Discrete Network Embedding

Xiaobo Shen[‡*], Shirui Pan[♯*], Weiwei Liu[♮], Yew-Soon Ong[‡], Quan-Sen Sun[§]
[‡]School of Computer Science and Engineering, Nanyang Technological University
[♯]Centre for Artificial Intelligence, FEIT, University of Technology Sydney
[♮]School of Computer Science and Engineering, The University of New South Wales
[§]School of Computer and Engineering, Nanjing University of Science and Technology
{njust.shenxiaobo,shiruipan,liuweiwei863}@gmail.com, asysong@ntu.edu.sg, sunquansen@njust.edu.cn

Abstract

Network embedding aims to seek low-dimensional vector representations for network nodes, by preserving the network structure. The network embedding is typically represented in continuous vector, which imposes formidable challenges in storage and computation costs, particularly in large-scale applications. To address the issue, this paper proposes a novel discrete network embedding (DNE) for more compact representations. In particular, DNE learns short binary codes to represent each node. The Hamming similarity between two binary embeddings is then employed to well approximate the ground-truth similarity. A novel discrete multi-class classifier is also developed to expedite classification. Moreover, we propose to jointly learn the discrete embedding and classifier within a unified framework to improve the compactness and discrimination of network embedding. Extensive experiments on node classification consistently demonstrate that DNE exhibits lower storage and computational complexity than state-of-the-art network embedding methods, while obtains competitive classification results.

1 Introduction

Networks are natural tools to model pairwise relationships and inter-dependence among data in a variety of applications including communication networks, social networks, citation networks, etc. It is well recognized that networks are highly complicated and sparse which impose significant challenges in many network analytic tasks, such as community detection, node classification, and link prediction. For years researchers have made efforts to solve each of these tasks separately until recently *network embedding* emerged as a general way for many of these problems.

Network embedding aims to transfer the graph representation into continuous feature values, where each node corresponding to a low dimensional vector. The key idea behind is to preserve the structure relationship of the network so that nodes with links are close to each other in the vector space. After learning the new representations, any network analytic tasks can be easily carried out by using off-the-shelf machine learning algorithms based on the new representations. Due to its effectiveness in many tasks, such as graph clustering [Tian *et al.*, 2014], link prediction [Dai *et al.*, 2017], and personalized recommendation [Zhang *et al.*, 2017], network embedding has attracted increasing attention in recent years.

Network embedding can be roughly categorized into three groups: (1) random walk-based algorithms, (2) matrix factorization-based methods, and (3) deep learning-based approaches. The random walk-based algorithms generate a large number of truncated random paths over a network and preserve the neighborhood relationship of nodes in the paths. By borrowing the idea of word representation in NLP, DeepWalk [Perozzi *et al.*, 2014] becomes the first elegant framework for network embedding in an online manner. Node2Vec [Grover and Leskovec, 2016] further exploits the biased random walks to capture the global structure information. Other methods either learn asymmetric proximity [Zhou *et al.*, 2017] or capture multiple sources of information in the network to learn network embedding [Pan *et al.*, 2016].

Matrix factorization-based aims to decompose a matrix into low dimensional latent factors which can be considered as the new representation of nodes. They vary in the objective functions. Early approaches such as modularity maximization and eigen-decomposition, aim to learn community indicative functions, while TADW [Yang *et al.*, 2015] exploits two-step of the transition probability matrix via inductive matrix factorization. Recently, it has been proven in [Qiu *et al.*, 2018] that many approaches, including LINE [Tang *et al.*, 2015b], PTE [Tang *et al.*, 2015a], DeepWalk and Node2Vec, are equivalent to matrix factorization methods. Recently, deep learning approaches [Cao *et al.*, 2016; Wang *et al.*, 2016] have been proposed to learn non-linear representations for nodes.

Despite the success of current network embedding approaches, the network embedding is learned in a continuous space, which in practice imposes severe challenges in the storage and computation cost, particular on large-scale datasets. Therefore, it is imperative to develop a compact

*indicates equal contribution.

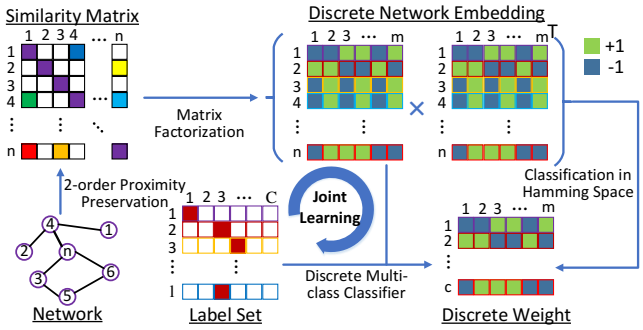


Figure 1: The Overview of the proposed DNE. DNE jointly learns the discrete network embedding and discrete multi-class classifier. The classification is conducted in the Hamming space.

network embedding framework, which has been less well studied and remains a very challenging area.

