Temporal Network Embedding with High-Order Nonlinear Information

Zhenyu Qiu,[1,2] **Wenbin Hu,**[1,4*] **Jia Wu,**[3] **Weiwei Liu,**[1†] **Bo Du,**[1] **Xiaohua Jia**[2]
[1]School of Computer Science, Wuhan Univiesity
[2]Department of Computer Science, City University of Hong Kong
[3]Department of Computing, Macquarie University
[4]Shenzhen Research Institute, Wuhan University, China
{qiuzy, hwb, remoteking}@whu.edu.cn, jia.wu@mq.edu.au, liuweiwei863@gmail.com, csjia@cityu.edu.hk

Abstract

Temporal network embedding, which aims to learn the low-dimensional representations of nodes in temporal networks that can capture and preserve the network structure and evolution pattern, has attracted much attention from the scientific community. However, existing methods suffer from two main disadvantages: 1) they cannot preserve the node temporal proximity that capture important properties of the network structure; and 2) they cannot represent the nonlinear structure of temporal networks. In this paper, we propose a high-order nonlinear information preserving (HNIP) embedding method to address these issues. Specifically, we define three orders of temporal proximities by exploring network historical information with a time exponential decay model to quantify the temporal proximity between nodes. Then, we propose a novel deep guided auto-encoder to capture the highly nonlinear structure. Meanwhile, the training set of the guide auto-encoder is generated by the temporal random walk (TRW) algorithm. By training the proposed deep guided auto-encoder with a specific mini-batch stochastic gradient descent algorithm, HNIP can efficiently preserves the temporal proximities and highly nonlinear structure of temporal networks. Experimental results on four real-world networks demonstrate the effectiveness of the proposed method.

Introduction

Network embedding has attracted increasing attention in recent years. The basic idea is to embed a network into a low-dimensional vector space where the inherent structural properties of the network are preserved so that the network analysis and prediction tasks can be conducted in the vector space (Zhu et al. 2018). Recent work on network embedding methods have been demonstrated to be effective in a variety of applications such as link prediction (Yu et al. 2017), classification (Sun, Yuan, and Wang 2018), and clustering (Xie et al. 2019; Peng et al. 2019). Generally, most of the existing methods for network embedding focus on static networks (Zhang, Lyu, and Zhang 2018; Wang et al. 2017; Perozzi, Al-Rfou, and Skiena 2014). However, interactions in real-world networks are dynamic in nature (e.g., Twitter, Facebook), where the edges evolve over time (Falzon et al. 2018). Static methods that ignore the temporal information of a given network can hardly capture the actual network properties (Zhu et al. 2018). Typically, a static network is constructed by aggregating interactions between nodes over a specific period of time while ignoring the time dimensional information (Falzon et al. 2018). The ignore of the temporal interactions will make network embedding failing to preserve temporal network structures.

In view of this, several temporal network embedding methods have been proposed. For example, Zou et al. (Zuo et al. 2018) adopted the Hawkes process to model the neighborhood formation process. Zhou et al. (Zhou et al. 2018) proposed a more fundamental mechanism, triadic closure process, to model the formation and evolution of networks. However, both methods focus on the modeling of network evolution pattern, but limit the scope of structural properties to low-order proximities. Zhu et al. (Zhu et al. 2018) proposed DHPE (Zhu et al. 2018) to preserve the high-order proximity, but it does so by applying a static model to each snapshot and thus fails to capture temporal evolution. Moreover, the underlying structure of temporal network is highly nonlinear (Luo et al. 2011), existing temporal network embedding methods are not able to preserve the nonlinear structure.

There are three challenges to learn temporal network representations:

- **Temporal proximity definition** (Challenge 1): Node proximity is always the first and foremost property of a network (Tang et al. 2015). Most temporal embedding methods simply preserve the traditional static proximity (Zhu et al. 2018) while ignoring the historical information. In fact, temporal network structures could provide more semantic meanings than static networks for node proximity quantification. For example, a static network simply counts the number of interactions between two nodes as the first-order proximity between them. While, it is generally accepted that the impact of an interaction decreases with time, and an interaction is more important if it happens in a closer proximity of time (Rümmele, Ichise, and Werthner 2015). Thus, it is the first challenge

*Corresponding author: Wenbin Hu
†Co-corresponding author: Weiwei Liu

in temporal network embedding to define the node temporal proximity based on the temporal interaction information to capture the accurate relationship between nodes.

- **Temporal proximity preserving** (Challenge 2): Although a number of static methods for node proximity preserving have been proposed (Grover and Leskovec 2016), the temporal node proximity preserving is still an open problem (especially for the high-order temporal proximity) due to the heavy computational cost.
- **Nonlinear structure preserving** (Challenge 3):Temporal network structure is highly nonlinear. It is a great challenge to preserve the highly nonlinear structure of temporal networks in temporal network embedding.

In this paper, we propose a high-order nonlinear information preserving (HNIP) embedding method to tackle the aforementioned challenges. We first define three orders of temporal proximities based on a time exponential decay model, namely, the first-, second-, and high-order temporal proximity. (Challenge #1). Then, we propose a temporal random walk (TRW) algorithm to preserve the temporal proximity in the walk path (Challenge #2). Finally, we propose a novel deep guided auto-encoder to learn node representations for networks. This auto-encoder has multiple layers of nonlinear functions to capture the highly nonlinear network structure (Challenge #3). In addition, the training process is guided by the temporal proximity preserved in random walk paths generated by TRW. By training the proposed auto-encoder with the walk path set generated by TRW, our method can preserve the temporal proximity and highly nonlinear structure of temporal networks.

