Network Embedding under Partial Monitoring for Evolving Networks

Yu Han[1] , **Jie Tang**[1] and **Qian Chen**[2]
[1]Tsinghua University
[2]Tencent Technology (SZ) Co., Ltd.
yuhanthu@126.com, jietang@tsinghua.edu.cn, qianchen@tencent.com

Abstract

Network embedding has been extensively studied in recent years. In addition to the works on static networks, some researchers try to propose new models for evolving networks. However, sometimes most of these dynamic network embedding models are still not in line with the actual situation, since these models have a strong assumption that we can achieve all the changes in the whole network, while in fact we cannot do this in some real world networks, such as the web networks and some large social networks. So in this paper, we study a novel and challenging problem, i.e., network embedding under partial monitoring for evolving networks. We propose a model on dynamic networks in which we cannot perceive all the changes of the structure. We analyze our model theoretically, and give a bound to the error between the results of our model and the potential optimal cases. We evaluate the performance of our model from two aspects. The experimental results on real world datasets show that our model outperforms the baseline models by a large margin.

1 Introduction

Many things are organized in the form of networks, and we can get a wealth of information from the structure of networks. Due to this fact, *network embedding*, which can be regarded as a kind of representation learning specific to networks, is attracting more and more attention from researchers in many fields. The target of network embedding is to embed the nodes of the networks into a continuous low-dimensional latent space, i.e., to learn the real-valued low-dimensional vector representations for the nodes based on the structure of the networks, and the learned vector representations of the nodes can be regarded as their features used for many downstream tasks, such as link prediction.

Various network embedding models have been developed, and the key idea of most of them is extracting one or more kinds of information, or more specifically the proximities among the nodes, from the network structure to learn the vector representations of the nodes so that the learned representations can preserve the extracted information. For example, DeepWalk [Perozzi *et al.*, 2014] and Node2vec [Grover and Leskovec, 2016] employ random walk to transform the network structure into sequences of nodes, and borrows the idea of Word2vec [Mikolov *et al.*, 2013] to learn the node representations. So the proximities among the nodes in the same sliding window can be preserved by the same way as Word2vec. LINE [Tang *et al.*, 2015] defines two kinds of proximities specifically (1st and 2nd order proximities) and adopts asynchronous stochastic gradient to learn node representations for these two kinds of proximities separately. Different network embedding models focus on different kinds of proximities and use different methods to learn distributed representations for the nodes. To some extent, one proximity results in one network embedding algorithm. Almost all these classical network embedding algorithms can only be employed on static networks. However, the real world networks are usually not static. They usually have two characteristics. First, they are usually dynamic and evolve all the time. In addition to this obvious characteristic, we find in practice that some networks change without notifying us, so we need to query the nodes to learn about the latest changes of the network structure. For example, in a web network, we have to crawl the web pages to discover their outgoing links and their changes [Bahmani *et al.*, 2012]. In addition, many large network platforms adopt log systems to record the behavior of the nodes. If we want to know whether two nodes established a relationship, we have to check the behavior logs of these two nodes. Unfortunately, as pointed by [Anagnostopoulos *et al.*, 2012], due to the great cost of the query, it is often impractical to query all the nodes at every moment. In other words, we can only probe part of the nodes to update the image of the network structure.

To perform network embedding on evolving networks, [Zhu *et al.*, 2016] proposes BCGD, which adopts *Adjacency Proximity*, a kind of proximity we will introduce in Section 3.3, to learn the vector representations of the nodes. However, there are two shortages for BCGD. First, it can only employ Adjacency Proximity, but not other kinds of proximities. More importantly, it is built on a graph sequence $< G_{t_0}, G_{t_1}, \cdots >$, in which each G represents a structure of the network at a time stamp, and it assumes that we can get the true network structure all the time. However, this assumption cannot hold for all the dynamic networks because of the second characteristic of real world networks we introduced

above. So in this paper, we try to solve this problem, which is ignored by most network embedding algorithms but very important in practice. We propose *CPNE* (Credit Probing Network Embedding) to learn the vector representations of the nodes on evolving networks in a partial monitoring mode, that means we can only probe part of the nodes in a budget to get the information we need to perform network embedding, which is more in line with the real situation.

First, we employ a framework based on matrix factorization to incorporate network embedding models. On this basis, we propose our algorithm to perform network embedding under partial monitoring on evolving networks. We give this algorithm an error bound by careful theoretical analysis. At last, we evaluate our model on some real world datasets from two aspects. First we test the ability of our model to approach the potential optimal embedding values. Then we test the performance of our model on a practical problem, namely link prediction. All the experiments show great advantages of our algorithm over the baseline models.

2 Related Works

There are two lines of research mostly related to our work, i.e., network embedding and dynamic network computation.

