Discrete Embedding for Latent Networks

Hong Yang[1] , **Ling Chen**[1] , **Minglong Lei**[2] , **Lingfeng Niu**[3] , **Chuan Zhou**[4] and **Peng Zhang**[5]
[1] Centre for Artificial Intelligence, University of Technology Sydney, Australia
[2] Faculty of Information Technology, Beijing University of Technology, China
[3] School of Economics and Management, University of Chinese Academy of Sciences, China
[4] Academy of Mathematics and Systems Science, Chinese Academy of Sciences, China
[5] Ant Financial Services Group, Hangzhou, China
hong.yang@student.uts.edu.au, Ling.Chen@uts.edu.au, leiml@bjut.edu.cn, niulf@ucas.ac.cn, zhouchuan@amss.ac.cn, zhangpeng04@gmail.com

Abstract

Discrete network embedding emerged recently as a new direction of network representation learning. Compared with traditional network embedding models, discrete network embedding aims to compress model size and accelerate model inference by learning a set of short binary codes for network vertices. However, existing discrete network embedding methods usually assume that the network structures (e.g., edge weights) are readily available. In real-world scenarios such as social networks, sometimes it is impossible to collect explicit network structure information and it usually needs to be inferred from implicit data such as information cascades in the networks. To address this issue, we present an end-to-end discrete network embedding model for latent networks (*DELN*) that can learn binary representations from underlying information cascades. The essential idea is to infer a *latent Weisfeiler-Lehman proximity matrix* that captures node dependence based on information cascades and then to factorize the latent Weisfiler-Lehman matrix under the binary node representation constraint. Since the learning problem is a mixed integer optimization problem, an efficient *maximal likelihood estimation based cyclic coordinate descent (MLE-CCD)* algorithm is used as the solution. Experiments on real-world datasets show that the proposed model outperforms the state-of-the-art network embedding methods.

1 Introduction

An increasing interest in learning discrete representations for large networks has been observed recently [Yang *et al.*, 2018]. Different from classical network embedding models that learn node representations in *continuous Euclidean spaces*, discrete network representation learning aims to learn compact representations for network nodes in *discrete Hamming spaces*, so that both the storage and the computation costs can be reduced.

The basic idea of learning discrete network embedding is to enforce binary constraints on representation vectors so that all the learnt vectors are restricted to the domain of $\{+1, -1\}$. To this end, the recent work in [Shen *et al.*, 2018] proposes a *discrete matrix factorization* model to learn short binary codes for network nodes based on binary code learning [Wang *et al.*, 2017]. According to their experiment results [Shen *et al.*, 2018], the proposed discrete matrix factorization model can reduce the storage of embeddings as well as the model size by 64 times, compared with conventional embedding models. Encouraged by the promising results, a sequence of efforts has been dedicated to learning discrete network representations, such as learning binary representations from attributed networks [Yang *et al.*, 2018] and low-bit representations from attributed networks [Yang *et al.*, 2019].

Although existing discrete network embedding models have achieved great success in compressing model size and accelerating model inference, they all assume that network structures (e.g., edge weights) are explicitly observable and are ready to be fed into the learning models. In real-world applications such as social networks, network structures are often hidden behind information cascades and need to be inferred from such information. For example, in a blogger network [Leskovec and Krevl, 2014], if a blog mentions a piece of information without linking to the information source, it is unknown where the blogger acquires the information and whether there is a (weighted) linkage between the blogger and the source node. However, a set of information cascades that record information propagation traces can be observed and collected, from which network structures can be inferred and restored. We refer to such networks as *latent networks* and focus on learning discrete embedding for latent networks.

Intuitively, discrete network embedding can be learned from latent networks via two steps. Firstly, network structures are reconstructed from information cascades, which has been widely studied in social network mining. For instance, the method in [Rodriguez *et al.*, 2011] infers network structures by formulating a generative probabilistic model of information cascades. Secondly, existing discrete network embedding models can be applied to the inferred network structures. However, such a two-step solution separates parameter tuning into two different models, which makes it hard to achieve satisfactory representations. Therefore, we aim to build an end-to-end model that learns discrete network embedding directly from latent networks hidden behind information cascades.

Compared to traditional network embedding learning for explicit network structures, learning compact representations from information cascades is confronted by the following two new challenges: *Challenge 1*, how to formulate the learning function by jointly considering the problem of inferring network structures from information propagation data and learning representations under the discrete representation constraint; *Challenge 2*, how to design an efficient algorithm to optimize the new learning function where existing binary code learning [Wang *et al.*, 2017] is inapplicable.

In light of the new challenges, we present an end-to-end learning model called discrete embedding from latent networks (*DELN*). To address *Challenge 1*, we formulate a *latent Weisfeiler-Lehman proximity matrix* that captures node dependence based on information cascades, which is then factorized under the binary node representation constraint. To tackle *Challenge 2*, we present an efficient *maximal likelihood estimation based cyclic coordinate descent (MLE-CCD)* algorithm to solve the mixed integer optimization problem.