The contributions of this paper can be summarized as the following:

- We define three orders of temporal proximities for accurate node temporal proximity quantification.
- We propose a TRW algorithm and a deep guided auto-encoder for temporal proximity and highly nonlinear structure preserving for temporal networks.
- We evaluate the effectiveness and efficiency of HNIP on a variety of applications of real-world networks.

Related Work

Network embedding methods for static networks have been widely studied in the past few years. Various methods such as random walk based method (DeepWalk (Perozzi, Al-Rfou, and Skiena 2014), Node2vec (Grover and Leskovec 2016)), factorization based method (M-NMF (Wang et al. 2017)), and deep learning method (SDNE (Wang, Cui, and Zhu 2016)) have demonstrated their effectiveness in preserving the structural properties of static networks.

Real-world networks are dynamic in nature. To capture the dynamic properties of real-world networks, temporal network embedding has drawn increasing interests. HTNE (Zuo et al. 2018) adopt the Hawkes process to model the neighborhood formation process for temporal network embedding. DynamicTriad (Zhou et al. 2018) imposes triad to model dynamic changes of network structure. CTDNE (Nguyen et al. 2018) defines a valid walk sequence to capture the interaction sequence of temporal network. NetWalk (Yu et al. 2018) learns representations based on deep neural network embedding and reservoir sampling. Despite the success of these methods, most of them can not tackle the three challenges presented in the introduction. A method that can meet these challenges is still lacking.

Problem Definition

In this section, we first provide the formal definition of the temporal network and other necessary definitions used in this paper. Then, we formally define the problem of temporal network embedding.

We consider a temporal network where links evolve over time instead of being static. By treating the added/deleted nodes as isolated nodes, all the changes in the network can be regarded as changes in the links (Chen and Tong 2015). So we consider the number of nodes as constant. The definition of the temporal network is as follows:

Definition 1 (Temporal Network). *A temporal network is defined as $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ is the node set and $E = \{e_k\}$ is the temporal link set. A temporal link $e_k = (v_i, v_j, w_k, t_k)$ is a quadruple, which represents that v_i and v_j interact with each other at time t_k, and w_k is the link weight. The value of w_k represents the strength of the interaction at t_k. There may be multiple links with different timestamp between two nodes. By assigning timestamp to each link, a temporal network can record the evolution of relationships among nodes.*

For each node pair (v_i, v_j), we can obtain a finite link set $\mathbf{a}_{i,j} = \{e_k\}$ that consists of all the recorded timed links with a specific weight between them in the designated time period of interest. Similar to the static adjacency matrix, we represent the temporal network by a time-stamped adjacency matrix.

Definition 2 (Time-stamped Adjacency Matrix). *A temporal network $G = (V, E)$ can be represented by a time-stamped adjacency matrix $\mathbf{A}$, in which each entry $\mathbf{a}_{i,j} = \{e_k\}$ is a set of links between nodes v_i and v_j. If there are no interactions between nodes v_i and v_j, we define $\mathbf{a}_{i,j} = \emptyset$.*

In this paper, we focus on the problem of temporal network embedding and aim to overcome the three challenges presented in introduction. The formal problem definition is as follows:

Definition 3 (Temporal Network Embedding). *Given a temporal network $G = (V, E)$ and the corresponding time-stamped adjacency matrix $\mathbf{A}$, temporal network embedding aims to learn a mapping function $f : v_i \to \mathbf{y}_i \in \mathbb{R}^d$. The objective of the function is to make the similarity between $\mathbf{y}_i$ and $\mathbf{y}_j$ preserve the temporal proximity between v_i and v_j.*

Node Temporal Proximity

The node proximity has been intensively studied in network embedding (Goyal and Ferrara 2018). However, traditional node proximity without considering time information can hardly capture the rich semantic meanings of temporal networks. By exploring the historical interaction between nodes, we define the first-, second-, and high-order temporal proximity in this section to quantify the temporal proximity between nodes.

First-order similarity is the most fundamental property of the network, which is quantified by link weight in static network embedding (Tang et al. 2015). In temporal network, we define the first-order temporal proximity based on two observations: 1) the influence of a interaction decreases with time and the tie between two nodes becomes stronger if the interaction between them happens in a more current time (Munasinghe and Ichise 2012); and 2) the tie between two nodes increases with the increase of interaction frequency (Rümmele, Ichise, and Werthner 2015). Formally, the first-order proximity is defined as follows:

Definition 4 (FIRST-ORDER TEMPORAL PROXIMITY). *Given a link set $\mathbf{a}_{i,j}$ between nodes v_i and v_j, the first-order temporal proximity $\mathbf{p}_{i,j}$ between nodes v_i and v_j is defined as:*

$$\mathbf{p}_{i,j} = \sum_{e_k \in \mathbf{a}_{i,j}} w_k \mathrm{exp}(-\lambda(T - t_k)), \tag{1}$$

where λ is the decay constant, and T is the current time.

In Equation (1), we adopt an exponential decay function to simulate the influence of interaction decay with time. For a given temporal network G, the first-order temporal proximity between nodes in G is represented by matrix $\mathbf{P} \in \mathbb{R}^{n \times n}$.

Since the real-world networks are usually sparse, many nodes are similar with each other but are not linked (Wang, Cui, and Zhu 2016). Following the second-order proximity in static network embedding (Wang, Cui, and Zhu 2016), we define the second-order temporal proximity based on $\mathbf{P}$.

Definition 5 (SECOND-ORDER TEMPORAL PROXIMITY). *Given $\mathbf{P}$, let $\mathbf{p}_i = \{\mathbf{p}_{i,1}, ...\mathbf{p}_{i,n}\}$ denote the i-th row of $\mathbf{P}$, which represents the first-order temporal proximity between node v_i and other nodes.The second-order temporal proximity between nodes v_i and v_j is defined to be the similarity between $\mathbf{p}_i$ and $\mathbf{p}_j$.*

Intuitively, the second-order temporal proximity between a pair of nodes describes the proximity of the pair's neighborhood structure. The definition of the second-order temporal proximity is based on a proven assumption that two nodes are similar if they share many common neighbors (Wang, Cui, and Zhu 2016).