Network embedding. Different network embedding models focus on different proximities. [Cao *et al.*, 2015] learns node representations by employing the k-step transition probabilities among the nodes. [Qiu *et al.*, 2017] tries to find the closed forms for the proximities used in some network embedding models. [Tang and Liu, 2009; Zheng *et al.*, 2016; Tu *et al.*, 2016; Wang *et al.*, 2017] utilize the proximity among the nodes belonging to the same community to learn node embedding. [Song *et al.*, 2009] employs three kinds of proximities to learn the nodes' representations separately, i.e., Katz measure, Rooted Pagerank and Escape probability. However, all these models ignore the two characteristics of real world networks, i.e., dynamics and partial observerbility.

Dynamic network computation. Most networks are dynamic and we often have to develop new algorithms to deal with the problems on the evolving networks. [Tong *et al.*, 2008] studies the author-conference bipartite graphs and tracks the properties of the nodes as the graphs evolve. [Han and Tang, 2017] analyzes the users' probability of joining the groups in a dynamic network. [Zhou *et al.*, 2018] proposes a dynamic network embedding algorithm by modeling triadic closure process. However, these models assume we can obtain all the changes of the network at every moment, which is not realistic for many real world network. [Anagnostopoulos *et al.*, 2012] introduces and formalize the problem of dynamic network computation, and summarizes the essential properties of many large networks in which the links are evolving all the time and we can only track these changes by explicitly probing the networks. In response to this situation, [Bahmani *et al.*, 2012] proposes a convenient algorithm to compute pagerank on evolving networks. However, it chooses only one node to probe at each moment, while our algorithm supports choosing a batch of nodes to probe at each moment.

3 Network Embedding Framework

To study network embedding on evolving networks, we need to first use a unified framework to cover most existing network embedding algorithms.

3.1 Formulation

We use $G = (V, A)$ to denote the structure of a network. $V = (v_1, v_2, \cdots, v_N)$ is the set of the nodes in the network, and $|V| = N$. A is the adjacency matrix, which is an $N \times N$ matrix. A can be real-valued if G is a weighted network.

Definition 1. ***Proximity Matrix.*** *For each kind of node proximity in a network, there is a corresponding proximity matrix. We use M to denote the proximity matrix, which is an $N \times N$ matrix and $M_{i,j}$ represents the value of the corresponding proximity from node v_i to v_j.*

Please note that M may not be Hermitian matrix, because the values of the proximities are often asymmetrical in directed networks.

Definition 2. ***Node Vector.*** *Node vectors are the nodes' distributed representations. The dimension of the vectors is denoted as p. So all the node vectors form an $N \times p$ matrix, denoted as X, whose i-th row is the node vector of v_i.*

Given graph G, the static network embedding can be formulated as learning a mapping functions , i.e., $f : V \Rightarrow \mathbb{R}^p$. Thus, the i-th row of X is actually $f(v_i)$.

3.2 Matrix Factorization

First, we incorporate network embedding models with a unified framework based on matrix factorization. Given one or more kinds of proximities, the learned distributed representations of the nodes should reflect the proximities adopted. We can use dot product to measure the result of network embedding. Thus we can achieve this goal by minimizing the objective function $\mathcal{O}_{NE} = \min_{X,Y} \|M - XY^T\|_F$, in which $\|\cdot\|_F$ represents Frobenius norm of the corresponding matrix. Since M is not necessarily a Hermitian matrix, following the usual practice [Mikolov *et al.*, 2013; Tang *et al.*, 2015], we introduce a matrix Y. Similar to X, Y is also an $N \times p$ matrix, in which each line is a vector corresponding to a node. However, Y is composed of vectors of nodes when they act as *contexts* [Mikolov *et al.*, 2013; Tang *et al.*, 2015].

There are many ways to optimize this objective function, such as stochastic gradient descent often used in recommendation system to factorize a rank matrix which is usually sparse [Koren *et al.*, 2009]. Here we adopt another popular tool, i.e., singular value decomposition (SVD), to do it, like [Levy and Goldberg, 2014]. We compute SVD for M, and have $M = U_0 \Sigma_0 W_0^T$, in which $\Sigma_0 = diag(\sigma_1, \sigma_2, \cdots, \sigma_N)$ is a diagonal matrix composed of all its singular values, and U_0 and W_0 are matrices containing all its left and right singular vectors. In practice, we only need to compute its top-p singular values $\Sigma = diag(\sigma_1, \sigma_2, \cdots, \sigma_p)$ and the corresponding singular vectors U and W to compute X and Y, which can speed up the computation a lot. Then we set $X = U\Sigma^{\frac{1}{2}}, \quad Y = \Sigma^{\frac{1}{2}} W^T$. According to the property of SVD, we have $M \approx XY^T$.

