

# FeatureNorm: L2 Feature Normalization for Dynamic Graph Embedding

Menglin Yang, Ziqiao Meng, and Irwin King  
Department of Computer Science and Engineering  
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong  
{mlyang, zqmeng, king}@cse.cuhk.edu.hk

**Abstract**—Dynamic graphs arise in a plethora of practical scenarios such as social networks, communication networks, and financial transaction networks. Given a dynamic graph, it is fundamental and essential to learn a graph representation that is expected not only to preserve structural proximity but also jointly capture the time-evolving patterns. Recently, graph convolutional network (GCN) has been widely explored and used in non-Euclidean application domains. The main success of GCN, especially in handling dependencies and passing messages within nodes, lies in its approximation to Laplacian smoothing. As a matter of fact, this smoothing technique can not only encourage must-link node pairs to get closer but also push cannot-link pairs to shrink together, which potentially cause serious feature shrink or oversmoothing problem, especially when stacking graph convolution in multiple layers or steps. For learning time-evolving patterns, a natural solution is to preserve historical state and combine it with the current interactions to obtain the most recent representation. Then the serious feature shrink or oversmoothing problem could happen when stacking graph convolution explicitly or implicitly according to current prevalent methods, which would make nodes too similar to distinguish each other. To solve this problem in dynamic graph embedding, we analyze the shrinking properties in the node embedding space at first, and then design a simple yet versatile method, which exploits L2 feature normalization constraint to rescale all nodes to hypersphere of a unit ball so that nodes would not shrink together, and yet similar nodes can still get closer. Extensive experiments on four real-world dynamic graph datasets compared with competitive baseline models demonstrate the effectiveness of the proposed method.

**Index Terms**—Dynamic graph embedding, Feature shrink, Normalization, Graph convolutional network

## I. INTRODUCTION

Graphs are ubiquitous non-Euclidean data structures applied in various scenes, such as social networks, communication networks, and financial transaction networks, etc. Learning node representations in a latent space while preserving structural properties and interactive information stored in the original graph is one of the fundamental problems, which has attracted much attention in machine learning communities.

Most existing node embedding works assume the graph is static and is associated with a fixed set of nodes and edges [1], [2], [3]. However, networks in many real-world application scenarios are intrinsically time-evolving. Evidence can be found in social networks, (traffic) communication networks, and financial transaction networks. Modeling dynamic graphs is challenging due to complicated cases brought by an evolving process: nodes can be added or removed, edges

can appear or disappear, and communities can be merged or split. For instance, in social and email communication networks, people tend to frequently add and sometimes remove their connections based on daily business affairs; in IP-IP networks, agents periodically send messages from one address to any other address in the whole network, which makes the temporal model tough to capture the true evolving regularities. In short, the inherent difficulties stated above result in a lack of convincing dynamic graph models so far.

A surge of graph research works emerged after the simplified graph convolutional neural network (GCN) was proposed by Kipf and Welling [4]. However, the latest developments on GCN show that applying multiple graph convolutional layers would introduce oversmoothing problem, i.e., nodes in a graph become more and more similar thus being indistinguishable. The reason is that graph convolution is a special type of Laplacian smoothing. By stacking graph convolutional layers repeatedly, the feature space will shrink together and the final node embedding matrix will converge to a low-rank matrix, which results in nodes being almost identical and indistinguishable representation. It was first introduced by Li, et al. [5] and further studied by [6], [7], [8]. Yet, we notice that very few works investigate this problem in dynamic graph models, although it harasses the development of dynamic graph research. In fact, in dynamic graph setting, stacking multiple graph convolutional layers to capture topological and temporal properties is an indispensable procedure according to present widely-used methods, but serious feature shrink or oversmoothing problems will also occur. Here we use the term “serious feature shrink” to express the same meaning as oversmoothing phenomenon, but at the same time to emphasize the cause of oversmoothing.

In short, in this paper we first analyze the feature shrink and oversmoothing problems in the dynamic graph embedding. Then we propose a simple yet versatile method, that is L2 feature normalization, to tackle the problem. Finally, extensive experiments show that the performance can be remarkably improved by the proposal. To summarize, the main contributions of our work are as follows:

- We analyze the feature shrink and oversmoothing problems in dynamic graph embedding, which is the first work to the best of our knowledge.
- An effective and versatile feature normalization method is proposed to tackle serious feature shrink or oversmooth-

ing issue in dynamic graph embedding.

- Comprehensive experiments demonstrate that our proposed method can not only improve the performance of various baselines obviously, but also can be served as a plug-in to boost the related models and unleash their true capabilities.

## II. RELATED WORKS

Dynamic graph embedding is an extension of static node embedding with an additional attention on the temporal-evolving information. Related works are generally carried out from two aspects: topological dependencies and temporal-evolving patterns.

**Topological dependencies learning.** Early works for node embedding mainly center on factorization-based approaches, resorting to dimensionality reduction of graph Laplacian matrix [9], [10], [11], [12]. These methods suffer from time and memory efficiency. To improve scalability, the random walk-based methods transfer the embedding into a network mining task, which is inspired by the success in the natural language process [13]. Recently, the generalizations of convolutions over graph-like data have achieved remarkable success. Defferrard et al. [14] first define graph convolutions using Chebyshev polynomial and reduce the computation significantly. Kipf and Welling [4] simplify graph convolution using the first-order Chebyshev polynomial filters with an symmetric normalized adjacency matrix. Hamilton et al. [15] propose GraphSAGE for inductive learning. Peter et al. [16] further put forward a graph attention network (GAT) to explore the multiple attention mechanism in neighborhood message passing. Although these methods are proposed in a static graph, they are currently widely used in the structure learning of dynamic graph embedding [17], [18], [19], [20], [21], [22].

**Temporal-evolving patterns learning.** To learn the temporal-varying patterns, most works adopt an temporal-smoothness or recurrent learning fashion (definitions see III-A), which is in line with the nature of the dynamism. In related literature, BCDG [23] utilizes matrix decomposition approach, which explores the smoothness across consecutive time steps to model the time-evolving patterns. NetWalk [13] takes a random walk method, using an autoencoder framework and minimizing the pairwise distance of vertex representation of each walk. DynGEM [24] also adopts the autoencoder framework that constrains the local and global structure proximity. With the success of graph convolution, most recent works mainly leverage an integrated GCN model to capture the structure information. The main difference lies in the temporal regularities learning. DySAT [17] brings self-attention mechanisms in both temporal and structural aspects. EvolveGCN [18] models the dynamism of the dynamic graph by using an RNN to capture the GCN parameters.

