attribute space. We argued that, because network structure and node attributes have different feature spaces, the data distributions of two distinct features spaces might be inconsistent, making the existing attributed network embedding algorithms fail to achieve satisfactory performance. This motivated us to find a latent attribute subspace, which can respect network structure in a more consistent manner to obtain high-quality node representations. By performing a series of linear and non-linear mappings on

node attributes, attri2vec embeds network nodes into a structure preserving attribute subspace. To preserve the network structure, DeepWalk [63] mechanism is employed, which makes nodes sharing similar neighbors embedded closely in the attribute subspace. In this way, network structure and node content features are integrated together seamlessly to obtain informative node representations. To learn node embeddings efficiently, we developed an online stochastic gradient descent algorithm to solve the formulated optimization problem, which improves both learning effectiveness and efficiency. As an additional advantage, attri2vec also provides a potential solution to solve the out-of-sample problem, where representations of new coming nodes can be learned using their available node attributes via the learned mapping function. Experiments of node classification and node clustering on four real-world networks verify the effectiveness and efficiency of attri2vec. We also study the ability of attri2vec to solve the out-of-sample problem, which yields promising results.

Chapter 6

Scalable Incomplete Network Embedding

6.1 Introduction

Network embedding, also known as network representation learning, aims to embed each node of a network into a low-dimensional vector space, by preserving network structure and other side information. As a result, network analytical tasks can be easily conducted by applying machine learning techniques to the new vector space. Due to the increasing popularity of networked applications, such as social networks, real-world networks are often of large scale, containing a large number of nodes, links, and high-dimensional content features. These challenges have motivated the development of many network embedding solutions in the field.

Two main streams of network embedding algorithms include (1) *structure preserving network embedding* methods, e.g., DeepWalk [63], LINE [76], node2vec [19], that preserve only network structure, and (2) *attributed network embedding* methods, e.g., TADW [96], HSCA [101], MVC-DNE [97], that augment network structure with node attributes. Because structure preserving network embedding methods only utilize network topology structure, they have shown to be inferior to attributed network embedding methods which combine both node content and structure information for embedding learning. Meanwhile, while existing attributed network embedding methods

can leverage node content, they often assume the input networks are complete and cannot handle missing data. In addition, they suffer from poor scalability due to high computational cost. In summary, two major drawbacks of the existing attributed network embedding methods include:

- **Vulnerable to missing data:** Real-world networks are often incomplete with missing edges and/or missing node content features [34], due to various reasons. First, privacy or legal restrictions make sensitive information on node attributes or part of connections among nodes inaccessible. Second, networks have too large size, making it prohibitively expensive or even impossible to directly acquire complete networks. Instead, a common practice is to obtain a smaller sample of large networks for analysis. The sampled network inevitably contains lots of missing nodes and links. Third, networks are dynamic in nature, and thus newly joined nodes often have very few links or content features. All these aspects result in noisy and incomplete networks. Although research has shown that jointly exploiting network structure and node attributes can enhance the embedding performance, existing attributed network embedding algorithms require node attributes to be all complete. When nodes in networks have no attributes or important dimensions of attributes are unobservable, existing methods are vulnerable to such missing data.

- **Poor scalability:** Most attributed network embedding algorithms rely on matrix factorization (e.g., TADW [96], HSCA [101]) or deep neural networks (e.g., MVC-DNE [97]) to fuse the information on network structure and node attributes towards learning a better joint representation. The matrix factorization or deep neural networks require high computational cost, where the time complexity is at least quadratic to the number of nodes, preventing them from scaling to large-scale networks with a large number of nodes and high-dimensional node features. Thus, there is a strong demand for developing efficient and effective network embedding algorithms.

The above observations motivate our research to find a better network representation that is not only robust to missing data in networks, but also scalable to large-scale networks.

In this chapter, we propose a new attributed network embedding algorithm, called SINE (Scalable Incomplete Network Embedding), that provides a probabilistic formulation that 1) models pairs of node and context relationships to capture broader structural dependency in random walks; 2) models pairs of node and attribute relationships to make the best of available node content information. Different from existing attributed network embedding algorithms, SINE provides a flexible way to leverage useful information and diminish the negative impacts on the learned embedding representation, caused by the existence of missing data on network structure and/or node attributes. We derive an online optimization strategy based on stochastic gradient descent, which enables SINE with high learning efficiency and the ability to scale up to large networks. We evaluate the efficacy and efficiency of SINE on real-world networks in various tasks, including node classification, clustering, and link prediction. As compared with the state-of-the-art baselines, SINE is demonstrated to be robust to missing links and node attributes, but also scalable to large-scale networks.

The main contribution of this chapter is threefold:

- We advance the existing attributed network embedding learning to a realistic missing data setting, allowing embedding leaning to be highly efficient and accurate for real-world networks.
- We propose a scalable and efficient algorithm to combine network structure and node attributes to learn a joint embedding representation, thereby diminishing negative impacts of missing information.
- We evaluate the effectiveness of the proposed method under different missing data settings, showing its superior performance to the state-of-the-art baselines.

6.2 Problem Definition and Preliminaries

6.2.1 Problem Definition

Assume an incomplete network $G = (\mathcal{V}, \mathcal{E}, \mathcal{A}, X, \Omega)$ is given, where $\mathcal{V}$ is the set of nodes, $\mathcal{E}$ is the set of observed edges, which may be very sparse, and $\mathcal{A}$ is the set

of node attributes. $X \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{A}|}$ is node feature matrix, with each observed element $X_{ij} \geqslant 0$ indicates the occurrence times/weights of attribute $a_j \in \mathcal{A}$ at node $v_i \in \mathcal{V}$, and $\Omega \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{A}|}$ indicates whether the node attribute value X_{ij} is observed, with $\Omega_{ij} = 1$ for observed X_{ij} and $\Omega_{ij} = 0$ for unobserved X_{ij}. For networks with attributes taking continuous values, discretization can be applied to convert the continuous attributes to discrete ones.

The objective of incomplete network embedding is to leverage incomplete information in $\mathcal{E}$ and X to learn a mapping function $\Phi : v_i \in \mathcal{V} \mapsto \mathbb{R}^{|\mathcal{V}|\times d}$. The learned node representations $\Phi(v_i)$ are expected to be (1) low-dimensional with $d \ll |\mathcal{V}|$, and (2) informative for the downstream tasks, such as node classification, node clustering and link prediction, *etc.*

6.2.2 Preliminaries: DeepWalk

The Skip-Gram model [55] learns word representations by capturing the semantic similarity between words sharing similar contexts. DeepWalk generalizes the idea of Skip-Gram from word representation learning to network embedding, by using random walks to collect node contexts. Given a truncated random walk with length L, $\{v_{r_1}, v_{r_2}, \cdots, v_{r_i}, \cdots v_{r_L}\}$, DeepWalk learns representation $\Phi(v_{r_i})$ for node v_{r_i} by using it to predict its context nodes, which is achieved by solving the following optimization problem,

$$\min_{\Phi} -\log \mathrm{P}(\{v_{r_{i-t}}, \cdots, v_{r_{i+t}}\} \setminus v_{r_i} | v_{r_i}), \tag{6.1}$$

where $\{v_{r_{i-t}}, \cdots, v_{r_{i+t}}\} \setminus v_{r_i}$ are the context nodes of v_{r_i} within t window size.

By making conditional independence assumption, the probability $\mathrm{P}(\{v_{r_{i-t}}, \cdots, v_{r_{i+t}}\} \setminus v_{r_i} | v_{r_i})$ can be expanded as

$$\mathrm{P}(\{v_{r_{i-t}}, \cdots, v_{r_{i+t}}\} \setminus v_{r_i} | v_{r_i}) = \prod_{j=i-t, j\neq i}^{i+t} \mathrm{P}(v_{r_j} | v_{r_i}). \tag{6.2}$$

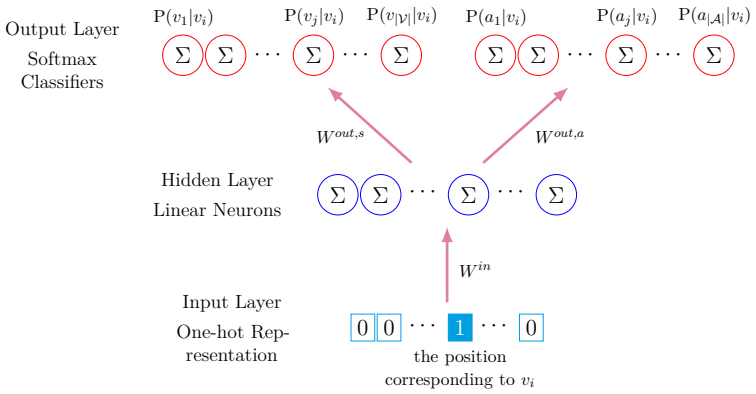


Fig. 6.1 The model architecture of SINE. For each node v_i, SINE learns its representation by using it to predict its context node v_j and its observable attribute a_j so that nodes sharing similar context nodes or similar observed attributes are embedded closely in the new vector space. In this way, the incomplete structure and node attribute information is utilized flexibly to learn informative node representations.

Following [102], considering all the generated random walks, the overall optimization problem of DeepWalk can be reformulated as

$$\min_{\Phi} -\sum_{i=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{V}|} n(v_i, v_j)\log \mathrm{P}(v_j|v_i), \tag{6.3}$$

where $n(v_i, v_j)$ is the total occurrence times of v_j as v_i's context nodes across all generated random walks within t window size. The overall optimization problem can be solved by stochastically sampling a node context pair (v_i, v_j) and minimizing the following partial objective:

$$\mathcal{O}_{ij}^{s} = -\log \mathrm{P}(v_j|v_i). \tag{6.4}$$

6.3 SINE: Scalable Incomplete Network Embedding

6.3.1 Model Architecture

Inspired by DeepWalk [63], SINE encodes network structure into node embeddings by allowing nodes sharing similar context nodes to have similar representations. This is achieved by minimizing the DeepWalk partial objective in Eq. (6.4) for each node context pair (v_i, v_j) collected from random walks.

In addition to topology structure, we also wish that nodes sharing similar attributes should be close in the new vector space. Following the mechanism of Skip-Gram [55], we make the learned node representations respect this property by using node v_i to predict its co-occurring attribute a_j for each observed node attribute co-occurrence pair (v_i, a_j). This is achieved by minimizing the following objective:

$$\mathcal{O}_{ij}^{a} = -\log \mathrm{P}(a_j|v_i). \tag{6.5}$$

As shown in Fig. 6.1, the learning framework of SINE is a three-layer neural network: the first layer is the one-hot representation for each node v_i, the hidden layer is the node representation $\Phi(v_i) \in \mathbb{R}^d$ constructed by a linear transformation from the input layer with weight matrix $W^{in} \in \mathbb{R}^{|\mathcal{V}|\times d}$, the output layer is the softmax conditional probability $\mathrm{P}(v_j|v_i)$ and $\mathrm{P}(a_j|v_i)$, for each node v_j and each attribute a_j, aggregated from the hidden layer with weight matrix $W^{out,s} \in \mathbb{R}^{d\times|\mathcal{V}|}$ and $W^{out,a} \in \mathbb{R}^{d\times|\mathcal{A}|}$, respectively.

Given the one-hot representation $\boldsymbol{p}^{(i)} \in \mathbb{R}^{|\mathcal{V}|}$ of node v_i with $\boldsymbol{p}_i^{(i)} = 1$ and $\boldsymbol{p}_j^{(i)} = 0$ for $j \neq i$, the node representation $\Phi(v_i)$ in the hidden layer is constructed as

$$\Phi(v_i) = {W^{in}}^{\mathrm{T}} \boldsymbol{p}^{(i)} = \boldsymbol{w}_i^{in}, \tag{6.6}$$

where $\boldsymbol{w}_i^{in}$ is the transpose of the i-th row of $W^{in} \in \mathbb{R}^{|\mathcal{V}|\times d}$ (the weight matrix from the input layer to the hidden layer).

In the output layer, for node context pair (v_i, v_j), the probability $\mathrm{P}(v_j|v_i)$ is modeled by the softmax signal:

$$\mathrm{P}(v_j|v_i) = \frac{\exp(\Phi(v_i) \cdot \boldsymbol{w}_j^{out,s})}{\sum_{k=1}^{|\mathcal{V}|} \exp(\Phi(v_i) \cdot \boldsymbol{w}_k^{out,s})}, \tag{6.7}$$

where $\boldsymbol{w}_j^{out,s}$ is the j-th column of $W^{out,s} \in \mathbb{R}^{d\times|\mathcal{V}|}$ (the weight matrix from the hidden layer to the output layer for predicting node context).

Similarly, for the node attribute co-occurrence pair (v_i, a_j), the probability $\mathrm{P}(a_j|v_i)$ is modeled by

$$\mathrm{P}(a_j|v_i) = \frac{\exp(\Phi(v_i) \cdot \boldsymbol{w}_j^{out,a})}{\sum_{k=1}^{|\mathcal{A}|} \exp(\Phi(v_i) \cdot \boldsymbol{w}_k^{out,a})}, \tag{6.8}$$

where $\boldsymbol{w}_j^{out,a}$ is the j-th column of $W^{out,a} \in \mathbb{R}^{d\times|\mathcal{A}|}$ (the weight matrix from the hidden layer to the output layer for predicting node attribute).

By aggravating the structure preserving objective in Eq. (6.4) and the node attribute preserving objective in Eq. (6.5), node representations $\Phi(\cdot)$ that well preserve the available structure and node content information, can be learned by solving the following overall optimization problem:

$$\min_{\Phi} \mathcal{O}, \tag{6.9}$$

where

$$\mathcal{O} = -\alpha_1 \sum_{i=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{V}|} n(v_i, v_j)\text{logP}(v_j|v_i) - \alpha_2 \sum_{i=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{A}|} \Omega_{ij}X_{ij}\text{logP}(a_j|v_i). \tag{6.10}$$

where α_1 and α_2 are the trade-off parameters that balance the structure preserving objective and the node content preserving objective. They are set as

$$\alpha_1 = \frac{1}{\sum_{i=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{V}|} n(v_i, v_j)},\ \alpha_2 = \frac{1}{\sum_{i=1}^{|\mathcal{V}|}\sum_{j=1}^{|\mathcal{A}|} \Omega_{ij}X_{ij}}.$$

In Eq. (6.10), we only concern about the non-zero values of $n(v_i, v_j)$ and $\Omega_{ij}X_{ij}$, i.e., the node context pairs collected from observed links and the observed node attribute co-occurrence pairs, whose numbers are much smaller than $|\mathcal{V}| \times |\mathcal{V}|$ and $|\mathcal{V}| \times |\mathcal{A}|$, respectively. In this way, the available network structure and node content information can be fully utilized, and the negative impacts of missing information is diminished.

6.3.2 Model Optimization

We solve the overall optimization problem in Eq. (6.9) by minimizing the partial objective in Eq. (6.4) and Eq. (6.5) alternately with stochastic gradient descent by sampling a node context pair (v_i, v_j) or a node attribute co-occurrence pair (v_i, a_j) at each iteration, according to the distribution of $n(v_i, v_j)$, and $\Omega_{ij}X_{ij}$, respectively.

For the sampled node context pair (v_i, v_j), negative sampling is used to speed up training. Thus, the partial objective $\mathcal{O}_{ij}^s$ in Eq. (6.4) can be reformulates as

$$\mathcal{O}_{ij}^s = -\log \sigma(\Phi(v_i) \cdot \boldsymbol{w}_j^{out,s}) - \sum_{k:v_k \in \mathcal{V}_{neg}} \log \sigma(-\Phi(v_i) \cdot \boldsymbol{w}_k^{out,s}), \tag{6.11}$$

where $\mathcal{V}_{neg}$ is the set of sampled negative nodes. The parameters are updated by gradient descent:

$$\begin{cases} \boldsymbol{w}_i^{in} = \boldsymbol{w}_i^{in} - \eta \dfrac{\partial \mathcal{O}_{ij}^s}{\partial \boldsymbol{w}_i^{in}}, \\ \boldsymbol{w}_j^{out,s} = \boldsymbol{w}_j^{out,s} - \eta \dfrac{\partial \mathcal{O}_{ij}^s}{\partial \boldsymbol{w}_j^{out,s}}, \\ \boldsymbol{w}_k^{out,s} = \boldsymbol{w}_k^{out,s} - \eta \dfrac{\partial \mathcal{O}_{ij}^s}{\partial \boldsymbol{w}_k^{out,s}}, \text{ for } v_k \in \mathcal{V}_{neg}, \end{cases} \tag{6.12}$$

where η is the learning rate. The gradients are calculated as follows

$$\begin{cases} \dfrac{\partial \mathcal{O}_{ij}^s}{\partial \boldsymbol{w}_i^{in}} = (\sigma(\Phi(v_i) \cdot \boldsymbol{w}_j^{out,s}) - 1)\boldsymbol{w}_j^{out,s} \\ \qquad\qquad + \displaystyle\sum_{k:v_k \in \mathcal{V}_{neg}} \sigma(\Phi(v_i) \cdot \boldsymbol{w}_k^{out,s})\boldsymbol{w}_k^{out,s}, \\ \dfrac{\partial \mathcal{O}_{ij}^s}{\boldsymbol{w}_j^{out,s}} = (\sigma(\Phi(v_i) \cdot \boldsymbol{w}_j^{out,s}) - 1)\Phi(v_i), \\ \dfrac{\partial \mathcal{O}_{ij}^s}{\boldsymbol{w}_k^{out,s}} = \sigma(\Phi(v_i) \cdot \boldsymbol{w}_k^{out,s})\Phi(v_i), \text{ for } v_k \in \mathcal{V}_{neg}. \end{cases} \tag{6.13}$$

Similarly, for each sampled node attribute co-occurrence pair (v_i, a_j), by adopting negative sampling, the partial objective $\mathcal{O}_{ij}^a$ in Eq. (6.5) is approximated by

$$\mathcal{O}_{ij}^a = -\log \sigma(\Phi(v_i) \cdot \boldsymbol{w}_j^{out,a}) - \sum_{k:v_k \in \mathcal{A}_{neg}} \log \sigma(-\Phi(v_i) \cdot \boldsymbol{w}_k^{out,a}), \tag{6.14}$$

Algorithm 2 SINE: Scalable Incomplete Network Embedding

Input:
An incomplete network $G = (\mathcal{V}, \mathcal{E}, \mathcal{A}, X, \Omega)$;
Output:
Node representation $\Phi(\cdot)$ for each $v_i \in \mathcal{V}$;
$\mathbb{S} \leftarrow$ generate a set of random walks on G;
$n(v_i, v_j) \leftarrow$ count frequency of node context pairs (v_i, v_j) in $\mathbb{S}$;
repeat
  draw a random number $r \in (0, 1)$;
  if $r \leqslant 0.5$ **then**
    $(v_i, v_j) \leftarrow$ sample a node context pair according to the distribution $n(v_i, v_j)$;
    $\mathcal{V}_{neg} \leftarrow$ draw K negative nodes;
    $(W^{in}, W^{out,s}) \leftarrow$ update parameters with $(v_i, v_j, \mathcal{V}_{neg})$ and Eq. (6.12);
  else
    $(v_i, a_j) \leftarrow$ sample a node attribute pair according to the distribution $\Omega_{ij} X_{ij}$;
    $\mathcal{A}_{neg} \leftarrow$ draw K negative attributes;
    $(W^{in}, W^{out,a}) \leftarrow$ update parameters with $(v_i, a_j, \mathcal{A}_{neg})$ and Eq. (6.15);
  end if
until maximum number of iterations expire;
construct node representation $\Phi(\cdot)$ with W^{in} and Eq. (6.6);
return $\Phi(\cdot)$;

where $\mathcal{A}_{neg}$ is the set of sampled negative attributes. The parameters are updated as

$$\begin{cases} \boldsymbol{w}_i^{in} = \boldsymbol{w}_i^{in} - \eta \dfrac{\partial \mathcal{O}_{ij}^a}{\partial \boldsymbol{w}_i^{in}}, \\ \boldsymbol{w}_j^{out,a} = \boldsymbol{w}_j^{out,a} - \eta \dfrac{\partial \mathcal{O}_{ij}^a}{\partial \boldsymbol{w}_j^{out,a}}, \\ \boldsymbol{w}_k^{out,a} = \boldsymbol{w}_k^{out,a} - \eta \dfrac{\partial \mathcal{O}_{ij}^a}{\partial \boldsymbol{w}_k^{out,a}}, \text{ for } a_k \in \mathcal{A}_{neg}. \end{cases} \tag{6.15}$$

The gradients are calculated as follows

$$\begin{cases} \dfrac{\partial \mathcal{O}_{ij}^a}{\partial \boldsymbol{w}_i^{in}} = (\sigma(\Phi(v_i) \cdot \boldsymbol{w}_j^{out,a}) - 1)\boldsymbol{w}_j^{out,a} \\ \qquad\qquad + \displaystyle\sum_{k:a_k \in \mathcal{A}_{neg}} \sigma(\Phi(v_i) \cdot \boldsymbol{w}_k^{out,a})\boldsymbol{w}_k^{out,a}, \\ \dfrac{\partial \mathcal{O}_{ij}^a}{\boldsymbol{w}_j^{out,a}} = (\sigma(\Phi(v_i) \cdot \boldsymbol{w}_j^{out,a}) - 1)\Phi(v_i), \\ \dfrac{\partial \mathcal{O}_{ij}^a}{\boldsymbol{w}_k^{out,a}} = \sigma(\Phi(v_i) \cdot \boldsymbol{w}_k^{out,a})\Phi(v_i), \text{ for } a_k \in \mathcal{A}_{neg}. \end{cases} \tag{6.16}$$

Algorithm 2 gives the major procedure of the proposed SINE algorithm. At step 1, SINE starts random walks with length L at each node for γ times, and at step 2, counts the frequency of node context pairs $n(v_i, v_j)$ within t window size. At step 3-14, SINE updates the parameters with stochastic gradient descent. At each iteration, SINE samples a random switch variable $r \in (0, 1)$ to determine whether the structure preserving objective or the node attribute preserving objective is to be minimized, with a threshold of 0.5. To sample node context pair and node attribute pair, the alias table method [40] is used, which takes only $O(1)$ time. After the iteration is finished, node representations $\Phi(\cdot)$ are constructed with W^{in} and Eq. (6.6).

The time complexity of SINE only relies on the maximum number of iterations and the dimension of learned node representations d. The maximum number of iterations is at the scale of $O(\max(nnz(\Omega_{ij}X_{ij}), |\mathcal{V}|))$, where $nnz(\Omega_{ij}X_{ij})$ is the number of non-zero values of $\Omega_{ij}X_{ij}$, i.e., the number of observed node attribute co-occurrence pairs, and $|\mathcal{V}|$ indicates the scale of node context pairs collected from random walks. Thus, SINE has an overall time complexity of $O(d \cdot \max(nnz(\Omega_{ij}X_{ij}), |\mathcal{V}|))$, ensuring its good scalability.

6.4 Experiments

In this section, we present experimental results on real-world networks to verify the effectiveness and efficiency of the proposed SINE algorithm in learning informative node representations for incomplete networks.

6.4.1 Benchmark Networks

Four real-world networks used in our experiments are detailed as follows:

Cora and **Citeseer**[1]: The Cora network contains 2,708 publications and 5,249 citations. The Citeseer network includes 3,312 publications and 4,732 citations. For Cora and Citeseer, each paper is represented by a 1,433-dimensional, and 3,703-dimensional

[1]https://linqs.soe.ucsc.edu/data

Table 6.1 Summary of Four Real-world Networks

	Cora	Citeseer	DBLP(Subgraph)	DBLP(Full)
$\lvert\mathcal{V}\rvert$	2,708	3,312	18,448	1,632,442
$\lvert\mathcal{E}\rvert$	5,278	4,732	45,611	2,327,450
$\lvert\mathcal{A}\rvert$	1,433	3,703	5,959	154,309
$nnz(X)$	49,216	105,165	108,016	10,413,178
# of Class	7	6	4	N/A

binary vector, with each dimension indicating the presence/absence of the corresponding word.

