Pattern Recognition 123 (2022) 108368

Contents lists available at ScienceDirect

Pattern Recognition

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

Discrete embedding for attributed graphs

Hong Yang[a], Ling Chen[b], Shirui Pan[c], Haishuai Wang[d], Peng Zhang[e,*]

[a] Faculty of Medicine and Health, The University of Sydney, Australia
[b] Faculty of Information Technology, University of Technology Sydney, Australia
[c] Faculty of Information Technology, Monash University, Australia
[d] Department of Computer Science and Engineering, Fairfield University, USA
[e] Cyberspace Institute of Advanced Technology, Guangzhou University, China

ARTICLE INFO

Article history:
Received 1 March 2021
Revised 23 September 2021
Accepted 9 October 2021
Available online 15 October 2021

Keywords:
Attributed graphs
Graph embedding
Weisfeiler-Lehman graph kernels
Learning to hash
Low-bit quantization

ABSTRACT

Attributed graphs refer to graphs where both node links and node attributes are observable for analysis. Attributed graph embedding enables joint representation learning of node links and node attributes. Different from classical graph embedding methods such as *Deepwalk* and *node2vec* that first project node links into low-dimensional vectors which are then linearly concatenated with node attribute vectors as node representation, attributed graph embedding fully explores data dependence between node links and attributes by either using node attributes as class labels to supervise structure learning from node links, or reversely using node links to supervise the learning from node attributes. However, existing attributed graph embedding models are designed in continuous Euclidean spaces which often introduce data redundancy and impose challenges to storage and computation costs. In this paper, we study a new problem of discrete embedding for attributed graphs that can learn succinct node representations. Specifically, we present a *Binarized Attributed Network Embedding* model (*BANE* for short) to learn binary node representation by factorizing a *Weisfeiler-Lehman proximity matrix* under the constraint of binary node representation. Furthermore, based on BANE, we propose a new *Low-bit Quantization for Attributed Network Representation* learning model (LQANR for short) to learn even more compact node representation of bit-width values. Theoretical analysis and empirical studies on real-world datasets show that the new discrete embedding models outperform benchmark methods.

1. Introduction

Attributed graphs are popularly used to describe a large body of networks where both node links and node attributes are observable for analysis. Typical examples of attributed graphs include social network data, academic citation data, and protein-protein interaction data [1]. To obtain knowledge from attributed graphs, graph embedding models are proposed to project node links into low-dimensional vectors. Then, the projected vectors are linearly concatenated with node attribute vectors to represent the nodes for downstream learning tasks such as link prediction [2], node classification [3], and social network recommendations [4,5].

The key idea of graph embedding is to design a mapping function which converts graph nodes into low-dimensional vectors. In general, a mapping function should fulfill three principles: 1) Scalability. Real-world graphs are naturally large-scale. Thus, graph embedding models should be able to handle large-scale graphs efficiently. 2) Sparsity. Generally there are only a few number of nodes labeled for training in a graph. Thus, graph embedding models are expected to learn low-dimensional vectors to reduce the number of weight parameters of downstream learning models. 3) Adaptability. Graphs are often evolving with time. Embedding mapping functions should be able to adapt with time and avoid frequently repetitive training.

Considering the above principles, a class of graph embedding models has been proposed. DeepWalk [6] represents the seed work which borrows the idea of word embedding, treats nodes as words and generates short random walks as sentences. Then, linguistic models such as Skip-gram are applied to the random walks and obtain node representations. Based on DeepWalk, node2vec [7] introduces the biased random walk that enables breadth first search (BFS) and depth first search (DFS) neighborhood exploration. Based on DeepWalk and node2vec, a number of sophisticated graph embedding models are proposed for handling large-scale networks. For example, LINE [8] uses a breadth-first search strategy to gener-

* Corresponding author.
E-mail addresses: h.yang@usydney.edu.au (H. Yang), Ling.chen@uts.edu.au (L. Chen), shirui.pan@monash.edu.au (S. Pan), hwang@fairfield.edu (H. Wang), p.zhang@gzhu.edu.cn (P. Zhang).

https://doi.org/10.1016/j.patcog.2021.108368
0031-3203/© 2021 Elsevier Ltd. All rights reserved.

ate context nodes on large-scale networks. GraphAttention [9] uses an attention model that can learn multi-scale representations for link prediction. SDNE [10] learns node representations where the proximity between two-hop neighbors is maintained by using a depth auto-encoder.

The above graph embedding models fall into a two-stage learning category, where node links are vectorized independently without using any auxiliary information from node attributes. As a result, they are incapable of capturing data dependence between node links and attributes, which is often referred to as *plain graph embedding*. To enable exploitation of the dependence information between node links and node attributes, *attributed graph embedding* models are proposed to jointly learn from node links and attributes. The principle is to use node attributes as class labels to supervise structure learning from node links, or vice versa. For example, the work [11] uses textual attributes to supervise random walks on networks and derives the Text-associated DeepWalk (TADW) model. On the contrary, the work [12] reversely uses node links to supervise the factorization of attributed matrices. The work [13] mutually uses node links and attributes as labels to supervise the learning from each other. Generally, attributed graph embedding outperforms plain graph embedding by considering data dependence between node attributes and node links.

However, existing attributed network embedding models are developed in continuous Euclidean spaces. By embedding the dependence information of node attributes and links, the learned vectors may contain redundant information that degenerates computation efficiency and increases storage cost, especially when networks are very large. Imagining the task of k-nearest neighbor search to recommend the top-k most similar friends in a large network of size n, assuming the representation vector is of length d, the similarity search will take time $O(n^2d)$. Thus, compact node representation is preferred to speedup computation and reduce storage cost. For example, if we use binary embedding and encode each node with 128 bits, we can store a data set of 1 million nodes with only 16M memory. Moreover, the speed of low-bit computation is faster than floating numbers, because the expensive oating-point multiplication operation can be replaced by a sequence of cheaper and faster bit shift operations of fixed-point numbers.

Recently, discrete representation learning has attracted increasing attention and a number of hash algorithms and binary code learning have been proposed to learn discrete representations in Hamming spaces. The idea of hashing algorithms and binary coding [14] is to encode high-dimensional feature vectors of documents, images and videos to compact binary codes, while preserving the similarity structure in the original space. In particular, binary code learning can generate succinct representations by encoding high-dimensional data into a set of short binary codes with similarity preservation. Binary coding is also referred to as *hashing* which maps data to discrete Hamming spaces [15]. The binary codes can facilitate representation and search of massive data because it only needs a relatively small size of binary bits to represent a data item, and binary computation in Hamming space is efficient by using the bit operations.

In this paper, we study the problem of discrete attributed graph embedding. The key challenge is *to aggregate information of both node links and attributes for discrete node representation*. Considering matrix factorization as the discrete graph embedding framework [13,16], the challenges of discrete embedding for attributed networks can be summarized as follows:

- *Challenge 1*: How to design a proximity matrix to capture data dependence between node links and node attributes in attributed graphs. To our best knowledge, none of existing network proximity matrices encodes both node links and attributes.
- *Challenge 2*: How to design a fast algorithm to solve the discrete representation learning problem. Existing models for embedding attributed networks are formulated in the Euclidean space. However, factorizing a network proximity matrix under binary constraints falls into the integer programming category which requires new efficient algorithms.
- *Challenge 3*: How to theoretically and empirically prove the effectiveness and efficiency of the learning model.

To solve the above challenges, we present a new *Binarized Attributed Network Embedding* model (*BANE* for short). Inspired by the *Weisfeiler-Lehman graph kernels* [17], we define a new *Weisfeiler-Lehman proximity matrix* to capture data dependence between node links and attributes. Then, based on the new proximity matrix, we formulate a *Weisfeiler-Lehman matrix factorization* learning function under the binary representation constraint. The learning problem falls into the category of *mixed integer optimization* and we use an efficient *cyclic coordinate descent* (CCD) algorithm [18] as the solution. We theoretically prove the advantages of the Weisfeiler-Lehman proximity matrix by analyzing its connections with Weisfeiler-Lehman graph kernels [19], Laplacian smoothing [20], and Graph Convolutional Networks (GCNs) [21]. Furthermore, based on BANE, we present a low-bit quantization model that can learn even more compact node representation of bit-width values. Experimental results on real-world datasets validate the performance of the proposed methods. The framework of the learning models is illustrated in Fig. 1.

The contributions of the paper are summarized as follows:

- We first study the problem of discrete embedding learning for attributed graphs, and present a new *Binarized Attributed Network Embedding* model (BANE for short) as the solution.
- We define a new *Weisfeiler-Lehman proximity matrix* to encode data dependence between node links and node attributes, based on which a new *Weisfeiler-Lehman matrix factorization* is presented to learn binary representation. Moreover, we theoretically prove the connections between the proposed Weisfeiler-Lehman matrix and Weisfeiler-Lehman graph kernels, Laplacian smoothing, and Graph Convolutional Networks (GCNs).
- Based on BANE, we further propose a *Low-Bit Quantization for Attributed Network Representation* learning model (LQANR for short). LQANR can learn more compact node representation of bit-width values with stably high accuracy. Also, we introduce a new *mixed-integer based alternating direction method of multipliers (ADMM)* algorithm to solve LQANR.
- We conduct experiments to validate the performance of the proposed models. The Matlab codes[1] and Pyhthon codes[2] are available online.

The paper is an extension of its former conference version [22], where more theoretical and empirical studies are added in Sections 5 and 6. In addition, a low-bit quantization model LQANR is also added for comparisons. The rest of the paper is organized as follows. Section 2 surveys related work. Section 3 introduces the preliminaries. Section 4 introduces the learning models of BANE and LQANR. Section 5 discusses the advantages of the proposed methods. Section 6 conducts experiments, and we conclude the work in the last section.