3.3 Proximity Matrix

For most existing network embedding models, each model adopts a specific proximity among the nodes to learn the node vectors. In this sense, one kind of node proximity results in one network embedding model. Computing the proximity matrix M is the first step to learn the node embedding X. Any kind of proximity can be used for network embedding. Here we introduce the computation method of the proximity matrix M for some kinds of proximities. Some proximities have been used by the existing network embedding models, while the others can be exploited to build new network embedding models.

Adjacency Proximity (AP) means that if there is a link (this link can be weighted) between two nodes, then these two nodes have some similarity. Obviously, the proximity matrix is just the adjacency matrix A, i.e., $M^{(AP)} = A$.

Jaccard's Coefficient Proximity (JC) can be used to measure the similarity between two finite sets [Jaccard, 1912]. Here the proximity matrix for Jaccard's Coefficient Proximity can be computed as $M_{i,j}^{(JC)} = \frac{|nbr(v_i) \cap nbr(v_j)|}{|nbr(v_i) \cup nbr(v_j)|}$, where $nbr(\cdot)$ denotes the neighbors of the corresponding node.

Katz Proximity (KP) is defined by [Katz, 1953] based on a straightforward intuition that if there are more paths between two nodes and the paths are shorter, the two nodes are more similar. The Katz Proximity between v_i and v_j can be defined as $M_{i,j}^{(KP)} = \sum_{l=1}^{\infty} \alpha^l \cdot |paths_{i,j}^{(l)}|$, where $paths_{i,j}^{(l)}$ is the set of all l-hop paths from v_i to v_j. Thus the Katz Proximity matrix can be computed as $M^{(KP)} = \sum_{l=1}^{\infty} \alpha^l A^l = (I - \alpha A)^{-1} - I$, where $\alpha < 1$ is an attenuation factor, I is the identity matrix, and $(\cdot)^{-1}$ represents the reverse of the corresponding matrix.

In addition to the proximities listed above, there are many other proximities, such as Preferential Attachment Proximity [Barabâsi *et al.*, 2002], Adamic-Adar Proximity [Adamic and Adar, 2003], SimRank Proximity [Jeh and Widom, 2002], High Order Proximity [Benson *et al.*, 2016]. Due to the space limit, we do not list all of them here.

4 Credit Probing Network Embedding

When a network evolves, its proximity matrix M changes accordingly, thus we need to keep updating the node vectors to reflect the latest M. However, in many real world networks, we can only probe part of the nodes to update the proximity matrix and the node vectors. In this section, on the basis of the unified network embedding framework, we propose our algorithm for network embedding under partial monitoring on evolving networks. Here we take Adjacency Proximity for an example, so $M = A$, and it can be generalized to other proximities just by replacing M with corresponding proximity matrix.

4.1 Algorithm

Specifically, we can describe our problem as follows. Let $< 0, 1, ..., T >$ be a sequence of time stamps. First we select a starting time stamp, for example, t_0, and until this moment we can get the complete information of the network structure, thus we can get relatively accurate proximity matrix M_{t_0} and the results X_{t_0} and Y_{t_0}. However, at the following time stamps, we can only probe part of the nodes, leading to that we cannot get a global view over the change of the network structure, and consequently we cannot get accurate proximity matrix M_{t_i} and accurate embedding results X_{t_i} and Y_{t_i}. The challenge is how to choose the appropriate nodes, whose number cannot exceed a budget denoted as K, to probe to make the results as accurate as possible. This problem can be formulated as follows.

Problem 1. *In a network, given a time stamps sequence $< 0, 1, ..., T >$, the starting time stamp (for example, t_0), the proximity and the dimension of the embedding, we need to figure out a strategy, denoted as π, to choose at most $K < N$ nodes to probe at each following time stamp, so that it minimizes the discrepancy between the approximate distributed representations, denoted as $\hat{f}_t(V)$, and the potentially best distributed representations $f_t^*(V)$, as described by the following objective function.*

$$\mathcal{O} = \min_{\pi} \sum_{t=1}^{T} Discrepancy(f_t^*(V), \hat{f}_t(V)). \tag{1}$$

It is a sequential decision problem. Obviously, the best strategy is to capture as much "change" as possible with limited "probing budget". If we probe a node which has no change, it is a "waste". Furthermore, probing a node having a big change is better than probing a node having a small change. In a network, some nodes are more likely to change than others, so how do we decide which nodes to choose in the next step?