DBLP(Subgraph) and **DBLP(Full)**: The DBLP(Full) network is constructed by the papers and their citation relationships of the DBLP bibliographic network[2]. There are 1,632,442 papers and 2,327,450 citations in all. To construct the DBLP(Subgraph) network, we extract papers from the four research areas: *Database*, *Data Mining*, *Artificial Intelligence*, *Computer Vision*, according to papers' venue information and remove papers with no citations. The DBLP(Subgraph) network contains 18,448 papers and 45,661 citations. From paper titles, for DBLP(Subgraph) and DBLP(Full), we construct 5,959-dimensional and 154,309-dimensional binary node feature vectors, respectively, with each dimension indicating the presence/absence of the corresponding word.

For all networks, the link direction is ignored. The statistics of these networks are summarized in Table 6.1.

6.4.2 Baseline Methods

We compare SINE with the following baseline methods:

- **DeepWalk** [63] / **node2vec** [19]: node2vec is equivalent to DeepWalk under the default setting $p = 1$ and $q = 1$. They learn node representations by preserving the similarity between nodes sharing similar contexts in random walks.

[2]https://aminer.org/citation (Version 3 is used)

- **LINE-1** [76]: LINE-1 denotes the version of LINE that learns node representations by modeling the first-order proximity.
- **LINE-2** [76]: LINE-2 is the version of LINE that preserves the second-order proximity.
- **SDNE** [87]: SDNE uses a semi-supervised autoencoder to learn deep node representations that preserve both the first-order and second-order proximity.
- **Attribute**: This baseline learns node representations from only node attributes with the SINE learning framework.
- **TADW** [96]: TADW incorporates node content features into DeepWalk's network representation learning paradigm via inductive matrix factorization [57].
- **HSCA** [101]: HSCA enhances TADW via enforcing a first-order proximity preserving objective.
- **UPP-SNE** [102]: UPP-SNE learns node representations by performing a structure-aware non-linear mapping on node content features.
- **MVC-DNE** [97]: MVC-DNE encodes network structure and node content attributes into node representations through the deep autoencoder based cross-view learning. The decoding based version is used.

Due to the large size of DBLP(Full), TADW, HSCA, UPP-SNE and MVC-DNE can not run on this dataset. Thus, on DBLP(full), only DeepWalk, LINE-1, LINE-2 and Attribute are compared with SINE in Section 6.4.6.

6.4.3 Experimental Settings

For DeepWalk, UPP-SNE and SINE, we set the length of random walks $L = 100$, the number of random walks starting from each node $\gamma = 40$, and the window size $t = 10$.

For fair comparison, DeepWalk and UPP-SNE are trained using the same strategy with SINE: we first collect node context pairs, and then update parameters with stochastic gradient descent by sampling a node context pair at each iteration. Negative

sampling is adopted by DeepWalk, LINE-1, LINE-2, UPP-SNE, Attribute, and SINE, where the number of negative samples K is set to 5 uniformly. For the 6 stochastic gradient descent based algorithms, the maximum number of iterations I is set to 100 million on Cora, Citeseer and DBLP(Subgraph), and 1 billion on DBLP(Full), and the learning rate η decreases from the starting value $\eta_0 = 0.025$ to $\eta_\tau = \eta_0(1 - \tau/I)$ after every 10,000 iterations, with τ being the number of elapsed iterations. Parameters of TADW and HSCA are set to their default values. For SDNE, its hyperparameters α and ν are set to 0.01, and β is set to 10. The number of neurons at each layer is set to 2708-256, 3312-256, and 18,448-1,024-256 on Cora, Citeseer, and DBLP(Subgraph), respectively. For MVC-DNE, on Cora, Citeseer, and DBLP(Subgraph), the number of neurons at each layer in structure view is respectively set to 2708-128, 3312-128, and 18,448-512-128, and the number of neurons at each layer in node attribute view is set to 1,433-128, 3,703-128, and 5,959-128. For SDNE and MVC-DNE, 500 epochs are run for both pre-training and parameter fine-tuning. Other parameters of SDNE and MVC-DNE are set according to [97]. For SINE and all baseline methods, the dimension of learned node representations is set to 256.

6.4.4 Performance Comparison on Incomplete Networks

In this section, we conduct experiments to compare the performance of SINE and baseline methods on incomplete networks. To better understand the ability of different network embedding algorithms to deal with missing data, we investigate the following four research questions:

- **Q1**: How is the performance affected when a portion of node attributes are missing compared with the complete network? (**complete attribute vs. incomplete attribute**)
- **Q2**: How does the performance change when the attributes of structurally important nodes are missing compared with missing attributes for randomly selected nodes? (**random vs. important X row missing**)

- **Q3**: How does the performance change when the attributes at important dimensions are missing compared with missing attributes at random dimensions? (**random vs. important X column missing**)
- **Q4**: How do different network embedding algorithms perform for link prediction when a portion of edges are missing?

To answer research questions Q1, Q2, and Q3, we compare the performance of different network embedding algorithms on Cora, Citeseer and DBLP(Subgraph) under 5 settings: (1) complete network, (2) randomly selecting 50% nodes and dropping all of their attributes (random X row missing), (3) selecting the top 50% structurally important nodes measured by degree and dropping their attributes (important X row missing), (4) missing node attributes at randomly selected 50% dimensions (random X column missing), and (5) missing node attributes at top 50% important dimensions measured by mutual information with class label (important X column missing). As the baselines of Attribute, TADW, HSCA and UPP-SNE require all nodes to have observed attributes, under settings (2) and (3), for nodes with no attributes, we fill their attribute values with the observed modes at each dimension. To compare the performance, using the learned node representations as features, we conduct multi-class node classification experiments on Cora, Citeseer and DBLP, and carry out node clustering experiments on Cora.

To answer research question Q4, we perform link prediction experiments on Cora and DBLP(Full). We randomly remove 30%, 50% and 70% links, learn node representations from the respective incomplete networks with different algorithms, and compare their performance for predicting missing links.

Comparison of Classification Performance

Using the learned node representations as features, we train an SVM classifier (with the LIBLINEAR implementation [15]) on the randomly selected 50% samples, and then classify the remainder 50% samples with the learned classifier. The random training and test data split is repeated for 10 times, and averaged Micro-F_1 and Macro-F_1 values are used to evaluate the classification performance.

Table 6.2 Node Classification Results on Cora

	Method	Complete	Incomplete			
			X Row Missing		X Column Missing	
			Random	Important	Random	Important
Micro-F_1	DeepWalk	0.8245	0.8245	0.8245	0.8245	**0.8245**
	LINE-1	0.7697	0.7697	0.7697	0.7697	0.7697
	LINE-2	0.7081	0.7081	0.7081	0.7081	0.7081
	SDNE	0.6047	0.6047	0.6047	0.6047	0.6047
	Attribute	0.7222	0.4905	0.4721	0.6581	0.2592
	TADW	**0.8543**	0.7387	0.3982	0.8383	0.7068
	HSCA	**0.8569**	0.7366	0.3791	**0.8530**	0.7309
	UPP-SNE	0.8191	0.5538	0.5438	0.8058	0.7201
	MVC-DNE	0.7528	0.6258	0.6500	0.7075	0.4978
	SINE	0.8340	**0.8329**	**0.8383**	0.8332	0.8010
Macro-F_1	DeepWalk	0.8184	0.8184	0.8184	0.8184	**0.8184**
	LINE-1	0.7673	0.7673	0.7673	0.7673	0.7673
	LINE-2	0.6986	0.6986	0.6986	0.6986	0.6986
	SDNE	0.5811	0.5811	0.5811	0.5811	0.5811
	Attribute	0.6927	0.4373	0.4130	0.6302	0.1427
	TADW	**0.8457**	0.7297	0.2903	0.8295	0.6890
	HSCA	**0.8487**	0.7151	0.2521	**0.8445**	0.7130
	UPP-SNE	0.8088	0.5322	0.5227	0.7985	0.7072
	MVC-DNE	0.7263	0.5872	0.6176	0.6808	0.4447
	SINE	0.8214	**0.8218**	**0.8266**	0.8228	0.7896

Tables 6.2-6.4 report node classification results of different network embedding algorithms on Cora, Citeseer and DBLP(Full), under the five settings: (1) complete, (2) random X row missing, (3) important X row missing, (4) random X column missing, and (5) important X column missing. For each setting, the best Micro-F_1 and Macro-F_1 values are **bold-faced**, and the second best performers are underlined.

By comparing column 3 with columns 4-7 in Tables 6.2-6.4, we can respond to research question Q1: when node attributes become incomplete, the performance of the existing attributed network embedding baselines (Attribute, TADW, HSCA, UPP-SNE and MVC-DNE) drops remarkably in most cases, while SINE often shows greater stability. This attributes to the flexible way that SINE leverages node attributes, making it able to best utilize observed node attributes and to diminish the negative impact caused by missing attributes.

To better understand research question Q2, we compare column 4 with column 5 in Tables 6.2-6.4. When the attributes of structurally important nodes are missing, the performance of TADW and HSCA degrades dramatically on all three datasets. Inter-

Table 6.3 Node Classification Results on Citeseer

	Method	Complete	Incomplete			
			X Row Missing		X Column Missing	
			Random	Important	Random	Important
Micro-F_1	DeepWalk	0.6038	0.6038	0.6038	0.6038	**0.6038**
	LINE-1	0.5684	0.5684	0.5684	0.5684	0.5684
	LINE-2	0.4673	0.4673	0.4673	0.4673	0.4673
	SDNE	0.4614	0.4614	0.4614	0.4614	0.4614
	Attribute	0.6883	0.4325	0.4287	0.6549	0.2264
	TADW	0.6957	0.5086	0.3267	**0.7014**	0.5199
	HSCA	0.6825	0.3708	0.3128	**0.7031**	0.5358
	UPP-SNE	**0.7124**	0.4418	0.4551	0.6858	0.5662
	MVC-DNE	0.6954	0.5642	0.5467	0.6578	0.3187
	SINE	**0.7136**	**0.6791**	**0.6882**	**0.7030**	0.5682
Macro-F_1	DeepWalk	0.5465	0.5465	0.5465	0.5465	**0.5465**
	LINE-1	0.5300	0.5300	0.5300	0.5300	0.5300
	LINE-2	0.4192	0.4192	0.4192	0.4192	0.4192
	SDNE	0.3986	0.3986	0.3986	0.3986	0.3986
	Attribute	0.6317	0.4051	0.4018	0.5951	0.1962
	TADW	0.6403	0.4673	0.2806	**0.6480**	0.4613
	HSCA	0.6292	0.3191	0.2589	**0.6505**	0.4740
	UPP-SNE	**0.6567**	0.4135	0.4323	0.6230	0.5027
	MVC-DNE	0.6269	0.4996	0.4788	0.5851	0.2714
	SINE	**0.6564**	**0.6201**	**0.6256**	0.6401	0.5017

estingly, on DBLP, the Attribute baseline experiences a performance gain. This might be due to the fact that structurally important nodes tend to have strong correlations with neighbor nodes on attribute values, resulting in information redundancy. When such redundancy is removed, Attribute achieves better classification performance. Due to the same reason, UPP-SNE and MVC-DNE share the same trends with Attribute on DBLP. However, under both settings, SINE exhibits more stable performance and outperforms all other baselines.

To answer research question Q3, we compare column 6 with column 7 in Tables 6.2-6.4. The performance of Attribute decreases dramatically when node attributes at important dimensions are missing, compared with the missing in randomly selected dimensions. Accordingly, attributed network embedding algorithms consistently experience a dramatic performance drop, to a level inferior to the only structure preserving network embedding algorithms. In this case, the remaining node attributes are of poor quality, making them deteriorate rather than complement network structure in learning

Table 6.4 Node Classification Results on DBLP(Subgraph)

	Method	Complete	Incomplete			
			X Row Missing		X Column Missing	
			Random	Important	Random	Important
Micro-F_1	DeepWalk	0.8005	0.8005	0.8005	0.8005	**0.8005**
	LINE-1	0.7810	0.7810	0.7810	0.7810	0.7810
	LINE-2	0.7125	0.7125	0.7125	0.7125	0.7125
	SDNE	0.6464	0.6464	0.6464	0.6464	0.6464
	Attribute	0.7509	0.5665	0.6275	0.6226	0.3952
	TADW	0.8070	0.7195	0.4544	0.7533	0.4928
	HSCA	0.8023	0.6227	0.4178	0.7532	0.4628
	UPP-SNE	0.8267	0.6092	0.6606	0.7078	0.4141
	MVC-DNE	0.7697	0.6693	0.7089	0.7025	0.6232
	SINE	**0.8370**	**0.8278**	**0.8370**	**0.8275**	0.7953
Macro-F_1	DeepWalk	0.7176	0.7176	0.7176	0.7176	**0.7176**
	LINE-1	0.6896	0.6896	0.6896	0.6896	0.6896
	LINE-2	0.6129	0.6129	0.6129	0.6129	0.6129
	SDNE	0.3973	0.3973	0.3973	0.3973	0.3973
	Attribute	0.6604	0.4403	0.5412	0.4870	0.1882
	TADW	0.7271	0.5539	0.2935	0.6443	0.2760
	HSCA	0.6926	0.4537	0.2436	0.6140	0.2549
	UPP-SNE	0.7668	0.5196	0.5988	0.6187	0.1985
	MVC-DNE	0.6699	0.5222	0.5961	0.5854	0.4022
	SINE	**0.7731**	**0.7620**	**0.7727**	**0.7618**	0.7122

network embeddings. By contrast, the performance of SINE drops less significantly, demonstrating its better robustness to missing node attributes.

From Tables 6.2-6.4, we can conclude that SINE achieves the best overall performance under all node attribute missing settings. This not only verifies the effectiveness of SINE in handling missing node attributes, but also signifies its great potential to solve real-world applications with missing data.

Clustering Performance Comparison

As a complement to node classification, we also conduct node clustering experiments on Cora. We feed node representations learned by different network embedding algorithms into the K-means clustering algorithm and group them into 7 categories. To alleviate the impact caused by random initialization, we run K-means for 20 times and report averaged Accuracy and NMI [73] values. The clustering results are presented in Table 6.5, with the best and second performer highlighted by **bold** and underline respectively.

Table 6.5 Node Clustering Results on Cora

	Method	Complete	Incomplete			
			X Row Missing		X Column Missing	
			Random	Important	Random	Important
Accuracy	DeepWalk	0.6097	0.6097	0.6097	0.6097	0.6097
	LINE-1	0.3444	0.3444	0.3444	0.3444	0.3444
	LINE-2	0.4162	0.4162	0.4162	0.4162	0.4162
	SDNE	0.3897	0.3897	0.3897	0.3897	0.3897
	Attribute	0.3869	0.3216	0.3240	0.3454	0.3021
	TADW	0.3561	0.3123	0.3045	0.4000	0.3316
	HSCA	0.3721	0.3257	0.3046	0.3956	0.3357
	UPP-SNE	0.6270	0.4496	0.4269	**0.6319**	0.5573
	MVC-DNE	0.6228	0.4058	0.3685	0.5352	0.3021
	SINE	**0.6323**	**0.6397**	**0.6355**	**0.6343**	**0.6297**
NMI	DeepWalk	0.4165	0.4165	0.4165	0.4165	0.4165
	LINE-1	0.0910	0.0910	0.0910	0.0910	0.0910
	LINE-2	0.1806	0.1806	0.1806	0.1806	0.1806
	SDNE	0.1409	0.1409	0.1409	0.1409	0.1409
	Attribute	0.1475	0.0528	0.0661	0.0836	0.0056
	TADW	0.1377	0.0321	0.0120	0.1603	0.0698
	HSCA	0.1646	0.0556	0.0113	0.1892	0.0767
	UPP-SNE	0.4377	0.2171	0.2003	0.4427	0.3490
	MVC-DNE	0.3737	0.1326	0.1119	0.3018	0.0067
	SINE	**0.4456**	**0.4458**	**0.4401**	**0.4468**	**0.4376**

Similar to node classification results, SINE achieves the best clustering performance with small variance across all node attribute missing settings.

Link Prediction Performance Comparison

To answer research question Q4, we carry out link prediction experiments on Cora and DBLP(Full). Specifically, we compare the performance of different network embedding algorithms on link prediction, when a portion of edges are missing. Following [19], we perform link prediction using the edge features constructed from the learned node representations with the operators listed in Table 6.6. We randomly remove 30%, 50% and 70% of edges. To construct the test set, for each removed edge (v_i, v_j), we randomly sample a negative node pair (v_i, v_k) with $(v_i, v_k) \notin \mathcal{E}$ as negative ground truth. To construct the training set, for each connected node pair (v_i, v_j), we randomly sample a negative node pair (v_i, v_k), with no edges between v_i and v_k observed in the remaining network.

Table 6.6 Operators to Construct Edge Features

Operator	Symbol	Definition
Average	$\boxplus$	$[\Phi(v_i) \boxplus \Phi(v_j)]_k = \frac{\Phi_k(v_i)+\Phi_k(v_j)}{2}$
Hadamard	$\boxdot$	$[\Phi(v_i) \boxdot \Phi(v_j)]_k = \Phi_k(v_i) \cdot \Phi_k(v_j)$
Weighted-L1	$\|\cdot\|_{\bar{1}}$	$\|\Phi(v_i) \cdot \Phi(v_j)\|_{\bar{1}k} = \|\Phi_k(v_i) - \Phi_k(v_j)\|$
Weighted-L2	$\|\cdot\|_{\bar{2}}$	$\|\Phi(v_i) \cdot \Phi(v_j)\|_{\bar{2}k} = (\Phi_k(v_i) - \Phi_k(v_j))^2$

Table 6.7 Heuristic scores for predicting the link between node pair (v_i, v_j) with their direct neighbor sets $\mathcal{N}(v_i)$ and $\mathcal{N}(v_j)$

Score	Definition
Common Neighbors	$\|\mathcal{N}(v_i) \cap \mathcal{N}(v_j)\|$
Jaccard's Coefficient	$\frac{\|\mathcal{N}(v_i) \cap \mathcal{N}(v_j)\|}{\|\mathcal{N}(v_i) \cup \mathcal{N}(v_j)\|}$
Adamic-Adar Score	$\sum_{v_k \in \mathcal{N}(v_i) \cap \mathcal{N}(v_j)} \frac{1}{\log \|\mathcal{N}(v_k)\|}$
Preferential Attachment	$\|\mathcal{N}(v_i)\| \cdot \|\mathcal{N}(v_j)\|$

SVM implemented by LIBLINEAR [15] is used to perform training and testing on edge features. Due to the memory limitation, for DBLP(Full), 5% training and test samples are randomly selected from the two original sets respectively. We also compare SINE with the common neighbor based heuristic link prediction methods, which are given in Table 6.7. The Area Under Curve (AUC) score is used to evaluate the link prediction performance.

Tables 6.8-6.9 report the link prediction results on Cora and DBLP(Full), with the best performer and the second best performer highlighted by **bold** and underline respectively. As can be seen, on both Cora and DBLP(Full), the proposed SINE algorithm with the Hadamard operator consistently achieves the best link prediction performance with all missing edge ratios. We can also observe that, compared with attributed network embedding algorithms, the only structure preserving network embedding algorithms are more vulnerable to missing edges. As the ratio of missing edges increases, apart from the Attribute baseline, all methods experience a performance drop, while SINE retains more robust results. There are two main reasons: first, SINE uses random walks to bridge nodes with no immediate links and preserve their similarity

Table 6.8 AUC Values for Link Prediction on Cora

Operator	Method	30%	50%	70%
	Common Neighbors	0.5115	0.5045	0.5019
	Jaccard's Coefficient	0.5115	0.5045	0.5019
	Adamic-Adar	0.5115	0.5045	0.5019
	Pref. Attachment	0.5445	0.5282	0.5229
Average	DeepWalk	0.5123	0.4952	0.5039
	LINE-1	0.5542	0.5307	0.5157
	LINE-2	0.5280	0.5220	0.5178
	SDNE	0.5585	0.5488	0.5384
	Attribute	0.5320	0.5326	0.5264
	TADW	0.5571	0.5434	0.5386
	HSCA	0.5543	0.5368	0.5272
	UPP-SNE	0.5336	0.5213	0.5214
	MVC-DNE	0.5348	0.5202	0.5269
	SINE	0.5507	0.5425	0.5375
Hadamard	DeepWalk	0.7632	0.6657	0.5563
	LINE-1	0.6702	0.6003	0.5403
	LINE-2	0.6763	0.5808	0.5255
	SDNE	0.5730	0.5516	0.5293
	Attribute	0.8095	0.7966	0.7989
	TADW	0.8361	0.7850	0.7202
	HSCA	0.8623	0.8108	0.7351
	UPP-SNE	0.8615	0.8160	0.7415
	MVC-DNE	0.6749	0.6090	0.5643
	SINE	**0.8804**	**0.8545**	**0.8289**
Weighted-L1	DeepWalk	0.8368	0.7302	0.6015
	LINE-1	0.6303	0.5900	0.5199
	LINE-2	0.6429	0.5726	0.5109
	SDNE	0.5643	0.5365	0.5219
	Attribute	0.7402	0.7212	0.7179
	TADW	0.7443	0.6611	0.5834
	HSCA	0.7725	0.6840	0.5859
	UPP-SNE	0.8512	0.7970	0.7095
	MVC-DNE	0.7883	0.7662	0.7418
	SINE	0.8438	0.8035	0.7643
Weighted-L2	DeepWalk	0.8368	0.7356	0.6041
	LINE-1	0.6102	0.5720	0.5131
	LINE-2	0.6631	0.4888	0.4874
	SDNE	0.5267	0.4915	0.5198
	Attribute	0.7398	0.7311	0.7333
	TADW	0.7279	0.6540	0.5742
	HSCA	0.7400	0.6630	0.5749
	UPP-SNE	0.8414	0.7937	0.7149
	MVC-DNE	0.7359	0.7331	0.7264
	SINE	0.8344	0.7965	0.7581

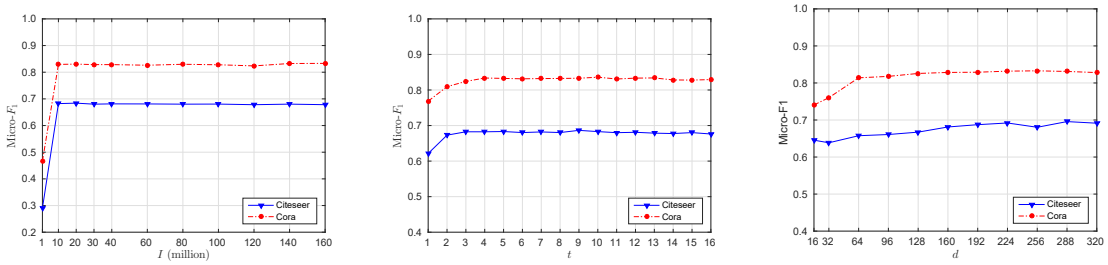
in the learned node representations; second, SINE makes the best of the available node features to complement network structure for more effective network embedding.

Table 6.9 AUC Values for Link Prediction on DBLP(Full)

Operator	Method	30%	50%	70%
	Common Neighbors	0.5024	0.5013	0.5006
	Jaccard's Coefficient	0.5024	0.5013	0.5006
	Adamic-Adar	0.5024	0.5013	0.5006
	Pref. Attachment	0.8898	0.8768	0.8513
Average	DeepWalk	0.9016	0.8896	0.8619
	LINE-1	0.5500	0.5389	0.5281
	LINE-2	0.8810	0.8689	0.8424
	Attribute	0.6215	0.6205	0.6185
	SINE	0.8747	0.8692	0.8533
Hadamard	DeepWalk	0.9841	0.9668	0.9270
	LINE-1	0.9835	0.9590	0.8999
	LINE-2	0.9766	0.9576	0.9146
	Attribute	0.8490	0.8500	0.8507
	SINE	**0.9887**	**0.9831**	**0.9670**
Weighted-L1	DeepWalk	0.9766	0.9463	0.8819
	LINE-1	0.8157	0.7573	0.6397
	LINE-2	0.8787	0.8601	0.8400
	Attribute	0.7818	0.7819	0.7778
	SINE	0.9382	0.9150	0.8497
Weighted-L2	DeepWalk	0.9390	0.9018	0.8160
	LINE-1	0.8055	0.7452	0.6172
	LINE-2	0.6910	0.6618	0.8246
	Attribute	0.7681	0.7701	0.7688
	SINE	0.9029	0.8787	0.8113

6.4.5 Experiments on Parameter Sensitivity

We also conduct experiments to investigate the sensitivity of SINE to three important parameters: the maximum number of iterations I, window size t, and the dimension of learned node embeddings d. We fix any two of the three parameters and study how the performance of SINE changes by varying the remaining one in turn. Fig. 6.2 reports the change of Micro-F_1 values for node classification on Cora and Citeseer with regards to the three parameters, under the settings where attributes in randomly selected 50% nodes are missing (random X row missing). Here, we observe a similar trend with an increase of all three parameters: SINE performs increasingly better and stabilizes after a threshold.