Binary code learning [Wang *et al.*, 2018] has gained increased interests in many large-scale applications. It encodes the original high-dimensional data into a set of short binary codes with similarity preservation. Thus the calculation and storage in Hamming space are both at very low cost. This triggers us to propose a novel discrete network embedding (DNE). DNE aims to learn the short binary code to represent each node. The advantages of DNE over conventional embedding methods lie in the low cost of computation and storage. As shown in the experiments, DNE reduces the storage of the embedding and model size to 64 times on the three datasets; meanwhile, it dramatically improves the efficiency of training and classification. The overview of the proposed DNE is illustrated in Figure 1. The main contributions of this work are summarized below:

- We propose a novel discrete network embedding (DNE) for compact representations. DNE represents each node using binary codes, which can significantly reduce the computational and storage costs. To the best of our knowledge, DNE is the first work that learns a discrete representation for networks.
- DNE leverages Hamming similarity between two binary embeddings to well approximate the ground-truth similarity. To improve discrimination of embedding, DNE jointly learns the discrete embedding and classifier. The proposed discrete multi-class classifier is able to expedite classification.
- Extensive experiments demonstrate that DNE achieves lower storage and computational complexity than state-of-the-art network embedding methods, while achieving comparable accuracy.

The rest of this paper is arranged as follows. In Sections 2 and 3, we discuss related work and background. Section 4 presents the proposed DNE method. The experiment is outlined in Section 5. The conclusion is drawn in Section 6.

2 Related Work

Network Embedding Learning Network embedding has become an effective method for many network analytic tasks, including node classification, clustering, and anomaly detection. Regarding the information they exploit for learning the representation, network embedding can also be distinguished into two groups (1) structure-preserving algorithms, and (2) content augmented algorithms.

Structure-preserving algorithms take in a network as the input and aim to preserve the structure relationship between nodes. While capturing the structure relations, they typically employ random walks [Perozzi *et al.*, 2014; Grover and Leskovec, 2016], matrix factorization [Tang and Liu, 2009; Zhou *et al.*, 2017], or deep learning approaches [Cao *et al.*, 2016; Wang *et al.*, 2016].

Content augmented algorithms assume that the node content is available on each node and aim to embed both network structure and node content into a low dimensional space. These algorithms include TADW [Yang *et al.*, 2015], TriDNR [Pan *et al.*, 2016], DANE [Li *et al.*, 2017], MGAE [Wang *et al.*, 2017], GraphSAGE [Hamilton *et al.*, 2017], and the graph convolutional network (GCN) [Kipf and Welling, 2016], to name a few.

Unfortunately, all these works learn the continuous representations, which impose serious challenges in the storage and computation cost on large-scale datasets. To solve this issue, this work aims to learn a discrete network embedding.

Binary Code Learning Binary coding technique, also known as Hashing [Wang *et al.*, 2018], has become a widely-studied solution to approximate nearest neighbor search, due to its great gains in storage and computation. The basic idea of hashing [Wang *et al.*, 2018; Weiss *et al.*, 2009; Gong *et al.*, 2013; Shen *et al.*, 2017b] is to map high-dimensional data into a low-dimensional discrete code space called Hamming space, while preserving similarity structure in the original space. Accordingly, each data point is represented by a short binary code called hash code consisting of a sequence of bits.

Existing hashing methods [Wang *et al.*, 2018] learn binary code from images, texts, videos and retrieval tasks. This work first investigates the binary coding technique in network embedding.

3 Notation and Background

We use bold uppercase, lowercase letters, and italic letters to denote matrices, vectors, and scalars, respectively. For any matrix $\mathbf{A}$, A_{ij} denotes the (i,j)-element of $\mathbf{A}$, and $\mathrm{Tr}(\mathbf{A})$ is the trace of $\mathbf{A}$ if $\mathbf{A}$ is square, $\mathbf{A}^\top$ denotes the transposed matrix of $\mathbf{A}$. Let $\mathbf{I}_m$ represent the identity matrix in $\mathbb{R}^{m\times m}$. We denote $\|\cdot\|_F$ as the Frobenius norm, and $\mathrm{sgn}(\cdot) : \mathbb{R} \to \{+1, -1\}$ as the round-off function.

Let $G = (V, E)$ be a network with n interconnected nodes and e edges. It can be also represented by the adjacency matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, where A_{ij} represents the link information from node i and node j, and it is defined as 0 if there is no edge. The goal is to learn representations of nodes $\mathbf{U} \in \mathbb{R}^{n\times m}$, where m ($m \ll n$) is the dimension of the representation.