We propose an algorithm to solve this problem based on a kind of reinforcement learning problem, namely Multi-armed Bandit (MAB) [Auer *et al.*, 2002; Garivier and Cappé, 2011; Chen *et al.*, 2013] problem. We try to choose the "productive" nodes according to their historical "rewards". We denote the reward of each node v_i at each time stamp t_j as r_{v_i,t_j}, which is the change it bring to the proximity matrix M from the last time stamp, formulated as $r_{v_i,t_j} = \|\Delta M\|_F$. We assume that the reward of each node v_i has a distribution with an expectation μ_{v_i}. We should choose the nodes from which we have obtained good rewards, which can be regarded as *exploitation*. Besides, we should also give a chance to the nodes that we have never probed or rarely probed because they have not fully showed themselves, which can be regarded as *exploration*. To choose the nodes, we must make a trade-off between exploitation and exploration. To this end, we maintain a "credit" for each node.

Definition 3. ***Node Credit.*** *If a node v_i has been probed for T_{v_i} times by time stamp t_j, its credit C_{v_i,t_j} is defined as*

$$C_{v_i,t_j} = \hat{\mu}_{v_i,t_j} + \lambda \sqrt{\frac{\ln t_j}{T_{v_i}}}, \tag{2}$$

in which $\hat{\mu}_{v_i,t_j}$ is the empirical mean of the rewards of v_i at time stamp t_j, and can be achieved by calculating the arithmetic mean of the historical rewards. λ is used to make a trade-off between exploitation and exploration.

The credit of a node can be determined by three factors, including the historical reward, the current time stamp and the

Algorithm 1: Credit Probing Network Embedding

Input: A network G, dimension p, proximity kind, time stamp $< 0, 1, ..., T >$, parameter λ, probing budget K.

Output: approximate node representation matrix X_{t_i} at each time stamp

```
Initialize μ̂_{v_i}, T_{v_i}, C_{v_i} for each node.
foreach time stamp t_j do
    probe the nodes with the highest credit.
    update the approximate network structure Ĝ_{t_j}.
    compute the approximate proximity matrix M̂_{t_j}.
    compute X_{t_i}.
    foreach node v_i do
        update T_{v_i}
        update C_{v_i,t_j} according to Eq.2.
    end
end
```

times that it has been probed so far. From this definition we can see that a higher historical reward and a smaller number of probe times can lead to a higher credit, which is in line with our purpose of balancing exploitation and exploration. We describe our algorithm to perform network embedding under partial monitoring on evolving networks in Algorithm 1, called *Credit Probing Network Embedding (CPNE)*. At each time stamp, we select the nodes with the highest credits to probe. Then we update the structure of the network and the proximity matrix with the feedback of the probing. Next, we compute the node vectors based on the image of the network structure and the proximity matrix. Finally, we update the credits for all the nodes based on Eq.2. To start this algorithm, we should first initialize the credits. There are several methods to implement the initial phase in step 1 in the algorithm. For example, we can initialize them with the information before partial monitoring.

4.2 Theoretical Analysis

In this section, we do some theoretical analysis for our algorithm and provide an error bound to the result of our algorithm. First, we should figure out a correct metric to measure the objective function in Eq.1. It is not trivial, because when we measure the difference between two sets of embedding values, denoted as X and X^*, it makes no sense to measure the difference between their entries' concrete values with metrics such as $\|X - X^*\|_F$. Instead, we should treat the embedding matrix as a whole, and solve this problem with the geometric meaning of the matrix. We can regard matrix X, i.e., the vector representations of the nodes, as a subspace of $\mathbb{R}^N$ or a linear transformation, which is a stretching of the length and a rotation of the direction of a vector. Since X is achieved by U and S, of which U is a unitary matrix and S is a vector composed of the singular values, the magnitude of the stretching is determined by S and the angle of the rotation is determined by U. Let X^* be the network embedding result obtained by selecting the optimal node set with the highest combinatorial reward expectation, and $\hat{X}$ be the network embedding result obtained by our model. So we can compare the difference between X^* and $\hat{X}$ by their corresponding unitary matrices U^* and $\hat{U}$, and the vectors S^* and $\hat{S}$. For the vectors S^* and $\hat{S}$, we can measure their difference by $L_2\text{-}norm$ of their difference vector called *Magnitude Gap* or L_2*-loss*, which is defined as follows.

Definition 4. ***Magnitude Gap***.

$$MG = \|S^* - \hat{S}\|_2. \tag{3}$$

For the unitary matrices U^* and $\hat{U}$, we can measure their difference by their *canonical angles* [Afriat, 1957; Stewart, 1990]. Please note that X is the embedding result matrix achieved with Y in pairs, which is the vectors of the nodes acting as contexts. In other words, X is the node vectors specific to Y which is obtained by W, so when we compare the difference between U^* and $\hat{U}$ we have to compare the difference between W^* and $\hat{W}$ together, which are also unitary matrices associated with U^* and $\hat{U}$ respectively. Let $\theta_1, \theta_2, ..., \theta_p$ be the canonical angles of U^* and $\hat{U}$, and $\phi_1, \phi_2, ..., \phi_p$ be the canonical angles of W^* and $\hat{W}$. To compare them, we construct two diagonal matrices $\Theta = diag(\theta_1, \theta_2, ..., \theta_p)$ and $\Phi = diag(\phi_1, \phi_2, ..., \phi_p)$. We use *Angle Gap* to measure the difference between (U^*, W^*) and $(\hat{U}, \hat{W})$, which is defined as follows.