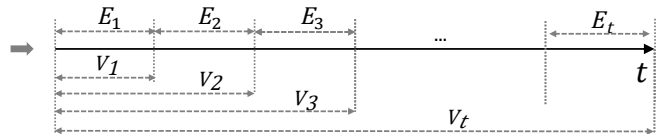


Fig. 1. Slicing scheme of nodes and edges in dynamic graph, where the edges reflect current connections and topology, and node set contains ever appeared nodes till present time step.

**Oversmoothing in GNNs.** Our work is also related to oversmoothing problem, and related works are as follows. Li et al. [5] first pay attention to the oversmoothing problem, and they prove that graph convolution is a special case of Laplacian smoothing. In other words, applying graph convolutional layers repeatedly will result in the embedding feature being indistinguishable. Xu et al. [7] propose a jumping knowledge network by utilizing skip connections for multi-hop message passing to overcome oversmoothing. Klicpera et al. [8] put forward a customized Pagerank to propagate messages. Klicpera et al. [25] consider adding residual connections like ResNet [26], using skip scheme to connect the cross-layer features. Rong et al. [27] argue that randomly removing a certain fraction of edges can reduce the oversmoothing problem. Zhao and Akoglu [6] use the total pairwise distance to avoid the feature being washed away. The above methods mainly design for static graphs and do not consider the temporal condition. Our work is to tackle oversmoothing, more precisely i.e., serious feature shrink in dynamic graph settings with temporal information.

## III. PROPOSED METHOD

### A. Problem Definition

Given a dynamic graph  $G$  with  $T$  graph snapshots  $\{G_1, \dots, G_t, \dots, G_T\}$ . Each snapshot  $G_t = (V_t, E_t)$  is a weighted or unweighted, directed or undirected graph.  $V_t$  is the observed node set from beginning to the time step  $t$ , meaning that we can handle variable-sized input, which is more reasonable for realistic scenes because the future new nodes are always agnostic to the current time step  $t$ . The number of nodes is non-decreasing since we need to learn their representations for further prediction in case that they appear again.  $E_t$  denotes the links that appear in the current time interval  $[t-1, t]$ , where the slicing scheme is illustrated in Figure 1. The goal of dynamic graph embedding is to acquire the node representation at time step  $t$  that can not only preserve topological dependencies but also capture the time-varying behaviors.

Dynamic graph embedding can be summarized as the paradigm illustrated in Figure 2. At each time step  $t$ , the input is the adjacency matrix  $\mathbf{A}_t \in \mathbb{R}^{n \times n}$  constructed from  $E_t$  which reflects the connections within nodes and determines the topology of  $G_t$ , and the corresponding node features  $\mathbf{X}_t \in \mathbb{R}^{n \times d_i}$ , where  $n$  is the number of nodes and  $d_i$  is dimension of the input feature. Dynamic graph modeling can be viewed as learning a mapping function  $f$  given the current network  $(\mathbf{A}_t, \mathbf{X}_t)$  and historical hidden state  $\mathbf{H}_{t-1} \in \mathbb{R}^{n \times d_h}$ .

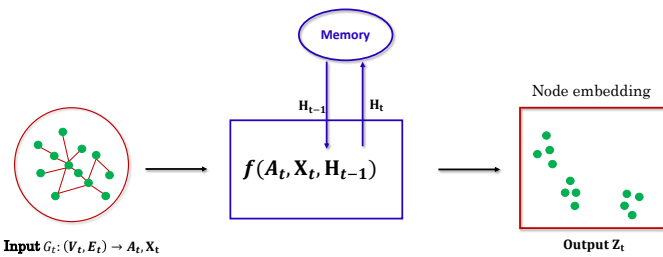


Fig. 2. Basic learning paradigm for dynamic graph embedding. From current  $V_t$  and  $E_t$ , we can extract the adjacency matrix  $A_t$  and then feed it into nonlinear function  $f$  together with node feature  $X_t$ . To capture long-distance dependency, a memory cell is added to utilize historical state  $H_{t-1}$  and preserve current hidden state  $H_t$ . The output node embedding  $Z_t$  is expected to jointly acquire topological information and temporal-evolving patterns.

to learn a low-dimension representation  $Z_t \in \mathbb{R}^{n \times d_o}$ , where  $d_h$  and  $d_o$  is the dimension of hidden state  $H_{t-1}$ , and output node embedding  $Z_t$ , respectively. The learning paradigm can be formalized as:

$$Z_t, H_t = f(A_t, X_t, H_{t-1}). \quad (1)$$

Here,  $f$  can be any nonlinear function. According to different temporal regularizers, we divide current prevalent methods into two fashions: recurrent learning and temporal-smoothness learning. In recurrent learning,  $f$  is instantiated as a recurrent network, e.g., RNN, GRU and LSTM [28], [29], where the transformations inside it are replaced by graph convolutions. The output node embedding  $Z_t$  is expected to jointly acquire topological information and temporal-evolving patterns. Beyond that, considering the stability and continuity of the node embedding in two consecutive times, some works [19], [30], [24] either (1) use historical state  $H_{t-1}$  to initialize  $X_t$  directly; (2) add temporal regularizer [23] additionally; or (3) make alignment between two consecutive time steps [31], which are termed as temporal-smoothness learning in this paper. Temporal-smoothness learning can enhance nodes inter-dependency, and also facilitate the acquisition of robust and stable embeddings. These two learning fashions are typical methods in the current research field.

