The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Dynamic Network Embedding by Modeling Triadic Closure Process

Lekui Zhou,[1] Yang Yang,[1]* Xiang Ren,[2] Fei Wu,[1] Yueting Zhuang[1]
[1] Department of Computer Science and Technology, Zhejiang University
[2] Department of Computer Science, University of Southern California
{luckiezhou, yangya, wufei, yzhuang}@zju.edu.cn, xiangren@usc.edu

Abstract

Network embedding, which aims to learn the low-dimensional representations of vertices, is an important task and has attracted considerable research efforts recently. In real world, networks, like social network and biological networks, are dynamic and evolving over time. However, almost all the existing network embedding methods focus on static networks while ignore network dynamics.

In this paper, we present a novel representation learning approach, *DynamicTriad*, to preserve both structural information and evolution patterns of a given network. The general idea of our approach is to impose *triad*, which is a group of three vertices and is one of the basic units of networks. In particular, we model how a *closed triad*, which consists of three vertices connected with each other, develops from an *open triad* that has two of three vertices not connected with each other. This triadic closure process is a fundamental mechanism in the formation and evolution of networks, thereby makes our model being able to capture the network dynamics and to learn representation vectors for each vertex at different time steps.

Experimental results on three real-world networks demonstrate that, compared with several state-of-the-art techniques, *DynamicTriad* achieves substantial gains in several application scenarios. For example, our approach can effectively be applied and help to identify telephone frauds in a mobile network, and to predict whether a user will repay her loans or not in a loan network.

Introduction

There is nothing permanent except change.
— *Heraclitus*

The goal of network embedding, also known as network representation learning, is to project a network to a low-dimensional space, where each vertex can be presented as a single point in the learned latent space. Network embedding has attracted considerable research efforts recently by benefiting social and biological network research. Most existing work, varying from Bayesian inference (Ho, Yin, and Xing 2016; Hoff, Raftery, and Handcock 2012) , multidimensional scaling (Sarkar and Moore 2005), matrix factorization (Erds, Gemulla, and Terzi 2014; Qi, Aggarwal, and Huang 2013; Zhu et al. 2014), to deep learning (Wang, Cui, and Zhu 2016; Grover and Leskovec 2016), were focusing on static networks, where the structure of vertices are fixed.

However, in real world, network is dynamic and evolving over time. Users develop their connections to each other in social networks, while interactions between proteins vary over time in biological networks. Representation learning algorithms that ignore the dynamics of a given network can hardly capture sufficient information.

Figure 1 illustrates the dynamic structure of a network at different time steps. We take social network as an example, where each vertex indicates a user, each edge denotes a friendship between two users, and the weight associated with each edge indicates tie strength, or the energy that users spend to keep their relationship (e.g., number of calls between two users in a mobile network). At time step t, user A and user B have similar structures: they both have three friends, who do not know each other. In addition, both of them spend most energy on a particular friend (i.e., spending 0.8 energy on a particular friend while spending 0.2 energy on others in total). Therefore, traditional network embedding methods, like DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) and node2vec (Grover and Leskovec 2016), will construct similar representation vectors for A and B. However, their evolution patterns are different. For example, at time step $t+1$, two pairs of A's friends connect to each other, while there is still no links among B's friends. In addition, user B distracts her energy to other friends at time step $t+1$, while user A keeps focusing on developing her relationship with the same friend as at time step t. Different evolution patterns of users reflects the differences between their own characters and social strategies, which are used to meet their social needs like developing connections to others. For instance, user A tends to introduce her friends to each other, while user B prefers to let her friends stay tight in their own communities. Therefore, whether the learned representations can well reflect the evolution patterns of vertices is a critical requirement for network embedding methods.

In this paper, our goal is to learn the embedding vectors for vertices by capturing the evolutionary structure properties of a network. That is, given a sequence of network snapshots $G_1, \cdots, G_T$ from time steps 1 to T, we aim to design an objective that learn the embedding vector $\boldsymbol{u}_i^t$ for vertex i

*Corresponding author: Yang Yang, yangya@zju.edu.cn

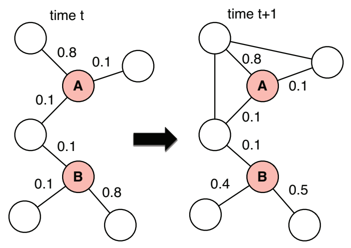


Figure 1: An illustration of dynamic social network. User A and B have different social strategies. For example, user A tends to introduce her friends to connect with each other, while user B tends to keep her friends in their own communities.

at each time step t.

However, it is very challenging to *preserve the temporal information* of dynamic networks. Existing work has a limited development of modeling how the latent space evolves over time. A typical solution used by most of current models is to assume that vertices will only move smoothly in the latent space over time by adding a regularization term in objective function (Sarkar and Moore 2005; Zhu et al. 2014). However, this assumption does not always hold as the structure of some vertex may evolve sharply. Besides, this method cannot capture the difference between the evolution patterns of vertices.

In this paper, we propose a novel dynamic network embedding approach, namely *DynamicTriad*. The general idea of our model is to impose *triad* (i.e., a group of three vertices) to model the dynamic changes of network structures. Roughly speaking, there are two types of triads: *closed triads* and *open triads*. In a closed triad, for any two vertices, there is a relationship between them. In an open triad, there are only two relationships, which means that two of the three vertices are not connected with each other. We model how a closed triad develops from an open triad, which is called as *triad closure process* and is a fundamental mechanism in the formation and evolution of dynamic networks (Coleman 1994; Huang et al. 2015). In particular, we design a uniform framework to quantify the probability of an open triad develops to a closed triad, and learn embedding vectors of each vertex at different time steps jointly.