(a) I (b) t (c) d

Fig. 6.2 Parameter sensitivity

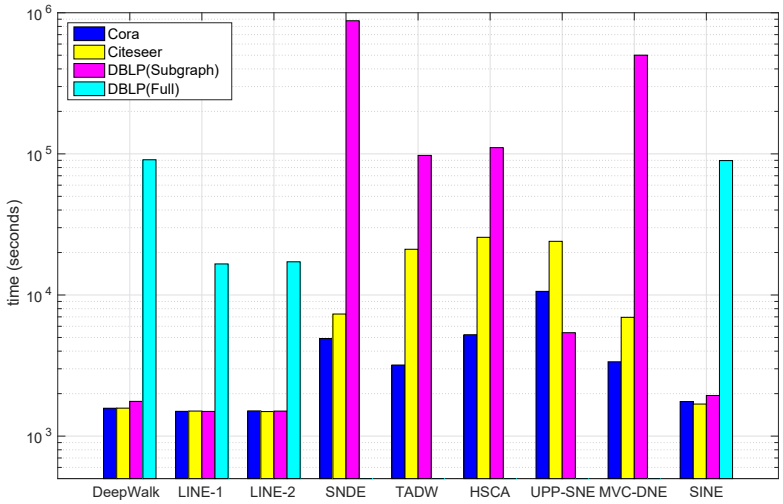


Fig. 6.3 The comparison of running time of different network embedding algorithms on the log scale

6.4.6 Running Time Comparison

We evaluate the running time of different network embedding algorithms in this section. We hope this empirical evaluation can help compare the efficiency and scalability of these algorithms, though they are implemented in different programming languages, with DeepWalk, UPP-SNE and SINE in C, LINE in C++, SDNE and MVC-DNE in Python, as well as TADW and HSCA in Matlab. Fig. 6.3 compares the CPU running time (log-scale) of different algorithms on Cora, Citeseer, DBLP(Subgraph) and DBLP(Full). On DBLP(Full), due to its large size, only the running time of DeepWalk, LINE-1, LINE-2 and SINE is available. We can see that, SINE is much

more efficient than SDNE and other attributed network embedding methods including TADW, HSCA, UPP-SNE and MVC-DNE. The running time of SINE is comparable to that of DeepWalk, LINE-1 and LINE-2. This demonstrates high efficiency of SINE and its ability to scale to large-scale networks on a single machine (without GPU support).

6.5 Conclusion

Large-scale networks often contain incomplete node attributes and/or missing links, which impose significant challenges to attributed network embedding. Because missing node attributes or links result in inaccurate node similarity estimation, most existing methods are vulnerable to missing data, and minor presence of missing information may significantly deteriorate the algorithm performance. In this chapter, we propose a novel scalable incomplete network embedding (SINE) method, which uses probabilistic learning of node-context and node-attribute relationships to tackle missing data on large-scale networks. SINE couples network node content and topology structures through a three-layer neural network, where each node learns its representation by considering context nodes and observable attributes of the node. By doing so, SINE fully minimizes the impact of missing data on the learning of node representations. A stochastic gradient descent based online algorithm is derived to ensure SINE can scale to large-scale networks. Extensive experiments and comparisons demonstrate the effectiveness and scalability of SINE for learning network embeddings on large-scale incomplete networks.

Part II

Task-dependent Attributed Network Embedding

Overview

Task-dependent attributed network embedding aims to learn node representations that can directly benefit the downstream tasks. To make the learned node representations more suitable for the targeted network analytic tasks, the attributed network embedding needs to adapt to the objectives of targeted tasks. Meanwhile, the learned node representations should be informative enough so that network structure and node content attributes can be effectively leveraged to solve the targeted tasks. In this thesis, we try to use attributed network embedding to solve two tasks, *i.e.*, collective classification and node similarity search, and develop two task-dependent attributed network embedding algorithms.

In Chapter 7, for the collective classification task, we propose a discriminative attributed network embedding algorithm that simultaneously learns (1) node representations by exploiting topological paths between labeled and unlabeled nodes, as well as node content attributes, and (2) a linear classifier that classifies node representations into different groups. By unleashing the discriminative power of labeled nodes through an empirical loss minimization objective, the learned node representations are expected to be directly benefit node classification with sparse labels.

In Chapter 8, to handle node similarity search, we propose a binary attributed network embedding algorithm that learns binary node representations from both network structure and node content though the continuation technique. The learned binary node representations not only reduce memory usage to represent nodes, but also allow fast bitwise comparisons to support much quicker network node similarity search compared to calculating Euclidean distance in the continuous node representation space.

Chapter 7

Discriminative Attributed Network Embedding for Collective Classification

7.1 Introduction

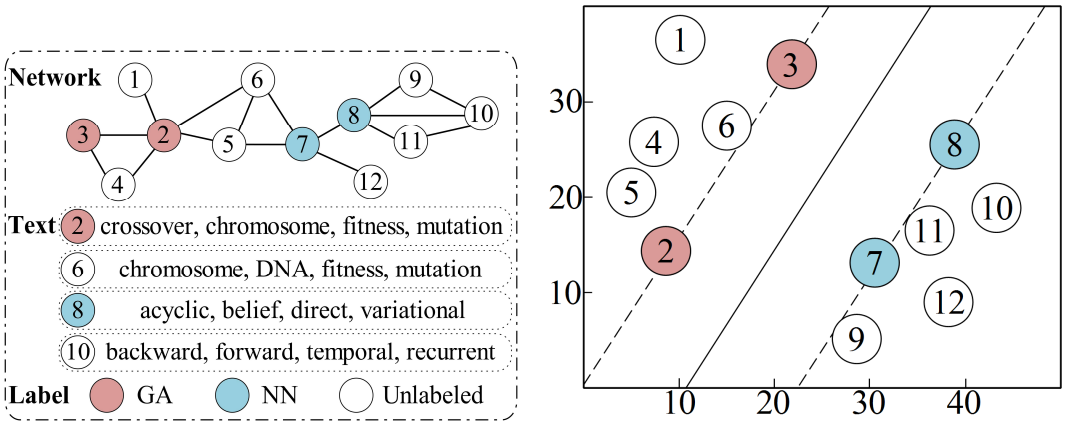
With the widespread use of social media, e-commerce, and cyber-physical systems, information networks, such as social networks, communication networks, and citation networks, have become increasingly ubiquitous and content-rich over time. Information networks are typically represented as a graph structure, where nodes denote instances (*e.g.*, users or publications) that are often characterized by rich content features, and edges denote relationships between nodes (*e.g.*, friendship or citation relationships). Node classification is one of the important tasks in analyzing such content-rich networks. Collective classification (CC) addresses this task by making joint predictions on labels of connected nodes. Among others, the Iterative Classification Algorithm (ICA) is a canonical method for incorporating neighbor labels as relational information into the collective classification process, which has been demonstrated to be most effective in classifying many real-world networks [31, 68].

In reality, existing methods for training ICA models yet require a connected, densely-labeled subgraph to be provided for learning. To ensure satisfactory performance,

this densely-labeled subgraph should be sufficiently large so that dependency relationships among network structure and node labels can be accurately learned. However, acquiring labels of inter-connected nodes in a network can be very expensive and time-consuming [16, 52]. Therefore, very often the provided network can be only sparsely labeled, with a very small portion of labeled nodes that may not comprise a connected subgraph. At low label density, most of the nodes have too few or even no connections to labeled nodes, making it difficult to accurately estimate statistical relationships between network structure, nodes' content features and labels. As a result, the performance of collective classification can significantly degrade on sparsely labeled networks.

To tackle the label sparsity, two branches of research studies have been proposed in recent literature. The first is semi-supervised collective classification [51, 52], which leverages unlabeled nodes to enhance classification performance. These methods typically first use a bootstrap step to predict labels for all unlabeled nodes, through a classifier trained on labeled nodes using only content features. After that, predicted labels, together with known labels, are used to construct relational features and perform collective inference. The second line of research focuses on increasing network connectivity between unlabeled and labeled nodes [17, 49, 69]. This is achieved by either adding extra edges based on node attribute similarity, or constructing a latent graph from different information sources. However, most of these methods consider either local dependencies between network structure and labels, or correlations between nodes' content features and labels, but ignore the significant role of their interactions in jointly classifying labels of related nodes.

In this chapter, we propose a novel discriminative matrix factorization (DMF) based algorithm that effectively learns a compact network representation by combining network structure, node content, together with sparsely labeled nodes, such that nodes represented in the new space can be well separated with minimum loss. Consider the example shown in Figure 7.1, where given a citation network with only four labeled nodes, we aim to predict the labels of unlabeled nodes into two categories: Genetic Algorithms (GA) and Neutral Networks (NN). According to network structure, node 6's category is ambiguous because it is connected to nodes both in GA and NN. However, given that it has similar node text content with node 2, it can be correctly classified as



(a) Input (b) Output

Fig. 7.1 A toy example demonstrating DMF objectives. Given a citation network composed of papers from two categories, Genetic Algorithms (GA) and Neutral Networks (NN), DMF aims to learn informative and discriminative node representations that combine network structure, node content, and sparse node labels to maximally separate nodes into different categorizes with minimum loss, as shown in (b).

GA. On the contrary, node 10 has uninformative node content about its category, but it has three paths to node 8. Hence, it can be classified as NN. Only by combining network structure, node content, and node labels, nodes can be mapped into new representations that are maximally separated to optimize classification performance, as shown in Figure 7.1(b).

Our main idea of tackling node label sparsity is twofold: (1) leverage random walks to find path dependencies between unlabeled nodes and sparsely labeled nodes in the network; and (2) learn a latent network representation that fully embeds nodes' content information and augmented network structure to minimize classification loss. Intuitively, the proposed DMF is motivated by observations that, for sparsely labeled networks, an unlabeled node in the network may not have labeled neighbors or have very low structure similarity to labeled nodes, so existing semi-supervised [51, 52] and connectivity-based methods [17, 49, 69] would not have satisfactory performance. Alternatively, we can use network walks to examine unlabeled nodes' correlations to labeled nodes. Because a given network has an exponential number of walk combinations, this essentially increases the relationship density for sparsely labeled networks.

In order to utilize network walks to guide node classification with minimum loss, we leverage an existing Text-Associated DeepWalk (TADW) approach [96], which embeds nodes' linkage information together with their content features into a low-dimensional space. To enable supervised information to be propagated from labeled nodes to unlabeled nodes, we formulate a new matrix factorization objective function that integrates network representation learning with an empirical loss minimization of a linear classifier on labeled nodes. An efficient optimization algorithm based on conjugate gradient methods is then developed to solve the new objective function. Our DMF algorithm learns a latent representation of the network that not only preserves intrinsic network structure, but also uncovers discriminative power learned from labeled nodes to directly benefit collective classification. Experimental results on real-world networks show that our DMF algorithm outperforms state-of-the-art baselines in sparsely labeled networks.

The main contributions of this chapter are threefold:

- We study the label sparsity problem in network settings that many real-world applications often encounter, but traditional algorithms are ineffective to address;
- We propose a unified matrix factorization framework that seamlessly integrates representation learning and discriminative learning into a new optimization problem, and derive solutions with improved efficiency;
- We combine node content, augmented network structure, and node labels to benefit node classification with minimum loss.

7.2 Problem Definition and Preliminaries

In this section, we formally define the research problem and review preliminaries of text-associated DeepWalk as the basis of our proposed solution.

7.2.1 Problem Definition

In our problem setting, an undirected network $G = (\mathcal{V}, \mathcal{E}, T, \mathcal{Y})$ is given, where $\mathcal{V}$ denotes a set of nodes, and $\mathcal{E}$ denotes a set of edges connecting nodes. T is the node text feature matrix, where the i-th column of T is a text feature vector $\boldsymbol{t}_i$ that characterizes node $v_i \in \mathcal{V}$. Each node $v_i \in \mathcal{V}$ is also associated with a class label $y_i \in \mathcal{Y}$, where we focus on a binary classification problem, *i.e.*, $\mathcal{Y} = \{+1, -1\}$. In network G, we assume that only a small number of nodes are labeled. Let $\mathcal{L}$ be the set of indices of labeled nodes, and $Y_{\mathcal{L}}$ be class labels of labeled nodes. The set of labeled nodes is represented as $\mathcal{V}_{\mathcal{L}}$ ($v_i \in \mathcal{V}_{\mathcal{L}}$, for each $i \in \mathcal{L}$).

Formally, given a small set of labeled nodes $\mathcal{V}_{\mathcal{L}} \in G$, our task **aims** to: (1) learn a task-specific node representation $\boldsymbol{x}$ for all nodes $\mathcal{V} \in G$, and simultaneously (2) learn a classifier $h(\boldsymbol{x})$ to predict labels of unlabeled nodes $\mathcal{V} \setminus \mathcal{V}_{\mathcal{L}}$ with maximum classification performance.

Intuitively, we aim to learn a latent representation of the network that not only preserves the underlying network structure, but also uncovers discriminative power inferred from a limited number of labeled nodes to maximize classification performance on unlabeled nodes. Because our proposed solution shares a similar matrix factorization principle as Text-associated DeepWalk (TADW) for network representation learning, in the following, we briefly review TADW.

7.2.2 Preliminaries: Text-Associated DeepWalk

Motivated by DeepWalk [63], Text-Associated DeepWalk (TADW) [96] incorporates node text features into network representation learning. DeepWalk originally generalizes the Skip-Gram model [55] from word representation learning to node representation learning. In DeepWalk, nodes in networks are taken as words in natural language and a set of short random walk sequences on the given network are generated to capture context for nodes. Given node v_i and its context nodes within t window size, $\{v_{i-t}, \cdots, v_{i+t}\} \setminus v_i$, the representation for node v_i, $\Phi(v_i)$, is learned by maximizing the

conditional probability:

$$\mathrm{P}(\{v_{i-t},\cdots,v_{i+t}\}\setminus v_i|\Phi(v_i)) = \prod_{j=i-t,j\neq i}^{i+t}\mathrm{P}(v_j|\Phi(v_i)). \tag{7.1}$$

The probability $\mathrm{P}(v_j|\Phi(v_i))$ is approximated using Hierarchical Softmax [56] to speed up training.

We denote the PageRank matrix for the given graph G as $P \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$. If the degree of node v_i is denoted as d_i, then $P_{ij} = 1/d_i$, if $(v_i, v_j) \in \mathcal{E}$; otherwise, $P_{ij} = 0$. In [96], DeepWalk is then proved to be equivalent to factorizing a matrix $M \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ given by

$$M = \log\frac{P + P^2 + \cdots + P^t}{t}, \tag{7.2}$$

where log is the element wise function for a matrix. Matrix M, which is referred to as *node-path-matrix*, can be factorized as two low-rank matrices $W \in \mathbb{R}^{k\times|\mathcal{V}|}$ and $H \in \mathbb{R}^{k\times|\mathcal{V}|}$ by solving the following problem

$$\min_{W,H}\|M - W^{\mathrm{T}}H\|_F^2 + \lambda(\|W\|_F^2 + \|H\|_F^2), \tag{7.3}$$

where the minimization of $\|M - W^{\mathrm{T}}H\|_F^2$ guarantees precise factorization. The minimization of $\|W\|_F^2$ and $\|H\|_F^2$ imposes low-rank constraint on W and H, and λ controls the trade-off between the two terms. Once factorization is finished, W and H can be concatenated together to form a $2k$-dimensional representation for each node.

To make the representation learned by matrix factorization based DeepWalk more informative, TADW imports node text features $T \in \mathbb{R}^{f_t\times|\mathcal{V}|}$ into the matrix factorization of (7.3), with f_t being the number of text features. The matrix factorization formulation is changed into

$$\min_{W,H}\|M - W^{\mathrm{T}}HT\|_F^2 + \lambda(\|W\|_F^2 + \|H\|_F^2), \tag{7.4}$$

where $H \in \mathbb{R}^{k\times f_t}$. For computational efficiency, in [96], M is approximated by $(P + P^2)/2$. After W and H are solved, nodes are represented by concatenating W and HT as feature vectors so that traditional machine learning algorithms can be directly applied to learn classifiers.

7.3 DMF: Discriminative Matrix Factorization

In this section, we formulate our new discriminative matrix factorization optimization problem and derive efficient algorithms to solve the problem.

7.3.1 The Optimization Problem

One of the major deficiencies of TADW is that it lacks discriminative power for a specific classification task such as node classification. When the network is sparsely labeled, learning a classifier from very few labeled nodes on the node representations learned by TADW cannot effectively capture statistical relationships between node representations and nodes' class labels to achieve good classification performance on unlabeled nodes. In this work, we aim to learn a task-specific and discriminative representation by enforcing matrix factorization with an empirical loss minimization of a linear classifier trained on labeled nodes. By enabling simultaneous learning of node representations and a linear classifier using these representations, we expect that the learned node representations are not only informative but also discriminative for better capturing statistical relationships between node representations and nodes' class labels.

The optimization of TADW in (7.4) can be rewritten as:

$$\min_{W,H} \frac{1}{2} \sum_{i,j=1}^{|\mathcal{V}|} (M_{ij} - \boldsymbol{w}_i^{\mathrm{T}} H \boldsymbol{t}_j)^2 + \frac{\lambda}{2} \left(\|W\|_F^2 + \|H\|_F^2 \right), \tag{7.5}$$

where $\frac{1}{2}$ is multiplied to the objective function for mathematical convenience. $\boldsymbol{w}_i$ and $\boldsymbol{t}_j$ are the i-th column of matrix W and j-th column of matrix T, respectively.

Assume that the representation of node v_n in the given network is provided in the form of a column vector $\boldsymbol{x}_n$. We can then learn a linear classifier $h(\boldsymbol{x}_n) = \boldsymbol{\eta}^{\mathrm{T}} \boldsymbol{x}_n + b$ with the principle of empirical loss minimization:

$$\min_{\boldsymbol{\eta},b} \sum_{n \in \mathcal{L}} \ell(Y_n, \boldsymbol{\eta}^{\mathrm{T}} \boldsymbol{x}_n + b) + \frac{\gamma}{2} \|\boldsymbol{\eta}\|_2^2, \tag{7.6}$$

where $\ell(\cdot, \cdot)$ is the loss function of the linear classifier.

Though, in [96], each column of W and HT are concatenated together to form node representations, we experimentally find out that, each column of W is informative enough to be taken as the representation of the corresponding node. Therefore, we utilize columns of W as node representations in our framework. Hence, in (7.6), we set $\boldsymbol{x}_n = \boldsymbol{w}_n$.

For mathematical convenience, we can incorporate the intercept term b into $\boldsymbol{\eta}$ by setting $\boldsymbol{x}_n = [\boldsymbol{w}_n^{\mathrm{T}}, 1]^{\mathrm{T}}$. Then (7.6) is accordingly changed into

$$\min_{\boldsymbol{\eta}} \sum_{n \in \mathcal{L}} \ell(Y_n, \boldsymbol{\eta}^{\mathrm{T}} \boldsymbol{x}_n) + \frac{\gamma}{2} \|\boldsymbol{\eta}\|_2^2, \tag{7.7}$$

where $\boldsymbol{\eta}^{\mathrm{T}} \boldsymbol{x}_n = \boldsymbol{\eta}_{1:k}^{\mathrm{T}} \boldsymbol{w}_n + \boldsymbol{\eta}_{k+1}$ and $\boldsymbol{\eta}_{k+1} = b$.

To make node representations learned by TADW discriminative for a specific classifier $h(\cdot)$, instead of solving problem (7.5) and problem (7.7) separately, we try to address these two problems simultaneously by solving

$$\begin{aligned} \min_{W,H,\boldsymbol{\eta}} \frac{1}{2} \sum_{i,j=1}^{|\mathcal{V}|} (M_{ij} - \boldsymbol{w}_i^{\mathrm{T}} H \boldsymbol{t}_j)^2 + \mu \sum_{n \in \mathcal{L}} \ell(Y_n, \boldsymbol{\eta}^{\mathrm{T}} \boldsymbol{x}_n) \\ + \frac{\lambda}{2} \left(\|W\|_F^2 + \|H\|_F^2 + \|\boldsymbol{\eta}\|_2^2 \right), \end{aligned} \tag{7.8}$$

where μ and λ are the parameters that control the trade-off between the minimization of different objectives. To make our algorithm more efficient, we use the half square loss function $\ell(a, b) = \frac{1}{2}(a - b)^2$ for the linear classifier. As node representations are expected to be discriminative, a loose penalty should be imposed on $\|W\|_F^2$. Thus, two multipliers, λ_1 and λ_2, are introduced to control the minimization of $\|H\|_F^2 + \|\boldsymbol{\eta}\|_2^2$ and $\|W\|_F^2$, respectively. Then, we can obtain the final objective function for our algorithm:

$$\begin{aligned} J(W, H, \boldsymbol{\eta}) = \frac{1}{2} \sum_{i,j=1}^{|\mathcal{V}|} \left(M_{ij} - \boldsymbol{w}_i^{\mathrm{T}} H \boldsymbol{t}_j \right)^2 + \frac{\mu}{2} \sum_{n \in \mathcal{L}} \left(Y_n - \boldsymbol{\eta}^{\mathrm{T}} \boldsymbol{x}_n \right)^2 \\ + \frac{\lambda_1}{2} \left(\|H\|_F^2 + \|\boldsymbol{\eta}\|_2^2 \right) + \frac{\lambda_2}{2} \|W\|_F^2. \end{aligned} \tag{7.9}$$

Our task is to solve this optimization problem:

$$\min_{W,H,\eta} J(W, H, \boldsymbol{\eta}). \tag{7.10}$$

7.3.2 Solving the Problem

We now propose to solve the optimization problem (7.10). Though the objective function (7.9) is not convex with respect to W, H and $\boldsymbol{\eta}$, it can be easily proved that when any two of these three variables W, H and $\boldsymbol{\eta}$ are fixed, $J(W, H, \boldsymbol{\eta})$ is convex with respect to the remaining variable. After W, H and $\boldsymbol{\eta}$ are initialized properly, we can optimize $J(W, H, \boldsymbol{\eta})$ alternatively as

$$\boldsymbol{\eta}^{(t)} = \arg\min_{\boldsymbol{\eta}} J(W^{(t-1)}, H^{(t-1)}, \boldsymbol{\eta}); \tag{7.11}$$

$$H^{(t)} = \arg\min_{H} J(W^{(t-1)}, H, \boldsymbol{\eta}^{(t)}); \tag{7.12}$$

$$W^{(t)} = \arg\min_{W} J(W, H^{(t)}, \boldsymbol{\eta}^{(t)}). \tag{7.13}$$

When W and H are fixed, $\boldsymbol{\eta}$ can be updated as follows:

$$\boldsymbol{\eta} = \arg\min_{\boldsymbol{\eta}' \in \mathbb{R}^{k+1}} \frac{\mu}{2} \sum_{n \in \mathcal{L}} \left(Y_n - \boldsymbol{\eta}'^{\mathrm{T}} \boldsymbol{x}_n \right)^2 + \frac{\lambda_1}{2} \|\boldsymbol{\eta}'\|_2^2, \tag{7.14}$$

which is a traditional linear regression problem and can be easily solved by the Conjugate Gradient (CG) algorithm.