Matrix Factorization is a powerful method for network embedding. [Yang *et al.*, 2015; Qiu *et al.*, 2018] reveal that the

state-of-the-art DeepWalk is equivalent to matrix factorization. It seeks a basis matrix $\mathbf{V} \in \mathbb{R}^{n\times m}$ and a representation matrix $\mathbf{U} \in \mathbb{R}^{n\times m}$ to well approximate the similarity matrix $\mathbf{S}$. This gives the following objective function

$$\min_{\mathbf{U},\mathbf{V}} \|\mathbf{S} - \mathbf{V}\mathbf{U}^\top\|_F^2 \tag{1}$$

$\mathbf{U}$ is viewed as the real-valued representation of nodes, and can be used in many tasks, e.g., node clustering, classification.

In this work, we consider the problem of classifying network nodes into several categories. There are two main drawbacks of existing network embedding techniques in this setting: 1) The storage cost of the real-valued embedding matrix $\mathbf{U}$ is expensive. In addition, the computation complexity of classification is prohibitive when the network is large. 2) The learned embedding is not optimal for classification. The learning model, i.e., (1) does not optimize the classification loss, and thus the learned embedding is not discriminative. The following section presents a discrete network embedding to overcome the limitations of existing approaches.

4 Discrete Network Embedding

In this section, we present a discrete network embedding (DNE) which enforces the embedding as binary codes. The proposed model considers both the embedding learning and classification design within a unified framework.

4.1 Formulation

Discrete Matrix Factorization Existing research shows that the state-of-the-art DeepWalk is equivalent to matrix factorization on the similarity matrix $\mathbf{S}$. The (i,j)-element of $\mathbf{S}$ is defined as

$$S_{ij} = \frac{\left[\mathbf{e}_i(\mathbf{A} + \mathbf{A}^2 + \cdots + \mathbf{A}^t)\right]_j}{t} \tag{2}$$

where $\mathbf{e}_i$ is an indicator vector, where the i-th element is 1, and the others are 0, $\mathbf{A}^t$ is the multiplication of $\mathbf{A}$ by t times. Following [Tu *et al.*, 2016], we set $\mathbf{S} = \frac{\mathbf{A}+\mathbf{A}^2}{2}$.

We are interested in imposing the binary constraint on the network embedding, i.e., $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n] \in \{+1, -1\}^{m\times n}$. Specifically, the Hamming similarity between $\mathbf{b}_i$ and $\mathbf{b}_j$ is defined as

$$\begin{aligned}\mathrm{sim}(i,j) &= \frac{1}{m}\sum_{k=1}^{m} \mathbb{1}(b_{ik} = b_{jk}) \qquad (3)\\ &= \frac{1}{2m}\left(\sum_{k=1}^{m}\mathbb{1}(b_{ik} = b_{jk}) + m - \sum_{k=1}^{m}\mathbb{1}(b_{ik} \neq b_{jk})\right)\\ &= \frac{1}{2m}\left(m + \sum_{k=1}^{m} b_{ik}b_{jk}\right) = \frac{1}{2} + \frac{1}{2m}\mathbf{b}_i^\top\mathbf{b}_j\end{aligned}$$

where $\mathbb{1}(\cdot)$ denotes the indicator function that returns 1 if the element is true and 0 otherwise. Specifically, $\mathrm{sim}(i,j) = 0$ if all the bits between $\mathbf{b}_i$ and $\mathbf{b}_j$ are different and $\mathrm{sim}(i,j) = 1$ if all the bits are the same.

We use the inner product of binary network embedding to well reconstruct the similarity matrix $\mathbf{S}$. This leads to the following discrete matrix factorization objective function:

$$\begin{aligned}\min_{\mathbf{B}} \quad & \|\mathbf{S} - \mathbf{B}^\top\mathbf{B}\|_F^2 \qquad (4)\\ \text{s.t.} \quad & \mathbf{B} \in \{+1,-1\}^{m\times n}, \mathbf{B}\mathbf{B}^\top = n\mathbf{I}_m, \mathbf{B}\mathbf{1} = \mathbf{0}\end{aligned}$$

where we enforce two additional constraints on $\mathbf{B}$: the bit uncorrelated constraint remove the redundancy among the bits; the bit balanced constraint splits the dataset as balanced as possible in each bit.

Discrete Multi-class Classifier We focus the multi-class classification problem in this work, while the setting can be easily extended to multi-label classification. In addition, the labels of a subset of nodes, i.e., $\mathcal{T} = \{(\mathbf{b}_1, c_1), \cdots, (\mathbf{b}_l, c_l)\}$ are given for training the classifier. We assume that the classifier is linear and parameterized with the weight matrix $\mathbf{W} = [\mathbf{w}_1, \cdots, \mathbf{w}_C]$, where C is the number of the classes. For any embedding $\mathbf{b}$, it is classified according to the maximum of the score vector

$$\mathbf{W}^\top\mathbf{b} = \left[\mathbf{w}_1^\top\mathbf{b}, \ldots, \mathbf{w}_C^\top\mathbf{b}\right]^\top \tag{5}$$