### B. Graph Convolutional Layer

In this part, we mainly introduce how to learn the structural representation of the graph in snapshot  $t$ , so we temporarily ignore the subscript  $t$  for clarity. At time step  $t$ , given a snapshot  $G = (V, E)$  and we denote  $A, D$  as the adjacency matrix and degree matrix, respectively. By applying a two-layer GCN [4], [32], we obtain the output structural node embeddings:

$$Z = \tilde{A} \text{ReLU}(\tilde{A} X W_1) W_2, \quad (2)$$

where  $\tilde{A} = \tilde{D}^{-\frac{1}{2}} \hat{A} \tilde{D}^{-\frac{1}{2}}$  is a symmetrically normalized adjacency matrix,  $\hat{A} = A + I$  and  $\tilde{D} = D + I$  are the augmented adjacency and degree matrices with added self-loops, respectively.  $W_1$  and  $W_2$  are the learnable matrices in GCN. Graph convolution can be interpreted as message fusion within connected nodes.

In the analysis of GCN's working mechanism, Li et al. [5] show that a graph convolution is a special form of Laplacian smoothing. This smoothing technique can force the representation of similar node pairs to be more tight. Therefore, GCN is able to obtain a pleasant performance in many tasks. Though, we also need to be aware that the above smoothing technique can push cannot-link node pairs to get together, especially when stacking graph convolution step by step or layer by layer. In this case, the node representation would become too smooth to be distinguished with each other, leading oversmoothing phenomenon.

By decomposing GCN, we can find that the smoothing mainly lies in feature shrinking properties. In dynamic graph embedding, to obtain robust and impressive performance, stacking multiple layers implicitly or explicitly is inevitable which would increase feature shrink gradually and potentially causes oversmoothing problem. In the following section, we will start with analyzing feature shrink.

### C. Feature Shrink and Oversmoothing

We begin with easily splitting one-step graph convolution operation into two basic steps: (1) aggregation step; (2) transformation step. The message passing in GCN [4] for a center node  $v_i$  is formulated as:

$$z_i = \sigma \left( \sum_{v_j \in N(v_i)} \tilde{a}_{ij} x_j W \right), \quad (3)$$

where  $\tilde{a}_{ij} = a_{ij}/d_i$  is a normalized weight between node  $v_j$  and  $v_i$ ,  $d_i$  is the degree of node  $v_i$ . This rule can be decomposed into two steps:

(1) Aggregation step. Aggregating message from the neighbors,  $N(v_i)$  of node  $v_i$ , where we treat each node is the neighbor of itself as well, and thus we have:

$$\tilde{x}_i = \sum_{v_j \in N(v_i)} \tilde{a}_{ij} x_j. \quad (4)$$

(2) Transformation step. Transforming the aggregated state  $\tilde{x}_i$  into a new space with a learnable matrix  $W$  and nonlinear activation function,

$$\tilde{z}_i = \sigma(\tilde{x}_i W). \quad (5)$$

Then it can be shown that the overall distance of node embedding is reduced in aggregation step, see Theorem 1.

**Theorem 1** [33] *Let the overall distance of node embeddings  $X$  be  $D(\tilde{X}) = \frac{1}{2} \sum_{v_i, v_j} \tilde{a}_{i,j} \|x_i - x_j\|_2^2$ . Then we have*

$$D(\tilde{X}) \leq D(X). \quad (6)$$

Theorem 1 indicates that after aggregation, the distance between the overall connected nodes will converge in adjacent areas. While, at the same time, the unconnected nodes (in the same connected component) will also shrink together due to their common connected nodes. We define the similarities of the sum of any two nodes within a cannot-link node pair as ‘‘negative smoothness’’ and denote the total similarities

of any two nodes within a must-link node pair as “positive smoothness”. Obviously, positive smoothness brings profits while the negative smoothness is detrimental for the final node embeddings. According to the Theorem 1, we have the following Corollary 1 in dynamic graph embedding setting.

**Corollary 1.** *In dynamic graph setting, we use  $\tilde{\mathbf{H}}_t$  and  $\mathbf{H}_t$  to denote two cases: applying graph convolution to  $\mathbf{X}_t$ ; not applying graph convolution to  $\mathbf{X}_t$ , respectively. Similarly, let the overall distance of hidden state at time step  $t$  be  $D(\tilde{\mathbf{H}}_t) = \frac{1}{2} \sum_{v_t^i, v_t^j} \tilde{a}_{t,i,j} \|\mathbf{h}_t^i - \mathbf{h}_t^j\|_2^2$ . Then we have:*

$$D(\tilde{\mathbf{H}}_t) \leq D(\mathbf{H}_t), \quad (7)$$

and

$$\sum_t D(\tilde{\mathbf{H}}_t) \leq \sum_t D(\mathbf{H}_t). \quad (8)$$

*Proof.* See Appendix B.

The above corollary reveals that using graph convolutions in dynamic graph embedding will make the embedding space shrink in comparison with no graph convolution operation, and the shrink will increase along with time steps.

#### D. Proposed Method

Here, we follow the basic assumptions for dynamic graph embedding in [23] that (1) nodes change their hidden representations gradually over time; (2) the representation of two nodes are more similar to each other when they interact frequently than two faraway nodes, thus we have the following optimization target:

$$\min \sum_{t=1}^T \sum_{i \in \mathcal{V}_t} \|\mathbf{z}_t^i - \mathbf{z}_{t-1}^i\|_2^2 + \sum_{(i,j) \in E} \|\mathbf{z}_t^i - \mathbf{z}_t^j\|_2^2 - \lambda \sum_{(i,j) \notin E} \|\mathbf{z}_t^i - \mathbf{z}_t^j\|_2^2. \quad (9)$$

For the first term, we actually do not need to add external regularizer if we adopt temporal-smoothness or recurrent learning fashion since they have already played the role of temporal regularization; For the second term, in view of the fact that we perform at least one aggregation operation at each time step, and then the distance of connected node pairs will reduce. In other words, the second term will decrease in the condition of using GCN. Then, according to Corollary 1, the overall distance of unconnected node pairs will also decrease at the same time. Therefore, the key point is to add a constraint to the third term (e.g., increase the distance between nodes in unconnected node pairs at time step  $t$  or keep it unchanged). Here, we adopt the method that is adding L2 feature normalization to push nodes on the unit hypersphere, maintaining the overall distance of  $D(\mathbf{Z}_t)$  unchanged and also building connections with cosine similarity, which can be formulated as:

$$D(\mathbf{Z}_t) = \sum_{(i,j) \in E} \|\mathbf{z}_t^i - \mathbf{z}_t^j\|_2^2 + \sum_{(i,j) \notin E} \|\mathbf{z}_t^i - \mathbf{z}_t^j\|_2^2. \quad (10)$$