The contributions of the paper are summarized as follows:

- This is the first effort to study the problem of learning discrete representations from latent networks where network structures are not explicitly observable.
- We present a DELN model to learn the discrete representations of latent networks, where a *latent Weisfeiler-Lehman proximity matrix* is defined to capture node dependence in latent networks, and a binary constraint is imposed on the latent Weisfeiler-Lehman matrix factorization to obtain discrete network representations.
- We present an efficient *maximal likelihood estimation based cyclic coordinate descent (MLE-CCD)* algorithm to factorize the latent Weisfeiler-Lehman matrix under the discrete representation constraint.
- We conduct experiments on real-world network data to validate the performance of the DELN model. The results demonstrate the effectiveness of our model.

2 Related work

Discrete network embedding is a powerful tool to compress network embedding models and accelerate model inference. The pioneer work in [Shen *et al.*, 2018] that uses binary code learning to obtain discrete network representations reports a 64 times reduction of model size on the public datasets of DBLP, YOUTUBE and FLICKER. Based on this work, the edge study in [Yang *et al.*, 2018] proposes a binarized embedding for attributed networks which enables joint representation learning from node links and node attributes. Moreover, motivated by the observation that although the binarized embedding can reduce the network representation size, and the strict binary constraint imposed on the learning function may incur uncontrollable accuracy loss on test sets, the recent work in [Yang *et al.*, 2019] proposes a low-bit quantization model for attributed network representation learning that can learn compact node representations with low bit-width values while preserving high representation accuracy. All these existing network embedding compression models are based on discrete matrix factorization which leads to a mixed integer programming problem. Hence, the idea of binary code learning or hashing algorithms [Wang *et al.*, 2017] can be borrowed as the solution. In spite of the success achieved by existing compressed models, they all assume that the input network structures are known a priori, which limits their applications in implicit networks where network structures cannot be explicitly observable.

Latent network inference refers to recovering network structures from information cascades. An information cascade refers to a special type of data that records an information propagation trace in a network. Several studies [Rodriguez *et al.*, 2011; Kalimeris *et al.*, 2018; Panagopoulos *et al.*, 2019] have proposed recovering latent network structures from information cascades. The work in [Rodriguez *et al.*, 2011] uses a generative probabilistic model for inferring diffusion networks by considering information propagation processes as discrete networks of continuous temporal processes. The work in [Gomez-Rodriguez *et al.*, 2012] infers network connectivity using submodular optimization and the work in [Myers and Leskovec, 2010] infers not only the connectivity but also a prior probability of infection for every edge using a convex program. In contrast, the recent study in [Kalimeris *et al.*, 2018] describes diffusion probabilities as a non-convex learning function.

Graph kernels. Graph kernels seek to learn the representation of sub structures for graphs [Yanardag and Vishwanathan, 2015; Atwood and Towsley, 2016]. Compared with the *random walk graph kernels* used in Node2Vec [Grover and Leskovec, 2016] and DeepWalk [Perozzi *et al.*, 2014], the Weisfeiler-Lehman graph kernels [Shervashidze *et al.*, 2011] are capable of capturing the joint correlation between nodes by combining information from both node features and network structures. Therefore, they are widely used in graph neural networks (GNNs) [Hamilton *et al.*, 2017]. However, the original Weisfeiler-Lehman graph kernels can be only applied to networks that are explicitly observable. In this work, we define the latent Weisfeiler-Lehman graph kernels based on network propagation data.

3 Preliminaries

Consider a latent network $\mathcal{G} = (\mathbf{V}, \mathbf{X}, \mathbf{W})$ consisting of a set of nodes $\mathbf{V} = \{v_i\}_{i=1}^{n}$, where n is the number of nodes, a set of feature vectors $\mathbf{X} = \{\mathbf{x_i}\}_{i=1}^{n}$, where $\mathbf{x_i} \in \Re^r$ is a r-dimensional feature vector for node v_i, and a set of edge weights $\mathbf{W} = \{w_{i,j}\}_{i,j=1}^{n}$, where $w_{i,j}$ is the edge weight between nodes v_i and v_j. Different from traditional networks where edge weights $\mathbf{W}$ are known a priori, edge weights of the latent network $\mathcal{G}$ are not explicitly observable. Instead, only a set of information cascades $\mathbf{C} = (\mathbf{c_1}, \cdots, \mathbf{c_k}) \in \Re^{k \times n}$ can be observed within a given time window $[0, T]$, where each cascade $\mathbf{c_t} \in \mathbf{C}$ is an n-dimensional vector $\mathbf{c_i} = (c_i^1, \cdots, c_i^n)$ that records the earliest time of the message arriving at the corresponding node, and $T \in \Re$ is the time period we are allowed to observe the network.

The aim is to learn a discrete representation matrix $\mathbf{B}$ from the propagation traces $\mathbf{C}$. An intuitive idea is to use a two-step approach that first learns the network weight matrix $\mathbf{W}$

through the propagation traces $\mathbf{C}$. Next, based on the learnt weight matrix $\mathbf{W}$, a binarized representation learning can be used to learn $\mathbf{B}$. Unfortunately, such a two-step approach does not have a unified optimization objective. It is very hard to tune parameters for two separate models.