2. Related work

Graph Embedding. Graph embedding is also called as network embedding or graph representation learning. Current graph

[1] MATLAB codes: https://github.com/ICDM2018-BANE/BANE.

[2] Python codes: https://github.com/benedekrozemberczki/BANE.

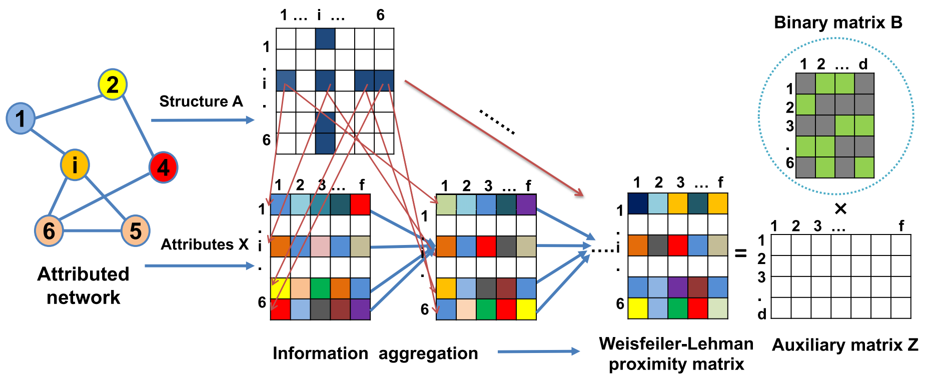


Fig. 1. The conceptual framework of the *Binarized Attributed Network Embedding (BANE)*. Given an attributed network $G = \{V, E, X\}$, derive a *Weisfeiler-Lehman proximity matrix* $P = (I - \gamma \tilde{D}^{-1}\tilde{L})^k X$ by aggregating structure matrix A and attribute matrix X. Factorizing matrix P into a binary node representation matrix B and an auxiliary matrix Z. In the figure, the colored cells and arrows explain the formulation of matrix P from the perspective of feature propagation. Given a target node i, the blue cells in the structure matrix A highlight non-zero entries corresponding to the neighbouring nodes of i. The red arrows denotes that the feature vectors of these non-zero entries will be propagated to the target node i. The blue arrows pointing to the target node i denote that a new feature vector of node i is formulated by aggregating feature vectors of itself and its neighboring nodes. Such an feature propagation will be repeated for k times and generate the final matrix P.

embedding methods can be categorized into *plain graph embedding* [23] and *attributed graph embedding* [24]. Different from plain graph embedding that independently vectorizes node links without using auxiliary information from node attributes, attributed graph embedding jointly models their dependence, by using node attributes as class labels to supervise the learning of node links, or vice versa. A typical attributed graph embedding model is the TADW model [11] that uses textual attributes to supervise random walks on networks. LANE [13] learns node representations for attributed networks by embedding the network structure proximity, attribute affinity and label proximity into a unified latent representation. AANE [12] learns node embedding by using symmetric matrix factorization on attribute affinity matrix, and simultaneously minimizing the representation difference between connected nodes. PPNE [25] learns node representations for attributed networks through jointly optimizing two objectives, i.e., the structure-driven objective and the attribute-driven objective. Similar works include Dynamic Attributed Network Embedding (DANE) [26] and Diffusion Network Embedding (DNE) [16].

A number of graph neural networks [27] are also developed to learn node embeddings from attributed networks. The early and most important work is Graph Convolutional Network (GCNs) [21]. GCNs take node attributes as input and construct node representations through the convolution of neighboring node representations at each hidden layer. The last layer is used to predict node labels by minimizing a cross-entropy loss. GraphSAGE [28] takes node content features as node representations, and then iteratively updates node representations by aggregating representations of neighboring nodes. Graph Attention Network (GAT) [9] then imports the self-attention mechanism into GCN to make the graph convolution performed in a more intelligent way. Graph Isomorphism Network (GIN) [29] creates injective multiset functions for the neighbor aggregation and generalizes the Weisfeiler-Lehman test. GraphNAS [30] can automatically design the best graph neural architecture using reinforcement learning.

Weisfeiler-Lehman Graph Kernels. Graph kernels [19] can be intuitively understood as functions measuring the similarity of pairs of graphs. The most common graph kernels are random walk kernels, shortest-path kernels, graphlet kernels, and Weisfeiler-Lehman graph kernels [17]. A random-Walk kernel measures the similarity of labeled graphs by comparing the random walks on graphs. Shortest-path kernels are similar to random-walk graph kernels but the walk paths are formed from the shortest paths. The graphlet kernels count the number of substructures, graphlets, in the graphs. Weisfeiler-Lehman Graph Kernels enumerate the shared subtrees in graphs. **Learning to Hash**. Hashing or binary coding [14] encodes high-dimensional feature vectors of documents, images and videos to compact binary codes, while preserving similarity structure in the original space. The binary codes can facilitate to represent and search of massive data because it only needs about one hundred binary bits to represent one data item, and binary computation in Hamming space is efficient by using the bit operations. Many learning-based hashing algorithms have been developed according to different scenarios. For example, the unsupervised methods [31], supervised methods [18], deep learning based hashing methods [32]. To the best of our knowledge, no prior studies have been focused on seeking binary representation for attributed network to preserve both network structure and node attributes.

Low-bit Quantization for Compression. Quantization methods including hashing are used to encode real-valued data to low-bit discrete data while preserving similarity structure in the original space [14]. The low-bit codes can facilitate representation and search of massive data because it only needs a relatively small size of bits to represent one data item, and computation in Hamming space is efficient by using the bit operations. Most Hashing methods use one single bit $-1/+1$ to quantize each projected dimension [33], such as the spectral hashing [31], supervised discrete hashing [18] and deep learning based hashing methods [32]. There are also works quantizing real-valued data to multiple-bit codes using Hashing method [33], such as double bit quantization, q-bit Manhattan quantization, and Variable Bit Quantization. Recently discrete network embedding approach is proposed to learn binary codes for plain network [34], and randomized hashing method [35] and binarized network embedding [22] are proposed for compressing embedding for attributed networks.

In this paper, we aim to learn discrete node representations for attributed networks by leveraging the strength of the state-of-the-art graph embedding, Weisfeiler-Lehman graph kernels, learning to hash, and low-bit quantization for compression methods.

3. Preliminaries

An attributed graph can be represented as $G = \{V, E, X\}$, where $V = \{v_i\}_{i=1}^n$ denotes nodes, $E = \{e_{ij}\}_{i,j=1}^n$ denotes undirected edges, and $X = \{x_i\}_{i=1}^n \in R^{n\times f}$ denotes attribute vectors of the nodes with f being the dimension of attribute vectors. In addition, the structure of network G can be derived from edges in E, denoted as an adjacency matrix A, where $A_{ij} = 1$ if $e_{ij} \in E$, otherwise, $A_{ij} = 0$. By adding a self-loop to each node in the network, we have $\tilde{A} = A + I$, where I is an identity matrix. $\tilde{D} = \text{diag}(\tilde{d}_1, \ldots, \tilde{d}_n)$ is a degree matrix of $\tilde{A}$, with $\tilde{d}_i = \sum_j \tilde{a}_{ij}$ being the degree of node v_i.

Given the attributed network G, we wish to embed each node $v_i \in V$ into a d-dimensional vector $b_i \in \{-1, +1\}^d$ in a Hamming space, where b_i is the ith row of matrix $B \in R^{n\times d}$. Ideally, matrix B can preserve the structure information A and the attribute information X in the original network G.

The key question is to design a proximity matrix which can jointly describe structure A and attribute X. For example, TADW [11] derives the proximity matrix by using the textual attributes X to supervise the random walk of structure A. The process can be taken as using the random walk kernel (supervised by node attributes) on graphs for node representation.

Instead of using random walk graph kernels, we use the Weisfeiler-Lehman graph kernels [17] to generate a new proximity matrix P that encodes both node attributes in X and edges in A. Specifically, we define the *Weisfeiler-Lehman proximity matrix* as follows,

Definition 1. (**Weisfeiler-Lehman Proximity Matrix**). Given a network G with adjacency matrix A and attribute matrix X, let $\tilde{D}$ be a degree matrix of $\tilde{A}$ and $\tilde{L} = \tilde{D} - \tilde{A}$, the Weisfeiler-Lehman proximity matrix P is defined as $P = (I - \gamma\tilde{D}^{-1}\tilde{L})^k X$, where $\gamma \in [0, 1]$ is a tradeoff parameter, and k is the number of aggregation layers.

Because the above Weisfeiler-Lehman proximity matrix is based on the Weisfeiler-Lehman graph kernels, the matrix naturally captures data dependence between node links and attributes. In particular, the proximity matrix has the following **properties**:

Property 1. The Weisfeiler-Lehman proximity matrix enables aggregation of node attributes and links from neighboring nodes to a target node. Parameter k controls the number of layers of neighboring nodes joining the aggregation. If $k = 1$ and $\gamma = 1$, matrix P equals a one-layer Weisfeiler-Lehman graph kernel (see *Section 5.1* for details).

Property 2. The Weisfeiler-Lehman proximity matrix enables the tradeoff of node aggregation between neighboring nodes and a target node, where γ is the smoothing parameter. That is, matrix P is actually a k-layer Laplacian smoothing [20] of the network (see *Section 5.2* for details).

Property 3. When $\gamma = 1$, matrix P defines a variant of k-layer graph convolutional networks (see *Section 5.3* for details).

In Section 5, we analyze the above properties by discussing the connections between the new proximity matrix and the Weisfeiler-Lehman graph kernels, Laplacian smoothing, and graph convolutional networks (GCNs).

4. The proposed methods

In this section, we first derive the learning function of the *Binarized Attributed Network Embedding* model (BANE) and a *Cyclic Coordinate Descent (CCD)* [18] algorithm in Section 4.1. Then, we formulate the learning function of *Low-Bit Quantization for Attributed Network Representation* learning model (LQANR) and an efficient *mixed-integer based alternating direction method of multipliers* algorithm in Section 4.2.