It is worthwhile to highlight our contributions as follows:

- We propose a novel representation learning model for dynamic networks. Our model is able to preserve the structural information and evolution pattern of a network.
- We develop a semi-supervised learning algorithm for parameter estimation efficiently.
- We construct experiments over three data sets and six groups of application scenarios. For instance, we apply our approach to identify telephone frauds in a mobile network, and to predict users who will not repay their loans in a loan network. Experimental results show that our model can achieve significant improvements over several state-of-the-art baselines (e.g., improve F1 by 34.4% on link prediction).

Problem Definition

In this section, we give necessary definitions used throughout this paper. We consider a set of M vertices $V = \{v_1, \cdots, v_M\}$ and the set of undirected edges among these users $E = \{e_{ij}\}$, where each edge e_{ij} indicates a relationship (e.g., friendship) between v_i and v_j. In most cases, edges will evolve over time instead of being constant. For instance, in a social network, a user may continued to extend her circles by developing friendships with others. Formally, given T time steps, we define dynamic networks as follows:

Definition 1. Dynamic network. A series of a dynamic network snapshots is a set of undirected graphs $\{G^1, \cdots, G^T\}$, where $G^t = (V, E^t, W^t)$ $(1 \le t \le T)$ represents how vertices are connected at time t. Each edge $e^t_{ij} \in E^t$ is associated with an edge weight $w^t_{ij} := \tilde{w}^t(e_{ij})$, where $\tilde{w}^t : E^t \mapsto \mathbb{R}^+$ is a function mapping from edges to positive real values.

Our goal is to learn the low-dimensional representations of users by capturing the evolutionary structure properties of a network and the social strategies of users. We define this problem as follows:

Definition 2. Dynamic network embedding. Given a series of dynamic networks $\{G^1, \cdots, G^T\}$, dynamic network embedding aims to learn a mapping function $f^t : v_i \mapsto \mathbb{R}^d$ for each time step t, where d is a positive integer indicating the number of embedding dimensions. The objective of the function f^t is to preserve the similarity between v_i and v_j on both the network structure at time step t and their tendencies to develop relationships with others in the future.

In the rest of this paper, for simplification, we define $\boldsymbol{u}^t_i := f^t(v_i)$, and $\boldsymbol{u} = \{\boldsymbol{u}^t_i\}_{i=\{1,\cdots,M\},t=\{1,\cdots,T\}}$. Note that, different from previous network embedding tasks, which focus on static networks, our task aims to preserve the evolutionary network structures.

Our Approach

Model Description

In this section, we present a framework, *DynamicTriad*, which is capable of learning desirable representations for users in dynamic networks. Overall, the objective of *DynamicTriad* is modeling the *triadic closure process*, which describes how *open triads* evolve into *closed traids*, and reflects the difference between characters of different vertices. We take social network as an example to introduce describe our model, which can also be applied on other types of networks. An implementation of the model is publicly available [1].

Triadic closure process. We begin with an example of open triads (v_i, v_j, v_k) at time t, where user v_i and v_j do not know each other but they are both friends of v_k (i.e., $e_{ik}, e_{jk} \in E^t$ and $e_{ij} \notin E^t$). Now, the user v_k will decide whether or not to introduce v_i and v_j, let them know each other, and build a connection between them at the next time step $t+1$. We naturally assume v_k will make her decision based on how

[1] https://github.com/luckiezhou/DynamicTriad

close she is with v_i and v_j (in the latent space), which is quantified by a d-length vector $\boldsymbol{x}_{ijk}^t$ as

$$\boldsymbol{x}_{ijk}^t = w_{ik}^t * (\boldsymbol{u}_k^t - \boldsymbol{u}_i^t) + w_{jk}^t * (\boldsymbol{u}_k^t - \boldsymbol{u}_j^t) \qquad (1)$$

where w_{ik}^t indicates the tie strength of v_i and v_k at time t, and $\boldsymbol{u}_i^t$ is the embedding vector of v_i at time t. Furthermore, we define a *social strategy parameter* $\boldsymbol{\theta}$, which is a d-dimension vector to extract social strategy information embedded in each node's latent vector.

Based on the above definitions, we define the probability that the open triad (v_i, v_j, v_k) evolves into a closed triad, i.e., v_i and v_j will develop a link between them at time $t+1$, under the introduction (or influence) by v_k, as

$$P_{\mathtt{tr}}^t(i,j,k) = \frac{1}{1+\exp(-\langle \boldsymbol{\theta}, \boldsymbol{x}_{ijk}^t \rangle)} \qquad (2)$$