The defined first- and second-order temporal proximity can well preserve the local temporal network property. To capture the network global structural property, we define the high-order temporal proximity.

Inspired by (Cao, Lu, and Xu 2016), we adopt the s-hop ($s = 1, 2, 3, ...$) transition probability matrix to describe the high-order temporal proximity. Assume $\mathbf{p}_{i,j}$ is proportional to the transition probability from v_i to v_j, we can define the one-hop transition probability matrix $\mathbf{M} = \mathbf{D}^{-1}\mathbf{P}$, where $\mathbf{D}$ is a diagonal matrix, and $\mathbf{d}_{i,i} = \sum_{j=1}^{n} \mathbf{p}_{i,j}$. Then the s-hop transition probability matrix is

$$\mathbf{M}^s = \underbrace{\mathbf{M} \cdots \mathbf{M}}_{s}. \tag{2}$$

The value of $\mathbf{m}_{i,j}^s$ represents the probability of node v_i reaching v_j after s-hop random walk. It is reasonable to assume that a higher value of $\mathbf{m}_{i,j}^s$ indicates a closer relationship between v_i and v_j. By integrating different hop transition probability metrics, we define the high-order temporal proximity as follows.

Definition 6 (HIGH-ORDER TEMPORAL PROXIMITY). *Given $\mathbf{M}^1, \mathbf{M}^2, ..., \mathbf{M}^S$, the high-order temporal proximity $\mathbf{h}_{i,j}$ between nodes v_i and v_j is defined as:*

$$\mathbf{h}_{i,j} = \sum_{s=1}^{S} \alpha_s \mathbf{m}_{i,j}^s, \tag{3}$$

where $\alpha_1 > \alpha_2 > ... > \alpha_S$.

The high-order temporal proximity describes the k-hop ($k > 2$) relationships between nodes. For a given temporal network G, the high-order temporal proximity between nodes in G is represented by matrix $\mathbf{H} \in \mathbb{R}^{n \times n}$.

High-Order Nonlinear Information Preserving Embedding

In order to capture the highly nonlinear structure of temporal networks, we propose a deep guided auto-encoder based on the traditional auto-encoder (Peng et al. 2018), which has demonstrated a strong ability to learn the complex structure of data (Bengio 2009). Unlike SDNE (Wang, Cui, and Zhu 2016) that uses an adjacency matrix to train a deep architecture for first- and second-order proximity preservation, we construct the training set by the TRW algorithm. Along with the TRW algorithm, the proposed deep guided auto-encoder can successfully preserve the temporal proximity and nonlinear structure.

Temporal Random Walk (TRW)

The random walk technique has been widely used in network embedding because the path it generated preserves the high-order relationship between nodes (Perozzi, Al-Rfou, and Skiena 2014; Grover and Leskovec 2016). Since there may be multiple links between two nodes in a temporal network, the traditional random walk algorithm is not applicable. In order to tackle the multiple links problem and preserve the high-order temporal proximity in the walk path, we propose the TRW algorithm in this section.

TRW contains a link sample step to tackle the multiple link problem. Given the current node v_i linked by multiple links to other nodes, we can get the link set $\Gamma_i = \bigcup_{j=1}^{n} \mathbf{a}_{i,j}$ associate with v_i. TRW first samples a link randomly from Γ_i. Let e_j be the selected link that links v_i and v_j, Then, TRW moves the walker from v_i to v_j to complete one-hop walk. The sample probability of e_j is

$$p(e_j) = \frac{w_j \mathrm{exp}(-\lambda(T - t_j))}{\sum_{e_x \in \Gamma_i} w_x \mathrm{exp}(-\lambda(T - t_x))}. \tag{4}$$

In addition, to make sure the walk path L_i that starts from v_i preserves the high-order temporal proximity between v_i and other nodes, we should to make the appearance probability of v_j in L_i equal to $\mathbf{h}_{i,j}$. For this purpose, we introduce a restart step in TRW: in each step of a random walk,

Algorithm 1 Temporal Random Walk

```
Input: network $G$, $\mathbf{A}$, walk length $S$, walk times $wt$, and
    restart ratio $r$.
Output: walk set $\Omega$.
 1: Initialize walk set $\Omega = \emptyset$;
 2: for each node $v_i$ in $G$ do
 3:   for $k = (i-1) \times wt + 1$ to $i \times wt$ do
 4:     initialize a walk path $L_k = \{v_i\}$, the walker start
        from $v_i$;
 5:     for $z = 1$ to $S - 1$ do
 6:       if $random(0,1) < r$ then
 7:         let the walker move to $v_i$, and add $v_i$ to $L_k$;
 8:       else
 9:         let $v_x$ be the current node, randomly select a
            link $e_j$ from $\Gamma_x$, assume $e_j$ associate with $v_x$
            and $v_j$;
10:         let the walker move to $v_j$, and add $v_j$ to $L_k$;
11:       end if
12:     end for
13:     add $L_k$ into $\Omega$
14:   end for
15: end for
16: return $\Omega$.
```

the walker can return to the starting node with a probability r. Finally, the TRW algorithm is presented in Algorithm 1.

To show that the walk path generated by TRW can preserve the high-order temporal proximity defined in this paper, we prove the following lemmas.