4.1. Binarized attributed network embedding (BANE)

Based on Definition 1, we factorize the Weisfeiler-Lehman proximity matrix $P = (I - \gamma\tilde{D}^{-1}\tilde{L})^k X$ which jointly encodes node attributes and links into a binary node representation matrix B and an auxiliary matrix Z. Formally, the learning function of the binarized Weisfeiler-Lehman matrix factorization can be defined as follows,

$$\min_{B,Z} \frac{1}{2}\|(I - \gamma\tilde{D}^{-1}\tilde{L})^k X - BZ\|_F^2 + \frac{\alpha}{2}\|Z\|_F^2, \tag{1}$$
$$s.t.: \; B \in \{-1, +1\}^{n\times d}, Z \in R^{d\times f},$$

where α is a regularization parameter with respect to the auxiliary matrix Z. Due to the binary constraint with respect to matrix B, Eq. (1) is NP-hard. Next, we introduce an efficient algorithm as the solution.

We present an alternating algorithm to solve Eq. (1). It updates one parameter at a time and converges fast.

Z **-Step**. Given B, solve the sub-problem with respect to Z in Eq. (1). The loss function can be written as follows,

$$\min_{Z} \frac{1}{2}\|(I - \gamma\tilde{D}^{-1}\tilde{L})^k X - BZ\|_F^2 + \frac{\alpha}{2}\|Z\|_F^2, \tag{2}$$
$$= -tr(P^T BZ) + \frac{1}{2}tr(Z^T B^T BZ) + \frac{\alpha}{2}tr(Z^T Z).$$

Note $P = (I - \gamma\tilde{D}^{-1}\tilde{L})^k X$, and $tr(.)$ is the trace norm. By calculating the derivative of Eq. (2), we derive a closed form solution as follows,

$$Z = (B^T B + \alpha I)^{-1} B^T P. \tag{3}$$

B **-Step**. It is difficult to solve B due to the discrete constraint. Given Z fixed, rewrite the objective function in Eq. (1) with respect to B as follows,

$$\min_{B} \frac{1}{2}\|(I - \gamma\tilde{D}^{-1}\tilde{L})^k X - BZ\|_F^2 \tag{4}$$
$$= \frac{1}{2}tr(Z^T B^T BZ) - tr(B^T PZ^T),$$
$$s.t.: B \in \{-1, +1\}^{n\times d}.$$

Under the observation that *a closed-form solution for one column of B can be achieved by fixing all the other columns*, the algorithm iteratively learns one bit of B at a time.

Let b^l be the lth column of B, and B' the matrix of B excluding b^l. Then, b^l is the lth bit for all the n samples. Similarly, let q^l be the lth column of $Q = PZ^T$, Q' the matrix of Q excluding q^l, z^l the lth row of Z and Z' the matrix of Z excluding z^l. Then we obtain

$$tr(Z^T B^T BZ) = z^l Z'^T B'^T b^l + const. \tag{5}$$

Following the same logic, we obtain

$$tr(B^T Q) = (q^l)^T b^l + const. \tag{6}$$

Plugging Eqs. (5) and (6) back into Eq. (4), we obtain the optimization problem with respect to b^l as follows,

$$\min_{b^l} z^l Z'^T B'^T b^l - (q^l)^T b^l \tag{7}$$
$$= (z^l Z'^T B'^T - (q^l)^T) b^l$$
$$s.t.: b^l \in \{-1, +1\}^{n\times 1}$$

Eq.(7) has a closed form solution as follows,

$$b^l = sign(q^l - B'Z'(z^l)^T). \tag{8}$$

By using this method, each bit b can be computed based on the pre-learned $d - 1$ bits of B'. The convergence of the alternating optimization is guaranteed theoretically, because every iteration decreases the objective function value and the objective function has a lower bound.

The details of the algorithm are given in Algorithm 1. Empirical results demonstrate that the algorithm takes a few iterations to converge. For example, in our experiments B is iteratively computed and the algorithm converges fast in about $3 - 10$ iterations.

Algorithm 1 Binarized Attributed Network Embedding (BANE).

```
Input: Graph structure $A$, node attribute $X$, dimension $d$, # of iterations $t_1$ and $t_2$, parameters $k$, $\gamma$, $\alpha$
Output: Binary node representation matrix $B$
1: Initialize $Z$, $B$ randomly
2: Repeat until converge or reach $t_1$
3: Z-Step: Calculate $Z$ using Eq. (3)
4: B-Step: Repeat until converge or reach $t_2$
5: for $l = 1, \cdots, d$ do
6:     update $b^l$ using Eq. (3)
7: end for
8: return matrix $B$
```

4.2. Low-bit quantization for attributed network representation learning (LQANR)

In this part, we discuss the challenging problem of designing a more compact discrete learning model than BANE. Specifically, considering the binary constraint of B in Eq. (1), how to learn a much sparser and smaller representation matrix B with low-bit values? The question is equivalent to embedding each node $v_i \in V$ into a d-dimensional low-bit vector $b_i \in \{-2^N, \dots, -2^1, -1, 0, 1, \dots, 2^N\}^d$, where N is an integer which determines the bit-width.

In fact, there has been a series of methods [14] proposed to reduce the size of network parameters. From the perspective of low-bit quantization for convolutional neural networks, low-bit compression of deep neural networks has been popularly studied recently, such as training binary neural networks with weights constrained to +1 and -1, ternary networks, and extremely low-bit neural networks. Compared to full-precision models, these compressed models are sparse and much smaller, which can potentially be accelerated with customized circuits and deployed to mobile devices. In particular, the early work [36] pointed out that network weights have a significant redundancy, and proposed to reduce the number of parameters by exploiting the linear structure of network, which motivated a line of low-rank matrix and tensor factorization based compression algorithms. The achievement in low-rank matrix and tensor factorization based compression motivates to learn low-bit quantization for attributed network embedding based on matrix factorization.

Consider an attributed network G, we need to embed each node $v_i \in V$ into a d-dimensional low-bit vector $b_i \in \{-2^N, \dots, -2^1, -1, 0, 1, \dots, 2^N\}^d$, where N is an integer which determines the bit-width. b_i is the ith row of matrix $B \in R^{n\times d}$. Matrix B should preserve the structure information A and the attribute information X in G. The representation learning function can be formulated by simultaneously learning the low-bit node representation and the layer-wise aggregation weights. Assume that α_k is the importance weight of matrix $P_k = (\tilde{D}^{-1}\tilde{A})^k X$, matrix $B \in R^{n\times d}$ is the low-bit node representation, matrix $Z_k \in R^{d\times f}$ is an auxiliary matrix with respect to layer k. Then, the learning problem can be formulated as follows,

$$\min_{B,Z_0,\dots,Z_K,\alpha} \sum_{k=0}^{K} \alpha_k \|P_k - BZ_k\|_F^2 + \beta \sum_{k=0}^{K} \|Z_k\|_F^2, \tag{9}$$
$$s.t.: \quad B \in \{-2^N, \dots, -2^1, -2^0, 0, 2^0, 2^1, \dots, 2^N\}^{n\times d},$$
$$\sum_{i=1}^{K} \alpha_k = 1, \alpha_k \geq 0, \quad Z_k \in R^{d\times f},$$

where β is a regularization parameter with respect to auxiliary matrices Z_k, K is the total number of layers we consider in the model. A large layer number K may cause over-smoothing, while a small K cannot fully take advantage of network information. Due to the integer constraint over the representation matrix B, Eq. (9) is hard to solve and requires an efficient algorithm.

We present an efficient algorithm to iteratively optimize variables Z_k, B and α. The algorithm updates one parameter at a time and converges very fast.

Z **-Step**. Given B and α fixed, solve the sub-problem with respect to Z_k in Eq. (9). The loss function becomes,

$$\min_{Z_k} \sum_{k=0}^{K} \alpha_k \|P_k - BZ_k\|_F^2 + \beta \sum_{k=0}^{K} \|Z_k\|_F^2 \tag{10}$$
$$= \sum_{k=0}^{K} \alpha_k tr(Z_k^T B^T B Z_k) - \sum_{k=0}^{K} \alpha_k tr(P_k^T B Z_k) + \beta \sum_{k=0}^{K} tr(Z_k^T Z_k)$$

where $tr(.)$ is a trace norm. By calculating the derivative of Eq. (10), we derive a closed form solution as follows,

$$Z_k = (\alpha_k B^T B + \alpha I)^{-1} \alpha_k B^T P_k. \tag{11}$$

B **-Step**. It is difficult to solve B due to the discrete constraint. Given Z_k and α fixed, rewrite the objective function in Eq. (9) with respect to B as follows,

$$\min_{B} \sum_{k=0}^{K} \alpha_k \|P_k - BZ_k\|_F^2, \tag{12}$$
$$s.t.: \quad B \in \{-2^N, \dots, -2^1, -2^0, 0, 2^0, 2^1, \dots, 2^N\}^{n\times d}.$$

Due to the discrete constraint, the optimization problem above is NP-hard.