Definition 5. ***Angle Gap***.

$$\begin{aligned} AG &= \sqrt{\|\sin\Theta\|_F^2 + \|\sin\Phi\|_F^2} \\ &= \sqrt{\frac{\|P_{U^*} - P_{\hat{U}}\|_F^2 + \|P_{V^*} - P_{\hat{V}}\|_F^2}{2}}, \end{aligned} \tag{4}$$

where P_{U^*} is the orthogonal projection operator of U^*, which can be achieved by $P_{U^*} = U^*U^{*\dagger} = U^*(U^{*T}U^*)^{-1}U^{*T}$, in which $(\cdot)^{-1}$ is the inverse of the corresponding matrix and $(\cdot)^\dagger$ is Moore-Penrose pseudoinverse. In the same way, we can get $P_{\hat{U}}$, P_{V^*} and $P_{\hat{V}}$ with $\hat{U}$, V^* and $\hat{V}$ respectively.

Obviously, for both of the two metrics Magnitude Gap and Angle Gap, the smaller their values, the better the algorithm. To be clear, we add subscript t_j to the notation to denote the value at time stamp t_j. For example, MG_{t_j} represents Magnitude Gap at t_j. We define the accumulative losses of MG and AG as $L_{MG} = \sum_{t_j=1}^{T} MG_{t_j}$ and $L_{AG} = \sum_{t_j=1}^{T} MG_{t_j}$. Then we will give a theoretical upper bound for each loss.

Let $D = (v_{i_1}, v_{i_2}, \cdots, v_{i_p})$ be a node set, and D^* be the optimal node set which have the highest combinatorial reward expectation among all possible node sets. Please note that the combinatorial reward of a set is not necessarily the sum of the individual rewards of the nodes in the set. We define $\Delta_{v_i}^{min} = \mu_{D^*} - \max\{\mu_D | D \neq D^*, v_i \in D\}$, and $\Delta_{v_i}^{max} = \mu_{D^*} - \min\{\mu_D | D \neq D^*, v_i \in D\}$, in which μ_D is the combinatorial reward expectation. Furthermore, we define $\Delta^{max} = \max \Delta_{v_i}^{max}$. If we normalize the rewards to $[0, 1]$, we can get the upper bounds for L_{MG} and L_{AG}. Specifically, for L_{MG}, we have the following theorem.

Theorem 1.

$$L_{MG} \leq \sum_{v_i \in V} \frac{4\lambda^2 N^2 \ln T}{\Delta_{v_i}^{min}} + (1 + \sum_{d=1}^{\infty} d^{1-2\lambda^2}) N \Delta^{max}.$$

To prove Theorem 1, we need to use an important theory of matrix perturbation, namely Mirsky theorem. Due to the space limit, we do not repeat it here. One can refer to [Mirsky, 1960] for details.

Proof. Let $\Delta M = M^* - \hat{M}$ be the difference matrix between the optimal proximity matrix and the proximity matrix obtained by CPNE. Then we can prove the following inequality by the combinatorial multi-armed bandit theory [Chen *et al.*, 2013].

$$\sum_{t_j=1}^{T} \|\Delta M_{t_j}\|_F \leq \sum_{v_i \in V} \frac{4\lambda^2 N^2 \ln T}{\Delta_{v_i}^{min}} + (1 + \sum_{d=1}^{\infty} d^{1-2\lambda^2}) N \Delta^{max}. \tag{5}$$

Then we have

$$\begin{aligned} L_{MG} &= \sum_{t_j=1}^{T} MG_{t_j} = \sum_{t_j=1}^{T} \|S_{t_j}^* - \hat{S_{t_j}}\|_F \\ &= \sum_{t_j=1}^{T} \sqrt{\sum_{i=1}^{p} (\sigma_{i,t_j}^* - \hat{\sigma}_{i,t_j})^2} \leq \sum_{t_j=1}^{T} \|\Delta M_{t_j}\|_F. \end{aligned} \tag{6}$$

The last inequality is derived from Mirsky theorem. We can get Theorem 1 by combining Formula 5 and Formula 6. □

For L_{AG}, we have the following theorem.

Theorem 2.

$$L_{AG} \leq \frac{\sqrt{\sum_{v_i \in V} \frac{8\lambda^2 N^2 \ln T}{\Delta_{v_i}^{min}} + 2(1 + \sum_{d=1}^{\infty} d^{1-2\lambda^2}) N \Delta^{max}}}{\delta},$$

in which the meaning of δ is explained in the following proof.