To optimize H with W and η fixed, we set $\boldsymbol{\nu} = \boldsymbol{vec}(H)$, $\tilde{\boldsymbol{x}}_{ij} = \boldsymbol{t}_j \otimes \boldsymbol{w}_i$. Then H can be updated by solving the following problem:

$$\boldsymbol{\nu}^* = \arg\min_{\boldsymbol{\nu} \in \mathbb{R}^{k f_t}} f(\boldsymbol{\nu}), \tag{7.15}$$

where

$$f(\boldsymbol{\nu}) = \frac{1}{2} \sum_{i,j=1}^{|\mathcal{V}|} (M_{ij} - \boldsymbol{\nu}^{\mathrm{T}} \tilde{\boldsymbol{x}}_{ij})^2 + \frac{\lambda_1}{2} \|\boldsymbol{\nu}\|_2^2. \tag{7.16}$$

To optimize W with H and η fixed, we set $\boldsymbol{\omega} = \boldsymbol{vec}(W^T)$, $\tilde{\boldsymbol{y}}_{ij} = (H\boldsymbol{t}_j) \otimes \boldsymbol{e}_i$, $\tilde{\boldsymbol{z}}_n = \boldsymbol{\eta}_{1:k} \otimes \boldsymbol{e}_n$, where $\boldsymbol{e}_i$ is the i-th column of the $|\mathcal{V}| \times |\mathcal{V}|$ identity matrix. Then W can be updated by solving the following problem:

$$\boldsymbol{\omega}^* = \arg\min_{\boldsymbol{\omega} \in \mathbb{R}^{k|\mathcal{V}|}} g(\boldsymbol{\omega}), \tag{7.17}$$

Algorithm 3 Discriminative Matrix Factorization

Input:
The PageRank matrix of graph G, A;
The index set of labeled nodes, $\mathcal{L}$;
Class labels of labeled nodes, $Y_{\mathcal{L}}$;
Output:
Matrix W, H and vector $\boldsymbol{\eta}$;
1: $M \leftarrow (P + P^2)/2$; *Construct node-path-matrix* M
2: $(W, H, \boldsymbol{\eta}) \leftarrow$ Random Initialization;
3: **repeat**
4: $\quad \boldsymbol{\eta} \leftarrow$ Updating by solving Eq. (7.14);
5: $\quad H \leftarrow$ Updating by solving Eq. (7.15);
6: $\quad W \leftarrow$ Updating by solving Eq. (7.17);
7: **until** Convergence or a certain # of iterations
8: **return** W, H and $\boldsymbol{\eta}$.

where

$$g(\boldsymbol{\omega}) = \frac{1}{2}\sum_{i,j=1}^{|\mathcal{V}|}(M_{ij} - \boldsymbol{\omega}^{\mathrm{T}}\tilde{\boldsymbol{y}}_{ij})^2 + \frac{\mu}{2}\sum_{n\in\mathcal{L}}(Y_n - \boldsymbol{\eta}_{k+1} - \boldsymbol{\omega}^{\mathrm{T}}\tilde{\boldsymbol{z}}_n)^2 + \frac{\lambda_2}{2}\|\boldsymbol{\omega}\|_2^2. \tag{7.18}$$

The algorithm for solving W, H and $\boldsymbol{\eta}$ is given in Algorithm 3. Firstly, the *node-path-matrix* M is constructed at step 1. Then W, H and $\boldsymbol{\eta}$ are initialized randomly. After that, $\boldsymbol{\eta}$, W and H are updated alternatively and iteratively. The iteration stops until convergence or a certain number of iterations. Finally, W, H, and $\boldsymbol{\eta}$ are returned as the solution. During the iteration, as the features constructed by W are randomly initialized, the learned $\boldsymbol{\eta}$ at step 4 is inaccurate which leads the discriminative information imported at step 6 to be noisy. To overcome this, we gradually increase the value of μ at step 6 from a small value to its final value.

Now we focus on solving problem (7.15) and problem (7.17) using the CG algorithm. For CG algorithm, the gradients of $f(\boldsymbol{\nu})$ and $g(\boldsymbol{\omega})$, and the multiplication of the Hessian matrix of $f(\boldsymbol{\nu})$ and $g(\boldsymbol{\omega})$ with vector $\boldsymbol{s}$ are calculated as follows:

$$\nabla f(\boldsymbol{\nu}) = \sum_{i,j=1}^{|\mathcal{V}|}(\boldsymbol{\nu}^{\mathrm{T}}\tilde{\boldsymbol{x}}_{ij} - M_{ij})\tilde{\boldsymbol{x}}_{ij} + \lambda_1\boldsymbol{\nu}, \tag{7.19}$$

$$\nabla g(\boldsymbol{\omega}) = \sum_{i,j=1}^{|\mathcal{V}|} (\boldsymbol{\omega}^{\mathrm{T}} \tilde{\boldsymbol{y}}_{ij} - M_{ij}) \tilde{\boldsymbol{y}}_{ij} + \mu \sum_{n \in \mathcal{L}} (\boldsymbol{\omega}^{\mathrm{T}} \tilde{\boldsymbol{z}}_n - Y_n + \boldsymbol{\eta}_{k+1}) \tilde{\boldsymbol{z}}_n + \lambda_2 \boldsymbol{\omega}, \tag{7.20}$$

$$\nabla^2 f(\boldsymbol{\nu}) \boldsymbol{s} = \sum_{i,j=1}^{|\mathcal{V}|} \tilde{\boldsymbol{x}}_{ij} \tilde{\boldsymbol{x}}_{ij}^{\mathrm{T}} \boldsymbol{s} + \lambda_1 \boldsymbol{s}, \tag{7.21}$$

$$\nabla^2 g(\boldsymbol{\omega}) \boldsymbol{s} = \sum_{i,j=1}^{|\mathcal{V}|} \tilde{\boldsymbol{y}}_{ij} \tilde{\boldsymbol{y}}_{ij}^{\mathrm{T}} \boldsymbol{s} + \mu \sum_{n \in \mathcal{L}} \tilde{\boldsymbol{z}}_n \tilde{\boldsymbol{z}}_n^{\mathrm{T}} \boldsymbol{s} + \lambda_2 \boldsymbol{s}. \tag{7.22}$$

If we calculate $\nabla f(\boldsymbol{\nu})$ and $\nabla^2 f(\boldsymbol{\nu})\boldsymbol{s}$ using (7.19) and (7.21), the time complexity would be at least $O(|\mathcal{V}|^2 k f_t)$. Accordingly, if we calculate $\nabla g(\boldsymbol{\omega})$ and $\nabla^2 g(\boldsymbol{\omega})\boldsymbol{s}$ using (7.20) and (7.22), it would require at least $O(|\mathcal{V}|^2 k)$ time. This way, the calculation of $\nabla f(\boldsymbol{\nu})$, $\nabla g(\boldsymbol{\omega})$, $\nabla^2 f(\boldsymbol{\nu})\boldsymbol{s}$, and $\nabla^2 g(\boldsymbol{\omega})\boldsymbol{s}$ is computational expensive and cannot scale to large-scale networks. Below, we propose efficient solutions to calculating the four variables, with detailed derivations given in the Appendix.

A. Calculation for $\nabla f(\boldsymbol{\nu})$

In Appendix A, we have shown that

$$\nabla f(\boldsymbol{\nu}) = \boldsymbol{vec}(WW^{\mathrm{T}}HTT^{\mathrm{T}} - WMT^{\mathrm{T}}) + \lambda_1 \boldsymbol{\nu}. \tag{7.23}$$

By changing the sequence of matrix multiplication in (7.23) carefully as

$$\nabla f(\boldsymbol{\nu}) = \boldsymbol{vec}(W((W^{\mathrm{T}}H)(TT^{\mathrm{T}})) - W(MT^{\mathrm{T}})) + \lambda_1 \boldsymbol{\nu}, \tag{7.24}$$

the calculation of $\nabla f(\boldsymbol{\nu})$ requires only $O(|\mathcal{V}|kf_t + |\mathcal{V}|f_t^2 + nnz(M)f_t)$ time. For most real-world networks, edges are relatively sparse, i.e., $O(|\mathcal{E}|) = O(|\mathcal{V}|)$. Thus, matrix M is also very sparse, i.e., $nnz(M) \ll |\mathcal{V}|^2$, and calculating $\nabla f(\boldsymbol{\nu})$ using (7.24) can improve the efficiency.

B. Calculation for $\nabla^2 f(\boldsymbol{\nu})\boldsymbol{s}$

We can prove that as shown in Appendix B

$$\nabla^2 f(\boldsymbol{\nu})\boldsymbol{s} = \boldsymbol{vec}(WW^{\mathrm{T}}STT^{\mathrm{T}}) + \lambda_1 \boldsymbol{s}, \tag{7.25}$$

where $S \in \mathbb{R}^{k \times f_t}$ satisfies $\boldsymbol{s} = \boldsymbol{vec}(S)$. Similarly, we can calculate $\nabla^2 f(\boldsymbol{\nu})\boldsymbol{s}$ as

$$\nabla^2 f(\boldsymbol{\nu})\boldsymbol{s} = \boldsymbol{vec}(W((W^{\mathrm{T}}S)(TT^{\mathrm{T}}))) + \lambda_1 \boldsymbol{s}, \tag{7.26}$$

which requires only $O(|\mathcal{V}|kf_t + |\mathcal{V}|f_t^2)$ time.

C. Calculation for $\nabla g(\boldsymbol{\omega})$

As shown in Appendix C, we prove that

$$\nabla g(\boldsymbol{w}) = \boldsymbol{vec}(W^{\mathrm{T}}HTT^{\mathrm{T}}H^{\mathrm{T}} - MT^{\mathrm{T}}H^{\mathrm{T}}) + \mu \boldsymbol{vec}(I_{|\mathcal{V}|\mathcal{L}}^{\mathrm{T}} E \boldsymbol{\eta}_{1:k}^{\mathrm{T}}) + \lambda_2 \boldsymbol{\omega}, \tag{7.27}$$

where $I_{|\mathcal{V}|\mathcal{L}}$ is the matrix stacked by rows corresponding to the index set $\mathcal{L}$ of the $|\mathcal{V}| \times |\mathcal{V}|$ identity matrix, and E is a $|\mathcal{L}|$-dimensional column vector stacked by $\boldsymbol{\omega}^T \tilde{\boldsymbol{z}}_n - Y_n + \boldsymbol{\eta}_{k+1} = \boldsymbol{\eta}_{1:k}^{\mathrm{T}} \boldsymbol{w}_n - Y_n + \boldsymbol{\eta}_{k+1}$ $(n \in \mathcal{L})$. For implementation efficiency, we take a similar approach to calculate $\nabla g(\boldsymbol{\omega})$ as

$$\nabla g(\boldsymbol{w}) = \boldsymbol{vec}(W^{\mathrm{T}}((HT)(HT)^{\mathrm{T}}) - M(HT)^{\mathrm{T}}) + \mu \boldsymbol{vec}(I_{|\mathcal{V}|\mathcal{L}}^{\mathrm{T}} E \boldsymbol{\eta}_{1:k}^{\mathrm{T}}) + \lambda_2 \boldsymbol{\omega}, \tag{7.28}$$

which requires only $O(|\mathcal{V}|k^2 + |\mathcal{V}|kf_t + nnz(M)k)$ time.

D. Calculation for $\nabla^2 g(\boldsymbol{\omega})\boldsymbol{s}$

It also can be proved in Appendix D that

$$\nabla^2 g(\boldsymbol{\omega})\boldsymbol{s} = \boldsymbol{vec}(SHTT^{\mathrm{T}}H^{\mathrm{T}}) + \mu \boldsymbol{vec}(I_{|\mathcal{V}|\mathcal{L}}^{\mathrm{T}} V \boldsymbol{\eta}_{1:k}^{\mathrm{T}}) + \lambda_2 \boldsymbol{s}, \tag{7.29}$$

where $S \in \mathbb{R}^{|\mathcal{V}|\times k}$ satisfies $\boldsymbol{s} = \boldsymbol{vec}(S)$ and V is a $|\mathcal{L}|$-dimensional column vector stacked by $\boldsymbol{e}_n^{\mathrm{T}} S\boldsymbol{\eta}_{1:k}$ ($n \in \mathcal{L}$). Similarly, $\nabla^2 g(\boldsymbol{\omega})\boldsymbol{s}$ can be calculated as

$$\nabla^2 g(\boldsymbol{\omega})\boldsymbol{s} = \boldsymbol{vec}(S((HT)(HT)^{\mathrm{T}})) + \mu\boldsymbol{vec}(I_{|\mathcal{V}|\mathcal{L}}^{\mathrm{T}} V\boldsymbol{\eta}_{1:k}^{\mathrm{T}}) + \lambda_2\boldsymbol{s}, \tag{7.30}$$

which needs only $O(|\mathcal{V}|k^2 + |\mathcal{V}|kf_t)$ time.

7.3.3 Node Classification Using Learned Representations

After problem (7.10) is solved, we can construct node representations using $W \in \mathbb{R}^{k\times|\mathcal{V}|}$, with each node $v_i \in \mathcal{V}$ being represented by a k-dimensional vector $\boldsymbol{w}_i$. Then, for each unlabeled node $v_n \in \mathcal{V} \setminus \mathcal{V}_{\mathcal{L}}$, $\boldsymbol{x}_n = \left[\boldsymbol{w}_n^{\mathrm{T}}, 1\right]^{\mathrm{T}}$. The class label Y_n of node v_n is determined by

$$Y_n = \begin{cases} +1, & h(\boldsymbol{x}_n) \geq 0, \\ -1, & h(\boldsymbol{x}_n) < 0, \end{cases} \tag{7.31}$$

where $h(\boldsymbol{x}) = \boldsymbol{\eta}^{\mathrm{T}}\boldsymbol{x}$ is the classifier constructed using $\boldsymbol{\eta}$. For an unlabeled node v_n, as its learned representation $\boldsymbol{x}_n$ not only preserves underlying network properties, but also uncovers discriminative power, classification using (7.31) is expected to achieve good classification performance on unlabeled nodes.

7.4 Experiments

In this section, we move forward to evaluate the performance of the proposed algorithm on three real-world networks and compare it against the state-of-the-art baselines.

7.4.1 Benchmark Networks

The three real-world networks used in our experiments are Cora, Citeseer and PubMed[1], as listed in Table 1. For each network, link directions and self-links are ignored. Therefore, two nodes are connected, if either of them has direct link to the other. The details about the three networks are described as follows:

[1]https://linqs.soe.ucsc.edu/data

Table 7.1 Summary of three real-world networks

Data Set	Cora	Citeseer	PubMed
# of Nodes	2,708	3,312	19,717
# of Links	5,429	4,732	44,338
# of Features	1,433	3,703	500
# in Largest Class	818	701	7,739
# in Smallest Class	180	249	4,103

Cora network contains 2,708 machine learning publications from seven classes: Case Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning, Theory. There are 5,249 citation links among these papers. Each paper is represented by a 0/1 vector of 1,433 dimensions indicating the presence/absence of the corresponding words. In our experiments, we construct a binary classification problem by taking the largest class, Neural Networks, as positive class and all other classes as negative class.

Citeseer network contains 3,312 scientific publications from six classes: Databases (DB), Machine Learning (ML), Information Retrieval (IR), Artificial Intelligence (AI), Human Computer Interaction (HCI) and Agents. Among these papers, there are 4,732 citation links. Each paper is represented by a 0/1 vector of 3,703 dimensions indicating the presence/absence of the corresponding words. We construct a binary classification problem by taking the largest class, DB, as positive class and the rest as negative class.

PubMed network contains 19,717 scientific publications from PubMed related to three types of diabetes research: Diabetes Mellitus Experimental, Diabetes Mellitus Type 1, Diabetes Mellitus Type 2. Each paper is represented by a TF-IDF weighted word vector from a dictionary composed by 500 unique words. There are 44,338 links among them. We construct a binary classification problem by taking the largest class, Diabetes Mellitus Type 2, as positive class and other classes as negative class.

7.4.2 Experimental Settings

To test the performance of node classification on sparsely labeled networks, we vary the percentage of labeled nodes from 2% to 20% by an increment of 2%, for Cora and Citeseer. To maintain the number of labeled nodes at the same scale, we vary the percentage of labeled nodes, for PubMed, from 0.2% to 2.0% by an increment of 0.2%. For each setting, we randomly select an initial set of nodes for labeling and repeat this process ten times. The average F_1 value over ten repetitions is reported as our evaluation metric.

In our experiments, the dimensions of node text features are reduced to 200 by applying Singular Value Decomposition (SVD) on TF-IDF matrix. For our DMF algorithm, parameters are set as: $k = 40$, $\mu = 0.04$, $\lambda_1 = 0.2$, and $\lambda_2 = 2 \times 10^{-5}$. The effect of these parameters on classification performance will be empirically studied later in the experiments. The maximum number of iterations is set to 50 (we will show later that DMF can quickly converge within about 10 iterations).

7.4.3 Baseline Methods

We compare the proposed DMF algorithm with the following baseline methods:

ICA: The ICA algorithm [58] is one of the most popularly used baselines for collective classification. We train a logistic regression classifier on labeled nodes using their node content features and relational features calculated as the number of positive neighbors and the number of negative neighbors. A collective inference is then performed with the learned classifier to predict labels of unlabeled nodes.

LNP: LNP [69] is a recently proposed algorithm for performing collective classification in sparsely labeled networks. It builds a latent graph by linearly combining five graphs: k-step graph, label similarity graph, attribute similarity graph, prediction confidence graph, and structure similarity graph. The weights of linear combination are trained using labeled nodes. After the latent graph is constructed, label propagation is performed to classify unlabeled nodes.

Semi-supervised RCI (Semi-RCI): Semi-RCI [52] is the start-of-the-art semi-supervised collective classification algorithm that exploits node content features together with neighbor features and neighbor labels to improve the learning performance of the ICA algorithm. Semi-RCI first bootstraps the labels of unlabeled nodes via the classifier trained on labeled nodes using node features and neighbor features. After that, it trains a hybrid classifier on all nodes by utilizing node features, features of nodes' neighbors, as well as labels of nodes' neighbors, and then collectively classifies unlabeled nodes.

TADW+SVM: This baseline method is based on the most recently proposed TADW algorithm [96] that leverages both nodes' linkage information and content features for network representation learning. We first learn node representations using the TADW algorithm in an unsupervised manner, and then train a linear SVM classifier using learned node representations. Unlabeled nodes are classified via applying the learned SVM classifier on the node representations learned by TADW.

7.4.4 Comparison of Classification Performance

We first carry out experiments to compare classification performance of all algorithms with different labeling rates. Average F_1 values for all algorithms on Cora, Citeseer and PubMed are given in Table 7.2, Table 7.3, and Table 7.4, respectively. For each labeling rate, the best average F_1 value is highlighted in **bold**. Note that, because the size of the PubMed network is too large for the LNP algorithm to run, the experimental results of LNP on PubMed are not reported in Table 7.4. From all three tables, we can see that our proposed DMF algorithm generally outperforms other baseline methods with all labeling rates on all three networks, although Semi-RCI performs slightly better on Cora when the labeling rate is 2%, 4% and 6%.

Apart from the proposed DMF algorithm, for all labeling rates on Cora and 2%-10% labeling rates on Citeseer, Semi-RCI has the best performance. When the labeling rate varies from 12% to 20% on Citeseer, the classification performance of Semi-RCI, LNP and ICA is comparable to each other. Experimental results on Cora and Citeseer indicate that Semi-RCI, ranking only next to DMF, is the second best performing method for node classification in sparsely labeled networks. However, on the larger

Table 7.2 Average F_1 (%) values on Cora

Labeling Rate	2%	4%	6%	8%	10%	12%	14%	16%	18%	20%
ICA	47.83	54.67	56.59	62.51	67.99	66.88	74.08	73.38	75.67	78.60
LNP	45.96	57.68	63.99	68.82	71.61	72.39	76.09	74.90	75.85	77.72
Semi-RCI	**71.32**	**75.96**	**78.73**	79.56	79.37	80.79	82.61	82.53	83.38	83.80
TADW+SVM	0.70	4.58	21.30	39.49	56.84	62.64	72.45	70.32	74.74	76.23
DMF	68.93	73.92	78.16	**80.54**	**80.65**	**82.91**	**83.51**	**83.75**	**84.28**	**84.44**

Table 7.3 Average F_1 (%) values on Citeseer

Labeling Rate	2%	4%	6%	8%	10%	12%	14%	16%	18%	20%
ICA	19.53	39.86	49.94	50.01	57.14	62.40	64.34	67.25	68.93	69.47
LNP	14.60	40.49	51.31	52.46	59.10	65.96	66.38	69.64	71.65	71.86
Semi-RCI	38.48	55.63	59.17	58.78	62.44	64.86	65.09	67.47	68.12	68.38
TADW+SVM	1.60	4.73	10.71	18.17	29.85	42.38	44.68	51.76	58.72	60.42
DMF	**55.19**	**65.36**	**69.09**	**71.18**	**72.50**	**73.09**	**74.32**	**74.72**	**74.93**	**74.49**

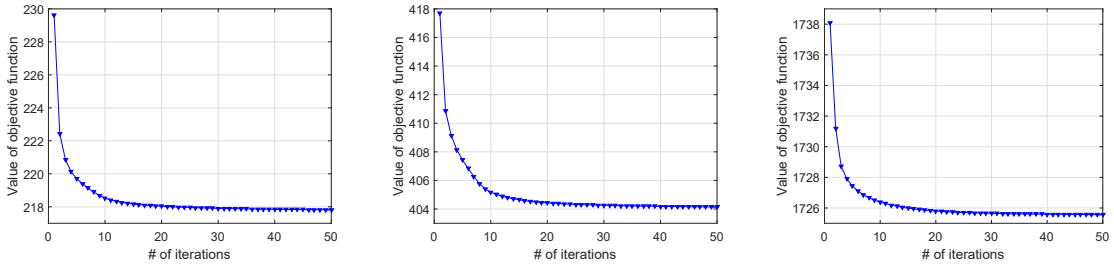
Table 7.4 Average F_1 (%) values on PubMed

(No results of LNP are reported here, because the PubMed network is too large for LNP.)

Labeling Rate	0.2%	0.4%	0.6%	0.8%	1.0%	1.2%	1.4%	1.6%	1.8%	2.0%
ICA	45.61	49.29	36.19	50.97	45.27	35.33	58.11	59.73	58.03	63.04
LNP	–	–	–	–	–	–	–	–	–	–
Semi-RCI	35.25	42.92	42.97	40.34	40.24	45.64	44.81	46.23	42.43	45.34
TADW+SVM	5.71	2.63	7.41	10.90	19.96	37.51	53.13	59.12	57.93	65.58
DMF	**52.96**	**57.04**	**57.13**	**59.33**	**62.35**	**63.79**	**68.11**	**68.66**	**67.31**	**70.42**

PubMed network, Semi-RCI cannot achieve satisfactory classification performance, which performs even worse than ICA. This may be caused by incorrect predictions made on unlabeled nodes at the bootstrap step. These errors propagate through collective inference and deteriorate the performance of Semi-RCI.

According to our analysis, the node representations learned by our DMF algorithm are more discriminative than those learned by TADW. Therefore, DMF is supposed to outperform TADW+SVM. As expected, for all labeling rates on Cora, Citeseer and PubMed, DMF consistently outperforms TADW+SVM to a large margin. For very sparsely labeled networks, the classification performance of TADW+SVM is



(a) Cora (b) Citeseer (c) PubMed

Fig. 7.2 Convergence of the objective function

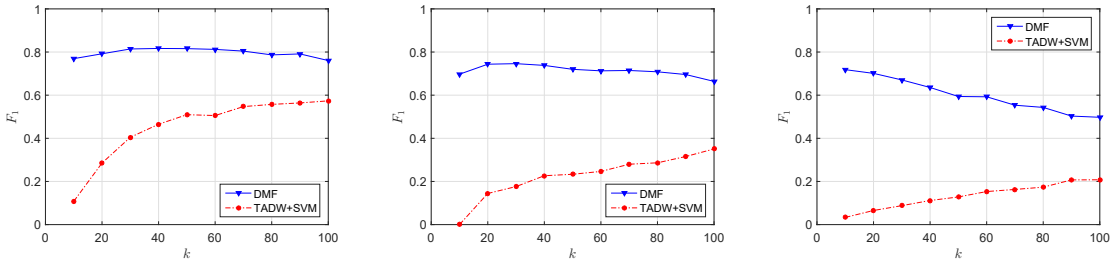
unsatisfactory, which shows that TADW+SVM severely suffers from label sparsity. Comparatively, DMF can still achieve satisfactory classification performance, which shows the effectiveness of DMF in dealing with label sparsity.