Before designing the representation learning model, the key problem is to design a proximity matrix that can jointly capture network structure $\mathbf{W}$ behind the information cascades $\mathbf{C}$ and node features $\mathbf{X}$. Based on the recent work on GNNs such as GraphSAGE [Hamilton *et al.*, 2017] and its extensions [Xu *et al.*, 2018], the Weisfeiler-Lehman graph kernels [Shervashidze *et al.*, 2011] are capable of encoding both node features $\mathbf{X}$ and network structure $\mathbf{W}$. However, network structure $\mathbf{W}$ often cannot be observed directly and we have the information cascades $\mathbf{C}$ instead. As a result, we define a *Latent Weisfeiler-Lehman matrix* to capture both network structure $\mathbf{W}$, behind information cascades $\mathbf{C}$, and node features $\mathbf{X}$ as follows.

Definition 1. (**Latent Weisfeiler-Lehman Matrix**). Given a hidden network $\mathcal{G}$ where only information cascades $\mathbf{C}$ and node features $\mathbf{X}$ are observable, let the hidden network structure be $\mathbf{W}$ which depends on information cascades $\mathbf{C}$. Denote $\mathbf{D}(\mathbf{W}, \mathbf{C})$ as a degree matrix of the hidden structure $\mathbf{W}$ and $\mathbf{L}(\mathbf{W}, \mathbf{C}) = \mathbf{D} - \mathbf{W}$. Then, the latent Weisfeiler-Lehman matrix $\mathbf{P}$ is defined as

$$\mathbf{P}(\mathbf{W}, \mathbf{C}) = (\mathbf{I} - \gamma \mathbf{D}(\mathbf{W}, \mathbf{C})^{-1} \mathbf{L}(\mathbf{W}, \mathbf{C}))^k \mathbf{X}, \qquad (1)$$

where $\mathbf{I}$ is an identity matrix, $\gamma \in [0, 1]$ is a tradeoff parameter, and k is the number of network layers.

According to the above definition, obtaining the latent Weisfeiler-Lehman matrix $\mathbf{P}$ needs to infer the latent adjacent matrix $\mathbf{W}$ based on cascades $\mathbf{C}$, i.e., maximizing the likelihood function $f(\mathbf{C}|\mathbf{W})$. Based on the previous work [Gomez-Rodriguez *et al.*, 2012], when using the exponential distribution function to simulate the propagation distribution $\phi(t_i|t_j, \mathbf{W})$ which denotes the probability of node v_j infecting v_i at time t_i, i.e.,

$$\phi(t_i|t_j, w_{j,i}) = \begin{cases} w_{j,i} \cdot e^{-w_{j,i}(t_i - t_j)}, & if\ t_j < t_i, \\ 0, & otherwise, \end{cases} \qquad (2)$$

the probability of observing a cascade $\mathbf{c_t}$ under the exponential propagation distribution can be represented as follows,

$$f(\mathbf{c_t}|\mathbf{W}) = \prod_{t_i \le T} \left(\sum_{t_j < t_i} \frac{\phi(t_i|t_j, \mathbf{W})}{1 - \phi(t_i|t_j, \mathbf{W})} \prod_{t_k < t_i} \Big(1 - \phi(t_i|t_k, \mathbf{W})\Big) \right) \times \left(\prod_{t_m > T} \prod_{t_i \le T} \Big(1 - \phi(t_m|t_i, \mathbf{W})\Big) \right), \qquad (3)$$

where the first part in $(\cdots)$ denotes the probability of observing all the nodes that are activated by the already activated nodes t_i, and the second part denotes the probability of not observing the nodes v_m that are not activated by the already activated nodes t_i in cascade $\mathbf{c_t}$.

4 The proposed model

In this part, we learn the discrete network representation B by factorizing the latent Weisfeiler-Lehman matrix $\mathbf{P}$ given in **Definition 1**. Concretely, given a matrix of propagation cascades $\mathbf{C}$ and node feature $\mathbf{X}$, we infer the latent Weisfeiler-Lehman matrix $\mathbf{P}$ based on a latent network structure $\mathbf{W}$ and the given $\mathbf{X}$, and then embed each node $v_i \in \mathbf{V}$ into a d-dimensional vector $\mathbf{b_i} \in \{-1, +1\}^d$ in a discrete Hamming space, where $\mathbf{b_i}$ is the i^{th} row of matrix $\mathbf{B}$. The learning function can be formulated by factorizing the latent Weisfeiler-Lehman matrix $\mathbf{P}$ under the constraints that the representations $\mathbf{B}$ are discrete and the best latent structure $\mathbf{W}$ can be inferred from the cascades $\mathbf{C}$ as in Eq.(4),

$$\min_{\mathbf{W}, \mathbf{B}, \mathbf{Z}} \|\mathbf{P} - \mathbf{B}\mathbf{Z}\|_F^2 + \alpha \|\mathbf{Z}\|_F^2 - \beta \sum_{\mathbf{c_k} \in \mathbf{C}} \log f(\mathbf{c_k}|\mathbf{W}),$$
$$s.t.:\ \mathbf{W} \ge 0,\ \mathbf{B} \in \{-1, +1\}^{n \times d},\ \mathbf{Z} \in \Re^{d \times f}, \qquad (4)$$

where α and β are regularization parameters.