To prove Theorem 2, we need to use another important theory of matrix perturbation, namely Wedin theorem. One can refer to [Wedin, 1972] for details.

Proof. Based on our model we have

$$(U^* U_0^*)^T M^* (W^* W_0^*) = \begin{pmatrix} \Sigma^* & 0 \\ 0 & \Sigma_0^* \end{pmatrix},$$

and

$$(\hat{U}\hat{U}_0)^T \hat{M} (\hat{W}\hat{W}_0) = \begin{pmatrix} \hat{\Sigma} & 0 \\ 0 & \hat{\Sigma}_0 \end{pmatrix},$$

in which Σ_0^* is the diagonal matrix composed of the rest $(N-p)$ singular values of M^*, and U_0^* and W_0^* are the corresponding left singular vectors and right singular vectors. The notations in the second equation have similar meanings. We can find a $\delta > 0$ such that $\min |\sigma(\hat{\Sigma_{t_j}}) - \sigma(\Sigma_{0,t_j}^*)| \geq \delta$ and $\min \sigma(\hat{\Sigma_{t_j}}) \geq \delta$. Then applying Wedin theorem, we have

$$\begin{aligned} L_{AG} &= \sum_{t_j=1}^{T} AG_{t_j} = \sum_{t_j=1}^{T} \|\sin \Theta_{t_j}\|_F^2 + \|\sin \Phi_{t_j}\|_F^2 \\ &\leq \sum_{t_j=1}^{T} \frac{\sqrt{\|M_{t_j}^* \hat{W}_{t_j} - \hat{U}_{t_j} \hat{\Sigma}_{t_j}\|_F^2 + \|M_{t_j}^* \hat{U}_{t_j} - \hat{W}_{t_j} \hat{\Sigma}_{t_j}\|_F^2}}{\delta} \\ &\leq \sum_{t_j=1}^{T} \frac{\sqrt{2\|\Delta M_{t_j}\|_F^2}}{\delta}. \end{aligned} \tag{7}$$

Then we can get Theorem 2 by combining Formula 5 and Formula 7. □

Usually, if we have a larger K, we will have a smaller Δ^{max} and a larger $\Delta_{v_i}^{min}$, so we can get a lower error bound.

5 Experiments

In this section we evaluate the performance of our model on several real world datasets from two aspects. First, we evaluate our model's ability to approach the potential optimal values, which is to minimize Magnitude Gap and Angle Gap. Then we evaluate the performance of our model for one of the most important real world problems, namely link prediction. We use the following two datasets to conduct our experiments.

AS. The graph of router comprising Internet is organized into subgraphs called autonomous systems (AS). An AS exchanges traffic flows with its peers, and we can construct a communication network from the Border Gateway Protocol logs. AS dataset contains 9 networks, 1 per week between March 31 2001 and May 26 2001 [Leskovec *et al.*, 2005]. AS dataset contains 10,900 nodes and 19,318 temporal edges.

Wechat. This dataset is collected from a large social network platform, namely Wechat[1]. We randomly select a user with the mean degree of all the WeChat users, and extract its one-hop and two-hop neighbors to construct a subnetwork, containing 15,320 nodes. All the information about the users is erased, and only the topological structure is kept for scientific research. If two users establish a friend relationship between each other, there will be a temporal edge record in the data. We collect the temporal edges from 2015-12-31 23:59:59 to 2017-05-31 23:59:59, and set the snapshot of the WeChat network at the beginning as the initial graph. We set the time interval as one month, then we get 18 time stamps. This network contains 226,988 temporal edges.

To save the space, we only show the result of the first experiment on AS, and that of the second experiment on Wechat.

5.1 Approaching the Potential Optimal Values

This experiment is to evaluate the performance for approaching the potential optimal values of network embedding. The metrics for this experiment are MG and AG. We use Adjacency Proximity to conduct the experiments. We compute S^*, U^* and W^* with the true network structure. We take the first two time stamps for initialization. We use the following four baseline models for the first experiment.

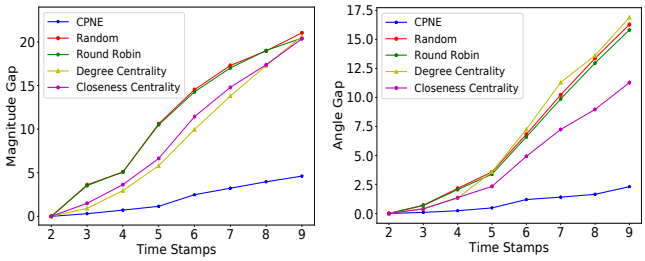
Random. We uniformly select nodes from a network to probe at each time stamp.