Particularly, we further impose the binary constraint on weight matrix, i.e., $\mathbf{w}_c \in \{+1,-1\}^m$ $(c = 1, \ldots, C)$. The binary constraint reduces the classification model size [Shen *et al.*, 2017a]. In addition, due to the binary nature of $\mathbf{b}$ and $\mathbf{w}_c$, the inner product $\mathbf{w}_c^\top\mathbf{b}$ can be efficiently computed. The classification efficiency can be greatly improved. Moreover, we propose a simple yet effective loss function for multi-class classification. Let $\mathbf{b}_i$ be the embedding of the i-th node, and belong to class c_i. Ideally, we expect the large inner product on $\mathbf{w}_{c_i}$, and small inner products on $\mathbf{w}_c$, $c \neq c_i$, such that the classification error can be minimized. Formally, the multi-class classification loss function can be defined as

$$\begin{aligned}\min_{\mathbf{W}} \quad & \sum_{i=1}^{l}\sum_{c=1}^{C}\left(\mathbf{w}_c^\top\mathbf{b}_i - \mathbf{w}_{c_i}^\top\mathbf{b}_i\right) \qquad (6)\\ \text{s.t.} \quad & \mathbf{W} \in \{+1,-1\}^{m\times C}\end{aligned}$$

Note that we use the simplest linear loss in this work, and other losses, such as square loss, hingle loss, and exponential loss can be also explored in our proposed framework.

Joint Learning Framework We have introduced the discrete network embedding and classification models. Next, we propose to learn a classification-oriented network embedding, such that the learned embedding is optimal for the classification task. To this end, we propose to jointly learn the embedding and classification within a unified framework, which leads to the following objective function

$$\begin{aligned}\min_{\mathbf{W},\mathbf{B}} \quad & \frac{1}{4}\|\mathbf{S} - \mathbf{B}^\top\mathbf{B}\|_F^2 + \lambda\sum_{i=1}^{l}\sum_{c=1}^{C}\left(\mathbf{w}_c^\top\mathbf{b}_i - \mathbf{w}_{c_i}^\top\mathbf{b}_i\right) \qquad (7)\\ \text{s.t.} \quad & \mathbf{B} \in \{+1,-1\}^{m\times n}, \mathbf{B}\mathbf{B}^\top = n\mathbf{I}_m, \mathbf{B}\mathbf{1} = \mathbf{0},\\ & \text{and} \quad \mathbf{W} \in \{+1,-1\}^{m\times C}\end{aligned}$$

where λ is a nonnegative parameter to balance embedding and classifier learning. In the next section, we propose a computationally tractable optimization algorithm to solve (7).

4.2 Optimization

In essence, (7) is a nonlinear mixed-integer optimization problem involving the discrete constraint on $\mathbf{B}$, $\mathbf{W}$ and non-convex orthogonal constraint on $\mathbf{B}$, which is generally NP-hard problem. We propose a tractable alternating algorithm to iteratively optimize each variable. The flowchart of DNE is described in Algorithm 1.

Embedding Learning Given $\mathbf{W}$, we solve the sub-problem with respect to $\mathbf{B}$. We first rewritten (6) into a compact matrix form, i.e., $\text{Tr}(\mathbf{W}^{o\top}\mathbf{B})$, where the i-th column of matrix $\mathbf{W}^o$ with size $m \times n$ is defined as $\mathbf{w}_i^o = \sum_{c=1}^{C} \mathbf{w}_c - C\mathbf{w}_{c_i}$. Due to the fact that $\mathbf{B}\mathbf{B}^\top = n\mathbf{I}_m$, the sub-problem with respect to $\mathbf{B}$ can be rewritten as

$$\begin{aligned} \min_{\mathbf{B}} \quad & -\frac{1}{2}\text{Tr}\left(\mathbf{B}\mathbf{S}\mathbf{B}^\top\right) + \lambda\text{Tr}\left(\mathbf{W}^{o\top}\mathbf{B}\right) \\ \text{s.t.} \quad & \mathbf{B} \in \{+1, -1\}^{m\times n}, \mathbf{B}\mathbf{B}^\top = n\mathbf{I}_m, \mathbf{B}\mathbf{1} = \mathbf{0} \end{aligned} \tag{8}$$

We further transform the uncorrelated and balanced constraints into two regularization terms. With the fact $\|\mathbf{B}\mathbf{B}^\top - n\mathbf{I}_m\|_F^2 = \|\mathbf{B}\mathbf{B}^\top\|_F^2 + \text{const}$, we have

$$\begin{aligned} \min_{\mathbf{B}} \quad & \mathcal{L}(\mathbf{B}) = -\frac{1}{2}\text{Tr}\left(\mathbf{B}\mathbf{S}\mathbf{B}^\top\right) + \lambda\text{Tr}\left(\mathbf{W}^{o\top}\mathbf{B}\right) \\ & \qquad + \frac{\mu}{4}\|\mathbf{B}\mathbf{B}^\top\|_F^2 + \frac{\rho}{2}\|\mathbf{B}\mathbf{1}\|_F^2 \\ \text{s.t.} \quad & \mathbf{B} \in \{+1, -1\}^{m\times n} \end{aligned} \tag{9}$$