One thing worth to mention is that v_i and v_j might be introduced by several of their common friends. Thus, our next aim is to jointly model how multiple open triads, with a common pair of unlinked vertices, evolve. To do that, we define set $B^t(i,j)$ as v_i and v_j's common neighbors at time step t, and define a vector $\boldsymbol{\alpha}^{t,i,j} = (\alpha_k^{t,i,j})_{k \in B^t(i,j)}$, where $\alpha_k^{t,i,j} = 1$ if the open triad (v_i, v_j, v_k) will develop into a closed triad at $t+1$. In other words, v_i and v_j will become friends under the influence of v_k. Straightforwardly, once (v_i, v_j, v_k) becomes closed, all open triads relevant to v_i and v_j will become closed. Thus, by further assuming independence between the influence of each common friend to v_i and v_j's potential link, we define the probability that a new link e_{ij} will be created at time step $t+1$ as

$$\begin{aligned} P_{\mathtt{tr}_+}^t(i,j) = \sum_{\boldsymbol{\alpha}^{t,i,j} \neq \mathbf{0}} \prod_{k \in B^t(i,j)} (P_{\mathtt{tr}}^t(i,j,k))^{\alpha_k^{t,i,j}} \times \\ (1 - P_{\mathtt{tr}}^t(i,j,k))^{(1-\alpha_k^{t,i,j})} \end{aligned} \qquad (3)$$

Meanwhile, if v_i and v_j have not been influenced by any of their common friends, the edge e_{ij} will not be created. We define its probability as

$$P_{\mathtt{tr}_-}^t(i,j) = \prod_{k \in B^t(i,j)} (1 - P_{\mathtt{tr}}^t(i,j,k)) \qquad (4)$$

To put the above two possible evolution traces of the open triad (v_i, v_j, v_k) together, we define the set $S_+^t = \{(i,j) | e_{ij} \notin E^t \wedge e_{ij} \in E^{t+1}\}$ to indicate the links being successfully created at time step $t+1$, and let the set $S_-^t = \{(i,j) | e_{ij} \notin E^t \wedge e_{ij} \notin E^{t+1}\}$ indicate those not being created. We then define the loss function of triad closure process as the negative log likelihood of the data:

$$\begin{aligned} L_{\mathtt{tr}}^t = &- \sum_{(i,j) \in S_+^t} \log P_{\mathtt{tr}_+}^t(i,j) \\ &- \sum_{(i,j) \in S_-^t} \log P_{\mathtt{tr}_-}^t(i,j) \end{aligned} \qquad (5)$$

Social homophily and temporal smoothness . We utilize two more assumptions to strengthen *DynamicTriad*: *social homophily* and *temporal smoothness*. Social homophily suggests that highly connected vertices should be embedded closely in the latent representation space. Formally, we define the distance between two vertices v_j and v_k's embedding $\boldsymbol{u}_j^t$ and $\boldsymbol{u}_k^t$ as

$$g^t(j,k) = ||\boldsymbol{u}_j^t - \boldsymbol{u}_k^t||_2^2 \qquad (6)$$

At the current time step t, we divide all pairs of vertices into two sets, namely edges $E_+^t = E^t$ and non-edges $E_-^t = \{e_{jk} | j \in \{1, \cdots, N\}, k \in \{1, \cdots, N\}, j \neq k, e_{jk} \notin E^t\}$. According to the homophily hypothesis, if two vertices are linked with each other, they tend to be embedded closer in the latent-representation space, which results in our ranking loss-based function for the social homophily as

$$L_{\mathtt{sh}}^t = \sum_{\substack{(j,k) \in E_+^t \\ (j',k') \in E_-^t}} h(w_{jk}, [g^t(j,k) - g^t(j',k') + \xi]_+) \qquad (7)$$

where $[x]_+ = \max(0, x)$ for any real number x, and $\xi \in \mathbb{R}^+$ is the margin value. Function $h(\cdot,\cdot)$ combines the weight and the measure of discrepancy, and is commonly defined as $h(w,x) = w \cdot x$.

It is natural to assume that a network will evolve smoothly over time, instead of being totally rebuilt in each time step. Therefore, we define the *temporal smoothness* by minimizing the Euclidean distance between embedding vectors in adjacent time steps. Formally, the corresponding loss function is

$$L_{\mathtt{smooth}}^t = \begin{cases} \sum_{i=1}^N ||\boldsymbol{u}_i^t - \boldsymbol{u}_i^{t-1}||_2^2 & t > 1 \\ 0 & t = 1 \end{cases} \qquad (8)$$

Therefore, the overall optimization problem given the first T time steps is

$$\mathop{\arg\min}_{\{\boldsymbol{u}_i^t\}, \boldsymbol{\theta}} \sum_{t=1}^T L_{\mathtt{sh}}^t + \beta_0 L_{\mathtt{tr}}^t + \beta_1 L_{\mathtt{smooth}}^t \qquad (9)$$

Notice that regularization terms are omitted since they are handled by normalization techniques during training.

Model Learning

In this section, we will introduce how to learn the proposed model in detail. Generally, we aim to find a configuration of model parameters $\{\boldsymbol{\theta}, \boldsymbol{u}\}$ that optimize Eq. (9).

Log-likelihood approximation. We start with the loss function of triad closure process $L_{\mathtt{tr}}^t$, which consists of two terms as Eq. (5) shows. The second term (i.e., $-\sum_{(i,j) \in S_-^t} \log P_{\mathtt{tr}_-}^t(i,j)$) can be easily computed as

$$\sum_{\substack{(i,j) \in S_-^t \\ k \in B^t(i,j)}} \langle \boldsymbol{\theta}, \boldsymbol{x}_{ijk}^t \rangle + \log(1 + \exp(-\langle \boldsymbol{\theta}, \boldsymbol{x}_{ijk}^t \rangle)) \qquad (10)$$