7.4.5 Convergence Analysis

In order to validate the convergence of DMF for finding solutions to the optimization problem (7.10), we fix the labeling rate on Cora and Citeseer as 10% and that on PubMed as 1%, and report the values of objective functions from iteration 1 to iteration 50 on all three networks in Figure 7.2. The results show that DMF can achieve fast convergence within about 10 iterations on all three networks.

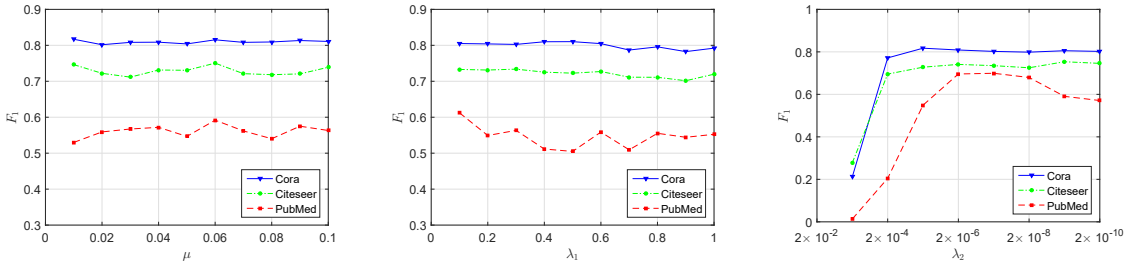
7.4.6 Parameter Sensitivity

An important parameter of the proposed DMF algorithm is the dimensionality k of the learned node representations. The learned representations with different dimensions may exhibit different informative and discriminative abilities. We thus perform experiments to empirically study the effect of different values of k on classification performance. We fix the labeling rate for Cora and Citeseer as 10% and the labeling rate as 1% for PubMed. F_1 values of DMF and TADW+SVM on three networks with different values of k are shown in Figure 7.3. From Figure 7.3, we can find that, on Cora and Citeseer, the proposed DMF algorithm is not sensitive to k, and on PubMed, the F_1 value decreases slightly with an increase of k. What can also be observed from Figure



(a) Cora (b) Citeseer (c) PubMed

Fig. 7.3 F_1 values with respect to different k values

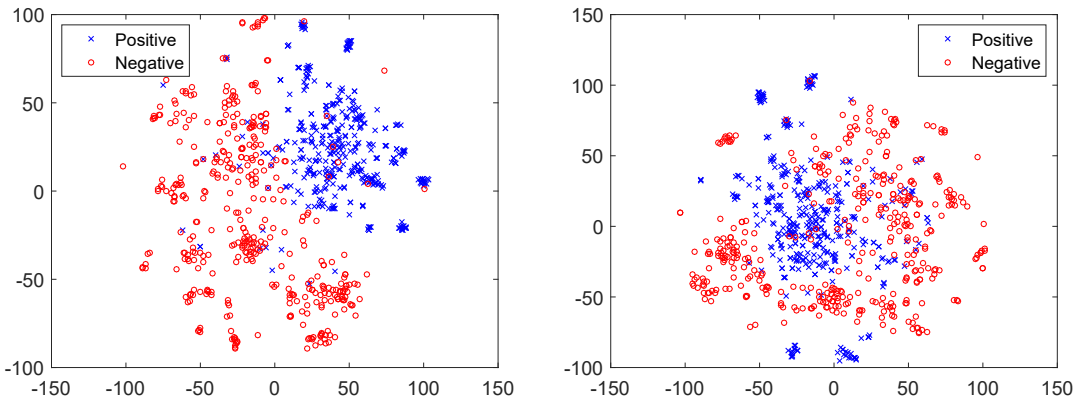


(a) μ (b) λ_1 (c) λ_2

Fig. 7.4 F_1 values with respect to different μ, λ_1 and λ_2 values

7.3 is that the F_1 value of TADW+SVM rises gradually with an increase of k. This indicates that, using less feature dimensions, the proposed DMF algorithm can still yield better classification performance.

We also carry out experiments using different μ, λ_1 and λ_2 values to investigate their impact on classification performance. In this set of experiments, we use the default setting on Cora, Citeseer with 10% labeling rate and on PubMed with 1% labeling rate. We take turns to fix two parameters out of the three and vary the third parameter to study its effect on classification performance. Figure 7.4(a), Figure 7.4(b), and Figure 7.4(c) show F_1 values in regard to different values of μ, λ_1 and λ_2, respectively. From Figure 7.4(a) and Figure 7.4(b), we can see that, on Cora and Citeseer, DMF is not sensitive to different values of μ and λ_1, while the classification performance of DMF slightly fluctuates with different values of μ and λ_1 on PubMed. From Figure 7.4(c), we can observe that, on Cora and Citeseer, DMF achieves a larger slope of improvement when λ_2 varies from 2×10^{-3} to 2×10^{-4}, and its performance starts to stabilize after λ_2 increases to 2×10^{-5}. On PubMed, there is a similar trend, while the performance



(a) DMF (b) TADW

Fig. 7.5 Visualization of learned representations

of DMF becomes stable at a later stage when $\lambda_2 = 2 \times 10^{-6}$ and starts to decrease when $\lambda_2 = 2 \times 10^{-8}$.

7.4.7 Visualization of Learned Representations

To validate whether the node representations learned by the proposed DMF algorithm are indeed more discriminative than those learned by TADW, we project the node representations learned by DMF and TADW on Cora with 20% labeling rate into two-dimensional space using the t-SNE package [83]. We randomly select 400 positive and 400 negative, and plot the visualization results in Figure 7.5. The results in Figure 7.5 show that different classes of data points generated by DMF are indeed better separated from each other, compared to the results generated by TADW.

7.5 Conclusion

In this chapter, we proposed a new discriminative matrix factorization (DMF) framework for classifying sparsely labeled networks. We argued that when the provided network is sparsely labeled, unlabeled nodes may have too few labeled neighbors, or have very low structure similarity in local proximity to labeled nodes. Hence, existing semi-supervised and connectivity-based algorithms cannot produce satisfactory perfor-

mance. Instead, we resorted to exploiting random walk based path dependencies to augment network structure, and learning a latent network representation that fully encodes nodes' content information and augmented network structure to minimize classification loss. By incorporating node content, augmented network structure, together with sparsely labeled nodes, to form a new objective function, DMF is able to learn a latent vector representation for each node, such that a classifier learned from the new vector representations has the minimum loss for classifying nodes in the whole network. Experiments on real-world networks demonstrated clear performance gain, compared to the state-of-the-art algorithms.

7.6 APPENDIX

A. Calculation for $\nabla f(\boldsymbol{\nu})$

As $\tilde{\boldsymbol{x}}_{ij} = \boldsymbol{t}_j \otimes \boldsymbol{w}_i = \boldsymbol{vec}(\boldsymbol{w}_i \boldsymbol{t}_j^{\mathrm{T}})$, we have

$$\sum_{i,j=1}^{|\mathcal{V}|} (\boldsymbol{\nu}^{\mathrm{T}} \tilde{\boldsymbol{x}}_{ij} - M_{ij}) \boldsymbol{w}_i \boldsymbol{t}_j^{\mathrm{T}} = WFT^{\mathrm{T}} = WW^{\mathrm{T}} HTT^{\mathrm{T}} - WMT^{\mathrm{T}}, \tag{7.32}$$

where $F_{ij} = \boldsymbol{\nu}^{\mathrm{T}} \tilde{\boldsymbol{x}}_{ij} - M_{ij}$, i.e., $F = W^{\mathrm{T}} HT - M$. Thus,

$$\nabla f(\boldsymbol{\nu}) = \boldsymbol{vec}(WW^{\mathrm{T}} HTT^{\mathrm{T}} - WMT^{\mathrm{T}}) + \lambda_1 \boldsymbol{\nu}. \tag{7.33}$$

B. Calculation for $\nabla^2 f(\boldsymbol{\nu})\boldsymbol{s}$

In (7.21), after substituting $\tilde{\boldsymbol{x}}_{ij} = \boldsymbol{t}_j \otimes \boldsymbol{w}_i$, we have

$$\sum_{i,j=1}^{|\mathcal{V}|} \tilde{\boldsymbol{x}}_{ij} \tilde{\boldsymbol{x}}_{ij}^{\mathrm{T}} \boldsymbol{s} = \sum_{i,j=1}^{|\mathcal{V}|} \left[(\boldsymbol{t}_j \boldsymbol{t}_j^{\mathrm{T}}) \otimes (\boldsymbol{w}_i \boldsymbol{w}_i^{\mathrm{T}}) \right] \boldsymbol{s}. \tag{7.34}$$

Let $S \in \mathbb{R}^{k \times f_t}$ such that $\boldsymbol{s} = \boldsymbol{vec}(S)$. Using the identity

$$(B^{\mathrm{T}} \otimes A)\boldsymbol{vec}(X) = \boldsymbol{vec}(AXB), \tag{7.35}$$

we have $\left[(\boldsymbol{t}_j\boldsymbol{t}_j^{\mathrm{T}}) \otimes (\boldsymbol{w}_i\boldsymbol{w}_i^{\mathrm{T}})\right] \boldsymbol{s} = \boldsymbol{vec}(\boldsymbol{w}_i\boldsymbol{w}_i^{\mathrm{T}} S \boldsymbol{t}_j\boldsymbol{t}_j^{\mathrm{T}})$. Thus,

$$\begin{aligned}\sum_{i,j=1}^{|\mathcal{V}|} \boldsymbol{w}_i\boldsymbol{w}_i^{\mathrm{T}} S \boldsymbol{t}_j\boldsymbol{t}_j^{\mathrm{T}} &= \sum_{i=1}^{|\mathcal{V}|} \boldsymbol{w}_i \left(\sum_{j=1}^{|\mathcal{V}|} \boldsymbol{w}_i^{\mathrm{T}} S \boldsymbol{t}_j\boldsymbol{t}_j^{T}\right) \\ &= \sum_{i=1}^{|\mathcal{V}|} \boldsymbol{w}_i \left(\sum_{j=1}^{|\mathcal{V}|} U_{ij}\boldsymbol{t}_j^{\mathrm{T}}\right) = WUT^{\mathrm{T}},\end{aligned} \tag{7.36}$$

where $U_{ij} = \boldsymbol{w}_i^{\mathrm{T}} S \boldsymbol{t}_j$, i.e., $U = W^{\mathrm{T}}ST$. Hence,

$$\sum_{i,j=1}^{|\mathcal{V}|} \tilde{\boldsymbol{x}}_{ij}\tilde{\boldsymbol{x}}_{ij}^{\mathrm{T}}\boldsymbol{s} = \boldsymbol{vec}(WUT^{\mathrm{T}}) = \boldsymbol{vec}(WW^{\mathrm{T}}STT^{\mathrm{T}}). \tag{7.37}$$

Thus,

$$\nabla^2 f(\boldsymbol{\nu})\boldsymbol{s} = \boldsymbol{vec}(WW^{\mathrm{T}}STT^{\mathrm{T}}) + \lambda_1\boldsymbol{s}. \tag{7.38}$$

C. Calculation for $\nabla g(\boldsymbol{\omega})$

As $\tilde{\boldsymbol{y}}_{ij} = (H\boldsymbol{t}_j) \otimes \boldsymbol{e}_i = \boldsymbol{vec}(\boldsymbol{e}_i\boldsymbol{t}_j^{\mathrm{T}}H^{\mathrm{T}})$, we have

$$\begin{aligned}\sum_{i,j=1}^{|\mathcal{V}|} (\boldsymbol{\omega}^{\mathrm{T}}\tilde{\boldsymbol{y}}_{ij} - M_{ij})\boldsymbol{e}_i\boldsymbol{t}_j^{\mathrm{T}}H^{\mathrm{T}} &= FT^{\mathrm{T}}H^{\mathrm{T}} \\ &= W^{\mathrm{T}}HTT^{\mathrm{T}}H^{\mathrm{T}} - MT^{\mathrm{T}}H^{\mathrm{T}}.\end{aligned} \tag{7.39}$$

As $\tilde{\boldsymbol{z}}_n = \boldsymbol{\eta}_{1:k} \otimes \boldsymbol{e}_n = \boldsymbol{vec}(\boldsymbol{e}_n\boldsymbol{\eta}_{1:k}^{\mathrm{T}})$, we have

$$\sum_{n\in\mathcal{L}} (\boldsymbol{\omega}^{\mathrm{T}}\tilde{\boldsymbol{z}}_n - Y_n + \boldsymbol{\eta}_{k+1})\boldsymbol{e}_n\boldsymbol{\eta}_{1:k}^{\mathrm{T}} = I_{|\mathcal{V}|\mathcal{L}}^{\mathrm{T}} E \boldsymbol{\eta}_{1:k}^{\mathrm{T}}. \tag{7.40}$$

Thus,

$$\nabla g(\boldsymbol{w}) = \boldsymbol{vec}(W^{\mathrm{T}}HTT^{\mathrm{T}}H^{\mathrm{T}} - MT^{\mathrm{T}}H^{\mathrm{T}}) + \mu\boldsymbol{vec}(I_{|\mathcal{V}|\mathcal{L}}^{\mathrm{T}} E \boldsymbol{\eta}_{1:k}^{\mathrm{T}}) + \lambda_2\boldsymbol{\omega}. \tag{7.41}$$

D. Calculation for $\nabla^2 g(\boldsymbol{\omega})\boldsymbol{s}$

Similar to (7.37),

$$\sum_{i,j=1}^{|\mathcal{V}|} \tilde{\boldsymbol{y}}_{ij}\tilde{\boldsymbol{y}}_{ij}^{\mathrm{T}}\boldsymbol{s} = \boldsymbol{vec}(SHTT^{\mathrm{T}}H^{\mathrm{T}}), \tag{7.42}$$

where $S \in \mathbb{R}^{|\mathcal{V}|\times k}$ satisfies $\boldsymbol{s} = \boldsymbol{vec}(S)$. As $\tilde{\boldsymbol{z}}_n = \boldsymbol{\eta}_{1:k} \otimes \boldsymbol{e}_n$,

$$\begin{aligned}\sum_{n\in\mathcal{L}} \tilde{\boldsymbol{z}}_n\tilde{\boldsymbol{z}}_n^{\mathrm{T}}\boldsymbol{s} &= \sum_{n\in\mathcal{L}} \left[(\boldsymbol{\eta}_{1:k}\boldsymbol{\eta}_{1:k}^{T}) \otimes (\boldsymbol{e}_n\boldsymbol{e}_n^{T})\right] \boldsymbol{s} \\ &= \boldsymbol{vec}(\sum_{n\in\mathcal{L}} \boldsymbol{e}_n\boldsymbol{e}_n^{\mathrm{T}}S\boldsymbol{\eta}_{1:k}\boldsymbol{\eta}_{1:k}^{\mathrm{T}}).\end{aligned} \tag{7.43}$$

As

$$\begin{aligned}\sum_{n\in\mathcal{L}} \boldsymbol{e}_n\boldsymbol{e}_n^{\mathrm{T}}S\boldsymbol{\eta}_{1:k}\boldsymbol{\eta}_{1:k}^{\mathrm{T}} &= \sum_{n\in\mathcal{L}} \boldsymbol{e}_n(\boldsymbol{e}_n^{\mathrm{T}}S\boldsymbol{\eta}_{1:k})\boldsymbol{\eta}_{1:k}^{\mathrm{T}} \\ &= I_{|\mathcal{V}|\mathcal{L}}^{\mathrm{T}}V\boldsymbol{\eta}_{1:k}^{\mathrm{T}},\end{aligned} \tag{7.44}$$

$$\nabla^2 g(\boldsymbol{\omega})\boldsymbol{s} = \boldsymbol{vec}(SHTT^{\mathrm{T}}H^{\mathrm{T}}) + \mu\boldsymbol{vec}(I_{|\mathcal{V}|\mathcal{L}}^{\mathrm{T}}V\boldsymbol{\eta}_{1:k}^{\mathrm{T}}) + \lambda_2\boldsymbol{s}. \tag{7.45}$$

Chapter 8

Binary Attributed Network Embedding for Efficient Search

8.1 Introduction

Networks offer a natural way to capture intrinsic relationships between entities – social interactions among people, collaboration between co-workers, biological interactions among proteins, flow-of-funds between financial transactions, and so on. Networks can be modeled as a graph, where nodes indicate entities and edges indicate pairwise relationships between two nodes. Searching similar nodes in networks is an essential network analytic task, which directly benefits many real-world applications. For the social security, potential terrorists can be detected by searching people with the same organization associations in the communication networks or online social networks. On e-commerce platforms, personalized recommendations can be effectively delivered by searching users with similar interests among users' social relations. In social networks, social actors with important structural roles, such as the center of a start or hole spanner, can be discovered by searching nodes with the same properties among the whole network. Searching similar nodes can also benefits other tasks, such as Web page retrieval in the World Wide Web [23], link prediction in social networks [72], and identity resolution in the bibliographic collaboration network [108].

To enable similarity search over networks, conventional methods usually leverage the structural properties of a network including common neighbors and structural context to estimate the similarity between nodes. Representative algorithms include Personalized PageRank [23], SimRank [28], P-Rank [110], TopSim [38], and Panther [108]. However, these methods suffer from two drawbacks:

- **High computational cost**. Most existing structure based search algorithms have an at least quadratic time complexity with respect to the number of nodes. For example, SimRank [28] has a complexity of $O(|\mathcal{V}|^2\bar{d}^2)$ to find the top-K nodes for all nodes, where $|\mathcal{V}|$ is the number of nodes in a network, and $\bar{d}$ is the average degree of all nodes. This computational overhead makes algorithms difficult to scale up to large-scale networks with millions or billions of nodes.
- **Incapable of capturing node content similarity**. In addition to network structure, network nodes are often associated with rich content, such as user profiles in social networks, texts in Web page networks. Node content contains crucial information that provides direct evidence to measure node similarity. The structure based similarity search methods fail to leverage the similarity measured by node content, leading to suboptimal search results.

Recently, network embedding [63, 76, 96, 104] has been proposed to facilitate network analytic tasks, which aims to embed network nodes into a low-dimensional continuous vector space, by preserving network structure and/or node content information. After learning new node representations, network analytic tasks can be easily carried out by applying off-the-shelf machine learning algorithms to the new embedding space. However, such a machine learning driven network embedding paradigm often results in node representations that are inefficient for large-scale similarity search in terms of both time and memory. Consider a network with 10 million nodes, if we learn 200-dimensional dense vector-format node representations for each node, it requires 15G memory to accommodate these representations using standard double precision numbers, which is prohibitively intractable for general computing devices. Given a query node, if we want to find its similar nodes among the whole network, it requires to calculate the Euclidean distance between the query node and all other nodes in the network, which requires 2 billion times of floating-point product operation and 2

billion times of floating-point addition operation. The high computational cost makes it unsuitable for real-time retrieval systems that require responsive solutions. In summary, searching similar nodes with continuous node representations inevitably incurs high time and memory cost, resulting in unsatisfactory performance on large-scale networks.

As an alternative way of exact nearest neighbor search in Euclidean space, hashing techniques [12] have been proposed to improve search efficiency. Hashing techniques aim to transform the numeric data in the Euclidean space into binary codes by preserving the similarity in the original space. As a consequence, data can be stored with low memory cost and similarity search can be conducted efficiently by calculating the Hamming distance between node binary codes with bitwise operations. Borrowing the idea of hashing, we propose to learn binary representations for network nodes, *i.e.*, transforming network nodes into binary codes rather than numeric vectors, such that the memory and time efficiency for similarity search can be significantly improved. Despite its potential, the binary node representation learning is confronted with the following two challenges:

- **Heterogeneity**. To guarantee search accuracy, binary node representations are expected to capture the information from both network structure and node content, i.e., preserving node similarity in both structure level and node content level. However, network structure and node content are not always consistent to or correlated with each other. How to fuse information from these two heterogeneous sources into binary codes and make them complement rather than deteriorate each other is a big challenge.
- **Scalability**. With the objective of preserving node structure level and content level proximity, discovering the optimal binary node representations is NP-hard. When it comes to large-scale networks with millions or billions of nodes/edges, and high dimensional node content features, it is impossible to find the exact optimal solutions in an efficient way. To make the learning highly scalable, in the promise of assuring the quality of solutions, approximation techniques together with online or parallel learning strategies need to be developed.

An intuitive solution to binary network embedding is to first learn continuous node representations and then binarize them into binary codes with the conventional hashing

techniques. However, because converting continuous embeddings into binary codes inevitably causes information loss, the learned binary codes cannot accurately capture node similarity on both structure and content level. As a result, as demonstrated later in our experiments, this two-step learning strategy usually results in unsatisfactory search accuracy.

In this chapter, we propose a novel Binary Network Embedding algorithm, called BinaryNE, to learn binary node representations directly from both network structure and node content features for efficient similarity search. BinaryNE learns binary node representations by simultaneously modeling node context and node attribute relations through a three-layer neural network, with the objective of capturing node similarity in both network structure and node content. To obtain binary codes, the *sign* function $\text{sgn}(\cdot)$ is employed as activation function in the hidden layer. However, as the gradient of the *sign* function is zero almost everywhere, traditional gradient decent based optimization strategies are infeasible for learning parameters, which is known as the *ill-posed gradient* problem. To address this problem, we adopt the state-of-the-art continuation technique [3, 10] and develop an online stochastic gradient descent algorithm to learn parameters, which guarantees the great scalability of BinaryNE. Experiments on six real-world networks show that BinaryNE exhibits much lower memory usage and quicker search speed than the state-of-the-art network embedding methods, while still achieving competitive search accuracy.

The main contribution of this chapter is threefold:

- We analyze the feasibility and advantage of learning binary node representations as a solution to efficient node similarity search over large-scale networks.
- We propose the BinaryNE algorithm to effectively learn high-quality binary node representations from both network structure and node content features.
- We conduct experiments to compare search performance of the BinaryNE algorithm and other network embedding algorithms, showing the superiority of BinaryNE in terms of memory usage and search time.

8.2 Problem Definition and Preliminaries

In this section, we give a formal definition of the binary network embedding problem, followed by a review on the preliminaries of DeepWalk.

8.2.1 Problem Definition

Given a network $G = (\mathcal{V}, \mathcal{E}, \mathcal{A}, X)$, where $\mathcal{V}$ is the set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, and $\mathcal{A}$ is the set of attributes. $X \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{A}|}$ is the node feature matrix, with each element $X_{ij} \geq 0$ indicating the occurrence times/weights of attribute $a_j \in \mathcal{A}$ at node $v_i \in \mathcal{V}$. For networks taking continuous attribute values, discretization can be used to convert continuous values to the categorical ones.

The BinaryNE algorithm aims to learn binary representations for network nodes, i.e., learning a mapping function $\Phi : v_i \in \mathcal{V} \mapsto \{+1, -1\}^d$, where d is the dimension of the embedding space. The learned binary node representations $\Phi(v_i)$ are expected to satisfy the following two properties: (1) **low-dimensional**, the dimension d should be much smaller than the dimension of node adjacent matrix representation $|\mathcal{V}|$, for the sake of search efficiency; (2) **informative**, to guarantee the quality of node similarity search, the learned binary node representations should capture node similarity measured by both network structure and node content features.

8.2.2 Preliminaries: DeepWalk

Borrowing the idea of Skip-Gram model [55], which learns word representations by preserving context similarity, DeepWalk leverages random walks to generate node context and represents nodes sharing similar context closely in the new embedding space. Given a random walk with length L, $\{v_{r_1}, v_{r_2}, \cdots, v_{r_i}, \cdots v_{r_L}\}$, for each node v_{r_i}, DeepWalk learns its representation by using it to predict its context nodes, which is realized by maximizing the occurrence probability of context nodes conditioned on this node:

$$\min_{\Phi} -\log \mathrm{P}(\{v_{r_{i-t}}, \cdots, v_{r_{i+t}}\} \setminus v_{r_i} | v_{r_i}), \tag{8.1}$$

where $\{v_{r_{i-t}}, \cdots, v_{r_{i+t}}\} \setminus v_{r_i}$ are the context nodes of v_{r_i} within t window size.