Due to the discrete constraint with respect to matrix $\mathbf{B}$, Eq.(4) is NP-hard. We introduce an efficient algorithm to solve the problem. In particular, we present an efficient algorithm to iteratively optimize each variable to solve the optimization problem in Eq.(4). The algorithm updates one parameter at a time and converges rapidly.

4.1 W-step:

Given cascades $\mathbf{C}$, estimate the network structure $\mathbf{W}$ and accordingly the latent Weisfeiler-Lehman matrix $\mathbf{P}$. The problem reduces to solve the last item of the log likelihood function given in Eq.(4), i.e.,

$$\mathcal{L}(\mathbf{W}) = \max_{\mathbf{W} \ge 0} \sum_{\mathbf{c_t} \in \mathbf{C}} \Big(\sum_{t_i \le T} \log \sum_{t_j < t_i} \frac{\Phi(\mathbf{W}; j, i)}{1 - \Phi(\mathbf{W}; j, i)} + \sum_{t_i \le T} \sum_{t_k < t_i} \log[1 - \Phi(\mathbf{W}; j, i)] + \sum_{t_m > T} \sum_{t_i \le T} \log[1 - \Phi(\mathbf{W}; i, m)] \Big). \qquad (5)$$

where $\Phi(\mathbf{W}; j, i) := \phi(t_i|t_j, \mathbf{W})$. Eq.(5) is concave. We let $\frac{\partial \mathcal{L}}{\partial w_{ji}} = 0$ and obtain a closed-form solution,

$$w_{ji} = \begin{cases} \frac{1}{z} \sum_{\mathbf{c_t} \in \mathbf{C}} I_{\mathbf{c_t}}(v_i, v_j) \frac{1}{t_i - t_j}, & t_i \le T, t_j < t_i, \\ \frac{1}{z} \sum_{\mathbf{c_t} \in \mathbf{C}} I_{\mathbf{c_t}}(v_i, v_j) \frac{1}{T - t_i}, & t_i > T, t_j \le T, \end{cases} \qquad (6)$$

where $I_{\mathbf{c_t}}(v_i, v_j)$ is an indicator which equals to 1 if cascade $\mathbf{c_t}$ satisfies the time constraint. z denotes a normalization of the total number of node pairs that meet the constraint. If the given time constraints are not met, the weight w_{ji} is set to $1/T$. Based on Eq.(6), it is easy to infer the latent Weisfeiler-Lehman matrix $\mathbf{P}$ according to Eq.(1) in **Defintion 1**.

4.2 Z-step:

Given $\mathbf{W}$ and $\mathbf{B}$ are fixed, we solve the sub-problem with respect to $\mathbf{Z}$ in Eq.(4). For simplicity, we use $\mathbf{P}$ to replace $\mathbf{P}(\mathbf{W}, \mathbf{C})$. Then, the loss function can be written as follows,

$$\min_{\mathbf{Z}} \|\mathbf{P} - \mathbf{B}\mathbf{Z}\|_F^2 + \alpha \|\mathbf{Z}\|_F^2, \qquad (7)$$
$$= -tr(\mathbf{P}^T \mathbf{B}\mathbf{Z}) + tr(\mathbf{Z}^T \mathbf{B}^T \mathbf{B}\mathbf{Z}) + \alpha tr(\mathbf{Z}^T \mathbf{Z}).$$

By calculating the derivative of Eq.(7), a closed form solution can be derived as follows,

$$\mathbf{Z} = (\mathbf{B}^T \mathbf{B} + \alpha \mathbf{I})^{-1} \mathbf{B}^T \mathbf{P}. \qquad (8)$$

4.3 B-step:

Given $\mathbf{Z}$ and $\mathbf{W}$ fixed, rewrite the objective function in Eq.(4) with respect to $\mathbf{B}$ as follows,

$$\begin{aligned}\min_{\mathbf{B}} &\|\mathbf{P} - \mathbf{B}\mathbf{Z}\|_F^2 \quad (9)\\ &= -tr(\mathbf{B}^T\mathbf{P}\mathbf{Z}^T) + tr(\mathbf{Z}^T\mathbf{B}^T\mathbf{B}\mathbf{Z}),\\ s.t. : &\mathbf{B} \in \{-1, +1\}^{n\times d}.\end{aligned}$$

According to the observation that a closed-form solution for one column of $\mathbf{B}$ can be achieved by fixing all of the other columns, the algorithm iteratively learns one bit of $\mathbf{B}$ at a time. Let $\mathbf{b}^l$ be the l^{th} column of $\mathbf{B}$, and $\mathbf{B}'$ the matrix of $\mathbf{B}$ excluding $\mathbf{b}^l$. Then, $\mathbf{b}^l$ is the one bit for all the n samples. Similarly, let $\mathbf{q}^l$ be the l^{th} column of $\mathbf{Q} = \mathbf{P}\mathbf{Z}^T$, $\mathbf{Q}'$ the matrix of $\mathbf{Q}$ excluding $\mathbf{q}^l$, $\mathbf{z}^l$ the l^{th} row of $\mathbf{Z}$ and $\mathbf{Z}'$ the matrix of $\mathbf{Z}$ excluding $\mathbf{z}^l$. Then, we obtain

$$tr(\mathbf{Z}^T\mathbf{B}^T\mathbf{B}\mathbf{Z}) = \mathbf{z}^l\mathbf{Z}^{'T}\mathbf{B}^{'T}\mathbf{b}^l + const. \quad (10)$$