However, one issue here is that the other term, $-\sum_{(i,j) \in S_+^t} \log P_{\mathtt{tr}_+}^t(i,j)$, is intractable as the hidden variable $\boldsymbol{\alpha}^{t,i,j}$ has an exponential number of possible values. To solve this, we adopt a procedure similar to the expectation-maximization (EM) algorithm (Dempster, Laird, and Rubin 1977) that optimizes the upper bound of this intractable term. Specifically, we compute the upper bound for this term as

$$
\begin{aligned}
&-\sum_{(i,j)\in S_+^t} \log P_{\mathtt{tr}_+}^t(i,j) \\
\le &\sum_{\substack{(i,j)\in S_+^t \\ k\in B^t(i,j)}} C_{ijk}^t \langle \boldsymbol{\theta}, x_{ijk}^t\rangle + \log(1+\exp(-\langle \boldsymbol{\theta}, x_{ijk}^t\rangle))
\end{aligned}
\tag{11}
$$

where

$$
C_{ijk}^t = 1 - \frac{P_{\mathtt{tr}}^t(i,j,k;\boldsymbol{\theta}^{(n)},\boldsymbol{u}^{(n)})}{1-\prod\limits_{k^*\in B^t(i,j)}(1-P_{\mathtt{tr}}^t(i,j,k^*;\boldsymbol{\theta}^{(n)},\boldsymbol{u}^{(n)}))}
\tag{12}
$$

which can be pre-calculated given i, j, k and t at the beginning of an iteration, and $\boldsymbol{\theta}^{(n)}$ and $\boldsymbol{u}^{(n)}$ are the model parameters under the current iteration. The derivation details are omitted due to the limited space.

Combining Eq. (10) and Eq. (11), we have got an upper bound for $L_{\mathtt{tr}}^t$

$$
\begin{aligned}
&\sum_{\{(i,j)|e_{ij}\notin E^t\},k\in B^t(i,j)} \log(1+\exp(-\langle \boldsymbol{\theta}, \boldsymbol{x}_{ijk}^t\rangle)) \\
&\quad + (I(e_{ij}\in E^{t+1})C_{ijk}^t + I(e_{ij}\notin E^{t+1}))\langle \boldsymbol{\theta}, \boldsymbol{x}_{ijk}^t\rangle
\end{aligned}
\tag{13}
$$

Sampling. It is expensive, especially for large graphs, to calculate all the combinations of positive and negative samples as defined in Eq. (7). In order to address this problem, we utilize a sampling technique, which is similar with (Wang et al. 2014).

Specifically, for social homophily loss function, given a positive sample (edge) e_{jk} at time step t, we first randomly choose a vertex $v_{j'}$ among v_j and v_k (i.e., $j'\in\{j,k\}$). We then randomly sample another vertex $v_{k'}$ from other vertices, which stratifies that $e_{j'k'} \notin E^t$ is a valid negative sample.

Repeat the sampling process for each edge e_{jk}, and we define the training set as $E_{\mathtt{sh}}^t = \{(j,k,j',k')|(j,k)\in E^t\}$, and the loss function can be represented as

$$
L_{\mathtt{sh},1}^t = \sum_{(j,k,j',k')\in E_{\mathtt{sh}}^t} h(w_{jk}, [g^t(j,k)-g^t(j',k')+\xi]_+)
\tag{14}
$$

For the loss function of triad closure process, for each $(j,k,j',k')\in E_{\mathtt{sh}}^t$, we first randomly choose a vertex from $\{v_j, v_k\}$, without loss of generality, we assume v_k is chosen. Then we aim to sample a vertex v_i, where $e_{ik}\in E^t$ and $e_{ij}\notin E^t$, so that we obtain an open triad (i,j,k) where v_k connects both v_i and v_j. The open triad can be either a positive or a negative instance and can be used to train our model, depending on whether it closes in the next time step (i.e. whether $e_{ij}\in E^{t+1}$). Let $E_{\mathtt{tr}}^t = \{(j,k,j',k',i)|(j,k,j',k')\in E_{\mathtt{sh}}^t\}$, combing Eq. (13), the loss function for triad closure process can be represented as

$$
\begin{aligned}
L_{\mathtt{tr},1}^t = &\sum_{(j,k,j',k',i)\in E_{\mathtt{tr}}^t} \log(1+\exp(-\langle \boldsymbol{\theta}, \boldsymbol{x}_{ijk}^t\rangle)) \\
&+ (I(e_{ij}\in E^{t+1})C_{ijk}^t + I(e_{ij}\notin E^{t+1}))\langle \boldsymbol{\theta}, \boldsymbol{x}_{ijk}^t\rangle
\end{aligned}
\tag{15}
$$

As $L_{\mathtt{tr},1}^t$ relies on information at time step $t+1$, there is a special case for the last time step T:

$$
L_{\mathtt{tr},2}^t = \begin{cases} L_{\mathtt{tr},1}^t, & t<T \\ 0, & t=T \end{cases}
\tag{16}
$$