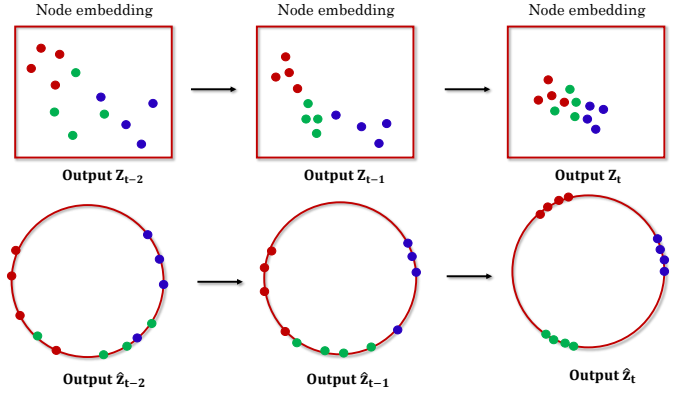


Fig. 3. Illustration of the main idea for L2 feature normalization. Each sub-figure illustrates the node embeddings of the corresponding time step. The upper three sub-figures demonstrate that the unconstrained node embeddings, including all nodes, are getting closer and closer along with time steps. The lower three sub-figures show that the node embeddings with the proposed normalization constraint makes connected (or similar) nodes gather in the adjacent area while pushes unconnected (or dissimilar) nodes far away.

As mentioned above, the value of the first term in Equation (10) is reduced after aggregation in graph convolution and thus the second term in Equation (10) (or third term without negative sign in Equation (9)) would be increased on condition that we keep the overall distance  $D(\mathbf{Z}_t)$  unchanged.

Then, to keep the overall distance of node embedding unchanged, we propose to utilize L2 feature normalization, constraining the node embeddings to a unit sphere. The detailed process of our algorithm is as follows: (1) deploy a proper number (generally two or three) of graph convolutional layers, getting the raw node embeddings; (2) center the node embeddings (the reason is described in the following part); (3) place L2 normalization onto the embedding matrix so that it can be pushed to a high dimensional unit sphere, which is illustrated in Figure 3; (4) feed the normalized node embeddings to next time step. Empirically, the centering step has little effect on the performance. It is worth mentioning that either using the node embeddings as the historical hidden state (recurrent learning), or as the initial value of the next input (temporal-smoothness learning), we could always achieve the above target. In this way, the must-link or similar nodes are still embedded in adjacent areas while the cannot-link or dissimilar nodes would be pushed apart.

Except for preventing all features shrinking together and being too smooth, more importantly, using L2 feature normalization can establish a direct connection between node similarity and Euclidean distance as well. By L2 feature normalization, we have the following derivation:

$$\begin{aligned} \|\mathbf{z}_t^i - \mathbf{z}_t^j\|_2 &= \sqrt{\|\mathbf{z}_t^i\|_2^2 + \|\mathbf{z}_t^j\|_2^2 - 2(\mathbf{z}_t^i)^T \mathbf{z}_t^j} \\ &= \sqrt{2 - 2 \langle \mathbf{z}_t^i, \mathbf{z}_t^j \rangle}, \end{aligned} \quad (11)$$

where the  $\langle \mathbf{z}_t^i, \mathbf{z}_t^j \rangle$  is the inner product of  $\mathbf{z}_t^i$  and  $\mathbf{z}_t^j$ , which denotes the similarity of node  $i$  and  $j$ , and  $\mathbf{z}_t^i$  is normalized node representation,  $\mathbf{z}_t^i \leftarrow \frac{\mathbf{z}_t^i}{\|\mathbf{z}_t^i\|_2}$ . The above derivation

demonstrates two points with large similarity are embedded close to each other while two points associated with a small similarity are far away from each other. In the following, we will illustrate details about how L2 feature normalization with centering keeps the distance of  $n$  nodes in  $D(\mathbf{Z}_t)$  unchanged.

$$\begin{aligned}
 D(\mathbf{Z}_t) &= \sum_{(i,j) \in V} \|\mathbf{z}_t^i - \mathbf{z}_t^j\|_2^2 = \sum_{(i,j) \in V} (1 + 1 - 2(\mathbf{z}_t^i)^T \mathbf{z}_t^j) \\
 &= 2n^2 - 2\mathbf{1}^T \mathbf{Z}_t \mathbf{Z}_t^T \mathbf{1} = 2n^2 - 2\|\mathbf{1}^T \mathbf{Z}_t\|_2^2 \\
 &= 2n^2 - \left\| \frac{1}{n} \sum_{i=1}^n \mathbf{z}_t^i \right\|_2^2.
 \end{aligned} \tag{12}$$

Then, by centering  $\mathbf{z}_t^i$ ,

$$\mathbf{z}_t^i \leftarrow \mathbf{z}_t^i - \frac{1}{n} \sum_{i=1}^n \mathbf{z}_t^i, \tag{13}$$

we can eliminate the second term in Equation (12), which is trivial since shifting would not change the distribution, so the final total distance is  $2n^2$ . Though  $n$  is non-decreasing along time steps, the total distance within  $n$  nodes is consistently  $2n^2$  by the derivation of Equation (12), which means that new increasing nodes will not change the original  $n$  nodes' distance. To summary, the proposed method has two steps: (1) to center all normalized node features; and (2) to normalize each node feature in the last layer of each time step.

### E. Computational Complexity

The proposed normalization contains two parts: (1) node centralization, which take  $O(|\mathcal{V}_t|)$  at each time step; (2) L2 feature normalization, which takes  $O(d|\mathcal{V}_t|)$  at each time step. Therefore, the total complexity is  $O(Tn)$  (assuming the number of new appeared nodes  $m \ll n$ ).

### F. Basic Model and Loss Function

As mentioned in III-A, there are two typical ways to learn the time-evolving patterns in dynamic graph embedding. One is temporal-smoothness learning and the other is recurrent learning. In general, the former method is stable and robust while the latter is more powerful to capture long-distance dependencies. Certainly, two methods can be integrated as proposed in [34]. Following the works in [24], [19], [35], we extract two prototypes to verify the effectiveness of our proposed method as shown in Figure 4 and Figure 5, which is also designed to exclude the influence of other factors and independently evaluate the effectiveness of the proposed method.