Using the conditional independence assumption, the probability $\mathrm{P}(\{v_{r_{i-t}}, \cdots, v_{r_{i+t}}\} \setminus v_{r_i} | v_{r_i})$ can be calculated as

$$\mathrm{P}(\{v_{r_{i-t}}, \cdots, v_{r_{i+t}}\} \setminus v_{r_i} | v_{r_i}) = \prod_{j=i-t, j \neq i}^{i+t} \mathrm{P}(v_{r_j} | v_{r_i}). \tag{8.2}$$

Following [102], after a set of random walks are generated, we can formulate the overall optimization problem as

$$\min_{\Phi} -\sum_{i=1}^{|\mathcal{V}|} \sum_{j=1}^{|\mathcal{V}|} n(v_i, v_j) \log \mathrm{P}(v_j | v_i), \tag{8.3}$$

where $n(v_i, v_j)$ is the occurrence time of node context pair (v_i, v_j) collected from all random walks with t window size and $\mathrm{P}(v_j|v_i)$ is modeled by softmax:

$$\mathrm{P}(v_j | v_i) = \frac{\exp(\Phi(v_i) \cdot \Psi(v_j))}{\sum_{k=1}^{|\mathcal{V}|} \exp(\Phi(v_i) \cdot \Psi(v_k))}.$$

The overall optimization problem can be solved by iteratively sampling a node context pair (v_i, v_j) and minimizing the following partial objective:

$$\mathcal{O}_{ij}^{s} = -\log \mathrm{P}(v_j | v_i). \tag{8.4}$$

8.3 Binary Network Embedding

This section details the optimization problem that we formulate for the binary network embedding, followed by the solution on how to solve it efficiently.

8.3.1 The Optimization Problem

Our objective is to learn informative binary network embeddings, with both network structure and node content features well preserved. The idea of DeepWalk can be borrowed here for capturing network structure. To capture node content level similarity, we try to represent nodes sharing similar attributes closely in the low-dimensional

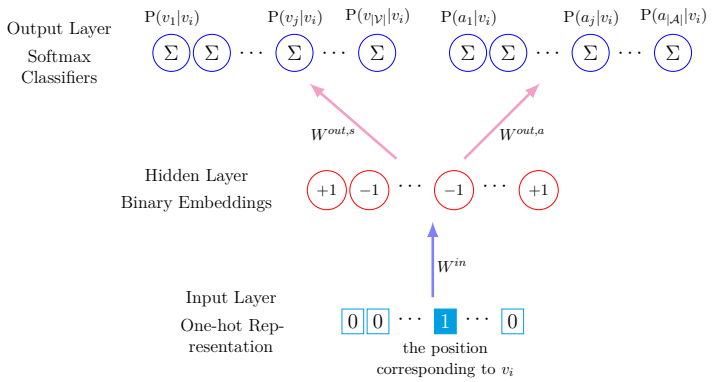


Fig. 8.1 The model architecture of BinaryNE. For each node v_i, BinaryNE learns its binary representation by using it to predict its context node v_j and its attribute a_j.

space. To achieve this goal, we apply the idea of Skip-Gram [55] again, by using each node to predict its content attributes. For each node attribute co-occurrence pair (v_i, a_j), we minimize the following objective:

$$\mathcal{O}_{ij}^{a} = -\log \mathrm{P}(a_j|v_i). \tag{8.5}$$

We illustrate the architecture of the proposed BinaryNE algorithm in Fig. 8.1, which is a three-layer neural network: the first layer is the one-hot representation for each node v_i, the hidden layer is the binary node representation $\Phi(v_i) \in \{+1, -1\}^d$ constructed from the input layer, and the output layer is the softmax conditional probability $\mathrm{P}(v_j|v_i)$ and $\mathrm{P}(a_j|v_i)$ for each context node v_j and each attribute a_j, modeled through node binary representations in the hidden layer.

Given node v_i's one-hot representation $\boldsymbol{p}^i \in \mathbb{R}^{|\mathcal{V}|}$ with $\boldsymbol{p}_k^i = 1$ for $k = i$, and $\boldsymbol{p}_k^i = 0$ for $k \neq i$. The binary node representation $\Phi(v_i)$ in the hidden layer is constructed by performing a linear transformation on $\boldsymbol{p}^i$ and activating with the *sign* function:

$$\begin{aligned} \Phi(v_i) &= \left[\mathrm{sgn}(\boldsymbol{p}^i \cdot W_{:1}^{in}), \mathrm{sgn}(\boldsymbol{p}^i \cdot W_{:2}^{in}), \cdots, \mathrm{sgn}(\boldsymbol{p}^i \cdot W_{:d}^{in})\right]^{\mathrm{T}} \\ &= \left[\mathrm{sgn}(W_{i1}^{in}), \mathrm{sgn}(W_{i2}^{in}), \cdots, \mathrm{sgn}(W_{id}^{in})\right]^{\mathrm{T}}, \end{aligned} \tag{8.6}$$

where $W^{in}_{:k}$ is the k-th column of $W^{in} \in \mathbb{R}^{|\mathcal{V}| \times d}$ (the weight matrix from the input layer to the hidden layer) and $\text{sgn}(\cdot)$ is the *sign* function, which is defined as

$$\text{sgn}(x) = \begin{cases} +1, \text{ if } x \geqslant 0, \\ -1, \text{ otherwise.} \end{cases}$$

In the output layer, for the node context pair (v_i, v_j), we model the probability $\text{P}(v_j|v_i)$ with softmax:

$$\text{P}(v_j|v_i) = \frac{\exp(\Phi(v_i) \cdot W^{out,s}_{:j})}{\sum_{k=1}^{|\mathcal{V}|} \exp(\Phi(v_i) \cdot W^{out,s}_{:k})},$$

where $W^{out,s}_{:j}$ is the j-th column of $W^{out,s} \in \mathbb{R}^{d \times |\mathcal{V}|}$ (the weight matrix from the hidden layer to the output layer for predicting node context). Similarly, for the node attribute co-occurrence pair (v_i, a_j), we model the probability $\text{P}(a_j|v_i)$ as

$$\text{P}(a_j|v_i) = \frac{\exp(\Phi(v_i) \cdot W^{out,a}_{:j})}{\sum_{k=1}^{|\mathcal{A}|} \exp(\Phi(v_i) \cdot W^{out,a}_{:k})},$$

where $W^{out,a}_{:j}$ is the j-th column of $W^{out,a} \in \mathbb{R}^{d \times |\mathcal{A}|}$ (the weight matrix from the hidden layer to the output layer for predicting node attribute).

To learn informative binary node embeddings, we integrate the structure proximity preserving objective in Eq. (8.4) with the node attribute similarity preserving objective in Eq. (8.5), and obtain the following overall optimization problem:

$$\min_{\Phi} \mathcal{O}, \tag{8.7}$$

where

$$\mathcal{O} = -\alpha_1 \sum_{i=1}^{|\mathcal{V}|} \sum_{j=1}^{|\mathcal{V}|} n(v_i, v_j) \text{log}\text{P}(v_j|v_i) - \alpha_2 \sum_{i=1}^{|\mathcal{V}|} \sum_{j=1}^{|\mathcal{A}|} X_{ij} \text{log}\text{P}(a_j|v_i). \tag{8.8}$$

Here, α_1 and α_2 are the trade-off parameters to balance the contribution of structure preserving objective and node content preserving objective. They are specified as

$$\alpha_1 = \frac{1}{\sum_{i=1}^{|\mathcal{V}|} \sum_{j=1}^{|\mathcal{V}|} n(v_i, v_j)}, \quad \alpha_2 = \frac{1}{\sum_{i=1}^{|\mathcal{V}|} \sum_{j=1}^{|\mathcal{A}|} X_{ij}}.$$

In Eq. (8.8), only the non-zero entries of $n(v_i, v_j)$ and X_{ij} are considered, whose number is much smaller than $|\mathcal{V}| \times |\mathcal{V}|$ and $|\mathcal{V}| \times |\mathcal{A}|$, respectively.

8.3.2 Solving the Optimization Problem

As the derivative of the *sign* function used to construct binary codes is zero almost everywhere, solving the optimization problem (8.7) with gradient descent is *ill-posed*. Following [10], we approximate the non-smooth *sign* function $\text{sgn}(x)$ with its smooth proxy $\tanh(\beta x)$, which satisfies the following property:

$$\lim_{\beta \to \infty} \tanh(\beta x) = \text{sgn}(x).$$

Thus, in Eq. (8.6), node representation $\Phi(v_i)$ is constructed as

$$\Phi(v_i) = \left[\tanh(\beta W_{i1}^{in}), \tanh(\beta W_{i2}^{in}), \cdots, \tanh(\beta W_{id}^{in})\right]^{\mathrm{T}}. \tag{8.9}$$

With this continuous approximation, we can solve the optimization problem (8.7) with stochastic gradient descent. At each iteration, we randomly select a node context pair (v_i, v_j) according to the distribution of $n(v_i, v_j)$ or a node attribute co-occurrence pair (v_i, a_j) according to the distribution of X_{ij}, and then update parameters towards minimizing the corresponding partial objective $\mathcal{O}_{ij}^s$ in Eq. (8.4) or $\mathcal{O}_{ij}^a$ in Eq. (8.5).

Given a sampled node context pair (v_i, v_j), for training efficiency, we adopt negative sampling [21] to approximate the partial objective $\mathcal{O}_{ij}^s$ in Eq. (8.4) as

$$\mathcal{O}_{ij}^s = -\log \sigma(\Phi(v_i) \cdot W_{:j}^{out,s}) - \sum_{k:v_k \in \mathcal{V}_{neg}} \log \sigma(-\Phi(v_i) \cdot W_{:k}^{out,s}), \tag{8.10}$$

where $\mathcal{V}_{neg}$ is the set of sampled negative nodes and $\sigma(\cdot)$ is the sigmoid function. Then we update the parameters with gradient descent:

$$\begin{aligned} W_{i:}^{in} &= W_{i:}^{in} - \eta \frac{\partial \mathcal{O}_{ij}^{s}}{\partial W_{i:}^{in}}, \\ W_{:j}^{out,s} &= W_{:j}^{out,s} - \eta \frac{\partial \mathcal{O}_{ij}^{s}}{\partial W_{:j}^{out,s}}, \\ W_{:k}^{out,s} &= W_{:k}^{out,s} - \eta \frac{\partial \mathcal{O}_{ij}^{s}}{\partial W_{:k}^{out,s}}, \text{ for } v_k \in \mathcal{V}_{neg}, \end{aligned} \tag{8.11}$$

where η is the learning rate. The gradients are calculated as

$$\begin{aligned} \frac{\partial \mathcal{O}_{ij}^{s}}{\partial W_{ir}^{in}} &= \beta[1 - \tanh(\beta W_{ir}^{in})^2][\sigma(\Phi(v_i) \cdot W_{:j}^{out,s}) - 1]W_{rj}^{out,s} \\ &\quad + \beta[1 - \tanh(\beta W_{ir}^{in})^2] \sum_{k:v_k \in \mathcal{V}_{neg}} \sigma(\Phi(v_i) \cdot W_{:k}^{out,s}) W_{rk}^{out,s}, \\ \frac{\partial \mathcal{O}_{ij}^{s}}{W_{:j}^{out,s}} &= [\sigma(\Phi(v_i) \cdot W_{:j}^{out,s}) - 1]\Phi(v_i), \\ \frac{\partial \mathcal{O}_{ij}^{s}}{W_{:k}^{out,s}} &= \sigma(\Phi(v_i) \cdot W_{:k}^{out,s})\Phi(v_i), \text{ for } v_k \in \mathcal{V}_{neg}. \end{aligned}$$

Similarly, after a node attribute co-occurrence pair (v_i, a_j) is sampled, with negative sampling [21], the partial objective $\mathcal{O}_{ij}^{a}$ in Eq. (8.5) is approximated as

$$\mathcal{O}_{ij}^{a} = -\log \sigma(\Phi(v_i) \cdot W_{:j}^{out,a}) - \sum_{k:v_k \in \mathcal{A}_{neg}} \log \sigma(-\Phi(v_i) \cdot W_{:k}^{out,a}), \tag{8.12}$$

where $\mathcal{A}_{neg}$ is the set of sampled negative attributes. We then update the parameters with gradient descent

$$\begin{aligned} W_{i:}^{in} &= W_{i:}^{in} - \eta \frac{\partial \mathcal{O}_{ij}^{a}}{\partial W_{i:}^{in}}, \\ W_{:j}^{out,a} &= W_{:j}^{out,a} - \eta \frac{\partial \mathcal{O}_{ij}^{a}}{\partial W_{:j}^{out,a}}, \\ W_{:k}^{out,a} &= W_{:k}^{out,a} - \eta \frac{\partial \mathcal{O}_{ij}^{a}}{\partial W_{:k}^{out,a}}, \text{ for } a_k \in \mathcal{A}_{neg}. \end{aligned} \tag{8.13}$$

Algorithm 4 BinaryNE: Binary Network Embedding

Input:
A given network $G = (\mathcal{V}, \mathcal{E}, \mathcal{A}, X)$;
Output:
Binary node embedding $\Phi(\cdot)$ for each $v_i \in \mathcal{V}$;
$\mathbb{S} \leftarrow$ generate a set of random walks on G;
$n(v_i, v_j) \leftarrow$ count the frequency of node context pairs (v_i, v_j) in $\mathbb{S}$;
$(W^{in}, W^{out,s}, W^{out,a}) \leftarrow$ initialization;
repeat
  draw a random number $\delta \in (0, 1)$;
  if $\delta \leqslant 0.5$ **then**
    $(v_i, v_j) \leftarrow$ sample a node context pair according to the distribution of $n(v_i, v_j)$;
    $\mathcal{V}_{neg} \leftarrow$ draw K negative nodes;
    $(W^{in}, W^{out,s}) \leftarrow$ update parameters with $(v_i, v_j, \mathcal{V}_{neg})$ and Eq. (8.11);
  else
    $(v_i, a_j) \leftarrow$ sample a node attribute pair according to the distribution of X_{ij};
    $\mathcal{A}_{neg} \leftarrow$ draw K negative attributes;
    $(W^{in}, W^{out,a}) \leftarrow$ update parameters with $(v_i, a_j, \mathcal{A}_{neg})$ and Eq. (8.13);
  end if
until maximum number of iterations expire;
construct node embedding $\Phi(\cdot)$ with W^{in} and Eq. (8.14);
return $\Phi(\cdot)$;

The gradients are calculated as

$$\begin{aligned}
\frac{\partial \mathcal{O}_{ij}^{a}}{\partial W_{ir}^{in}} &= \beta[1 - \tanh(\beta W_{ir}^{in})^2][\sigma(\Phi(v_i) \cdot W_{:j}^{out,a}) - 1]W_{rj}^{out,a} \\
&\quad + \beta[1 - \tanh(\beta W_{ir}^{in})^2] \sum_{k:v_k \in \mathcal{A}_{neg}} \sigma(\Phi(v_i) \cdot W_{:k}^{out,a}) W_{rk}^{out,a}, \\
\frac{\partial \mathcal{O}_{ij}^{a}}{W_{:j}^{out,a}} &= [\sigma(\Phi(v_i) \cdot W_{:j}^{out,a}) - 1]\Phi(v_i), \\
\frac{\partial \mathcal{O}_{ij}^{a}}{W_{:k}^{out,a}} &= \sigma(\Phi(v_i) \cdot W_{:k}^{out,a})\Phi(v_i), \text{ for } a_k \in \mathcal{A}_{neg}.
\end{aligned}$$

After the parameters are learned, for node $v_i \in \mathcal{V}$, we construct its embedding $\Phi(v_i)$ as

$$\Phi(v_i)_r = \begin{cases} +1, & \text{if } \tanh(\beta W_{ir}^{in}) \geqslant 0, \\ -1, & \text{if } \tanh(\beta W_{ir}^{in}) < 0. \end{cases} \tag{8.14}$$

To obtain binary codes for efficient Hamming distance calculation, we store the -1 value of $\Phi(v_i)_r$ as 0.

Algorithm 4 provides the pseudocode of the proposed BinaryNE algorithm. At Step 1, a set of random walks with length L are generated by starting random walks at each node $v_i \in \mathcal{V}$ for γ times. At Step 2, on the generated random walks, with t window size, BinaryNE collects node context pairs (v_i, v_j) and counts their occurrence frequencies $n(v_i, v_j)$. At Step 3, W^{in} is initialized with random numbers, and $W^{out,s}$ and $W^{out,a}$ are initialized with zero. At Step 4-15, the parameters are updated with stochastic gradient descent. Each iteration starts from drawing a random switch variable $\delta \in (0, 1)$ to determine which partial objective to be optimized. To optimize the structure preserving partial objective, BinaryNE randomly draws a node context pair (v_i, v_j) according to the distribution of $n(v_i, v_j)$, and draws K negative nodes, forming $\mathcal{V}_{neg}$, then updates the parameters with Eq. (8.11). To optimize the attribute preserving objective, BinaryNE draws a node attribute co-occurrence pair (v_i, a_j) according the distribution of X_{ij} and draws a negative attribute set $\mathcal{A}_{neg}$ with size K, then updates the parameters with Eq. (8.13). For efficient node context pair and node attribute pair sampling, BinaryNE adopts the alias table [40] method, which takes only $O(1)$ time at each sampling. Finally, BinaryNE constructs binary node representations $\Phi(\cdot)$ with W^{in} and Eq. (8.14).

The time complexity of BinaryNE is determined by only the dimension of node embeddings d and the maximum number of iterations. The scale of the maximum number of iterations is $O(\max(nnz(X), |\mathcal{V}|))$, where $nnz(X)$ is the number of non-zero entries of X, and $|\mathcal{V}|$ is the scale of node context pairs collected via random walks. BinaryNE has a time complexity of $O(d \cdot \max(nnz(X), |\mathcal{V}|))$, which guarantees its ability to scale up to large-scale graphs.

8.4 Experiments

In this section, we conduct experiments on six real-world networks to evaluate the performance of binary node representations learned by BinaryNE for node similarity search, including search precision, response time and memory usage.

Table 8.1 Summary of six real-world networks

	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	$\|\mathcal{A}\|$	$nnz(X)$	# of Class
Cora	2,708	5,278	1,433	49,216	7
Citeseer	3,312	4,732	3,703	105,165	6
BlogCatalog	5,196	171,743	8,189	369,435	6
Flickr	7,575	239,738	12,047	182,517	9
DBLP(Subgraph)	18,448	45,611	5,959	108,016	4
DBLP(Full)	1,632,442	2,327,450	154,309	10,413,178	N/A

8.4.1 Datasets

Six real-world networks are used in the experiments, with the details as follows:

- **Cora**[1]. The Cora network is composed of 2,708 machine learning publications and their citation relationships. Theses publications are categorized into seven groups. Each publication is represented by a 1,433-dimensional binary vector, with each dimension denoting the presence/absence of the corresponding word.
- **Citeseer**[1]. Citeseer is another citation network with 3,312 papers and 4,732 citation relations. There are 6 classes among papers. According to the occurrence of the corresponding word, each paper is described by a 3,703-dimensional binary vector.
- **BlogCatalog**[2]. The BlogCatalog network is an online social network formed by BlogCatalog, a blogger community. The BlogCatalog network contains 5,196 users and 171,743 follower-followee relations. Users' groups are defined as the categories of their blogs. The keywords of users' blogs are used to construct users' feature vectors. Here, binary feature vectors are constructed, with only the keyword occurrence state concerned.
- **Flickr**[2]. Flickr is an online photo sharing platform. The Flickr network includes 7,575 users and 239,738 follower-followee relations. These users join in

[1]https://linqs.soe.ucsc.edu/data
[2]http://people.tamu.edu/ xhuang/Code.html

9 predefined groups. Users' features are described by the tags of their images. Each user is represented by 12,047-dimensional binary vector, according to the occurrence/absence of the corresponding tag.

- **DBLP(Subgraph)** and **DBLP(Full)**. The DBLP(Full) network is formed by the papers, paper titles, and paper citations of the DBLP bibliographic network[3]. In DBLP(Full), there are in total 1,632,442 papers and 2,327,450 citations. The DBLP(Subgraph) is a subgraph of the DBLP(Full) network, constructed by papers from four research areas: *Database, Data Mining, Artificial Intelligence, and Computer Version*, which also act as paper labels. DBLP(Subgraph) contains 18,448 papers and 45,611 citation relations. For DBLP(Full) and DBLP(Subgraph), papers' titles are used to construct binary bag-of-words feature vectors.

For each network, the direction of links is ignored. **Cora**, **Citeseer**, **BlogCatalog**, **Flickr**, and **DBLP(Subgraph)** are used to evaluate the performance of the binary node representations learned by BinaryNE on node similarity search, including search precision, query time and memory usage. **DBLP(Full)** is used to investigate the scalability of node similarity search with BinaryNE binary codes.

8.4.2 Baseline Methods

BinaryNE is compared with two groups of state-of-the-art methods, which are measured by two metrics (Euclidean and Hamming distance) respectively:

- Continuous embeddings measured by Euclidean distance:
  - **DeepWalk/node2vec** [63, 19] preserves the similarity between nodes sharing similar context in random walks. node2vec is equivalent to DeepWalk with the default parameter setting $p = q = 1$.
  - **LINE1** [76] denotes the version of LINE that captures the first-order proximity.
  - **LINE2** [76] represents the version of LINE that models the second-order proximity.

[3]https://aminer.org/citation (Version 3 is used)

  - **SDNE** [87] learns deep non-linear node representations via a semi-supervised deep autoencoder.
  - **TADW** [96] learns node embeddings that capture both network structure and node content similarity via inductive matrix factorization [57].
  - **UPP-SNE** [102] performs a non-linear mapping on node content features to learn node embeddings that preserve both network structure and node content features.
  - **MVC-DNE** [97] fuses network structure and node content features into node embeddings through deep cross-view learning.
  - **SINE** [105] learns node representations by using node representations to simultaneously predict context nodes and node content attributes.
  - **Feature**. Node raw content feature is also used as a baseline for similarity search. For each node $v_i \in \mathcal{V}$, its feature vector is $X_{i:}$, with $X_{i:}$ being the i-th row of X.

- Discrete embeddings measured by Hamming distance:
  - **Quantized Continuous Embeddings**. To obtain binary node representations, a naive way is to quantize the continuous node embeddings into binary codes. As a baseline, we binarize the continuous embeddings learned by above baseline methods with Spectral Hashing [92], and denote these methods as DeepWalk+Q, LINE1+Q, LINE2+Q, SDNE+Q, TADW+Q, UPP-SNE+Q, MVC-DNE+Q, SINE+Q, and Feature+Q.
  - **NetHash** [93]. It is the state-of-the-art discrete attributed network embedding method. Each dimension of the NetHash embeddings is randomly selected from the content feature ID set aggregated from neighborhood. As the learned discrete node representations do not take binary values, the Hamming distance cannot be efficiently calculated with bitwise operations.

8.4.3 Experimental Settings

For all methods, we set the dimension of embeddings $d = 128$. For DeepWalk, UPP-SNE, SINE, and BinaryNE, we set the length of random walks $L = 100$, the number of random walks starting from per node $\gamma = 40$, and the window size $t = 10$.

For fair comparisons, we use the same strategy to train DeepWalk, UPP-SNE, SINE, and BinaryNE: we first collect node context pairs from the generated random walks, and update parameters with stochastic gradient descent by sampling node context pairs. For DeepWalk, LINE, UPP-SNE, SINE and BinaryNE, we set the maximum number of iterations to 100 million for Cora and Citeseer, 200 million for BlogCatalog, Flickr and DBLP(Subgraph). For DeepWalk and BinaryNE, we set the maximum number of iterations to 1 billion for DBLP. For DeepWalk, LINE, UPP-SNE, SINE and BinaryNE, we gradually decrease the learning rate η from 0.025 to 2.5×10^{-6}.

For SDNE, its hyperparameters α and ν are set to 0.01, and β is set to 10, and the number of neurons at each layer is set to 2708-512-128, 3312-512-128, 5,196-512-128, 7,575-512-128 and 18,448-512-128 for Cora, Citeseer, BlogCatalog, Flickr and DBLP(Subgraph) respectively. For MVC-DNE, on Cora, Citeseer, BlogCatalog, Flickr and DBLP(Subgraph), the number of neurons at each layer in the structure view is respectively set to 2708-512-64, 3312-512-64, 5,196-512-64, 7,575-512-64 and 18,448-512-64, and the number of neurons at each layer in the node content feature view is respectively set to 1,433-512-64, 3,703-512-64, 8,189-512-64, 12,047-512-64 and 5,959-512-64. For SDNE and MVC-DNE, 500 epochs are respectively used for pre-training and parameter fine-tuning. We set other parameters of SDNE and MVC-DNE according to [97].