Putting it all together, the overall loss function at training step is

$$
L = \sum_{t=1}^{T}(L_{\mathtt{sh},1}^t + \beta_0 L_{\mathtt{tr},2}^t) + \beta_1 \sum_{t=1}^{T-1}\sum_{i=1}^{N} ||\boldsymbol{u}_i^{t+1}-\boldsymbol{u}_i^t||_2^2
\tag{17}
$$

where the first two terms share a same set of samples, and the third term corresponds to the temporal smoothness. Note that due to space limitation, the derivation of the gradients are omitted, which can be found in the project's online homepage.

```
Input: Dynamic network $G_1, \cdots, G_T$;
Output: Embedding vectors for graph nodes $\boldsymbol{u}_i^t$;
Initialize model parameters $\boldsymbol{\theta}^{(n)}, \boldsymbol{u}^{(n)}$ randomly;
Sample $E'$ from all existing edges in $G_i$
for $n \leftarrow 1$ to N do
    Sample $E_{\mathtt{sh}}^t$ according to $E'$
    Sample $E_{\mathtt{tr}}^t$ according to $E_{\mathtt{sh}}^t$
    Compute each $C_{ijk}^t$ given $\boldsymbol{\theta}^{(n)}, \boldsymbol{u}^{(n)}$
    for $b \leftarrow \mathtt{batch}(E_{\mathtt{sh}}^t, E_{\mathtt{tr}}^t)$ do
        Compute loss $L$ on $b$ according to Eq. (17)
        Compute gradients $\frac{\partial L}{\partial \boldsymbol{\theta}}|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(n)}}$ and $\frac{\partial L}{\partial \boldsymbol{u}}|_{\boldsymbol{u}=\boldsymbol{u}^{(n)}}$
        Update $\boldsymbol{\theta}^{(n)}, \boldsymbol{u}^{(n)}$ according to the gradients
    end
    $\boldsymbol{\theta}^{(n+1)} \leftarrow \boldsymbol{\theta}^{(n)}$, $\boldsymbol{u}^{(n+1)} \leftarrow \boldsymbol{u}^{(n)}$
end
$\boldsymbol{u}_i^t \leftarrow \{\boldsymbol{u}^{(N)}\}_i^t$
```

Algorithm 1: Training process of *DynamicTriad*

Optimization. In order to minimize Eq. (17), we adopt the stochastic gradient decent (SGD) framework with Adagrad method (Duchi, Hazan, and Singer 2011). See details of our training framework in Algorithm 1.

Experimental Results

In this section, we employ three real-world networks to validate the effectiveness of the proposed model, *DynamicTriad*, on six application scenarios.

Data Sets

- *Mobile.* This is a mobile network provided by China Telecom[2]. It consists of more than 2 million call logs between 340,751 users over 15 days. In this data set, users are considered as vertices, and time steps are defined as consecutive non-overlapping one-day periods. In each time step t, if one user has called another, we create an edge between them in G^t, whose weight is determined by the number of calls between the users. Through this definition, we obtain 2,200,203 edges in total. Furthermore, each vertex has a label indicating if the user is a telephone fraudster.
- *Loan.* This network is provided by PPDai[3]. It has a similar structure with Mobile, and consists of 1,603,712 call logs between 200,000 registered users of PPDai over 13 months. We construct the network in the same way as Mobile except that the length of each time step is set to one month. In Loan, each vertex has a time-sensitive label at time t to indicate if the user has ever failed to repay a loan during that time.
- *Academic.* We derive a co-authorship network from the academic network of AMiner[4] (Tang et al. 2008). It consists of 51,060 researchers as vertices and 794,552 coauthor relationships as edges. Each time step is a 4-year period and there is a 2-year interval between the start time of consecutive time steps. At time step t, we create an edge between two users weighted by the number of coauthorships between them during that period. At each time step, we say a researcher belong to a particular community if over half of her papers were published in correponding conferrences[5], and the information is available as labels in this data set.

  Statistics of the above data sets are shown in Table 1.

Table 1: Statistics of each data set.

Data Set	#vertices	#edges	#time steps
Mobile	340,751	2,200,203	15
Loan	200,000	1,603,712	13
Academic	51,060	794,552	16

Tasks and Baselines

On each data set, we first learn embedding vectors $\boldsymbol{u}$ of all vertices at different time steps according to the corresponding dynamic network $\{G^1, \cdots, G^T\}$ by our model. We then apply the embedding vectors on the following six tasks:

- *Vertex classification.* In this task, we aim to determine each vertex's label at a particular time step t by training a classifier. In particular, the classifier takes a vertex's embedding vector $\boldsymbol{u}_i^t$ as its features.
- *Vertex prediction.* Different from vertex classification, this tasks aims to use each vertex's embedding vector $\boldsymbol{u}_i^t$ at current time step t to predict its label at the next time step $t+1$.
- *Link reconstruction.* In this task, we aim to determine if there is an edge between two given vertices v_i and v_j based on the absolute difference in positions between their corresponding embedding vectors, i.e., $|\boldsymbol{u}_i^t - \boldsymbol{u}_j^t|$.
- *Link prediction.* This task is similar to link reconstruction, with the only difference that we aim to predict the existence of an edge in the next time step $t+1$ based on the absolute difference in positions between their embedding vectors in the current time step t.
- *Changed link reconstruction and prediction.* As only a small percentage of links will change over time, we put focus on the changed links and study how they evolve (being added or removed). In particular, these two tasks are similar with link reconstruction and link prediction respectively, with the difference that we only consider newly added or removed links and ignore others in the training and test process.

Baseline methods. We compared the proposed method (*DynamicTriad*) with several state-of-the-art methods using their published codes or our implementation. In each task, we first use different methods to obtain embedding vectors of vertices. Then, we gather all samples from different time steps. Using a Logisitic Regression model as classifier, we repeat 5-fold cross validation on the gathered sample set for 10 times, and compare the average performance. We have conducted our experiments on dimensions 16, 32, 48 and 64, and we have chosen $d = 48$ for presentation due to space limitation. For each graph embedding algorithm mentioned in this section, we perform a grid search on the values of hyper parameters, and we choose a specific combination of them for each task on each data set, which results in the best performance regarding to the F1 score metric.