For clarity, we term these two frameworks as DynGCN and GRUGCN, respectively. In each snapshot  $G_t(V_t, E_t)$ , the input feature matrix  $\mathbf{X}_t$  contains two parts: one is nodes that have ever emerged in the history, and the other is the new appear nodes. For the first part, we follow the basic idea in [24], [19], [35], that is inheriting the historical embedding results [24] as the next step input. For the second new part, we use the golort initialization method [36] to initialize nodes value. Then each GCN block is a typical two-layer

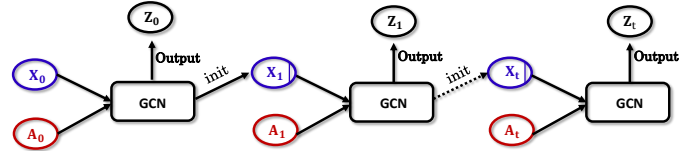


Fig. 4. Illustration of temporal-smoothness learning, one typical temporal-evolving learning fashion, where the previous node hidden state is used to initialize the next-step state.

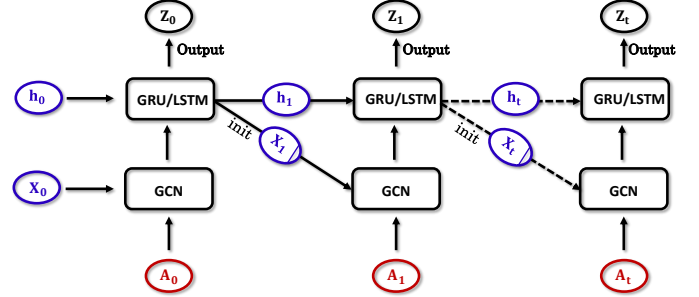


Fig. 5. Illustration of recurrent learning, the other typical temporal-evolving learning fashion, where additional recurrent modules are added to capture long-distance dependencies, and the previous node hidden state is used as next node initial value.

graph convolution which is parameter-sharing across time steps, with adding dropout between two graph convolutional layers. In Figure 5, an additional recurrent block is added to enhance long-distance learning, which is the prototype of models in [19], [35]. These two frameworks are widely used in dynamic graph embedding, currently.

To predict the occurrence of nodes appearing in the local neighborhood around  $v_i$  at  $t$ , we use a binary cross-entropy loss function to push nodes co-occurring to have similar representation and restrain nodes not-occurring to have different representation. The loss function is:

$$\begin{aligned}
 l_t &= \sum_t \left( \sum_{(v_i, v_j) \in E_t} -\log(\sigma(\langle \mathbf{z}_t^i, \mathbf{z}_t^j \rangle)) \right. \\
 &\quad \left. - \lambda \cdot \sum_{(v'_i, v'_j) \notin E_t} \log(1 - \sigma(\langle \mathbf{z}_t^{i'}, \mathbf{z}_t^{j'} \rangle)) \right), \tag{14}
 \end{aligned}$$

where  $\sigma$  is the sigmoid function and  $\{(v'_i, v'_j) \notin E_t\}$  is a negative set sampled with negative sample strategy, and  $\lambda$  is a hyperparameter. Here, we only consider negative samples that have never appeared, that is, links that appeared before time  $t$  are assumed positive links, so we avoid sampling such node pairs.

## IV. EXPERIMENTS

### A. Dataset

There are numerous dynamic networks in the real world, we choose four real-world datasets that closely represent the dynamic graphs in most cases, which are publicly available:

TABLE I  
DATA SUMMARY AND SLICED TIME STEPS

Dataset	DNC	UCI	FBW	ENRON
#Nodes	1,891	1,899	46,952	87,273
#Links	39,264	59,835	876,993	1,148,072
#Sliced edges	~2,000	~3,000	~50,000	~50,000
#Total time steps	12	12	18	23
#Train steps	1-9	1-9	1-14	1-17
#Test steps	10-12	10-12	15-18	18-22

DNC<sup>1</sup> and Enron<sup>2</sup> email networks, UCI<sup>3</sup> community networks and Facebook wall posts<sup>4</sup>(FBW) communication networks.

**DNC** and **Enron** are two email communication networks that come from the Enron company and the 2016 Democratic National Committee email leak event, respectively. In DNC, a node corresponds to a person and the link denotes that a person has sent an email to another person. In Enron, a node is the employee, and links are emails connections between employees. **UCI** contains sent messages between the users of an online community of students from the University of California, Irvine. A node represents a user, and the edge denotes a sent message. Multiple edges indicate multiple connections. **FBW** is a subset of posts from some users to other walls on Facebook. The nodes are the Facebook users, and an edge represents one post, linking the users writing a post to the users whose wall is written on. There also exist multiple edges pointing to a single user. Moreover, one user can be written on their walls, so we keep these links.

There are different time spans of the four datasets, we slice each dataset into different time steps according to a roughly same number of links along the timeline. For the reason that keeping the links appeared on the same day in a time interval, the number of nodes in each time step is slightly different. We summary the dataset and the sliced time steps in TABLE I.

### B. Comparison Models

Here we compare a variety of competing unsupervised node embedding methods. First, we compare the typical static graph embedding models: GCN [32] which consists of two-layer graph convolution, using an encoder-decoder framework to learn node embeddings; GAT [16] is a variation of GCN where each graph convolution is replaced with attention mechanism; SAGE [15] is an inductive framework that combine node attribute information to learn representations based on previously unseen graph data. Then, we utilize the basic frameworks DynGCN and GRUGCN mentioned above, to verify our proposed normalization method. Furthermore, we compare against the latest released work EvolveGCN [18], which has two versions EGCNO and EGCNH. EvolveGCN uses RNN to evolve GCN parameters, aiming at preserving temporal patterns in its weights.

