# GTCN: Dynamic Network Embedding Based on Graph Temporal Convolution Neural Network

Zhichao Huang, Jingkuan Zhang, Lijia Ma(✉), and Fubing Mao

College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China
ljma@szu.edu.cn

**Abstract.** Network embedding aims to learn the low-dimensional node representations from high-dimensional network structures of complex systems. Embedding in dynamic networks is a very difficult but important problem due to the dynamics of network structures in real-world systems and the high computational complexity. In this paper, we propose a novel **G**raph **T**emporal **C**onvolution **N**etwork (short for GTCN) for the dynamic network embedding. In GTCN, a graph convolution network is used to learn the embedding representations of nodes in each snapshot, while a temporal convolutional network is adopted to parallelly reveal the evolution of node structures in dynamic networks. Extensive experiments on six real dynamic networks show that GTCN has a better performance than the state-art-of-art dynamic network embedding methods in tackling prediction tasks.

**Keywords:** Dynamic network · Network embedding · Graph convolution · Temporal convolution

## 1 Introduction

With the development of graphics technology, networks with link structures have become more and more worth studying [1]. In an actual system, nodes and links of a network represent the entities and communications of the system, respectively. However, with the rapid development of the Internet and information systems, communications and entities have grown exponentially. It has been difficult to calculate or express the structural properties of nodes and links in the network [2]. Network embedding has been proved an effective method to learn low-dimension feature of nodes in networks. The low-dimension feature not only preserves the structural features of nodes, but also obtains dense feature representations [3, 4].

Previous embedding methods mainly work on static networks [2, 5–7]. These models are usually designed to preserve local structures by GCN [7–9]. The embedding performance of this encoder-decoder architecture is determined by a loss function which determines the distance of local structures between the original and decoded networks. Classical network embedding methods include the matrix factorization-based approaches (DNR [10]) and the random walk based approaches (such as Deepwalk [5]


D.-S. Huang and K.-H. Jo (Eds.): ICIC 2020, LNCS 12464, pp. 583–593, 2020.
https://doi.org/10.1007/978-3-030-60802-6_51

and Node2vec [2]).The GCN architecture continuously merges information of neighbor nodes by means of message propagation.

The relationship between nodes in social networks and biomolecular networks always changes over time. Compared with static networks, dynamic networks are more suitable for these systems, in which the addition and deletion of nodes and links represent changes in node relationships in the system [11–13]. Moreover, in these networks, network embedding needs to consider both the structure and temporal features of the networks.

In recent years, some dynamic network embedding methods [14–16] have been proposed. For instance, the methods like dynGEM [16] and dynTriad [14] generated dynamic graph embedding based on a simplified assumption that graphs change smoothly. [15] improved classic skip-gram method for dynamic network embedding by updating the embedding of nodes whose links change most dramatic node embedding. dynRNN [17] and dynAERNN [17] used a recursive layer to solve the problem of inconsistent time spans. It is well known that Recurrent Neural Network (RNN) [18] is difficult to train and have high computational cost. Recently, many methods have been proposed for processing temporal data [19–21].

In this paper, we propose a novel model (called GTCN), which preserves both the structural and the temporal feature of dynamic network. GTCN uses GCNto get the embedding vector of each node in each snapshot. Then, it exploits TCN [22] to generate the nodes embedding that evolves over time. Extensive experiments on six real-world networks demonstrate that GTCN outperforms the state-of-the-art methods in link prediction.