Lemma 1 *The one-hop transition probability $p(v_i \rightarrow v_j)$ from v_i to v_j in TRW is $\mathbf{m}_{i,j}$.*

Proof 1 *The link set between v_i and v_j is $\mathbf{a}_{i,j}$, select a link e_x from Γ_i, and the probability that e_z belongs to $\mathbf{a}_{i,j}$ is*

$$p(e_z \in \mathbf{a}_{i,j}) = \frac{\sum_{e_y \in \mathbf{a}_{i,j}} w_y \exp(-\lambda(T - t_y))}{\sum_{e_x \in \Gamma_i} w_x \exp(-\lambda(T - t_x))}. \quad (5)$$

Thus, the one-hop transition probability $p(v_i \rightarrow v_j)_1$ is equal to $p(e_z \in \mathbf{a}_{i,j})$. Then, according to the definition of $\mathbf{p}_{i,j}$ and $\mathbf{m}_{i,j}$, we have

$$p(v_i \rightarrow v_j)_1 = \frac{\mathbf{p}_{i,j}}{\sum_{j=1}^{n} \mathbf{p}_{i,j}} = \mathbf{m}_{i,j}. \quad (6)$$

Lemma 2 *Let α_s in Equation (3) be $\frac{(1+r(S-s))(1-r)^s}{S}$, L_i be the walk path starting from v_i, and the length of L_i be S, the appearance probability $p(v_i \rightarrow v_j)_S$ of v_j in L_i is equal to $\mathbf{h}_{i,j}$.*

Proof 2 *According to Lemma 1, the one-hop transition probability matrix in TRW without a restart step is $\mathbf{M}$; by adding the restart step, the one-hop transition probability matrix is*

$$\mathbf{R} = r\mathbf{I} + (1-r)\mathbf{M}. \quad (7)$$

and the s-hop transition probability matrix with restart is

$$\mathbf{R}^s = r\mathbf{I} + (1-r)\mathbf{R}^{s-1}\mathbf{M}, \quad (8)$$

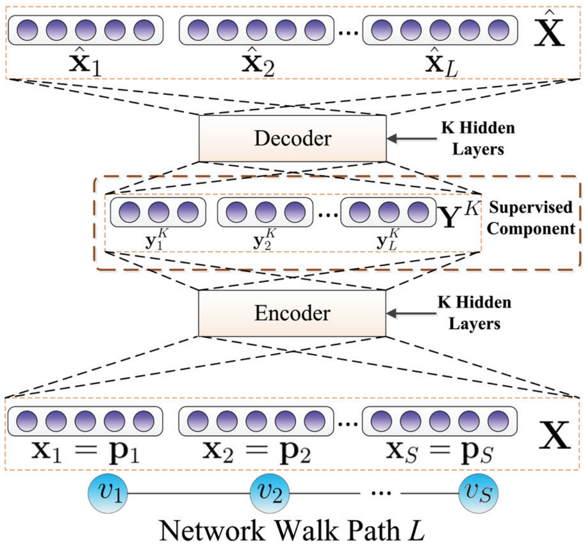

Figure 1: The architecture of the deep guided auto-encoder.

where $\mathbf{P}^0 = \mathbf{I}$. Then,

$$\mathbf{R}^s = (1-r)^s \mathbf{M}^s + \sum_{i=1}^{s} r(1-r)^{(s-i)} \mathbf{A}^{s-i}. \quad (9)$$

Therefore,

$$p(v_i \rightarrow v_j)_S = \frac{1}{S} \sum_{s=1}^{S} \mathbf{r}_{i,j}^s = \sum_{s=1}^{S} \frac{(1 + r(S-s))(1-r)^s}{S} \mathbf{m}_{i,j}^s = \mathbf{h}_{i,j}. \quad (10)$$

Deep Guided Auto-Encoder

The architecture of the proposed deep guided auto-encoder is presented in Fig. 1. First, in order to capture the highly nonlinear temporal network structure, this deep model utilizes a deep auto-encoder to learn the vector representation of nodes. As illustrated in Fig. 1, by inputting the corresponding vector $\mathbf{p}_i$ of each node v_i to the deep auto-encoder and minimizing the reconstruction error, we can preserve the second-order temporal proximity of each node in the embedding space. In addition, a walk path L_i generated by TRW has two characteristics: 1) in most cases, there exists a link between two adjacent nodes in L_i, and the probability that v_x is followed by v_y is proportional to $\mathbf{p}_{x,y}$; 2) let v_i be the first node and v_x be any other node in L_i; there must be a path between v_i and v_x, and the appearance probability of v_x is equal to $\mathbf{h}_{i,x}$. Let $\mathbf{X}_i \in \mathbb{R}^{S \times n}$ be the matrix corresponding to L_i, where the j-th row $\mathbf{x}_{i_j}$ is the corresponding vector $\mathbf{p}_j$ of the j-th node in L_i, and S is the length of L_i. As illustrated in Fig. 1, we take the $\mathbf{X}_i$ as input and design a supervised component to guide the training process to preserve the first- and high-order temporal proximity.

Loss Functions. In order to preserve the first-, second-, and high-order temporal proximity, we formulate three loss functions respectively, and then combine them to get the objective function. The main notations used in this section are detailed in Table 1.