- *DeepWalk (Perozzi, Al-Rfou, and Skiena 2014).* This is a representative embedding method for static network. We use DeepWalk[6] to learn embedding vectors for with different combination of hyper parameters 'walk length (wl)' and 'window size (ws)'. All combinations of hyper parameters are tested given $wl \in \{20, 40, 60, 80\}$ and $ws \in \{3, 5, 7\}$.
- *node2vec (Grover and Leskovec 2016).* In this method, we use node2vec[7] to learn embedding vectors with different combinations of its parameters p and q, while fixing 'walk length' and 'window size' to the chosen values in DeepWalk experiments. All combinations of hyper parameters are tested given $p \in \{0.5, 1, 1.5, 2, 5\}$ and $q \in \{0.5, 1, 1.5, 2, 5\}$.
- *Temporal Network Embedding (TNE) (Zhu et al. 2016).* This is a dynamic network embedding algorithm based

[2]The major mobile service provider in China

[3]An unsecured micro-credit loan platform in China

[4]http://www.aminer.org, an academic search engine

[5]The representing conferences being considered are ASPLOS, FAST, HPCA, ISCA for computer architecture, SIGMOD, SIGKDD, SIGIR, VLDB and ICDE for data mining, and STOC, FOCS, LICS and CAV for computing theory.

[6]https://github.com/phanein/deepwalk

[7]https://github.com/aditya-grover/node2vec

Table 2: Model performance. C.link reconstruction/prediction is short for changed link reconstruction/prediction.

Data Set	Algorithm	**Vertex classification**			**Link reconstruction**			**C.Link reconstruction**		
		Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall	F1
Academic	DeepWalk	0.541	0.755	0.630	0.682	0.705	0.694	0.747	0.662	0.702
	TNE	0.266	0.554	0.359	0.564	0.584	0.574	0.650	0.576	0.611
	node2vec	0.527	0.768	0.625	0.971	0.976	0.974	0.909	0.889	0.899
	DynamicTriad	**0.618**	**0.818**	**0.704**	**0.983**	**0.988**	**0.985**	**0.921**	**0.928**	**0.925**
Mobile	DeepWalk	0.015	**0.812**	0.030	0.866	0.858	0.862	0.674	0.706	0.689
	TNE	0.006	0.558	0.013	0.504	0.520	0.512	0.502	0.508	0.505
	node2vec	0.016	0.748	0.032	0.986	0.985	0.986	**0.723**	**0.830**	**0.773**
	DynamicTriad	**0.033**	0.553	**0.062**	**0.992**	**0.986**	**0.989**	0.684	0.732	0.707
Loan	DeepWalk	0.112	0.263	0.157	0.829	0.857	0.843	0.683	0.704	0.693
	TNE	0.109	**0.527**	0.181	0.741	0.818	0.777	0.590	0.597	0.594
	node2vec	0.111	0.253	0.154	0.988	0.983	0.986	0.762	**0.894**	0.822
	DynamicTriad	**0.114**	0.501	**0.185**	**0.995**	**0.994**	**0.994**	**0.833**	0.872	**0.852**

Data Set	Algorithm	**Vertex prediction**			**Link prediction**			**C.Link prediction**		
		Prec.	Recall	F1	Prec.	Recall	F1	Prec.	Recall	F1
Academic	DeepWalk	0.516	0.692	0.591	0.596	0.629	0.612	0.750	0.612	0.674
	TNE	0.264	0.540	0.355	0.544	0.551	0.548	0.673	0.570	0.617
	node2vec	0.516	0.705	0.596	0.832	0.701	0.772	0.940	0.843	0.889
	DynamicTriad	**0.584**	**0.790**	**0.671**	**0.888**	**0.790**	**0.836**	**0.965**	**0.885**	**0.924**
Mobile	DeepWalk	0.012	**0.694**	0.024	0.661	0.657	0.659	0.689	0.656	0.671
	TNE	0.006	0.554	0.012	0.504	0.509	0.507	0.493	0.504	0.498
	node2vec	0.014	0.655	0.027	0.797	0.673	0.730	**0.799**	**0.676**	**0.732**
	DynamicTriad	**0.033**	0.547	**0.062**	**0.968**	**0.915**	**0.940**	0.728	0.657	0.690
Loan	DeepWalk	0.112	0.274	0.159	0.670	0.736	0.702	0.650	0.708	0.677
	TNE	0.108	**0.542**	0.180	0.635	0.743	0.685	0.672	0.557	0.609
	node2vec	0.112	0.258	0.156	0.771	0.730	0.750	0.870	0.655	0.747
	DynamicTriad	**0.113**	0.500	**0.185**	**0.938**	**0.858**	**0.896**	**0.911**	**0.791**	**0.847**

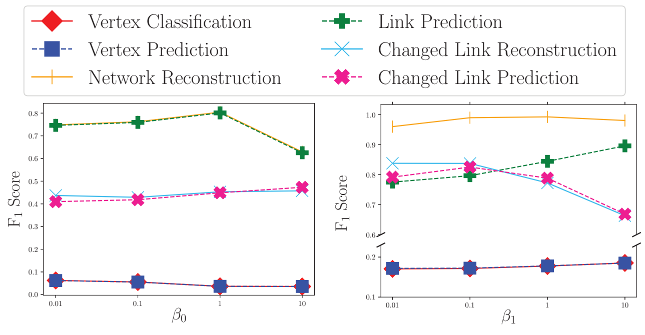
on matrix factorization[8]. We try different values of TNE's parameter λ (option "-l" in the implementation) and report the best performance. All combinations of hyper parameters are tested given $\lambda \in \{0.001, 0.01, 0.1, 1, 10\}$.