## 2 Problem Definition

A dynamic network can be represented by $G = \{\mathcal{G}^1, \mathcal{G}^2, \ldots .\mathcal{G}^T\}$, where $T$ is the total time step, while $G^t$ is the snapshot of the dynamic graph at time stamp $t$. Each $\mathcal{G}^t = \{\mathcal{V}^t, \mathcal{E}^t\}$ represents undirected graph, where $\mathcal{V}^t$ denotes node set and $\mathcal{E}^t$ denotes link set at time $t$. Additionally, we can represent its links as a symmetric adjacency matrix $\mathrm{A}^t = [A_{ij}^t] \in \mathcal{R}^{n\times n}$, where $n$ is the number of nodes, with each entry $\{i,j\}$ of the matrix as

$$A_{ij}^t = \begin{cases} 1 & \text{if } \mathrm{e}_{\mathrm{ij}}^{\mathrm{t}} \in \mathcal{E}_{\mathrm{t}} \\ 0 & \text{otherwise} \end{cases}.$$

Dynamic network embedding aims to obtain the latent representation of each node, so that the latent representation $\mathrm{emb}_i^{t,2} \in \mathcal{R}^{f_2\times 1}$ (the embedding of node $i$) can capture both the current time graph structure information and the time information of the graph sequence.

## 3 Our Solution: GTCN

Our model consists of two parts. First, we apply GCN to deal with current time graph structure feature. Then, we exploit TCN to obtain past time temporal information.

### 3.1 Current Time Graph Structure Feature

This layer is consisted of 2 GCNs [7], used to grasp the current topology features of the graph.

For the first layer of GCN, the input are the adjacency matrix $\mathrm{A}^t \in \mathcal{R}^{n\times n}$ and the node feature matrix $\mathrm{F}^t \in \mathcal{R}^{n\times n}$ at time $t$.

Because the dataset used does not have node features, in general, the identity matrix is used as the node feature. Here, we use the following methods to calculate node features.

$$\begin{aligned}\hat{\mathrm{F}}^t &= \mathrm{I}_N + \mathrm{A}^t \\ \mathrm{F}^t &= \left(1 + t_d \cdot \mathrm{F}^{t-1}\right) \odot \hat{\mathrm{F}}^t,\end{aligned} \tag{1}$$

where $t \in \{2, \ldots, T\}$, and $\mathrm{F}^1 = \mathrm{I}_N + \mathrm{A}^1$. $\mathrm{I}_N \in \mathcal{R}^{n\times n}$ is the identity matrix, $t_d$ is time decay value, a hyper parameter and $\odot$ is element-wise multiply. That means the longer time the link remains, the higher the similarity between two nodes the link connected, but once the link disappears, the similarity of the corresponding two nodes become zero and re-accumulated.

The output of the first GCN layer $l_1$ of a single snapshot is calculated as follows:

$$\mathrm{Z}_s^{t,l_1} = \mathbf{relu}\left(\hat{\mathrm{A}}^t \cdot \mathrm{F}^t \cdot \mathrm{W}_s^{t,l_1}\right), \tag{2}$$

where $\mathrm{Z}_s^{t,l_1} \in \mathcal{R}^{n\times d_1}$ and $\mathrm{W}_s^{t,l_1} \in \mathcal{R}^{n\times d_1}$ is the parameters with dimensionality $d_1$ of the first GCN layer to be learned and **relu** is defined as:

$$\mathbf{relu}(x) = \begin{cases} 1 & \text{if } \mathrm{x} > 0 \\ 0 & \text{otherwise} \end{cases},$$

and $\hat{\mathrm{A}}^t$ is defined as:

$$\begin{aligned}\tilde{\mathrm{A}}^t &= \mathrm{A}^t + \mathrm{I}_N \\ \tilde{\mathrm{D}}_{ii}^t &= \sum_j \tilde{\mathrm{A}}_{ij}^t \\ \hat{\mathrm{A}}^t &= (\tilde{\mathrm{D}}^t)^{-\frac{1}{2}} \cdot \tilde{\mathrm{A}}^t \cdot (\tilde{\mathrm{D}}^t)^{-\frac{1}{2}}.\end{aligned}$$

For the second GCN layer $l_2$, the output of the first GCN layer $\mathrm{Z}_s^{t,l_1}$ serves as the node feature of the second GCN layer and the output of second GCN layer is calculated as:

$$\mathrm{Z}_s^{t,l_2} = \hat{\mathrm{A}}^t \cdot \mathrm{Z}_s^{t,l_1} \cdot \mathrm{W}_s^{t,l_2}, \tag{3}$$

where $\mathrm{Z}_s^{t,l_2} \in \mathcal{R}^{n \times d_2}$ and $\mathrm{W}_s^{t,l_2} \in \mathcal{R}^{d_1 \times d_2}$ is the parameters with dimensionality $d_2$ of the second GCN layer.