Here, we introduce an auxiliary variable Q to decouple the parameters in the objective and the discrete constraint. The idea is largely motivated by the successful application of ADMM in mixed integer programs [37]. Then, the objective function in Eq. (12) can be written as,

$$\min_{B,Q} \sum_{k=0}^{K} \alpha_k \|P_k - BZ_k\|_F^2 + Ic(Q), \tag{13}$$
$$s.t.: B = Q, \quad Q \in \{-2^N, \dots, -2^1, -2^0, 0, 2^0, 2^1, \dots, 2^N\}^{n\times d},$$

where I_c is defined as an indicator function. $I_C(Q) = 0$ if $Q \in \{-2^N, \dots, -2^0, 0, 2^0, \dots, 2^N\}$; otherwise, $I_C(Q) = +\infty$. The augmented Lagrange of Eq. (13), for parameter $\rho > 0$, can be formulated as,

$$L_\rho(B, Q, \lambda) = \sum_{k=0}^{K} \alpha_k \|P_k - BZ_k\|_F^2 + Ic(Q) + \frac{\rho}{2}\|B - Q + \lambda\|_F^2 - \frac{\rho}{2}\|\lambda\|_F^2. \tag{14}$$

Equation (14) can be solved by repeating the following iterations,

$$B^{t+1} := \arg\min_B L_\rho(B, Q^t, \lambda^t), \tag{15}$$

$$Q^{t+1} := \arg\min_Q L_\rho(B^{t+1}, Q, \lambda^t), \tag{16}$$

$$\lambda^{t+1} := \lambda^t + B^{t+1} - Q^{t+1}. \tag{17}$$

Benefit from the decoupling of ADMM, Eq. (15) is an unconstrained objective function. We can easily calculate the gradient with respect to matrix B,

$$\frac{\partial L_\rho(B, Q^t, \lambda^t)}{\partial B} = \sum_{k=0}^{K} 2\alpha_k B Z_k Z_k^T - \sum_{k=0}^{K} 2\alpha_k P_k Z_k^T + \rho(B - Q^t + \lambda^t). \tag{18}$$

The closed form solution is given in Eq. (18), where I is an identity matrix.

$$B^{t+1} = \left(\sum_{k=0}^{K} \alpha_k P_k Z_k^T + \rho Q^t - \rho \lambda^t \right)^{-1} \left(\sum_{k=0}^{K} \alpha_k Z_k Z_k^T + \rho I \right). \tag{19}$$

In order to solve Q, Eq. (16) can be rewritten as,

$$\min_{Q} \; \|Q - B^{t+1} - \lambda^t\|_F^2, \tag{20}$$
$$s.t.: \; Q \in \{-2^N, \dots, -2^1, -2^0, 0, 2^0, 2^1, \dots, 2^N\}^{n\times d}.$$

The optimal solution of Q is

$$Q = \prod_{0,\pm1,\pm2,\dots,\pm N} (B^{t+1} + \lambda^t), \tag{21}$$

where $\prod_{0,\pm1,\pm2,\dots,\pm N}$ denotes the projection of $(B^{t+1} + \lambda^t)$ with respect to the discrete set. After either the predefined iterations or the convergence of ADMM, Q is assigned to B and the algorithm continues to update α and Z_k.

α **-Step**. Given Z_k and B fixed, rewrite the objective function in Eq. (9) with respect to α as follows,

$$\min_{\alpha} \; \sum_{k=0}^{K} \alpha_k \|P_k - BZ_k\|_F^2, \tag{22}$$
$$s.t.: \sum_{k=0}^{K} \alpha_k = 1, \alpha_k \geq 0.$$

The optimal solution to α in Eq. (22) is $\alpha_k = 1$ corresponding to the minimum $\|P_k - BZ_k\|_F^2$ and $\alpha_k = 0$ otherwise. This solution means that only one order of P_k is finally selected. However, the solution of a single order does not meet our objective on exploring the complementary property of multiple orders to get a better embedding.

Alternatively, we use a trick based on the work [38] to avoid the single order solution. We set $\alpha_k^r \leftarrow \alpha_k$ with $r > 1$ and obtain the Lagrange of Eq. (23) as below,

$$L(\alpha, \eta) = \sum_{k=0}^{K} \alpha_k^r \|P_k - BZ_k\|_F^2 - \eta \left(\sum_{k=0}^{K} \alpha_k - 1 \right). \tag{23}$$

By setting the derivative of $L(\alpha, \eta)$ with respect to α_k and η to zero, we obtain

$$\begin{cases} \frac{\partial L(\alpha,\eta)}{\partial \alpha_k} & = \gamma \alpha_k^{r-1}(P_k - BC_i) - \eta = 0, \\ \frac{\partial L(\alpha,\eta)}{\partial \lambda} & = \sum_{k=0}^{K} \alpha_k - 1 = 0. \end{cases} \tag{24}$$

Then, α_k can be solved as follows,

$$\alpha_k = \frac{(1/\|P_k - BZ_k\|_F^2)^{1/(r-1)}}{\left(\sum_{k=0}^{K} 1/\|P_k - BZ_k\|_F^2\right)^{1/(r-1)}}. \tag{25}$$

Because $\|P_k - BZ_k\|_F^2 \geq 0$, we have $\alpha_k \geq 0$ naturally.

The details of the algorithm are given in Algorithm 2. Empirical studies show that the algorithm takes a few iterations to converge. For example, in our experiments B is iteratively computed and converges around $2-10$ iterations.

5. Performance analysis

In this section, we answer the question *why the Weisfeiler-Lehman proximity matrix P in* Definition 1 can effectively capture data dependence between node links and attributes. The proximity matrix P is built on the Weisfeiler-Lehman graph kernels [17], which essentially is an information aggregation process that aggregates neighboring nodes' information to a target node. The parameter k in matrix P controls the number of layers of neighboring nodes, and the parameter γ controls the degree of Laplacing smoothing. Next, we discuss the connections of matrix P with Weisfeiler-Lehman graph kernels, Laplacian smoothing and GCNs.

Algorithm 2 Low-Bit Quantization for Attributed Network Representation Learning.

```
Input: Graph structure $A$, node attribute $X$, dimension $d$, # of iterations $t_1$ and $t_2$, parameters $K$, $\beta$, $\rho$, $r$, $N$
Output: Low-bit representation matrix $B$
1: Initialize $Z_k$, $B$, $\lambda$ randomly, Let auxiliary matrix $Q = B$
2: Repeat until converge or reach $t_1$
3:   $Z_k$-Step: Calculate $Z_k$ by $Z_k = (\alpha_k B^T B + \alpha I)^{-1} \alpha_k B^T P_k$
4:   $B$-Step: Repeat until converge or reach $t_2$
5:     Update $B$ using $B^{t+1} = (\sum_{k=0}^{K} \alpha_k P_k Z_k^T + \rho Q^t - \rho\lambda^t)^{-1} (\sum_{k=0}^{K} \alpha_k Z_k Z_k^T + \rho I)$
6:     Update $Q$ using $Q = \prod_{0,\pm1,\pm2,\dots,\pm N} (B^{t+1} + \lambda^t)$
7:     Update $\lambda$ using $\lambda^{t+1} := \lambda^t + B^{t+1} - Q^{t+1}$
8:   $\alpha$-Step: Calculate $\alpha_k = \frac{(1/\|P_k - BZ_k\|_F^2)^{1/(r-1)}}{(\sum_{k=0}^{K} 1/\|P_k - BZ_k\|_F^2)^{1/(r-1)}}$.
9: return matrix $B$
```

5.1. Connection with Weisfeiler-Lehman graph kernels

The idea of aggregating information from neighboring nodes to a target node originated from the Weisfeiler-Lehman graph kernels [17], where the parameter k controls the layers of neighboring nodes joining the information aggregation. The original idea of the Weisfeiler-Lehman algorithm is to augment the node labels by the sorted set of node labels of their neighboring nodes, and then compress these augmented labels into new, short labels.

Theorem 1. *Let $k = 1$, $\gamma = 1$, and $P = (I - \gamma \tilde{D}^{-1}\tilde{L})^k X$, then P is a one-layer Weisfeiler-Lehman graph kernel.*

Proof. When $\gamma = 1, k = 1$, then $P = \tilde{D}^{-1}\tilde{A}X = P^{(1)}$. Let $h_i^{(k)}$ be the information of node v_i in the kth iteration, and N_i be the neighbors of v_i. Define a linear aggregation function, integrating neighboring nodes' information and the target node's information under the Weisfeiler-Lehman algorithm, we can obtain the following information propagation rule,

$$h_i^{(k)} = h_i^{(k-1)} + \sum_{j\in N_i} h_j^{(k-1)}. \tag{26}$$

Such an information propagation rule can be further rewritten into a compact matrix form as follows,

$$H^{(k)} = \tilde{A} H^{(k-1)}, \tag{27}$$

where $\tilde{A} = A + I$, which adds a self-loop to each node in the network. $\tilde{D} = \mathrm{diag}(\tilde{d}_1, \dots, \tilde{d}_n)$ is the degree matrix of $\tilde{A}$. Normalizing the matrix $\tilde{A}$ by its degree matrix $\tilde{D}$, we obtain

$$H^{(k)} = \tilde{D}^{-1}\tilde{A} H^{(k-1)}. \tag{28}$$

At the beginning of the aggregation, i.e., $k = 1$, $H^{(0)} = X$, then $H^{(1)} = P^{(1)}$. Thus $P^{(1)}$ is a one layer Weisfeiler-Lehman graph kernel. □

Apparently, Weisfeiler-Lehman graph kernel is essentially the process of aggregating neighboring nodes' information towards the target node.

5.2. Connection with laplacian smoothing

We analyze the relationship between matrix P and the Laplacian smoothing [20]. Given the representation h_i of node v_i, the laplacian smoothing can be considered as a new representation of a target node by using a weighted information aggregation of the target node itself and its neighbors, i.e.,

$$y = (1-\gamma) h_i + \gamma \sum_{j\in N_i} \frac{\tilde{a}_{ij}}{\tilde{d}_i} h_j, \quad \gamma \in [0, 1], \tag{29}$$

where $\tilde{a}_{ij}$ denotes an element of matrix $\tilde{A}$ (the adjacency matrix with self-loop), and $\tilde{d}_i$ is the degree of node v_i.