As the content feature dimension of BlogCatalog, Flickr and DBLP(Subgraph) is too large for TADW, before running TADW on them, we reduce the dimension of their node content features to 200 with SVD. Default settings are used to train NetHash. For BinaryNE, we gradually increase the parameter β from 0.01 to 1.

8.4.4 Evaluation Metrics

For each node in a network, we in turn query its top-K similar nodes. K is set to 100, 200, and 500, respectively. We adopt averaged *precision* and *MAP (Mean Averaged Precision)* as evaluation metrics.

For querying nodes similar to node v_i, the $precision@K(v_i)$ is defined as

$$precision@K(v_i) = \frac{|\{v_j|rank(v_j) \leq K, C(v_i) = C(v_j)\}|}{K},$$

where $rank(v_j)$ is the position of v_j in the rank list of nodes similar to v_i, and $C(v_i) = C(v_i)$ indicates that node v_i and v_j have the same class label, with $C(\cdot)$ used to denote node class label. As we in turn take all nodes in $\mathcal{V}$ as query nodes, we report the averaged $precision@K$ as final results.

MAP (Mean Average Precision) is an information retrieval metric with good discrimination and stability. Different from precision, MAP takes into account the order in which relevant nodes are placed in the returned similar node rank list. When we vary the query node v_i over $\mathcal{V}$, the MAP value is calculated as

$$AP@K(v_i) = \frac{\sum_{k=1}^{K} precision@k(v_i) \cdot relavant@k(v_i)}{|\{v_j|C(v_j) = C(v_i), v_j \in \mathcal{V}\}|},$$
$$MAP@K = \frac{\sum_{i=1}^{|\mathcal{V}|} AP@K(v_i)}{|\mathcal{V}|},$$

where $relavant@k(v_i)$ is an indicator function equaling 1 if the k-th retrieved node is relevant and 0 otherwise.

8.4.5 Similarity Search Results

Tables 8.2-8.6 give similarity search results on Cora, Citeseer, BlogCatalog, Flickr and DBLP(Subgraph). For query time, we only consider the time consumed by calculating the distance between the query node and all remainder nodes, which contributes to the main computational overhead for similarity search, and report the time averaged over all query nodes (in milliseconds). We also provide the search speedup of BinaryNE

Table 8.2 Similarity search results on Cora

Metric	Method	precision@100	MAP@100	precision@200	MAP@200	precision@500	MAP@500	Query time (ms)	Speedup
Euclidean	DeepWalk	**0.6288**	**0.1325**	**0.5555**	**0.2170**	0.4016	**0.3291**	1.61	23.0 ×
	LINE1	0.3966	0.0664	0.2980	0.0811	0.2233	0.1084	1.62	23.1 ×
	LINE2	0.3424	0.0479	0.2874	0.0643	0.2415	0.0993	1.67	23.9 ×
	SDNE	0.2956	0.0366	0.2562	0.0498	0.2239	0.0802	1.87	26.7 ×
	TADW	0.2204	0.0157	0.2078	0.0250	0.1944	0.0490	1.65	23.6 ×
	UPP-SNE	0.6098	0.1228	0.5314	0.1938	**0.4049**	0.3032	1.62	23.1 ×
	MVD-DNE	0.3641	0.0425	0.3257	0.0656	0.2780	0.1143	1.84	26.3 ×
	SINE	0.4389	0.0717	0.3745	0.1041	0.3014	0.1642	1.88	26.9 ×
	Feature	0.2240	0.0166	0.2060	0.0252	0.2189	0.0551	17.57	251.0 ×
Hamming	DeepWalk+Q	0.4043	0.0685	0.3236	0.0907	0.2494	0.1293	0.06	0.9 ×
	LINE1+Q	0.3672	0.0599	0.2901	0.0753	0.2298	0.1060	0.06	0.9 ×
	LINE2+Q	0.2854	0.0331	0.2446	0.0446	0.2108	0.0713	0.07	1.0 ×
	SDNE+Q	0.2525	0.0222	0.2311	0.0333	0.2106	0.0607	0.06	0.9 ×
	TADW+Q	0.1914	0.0101	0.1883	0.0177	0.1860	0.0394	0.06	0.9 ×
	UPP-SNE+Q	0.3277	0.0382	0.2852	0.0551	0.2424	0.0914	0.07	1.0 ×
	MVC-DNE+Q	0.2638	0.0204	0.2461	0.0332	0.2246	0.0648	0.07	1.0 ×
	SINE+Q	0.2900	0.0283	0.2593	0.0422	0.2274	0.0740	0.07	1.0 ×
	Feature+Q	0.2605	0.0210	0.2398	0.0329	0.2177	0.0620	0.07	1.0 ×
	NetHash	0.4546	0.0757	0.3852	0.1097	0.2993	0.1656	1.17	16.7 ×
	BinaryNE	**0.5828**	**0.1089**	**0.5210**	**0.1767**	**0.4165**	**0.2963**	0.07	

Table 8.3 Similarity search results on Citeseer

Metric	Method	precision@100	MAP@100	precision@200	MAP@200	precision@500	MAP@500	Query time (ms)	Speedup
Euclidean	DeepWalk	0.4233	0.0508	0.3843	0.0861	0.3158	0.1543	1.98	24.8 ×
	LINE1	0.2878	0.0282	0.2369	0.0373	0.1994	0.0572	2.03	25.4 ×
	LINE2	0.2548	0.0216	0.2197	0.0294	0.1956	0.0483	1.98	24.8 ×
	SDNE	0.2256	0.0155	0.2065	0.0227	0.1927	0.0417	2.39	29.9 ×
	TADW	0.2003	0.0103	0.1910	0.0166	0.1833	0.0339	1.99	24.9 ×
	UPP-SNE	**0.4973**	**0.0594**	**0.4552**	**0.1015**	**0.3794**	**0.1861**	1.95	24.4 ×
	MVC-DNE	0.3471	0.0286	0.3173	0.0463	0.2780	0.0864	2.23	27.9 ×
	SINE	0.3728	0.0381	0.3366	0.0612	0.2852	0.1070	2.37	29.6 ×
	Feature	0.2532	0.0140	0.2471	0.0249	0.2320	0.0530	55.43	692.9 ×
Hamming	DeepWalk+Q	0.3094	0.0302	0.2600	0.0416	0.2185	0.0658	0.08	1.0 ×
	LINE1+Q	0.2984	0.0287	0.2570	0.0407	0.2213	0.0668	0.08	1.0 ×
	LINE2+Q	0.2303	0.0157	0.2087	0.0229	0.1921	0.0415	0.08	1.0 ×
	SDNE+Q	0.2045	0.0112	0.1950	0.0178	0.1871	0.0359	0.08	1.0 ×
	TADW+Q	0.1865	0.0085	0.1848	0.0148	0.1813	0.0316	0.06	0.8 ×
	UPP-SNE+Q	0.3324	0.0322	0.2800	0.0448	0.2329	0.0715	0.08	1.0 ×
	MVC-DNE+Q	0.2562	0.0151	0.2417	0.0252	0.2233	0.0508	0.08	1.0 ×
	SINE+Q	0.2544	0.0157	0.2374	0.0252	0.2178	0.0493	0.08	1.0 ×
	Feature+Q	0.2692	0.0175	0.2472	0.0279	0.2217	0.0528	0.09	1.1 ×
	NetHash	0.3866	0.0378	0.3417	0.0583	0.2851	0.0999	1.35	16.9 ×
	BinaryNE	**0.5013**	**0.0608**	**0.4626**	**0.1055**	**0.3905**	**0.1964**	0.08	

compared with baselines. For continuous and discrete embeddings, the best and second best performer is highlighted by **bold** and underline, respectively.

From Tables 8.2-8.6, we can see that BinaryNE consistently achieves the best precision and MAP among discrete embedding methods with significant advantage over the second best performers, and provides comparable or better search results than continuous embedding methods. This is attributed to BinaryNE's ability to effectively

Table 8.4 Similarity search results on BlogCatalog

Metric	Method	precision@100	MAP@100	precision@200	MAP@200	precision@500	MAP@500	Query time (ms)	Speedup
	DeepWalk	0.4393	0.0330	0.3863	0.0530	0.3041	0.0873	3.68	30.7 ×
	LINE1	0.3844	0.0275	0.3160	0.0411	0.2376	0.0616	3.59	29.9 ×
	LINE2	0.2395	0.0138	0.2228	0.0225	0.2024	0.0417	3.60	30.0 ×
	SDNE	0.3077	0.0173	0.2780	0.0275	0.2430	0.0504	3.59	29.9 ×
Euclidean	TADW	**0.7865**	**0.0814**	**0.7528**	**0.1526**	**0.6710**	**0.3245**	3.61	30.1 ×
	UPP-SNE	0.4902	0.0380	0.4480	0.0641	0.3712	0.1151	3.61	30.1 ×
	MVC-DNE	0.5842	0.0498	0.5201	0.0818	0.4252	0.1458	3.57	29.8 ×
	SINE	0.3508	0.0217	0.3065	0.0327	0.2593	0.0567	3.61	30.1 ×
	Feature	0.2424	0.0113	0.2239	0.0177	0.2023	0.0333	190.77	1589.8 ×
	DeepWalk+Q	0.2829	0.0132	0.2576	0.0210	0.2279	0.0392	0.13	1.1 ×
	LINE1+Q	0.2888	0.0152	0.2531	0.0225	0.2170	0.0385	0.13	1.1 ×
	LINE2+Q	0.2182	0.0072	0.2098	0.0124	0.1996	0.0264	0.13	1.1 ×
	SDNE+Q	0.2484	0.0097	0.2315	0.0160	0.2120	0.0318	0.13	1.1 ×
	TADW+Q	0.5798	0.0506	0.5092	0.0817	0.4047	0.1396	0.13	1.1 ×
Hamming	UPP-SNE+Q	0.3429	0.0204	0.3013	0.0309	0.2553	0.0534	0.13	1.1 ×
	MVC-DNE+Q	0.4153	0.0276	0.3620	0.0425	0.2989	0.0733	0.13	1.1 ×
	SINE+Q	0.2790	0.0126	0.2583	0.0203	0.2337	0.0392	0.13	1.1 ×
	Feature+Q	0.2313	0.0093	0.2163	0.0150	0.2003	0.0295	0.11	0.9 ×
	NetHash	0.3811	0.0246	0.3388	0.0385	0.2882	0.0684	3.14	26.2 ×
	BinaryNE	**0.7297**	**0.0721**	**0.6896**	**0.1320**	**0.6112**	**0.2760**	0.12	

Table 8.5 Similarity search results on Flickr

Metric	Method	precision@100	MAP@100	precision@200	MAP@200	precision@500	MAP@500	Query time (ms)	Speedup
	DeepWalk	0.2029	0.0097	0.1913	0.0162	0.1734	0.0299	5.19	28.8 ×
	LINE1	0.2313	0.0125	0.2068	0.0197	0.1745	0.0332	5.11	28.4 ×
	LINE2	0.1576	0.0074	0.1468	0.0122	0.1358	0.0240	5.34	29.7 ×
	SDNE	0.1532	0.0060	0.1461	0.0103	0.1381	0.0212	5.25	29.2 ×
Euclidean	TADW	0.3287	0.0227	0.2918	0.0364	0.2367	0.0614	4.58	25.4 ×
	UPP-SNE	**0.3792**	**0.0286**	**0.3517**	**0.0482**	**0.3104**	**0.0905**	5.19	28.8 ×
	MVC-DNE	0.3364	0.0208	0.3013	0.0332	0.2536	0.0590	5.06	28.1 ×
	SINE	0.3056	0.0183	0.2605	0.0268	0.2087	0.0431	5.24	29.1 ×
	Feature	0.1379	0.0055	0.1275	0.0082	0.1190	0.0152	410.90	2282.8 ×
	DeepWalk+Q	0.1683	0.0052	0.1575	0.0083	0.1449	0.0161	0.19	1.1 ×
	LINE1+Q	0.1698	0.0053	0.1571	0.0084	0.1430	0.0158	0.20	1.1 ×
	LINE2+Q	0.1627	0.0049	0.1556	0.0082	0.1459	0.0166	0.19	1.1 ×
	SDNE+Q	0.1666	0.0051	0.1573	0.0083	0.1461	0.0164	0.19	1.1 ×
	TADW+Q	0.2084	0.0094	0.1900	0.0147	0.1680	0.0265	0.15	0.8 ×
Hamming	UPP-SNE+Q	0.2849	0.0166	0.2466	0.0243	0.2036	0.0396	0.19	1.1 ×
	MVC-DNE+Q	0.2582	0.0129	0.2287	0.0196	0.1942	0.0341	0.18	1.0 ×
	SINE+Q	0.2188	0.0102	0.1933	0.0147	0.1674	0.0248	0.20	1.1 ×
	Feature+Q	0.1809	0.0084	0.1602	0.0119	0.1402	0.0199	0.16	0.9 ×
	NetHash	0.2035	0.0090	0.1814	0.0133	0.1594	0.0232	4.28	23.8 ×
	BinaryNE	**0.5862**	**0.0552**	**0.5380**	**0.0957**	**0.4544**	**0.1817**	0.18	

encode both network structure and node content features into node binary codes, which are informative enough to measure node similarity accurately.

On the other hand, BinaryNE remarkably improves search efficiency, providing more than 23 times faster search speed than continuous network embedding methods, and more than 15 times than NetHash, which is the second best performer among discrete embedding methods in terms of precision and MAP on Cora, Citeseer, and DBLP(Subgraph). Compared with the Euclidean distance in the continuous embedding

Table 8.6 Similarity search results on DBLP(Subgraph)

Metric	Method	precision@100	MAP@100	precision@200	MAP@200	precision@500	MAP@500	Query time (ms)	Speedup
Euclidean	DeepWalk	0.7121	0.0113	0.6958	0.0214	0.6676	0.0490	13.14	29.2 ×
	LINE1	0.6950	0.0109	0.6562	0.0198	0.5503	0.0374	13.63	30.3 ×
	LINE2	0.6035	0.0085	0.5372	0.0137	0.4514	0.0242	12.92	28.7 ×
	SDNE	0.3963	0.0041	0.3664	0.0066	0.3390	0.0137	12.90	28.7 ×
	TADW	0.6643	0.0096	0.6277	0.0171	0.5575	0.0337	12.86	28.6 ×
	UPP-SNE	**0.7443**	**0.0122**	**0.7294**	**0.0232**	**0.6982**	**0.0527**	13.44	29.9 ×
	MVC-DNE	0.5331	0.0061	0.5072	0.0107	0.4743	0.0228	13.12	29.2 ×
	SINE	0.7365	0.0118	0.7130	0.0220	0.6714	0.0481	13.18	29.3 ×
	Feature	0.5066	0.0052	0.4806	0.0092	0.4370	0.0191	606.69	1348.2 ×
Hamming	DeepWalk+Q	0.6522	0.0097	0.5910	0.0159	0.5071	0.0290	0.46	1.0 ×
	LINE1+Q	0.6595	0.0097	0.6086	0.0165	0.5354	0.0316	0.44	1.0 ×
	LINE2+Q	0.5166	0.0065	0.4619	0.0101	0.4089	0.0186	0.46	1.0 ×
	SDNE+Q	0.3834	0.0034	0.3672	0.0058	0.3542	0.0126	0.45	1.0 ×
	TADW+Q	0.5370	0.0064	0.4968	0.0106	0.4518	0.0213	0.42	0.9 ×
	UPP-SNE+Q	0.5409	0.0066	0.5049	0.0111	0.4590	0.0219	0.48	1.1 ×
	MVC-DNE+Q	0.4320	0.0037	0.4195	0.0066	0.4031	0.0148	0.48	1.1 ×
	SINE+Q	0.5681	0.0071	0.5257	0.0119	0.4755	0.0234	0.47	1.0 ×
	Feature+Q	0.4793	0.0051	0.4471	0.0087	0.4047	0.0170	0.43	1.0 ×
	NetHash	0.6606	0.0097	0.6242	0.0171	0.5750	0.0357	7.31	16.2 ×
	BinaryNE	**0.7558**	**0.0125**	**0.7426**	**0.0241**	**0.7176**	**0.0559**	0.45	

space and the Hamming distance in the non-binary discrete embedding space, the Hamming distance measured by binary representations can be calculated far more efficiently with the bitwise operations.

Among the continuous network embedding baselines, on Citeseer, BlogCatalog, Flickr, and DBLP(Subgraph), the attributed network embedding (TADW or UPP-SNE) achieves the best search precisions and on Cora, the attributed network embedding method UPP-SNE performs comparably to DeepWalk, the best performer. On the five networks, raw node content features consistently fail to achieve satisfactory precisions. By integrating network structure and node content in measuring node similarity, attributed network embedding is superior to structure preserving network embedding and node raw content features.

When the continuous network embeddings are quantized to binary values for efficient search with Hamming distance, except for raw node features, all embeddings experience a search precision drop, which is dramatic in many cases. This is consistent to our expectation that quantized binary codes are inevitably less informative than their original continuous representations. The results demonstrate that it is suboptimal to separately learn continuous network embeddings and quantize them into binary codes. In comparison, BinaryNE directly encodes network structure and node content features into binary node representations, achieving superior search precisions.

NetHash constructs node discrete representations by randomly sampling the IDs of node content features aggregated from neighborhood. With both network structure and node content features leveraged, NetHash achieves the second best search precisions among the discrete network embedding methods on Cora, Citeseer, and DBLP(Subgraph). As the discrete embeddings do not take binary values, bitwise operations cannot be performed to calculate Hamming distance. As a result, its query speedup over continuous network embedding is quite limited.

8.4.6 A Case Study on Relevant Paper Search

In this subsection, we conduct a case study on relevant paper search on the DBLP citation network. We select the paper "Learning Classifiers from Only Positive and Unlabeled Data" published on KDD-2008 as the query paper, which is a highly cited paper on the topic of "Positive Unlabeled Learning". We retrieve the top-5 similar papers with the node representations learned by DeepWalk+Q, Feature+Q, TADW+Q, NetHash and BinaryNE, by calculating the Hamming distance between the query paper and candidate papers. Table 8.7 reports the search results. As can be seen, DeepWalk+Q, Feature+Q, TADW+Q and NetHash only retrieve one relevant paper, and no relevant papers are discovered by Feature+Q. By contrast, the proposed BinaryNE algorithm achieves the best search results, with two relevant papers (1 and 5) discovered.

8.4.7 Comparison of Memory Usage

In Table 8.8, we compare the memory used for accommodating the continuous node representations learned by DeepWalk, the non-binary discrete node representations learned by NetHash, and the binary codes learned by BinaryNE. Compared with DeepWalk and NetHash, with the same dimension, the binary representations learned by BinaryNE significantly reduce the memory consumption by 64 and 32 times, respectively. For the DBLP network with more than 1 million nodes, the memory used for storing the continuous node representations is more than 1.5G, which is intractable for computing devices with low memory configuration to perform node similarity search.

Table 8.7 Top-5 relevant paper search on DBLP

Query: **Learning Classifiers from Only Positive and Unlabeled Data**
DeepWalk+Q:
1. Finding Transport Proteins in a General Protein Database
2. A Bayesian Network Framework for Reject Inference
3. Making Generative Classifiers Robust to Selection Bias
4. Building Text Classifiers Using **Positive and Unlabeled** Examples ✓
5. Audience Selection for On-line Brand Advertising: Privacy-friendly Social Network Targeting
Feature+Q:
1. Learning Coordination Classifiers
2. Learning from Little: Comparison of Classifiers Given Little Training
3. Learning a Two-stage SVM/CRF Sequence Classifier
4. Delegating Classifiers
5. On the Chance Accuracies of Large Collections of Classifiers
TADW+Q:
1. Efficient Learning of Naive Bayes Classifiers under Class-conditional Classification Noise
2. Learning to Classify Texts Using **Positive and Unlabeled** Data ✓
3. Semi-Supervised Learning with Very Few Labeled Training Examples
4. Calculation of the Learning Curve of Bayes Optimal Classification Algorithm for Learning a Perceptron With Noise
5. How To Use What You Know
NetHash:
1. Making Generative Classifiers Robust to Selection bias
2. A Bayesian Network Framework for Reject Inference
3. Building Text Classifiers Using **Positive and Unlabeled** Examples ✓
4. Finding Transport Proteins in a General Protein Database
5. Active Learning in Partially Supervised Classification
BinaryNE:
1. Learning to Classify Texts Using **Positive and Unlabeled** Data ✓
2. Learning the Common Structure of Data
3. Enhancing Supervised Learning with Unlabeled Data
4. Learning from Multiple Sources
5. Text Classification from **Positive and Unlabeled** Documents ✓

Table 8.8 The memory usage of DeepWalk, NetHash and BinaryNE embeddings

Dataset	DeepWalk		NetHash		BinaryNE
	Memory	Reduction	Memory	Reduction	Memory
Cora	2.64M	64×	1.32M	32×	42.32K
Citeseer	3.23M	64×	1.62M	32×	51.75K
BlogCatalog	5.07M	64×	2.54M	32×	81.19K
Flickr	7.40M	64×	3.70M	32×	118.36K
DBLP(Subgraph)	18.02 M	64×	9.01 M	32×	288.25 K
DBLP(Full)	1.56G	64×	797.09M	32×	24.91M

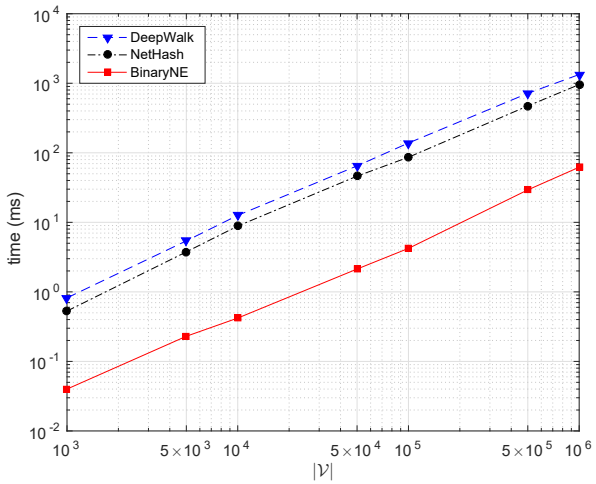
By contrast, the binary node representations learned by BinaryNE only consume 25M memory for the DBLP network, which is more practical for general devices. The low memory consumption makes BinaryNE more desirable for real-world applications.

8.4.8 Experiments on Search Scalability

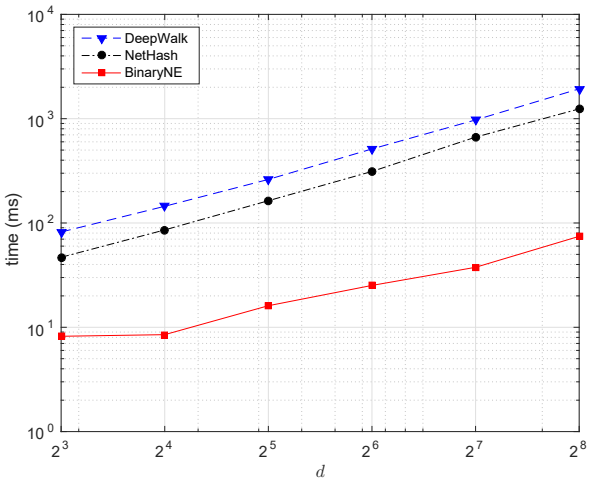
We also conduct experiments on the large DBLP(Full) network to test the search scalability of different types of network embeddings with respect to network size $|\mathcal{V}|$ and embedding dimension d. We compare the binary embeddings generated by BinaryNE with those by DeepWalk and NetHash, which respectively take continuous numeric values and non-binary discrete values.

To study the search scalability on network size $|\mathcal{V}|$, we first learn 128-dimensional embeddings with DeepWalk, NetHash and BinaryNE on the whole DBLP(Full) network, and then randomly sample a series of node subsets with increasing sizes. Among each node subset, we randomly select 1,000 nodes as query nodes and search similar nodes with the learned node representations. Fig. 8.2(a) shows query time (in milliseconds) with regard to different network sizes, where both query time (in milliseconds) and $|\mathcal{V}|$ are in logarithmic scales. As is shown, node similarity search with different embedding methods scales linearly with the increase of network size, whereas BinaryNE provides more than 10 times faster query speed than Deepwalk and NetHash.