where μ and ρ are two nonnegative regularization parameters. (9) is challenging due to the discrete constraint, and has no closed-form solution. Inspired by the recent advance in non-convex optimization, we propose to optimize (9) with *proximal gradient* method. The main idea is to iteratively optimize a surrogate function. In specific, we iteratively optimize a local function $\hat{\mathcal{L}}_i(\mathbf{B})$ that linearizes $\mathcal{L}(\mathbf{B})$ at the point $\mathbf{B}^{(i)}$, and employ $\hat{\mathcal{L}}_i(\mathbf{B})$ as a surrogate of $\mathcal{L}(\mathbf{B})$. Given $\mathbf{B}^{(i)}$, the next discrete point $\mathbf{B}^{(i+1)}$ can be derived by optimizing the following objective function

$$\begin{aligned} \min_{\mathbf{B}} \quad & \hat{\mathcal{L}}_i(\mathbf{B}) = \mathcal{L}(\mathbf{B}^{(i)}) + \langle \nabla\mathcal{L}(\mathbf{B}^{(i)}), \mathbf{B} - \mathbf{B}^{(i)}\rangle \\ & \qquad + \frac{\tau}{2}\|\mathbf{B} - \mathbf{B}^{(i)}\|_F^2 \\ \text{s.t.} \quad & \mathbf{B} \in \{-1, 1\}^{m\times n} \end{aligned} \tag{10}$$

where $\tau > 0$ is a constant, $\nabla\mathcal{L}(\mathbf{B}^{(i)}) = -\mathbf{B}^{(i)}\mathbf{S} + \lambda\mathbf{W}^o + \mu\mathbf{B}^{(i)}\mathbf{B}^{(i)\top}\mathbf{B}^{(i)} + \rho\mathbf{B}^{(i)}\mathbf{1}\mathbf{1}^\top$. Then the updating rule for $\mathbf{B}^{(i+1)}$ can be defined as

$$\mathbf{B}^{(i+1)} = \text{sgn}\left(\mathcal{C}(\tau\mathbf{B}^{(i)} - \nabla\mathcal{L}(\mathbf{B}^{(i)}),\ \mathbf{B}^{(i)})\right) \tag{11}$$

Note that the function $\mathcal{C}(x, y) = \begin{cases} x, x \neq 0 \\ y, x = 0 \end{cases}$, $\mathcal{C}$ is applied in an element-wise manner in (11) to eliminate the zero entries.

Classifier Learning Given $\mathbf{B}$, we solve the sub-problem with respect to $\mathbf{W}$, i.e., (6). We iteratively optimize the projection weight of each class. The objective function with respect to $\mathbf{w}_c$ can be reduced to

$$\begin{aligned} \min_{\mathbf{w}_c} \quad & \mathbf{w}_c^\top\left(\sum_{i=1}^{l}\mathbf{b}_i - C\sum_{i=1,c_i=c}^{l}\mathbf{b}_i\right) \\ \text{s.t.} \quad & \mathbf{w}_c \in \{+1, -1\}^m \end{aligned} \tag{12}$$

Clearly, (12) has the optimal solution

$$\mathbf{w}_c = \text{sgn}\left(C\sum_{i=1,c_i=c}^{l}\mathbf{b}_i - \sum_{i=1}^{l}\mathbf{b}_i\right) \tag{13}$$

Algorithm 1 Discrete Network Embedding

Input: Network: $G(V, E)$, embedding dimensionality: m, the label of the training subset $\mathcal{T}$, parameter: λ, μ, ρ, iteration number: T_{in}, T_{out}.
Output: embedding: $\mathbf{B}$, classifier: $\mathbf{W}$.

```
Initialize W and B randomly.
repeat
    for i = 1 → T_in do                    ▷ Embedding Learning
        Update the embedding B via (11);
    end for
    for c = 1 → C do                       ▷ Classifier Learning
        Update the weight vector w_c via (13);
    end for
until converge or reach maximum iterations
```