Following the same logic, we obtain

$$tr(\mathbf{B}^T\mathbf{Q}) = (\mathbf{q}^l)^T\mathbf{b}^l + const. \quad (11)$$

Substituting Eqs.(10) and (11) into Eq.(9), we obtain the optimization problem with respect to $\mathbf{b}^l$ as follows,

$$\begin{aligned}\min_{\mathbf{b}^l} &\mathbf{z}^l\mathbf{Z}^{'T}\mathbf{B}^{'T}\mathbf{b}^l - (\mathbf{q}^l)^T\mathbf{b}^l \quad (12)\\ &= (\mathbf{z}^l\mathbf{Z}^{'T}\mathbf{B}^{'T} - (\mathbf{q}^l)^T)\mathbf{b}^l,\\ s.t. : &\mathbf{b}^l \in \{-1, +1\}^{n\times 1}.\end{aligned}$$

Eq.(12) has a closed form solution as follows,

$$\mathbf{b}^l = sign(\mathbf{q}^l - \mathbf{B}'\mathbf{Z}'(\mathbf{z}^l)^T). \quad (13)$$

Using this method, each bit $\mathbf{b}$ can be computed based on the pre-learned $d-1$ bits of $\mathbf{B}'$. 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.

5 Experiments

The purpose of the experiments is to answer two questions: first, whether DELN can capture node dependence in latent networks by using the latent Weisfeiler-Lehman matrix; and second, whether DELN performs better than the state-of-the-art embedding models in terms of accuracy and model size?

Algorithm 1 Discrete Embedding Latent Networks (DELN)

```
Require: Cascades $\mathbf{C}$, feature $\mathbf{X}$, dimension $d$, # of iterations $\tau_1$ and $\tau_2$, parameters $T$, $\alpha$, $\beta$,
Ensure: Discrete representation matrix $\mathbf{B}$
 1: Initialize $\mathbf{W}$, $\mathbf{Z}$, $\mathbf{B}$ randomly
 2: W-Step: Calculate $\mathbf{W}$ using Eq.(6)
 3: Calculate $\mathbf{P}$ using Eq.(1)
 4: Repeat until converge or reach $\tau_1$
 5: Z-Step: Calculate $\mathbf{Z}$ using Eq.(8)
 6: B-Step: Repeat until converge or reach $\tau_2$
 7: for $l = 1, \cdots, d$ do
 8:     update $b^l$ using Eq.(13)
 9: end for
10: return matrix $\mathbf{B}$
```

Datasets	Nodes	Edges	Attributes	Labels
Cora	2,708	5,429	1,433	7
Citeseer	3,327	4,732	3,703	6
Wiki	2,405	17,981	4,973	19
BlogCatalog	5,196	171,743	8,189	6

Table 1: Dataset Description

Networks	Cora	Citeseer	Wiki	Blogcatalog
Visible links	5,429	4,732	17,981	171,743
Latent links	10,303	8,314	24,349	232,566

Table 2: Two Types of Network Structures

5.1 Experimental Setup

Datasets

Table 1 summarizes the datasets, where Wiki [Yang *et al.*, 2015] is a network of webpages, Citeseer [Lu and Getoor, 2003; Sen *et al.*, 2008] is a scientific network where the nodes represent papers and the edges are paper citations, Cora [Lu and Getoor, 2003; Sen *et al.*, 2008] is another citation network which focuses on publications in the machine learning area, and BlogCatalog [Huang *et al.*, 2017] is a social network which concerns blog users. All the networks have attributes which describe node features.

Table 2 shows the latent structures of the four datasets inferred from the information cascades by the diffusion sampling algorithm similar to the recent work in [Shi *et al.*, 2019]. The information cascades are generated by simulating message propagation sequences sampled by multi-trace random walks, where time-interval sampling is used to generate the transmission time between nodes [Rodriguez *et al.*, 2011].

Baseline Methods

DeepWalk [Perozzi *et al.*, 2014] maximizes node co-occurrences in random walks. *Node2vec* [Grover and Leskovec, 2016] extends DeepWalk to a biased setting where both DFS and BFS neighbors are explored. *GraRep* [Cao *et al.*, 2015] is based on matrix factorization which exploits high order proximity in networks. *Spectral Clustering (Spectral)* [Ng *et al.*, 2002] explores the spectral features of adjacent matrices. *TADW* [Yang *et al.*, 2015] combines attributes and structures for matrix tri-factorization. *LANE* [Huang *et al.*, 2017] is also an attribute network embedding model. The structural embedding is based on spectral techniques. *BANE* [Yang *et al.*, 2018] learns embedding with attribute information. The embeddings are binary vectors.