We first explain how to utilize the auto-encoder to preserve the second-order temporal proximity in our model. Just like a traditional deep auto-encoder, each layer of the auto-encoder used in our deep model is a fully connected layer, and a sigmoid function $\sigma(x) = \frac{1}{1+\exp(x)}$ is adopted as the activation function of the hidden layer. Therefore, the hidden representations of each hidden layer are as follows:

$$\begin{aligned} \mathbf{Y}_i^k &= \sigma(\mathbf{W}_k \mathbf{Y}_i^{k-1} + \mathbf{b}_k), k = 1, ..., K \\ \hat{\mathbf{Y}}_i^k &= \sigma(\hat{\mathbf{W}}_k \hat{\mathbf{Y}}_i^{k-1} + \hat{\mathbf{b}}_k), k = 1, ..., K\text{'} \end{aligned} \tag{11}$$

where $\mathbf{Y}_i^0 = \mathbf{X}_i$, $\hat{\mathbf{Y}}_i^K = \hat{\mathbf{X}}_i$. Our goal is to minimize the reconstruction error of the output and the input. Then, the loss function is as follows:

$$Ł_2 = \sum_{i=1}^{|\Omega|} \sum_{j=1}^{S} ||\mathbf{x}_{i_j} - \hat{\mathbf{x}}_{i_j}||_2^2. \tag{12}$$

Since $\mathbf{x}_{i_j} = \mathbf{p}_j$, and $\mathbf{p}_j$ characterizes the temporal neighborhood structure of node v_j, the reconstruction process will make the nodes with similar neighborhood structures to have the similar representations (Wang et al. 2017). Therefore, the second-order temporal proximity can be preserved by minimizing $Ł_2$.

In addition, to address the sparsity problem of real-world networks, we impose more penalty to the reconstruction error of the non-zero elements than to that of zero elements.

Table 1: Notation Description

Notations	Description
n	number of nodes
$2K$	number of hidden layers
Ω	random walk set
L_i	the i-th wall path in Ω
S	the lenght of walk path
$\mathbf{X}_i = \{\mathbf{x}_{i_j}\}_{j=1}^S$	the input matrix corresponding to L_i
$\hat{\mathbf{X}}_i = \{\hat{\mathbf{x}}_{i_j}\}_{j=1}^S$	the reconstructed matrix of $\mathbf{X}_i$
$\mathbf{Y}_i^k = \{\mathbf{y}_{i_j}^k\}_{j=1}^S$	the k-th encoder layer representations of $\mathbf{X}_i$
$\hat{\mathbf{Y}}_i^k = \{\hat{\mathbf{y}}_{i_j}\}_{j=1}^S$	the k-th decoder layer representations of $\mathbf{X}_i$
$\mathbf{W}_k$	the k-th layer weight matrix of the encoder
$\hat{\mathbf{W}}_k$	the k-th layer weight matrix of the decoder
$\mathbf{b}_k$	the k-th layer biases of the encoder
$\hat{\mathbf{b}}_k$	k-th layer biases of the decoder

Then, $Ł_2$ is revised as follows:

$$\begin{aligned} Ł_2 &= \sum_{i=1}^{|\Omega|} \sum_{j=1}^{S} ||(\mathbf{x}_{i_j} - \hat{\mathbf{x}}_{i_j}) \odot \mathbf{z}_{i_j}||_2^2 \\ &= \sum_{i=1}^{|\Omega|} ||(\mathbf{X}_i - \hat{\mathbf{X}}_i) \odot \mathbf{Z}_i||_2^2, \end{aligned} \tag{13}$$

where $\odot$ is the Hadamard product, $\mathbf{Z}_i \in \mathbb{R}^{S \times n}$. If $\mathbf{x}_{k_{i,j}} = 0$, $\mathbf{z}_{k_{i,j}} = 1$, else $\mathbf{z}_{k_{i,j}} = \beta > 1$.

Because there exists a link between two adjacent nodes in L_i, and the probability that v_x is followed by v_y is proportional to $\mathbf{p}_{x,y}$, we also need to minimize the pairwise distance among all adjacency nodes of each network walk path in the embedding space to preserve the first-order temporal proximity, which can be formally described as follows:

$$\begin{aligned} Ł_1 &= \sum_{i=1}^{|\Omega|} \sum_{j=1}^{S-1} ||\mathbf{y}_{i_j}^K - \mathbf{y}_{i_{j+1}}^K||_2^2 \\ &= \sum_{i=1}^{|\Omega|} tr(\mathbf{Y}_i^{K^T} \mathbf{L}_i^1 \mathbf{Y}_i^K), \end{aligned} \tag{14}$$

where $\mathbf{L}_i^1 = \mathbf{D}_i^1 - \mathbf{O}_i^1$ is the Laplacian matrix. $\mathbf{O}_i^1 \in \mathbb{R}^{S \times S}$, and $\mathbf{o}_{i_{j,j+1}}^1 = 1, j = 1, ..., S-1$, and the other elements of $\mathbf{O}_i^1$ is 0. $\mathbf{D}_i^1$ is a diagonal matrix, and $\mathbf{d}_{i_{j,j}}^1 = \sum_{k=1}^S \mathbf{o}_{i_{j,k}}^1$.

Because the first node in L_i has at least one path linked to other nodes, we need to minimize the pairwise distance between the first node in L_i and other nodes in the embedding space to preserve the high-order temporal proximity, which can be formally described as follows:

$$\begin{aligned} Ł_h &= \sum_{i=1}^{|\Omega|} \sum_{j=2}^{S} ||\mathbf{y}_{i_1}^K - \mathbf{y}_{i_j}^K||_2^2 \\ &= \sum_{i=1}^{|\Omega|} tr(\mathbf{Y}_i^{K^T} \mathbf{L}_i^2 \mathbf{Y}_i^K), \end{aligned} \tag{15}$$

where $\mathbf{L}_i^2 = \mathbf{D}_i^2 - \mathbf{O}_i^2$ is the Laplacian matrix. $\mathbf{O}_i^2 \in \mathbb{R}^{S \times S}$, and $\mathbf{o}_{i_{1,j}}^2 = 1, j = 2, ..., S$, and the other elements of $\mathbf{O}_i^2$ is 0. $\mathbf{D}_i^2$ is a diagonal matrix, and $\mathbf{d}_{i_{j,j}}^2 = \sum_{k=1}^S \mathbf{o}_{i_{j,k}}^2$.