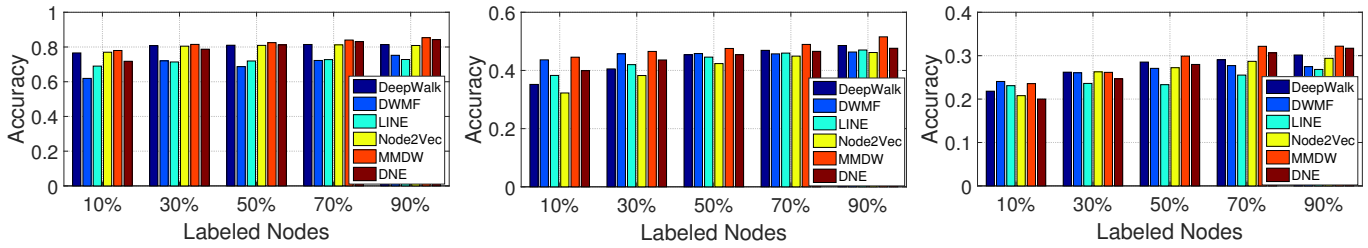
4.3 Complexity Analysis

The computational complexity of training the proposed DNE consists of the following two parts. For the embedding learning, calculating the gradient of $\mathbf{B}$ requires $\mathcal{O}(\iota m + m^2 n)$, where $\iota = \text{nnz}(\mathbf{S})$, nnz denotes the number of non-zeros. Updating $\mathbf{B}$ requires $\mathcal{O}(T_{in}(\iota m + m^2 n))$, where T_{in} is the number of iterations. For the classifier learning, updating $\mathbf{W}$ requires $\mathcal{O}(lmC)$. Suppose that the algorithm takes T_{out} iterations to convergence. The overall complexity of DNE is $\mathcal{O}\left(T_{out}(T_{in}(\iota m + m^2 n) + lmC)\right)$. In practice, T_{in}, ι, m, and T_{out} are usually small, thus training DNE is fast. Given a new binary embedding $\mathbf{b}$, the classification reduces to retrieve its nearest neighbor from $\{\mathbf{w}_c\}_{c=1}^{C}$. The search is very fast via bit operations. The computational complexity of classification in DNE is faster than existing embedding methods, which require $\mathcal{O}(mC)$.

In terms of space complexity, DNE needs to store the network embedding $\mathbf{B}$ and the weight matrix $\mathbf{W}$, which count for the storage of $\mathcal{O}(mn)$ bits and $\mathcal{O}(mC)$ bits, respectively. The existing network embedding methods, e.g., DeepWalk, need to store $\mathcal{O}(mn)$ and $\mathcal{O}(mC)$ real-valued numbers. Therefore, DNE has lower space complexity than existing network embedding methods.

5 Experiments

In this section, we evaluate the proposed discrete network embedding (DNE) on multi-class node classification tasks. All the computations reported in this study are performed on a Ubuntu 64-Bit Linux workstation with 24-core Intel Xeon CPU E5-2620 2.10 GHz and 128 GB memory.



(a) DBLP (b) YOUTUBE (c) FLICKR

Figure 2: Accuracy (%) of node classification on three datasets, (a) DBLP, (b) YOUTUBE, (c) FLICKR.

5.1 Datasets

We conduct experiments on three datasets, whose details are as follows.

- **DBLP**[1]: is a citation network in computer science. Each paper may cite or be cited by other papers, which naturally forms a citation network. In this experiment, the citation network is collected from 4 research areas, e.g., database, data mining, artificial intelligent and computer vision. It consists of 60,744 papers (nodes), 52,890 edges in total, and each node is associated with its title information.
- **YOUTUBE**[2]: is a social network of Youtube users. It contains 1,157,827 nodes and 4,945,382 edges. The labels represent groups of users who enjoy common video genres.
- **FLICKR**[2]: is a social network of Flickr users. It contains 80,513 users from a photo sharing website and 5,899,882 friendships between them. The labels represent the group membership of users. The network has 195 labels and a user may have multiple labels.

For constructing multi-class datasets, we remove the nodes that have multiple labels in YOUTUBE and FLICKR datasets. In addition, the isolated nodes are removed in the three datasets.

5.2 Comparison Methods

To demonstrate the effectiveness of the proposed DNE, we compare it with five state-of-the-art network embedding methods. The details of these methods are given below.

- **DeepWalk** [Perozzi *et al.*, 2014]: DeepWalk performs random walks over networks and employs Skip-Gram model to learn node embeddings.
- **DWMF** [Yang *et al.*, 2015]: DeepWalk has been shown to be equivalent to matrix factorization [Yang *et al.*, 2015; Qiu *et al.*, 2018]. DWMF is trained on the similarity matrix $(\mathbf{A} + \mathbf{A}^2)/2$ to obtain node representation.
- **LINE** [Tang *et al.*, 2015b]: LINE is the embedding method for large-scale networks. We employ the second-order proximity LINE (2nd-LINE) to learn representations.

[1]http://arnetminer.org/citation

[2]http://socialnetworks.mpi-sws.org/data-imc2007.html

	Method	DBLP	YOUTUBE	FLICKR
Model Size	DeepWalk	4KB	47KB	195KB
	DNE	64B	752B	3KB
Time (ms)	DeepWalk	0.98	2.48	39.46
	DNE	0.53	0.91	21.92

Table 2: Testing time (in milliseconds) and model size of classification in DeepWalk and DNE on three datasets.

- **Node2Vec** [Grover and Leskovec, 2016]: Node2Vec is a biased random walk algorithm based on DeepWalk to efficiently explore neighborhood architecture.
- **MMDW** [Tu *et al.*, 2016]: Max-Margin DeepWalk (MMDW) is a discriminative network embedding model that obtains discrimination characteristic via the max-margin classifier.