<sup>1</sup><http://konect.cc/networks/dnc-temporalGraph/>

<sup>2</sup><http://www.cs.cmu.edu/enron/>

<sup>3</sup><http://konect.cc/networks/opsahl-ucsocial>

<sup>4</sup><http://konect.cc/networks/facebook-wosn-wall>

### C. Evaluation Tasks and Metric

In dynamic graph embedding, link prediction is one of the most common methods used to evaluate the quality of node embedding. Dynamic link prediction is defined as given  $G = \{G_1, \dots, G_t\}$ , to predict links in the future  $m$  snapshots  $\{G_{t+1}, G_{t+2}, \dots, G_{t+m}\}$ , including ever appeared links (transductive learning) and new links (inductive learning). The links that appeared in  $\{G_{t+1}, G_{t+2}, \dots, G_{t+m}\}$  are considered as positive samples and node pairs without links are as negative samples. Then the models are evaluated by correctly classifying positive and negative links. We sample an equal amount of negative node pairs and compute the average precision (termed as AP) and area under the ROC curve (termed as AUC) scores as the metric.

In addition, we define the ratio of negative-to-positive smoothness as an auxiliary metric to evaluate the model. The target of node embedding is to make similar nodes cluster together and embed in the adjacent area while push nodes of no similarity are far away from each other. Therefore, the ratio of negative-to-positive smoothness can well evaluate the final embeddings. In addition, the ratio of negative-to-positive smoothness shows how much the negative effects of Laplacian smoothing in graph convolution-based models brings, which is defined as:

$$R_{neg} = \frac{n_s}{p_s}, \quad (15)$$

where  $p_s = \sum_{(i,j) \in E_t} \mathbf{z}_i^i \mathbf{z}_t^j$  and  $n_s = \sum_{(i,j) \notin E_t} \mathbf{z}_i^i \mathbf{z}_t^j$ . Since the number of unconnected nodes is huge, we randomly sample an equal number of negative samples and average their similarities.

### D. Experimental Setup

We roughly split 80% graph snapshots for training and the rest 20% snapshots for testing, and the detail is shown in Table I. If no specification, we set the same experimental setting for each model. During training, we utilize Adam SGD optimizer [37] with weight decay and the  $\lambda$  value is  $5 \times 10^{-7}$  to optimize the loss function. At each time step, we add dropout [38] before each graph convolutional layer, and the dropout rate is 0.25. To fully utilize the time information, we do not split any validation set to tune the parameters. Instead, we select the model based on the loss value during training. The training epoch is fixed as 100. For new appear nodes in the time step  $t$ , the initial value is initialized with dimension  $n \times 32$  ( $n = |\mathcal{V}_t|$ ) if there is no feature available. The dimension of the middle and the output layer is set to 32, which is to reduce the effort of hyperparameter tuning. Finally, we randomly sample 5 different seeds and repeat the experiment. Our code is publicly available <https://github.com/marlin-codes/FeatureNorm>

### E. Experimental Results

Table II summarizes the averaged AUC scores and AP scores across all test snapshots on four datasets. From the results, we have the following observations: (1) Being equipped with the proposed normalization method, the baseline models, including DynGCN, GRUGCN, EGCNO, and EGCNH, are

TABLE II  
AVERAGED AUC SCORES AND AP SCORES ACROSS ALL TEST SNAPSHOTS

	DNC		UCI		FBW		Enron	
	AUC	AP	AUC	AP	AUC	AP	AUC	AP
GCN	91.79 ± 0.47	92.20 ± 0.45	73.01 ± 0.66	67.06 ± 0.53	71.92 ± 0.40	69.52 ± 0.69	88.19 ± 0.29	90.79 ± 0.17
GAT	89.48 ± 0.82	91.93 ± 0.75	74.86 ± 1.62	75.35 ± 1.70	75.31 ± 0.47	76.86 ± 0.66	86.81 ± 1.04	89.41 ± 0.63
SAGE	86.10 ± 2.61	85.90 ± 2.61	69.15 ± 1.34	65.63 ± 1.34	58.27 ± 0.55	53.54 ± 0.55	69.12 ± 1.18	61.60 ± 1.18
DynGCN	86.30 ± 1.47	86.98 ± 1.28	73.01 ± 1.83	67.48 ± 2.46	70.14 ± 1.06	70.84 ± 1.34	82.38 ± 1.02	85.38 ± 0.80
<b>DynGCN+FN</b>	<b>91.42 ± 0.30</b>	<b>93.39 ± 0.44</b>	<b>76.42 ± 0.58</b>	<b>74.08 ± 0.83</b>	<b>79.89 ± 0.71</b>	<b>81.25 ± 0.44</b>	<b>85.85 ± 0.61</b>	<b>89.09 ± 0.39</b>
GRUGCN	<b>94.12 ± 1.94</b>	<b>95.11 ± 0.95</b>	72.57 ± 1.47	68.57 ± 4.20	71.41 ± 2.40	73.25 ± 2.35	84.17 ± 2.87	88.60 ± 1.38
<b>GRUGCN+FN</b>	93.02 ± 0.62	94.86 ± 0.42	<b>75.41 ± 2.10</b>	<b>74.87 ± 1.89</b>	<b>79.44 ± 0.39</b>	<b>82.58 ± 0.35</b>	<b>87.88 ± 0.44</b>	<b>90.75 ± 0.35</b>
EGCO	87.43 ± 0.60	88.33 ± 0.63	<b>74.16 ± 3.78</b>	70.69 ± 5.10	66.46 ± 0.36	63.71 ± 0.77	81.62 ± 0.32	79.68 ± 0.44
<b>EGCO+FN</b>	<b>92.65 ± 0.64</b>	<b>94.74 ± 0.42</b>	73.70 ± 2.41	<b>75.25 ± 1.80</b>	<b>72.30 ± 0.38</b>	<b>76.09 ± 0.57</b>	<b>87.72 ± 0.54</b>	<b>90.95 ± 0.35</b>
EGCH	89.27 ± 0.41	90.72 ± 0.49	<b>78.81 ± 1.95</b>	<b>77.16 ± 0.83</b>	69.65 ± 0.31	69.48 ± 0.49	82.06 ± 0.21	80.03 ± 0.14
<b>EGCH+FN</b>	<b>92.78 ± 0.31</b>	<b>95.08 ± 0.13</b>	74.65 ± 2.91	76.41 ± 1.73	<b>74.34 ± 0.44</b>	<b>78.64 ± 0.21</b>	<b>87.72 ± 0.51</b>	<b>91.35 ± 0.32</b>

\* The best results of the proposed L2 feature normalization (+FN) and basic dynamic graph models are bold.

all improved obviously in most cases; (2) In comparison with the static graph embedding, the dynamic graph models obtain higher AUC and AP scores, indicating the informativeness of temporal-evolving patterns; (3) Some static graph models outperform dynamic embedding in several non-constraint cases (e.g., AUC scores are from GCN and DynGCN performing on DNC/UCI/FBW/ENRON datasets) while being defeated or being improved much by our proposed method (+FN), demonstrating that the damage of incremental feature shrink and the constructiveness of our proposed normalization; (4) Being deployed with L2 feature normalization, taking the performance of DynGCN on the four datasets as an example, the proposed models achieve 3%~10% improvements; (5) The L2 normalization can also benefit robustness, where the evidence can be found by comparing the standard variance of baseline model with the normalized version, which further demonstrates the superiority of the proposed normalization.