Finally, we need to use the structure embedding vector [23] of each node $\mathrm{Z}_s^{t,l_2}$ to reconstruct the adjacency matrix.

$$\hat{\mathrm{A}}_s^t = \mathbf{sigmoid}\left(\mathrm{Z}_s^{t,l_2} \cdot (\mathrm{Z}_s^{t,l_2})^T\right), \tag{4}$$

where $(\mathrm{Z}_s^{t,l_2})^T$ means transpose of matrix $\mathrm{Z}_s^{t,l_2}$. **sigmoid** is an activation function, defined as $\mathbf{sigmoid}(x) = \frac{1}{1+e^x}$.

Then, we compare the reconstructed matrix $\hat{\mathrm{A}}_s^t$ with the ground truth adjacency matrix $\mathrm{A}_t$ at the current moment $t$ as structure loss. Our goal is to minimize the structure loss of time step $t$.

$$\mathcal{L}_{\mathcal{S}} = \sum_{t=1} norm_s^t \left\| \left(\hat{\mathrm{A}}_s^t - \mathrm{A}^t\right) \odot \mathrm{B}_s^t \right\|_F^2, \tag{5}$$

where $norm_s^t$ is the normalization coefficient, defined as:

$$norm_s^t = \frac{n \times n}{2(n \times n - \|\mathcal{E}^t\|)},$$

$\mathrm{B}_s^t \in \mathcal{R}^{n \times n}$ is the penalty matrix

$$\mathrm{B}_{s,ij}^t = \begin{cases} \frac{n \times n - \|\mathcal{E}^t\|}{\|\mathcal{E}^t\|} & \text{if } \mathrm{e}_{\mathrm{ij}}^{\mathrm{t}} \in \mathcal{E}^{\mathrm{t}} \\ 0 & \text{otherwise} \end{cases},$$

where $\mathrm{B}_{s,ij}^t$ is the element at the $i_{th}$ row and $j_{th}$ column of matrix $\mathrm{B}^t$ and $\odot$ is element-wise multiply.

### 3.2 Past Time Temporal Feature

In this section, we will use TCN to solve the temporal problem. In order to predict the adjacency matrix of $\mathcal{G}^{t+1}$, we need to consider all the time series information before time $t+1$, that is $\{\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^t\}$.

The input of this layer is the structural feature of each node of the graph at time $\{1, 2, \ldots, t\}$. Here we use the node representation $\mathrm{Z}_s^{t,l_2}$ obtained from the GCN above to be the structural feature. We use $\mathrm{Z}_{s,i*}^{t,l_2} \in \mathcal{R}^{1 \times d_2}$, the $i_{th}$ row of the matrix, to represent the structural feature of node $i$ in $\mathcal{G}_t$, then our input for the prediction of each node of $\mathcal{G}^t$ is $\{\mathrm{Z}_{s,i*}^{1,l_2}, \mathrm{Z}_{s,i*}^{2,l_2}, \ldots, \mathrm{Z}_{s,i*}^{t,l_2}\}$.

Here we use 2 layers of TCN [22] as a model for processing temporal information. Each layer of TCN consists of a dilated convolution, a causal convolution, and a skip connection.