- *DynamicTriad*. This is the proposed model. We tested all combinations of parameters β_0 and β_1 given $\beta_0 \in \{0.01, 0.1, 1, 10\}$ and $\beta_1 \in \{0.01, 0.1, 1, 10\}$.

Quantitative Results

Comparison result. Table 2 demonstrates the comparison results of different methods. Overall, we see that *DynamicTriad* outperforms other methods (e.g., +25.1% in terms of F1 averagely) in most cases, especially in link prediction, which suggests the effectiveness of our method to preserve both structural and temporal information of dynamic networks. The exception happens in the changed link reconstruction and changed link prediction task on Mobile, where node2vec outperforms *DynamicTriad*. One potential reason is, in the Mobile data sets, there are quite a few calls made by fraudsters or couriers, whose contacts change a lot over time. Thus it brings much noise to *DynamicTriad* when modeling the changes of edges connected with fraudsters or couriers. However, the good performance of *DynamicTriad* in vertex classification and prediction suggests that the network evolution patterns provide effective information to make the vertex representations more discriminative.

[8]https://github.com/linhongseba/Temporal-Network-Embedding



(a) Mobile, $\beta_1 = 10$.

(b) Loan, $\beta_0 = 0.1$.

Figure 2: Hyper parameter analysis.

We also notice that most of the results in prediction tasks are slightly worse than those in corresponding classification tasks. This is caused by the information loss between consecutive time steps, which is the core problem addressed by dynamic network embedding methods. Comparing with other methods, *DynamicTriad* performs more stable.

Meanwhile, TNE, as a dynamic network embedding

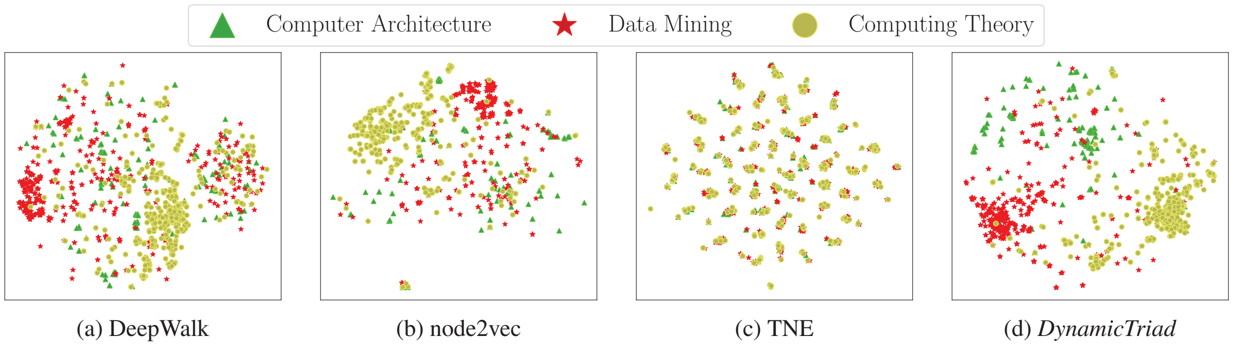


Figure 3: 2D t-SNE projections of embeddings of 1040 researchers, who are in the largest component of the labelled subgraph, from the Academic data set. The labels are taken from the 6th time step while the embedding vectors are from the 5th.

method, is incompetent in several experiments even comparing with static network embedding methods (i.e., DeepWalk and node2vec). One potential reason could be that the matrix factorization scheme adopted in TNE fails to balance positive and negative samples given a sparse adjacency matrix.

Parameter analysis Our model has two hyper parameters β_0 and β_1, where β_0 is the weight of triad closure process in Eq. (9), and β_1 denotes the weight of temporal smoothness. We study the sensitivity of each parameter by fixing the other one. From Figure 2, we see that, though not very sensitive, β_0 and β_1 improve the performance of *DynamicTriad* in most tasks if a proper value is chosen, and it otherwise will damage the performance.

Qualitative Results

We demonstrate the embedding vectors of several researchers in the Academic data set learned by different methods. We project the embedding vectors to a 2-dimensional space with the t-SNE method (Maaten and Hinton 2008) and compare the results of researchers from three different research communities: computer architecture, computing theory, and data mining. As Figure 3 shows, *DynamicTriad* (Figure 3d) can clearly separate researchers from different communities comparing with other methods. More specifically, TNE (Figure 3c) fails to distinguish three communities from each other totally. DeepWalk and node2vec are able to identify the computing theory community. However, they confuse the computer architecture community and the data mining community. By a careful investigation, we find that researchers from these two communities have similar collaboration patterns. However, the computer architecture community was built much earlier than the data mining community, while the number of data mining papers has a significant growth recently. Thus our method, which considers network evolution, can better capture the difference between these communities.

Related Work

Recent years, learning representations for networks has attracted considerable research effort. For example, conventional dimensionality reduction methods (Belkin and Niyogi 2001; Roweis and Saul 2000; Jolliffe 2002; Tenenbaum, De Silva, and Langford 2000; Kruskal 1964) have been proposed to learn low dimensional representations of graphs by employing their spectral properties. Some recent researches on graph embedding are inspired by the advancements in natural language processing, in particular the adoption of skip-gram models in representation learning of words (Mikolov et al. 2013). Analogies of sentences are defined in graphs by paths sampled by various strategies (Perozzi, Al-Rfou, and Skiena 2014; Grover and Leskovec 2016; Tang et al. 2015; Dong, Chawla, and Swami 2017).