Round robin. We cycle through the nodes and probing them in this order.

Degree centrality. At each time stamp, we choose the nodes with the highest values of degree centrality in the latest image of the network structure to probe.

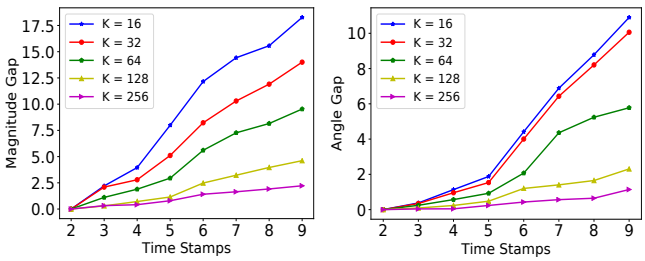
Closeness centrality. This method is similar to Degree Centrality except that it takes the nodes' closeness centrality as the score to choose the nodes.

[1] http://www.wechat.com



(a) Magnitude Gap on AS. (b) Angle Gap on AS.

Figure 1: Metric values on AS. The left sub-figure shows the values of MG, while the right one shows those of AG. Each curve represents an algorithm. Since MG and AG are both cumulative error values, the curves are non-decreasing, and the smaller the metric value, the better the algorithm.



(a) Magnitude Gap on AS. (b) Angle Gap on AS.

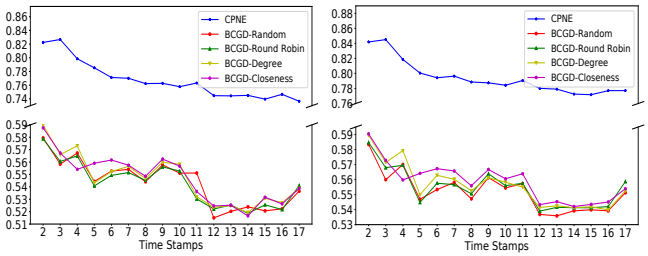
Figure 2: Metric values on AS with different K values. The left sub-figure shows the values of MG, while the right one shows those of AG. Each curve represents a K value. Usually, the larger the K, the lower the MG and AG.

Comparison of the Models

We compute all the metric values of our model and the baseline models at each time stamp on AS. We set $K = 128$, $\lambda = 1$ and $p = 10$ for all the models. Then we plot the values of each metric in one sub-figure in Fig.1, in which the X axes represent the time stamps while the Y axes represent the metric values. Then we can see that our model shows significant superiority over all the baseline models in both metrics. From Fig.1 we can also find that for the four baseline models, the performances of Closeness Centrality is somehow slightly better than the other three baseline models. This may be because the nodes with higher closeness centrality usually change more than other nodes in this network, and deserve more attention when the network evolves.

Parameter Sensitivity

In this section, we take a look at the sensitivity of the hyper-parameters K. K is the budget of the number of the nodes that we can probe at each time stamp. Usually, the larger the K, the smaller the metric values may be, and the more accurate the learned node vectors. We set $K = 16$, 32, 64, 128, 256 sequentially, and run CPNE on AS. We plot the results in Fig.2. From Fig.1 and Fig.2 we can see that even if we only choose 16 nodes to probe at each time stamp with our algorithm, we can still achieve a better performance than most other algorithms when they choose 128 nodes to probe at each time stamp.



(a) K = 500. (b) K = 1000.

Figure 3: AUC values at each time stamp on WeChat dataset.

5.2 Link Prediction

In addition to the experiments of testing our model's performance to approach the potential optimal embedding values, we also test the performance of our model for a real world application, i.e., link prediction.

We take BCGD [Zhu *et al.*, 2016] as our baseline. However, BCGD is based on the assumption that all the changes of the network can be perceived. To make a valid comparison, we set the four node probing strategies mentioned above for BCGD separately. Thus we get four baseline methods. We use the node vectors learned by our model and the baseline models at each time stamp (except the last time stamp) to predict the new links in the next time interval. We take the new links emerging in the next time interval as positive instances, and randomly sample the equal number of node pairs that never be linked during the next time interval as the negative instances. We use the dot product of the vectors of two nodes to measure their probability of being linked. We adopt *AUC* (Area Under the receiver operating characteristic Curve) as our metric. To be fair, we set $p = 20$ for all the models. For λ of our model, we set it to be 1. For λ, ζ, and δ of BCGD, we set them in accordance with the paper presenting it.

Experimental Results

We conduct the experiment for $K = 500$ and 1000, and plot the experimental results in Fig.3(a) and Fig.3(b) respectively, in which the X axes represent the time stamps while the Y axes represent the AUC values. We can see that our model outperforms all the baseline models significantly. Specifically, CPNE gets improvements of 36.10% and 39.12% over the best baseline for $K = 500$ and 1000 respectively at the last time stamp for link prediction.