Theorem 2. *The Weisfeiler-Lehman proximity matrix* $P = (I - \gamma \tilde{D}^{-1}\tilde{L})^k * X$ *is a k-layer network Laplacian smoothing.*

Proof. Given an input matrix $H^{(k-1)}$, the new representation $H^{(k)}$ can be learned by a function with a parameter matrix $W^{(k-1)}$. By following Eq. (29) on each dimension of $H^{(k-1)}$, we have

$$h_i^{(k)} = [(1-\gamma)h_i^{(k-1)} + \gamma \sum_{j\in N_i} \frac{\tilde{a}_{ij}}{\tilde{d}_i} h_j^{(k-1)}]w_i^{(k-1)}. \tag{30}$$

Then, the compact matrix form can be written as follows,

$$\begin{aligned} H^{(k)} &= [(1-\gamma)H^{(k-1)} + \gamma \tilde{D}^{-1}\tilde{A}H^{(k-1)}]W^{(k-1)} \\ &= [H^{(k-1)} - \gamma \tilde{D}^{-1}(\tilde{D} - \tilde{A})H^{(k-1)}]W^{(k-1)} \\ &= (I - \gamma \tilde{D}^{-1}\tilde{L})H^{(k-1)}W^{(k-1)}. \end{aligned} \tag{31}$$

After repeated Laplacian smoothing calculations on attributes X, we obtain the update rule,

$$\begin{aligned} H^{(1)} &= (I - \gamma \tilde{D}^{-1}\tilde{L})XW^{(0)} \\ H^{(2)} &= (I - \gamma \tilde{D}^{-1}\tilde{L})H^{(1)}W^{(1)} \\ &= (I - \gamma \tilde{D}^{-1}\tilde{L})^2 XW^{(0)}W^{(1)} \\ &\cdots\cdots\cdots, \\ H^{(k)} &= (I - \gamma \tilde{D}^{-1}\tilde{L})^k XW^{(0)}W^{(1)}\ldots W^{(k-1)}. \end{aligned} \tag{32}$$

Let $W = W^{(0)}W^{(1)}\ldots W^{(k-1)}$, then $H^{(k)} = (I - \gamma \tilde{D}^{-1}\tilde{L})^k X \cdot W = P \cdot W$. This result shows that our Weisfeiler-Lehman proximity matrix P is a k-layer Laplacian smoothing. □

The k-layer Laplacian smoothing enables a node to incorporate deep information from neighbors. Our experiments also validate the method.

5.3. Connection with graph convolutional networks

The Weisfeiler-Lehman proximity matrix P defines a variant of graph convolutional networks.

Theorem 3. *Let* $\gamma = 1$, $W \in R^{m\times d}$, *then* $Z = P \cdot W$ *is a k-layer graph convolutional network.*

Proof. Let $\gamma = 1$, then the update rule is $H^{(k)} = (I - \tilde{D}^{-1}\tilde{L})H^{(k-1)}W^{k-1}$. By replacing the normalized Laplacian matrix $\tilde{D}^{-1}\tilde{L}$ with a symmetrically normalized Laplacian matrix $\tilde{D}^{-\frac{1}{2}}\tilde{L}D^{-\frac{1}{2}}H^{(k-1)}$, the update rule becomes,

$$H^{(k)} = \tilde{D}^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}H^{(k-1)}W^{(k-1)}. \tag{33}$$

This is exactly a one-layer graph convolutional network with a linear activation function. The update rule is an alternative of graph convolutional networks. Similarly, the k-layer graph convolutional network becomes $H^{(k)} = P \cdot W$. □

Theorem 3 indicates that the Weisfeiler-Lehman proximity matrix resembles the layer-wise information aggregation matrix in GCNs. While GCNs use a nonlinear activation function as the output, our algorithm employs a binary mapping (hashing) for embedding. Furthermore, we restrict $\gamma \in [0, 1]$ and gain more flexibility in incorporating a target node's information and its neighboring nodes' information.

6. Experiments

In this section, we evaluate the performance of BANE and LQANR on node classification and link prediction tasks. Node classification is popularly used to estimate the performance of network embedding methods. The link prediction task is a popular testbed for evaluating model efficiency.

Table 1
Dataset Description.

Datasets	# Nodes	# Edges x$\|E\|$	# Attributes	# Labels
Cora	2708	5429	1433	7
Citeseer	3327	4732	3703	6
Wiki	2405	17,981	4973	19
BlogCatalog	5196	171,743	8189	6

6.1. Experimental setup

Datasets. Four real-world attributed networks are used as testbed. They are popularly used in previous work [11,13]. Statistics of the datasets are summarized in Table 1.

- **Cora** contains 2708 machine learning papers from seven classes and 5429 links. The links are citation relationships between the documents. Each document is described by a binary vector of 1433 dimensions indicating the presence of the corresponding word.
- **Citeseer** contains 3312 publications from six classes and 4732 links. Similar to Cora, the links are citation relationships between the documents and each paper is described by a binary vector of 3703 dimensions.
- **Wiki** contains 2405 documents from 19 classes and 17,981 links. It is a co-occurrence network of words appearing in the first million bytes of Wikipedia dump.
- **BlogCatalog** is a blogger community, where users interact with each other and form a network. Users are allowed to generate keywords as a short description of their blogs. These keywords are severed as node attributes. Users also register their blogs under predefined categories, where are set as labels.

Baseline Methods. We compare our method with state-of-the-art methods. DeepWalk and node2vec use plain network structure for embedding. TADW, HSCA and LANE use both network structure and attributes.

- **DeepWalk** [6] involves language modeling techniques to analyze the truncated random walks on a graph. It embeds the walking tracks as sentences, and each vertex corresponds to a unique word.
- **Node2vec** [7] uses a biased random walk algorithm that can efficiently explore neighborhood architecture.
- **TADW** [11] incorporates textual features of nodes into network representation learning under the framework of matrix factorization. It factorizes network structure matrix into the product of three matrices by applying the inductive matrix completion. Then, it builds a unified matrix for network representations by concatenating the two decomposed matrix.
- **HSCA** [39] proposes to explicitly enforce the homophily property of connected nodes in the learned representation space so as to learn an effective network representation. By simultaneously augmenting homophily, structural context, and node attributes, the representations can better capture the interplay between node content information and network structure.
- **LANE** [13] models the structural proximities in the attributed network and labels based on pairwise similarities. Then, it jointly maps them into an identical embedding space via three relevant correlation projections.

Settings and Metrics. For fair comparisons, we set the embedding dimension $d = 100$ for all baselines. All the parameters are set to be the default values. For node classification, we randomly sample a portion of labeled nodes for training and the rest for testing. The training ratios range from 10% to 90% with an increasing step of 20%. We use 10-fold cross validation and repeat the testing for 10 times. The performance of all the methods are evaluated in

Table 2
Node Classification Results (d=100).

		Micro-F1 (%)					Macro-F1(%)				
Datasets	Ratios	10%	30%	50%	70%	90%	10%	30%	50%	70%	90%
Cora	DeepWalk	63.71	73.50	78.83	80.29	81.20	61.02	71.65	77.63	79.08	79.83
	Node2vec	67.10	77.30	81.22	82.68	83.52	66.56	76.50	80.14	81.61	82.28
	TADW	81.50	84.97	85.78	86.23	86.93	79.71	83.35	84.26	84.44	85.35
	HSCA	75.21	81.25	85.10	85.97	86.38	73.42	80.10	84.01	84.41	84.82
	LANE	67.21	70.15	73.38	76.91	80.81	66.39	68.49	72.67	75.32	79.95
	BANE	**81.88**	**85.32**	**86.35**	**87.06**	**88.30**	**80.23**	**84.26**	**85.19**	**85.76**	**87.11**
Citeseer	DeepWalk	43.24	49.06	54.41	56.16	56.31	40.57	45.65	49.33	50.32	49.17
	Node2vec	48.56	55.77	62.55	63.66	63.69	46.78	53.92	58.09	59.42	60.47
	TADW	69.38	71.48	72.18	72.75	72.84	61.80	64.62	65.83	66.54	67.03
	HSCA	69.47	71.54	72.61	73.66	73.96	61.62	64.80	65.98	66.70	67.21
	LANE	53.81	60.72	61.65	63.58	67.77	50.33	57.05	58.14	60.63	63.60
	BANE	**70.24**	**72.55**	**73.78**	**74.55**	**75.08**	**62.37**	**65.73**	**67.63**	**68.44**	**69.35**
Wiki	DeepWalk	56.95	61.44	63.71	65.33	66.55	45.36	48.37	50.63	52.28	52.81
	Node2vec	57.83	62.25	63.70	65.31	66.36	45.88	49.90	50.78	52.22	52.04
	TADW	67.04	71.25	72.36	73.19	74.33	46.76	51.45	52.76	53.07	53.22
	HSCA	68.75	71.87	73.35	74.71	77.05	46.30	52.03	53.57	54.57	54.90
	LANE	62.95	69.04	70.45	72.01	73.24	46.38	50.73	52.34	54.62	55.12
	BANE	**71.41**	**77.07**	**78.91**	**79.76**	**80.49**	**46.81**	**54.83**	**56.95**	**58.43**	**58.04**
BlogCatalog	DeepWalk	69.58	78.24	79.37	80.78	81.12	68.65	76.85	78.46	80.01	80.54
	Node2vec	72.43	79.05	82.36	83.40	84.95	71.54	77.27	80.81	80.95	82.03
	TADW	82.50	86.56	87.82	89.20	89.78	82.29	86.35	87.60	89.04	89.53
	HSCA	82.10	85.89	87.64	89.01	89.47	81.56	85.36	87.02	88.43	89.11
	LANE	85.23	88.56	**89.64**	**89.89**	**90.08**	85.05	88.27	**89.35**	**89.59**	**89.95**
	BANE	**86.21**	**89.04**	89.55	89.85	89.88	**85.71**	**88.74**	89.30	89.55	89.59

terms of Micro-F1 and Macro-F1. For link prediction, we randomly sample 90% neighbors of each node for training and the rest for testing. We also repeat the recommendation procedure 10 times and evaluate the performance of all the methods in terms of AUC, which represents the probability that a randomly selected unobserved link is more similar than a randomly selected non-existent one.