Table III shows the ratio of negative smoothness to positive smoothness. The results contain two parts, one is the averaged ratio across all training time spans, which reflects the whole node embedding quality. The other part is the final embedding representation which is the latest one used for evaluation. From the results shown in Table III, we discover that after applying L2 feature normalization, the ratio of negative smoothness to positive smoothness is dramatically reduced in most cases. Certainly, the decreasing negative smoothness means that the ratio of positive smoothness is increasing. Yet, we also find in some cases, taking performance of ECGNH on UCI dataset as an example where the ratio is not reduced after the normalization but the interesting thing is the corresponding performance (in Table II) is also not improved. Thus, we further confirm that feature shrink is part of the reason for relatively low performance while our proposed method can unleash models' true capacity.

## V. ANALYSIS AND DISCUSSION

Although some parameters like learning rate, dropout rate, embedding dimension can affect the final performance, they actually show consistent performance with the same environ-

mental setting which is not the main concern in our paper. Here, we care more about the performance of long-distance prediction and the difference with the related normalization method.

### A. Long Distance Learning

Long-distance prediction is essential for some practical scenarios, which can also indicate the embedding qualities include stability and robustness. In the last section, we used 80% and 20% of the dataset for training and test, respectively. Here we further extend the steps of the test data, splitting 60% and 40% for model learning and evaluation, respectively. The results are shown in Figure 6 including AUC scores and AP scores. Based on the results, we have the following findings: (1) There is a descending tendency in the performance of AUC and AP along with time steps, which indicates the dependencies of time-evolving patterns matters; (2) Adding the proposed normalization benefits the embeddings from two aspects: one is to obtain a better or comparable performance and the other is long durability since models with the proposed method are able to obtain high AUC and AP scores as well as time goes by.

### B. Difference with Related Works

The underlying principle to constrain the overall distance unchanged of the proposed method is similar with PairNorm [6]. So it is necessary to point out the difference and make an experimental comparison.

From the task aspect, PairNorm is designed for deeper graph neural networks, evaluated on the node classification task while we care more for similarity for node embeddings and dynamic link predictions. The difference will result in distinct designs for the corresponding settings and tasks. From the model design aspect, PairNorm is proposed to keep the total pairwise distance unchanged in the background of a static graph. However, in a dynamic setting, we need to consider that the number of nodes in each time can be different. From the implementation aspect, PairNorm adds the constraint between

TABLE III  
THE RATIO OF NEGATIVE SMOOTHNESS-TO-POSITIVE SMOOTHNESS

	DNC		UCI		FBW		Enron	
	Average $R_{neg}$	Latest $R_{neg}$	Average $R_{neg}$	Latest $R_{neg}$	Average $R_{neg}$	Latest $R_{neg}$	Average $R_{neg}$	Latest $R_{neg}$
DynGCN	0.4591	0.4667	0.2265	0.1601	0.1527	0.1018	0.4198	0.1820
<b>DynGCN+FN</b>	<b>0.0277</b>	<b>0.0123</b>	<b>0.0766</b>	<b>0.0115</b>	<b>0.0069</b>	<b>0.0025</b>	<b>0.0346</b>	<b>0.0291</b>
GRUGCN	0.9444	0.9255	0.1157	0.0685	0.0935	0.1050	0.2026	0.3844
<b>GRUGCN+FN</b>	<b>0.0138</b>	<b>0.0246</b>	<b>0.0296</b>	<b>0.0094</b>	<b>0.0031</b>	<b>0.0029</b>	<b>0.0397</b>	<b>0.0119</b>
EGCO	0.0183	0.0146	<b>0.0150</b>	0.0162	0.0388	0.0132	0.1454	0.083
<b>EGCO+FN</b>	<b>0.0058</b>	<b>0.0006</b>	0.0181	<b>0.0143</b>	<b>0.0019</b>	<b>0.0010</b>	<b>0.0098</b>	<b>0.0081</b>
EGCH	0.0329	0.0270	0.0139	0.0198	0.0344	0.0254	0.1685	0.1361
<b>EGCH+FN</b>	<b>0.0064</b>	<b>0.0070</b>	<b>0.0153</b>	<b>0.0090</b>	<b>0.0016</b>	<b>0.0007</b>	<b>0.0088</b>	<b>0.0047</b>

\* The best results of the proposed L2 feature normalization (+FN) and basic dynamic graph models are bold.