6 Conclusion

In this paper, we study the problem of network embedding under partial monitoring for evolving networks. There are still many challenges in network embedding on dynamic networks. For the future work, we will try other reinforcement learning algorithms to solve such problems. In addition, how to employ deep learning models to learning embedding values in such a setting is also interesting and meaningful.

References

[Adamic and Adar, 2003] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

[Afriat, 1957] Sydney N Afriat. Orthogonal and oblique projectors and the characteristics of pairs of vector spaces. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 53, pages 800–816. Cambridge Univ Press, 1957.

[Anagnostopoulos *et al.*, 2012] Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, Eli Upfal, and Fabio Vandin. Algorithms on evolving graphs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 149–160. ACM, 2012.

[Auer *et al.*, 2002] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[Bahmani *et al.*, 2012] Bahman Bahmani, Ravi Kumar, Mohammad Mahdian, and Eli Upfal. Pagerank on an evolving graph. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 24–32. ACM, 2012.

[Barabâsi *et al.*, 2002] Albert-Laszlo Barabâsi, Hawoong Jeong, Zoltan Néda, Erzsebet Ravasz, Andras Schubert, and Tamas Vicsek. Evolution of the social network of scientific collaborations. *Physica A: Statistical mechanics and its applications*, 311(3):590–614, 2002.

[Benson *et al.*, 2016] Austin R Benson, David F Gleich, and Jure Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.

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

[Chen *et al.*, 2013] Wei Chen, Yajun Wang, and Yang Yuan. Combinatorial multi-armed bandit: General framework, results and applications. In *Proceedings of the 30th ICML*, pages 151–159, 2013.

[Garivier and Cappé, 2011] Aurélien Garivier and Olivier Cappé. The kl-ucb algorithm for bounded stochastic bandits and beyond. In *COLT*, pages 359–376, 2011.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD*, pages 855–864. ACM, 2016.

[Han and Tang, 2017] Yu Han and Jie Tang. Who to invite next? predicting invitees of social groups. In *IJCAI*, pages 3714–3720, 2017.

[Jaccard, 1912] Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.

[Jeh and Widom, 2002] Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 538–543. ACM, 2002.

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

[Koren *et al.*, 2009] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

[Leskovec *et al.*, 2005] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 177–187. ACM, 2005.

[Levy and Goldberg, 2014] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

[Mikolov *et al.*, 2013] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[Mirsky, 1960] Leon Mirsky. Symmetric gauge functions and unitarily invariant norms. *The quarterly journal of mathematics*, 11:50–59, 1960.

[Perozzi *et al.*, 2014] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD*, pages 701–710. ACM, 2014.

[Qiu *et al.*, 2017] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifyingdeepwalk, line, pte, and node2vec. *arXiv preprint arXiv:1710.02971*, 2017.

[Song *et al.*, 2009] Han Hee Song, Tae Won Cho, Vacha Dave, Yin Zhang, and Lili Qiu. Scalable proximity estimation and link prediction in online social networks. In *Proceedings of the 9th ACM SIGCOMM*, pages 322–335. ACM, 2009.

[Stewart, 1990] Gilbert W Stewart. Matrix perturbation theory. 1990.

[Tang and Liu, 2009] Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD*, pages 817–826. ACM, 2009.

[Tang *et al.*, 2015] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. ACM, 2015.

[Tong *et al.*, 2008] Hanghang Tong, Spiros Papadimitriou, Philip S Yu, and Christos Faloutsos. Proximity tracking on time-evolving bipartite graphs. In *Proceedings of the 2008 SIAM International Conference on Data Mining*, pages 704–715. SIAM, 2008.

[Tu *et al.*, 2016] Cunchao Tu, Hao Wang, Xiangkai Zeng, Zhiyuan Liu, and Maosong Sun. Community-enhanced network representation learning for network analysis. *arXiv preprint arXiv:1611.06645*, 2016.

[Wang *et al.*, 2017] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community preserving network embedding. 2017.

[Wedin, 1972] Per-Åke Wedin. Perturbation bounds in connection with singular value decomposition. *BIT Numerical Mathematics*, 12(1):99–111, 1972.

[Zheng *et al.*, 2016] Vincent W Zheng, Sandro Cavallari, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. From node embedding to community embedding. *arXiv preprint arXiv:1610.09950*, 2016.

[Zhou *et al.*, 2018] Le-kui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic network embedding by modeling triadic closure process. In *AAAI*, 2018.

[Zhu *et al.*, 2016] Linhong Zhu, Dong Guo, Junming Yin, Greg Ver Steeg, and Aram Galstyan. Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE Transactions on Knowledge and Data Engineering*, 28(10):2765–2777, 2016.