Table 3
Link Prediction Results on the Four Datasets.

	Cora	Wiki	BlogCatalog	Citeseer
DeepWalk	83.10	80.46	**63.29**	80.56
Node2vec	81.59	78.91	60.31	80.24
TADW	89.77	89.86	60.40	93.80
HSCA	87.01	87.45	60.35	93.50
LANE	86.07	77.21	58.97	77.18
BANE	**93.50**	**90.90**	61.44	**95.59**

6.2. Node classification results

For all the datasets, we reduce the dimension of node attributes to 200 by using SVD decomposition on X. The preprocessing reduces the number of parameters in factorization. We use SVM for node classification. The embedding dimension d is set to 100 and the regularization parameter α is set to 0.001.

Table 2 lists the results of node classification. We summarize as follows. **First**, BANE significantly outperforms DeepWalk and node2vec on all the four datasets with respect to both Micro-F1 and Macro-F1 under five different training ratios from 10% to 90%. The results indicate that combining node links and attributes can substantially improve embedding accuracy. **Second**, BANE outperforms all the structure and attributes embedding algorithms on Cora, Citeseer and Wiki in terms of both Micro-F1 and Macro-F1 under different training ratios. The classification results are significantly higher than the other baseline methods by 3% on the Wiki dataset. The accuracy is marginally lower than LANE on the BlogCatalog dataset when training ratio increasing from 50% to 90%. The results indicate the effectiveness and robustness of BANE to handle both structure and attribute information. **Third**, BANE is the only binarized representation method. The results show that binary representation does not necessarily lead to accuracy loss. In fact, it may avoid the trap of over-fitting. **Fourth**, BANE performs stably better than all the other benchmarks when the training ratio is low. For example, the Micro-F1 result on Wiki with 10% training reaches 0.714, which is much higher than the second highest 0.687 from HSCA. The accuracy results of most baseline methods drop rapidly when the training ratio decreases, because their node representations are noisy and inconsistent from training to testing. Instead, BANE learns jointly from node links and attributes by using high layer Weisfeiler-Lehman matrix. Thus, the results of BANE contain less noise and are more robust.

6.3. Link prediction results

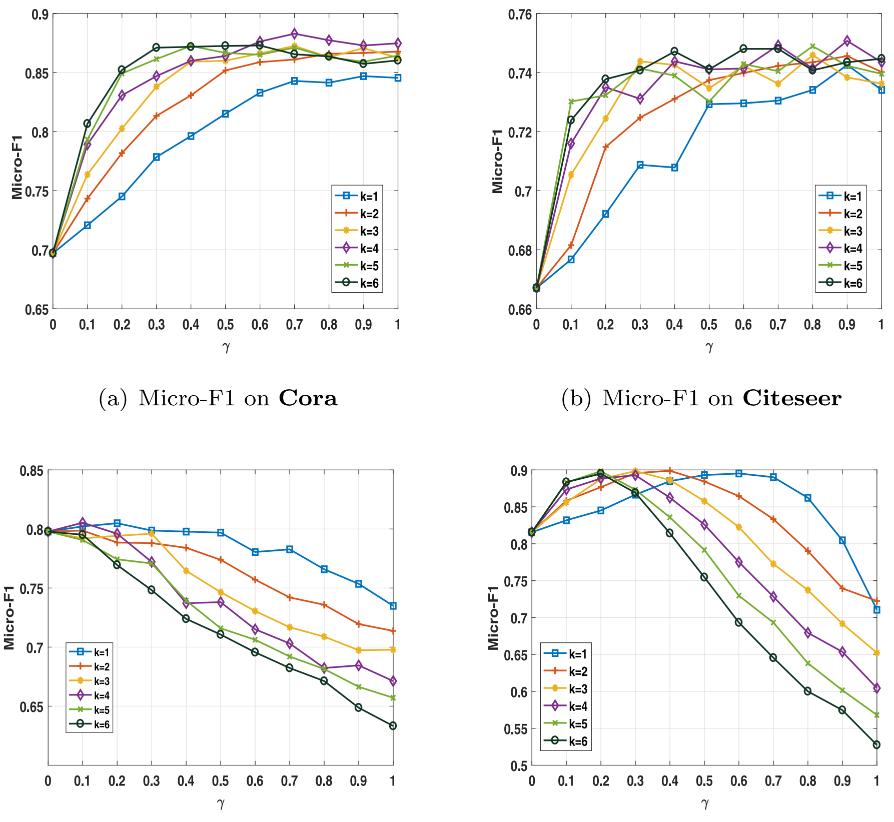
Table 3 shows the results of link predictions on the four datasets. We randomly sample 90% neighbors of each node for training and the rest for testing. We measure the performance by AUC. The observations are as follows. **First**, our method significantly outperforms baselines on Cora, Wiki and Citeseer. The AUC scores reach 93.5% on Cora and 95.6% on Citeseer. **Second**, converting real-valued numbers into binary representation improves the link prediction accuracy. This is because the binary representation can alleviate the over-fitting problem and it is more intuitional to express the Yes/No option for recommendation. Moreover, binary representation can replace the dot-product similarity computation with bit-wise Hamming distance. Thus, the speed of training can be significantly improved.

6.4. Parameter study

We test the three parameters, tradeoff parameter γ, the layer of aggregation k, and the embedding dimension d.

6.4.1. Tradeoff parameter γ and the layer parameter k

We test parameter k by varying its value from 1 to 6, and γ from 0 to 1 with a stepsize of 0.1. The training ration is set to



(a) Micro-F1 on **Cora** (b) Micro-F1 on **Citeseer** (c) Micro-F1 on **Wiki** (d) Micro-F1 on **BlogCatalog**

Fig. 2. Node classification results in terms of *Micro-F1* with respect to parameters γ and k. (a) Cora, (b) Citeseer, (c) Wiki, and (d) BlogCatalog.

0.9. The Micro-F1 results on the four datasets are shown in Fig. 2. From the figure, we have the following observations. **First**, the Cora and Citeseer datasets reflect similar patterns. The classification accuracy increases with γ and achieves the highest value when γ reaches 0.6 to 0.8. After that, the accuracy gradually drops. When $\gamma = 0$, we only use the node attributes, so the classification results are the lowest. For example, 0.7 on Cora and 0.67 on Citeseer. Generally, when k falls into the range of 4 to 6, we obtain the best results, which shows that the number of layers of node neighbors is important. **Second**, on Wiki the best result is observed when $k = 1$. Increase k results in lower accuracy results. When γ between 0 and 0.3, we obtain the best performance. Then, the accuracy drops with increasing γ. **Third**, on BlogCatalog when γ is less than 0.4, a larger k obtains a better accuracy result. However, when γ is large, a small k shows better accuracy and the overall result drops quickly.

6.4.2. Node embedding dimension d

We test the embedding dimension d from 20 to 300 with a stepsize of 20. The node classification results on the datasets are shown in Fig. 3a. We can observe that the performance of network embedding improves with d increasing to 160. Then, the results become stable when code length continuously increases to 300.

The link prediction results with varying embedding dimension d are shown in Fig. 3b. With the increasing of embedding dimension, the AUC scores increase rapidly to top when the dimension ranges from 60 to 100. Then, the results remain stable when increasing d, until the dimension reaches 260. The results show that the binary representation can deliver competitive link prediction results even though the embedding dimension is low.

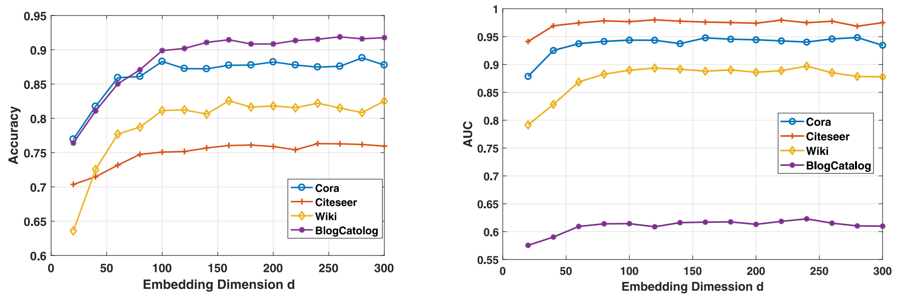
6.5. Binarized vs real-valued Weisfeiler-Lehman matrix factorization

We also compare the original binary BANE model with its real-valued variant (BANE-r for short) by removing the binary constraint in Eq. (1).

The overall procedure is the same as BANE. We can easily get the closed form solution for both B and Z at each update. Table 4 shows the classification accuracy on the four datasets with training ratios range from 10% to 90% and the embedding dimension $d = 100$.

When comparing BANE-r with BANE, we can observe that the real-valued embedding receives slightly higher accuracy results than binary embedding on Cora, Citeseer and Blogcatalog when the training ratios increase from 30% to 90%. Nevertheless, if the training ratio is as low as 10%, the binary embedding beats the real-valued embedding. For example, the classification Micro-F1 on the Citeseer dataset with 10% training ratio is 70.24 of BANE versus 67.91 of BANE-r. On the Wiki dataset, the Micro-F1 scores of BANE are higher than that of BANE-r at all training ratios, but the Macro-F1 scores are lower.

From the comparison results, we can conclude that the binary embedding BANE obtains competitive embedding results as real-valued embedding, especially when the training ratio is low. The reasons may be as follows: **First**, binary constraints can be viewed as adding non-linear features to the linear matrix factorization, so linear classification on binary codes is equivalent to learning a



(a) Classification *w.r.t.* Micro-F1 (b) Link Prediction Results

Fig. 3. Comparisons with respect to dimensions (bits) *d*. (a) Classification Results; (b) Link Prediction Results.