For convenience, we set $\mathrm{Z}_{s,i*}^{t,l_2}$ as:

$$\mathrm{H}_i^{t,0} = (\mathrm{Z}_{s,i*}^{t,l_2})^T,$$

where $\mathrm{H}_i^{t,0} \in \mathcal{R}^{d_2 \times 1}$ means the embedding vector of the node $i$ of $\mathcal{G}^t$ in the layer 0, *i.e.*, the $i_{th}$ row of $\mathrm{Z}_s^{t,l_2}$.

For each layer $l$, dilated convolution and causal convolution is calculated as follows:

$$\hat{\mathrm{H}}_i^{t,l} = \mathbf{relu}\left(\sum_{j=0}^{k-1} \mathrm{W}_j^l \cdot \mathrm{H}_i^{(t-dj),(l-1)}\right), \tag{6}$$

where $\mathrm{W}_j^l \in \mathcal{R}^{f_i \times f_{i-1}}$, $f_l$ is the number of filter (dimension of output) in $l_{th}$ layer, $d$ is the dilation factor and $k$ is the filter size, and $t - dj$ means the past focused step. Therefore, you can consider $d$ as the number of steps between two adjacent columns of the filter. When $d$ is equal to 1, it is equivalent to usual causal convolution. The larger $d$ enables an exponentially large receptive field, *i.e.*, more information of graph dynamic evolution.

For each layer, skip connection (residual connection) [24], add the original input to the output,

$$\mathrm{H}_i^{t,l} = \mathbf{relu}\left(\mathrm{W}^l \cdot \mathrm{H}_i^{t,(l-1)} + \hat{\mathrm{H}}_i^{t,l}\right), \tag{7}$$

where $\mathrm{W}^l \in \mathcal{R}^{f_i \times f_{i-1}}$, used to make the dimension of original input and output equal.

Then, we use a linear layer and the output of the last TCN layer (here we use two layer TCN, so $\mathrm{emb}_i^{t,3}$ is the output of the last TCN layer) to reconstruct the vector representation of the node $i$ at the next time step $t+1$,

$$\hat{\mathrm{V}}_i^{t+1} = \mathbf{sigmoid}\left(\mathrm{W}_d \cdot \mathrm{H}_i^{t,2} + \mathrm{B}_d\right), \tag{8}$$

where $\mathrm{W}_d \in \mathcal{R}^{n \times f_2}$ and $\mathrm{B}_d \in \mathcal{R}^{n \times 1}$ are the parameters of the linear layer.

After obtaining the prediction vector representation of node $i$ at next time step $t+1$, we need to compare the difference between the predicted value and the ground truth, and use this as our temporal loss to optimize,

$$\mathcal{L}_{\mathcal{T}} = norm_t^t \sum_{t=0}^{T} \sum_{i=0}^{n} \left\| \left(\hat{\mathrm{V}}_i^{t+1} - \mathrm{V}_i^{t+1}\right) \odot \mathrm{B}_{t,i*}^t \right\|_F^2, \tag{9}$$

where $T$ is the total time step before $t+1$ and $n$ is the number of nodes.$\mathrm{V}_i^{t+1}$ is the $i_{th}$ row of the adjacency matrix A at time $t+1$. $norm_t^t$ and $\mathrm{B}_t^t$ are the same as $norm_s^t$ and $\mathrm{B}_s^t$ separately except that it used $t+1$ nodes and edges information. And $\mathrm{B}_{t,i*}^t \in \mathcal{R}^{n\times 1}$ is the $i_{th}$ row of $\mathrm{B}_t^t$.

### 3.3 Total Architecture

In this section, we will present the key components ofour overall model.

**Current Time Graph Structure Feature Block:** For this module, we aim to capture the topological features and the neighbor characteristics of the nodes at each time graph $\mathcal{G}_t$. The parameters at each time step are not shared, the output of this module will be used as input to the next temporal module.

**Past Time Temporal Feature Block:** In this module, we will capture the temporal features of the graph dynamic changing. We use all the graph structure information before $t+1$ as input, *i.e.*, $\{\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^t\}$, and use TCN to predict $\mathcal{G}^{t+1}$.