Most of the aforementioned researches focuses on static networks, where the network itself as well as the learned representations are fixed. However, in reality, networks evolve over time. There is only few research on dynamic network embedding, which mainly consider temporal smoothness (Sarkar and Moore 2005; Zhu et al. 2016). Meanwhile, there are many studies on identifying fundamental factors that cause network dynamics (Kossinets and Watts 2006; Tantipathananandh, Berger-Wolf, and Kempe 2007; Sun et al. 2007; Zhuang et al. 2013; Zhang et al. 2016). Our idea is employing one of these factors, the triad closure process (Coleman 1994), as a guidance to learn dynamic representations of networks.

Conclusion

In this paper, we present a novel representation learning algorithm for dynamic networks, and a semi-supervised algorithm to learn dynamic representations of vertices. To validate the effectiveness of both our model and learning algorithm, we construct experiments on three real-world networks. Experimental results demonstrate that compared to several state-of-the-art techniques, our approach achieves substantial gains in six applications.

Acknowledgements

This work is supported by the Fundamental Research Funds for the Central Universities, 973 Program (2015CB352302), NSFC (U1611461, 61625107, 61402403), Key Program of Zhejiang Province (2015C01027), Key R&D Program of Zhejiang Province (2018C01006) and China Knowledge Center for Engineering.

References

Belkin, M., and Niyogi, P. 2001. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, volume 14, 585–591.

Coleman, J. S. 1994. Foundations of social theory. *American Political Science Review* 85(1):263.

Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B* 39(1):1–38.

Dong, Y.; Chawla, N. V.; and Swami, A. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 135–144. ACM.

Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12(Jul):2121–2159.

Erds, D.; Gemulla, R.; and Terzi, E. 2014. Reconstructing graphs from neighborhood data. *ACM Transactions on Knowledge Discovery From Data* 8(4):23.

Grover, A., and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *KDD'16*, 855–864.

Ho, Q.; Yin, J.; and Xing, E. P. 2016. Latent space inference of internet-scale networks. *Journal of Machine Learning Research* 17(78):1–41.

Hoff, P. D.; Raftery, A. E.; and Handcock, M. S. 2012. Latent space approaches to social network analysis. *Journal of the American Statistical Association* 97(460):1090–1098.

Huang, H.; Tang, J.; Liu, L.; Luo, J.; and Fu, X. 2015. Triadic closure pattern analysis and prediction in social networks. *IEEE Transactions on Knowledge and Data Engineering* 27(12):3374–3389.

Jolliffe, I. 2002. *Principal component analysis*. Wiley Online Library.

Kossinets, G., and Watts, D. J. 2006. Empirical analysis of an evolving social network. *science* 311(5757):88–90.

Kruskal, J. B. 1964. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* 29(1):1–27.

Maaten, L. v. d., and Hinton, G. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research* 9(Nov):2579–2605.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, 3111–3119.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, 701–710. New York, NY, USA: ACM.

Qi, G.; Aggarwal, C. C.; and Huang, T. S. 2013. Link prediction across networks by biased cross-network sampling. In *ECML/PKDD'13*, 793–804.

Roweis, S. T., and Saul, L. K. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290(5500):2323–2326.

Sarkar, P., and Moore, A. W. 2005. Dynamic social network analysis using latent space models. *ACM SIGKDD Explorations Newsletter* 7(2):31–40.

Sun, J.; Faloutsos, C.; Papadimitriou, S.; and Yu, P. S. 2007. Graphscope: parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 687–696. ACM.

Tang, J.; Zhang, J.; Yao, L.; Li, J.; Zhang, L.; and Su, Z. 2008. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 990–998. ACM.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, 1067–1077. ACM.

Tantipathananandh, C.; Berger-Wolf, T.; and Kempe, D. 2007. A framework for community identification in dynamic social networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 717–726. ACM.

Tenenbaum, J. B.; De Silva, V.; and Langford, J. C. 2000. A global geometric framework for nonlinear dimensionality reduction. *science* 290(5500):2319–2323.

Wang, Z.; Zhang, J.; Feng, J.; and Chen, Z. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, 1112–1119. Citeseer.

Wang, D.; Cui, P.; and Zhu, W. 2016. Structural deep network embedding. In *KDD'16*, 1225–1234.

Zhang, T.; Cui, P.; Faloutsos, C.; Lu, Y.; Ye, H.; Zhu, W.; and Yang, S. 2016. Come-and-go patterns of group evolution: A dynamic model. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1355–1364. ACM.

Zhu, L.; Guo, D.; Yin, J.; Steeg, G. V.; and Galstyan, A. 2014. Scalable link prediction in dynamic networks via non-negative matrix factorization. *arXiv preprint arXiv:1411.3675*.

Zhu, L.; Guo, D.; Yin, J.; Ver Steeg, G.; and Galstyan, A. 2016. Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE Transactions on Knowledge and Data Engineering* 28(10):2765–2777.

Zhuang, H.; Sun, Y.; Tang, J.; Zhang, J.; and Sun, X. 2013. Influence maximization in dynamic social networks. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, 1313–1318. IEEE.