Settings and Metrics

For all of the models, we set the embedding dimension as $d = 128$. The parameters of all baselines are set as the default values. To evaluate the generated embeddings through a node classification task, the embedding vectors are fed into a logistic regression classifier [Fan *et al.*, 2008]. Following the setups in Deepwalk, we randomly sample a portion of nodes for training and the rest for testing. The ratio of training samples ranges from 10% to 90%. All of the compared models

		Micro-F1 (%)					Macro-F1(%)				
Datasets	Models	10%	30%	50%	70%	90%	10%	30%	50%	70%	90%
Cora	DeepWalk	0.6651	0.7443	0.7836	0.8036	0.8258	0.6489	0.7328	0.7734	0.7928	0.8114
	Node2vec	0.7776	0.8131	0.8299	0.8392	0.8394	0.7644	0.8055	0.8236	0.8309	0.8327
	GraRep	0.7605	0.7963	0.8168	0.8242	0.8330	0.7445	0.7831	0.8034	0.8102	0.8184
	Spectral	0.7661	0.8250	0.8411	0.8533	0.8608	0.7349	0.8107	0.8260	0.8397	0.8488
	TADW	0.7872	0.8126	0.8366	0.8516	0.8532	0.7648	0.7928	0.8083	0.8150	0.8175
	LANE	0.6822	0.7264	0.7478	0.7621	0.8035	0.6744	0.6918	0.7312	0.7590	0.7944
	BANE	0.8015	0.8298	0.8519	0.8569	0.8763	0.7892	0.8187	0.8404	0.8462	0.8657
	DELN	**0.8076**	**0.8375**	**0.8619**	**0.8708**	**0.8892**	**0.7928**	**0.8262**	**0.8489**	**0.8532**	**0.8829**
Citeseer	DeepWalk	0.4695	0.5928	0.6561	0.6890	0.7051	0.4358	0.5542	0.6146	0.6485	0.6705
	Node2vec	0.5682	0.6596	0.6909	0.7100	0.7141	0.5172	0.6042	0.6355	0.6589	0.6630
	GraRep	0.5993	0.6843	0.7029	0.7119	0.7194	0.5326	0.6245	0.6456	0.6570	0.6653
	Spectral	0.5911	0.6852	0.7018	0.7088	0.7168	0.5167	0.6223	0.6442	0.6574	0.6646
	TADW	0.6279	0.6598	0.6714	0.6783	0.6826	0.5673	0.6020	0.6204	0.6287	0.6351
	LANE	0.5266	0.5809	0.5993	0.6272	0.6552	0.4961	0.5580	0.5830	0.6124	0.6307
	BANE	0.6451	0.6926	0.7288	0.7361	0.7503	0.6022	0.6521	0.6894	0.6946	0.7084
	DELN	**0.6554**	**0.7020**	**0.7349**	**0.7565**	**0.7650**	**0.6095**	**0.6634**	**0.6976**	**0.7236**	**0.7312**
Wiki	DeepWalk	0.6187	0.7198	0.7577	0.7761	0.7911	0.4650	0.5804	0.6299	0.6392	0.6806
	Node2vec	0.6548	0.7373	0.7688	0.7882	0.8026	0.4799	0.5864	0.6370	0.6753	0.7202
	GraRep	0.6778	0.7634	0.7824	0.8105	0.8202	0.4984	0.6063	0.6700	0.6858	0.7461
	Spectral	0.7062	0.7724	0.7923	0.8013	0.8070	0.5364	0.6325	0.6660	0.6727	0.7163
	TADW	0.6714	0.7081	0.7246	0.7353	0.7462	0.5702	0.6033	0.6257	0.6536	0.6589
	LANE	0.6328	0.6845	0.7013	0.7334	0.7385	0.5571	0.5858	0.6140	0.6471	0.6631
	BANE	0.7126	0.7670	0.7819	0.7929	0.8088	0.5619	0.6376	0.6550	0.6973	0.7537
	DELN	**0.7245**	**0.7747**	**0.7964**	**0.8114**	**0.8326**	**0.5874**	**0.6747**	**0.6819**	**0.7147**	**0.7752**

Table 3: Node Classification Results (d=128)

run 10 times and the averaged results are reported. The performance is evaluated in terms of Micro-F1 and Macro-F1. The original attributes of all networks are reduced to vectors of 128 dimensions with SVD.

5.2 Experiment Results

A Case Study on BlogCatalog

We conduct a case study on BlogCatalog to answer the first question as to whether the proposed DELN model can capture node dependence in the latent networks recovered from information cascades. Because Deepwalk can only work on plain networks without attributes, we concatenate attributes with the embedding vectors generated by Deepwalk for a fair comparison. Different from Deepwalk and BANE that can only make use of the explicitly observable links of size 171,743, DELN, built on 5,196 information cascades, obtains the best Micro-F1 scores. This is because DELN can capture a latent network structure of size 232,566 inferred from the 5,196 information cascades. As shown in Table 2, there are only 171,743 links explicitly observable for building Deepwalk and BANE. In contrast, DELN can infer a larger latent structure of 232,566 links. More results are shown in Figure 1. From these results, we can conclude that, by recovering the latent network structure, DELN achieves higher Micro-F1 scores than both Deepwalk and BANE, and the performance gain of DELN is significant. *The results answer the first question that DELN can capture node dependence in latent networks using the latent Weisfeiler-Lehman matrix.*

Node Classification Task

In this part, we test the models on Cora, Wiki and Citeseer. The aim is to answer the second question as to whether the proposed model can learn embeddings that are both accurate and efficient in terms of model size. Table 3 lists all the comparison results under different ratios of training data with respect to Micro-F1 and Macro-F1 scores.