**Objective:** we define the loss for optimization as:

$$\mathcal{L} = \min(\mathcal{L}_{\mathcal{S}} + \alpha\mathcal{L}_{\mathcal{T}} + \lambda\mathcal{L}_{reg}), \tag{10}$$

where $\mathcal{L}_{\mathcal{S}}$ is the graph structure feature loss and $\mathcal{L}_{\mathcal{T}}$ is the graph dynamic feature loss.$\alpha$ is the parameters to control the contribution of $\mathcal{L}_{\mathcal{S}}$ and $\mathcal{L}_{\mathcal{T}}$, which is between 0 and 1. $\lambda$ is the parameter to control contribution of $\mathcal{L}_{reg}$ and $\mathcal{L}_{reg}$ is an L2-norm regularizer to prevent overfitting, defined as:

$$\mathcal{L}_{reg} = \frac{1}{2}\|W\|_F^2,$$

where $W$ is all the parameters that need to be learned.

**Optimization:** We use the adam [25] optimizer.

## 4 Experimental Results

In this section, firstly we introduce 6 real-world dynamic networks as our data set, and compare 5 models including 1 static graph embedding method and 4 dynamic embedding approaches. Then, we analyze the experimental results of the model under different data sets.

**Table 1.** Statistics of dataset used in our experiments.

| Network | Nodes | Links | Time-step |
|---|---|---|---|
| fb-forum | 899 | 33720 | 7 |
| Hypertext | 113 | 20818 | 7 |
| Hospital | 75 | 32424 | 8 |
| School | 242 | 125773 | 7 |
| Email | 167 | 82927 | 7 |
| Workplace | 92 | 9827 | 8 |

### 4.1 Experimental Settings

**Datasets.** It consists of 6 real-life networks that include web forums, networks for medical, school, workplace and citation relationship. The basic properties of the data set are described in Table 1.

**Comparison Models.** We compare the performance of our model with other state-of-the-art models including one static graph embedding model (SDNE [26]) and four dynamic graph embedding models (dynGEM[16], Dyn2vecAE [17], Dyn2vecRNN [17] and Dyn2vecAERNN [17]).

**Task.** In real networks, links between nodes tend to follow time changes. Therefore, it is very important to use past node link information to predict changes in node connections at the next moment. Here, we use link prediction as the criterion for judging the pros and cons of our model. In the experiment, we use the information of the past nodes $1, 2, 3, \ldots, t$ to predict the embedding of nodes at the next moment $t+1$, and calculate the relationship between nodes at $t+1$ snapshot with the binary classification model. Finally, we compare the predicted node association with the ground truth connections and adopt Mean Average Precision (mAP) as judging criteria.

We conduct experiments on the tasks of link prediction in dynamic graphs. Firstly, we train the model and acquire the corresponding learned parameters on snapshots $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_t\}$ and apply $\hat{\mathrm{V}}_{t+1}$ to predict links in $\mathcal{G}_{t+1}$ during evaluation. At each snapshot of the graph, links could be disappeared or added. We compare the performance of different models in link prediction based on their own abilities.

**Metrics.** To evaluate the performance of link prediction learned by our model, we use Mean Average Precision (mAP) and precision@k as the metrics.

**Detailed Settings.** For our datasets with short time steps, when training the model to predict $\mathcal{G}_{t+1}$, we will exploit all the graphs before time $t+1$, *i.e.*, $\{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_t\}$. In our experiment, we predict the $T$, $T-1$, $T-2$ and $T-3$ graphs for each data set, and train 20 times for each experiment. Finally, the average of the mAPs for 20 times with 4 time steps is selected as the final assessment metric.