5.3 Experiment Setup

In this work, we set the dimension of network embedding as 128 for all the methods for fair comparison. For DeepWalk, window size, walk length, and walks per node are set as 10, 40, 40, respectively. For LINE, the number of negative samples is set to 5. For Node2Vec, window size, walk length and walks per node are set the same with DeepWalk, and return parameter p and in-out parameter q are set as 1 and 2, respectively. For the proposed DNE, μ and ρ are set as 0.01, 0.01, and 0.5, respectively; τ and λ are selected from the range of $[1, 10]$ and $[0, 1]$ by cross validation respectively.

For node classification, the representations for the nodes are first obtained from the network embedding methods and then used as features to train a classifier. We randomly sample a portion of the labeled nodes for training classifier and the rest nodes are used for testing. The training ratio increases from 10% to 90% for the three datasets. For DeepWalk, DWMF, LINE, and Node2Vec, the multi-class SVM by Crammer and Singer [Crammer and Singer, 2001] is employed as the classifier, which is implemented by LIBLINEAR package [Fan *et al.*, 2008]. For MMDW and DNE, the classifiers are learned on labeled nodes only and they are jointly learned with network embedding.

5.4 Results

Accuracy Figure 2 shows the accuracy results of all the methods on three datasets. We find some interesting points:

Dataset	DeepWalk		DWMF		LINE		Node2Vec		MMDW		DNE
	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup	Time(s)	Speedup	Time(s)
DBLP	646	46×	256	18×	170	12×	428	30×	1624	116×	14
YOUTUBE	416	52×	149	19×	151	19×	356	45×	693	86×	8
FLICKR	3689	24×	4473	29×	3776	24×	9027	58×	12780	83×	154

Table 1: Training time (in seconds) of network embedding methods on three datasets, including the time of embedding and classifier learning. 'Speedup' indicates the speedup (×) of DNE over baselines.

Dataset	DeepWalk		DNE
	Mem.	Red.	Mem.
DBLP	17.30MB	64×	276.81KB
YOUTUBE	11.34MB	64×	181.39KB
FLICKR	60.40MB	64×	966.42KB

Table 3: Memory usage of network embedding in DeepWalk and DNE on three datasets. 'Mem.' denotes memory usage. 'Red.' denotes memory reduction (×) of DNE over DeepWalk.

- As training ratio increases, the accuracies of all the methods generally rise.
- Among the comparisons, MMDW has the best performance due to its discrimination, but it is slower than other methods. DWMF outperforms DeepWalk on YOUTUBE, and DeepWalk outperforms DWMF on DBLP and FLICKR.
- The proposed DNE generally has comparable accuracy with the comparison methods. DNE is better than most unsupervised embedding methods in some cases of large training ratios. It reveals that the discrete embedding obtains high discrimination as DNE jointly learns the embedding and classifier.

Time Table 1 shows the training time of all the methods on three datasets. It includes the time of learning embedding and classifier. As can seen from this table, the proposed DNE is the fastest method among all the baselines. For example, DNE is 46 and 116 times faster than DeepWalk and MMDW, respectively. LINE takes the second place, which is followed by DWMF, DeepWalk, and Node2Vec. MMDW is the slowest among all the methods. The results verify our computational analysis.

In addition, we evaluate the classification efficiency. The time of classification for the comparison methods are quite close, thus we only compare DeepWalk and DNE. Table 2 reports the time of classification for DeepWalk and DNE. As can be seen, DNE is clearly faster than DeepWalk for around 2 times among all the cases, which also verifies our complexity analysis and motivation. DNE uses a discrete classifier via binary code. The classification is calculated in the Hamming metric, which is more efficient than the conventional Euclidean calculation in DeepWalk.

Memory Usage Table 3 reports the memory usage of DeepWalk and DNE for the storage of network embedding. We clearly observe that compared with DeepWalk, DNE significantly reduces the memory storage of embedding for 64 times within the same dimension. DNE only needs to store less than 1M of binary codes to represent the three datasets, even for the large-scale FLICKR dataset. In addition, the model sizes of classifiers in DeepWalk and DNE are shown in Table 2. Due to the discrete representation of classifier, we see that the model sizes of DNE are much smaller than those of DeepWalk. These results imply that DNE can work well in some extreme scenarios, e.g., resource-scarce devices.

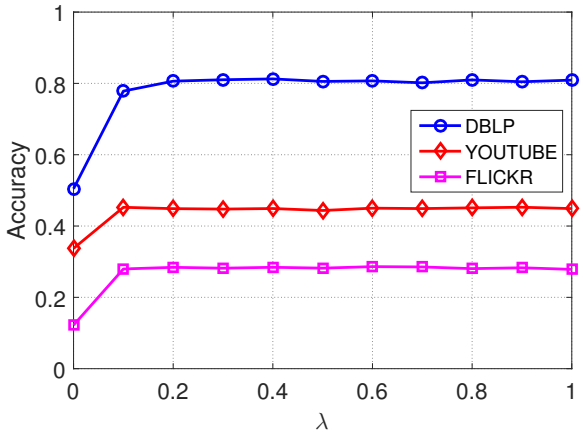


Figure 3: Accuracy of node classification with respect to different λ on three datasets.