Finally, to preserve the first-, second-, and high-order temporal proximities simultaneously, we combine $Ł_1$, $Ł_2$, and $Ł_h$ and jointly minimize the following objective function:

$$Ł = Ł_1 + \mu Ł_2 + \nu Ł_h + \varphi \frac{1}{2} \sum_{k=1}^{K} (||\mathbf{W}_k||_F^2 + ||\hat{\mathbf{W}}_k||_F^2), \tag{16}$$

where the last term in Ł is an Ł2-norm regularization term to prevent overfitting.

Optimization. To minimize the objective function Ł, we need to optimize the following parameters: $\mathbf{W}_k, \hat{\mathbf{W}}_k, \mathbf{b}_k, \hat{\mathbf{b}}_k, k = 1, ..., K$. We use mini-batch stochastic gradient descent is used to optimize these parameters. We use $\mathbf{X}_i$ as a mini training batch, and thus the

Algorithm 2 Training Process

```
Input: the walk set $\Omega$, the first-order temporal proximity matrix $\mathbf{P}$, the parameters $\mu, \nu, \varphi$, the learning rate $\omega$.
Output: Network representations $\mathbf{Y}^K$.
 1: Pretrain the model by deep belief network to initial parameters $\mathbf{W}_k, \hat{\mathbf{W}}_k, \mathbf{b}_k, \hat{\mathbf{b}}_k, k = 1, ..., K$;
 2: repeat
 3:   for each walk path $L_i$ in $\Omega$ do
 4:     construct $\mathbf{X}_i$ according to $L_i$;
 5:     perform a feed-forward pass computer $Ł_m$;
 6:     for $k = K, K-1, ..., 1$ do
 7:       update $\hat{\mathbf{W}}_k = \hat{\mathbf{W}}_k - \omega \frac{\partial Ł_m}{\partial \hat{\mathbf{W}}_k}$;
 8:       update $\hat{\mathbf{b}}_k = \hat{\mathbf{b}}_k - \omega \frac{\partial Ł_m}{\partial \hat{\mathbf{b}}_k}$;
 9:     end for
10:     for $k = K, K-1, ..., 1$ do
11:       update $\mathbf{W}_k = \mathbf{W}_k - \omega \frac{\partial Ł_m}{\partial \mathbf{W}_k}$;
12:       update $\mathbf{b}_k = \mathbf{b}_k - \omega \frac{\partial Ł_m}{\mathbf{b}_k}$;
13:     end for
14:   end for
15: until converge
16: input $\mathbf{P}$ into the neural network to get $\mathbf{Y}^K$;
17: return $\mathbf{Y}^K$ as the embedding result.
```

objective function of one mini training batch is

$$Ł_m = ||(\mathbf{X}_i - \hat{\mathbf{X}}_i) \odot \mathbf{Z}_i||_2^2 + \mu tr({\mathbf{Y}_i^K}^T \mathbf{L}_i^1 \mathbf{Y}_i^K) + \nu tr({\mathbf{Y}_i^K}^T \mathbf{L}_i^2 \mathbf{Y}_i^K) + \varphi \frac{1}{2} \sum_{k=1}^{K} (||\mathbf{W}_k||_F^2 + ||\hat{\mathbf{W}}_k||_F^2). \tag{17}$$

The partial derivatives of the parameters in $Ł_m$ are estimated using the back-propagation algorithm. Similar to (Wang et al. 2017), in order to find a good region of parameters space and accelerate the training process, we adopt a deep belief network (Hinton, Osindero, and Teh 2006) to pretrain the parameters first. Finally, the training process of the deep guided auto-encoder is presented in Algorithm 2. The training complexity of HNIP is $O(nwtLdI)$, where n is the number of nodes, wt is the walk times, L is the length of each path, d is the maximum dimension of the hidden layer, and I is the iteration number.

Experimental Results

In this section, we employ four real-world networks to validate the effectiveness of HNIP on four application scenarios.

Temporal Networks

- **Leskovec-Ng** (Chen and III 2017): This is a co-author network containing the coauthors Prof. Jure Leskovec or Prof. Andrew Ng at Stanford University from year 1995 to year 2014. It has 191 nodes, 1,930 temporal links, and 2 different labels.
- **DBLP** (Zuo et al. 2018): This is also a co-author network derived from DBLP of ten research areas. It has 28,085 nodes, 236,894 temporal links, and 10 different labels.
- **Facebook** (Viswanath et al. 2009): This is a Facebook friendship network where nodes are users and links represent friendship. It has 46,952 nodes and 177,661 temporal links.
- **Twitter** (Conover et al. 2011): This is a retweet network of Twitter, where nodes are Twitter uses and links represent whether the users have retweeted each other. It has 18,470 nodes and 211,022 temporal links.

Baseline Methods and Experimental Settings

We compare HNIP with the following five state-of-the-art network embedding methods.

- **DeepWalk** (Perozzi, Al-Rfou, and Skiena 2014): This is a static embedding method that first applies random walks to generate sequences of nodes from the networks, and then uses these sequences as input to the Skip-gram model to learn representations.
- **Node2vec** (Grover and Leskovec 2016): This is a static embedding method that is generalized from DeepWalk by introducing a biased random walk procedure to explore the neighborhood of a node.
- **SDNE** (Wang, Cui, and Zhu 2016): This is a static embedding method that uses deep auto-encoders to jointly preserve first-order and second-order proximity.
- **CTDNE** (Nguyen et al. 2018): This is a dynamic embedding method that learns representations from temporal random walks that represent actual temporally valid sequences of node interactions.
- **NetWalk** (Yu et al. 2018): This is a dynamic embedding method that learns representations based on deep neural network embedding and reservoir sampling.