**Table 2.** Experiment results on dynamic link prediction (mAP)

| Network | SDNE | | dynGEM | | dynAE | |
|---|---|---|---|---|---|---|
| | Average | Max | Average | Max | Average | Max |
| fb-forum | 0.0001 | 0.0001 | 0.0001 | 0.0002 | 0.0005 | 0.0005 |
| Hypertext | 0.4294 | 0.4383 | 0.3690 | 0.3841 | 0.3690 | 0.4211 |
| Hospital | 0.4738 | 0.4865 | 0.4637 | 0.4730 | 0.4515 | 0.4679 |
| School | 0.3619 | 0.3969 | 0.3484 | 0.3548 | 0.2623 | 0.3009 |
| Email | 0.0003 | 0.0003 | 0.0001 | 0.0002 | 0.2270 | 0.2568 |
| Workplace | 0.4182 | 0.4346 | 0.3735 | 0.4005 | 0.1970 | 0.2118 |
| Network | DynRNN | | DynAERNN | | GTCN [Ours] | |
| | Average | Max | Average | Max | Average | Max |
| fb-forum | 0.0002 | 0.0002 | 0.0051 | 0.0054 | **0.0274** | **0.0315** |
| Hypertext | 0.3818 | 0.3994 | 0.4039 | 0.4102 | **0.4816** | **0.5022** |
| Hospital | 0.4826 | 0.5111 | **0.5195** | **0.5339** | 0.4716 | 0.4885 |
| School | 0.4526 | 0.4712 | 0.3022 | 0.3121 | **0.5214** | **0.5406** |
| Email | 0.1605 | 0.2004 | 0.2475 | 0.2517 | **0.2661** | **0.2812** |
| Workplace | 0.2966 | 0.3296 | 0.3816 | 0.4052 | **0.4882** | **0.5033** |

Here, we choose 2-layer GCNs to extract the topology structural feature of each snapshot of graphs, and 3-layer TCNs to obtain the dynamic evolution feature.

All the experiments are conducted on Linux platform with 2 GPU cores (Tesla P100) and 32 GB RAM.

### 4.2 Result and Analysis

In this section we present the performance of different models for link prediction on different datasets.

Table 2 shows the performance (mAP) of each algorithm on each data set. Among them, we take mAP as the criterion for judging the pros and cons of the model. The larger the mAP, the better performance of the model has.

**Table 3.** Experiment results on dynamic link prediction (Precision@k part-1)

| Method | Fb-forum(avg) | | | Hypertext(avg) | | | Email(avg) | | |
|---|---|---|---|---|---|---|---|---|---|
| | P@100 | P@500 | P@1000 | P@100 | P@500 | P@1000 | P@100 | P@500 | P@1000 |
| SDNE | 0.000 | 0.001 | 0.000 | 0.323 | **0.404** | **0.397** | 0.000 | 0.000 | 0.001 |
| dynGEM | 0.000 | 0.001 | 0.000 | 0.065 | 0.164 | 0.220 | 0.000 | 0.000 | 0.000 |
| dynAE | 0.000 | 0.000 | 0.000 | 0.095 | 0.237 | 0.260 | 0.000 | 0.011 | 0.031 |
| dynRNN | 0.003 | 0.001 | 0.000 | 0.373 | 0.340 | 0.354 | 0.090 | 0.174 | 0.190 |
| dynAERNN | 0.003 | 0.001 | 0.003 | 0.395 | 0.383 | 0.367 | 0.068 | **0.202** | **0.229** |
| GTCN [ours] | **0.003** | **0.006** | **0.005** | **0.425** | 0.387 | 0.372 | **0.138** | 0.122 | 0.125 |