When compared with the local network embedding methods such as DeepWalk and node2vec, DELN obtains better results under different ratios of training data. Moreover, the improvement is significant when the training ratio is relatively low. Even though we add attribute information on the embedding vectors learned from DeepWalk and node2vec for node classification, these *local network embedding* methods are not as stable as DELN.

When compared with the *global embedding methods* such as GraRep and Spectral Clustering, DELN also obtains better results. By introducing the global information or high-order network proximity information, GraRep and Spectral Clustering are also very competitive. For example, GraRep achieves similar results to DELN on Wiki, especially when the training ratio is high. Spectral clustering is as stable as DELN on Wiki. Compared with the BANE model, DELN obtains better results, especially when the training ratio is relatively low.

We also compare the size of the models to evaluate the ef-

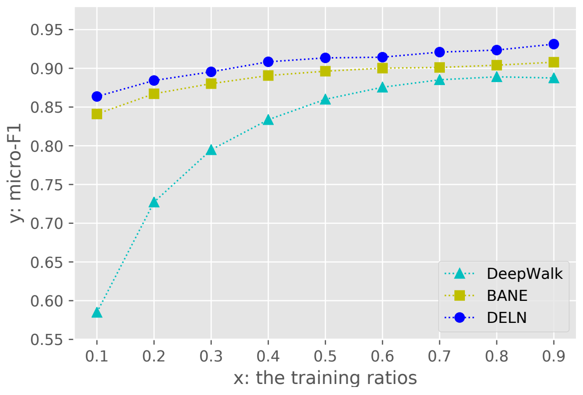

Figure 1: Comparisons *w.r.t.* the ratio of training data on BlogCatalog. DELN outperforms Deepwalk and BANE because DELN, built on 232,566 latent links inferred from 5,196 cascades, captures more structure information than Deepwalk and BANE which are built on the explicitly observable links of sizes 171 and 743.

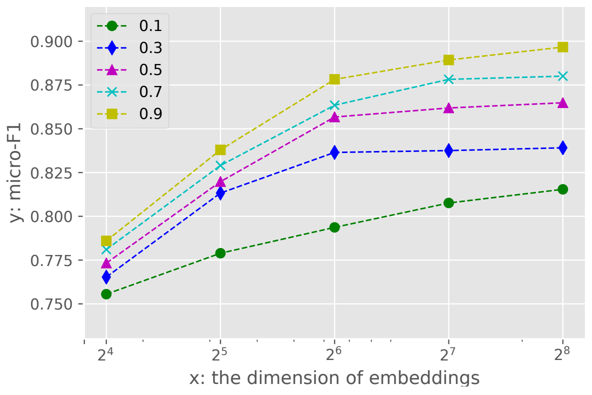

Figure 2: Comparisons *w.r.t.* embedding dimension on Cora. The embedding dimension changes from 16 to 256. The Micro-F1 scores under different training ratios (0.1, 0.3, 0.5, 0.7, 0.9) become stable when the dimension reaches 64.

ficiency of memory consumption. Table 4 shows the sizes of embedding vectors on Cora, Citeseer and Wiki. We select a global embedding model GraRep and a local embedding model DeepWalk for comparisons. From the results in Table 4, we can observe that the embedding sizes of DELN are only $1/60$ of DeepWalk and $1/30$ of GraRep. *The results show that DELN, compared with the state-of-the-art methods, is capable of learning more compact representations without accuracy loss, which answers the second question.*

5.3 Parameter Analysis

We test DELN with respect to different parameters to validate its robustness. We test the Micro-F1 and Macro-F1 scores of DELN with d varying from 16 to 256. The results of different training ratios are plotted in Figure 2. From the results we can observe that the performance under different training ratios becomes stable when d reaches 64. The changing dimension

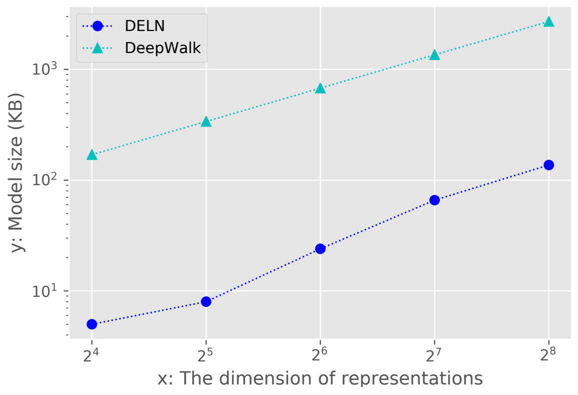

Figure 3: Embedding size *w.r.t.* embedding dimension on Cora. DELN takes smaller memory space than Deepwalk.