We propose a multi-layer deep neural network in this HNIP, where the number of layers varies with different datasets. The network structure corresponding to different datasets is detailed in Table 2.

For HNIP, the walk length T is set to 40, the walk times wt is set to 10, and the restart ratio r is set to 0.2. Other hyper-parameters are tuned by using a grid search on the validation set. For DeepWalk, CTDNE, and NetWalk, we set the window size as 10, the walk length as 40, and the walks times as 10. For SDNE, we set the neural network structure according to Table 2. The embedding size is set to be 128 for all the methods. For the three static embedding methods of DeepWalk, Node2vec, and SDNE, we first transform the temporal networks into static networks, and then perform network embedding.

Table 2: Neural Network Structures

Dataset	#nodes in each layer
Leskovec-Ng	191-128-128
DBLP	28085-5000-1000-128
Facebook	46952-5000-1000-128
Twitter	18470-1000-128

Table 3: $precision@k$ on DBLP Network

Algorithm	$P@100$	$P@1000$	$P@5000$	$P@10000$
DeepWalk	0.33	0.146	0.134	0.057
Node2vecE	0.25	0.122	0.103	0.062
SDNE	0.35	0.156	0.088	0.032
CTDNE	0.41	0.236	0.108	0.052
NetWalk	0.34	0.234	0.106	0.084
HNIP	**0.43**	**0.242**	**0.159**	**0.106**

Table 4: $precision@k$ on Twitter Network

Algorithm	$P@100$	$P@1000$	$P@5000$	$P@10000$
DeepWalk	0.17	0.152	0.034	0.045
Node2vec	0.17	0.102	0.105	0.112
SDNE	0.05	0.056	0.092	0.101
CTDNE	0.22	0.124	0.117	0.122
NetWalk	0.13	0.125	0.059	0.136
HNIP	**0.27**	**0.190**	**0.081**	**0.147**

Link Prediction

The aim of this task is to predict whether two nodes will interact with each other in the future. In this section, we experiment on the DBLP and Twitter networks. For each dataset, we divide the dynamic network into two parts by an assigned time point St. The first part is the training set and the latter is the test set. We first learn the embedding using the training set and predict the most likely links that exist in the test set from the learned embedding. In this paper, we set $St = 0.8$, where $T = 1$. The $precision@k$ (Goyal and Ferrara 2018) is adopted as the evaluation metric in this task. The results of all embedding methods on DBLP and Twitter are presented in Tables 3 and 4.

In Tables 3 and 4, the best performing algorithm is highlighted in bold. Obviously, HNIP outperforms all other algorithm in both DBLP network and Twitter network, which demonstrates that, by incorporating temporal proximity and a highly nonlinear network structure, HNIP has strong predictive ability. In addtion, the performances of temporal embedding methods (CTDNE, NetWalk, HNIP) are better than those of static methods (DeepWalk, Node2Vec, SDNE), which indicates that by modeling the network evolution pattern temporal, the temporal embedding method has stronger predictive ability than the static embedding method.

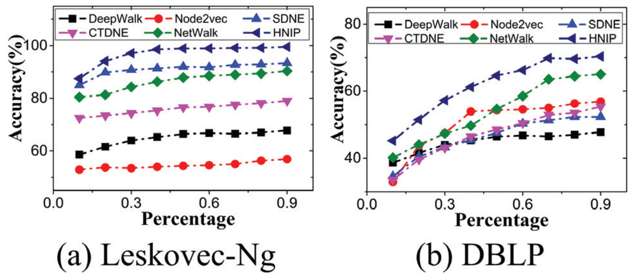

Figure 2: Classification accuracy on Leskovec-Ng network and DBLP network.

Table 5: $precision@k$ on Facebook Network

Algorithm	$P@100$	$P@1000$	$P@5000$	$P@10000$
DeepWalk	0.38	0.441	**0.423**	**0.469**
Node2vec	0.35	0.415	0.354	0.406
SDNE	**0.45**	0.432	0.375	0.464
CTDNE	0.24	0.265	0.287	0.256
NetWalk	0.31	0.237	0.156	0.154
HNIP	**0.45**	**0.451**	0.401	0.402

Node Classification

Given the learned node representations as node features, in this task we aim to train a classifier based on the node features to predict node labels. Following (Wang et al. 2017), we use the KNN algorithm to train the classifiers. When training the classifier, we randomly sample 10% to 90% of the nodes as the training set and use the left nodes as the test set to test the performance. The prediction accuracy is used as the evaluation metric. The results of all methods on Leskovec-Ng and DBLP are presented in Fig. 2.

As illustrated in Fig. 2, in the Leskovec-Ng network and DBLP network, the classification accuracy of all algorithms increases with the increase of training set size. The curve of HNIP is consistently above the curves of baselines, which demonstrates that the learned network representations of HNIP can better generalize to the classification task than to the baselines. This is because HNIP considers the temporal proximity of the nodes and can preserve the accurate node relationships. The nonlinear structure preservation also contributes to the performance of HNIP in the classification task.

Network Reconstruction

In this section, we concentrate on the network reconstruction task and experiment on a Facebook network. In this experiment, we transfer the temporal Enron network to a static network by simply setting the time stamp of each link to 1 and removing duplicate links. Then we randomly hide 20% of the links and learn the embedding using the rest of the 80% links. The $precision@k$ are adopted as the evaluation metric. The results is detailed in Table 5.

The results for the Facebook network shows that static embedding methods have stronger reconstructive ability than temporal embedding methods (except HNIP). This is not surprising because most temporal embedding methods focus on the modeling of the evolution pattern, whereas the reconstruction task requires a strong ability to preserve the network structure. Fortunately, HNIP can achieve near optimal performance, which indicates that HNIP can model the evolution pattern, and has a strong ability to preserve the temporal network structure.