**Table 4.** Experiment results on dynamic link prediction (Precision@k part-2)

| Method | School (avg) | | | Workplace (avg) | | | Hospital (avg) | | |
|---|---|---|---|---|---|---|---|---|---|
| | P@100 | P@500 | P@1000 | P@100 | P@500 | P@1000 | P@100 | P@500 | P@1000 |
| SDNE | 0.313 | 0.340 | 0.320 | 0.270 | 0.275 | 0.245 | 0.350 | 0.408 | 0.387 |
| dynGEM | 0.315 | 0.342 | 0.309 | 0.183 | 0.227 | 0.206 | 0.208 | 0.326 | 0.359 |
| dynAE | 0.118 | 0.136 | 0.124 | 0.160 | 0.131 | 0.135 | 0.138 | 0.269 | 0.322 |
| dynRNN | 0.318 | 0.242 | 0.200 | 0.290 | 0.311 | 0.281 | 0.255 | 0.423 | 0.442 |
| dynAERNN | 0.330 | 0.251 | 0.218 | 0.258 | 0.232 | 0.224 | 0.350 | **0.445** | **0.453** |
| GTCN [ours] | **0.705** | **0.575** | **0.502** | **0.615** | **0.430** | **0.353** | **0.385** | 0.390 | 0.396 |

From Table 2, we can see that SDNE, dynGEM, dynAE, dynRNN and dynAERNN perform very poorly on fb-forum data set (smaller than 0.01 and 20 times or smaller than the mAP of GTCN). The reason is that the edges in fb-forum change very sharply at each moment, and both SDNE and dynGEM just consider the adjacency matrix of the graph at the current moment as the training set to predict the graph structure of next time step. Therefore, the violent changes cause SDNE and dynGEM to become very difficult to predict next snapshot of the dynamic network. The dynAE, dynRNN and dynAERNN methods can find long historical data information, but the structure of Auto-encoder makes it difficult for them to mine the characteristics of sparse graph structures. However, GTCN achieves better performance since GTCN uses TCN to discover historical information, and then adopts GCN to cooperate with time feature to some extent to alleviate the impact of the sparse graph structure. However, the degree of the node of fb-forum is very small which results in the adjacency matrix being sparse. Thus, the number of edges that can be trained is small, the training is insufficient and the number of nodes is too large which results in lower accuracy for prediction. Hence, all the algorithms are not very good and even the best performance (mAP) can only reach about 0.02. For email dataset, we also find the same situation that using only the information from the previous moment is not enough to discover the structural characteristics of the dynamic network.

Compared to other dynamic graph embedding algorithms, GTCN performs the best on most of data sets. We attribute this to the GCN and the preprocessing of timing features which contain the node features of each snapshot with a rough summary of temporal information. Compared to dynGEM mentioned above, the model only focuses on the changes in the links of the previous 1 time step graph, which is too short-sighted. Compared with dynAE and dynRNN, the AE and RNN structures are not as good as GCN for capturing the neighbor structure of each node in the graph. Furthermore, the combination of structural features and temporal features are also impossible for AE and RNN.

Table 3 and Table 4 show the precision@ k values for various data sets. The larger the precision@ k is, the better the model performs. As we explained above, fb-forum has many nodes, but links between nodes are sparse and change very sharply, which leads to poor model learning. However, our model can still achieve better performance than other models. For other small data sets, our model can also outperform than other models in most of the datasets in P@100, P@500 and P@1000.

## 5 Conclusion

In this paper, we have proposed a dynamic network embedding method GTCN. The model first captured the topology properties of the nodes in the graph by the GCN, and then adopted TCN to capture the dynamic temporal changes of the nodes. Experimental results have showed that our model not only captured the topology properties of nodes, but also extracted dynamic temporal changes. They have also indicated that our proposed method outperforms the state-of-the-art methods in link predictions.

**Acknowledgement.** This work was supported by the National Natural Science Foundation of China under Grant 61803269, in part by the Natural Science Foundation of Guangdong Province under Grant 2020A1515010790, and in part by the Technology Research Project of Shenzhen City under Grant by JCYJ20190808174801673.