Datasets	DeepWalk	GraRep	DELN
Cora	3,815KB(57.8×)	2,366KB(35.8×)	66KB
Citeseer	4,673KB(59.9×)	2,083KB(26.7×)	78KB
Wiki	3,181KB(60.0×)	2,122KB(40.0×)	53KB

Table 4: Comparisons of Model Sizes

only has an impact when the training ratio is low, such as 0.1. The model size with respect to the dimensions is also compared and the results are listed in Figure 3. We observe that the model size of DELN is consistently smaller than that of DeepWalk, which demonstrates that DELN performs more stably and better than Deepwalk in terms of model size.

6 Conclusions

In this paper we study the challenging problem of learning discrete representations from latent networks where network structures (e.g., edge weights) are not explicitly observable. We present a new DELN model to learn discrete representations from latent networks, where a *latent Weisfeiler-Lehman matrix* is defined to capture node dependence in the latent networks, and a binary constraint is imposed on the latent Weisfeiler-Lehman matrix factorization to obtain discrete representations. An efficient maximal likelihood estimation based cyclic coordinate descent (MLE-CCD) algorithm is used as the solution. The experiment results validate that DELN not only captures node dependence in latent networks by using the latent Weisfeiler-Lehman matrix, it also performs better than the state-of-the-art network embedding models in terms of accuracy and model size.

Acknowledgements

This research was partially supported by the National Key Research and Development Program of China (No. 2016YFB0801003), the NSFC (No. 61872360), the Youth Innovation Promotion Association CAS (No. 2017210), an Australian Government Research Training Program Scholarship , and Australia ARC Discovery Project DP180100966.

References

[Atwood and Towsley, 2016] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1993–2001, 2016.

[Cao *et al.*, 2015] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *ACM International on Conference on Information and Knowledge Management*, pages 891–900. ACM, 2015.

[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. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[Gomez-Rodriguez *et al.*, 2012] Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. *ACM Transactions on Knowledge Discovery from Data*, 5(4):21, 2012.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864. ACM, 2016.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[Huang *et al.*, 2017] Xiao Huang, Jundong Li, and Xia Hu. Label informed attributed network embedding. In *ACM International Conference on Web Search and Data Mining*, pages 731–739. ACM, 2017.

[Kalimeris *et al.*, 2018] Dimitris Kalimeris, Yaron Singer, Karthik Subbian, and Udi Weinsberg. Learning diffusion using hyperparameters. In *International Conference on Machine Learning*, pages 2425–2433, 2018.

[Leskovec and Krevl, 2014] Jure Leskovec and Andrej Krevl. Snap datasets: Stanford large network dataset collection, 2014.

[Lu and Getoor, 2003] Qing Lu and Lise Getoor. Link-based classification. In *International Conference on Machine Learning*, pages 496–503, 2003.

[Myers and Leskovec, 2010] Seth Myers and Jure Leskovec. On the convexity of latent social network inference. In *Advances in Neural Information Processing Systems*, pages 1741–1749, 2010.

[Ng *et al.*, 2002] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856, 2002.

[Panagopoulos *et al.*, 2019] George Panagopoulos, Michalis Vazirgiannis, and Fragkiskos D Malliaros. Influence maximization via representation learning. *arXiv preprint arXiv:1904.08804*, 2019.

[Perozzi *et al.*, 2014] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014.

[Rodriguez *et al.*, 2011] M. Rodriguez, D. Balduzzi, and B. Scholkopf. Uncovering the temporal dynamics of diffusion networks. In *International Conference on Machine Learning*, pages 561–568, 2011.

[Sen *et al.*, 2008] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

[Shen *et al.*, 2018] Xiaobo Shen, Shirui Pan, Weiwei Liu, Yew-Soon Ong, and Quan-Sen Sun. Discrete network embedding. In *International Joint Conference on Artificial Intelligence*, pages 3549–3555, 2018.

[Shervashidze *et al.*, 2011] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[Shi *et al.*, 2019] Yong Shi, Minglong Lei, Hong Yang, and Lingfeng Niu. Diffusion network embedding. *Pattern Recognition*, 88:518–531, 2019.

[Wang *et al.*, 2017] Jingdong Wang, Ting Zhang, Nicu Sebe, Heng Tao Shen, et al. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):769–790, 2017.

[Xu *et al.*, 2018] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018.

[Yanardag and Vishwanathan, 2015] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374, 2015.

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

[Yang *et al.*, 2018] Hong Yang, Shirui Pan, Peng Zhang, Ling Chen, Defu Lian, and Chengqi Zhang. Binarized attributed network embedding. In *IEEE International Conference on Data Mining*, pages 1476–1481. IEEE, 2018.

[Yang *et al.*, 2019] Hong Yang, Shirui Pan, Ling Chen, and Peng Zhang. Low-bit quantization for attributed network representation learning. In *International Joint Conference on Artificial Intelligence*, 2019.