Conclusion

In this paper, we propose the embedding method HNIP for temporal network embedding. Specifically, we first defined three different order temporal proximities by exploring the network historical information with a time exponential decay model to quantify the temporal proximity between nodes. Then, we proposed a deep guided auto-encoder, which is able to capture the highly nonlinear structure. Meanwhile, the training set of the deep guided auto-encoder is generated by the TRW algorithm. By training the proposed auto-encoder with a specific mini-batch stochastic gradient descent algorithm, HNIP can efficiently preserve the temporal proximities and highly nonlinear structure of temporal networks. Experimental results on four real-world networks demonstrated the effectiveness of the proposed method.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (No. 61976162, 61976161, and 61572369), in part by the Key Projects of Guangdong Natural Science Foundation (No. 2018B030311003).

References

Bengio, Y. 2009. Learning deep architectures for AI. *Foundations and Trends in Machine Learning* 2(1):1–127.

Cao, S.; Lu, W.; and Xu, Q. 2016. Deep neural networks for learning graph representations. In *Proceedings of the 30th AAAI*, 1145–1152.

Chen, P., and III, A. O. H. 2017. Multilayer spectral graph clustering via convex layer aggregation: Theory and algorithms. *IEEE Trans. Signal and Information Processing over Networks* 3(3):553–567.

Chen, C., and Tong, H. 2015. Fast eigen-functions tracking on dynamic graphs. In *Proceedings of the 2015 SIAM*, 559–567.

Conover, M.; Ratkiewicz, J.; Francisco, M.; Gonç alves, B.; Flammini, A.; and Menczer, F. 2011. Political polarization on twitter. In *ICWSM*.

Falzon, L.; Quintane, E.; Dunn, J.; and Robins, G. 2018. Embedding time in positions: Temporal measures of centrality for social network analysis. *Social Networks* 54:168–178.

Goyal, P., and Ferrara, E. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowl.-Based Syst.* 151:78–94.

Grover, A., and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD*, 855–864.

Hinton, G. E.; Osindero, S.; and Teh, Y. W. 2006. A fast learning algorithm for deep belief nets. *Neural Computation* 18(7):1527–1554.

Luo, D.; Ding, C. H. Q.; Nie, F.; and Huang, H. 2011. Cauchy graph embedding. In *Proceedings of the 28th ICML*, 553–560.

Munasinghe, L., and Ichise, R. 2012. Time score: A new feature for link prediction in social networks. *IEICE Transactions* 95-D(3):821–828.

Nguyen, G. H.; Lee, J. B.; Rossi, R. A.; Ahmed, N. K.; Koh, E.; and Kim, S. 2018. Dynamic network embeddings: From random walks to temporal random walks. In *IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018*, 1085–1092.

Peng, X.; Feng, J.; Xiao, S.; Yau, W.; Zhou, J. T.; and Yang, S. 2018. Structured autoencoders for subspace clustering. *IEEE Trans. Image Processing* 27(10):5076–5086.

Peng, X.; Huang, Z.; Lv, J.; Zhu, H.; and Zhou, J. T. 2019. COMIC: Multi-view clustering without parameter selection. In *Proc 36th Int Conf Machine Learning*, volume 97, 5092–5101.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk:online learning of social representations. In *Proceeding of the 20th ACM SIGKDD*, 701–710.

Rümmele, N.; Ichise, R.; and Werthner, H. 2015. Exploring supervised methods for temporal link prediction in heterogeneous social networks. In *Proceedings of the 24th WWW*, 1363–1368.

Sun, Y.; Yuan, Y.; and Wang, G. 2018. Scalable multiplex network embedding. In *Proceedings of the 27th IJCAI*, 3082–3088.

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 WWW*, 1067–1077.

Viswanath, B.; Mislove, A.; Cha, M.; and Gummadi, K. P. 2009. On the evolution of user interaction in facebook. In *WOSN*, 37–42.

Wang, X.; Cui, P.; Wang, J.; Pei, J.; Zhu, W.; and Yang, S. 2017. Community preserving network embedding. In *Proceedings of the 31st AAAI*, 203–209.

Wang, D.; Cui, P.; and Zhu, W. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD*, 1225–1234.

Xie, D.; Zhang, X.; Gao, Q.; Han, J.; Xiao, S.; and Gao, X. 2019. Multiview clustering by joint latent representation and similarity learning. *IEEE transactions on cybernetics*.

Yu, W.; Cheng, W.; Aggarwal, C. C.; Chen, H.; and Wang, W. 2017. Link prediction with spatial and temporal consistency in dynamic networks. In *Proceedings of the 26th IJCAI*, 3343–3349.

Yu, W.; Cheng, W.; Aggarwal, C. C.; Zhang, K.; Chen, H.; and Wang, W. 2018. Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In *Proceedings of the 24th ACM SIGKDD*, 2672–2681.

Zhang, Y.; Lyu, T.; and Zhang, Y. 2018. COSINE: community-preserving social network embedding from information diffusion cascades. In *Proceedings of the 32nd AAAI*, 2620–2627.

Zhou, L.; Yang, Y.; Ren, X.; Wu, F.; and Zhuang, Y. 2018. Dynamic network embedding by modeling triadic closure process. In *Proceedings of the 32nd AAAI*, 571–578.

Zhu, D.; Cui, P.; Zhang, Z.; Pei, J.; and Zhu, W. 2018. High-order proximity preserved embedding for dynamic networks. *IEEE Trans. Knowl. Data Eng.* 30(11):2134–2144.

Zuo, Y.; Liu, G.; Lin, H.; Guo, J.; Hu, X.; and Wu, J. 2018. Embedding temporal network via neighborhood formation. In *Proceedings of the 24th ACM SIGKDD*, 2857–2866.