Table 4
Node classification results between real-valued embedding and binarized embedding.

		Micro-F1					Macro-F1				
Datasets	Models	10%	30%	50%	70%	90%	10%	30%	50%	70%	90%
Cora	BANE-r	80.94	**86.70**	**87.56**	**87.87**	**89.00**	79.75	**85.64**	**86.46**	**86.61**	**87.92**
	BANE	**81.88**	85.32	86.35	87.06	88.30	80.23	84.26	85.19	85.76	87.11
Citeseer	BANE-r	67.91	**74.15**	**75.17**	**75.82**	**76.01**	61.77	**69.11**	**70.47**	**71.18**	**71.78**
	BANE	**70.24**	72.55	73.78	74.55	75.08	**62.37**	65.73	67.63	68.44	69.35
Wiki	BANE-r	63.82	71.04	74.76	75.65	77.44	**48.71**	**60.55**	**65.53**	**67.20**	**72.21**
	BANE	**71.41**	**77.07**	**78.91**	**79.76**	**80.49**	46.81	54.83	56.95	58.43	58.04
Blogcatalog	BANE-r	82.75	**89.07**	**90.39**	**91.15**	**92.02**	82.47	**88.90**	**90.23**	**91.01**	**91.83**
	BANE	**86.21**	89.04	89.55	89.95	89.88	**85.71**	88.74	89.30	89.75	89.59

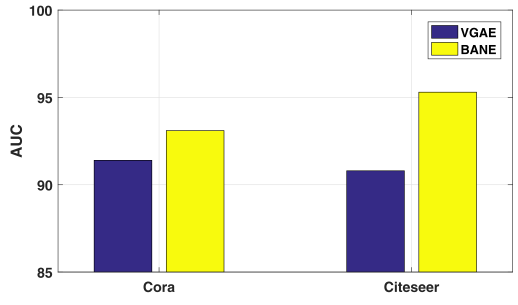


Fig. 4. Comparison BANE with VGAE for Link Prediction.

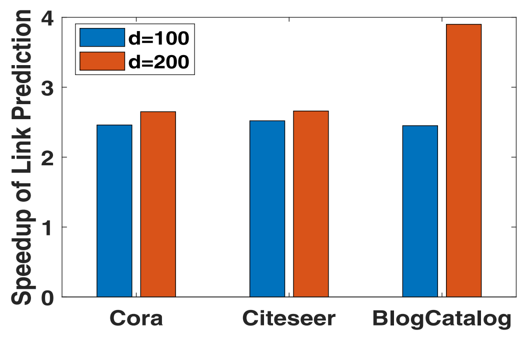


Fig. 5. Speedup of link prediction by LQANR.

nonlinear classifier on the original data. **Second**, the limited two values of binary codes can alleviate the possible over-fitting problem and obtain encouraging results, even when the training ratio is very small.

6.6. Comparison with GCNs

We also compare our algorithm with a variant of GCNs, i.e., the Variational Graph Auto-Encoders (VGAE) [40], which learns a GCN as an autoencoder for link prediction. VGAE uses 85%, 5%, 10% edges for training, validation, and testing. To simulate the settings of VGAE as close as possible, we randomly select 85% edges for training and 10% for testing for BANE. We repeat the process 10 times and calculate their average. The results are shown in Fig. 4.

The results show that BANE beats VGAE for the given link prediction task. This is because the tradeoff parameter γ in the Weisfeiler-Lehman proximity matrix provides extra flexibility to model data dependence between node links and attributes. However, finding the best parameter γ to fully unleash the power of the BANE model is not a trivial work, we will consider to use automated machine learning to search the best parameter in the future work.

6.7. Low-bit quantization representation (LQANR)

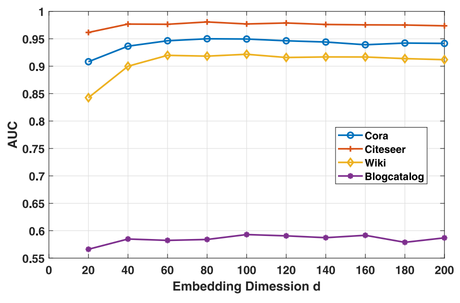
6.7.1. Speedup of LQANR

As shown in Fig. 5, low-bit representation also accelerates link prediction speed by replacing the dot-product similarity computation with bit-wise Hamming distance. The figure shows the speedup of 100 and 200 dimension nearest search via hamming distance compared to dot-product. The results show that large-sized networks gain significant speedup.

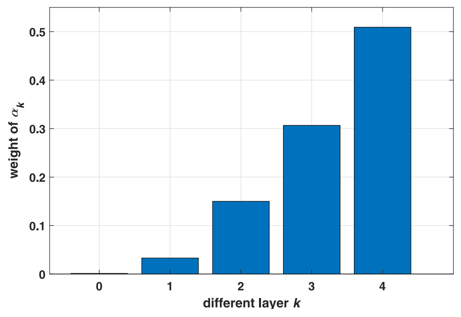
6.7.2. Parameter study

We test four parameters, bit-width decided by N, embedding dimension d, proximity matrix maximum order K and weights of P_k impacted by r.

First, we study different kinds of bit-width for discrete node embedding. We test binary quantization, ternary quantization, one-bit shift quantization and two-bits shift quantization. The node classification result on the Cora dataset with respect to different bit-width values is shown in Table 5. We can observe that the classification accuracy increases with bit-width. For example, the



(a) Link Prediction *w.r.t.* d



(b) Weights of α_k on Citeseer

Fig. 6. Parameter studies *w.r.t.* embedding d and α_k .

Table 5
Node classification (Micro-F1) *w.r.t.* bit-width.

Bit-width	10%	30%	50%	70%	90%
(1,-1)	80.16	84.13	84.9	85.38	86.33
(-1,0,1)	83.00	85.91	86.74	87.27	87.70
(-2,-1,0,1,2)	83.40	86.46	87.22	87.68	88.33
(-4, ..., 4)	83.51	86.53	87.34	87.70	88.85

Table 6
Node classification (Micro-F1) *w.r.t.* K on Cora.

Order K	10%	30%	50%	70%	90%
K=2	82.29	85.46	85.97	86.12	86.37
K=3	82.69	85.58	86.44	86.90	87.56
K=4	82.67	85.62	86.62	**87.36**	87.44
K=5	**83.00**	**85.91**	**86.74**	87.27	87.70
K=6	82.63	85.70	86.43	87.03	**88.11**

Micro-F1 score increases from 80.16 when B is represented by {-1,1} to 83.51 when B is represented by {-4,-2,...,2,4}.

Second, we test the embedding dimension d from 20 to 200 with a stepsize of 20. The link prediction results are shown in Fig. 6a. We can observe that the performance of network embedding improves with d increasing from 20 to 100, and then remains stable when code length continuously increases. On the BlogCatalog dataset, the link prediction results are the lowest. This is because BlogCatolog contains more complicated structure and attribute information than the other datasets.

Third, we test node classification with different K. The results on Cora with K arranging from 2 to 6 are shown in Table 6. It shows that $K = 5$ is the best choice for Cora in many cases. The reason is that when K is too large, it can cause over-smoothing for node attributes. However, small K cannot fully propagate node attribute information in networks (Table 7).

Last, we test weights of P_k. Different k-hop matrices capture different steps of neighboring node attributes. The layer-wise weights α_k are impacted by r. We test on different datasets and find the best node classification results with the best parameter r. r is usually between 1 and 10 for the tested datasets. We plot the distribution of α_k on Citeseer with $K = 5$ and $r = 1.6$. From Fig. 6b, we can observe that the higher order P_k contributes heavier weights, which means combining more layers leads to better results.

6.7.3. Comparison

We compare LQANR with BANE on Cora, Citeseer, and BlogCatalog with respect to Micro-F1 and Macro-F1 under different training ratios. The results validate that LQANR performs slightly better than BANE, which is also very effectiveness and robustness. Moreover, LQANR can obtain any low-bit embedding, which is more flexible and accurate to capture attributed network information.

7. Conclusions

In this paper we study a new problem of *discrete embedding for attributed networks*, where we define a new *Weisfier-Lehman proximity matrix* to jointly encode data dependence between node links and attributes. Based on the new proximity matrix, we formulate a new binarized Weisfier-Lehman matrix factorization model to obtain binary node representation. Moreover, we extend the binary representation learning to even low-bit quantization learning for attributed networks. Theoretical studies show the close connections of the new proximity matrix with Weisfier-Lehman graph kernels, network smoothing, and graph convolutional networks (GCNs). Empirical results also validate the promising results compared with popular network embedding models. In the future, we will consider to use the automated machine learning methods (AutoML) to search the best parameters for the proposed BANE model. We expect that the Weisfier-Lehman proximity matrix can precisely capture data dependence between node links and attributes for any given large networks with minimal human efforts.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

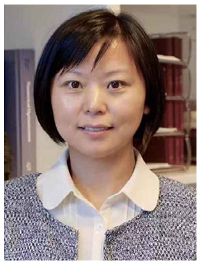
Table 7
Node Classification Results (d=100).