## References

1. Facchetti, G., Iacono, G., Altafini, C.: Computing global structural balance in large-scale signed social networks. Proc. Natl. Acad. Sci. **108**(52), 20953–20958 (2011)
2. Grover, A., Leskovec, J.: Node2vec: scalable feature learning for networks. In: ACMSIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, USA, pp. 855–864 (2016)
3. Hamilton, W.L., Ying, R., Leskovec, J.: Representation learning on graphs: Methods and applications. CoRR**abs/1709.05584** (2017)
4. Zhang, D., Yin, J., Zhu, X., Zhang, C.: Network representation learning: a survey. IEEE Trans. Big Data (2018). https://doi.org/10.1109/tbdata.2018.2850013
5. Perozzi, B., Al-Rfou, R., Skiena, S.: DeepWalk: online learning of social representations. In: ACMSIGKDD International Conference on Knowledge Discovery and Data Mining, NewYork, USA, pp. 701–710 (2014)
6. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: large-scale information network embedding. In: World Wide Web Conference. Florence, Italy, May 2015
7. Kipf, T.N., Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks. In: International Conference on Learning Representations. Palais des Congrès Neptune, Toulon, France (2017)
8. Wang, X., Cui, P., Wang, J., Pei, J., Zhu, W., Yang, S.: Community preserving network embedding. In: AAAI Conference on Artificial Intelligence, San Francisco, California USA. AAAI (2017)
9. Yang, L., Cao, X., He, D., Wang, C., Wang, X., Zhang, W.: Modularity based community detection with deep learning. In: International Joint Conference on Artificial Intelligence, New York, USA, pp. 2252–2258 (2016)
10. Skrlj, B., Kralj, J., Konc, J., Robnik-Sikonja, M., Lavrac, N.: Deep node ranking: an algorithm for structural network embedding and end-to-end classification. CoRR**abs/1902.03964** (2019)
11. K., E.G., Mirzasoleiman, B., Grosu, R., Leskovec, J.: Dynamic network model from partial observations. In: Advances in Neural Information Processing Systems. Montréal, Canada (2018)
12. Rossetti, G., Cazabet, R.: Community discovery in dynamic networks: a survey. CoRR**abs/1707.03186** (2017)

13. Sekara, V., Stopczynski, A., Lehmann, S.: Fundamental structures of dynamic social networks. Proc. Nat. Acad. Sci. United States Am, p. 201602803 (2016)
14. Du, L., Wang, Y., Song, G., Lu, Z., Wang, J.: Dynamic network embedding by modeling triadic closure process. In: AAAI Conference on Artificial Intelligence, New Orleans, USA (2018)
15. Du, L., Wang, Y., Song, G., Lu, Z., Wang, J.: Dynamic network embedding: an extended approach for skip-gram based network embedding. In: International Joint Conference on Artificial Intelligence (2018)
16. Goyal, P., Kamra, N., He, X., Liu, Y.: Dyngem: deep embedding method for dynamic graphs. CoRR**abs/1805.11273** (2018)
17. Goyal, P., Chhetri, S.R., Canedo, A.: Dyngraph2vec: capturing network dynamics using dynamic graph representation learning. Knowl. Based Syst. **187**, 104816 (2020)
18. Lipton, Z.C.: A critical review of recurrent neural networks for sequence learning. Computer Science (2015)
19. Sankar, A., Wu, Y., Gou, L., Zhang, W., Yang, H.: Dysat: deep neural representation learning on dynamic graphs via self-attention networks. In: International Conference on Web Search and Data Mining, Houson, USA, pp. 519–527 (2020)
20. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, Long Beach, USA, pp. 5998–6008 (2017)
21. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. In: International Conference on Learning Representation, Vancouver, Canada (2018)
22. Bai, S., Kolter, J.Z., Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. CoRR **abs/1803.01271** (2018)
23. Pan, S., Hu, R., Long, G., Jiang, J., Yao, L., Zhang, C.: Adversarially regularized graph autoencoder for graph embedding. In: International Joint Conference on Artificial Intelligence, Stockholm, Sweden, pp. 2609–2615 (2018)
24. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, USA, pp. 770–778 (2016)
25. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. Computer Science (2014)
26. Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, USA, pp. 1225–1234 (2016)