Parameter Analysis We study the sensitivity of the key parameter, i.e., λ, in our proposed DNE. The training ratio is fixed as 0.5, and λ is ranged between [0, 1] with a step of 0.1, and classification accuracy with respect to different λ is shown in Figure 3. From Figure 3, we observe that the classification accuracy improves as λ increases from 0, and remains stable after λ reaches to around 0.2. It shows that the good performance can be achieved from the wide range between [0.2, 1]. The parameter analysis reveals that the joint learning of classifier indeed enhances the discrimination of the discrete embedding.

6 Conclusion

This work focuses on the challenging problem of seeking compact representation for network embedding. We propose a novel discrete network embedding (DNE) that leverages short binary codes to represent each node. A discrete multi-class classifier is jointly learned to improve the discrimination of embedding. Compared to existing network embedding methods, DNE enjoys both computational and memory efficiency. Extensive experiments on node classification demonstrate DNE exhibits lower storage and computational complexity than state-of-the-art network embedding methods, while achieving satisfactory accuracy.

Acknowledgments

This research is supported by the National Science Foundation of China (Grant No. 61673220), and partially supported under the Data Science and Artificial Intelligence Center (D-SAIR) at the Nanyang Technological University.

References

[Cao *et al.*, 2016] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *AAAI*, pages 1145–1152, 2016.

[Crammer and Singer, 2001] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *JMLR*, 2:265–292, 2001.

[Dai *et al.*, 2017] Quanyu Dai, Qiang Li, Jian Tang, and Dan Wang. Adversarial network embedding. *arXiv preprint arXiv:1711.07838*, 2017.

[Fan *et al.*, 2008] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *JMLR*, 9:1871–1874, 2008.

[Gong *et al.*, 2013] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *TPAMI*, 35(12):2916–2929, 2013.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864, 2016.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1025–1035, 2017.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[Li *et al.*, 2017] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed network embedding for learning in a dynamic environment. In *CIKM*, pages 387–396, 2017.

[Pan *et al.*, 2016] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. Tridnr: Tri-party deep network representation. In *IJCAI*, pages 1895–1901, 2016.

[Perozzi *et al.*, 2014] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In *KDD*, pages 701–710, 2014.

[Qiu *et al.*, 2018] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *WSDM*, pages 459–467, 2018.

[Shen *et al.*, 2017a] Fumin Shen, Yadong Mu, Yang Yang, Wei Liu, Li Liu, Jingkuan Song, and Heng Tao Shen. Classification by retrieval: Binarizing data and classifiers. In *SIGIR*, pages 595–604, 2017.

[Shen *et al.*, 2017b] Xiaobo Shen, Weiwei Liu, Ivor W. Tsang, Fumin Shen, and Quan-Sen Sun. Compressed k-means for large-scale clustering. In *AAAI*, pages 2527–2533, 2017.

[Tang and Liu, 2009] Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *KDD*, pages 817–826, 2009.

[Tang *et al.*, 2015a] Jian Tang, Meng Qu, and Qiaozhu Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *KDD*, pages 1165–1174, 2015.

[Tang *et al.*, 2015b] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: large-scale information network embedding. In *WWW*, pages 1067–1077, 2015.

[Tian *et al.*, 2014] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. Learning deep representations for graph clustering. In *AAAI*, pages 1293–1299, 2014.

[Tu *et al.*, 2016] Cunchao Tu, Weicheng Zhang, Zhiyuan Liu, and Maosong Sun. Max-margin deepwalk: Discriminative learning of network representation. In *IJCAI*, pages 3889–3895, 2016.

[Wang *et al.*, 2016] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *KDD*, pages 1225–1234, 2016.

[Wang *et al.*, 2017] Chun Wang, Shirui Pan, Guodong Long, Xingquan Zhu, and Jing Jiang. MGAE: marginalized graph autoencoder for graph clustering. In *CIKM*, pages 889–898, 2017.

[Wang *et al.*, 2018] Jingdong Wang, Ting Zhang, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. A survey on learning to hash. *TPAMI*, 40(4):769–790, 2018.

[Weiss *et al.*, 2009] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2009.

[Yang *et al.*, 2015] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Y. Chang. Network representation learning with rich text information. In *IJCAI*, pages 2111–2117, 2015.

[Zhang *et al.*, 2017] Chuxu Zhang, Lu Yu, Yan Wang, Chirag Shah, and Xiangliang Zhang. Collaborative user network embedding for social recommender systems. In *SDM*, pages 381–389, 2017.

[Zhou *et al.*, 2017] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. Scalable graph embedding for asymmetric proximity. In *AAAI*, pages 2942–2948, 2017.