		Micro-F1 (%)					Macro-F1(%)				
Datasets	Models	10%	30%	50%	70%	90%	10%	30%	50%	70%	90%
Cora	BANE	81.88	85.32	86.35	87.06	88.30	80.23	84.26	85.19	85.76	87.11
	LQANR	83.00	85.91	86.74	87.27	88.21	81.79	84.79	85.57	85.95	86.95
Citeseer	BANE	70.24	72.55	73.78	74.55	75.08	62.37	65.73	67.63	68.44	69.35
	LQANR	70.41	72.73	73.80	74.67	75.20	62.94	66.11	67.80	68.72	69.62
BlogCatalog	BANE	86.21	89.04	89.55	89.85	89.88	85.71	88.74	89.30	89.55	89.59
	LQANR	86.24	89.29	89.95	90.44	90.75	85.91	89.10	89.79	90.31	90.55

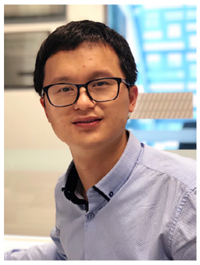
References

[1] X. Pan, H.-B. Shen, Scoring disease-microrna associations by integrating disease hierarchy into graph convolutional networks, Pattern Recognit. 105 (2020) 107385.
[2] W. Zan, C. Zhou, H. Yang, Y. Hu, L. Guo, iWalk: interest-aware random walk for network embedding, in: 2018 International Joint Conference on Neural Networks (IJCNN), IEEE, 2018, pp. 1–8.
[3] H. Yang, L. Chen, M. Lei, L. Niu, C. Zhou, P. Zhang, Discrete embedding for latent networks, in: C. Bessiere (Ed.), Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20, International Joint Conferences on Artificial Intelligence Organization, 2020, pp. 1223–1229. Main track
[4] F. Xiong, X. Wang, S. Pan, H. Yang, H. Wang, C. Zhang, Social recommendation with evolutionary opinion dynamics, IEEE Trans. Syst. Man Cybern. (2018).
[5] P. Wang, P. Zhang, C. Zhou, Z. Li, H. Yang, Hierarchical evolving Dirichlet processes for modeling nonlinear evolutionary traces in temporal data, Data Min. Knowl. Discov. 31 (1) (2017) 32–64.
[6] B. Perozzi, R. Al-Rfou, S. Skiena, DeepWalk: online learning of social representations, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2014, pp. 701–710.
[7] A. Grover, J. Leskovec, node2vec: scalable feature learning for networks, in: KDD, ACM, 2016, pp. 855–864.
[8] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, Q. Mei, Line: large-scale information network embedding, in: WWW, 2015, pp. 1067–1077.
[9] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, Y. Bengio, Graph attention networks, in: International Conference on Learning Representations, 2018.
[10] D. Wang, P. Cui, W. Zhu, Structural deep network embedding, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1225–1234.
[11] C. Yang, Z. Liu, D. Zhao, M. Sun, E.Y. Chang, Network representation learning with rich text information, in: IJCAI, 2015, pp. 2111–2117.
[12] X. Huang, J. Li, X. Hu, Accelerated attributed network embedding, in: SDM, SIAM, 2017, pp. 633–641.
[13] X. Huang, J. Li, X. Hu, Label informed attributed network embedding, in: WSDM, ACM, 2017, pp. 731–739.
[14] J. Wang, T. Zhang, N. Sebe, H.T. Shen, et al., A survey on learning to hash, TPAMI (2017).
[15] W. Wu, B. Li, L. Chen, X. Zhu, C. Zhang, *k*-ary tree hashing for fast graph classification, TKDE (2017).
[16] Y. Shi, M. Lei, H. Yang, L. Niu, Diffusion network embedding, Pattern Recognit. 88 (2019) 518–531.
[17] N. Shervashidze, P. Schweitzer, E.J.v. Leeuwen, K. Mehlhorn, K.M. Borgwardt, Weisfeiler-Lehman graph kernels, JMLR 12 (Sep) (2011) 2539–2561.
[18] F. Shen, C. Shen, W. Liu, H. Tao Shen, Supervised discrete hashing, in: CVPR, 2015, pp. 37–45.
[19] S.V.N. Vishwanathan, N.N. Schraudolph, R. Kondor, K.M. Borgwardt, Graph kernels, JMLR 11 (Apr) (2010) 1201–1242.
[20] G. Taubin, A signal processing approach to fair surface design, in: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, ACM, 1995, pp. 351–358.
[21] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, et al., Recent advances in convolutional neural networks, Pattern Recognit. 77 (2018) 354–377.
[22] H. Yang, S. Pan, P. Zhang, L. Chen, D. Lian, C. Zhang, Binarized attributed network embedding, in: 2018 IEEE International Conference on Data Mining (ICDM), IEEE, 2018, pp. 1476–1481.
[23] S. Yan, D. Xu, B. Zhang, H.-J. Zhang, Q. Yang, S. Lin, Graph embedding and extensions: a general framework for dimensionality reduction, TPAMI 29 (1) (2007) 40–51.
[24] S. Chang, W. Han, J. Tang, G.-J. Qi, C.C. Aggarwal, T.S. Huang, Heterogeneous network embedding via deep architectures, in: KDD, ACM, 2015, pp. 119–128.
[25] C. Li, S. Wang, D. Yang, Z. Li, Y. Yang, X. Zhang, J. Zhou, PPNE: property preserving network embedding, in: International Conference on Database Systems for Advanced Applications, Springer, 2017, pp. 163–179.
[26] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, H. Liu, Attributed network embedding for learning in a dynamic environment, in: CIKM, ACM, 2017, pp. 387–396.
[27] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S.Y. Philip, A comprehensive survey on graph neural networks, IEEE Trans. Neural Netw. Learn. Syst. (2020).
[28] W.L. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, NIPS, 2017.
[29] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks? in: International Conference on Learning Representations, 2018.
[30] Y. Gao, H. Yang, P. Zhang, C. Zhou, Y. Hu, Graph neural architecture search, in: C. Bessiere (Ed.), Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20, International Joint Conferences on Artificial Intelligence Organization, 2020, pp. 1403–1409. Main track
[31] W. Liu, C. Mu, S. Kumar, S.-F. Chang, Discrete graph hashing, in: NIPS, 2014, pp. 3419–3427.
[32] F. Shen, Y. Xu, L. Liu, Y. Yang, Z. Huang, H.T. Shen, Unsupervised deep hashing with similarity-adaptive and discrete optimization, TPAMI (2018).
[33] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, Y. Bengio, Binarized neural networks, in: Proceedings of the 30th International Conference on Neural Information Processing Systems, 2016, pp. 4114–4122.
[34] X. Shen, S. Pan, W. Liu, Y. Ong, Q. Sun, Discrete network embedding, in: IJCAI, 2018, pp. 3549–3555.
[35] W. Wu, B. Li, L. Chen, C. Zhang, Efficient attributed network embedding via recursive randomized hashing, in: IJCAI-18, 2018, pp. 2861–2867.
[36] M. Denil, B. Shakibi, L. Dinh, M.A. Ranzato, N. de Freitas, Predicting parameters in deep learning, Advances in Neural Information Processing Systems, vol. 26, 2013.
[37] C. Leng, H. Li, S. Zhu, R. Jin, Extremely low bit neural network: Squeeze the last bit out with ADMM, AAAI, 2017.
[38] T. Xia, D. Tao, T. Mei, Y. Zhang, Multiview spectral embedding, TSMC-B 40 (6) (2010) 1438–1446.
[39] D. Zhang, J. Yin, X. Zhu, C. Zhang, Homophily, structure, and content augmented network representation learning, in: ICDM, IEEE, 2016, pp. 609–618.
[40] T.N. Kipf, M. Welling, Variational graph auto-encoders, NIPS Workshop on Bayesian Deep Learning, 2016.

Hong Yang is a Senior Postdoctoral Scientist with Faculty of Medicine and Health, the University of Sydney, Australia. She received PhD from the Australian Artificial Intelligence Institute (AAII), University of Technology Sydney, Australia. She obtained her Master degree from University of Chinese Academy of Sciences, and her Bachelor degree from Xidian University. Her research interests include graph data analytics and medical image processing. She has published 16 research papers in major data mining journals and conferences.

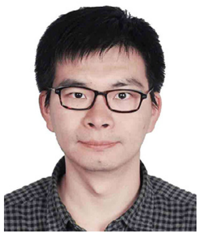


Ling Chen is an Associate Professor with the Australian Artificial Intelligence Institute (AAII), University of Technology Sydney, Australia. She received PhD from Nanyang Technological University, Singapore. Her research interests include data mining and machine learning, especially on structured data such as graph data and spatio-temporal data. She also works on social network and social media analysis and applications. Her papers appear in major conferences and journals including KDD, IJCAI, IEEE TNNLS and IEEE TKDE.



Shirui Pan received PhD in computer science from the University of Technology Sydney, Australia. He is currently a lecturer with the Faculty of Information Technology, Monash University, Australia. Prior to this, he was a Lecturer with the School of Software, University of Technology Sydney. His research interests include data mining and machine learning. To date, Dr Pan has published over 60 research papers in top-tier journals and conferences, including the IEEE Transactions on Neural Networks and Learning Systems (TNNLS), IEEE Transactions on Knowledge and Data Engineering (TKDE), IEEE Transactions on Cybernetics (TCYB), KDD, AAAI, and CVPR.

Haishuai Wang is a Visiting Assistant Professor of Biomedical Informatics at Harvard University, and a tenure-track Assistant Professor of Computer Science at Fairfield University. Prior to that, he was a Research Fellow at Harvard University and Postdoc Fellow at Washington University in St. Louis. He completed PhD in Computer Science from the University of Technology Sydney and Washington University in St. Louis. His research areas include data mining, machine learning and health informatics. His research focuses on developing machine learning algorithms to analyze complex data, ranging from clinical data, time series data, biological data, to large-scale networks.



Peng Zhang is a Professor with Guangzhou University, China. He received PhD from University of the Chinese Academy of Sciences. He was a lecturer with University of Technology Sydney, an associate professor with Chinese Academy of Sciences, and a senior staff engineer with the Alibaba Group. He has been researching into data mining, data streams, and social network analysis, with over 150 publications in TPAMI, TKDE, TNNLS, KDD, WWW, ICDM, AAAI, IJCAI, etc. He has served on many program committees of international conferences, including PC member for KDD, ICLR, ICML, NeurIPS, IJCAI, and AAAI conferences. He also served as the founding editorial board of Springer Annals of Data Science, and Springer Journal of Big Data.