To study the search scalability in terms of embedding dimension d, we learn DeepWalk, NetHash and BinaryNE embeddings with varying dimensions (8, 16, 32, 64, 128 and 256). We randomly select 100 nodes as query nodes, and search similar nodes across the whole DBLP(Full) network. Figure 8.2(b) shows query time (in milliseconds) with varying embedding dimensions, with both axes in logarithmic scales. We can see that, in general, similarity search with three methods scales almost linearly with regards to embedding dimension, but BinaryNE is consistently more efficient than DeepWalk and Nethash (with more than 10 times search speedup in most cases).



(a) $|\mathcal{V}|$



(b) d

Fig. 8.2 Query time with varying $|\mathcal{V}|$ and d

8.4.9 Comparison of Embedding Learning Time

In this subsection, we select the Cora and Citeseer network to evaluate the efficiency of learning node representations with different network embedding methods. Fig. 8.3 compares the CPU time (in seconds) consumed by different network embedding methods. As shown in the figure, BinaryNE is far more efficient in learning node representations than SDNE, TADW, UPP-SNE and MVC-DNE, and its efficiency is comparable to that of DeepWalk, LINE1, LINE2 and SINE, which have been demonstrated to be efficient

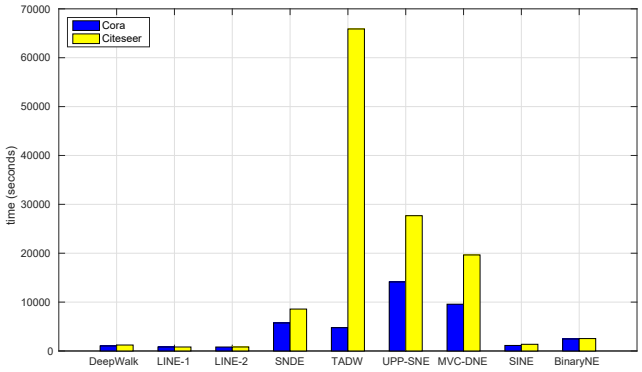
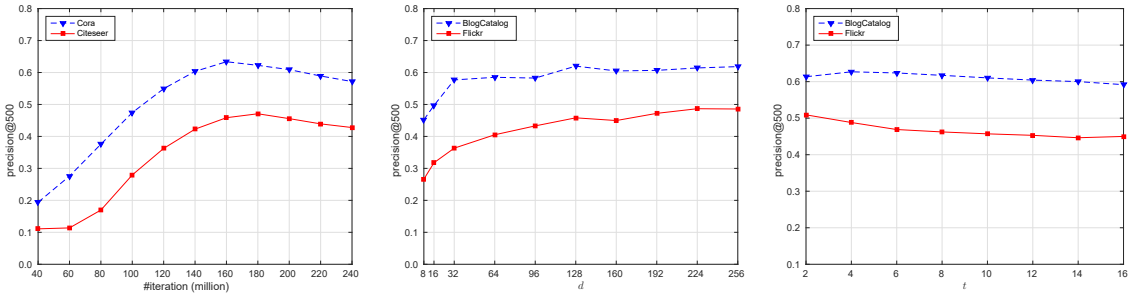


Fig. 8.3 The time consumed by different network embedding methods for learning node representations



(a) #iteration (b) d (c) t

Fig. 8.4 The sensitivity of BinaryNE with parameters: the number of iterations, the dimension of learned embeddings d, and the window size t

on large-scale networks. This proves the ability of BinaryNE to scale to large-scale networks for learning node representations, like DeepWalk, LINE and SINE.

8.4.10 Experiments on Parameter Sensitivity

We also perform a case study on BlogCatalog and Flickr to investigate the sensitivity of BinaryNE regarding to three important parameters: the number of iterations, the dimension of learned embeddings d, and the window size t used for collecting node context pairs. We take turns to fix any two parameters and study the effect of the remaining parameter on the search performance measured by precision@500. Fig. 8.4 shows the performance of BinaryNE with respect to varying parameters. As the number of iterations increases, the performance of BinaryNE gradually increases and then

declines slightly. This indicates that, in general, more iterations would be helpful for BinaryNE to find the local minimal solution, but excessive iterations tend to make the model parameters deviate from the local minima. When the embedding dimension d increases, the performance of BinaryNE increases and stabilizes later. This shows that, embeddings with higher dimensions provide more information to measure node similarity. Interestingly, when the window size t increases from 2 to 16, the search precision drops slightly. This is probably because a larger window size imports broader contextual structure, but may introduce more noise to measure node similarity.

8.5 Conclusion

Learning binary node representations is a desirable solution to similarity search over large-scale networks, with efficient bitwise Hamming distance calculation and low memory usage. In this chapter, we proposed the BinaryNE algorithm to embed network nodes into a binary space, with well preserved network structure and node content features. Through a three-layer neural network, BinaryNE learns node representations by modeling node structural context and node attribute relations. The *sign* function is adopted as the activation function in the hidden layer to obtain binary node representations. To deal with the *ill-posed gradient* problem caused by the non-smoothness of the *sign* activation function, the state-of-the-art continuation technique [3, 10] is employed. Model parameters are efficiently learned through an online stochastic gradient descent algorithm, which ensures the low time complexity and great scalability of BinaryNE. Extensive experiments on six real-world networks show that BinaryNE exhibits much lower memory usage and computational cost than continuous network embedding algorithms, but with comparable or even better search precisions.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

In the era of Big Data, network embedding provides an efficient way to analyzing large-scale graphs. Existing research has proved that network embedding can benefit a variety of network analytic tasks, such as node classification, link prediction and community detection. However, current research on network embedding mainly focuses on plain network embedding, fails to leverage the key information in widely existing node content attributes, which are essential for measuring node similarity. In this thesis, to boost the research on network embedding with node content, we develop a series of attributed network embedding algorithms, including task-dependent attributed network embedding algorithms and task-general attributed network embedding algorithms.

In Part I, to address the three main challenges in attributed network embedding, *i.e.*, *heterogeneity*, *data sparsity* and *scalability*, we propose three attribute network embedding algorithms: HSCA algorithm that effectively integrates information from homophily (the first-order proximity), structural context, and node content into a unified node representation through regularized inductive matrix factorization; attri2vec algorithm that learns node representations through performing a network structure guided mapping on node content attributes, which overcomes the discrepancy between network structure and node content attributes; and SINE algorithm that learns node representations for large-scale incomplete attributed networks by separately modeling

neighboring node pair relations and node-attribute relations through a three-layer neural network.

In Part II, we step further to make attributed network embedding directly benefit the targeted network analytic tasks. We select two targeted tasks, *i.e.*, *collective classification* and *node similarity search*, and develop two corresponding task-dependent attributed network embedding algorithms: DMF that seamlessly integrates the learning of node representations and a linear classifier into a unified framework, making the learned node representations discriminative for node labels; and BinaryNE that learns binary node representations to support quick similarity search by simultaneously capturing the structure similarity and content similarity between node pairs in a probabilistic way.

On nine real-world attributed networks, to evaluate the effectiveness of the proposed attributed network embedding algorithms, we conduct extensive experiments, including node classification, node clustering, link prediction, node similarity search and visualization. The experimental results show that the proposed algorithms are able to significantly outperform the state-of-the-art baselines.

9.2 Future Work

Though some efforts have been made, the research on attributed network embedding is still in early stage. In the future, to foster the research in this area, attributed network embedding can be explored in these aspects:

- **Deep models**. Deep learning has achieved great success on modeling image, text and audio data, because of their ability in extracting abstract meaningful latent features. However, due to the irregular connectivity structure, deep learning cannot be directly applied to networks to learn node representations [94]. Though some efforts [9, 87, 97] have been made to leverage deep learning to do network embedding or attributed network embedding, more in-depth study is still required. For example, how to learn deep node representations for attributed networks, which carry the information from both network structure and node

content attributes, while effectively overcome the discrepancy between these two information sources.

- **Dynamics**. Current research on network embedding has mainly concerned static networks. However, in real-life scenarios, networks are not always static. The underlying network structure may evolve over time, i.e., new nodes/edges appear while some old nodes/edges disappear. Meanwhile, the node content attributes may be dynamically updated. In this thesis, our attri2vec algorithm proposes a solution to the out-of-sample problem, *i.e.*, inferring the representations of new coming nodes. However, more efforts need to be made on dynamic attributed network embedding.

- **Scalability**. Nowadays, real-world attributed networks usually present in large scales, with billions of nodes and edges, as well as high-dimensional content features. The scalability constitutes a big challenge to attributed network embedding. Though our SINE algorithm is able to scale to large networks, it does not effectively capture the interplay between network structure and node content attributes. Designing scalable and effective attributed network embedding algorithms needs further exploration.

- **Robustness**. Real-world networks are often noisy and uncertain, which makes traditional network embedding algorithms unable to produce stable and robust representations. ANE (Adversarial Network Embedding) [13] and ARGA (Adversarially Regularized Graph Autoencoder) [61] learn robust node representations via enforcing an adversarial learning regularizer [88]. It is of great importance to have more research efforts on enhancing the robustness of attributed network embedding.

References

[1] Adamic, L. A. and Adar, E. (2003). Friends and neighbors on the web. *Social Networks*, 25(3):211–230.

[2] Aiello, L. M., Barrat, A., Schifanella, R., Cattuto, C., Markines, B., and Menczer, F. (2012). Friendship prediction and homophily in social media. *ACM Transactions on the Web (TWEB)*, 6(2).

[3] Allgower, E. L. and Georg, K. (2012). *Numerical continuation methods: an introduction*, volume 13. Springer Science & Business Media.

[4] Belkin, M. and Niyogi, P. (2002). Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems*, pages 585–591.

[5] Bhagat, S., Cormode, G., and Muthukrishnan, S. (2011). Node classification in social networks. *Social Network Data Analytics*, pages 115–148.

[6] Bhuyan, M. H., Bhattacharyya, D. K., and Kalita, J. K. (2014). Network anomaly detection: methods, systems and tools. *Communications Surveys & Tutorials, IEEE*, 16(1):303–336.

[7] Bianconi, G., Pin, P., and Marsili, M. (2009). Assessing the relevance of node features for network structure. *Proceedings of the National Academy of Sciences*, 106(28):11433–11438.

[8] Cao, S., Lu, W., and Xu, Q. (2015). GraRep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 891–900. ACM.

[9] Cao, S., Lu, W., and Xu, Q. (2016). Deep neural networks for learning graph representations. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 1145–1152.

[10] Cao, Z., Long, M., Wang, J., and Philip, S. Y. (2017). Hashnet: Deep learning to hash by continuation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5609–5618.

[11] Chen, J., Zhang, Q., and Huang, X. (2016). Incorporate group information to enhance network embedding. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*, pages 1901–1904.

[12] Chi, L. and Zhu, X. (2017). Hashing techniques: A survey and taxonomy. *ACM Computing Surveys (CSUR)*, 50(1):11.

[13] Dai, Q., Li, Q., Tang, J., and Wang, D. (2018). Adversarial network embedding. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[14] Donnat, C., Zitnik, M., Hallac, D., and Leskovec, J. (2017). Spectral graph wavelets for structural role similarity in networks. *arXiv preprint arXiv:1710.10321*.

[15] Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9(Aug):1871–1874.

[16] Gallagher, B. and Eliassi-Rad, T. (2010). Leveraging label-independent features for classification in sparsely labeled networks: An empirical study. In *Advances in Social Network Mining and Analysis*, pages 1–19. Springer.

[17] Gallagher, B., Tong, H., Eliassi-Rad, T., and Faloutsos, C. (2008). Using ghost edges for classification in sparsely labeled networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 256–264. ACM.

[18] Gao, S., Denoyer, L., and Gallinari, P. (2011). Temporal link prediction by integrating content and structure information. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, pages 1169–1174.

[19] Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864.

[20] Guo, T., Pan, S., Zhu, X., and Zhang, C. (2018). CFOND: Consensus factorization for co-clustering networked data. *IEEE Transactions on Knowledge and Data Engineering*.

[21] Gutmann, M. U. and Hyvärinen, A. (2012). Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13(Feb):307–361.

[22] Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034.

[23] Haveliwala, T. H. (2002). Topic-sensitive pagerank. In *Proceedings of the 11th International Conference on World Wide Web*, pages 517–526. ACM.

[24] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

[25] Hotelling, H. (1936). Relations between two sets of variates. *Biometrika*, 28(3/4):321–377.

[26] Huang, X., Li, J., and Hu, X. (2017a). Accelerated attributed network embedding. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 633–641. SIAM.

[27] Huang, X., Li, J., and Hu, X. (2017b). Label informed attributed network embedding. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining*, pages 731–739. ACM.

[28] Jeh, G. and Widom, J. (2002). SimRank: a measure of structural-context similarity. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 538–543. ACM.

[29] Jin, R., Lee, V. E., and Hong, H. (2011). Axiomatic ranking of network role similarity. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 922–930. ACM.

[30] Katz, L. (1953). A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43.

[31] Kazienko, P. and Kajdanowicz, T. (2012). Label-dependent node classification in the network. *Neurocomputing*, 75(1):199–209.

[32] Kessler, M. M. (1963). Bibliographic coupling between scientific papers. *American Documentation*, 14(1):10–25.

[33] Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *Proceddings of the 5th International Conference on Learning Representations (ICLR)*.

[34] Kossinets, G. (2006). Effects of missing data in social networks. *Social Networks*, 28:247–268.

[35] Kuang, D., Ding, C., and Park, H. (2012). Symmetric nonnegative matrix factorization for graph clustering. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 106–117. SIAM.

[36] Kusumoto, M., Maehara, T., and Kawarabayashi, K.-i. (2014). Scalable similarity search for simrank. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pages 325–336. ACM.

[37] Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196.

[38] Lee, P., Lakshmanan, L. V., and Yu, J. X. (2012). On top-k structural similarity search. In *Proceedings of the 28th IEEE International Conference on Data Engineering*, pages 774–785. IEEE.

[39] Leskovec, J. and Mcauley, J. J. (2012). Learning to discover social circles in ego networks. In *Advances in Neural Information Processing Systems*, pages 539–547.

[40] Li, A. Q., Ahmed, A., Ravi, S., and Smola, A. J. (2014). Reducing the sampling complexity of topic models. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 891–900. ACM.

[41] Li, C., Li, Z., Wang, S., Yang, Y., Zhang, X., and Zhou, J. (2017a). Semi-supervised network embedding. In *International Conference on Database Systems for Advanced Applications*, pages 131–147.

[42] Li, C., Wang, S., Yang, D., Li, Z., Yang, Y., Zhang, X., and Zhou, J. (2017b). PPNE: Property preserving network embedding. In *International Conference on Database Systems for Advanced Applications*, pages 163–179.

[43] Li, J., Zhu, J., and Zhang, B. (2016). Discriminative deep random walk for network classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 1004–1013.

[44] Liao, L., He, X., Zhang, H., and Chua, T.-S. (2018). Attributed social network embedding. *IEEE Transactions on Knowledge and Data Engineering*.

[45] Lin, Y., Liu, Z., Sun, M., Liu, Y., and Zhu, X. (2015). Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 2181–2187.

[46] Liu, Z., Zheng, V. W., Zhao, Z., Zhu, F., Chang, K. C.-C., Wu, M., and Ying, J. (2018). Distance-aware DAG embedding for proximity search on heterogeneous graphs. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 2355–2362.

[47] Lü, L. and Zhou, T. (2011). Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170.

[48] Lyu, T., Zhang, Y., and Zhang, Y. (2017). Enhancing the network embedding quality with structural similarity. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 147–156.

[49] Macskassy, S. A. (2007). Improving learning in networked data by combining explicit and mined links. In *AAAI Conference on Artificial Intelligence*, volume 22, pages 590–595.

[50] Malliaros, F. D. and Vazirgiannis, M. (2013). Clustering and community detection in directed networks: A survey. *Physics Reports*, 533(4):95–142.

[51] McDowell, L. K. and Aha, D. W. (2012). Semi-supervised collective classification via hybrid label regularization. In *Proceedings of the 29th International Conference on International Conference on Machine Learning*, pages 1243–1250. Omnipress.

[52] McDowell, L. K. and Aha, D. W. (2013). Labels or attributes?: rethinking the neighbors for collective classification in sparsely-labeled networks. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, pages 847–852. ACM.

[53] McDowell, L. K., Gupta, K. M., and Aha, D. W. (2009). Cautious collective classification. *Journal of Machine Learning Research*, 10(Dec):2777–2836.

[54] McPherson, M., Smith-Lovin, L., and Cook, J. (2001). Birds of a feather: homophily in social networks. *Annual Review of Sociology*, 27:415–444.

[55] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

[56] Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252.

[57] Natarajan, N. and Dhillon, I. S. (2014). Inductive matrix completion for predicting gene–disease associations. *Bioinformatics*, 30(12):i60–i68.

[58] Neville, J. and Jensen, D. (2000). Iterative classification in relational data. In *Proceedings of AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 13–20.

[59] Newman, M. E. (2006). Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74(3):036104.

[60] Ou, M., Cui, P., Pei, J., Zhang, Z., and Zhu, W. (2016). Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1105–1114.

[61] Pan, S., Hu, R., Long, G., Jiang, J., Yao, L., and Zhang, C. (2018). Adversarially regularized graph autoencoder. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*.

[62] Pan, S., Wu, J., Zhu, X., Zhang, C., and Wang, Y. (2016). Tri-party deep network representation. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pages 1895–1901.

[63] Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). DeepWalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 701–710.

[64] Rahimi, A. and Recht, B. (2008). Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, pages 1177–1184.

[65] Reagans, R. and McEvily, B. (2003). Network structure and knowledge transfer: The effects of cohesion and range. *Administrative Science Quarterly*, 48(2).

[66] Ribeiro, L. F., Saverese, P. H., and Figueiredo, D. R. (2017). struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 385–394.

[67] Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326.

[68] Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. (2008). Collective classification in network data. *AI Magazine*, 29(3):93.

[69] Shi, X., Li, Y., and Yu, P. (2011). Collective prediction with latent graphs. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, pages 1127–1136. ACM.

[70] Small, H. (1973). Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for Information Science*, 24(4):265–269.

[71] Song, H. H., Cho, T. W., Dave, V., Zhang, Y., and Qiu, L. (2009). Scalable proximity estimation and link prediction in online social networks. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, pages 322–335.

[72] Srilatha, P. and Manjula, R. (2016). Similarity index based link prediction algorithms in social networks: a survey. *Journal of Telecommunications and Information Technology*.

[73] Strehl, A. and Ghosh, J. (2002). Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3(Dec):583–617.

[74] Subbaraj, K. and Sundan, B. (2015). What happens next? prediction of disastrous links in covert networks. *Disaster Advances*, 8:53–60.

[75] Tang, J., Liu, J., and Mei, Q. (2016). Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th International Conference on World Wide Web*, pages 287–297.

[76] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. (2015). LINE: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077.

[77] Tang, L. and Liu, H. (2009a). Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 817–826. ACM.

[78] Tang, L. and Liu, H. (2009b). Scalable learning of collective behavior based on sparse social dimensions. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 1107–1116. ACM.

[79] Tang, L. and Liu, H. (2011). Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23(3):447–478.

[80] Tenenbaum, J. B., De Silva, V., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323.

[81] Tsourakakis, C. E. (2014). Toward quantifying vertex similarity in networks. *Internet Mathematics*, 10(3-4):263–286.

[82] Tu, C., Zhang, W., Liu, Z., and Sun, M. (2016). Max-Margin DeepWalk: discriminative learning of network representation. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pages 3889–3895.

[83] Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(2579-2605):85.

[84] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2018). Graph attention networks. In *Proceddings of the 6th International Conference on Learning Representations (ICLR)*.

[85] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408.

[86] Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416.

[87] Wang, D., Cui, P., and Zhu, W. (2016a). Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1225–1234. ACM.

[88] Wang, H., Wang, J., Wang, J., Zhao, M., Zhang, W., Zhang, F., Xie, X., and Guo, M. (2018). GraphGAN: Graph representation learning with generative adversarial nets. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 2508–2515.

[89] Wang, S., Tang, J., Aggarwal, C., and Liu, H. (2016b). Linked document embedding for classification. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 115–124. ACM.

[90] Wang, S., Tang, J., Morstatter, F., and Liu, H. (2016c). Paired restricted Boltzmann machine for linked data. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*, pages 1753–1762.

[91] Wang, X., Cui, P., Wang, J., Pei, J., Zhu, W., and Yang, S. (2017). Community preserving network embedding. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 203–209.

[92] Weiss, Y., Torralba, A., and Fergus, R. (2009). Spectral hashing. In *Advances in Neural Information Processing Systems*, pages 1753–1760.

[93] Wu, W., Li, B., Chen, L., and Zhang, C. (2018). Efficient attributed network embedding via recursive randomized hashing. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 2861–2867.

[94] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2019). A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*.

[95] Xie, M., Yin, H., Wang, H., Xu, F., Chen, W., and Wang, S. (2016). Learning graph-based POI embedding for location-based recommendation. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*, pages 15–24.

[96] Yang, C., Liu, Z., Zhao, D., Sun, M., and Chang, E. Y. (2015). Network representation learning with rich text information. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 2111–2117.

[97] Yang, D., Wang, S., Li, C., Zhang, X., and Li, Z. (2017). From properties to links: deep network embedding on incomplete graphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 367–376. ACM.

[98] Yang, Z., Cohen, W. W., and Salakhutdinov, R. (2016). Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pages 40–48.

[99] Zhang, C., Zhang, K., Yuan, Q., Peng, H., Zheng, Y., Hanratty, T., Wang, S., and Han, J. (2017a). Regions, periods, activities: Uncovering urban dynamics via cross-modal representation learning. In *Proceedings of the 26th International Conference on World Wide Web*, pages 361–370.

[100] Zhang, D., Yin, J., Zhu, X., and Zhang, C. (2016a). Collective classification via discriminative matrix factorization on sparsely labeled networks. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*, pages 1563–1572.

[101] Zhang, D., Yin, J., Zhu, X., and Zhang, C. (2016b). Homophily, structure, and content augmented network representation learning. In *Proceedings of the 16th IEEE International Conference on Data Mining*, pages 609–618. IEEE.

[102] Zhang, D., Yin, J., Zhu, X., and Zhang, C. (2017b). User profile preserving social network embedding. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 3378–3384.

[103] Zhang, D., Yin, J., Zhu, X., and Zhang, C. (2018a). MetaGraph2Vec: Complex semantic path augmented heterogeneous network embedding. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 196–208. Springer.

[104] Zhang, D., Yin, J., Zhu, X., and Zhang, C. (2018b). Network representation learning: A survey. *IEEE Transactions on Big Data*.

[105] Zhang, D., Yin, J., Zhu, X., and Zhang, C. (2018c). SINE: Scalable incomplete network embedding. In *IEEE International Conference on Data Mining*, pages 737–746. IEEE.

[106] Zhang, D., Yin, J., Zhu, X., and Zhang, C. (2019a). Attributed network embedding via subspace discovery. *arXiv preprint arXiv:1901.04095*.

[107] Zhang, D., Yin, J., Zhu, X., and Zhang, C. (2019b). Search efficient binary network embedding. *arXiv preprint arXiv:1901.04097*.

[108] Zhang, J., Tang, J., Ma, C., Tong, H., Jing, Y., and Li, J. (2015). Panther: Fast top-k similarity search on large networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1445–1454. ACM.

[109] Zhang, X., Chen, W., and Yan, H. (2016c). TLINE: Scalable transductive network embedding. In *Asia Information Retrieval Symposium*, pages 98–110. Springer.

[110] Zhao, P., Han, J., and Sun, Y. (2009). P-Rank: A comprehensive structural similarity measure over information networks. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 553–562. ACM.

[111] Zhou, C., Liu, Y., Liu, X., Liu, Z., and Gao, J. (2017). Scalable graph embedding for asymmetric proximity. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 2942–2948.

[112] Zhu, S., Yu, K., Chi, Y., and Gong, Y. (2007). Combining content and link for classification using matrix factorization. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 487–494.