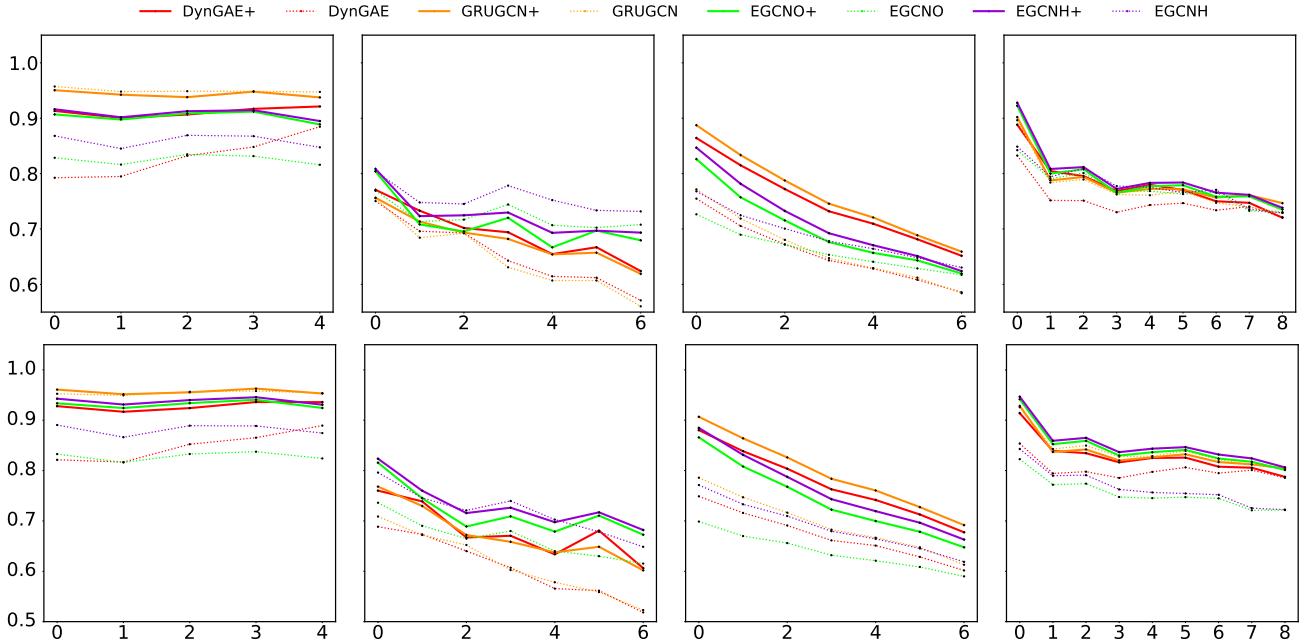


Fig. 6. AUC (the upper four sub-figures) and AP (the lower four sub-figures) scores for long time-step prediction. From left to right, the corresponding dataset is DNC, UCI, FBW and Enron.

graph convolutional layer<sup>5</sup>. Then, the following operations, including softmax and classifier, will change the total distance and similarities. However, our normalization is assigned to the final layer of graph convolutions in each time step which pushes the embeddings on the unit hypersphere and the total distance keeps unchanged. In addition, the proposal also builds a connection between cosine similarity and embedding distance. In the end, we impose PairNorm (PN) and PairNorm-SI (PN-SI) mentioned in [6] to dynamic graph models for an experimental comparison.

According to the AUC and AP scores shown in Table IV, we can conclude that PairNorm does not work in the task of dynamic graph embedding, and even hinders the performance of the model. However, our proposed normalization method can significantly improve the AUC and AP scores of the original model.

## VI. CONCLUSIONS

In this paper, we inspect the feature shrinking and over-smoothing problem in dynamic graph embedding. First, we analyze two basic operations in graph convolution, and then come to two typical learning frameworks in dynamic graph embedding: temporal-smoothness learning and recurrent learning. By theoretical analysis on overall embedding distance, we discover nodes including must-link pairs and cannot-link pairs would get closer if there is no constraint applied to graph convolution. Actually, most dynamic graph embedding models with graph convolution exist such a problem, but seldom works notice it. In this work, we propose L2 feature normalization to constrain the overall distance of pairwise nodes at time step  $t$  unchanged, which can be also regarded as to constrain overall similarity unchanged. In this way, the positive smoothness will be further improved while negative smoothness can be reduced. In addition, the proposal can be

<sup>5</sup><https://github.com/LingxiaoShawn/PairNorm/blob/master/models.py>



TABLE IV  
AVERAGED AUC SCORES AND AP SCORES WITH DIFFERENT NORMALIZATION METHODS

	DNC		UCI		FBW		Enron	
	AUC	AP	AUC	AP	AUC	AP	AUC	AP
DynGCN	86.30 ± 1.47	86.98 ± 1.28	73.01 ± 1.83	67.48 ± 2.46	70.14 ± 1.06	70.84 ± 1.34	82.38 ± 1.02	85.38 ± 0.80
DynGCN+PN	82.35 ± 2.04	83.63 ± 1.71	68.99 ± 4.35	59.07 ± 4.25	63.35 ± 4.22	63.87 ± 3.12	84.66 ± 2.43	86.80 ± 2.32
DynGCN+PN-SI	85.16 ± 1.27	85.17 ± 1.32	69.46 ± 4.00	60.17 ± 3.83	62.73 ± 4.37	63.12 ± 3.34	83.18 ± 4.50	86.16 ± 3.21
<b>DynGCN+FN</b>	<b>91.42 ± 0.30</b>	<b>93.39 ± 0.44</b>	<b>76.42 ± 0.58</b>	<b>74.08 ± 0.83</b>	<b>79.89 ± 0.71</b>	<b>81.25 ± 0.44</b>	<b>85.85 ± 0.61</b>	<b>89.09 ± 0.39</b>
GRUGCN	94.12 ± 1.94	95.11 ± 0.95	72.57 ± 1.47	68.57 ± 4.20	71.41 ± 2.40	73.25 ± 2.35	84.17 ± 2.87	88.60 ± 1.38
GRUGCN+PN	91.55 ± 2.10	93.45 ± 1.30	74.48 ± 4.18	70.22 ± 4.80	73.73 ± 4.84	76.99 ± 4.53	84.37 ± 4.07	86.85 ± 3.57
GRUGCN+PN-SI	89.93 ± 2.89	92.66 ± 1.68	73.64 ± 4.34	69.79 ± 5.66	74.18 ± 4.56	76.63 ± 4.09	84.26 ± 4.16	86.39 ± 3.55
<b>GRUGCN+FN</b>	<b>93.02 ± 0.62</b>	<b>94.86 ± 0.42</b>	<b>75.41 ± 2.10</b>	<b>74.87 ± 1.89</b>	<b>79.44 ± 0.39</b>	<b>82.58 ± 0.35</b>	<b>87.88 ± 0.44</b>	<b>90.75 ± 0.35</b>
EGCO	87.43 ± 0.60	88.33 ± 0.63	<b>74.16 ± 3.78</b>	70.69 ± 5.10	66.46 ± 0.36	63.71 ± 0.77	81.62 ± 0.32	79.68 ± 0.44
EGCO+PN	77.95 ± 2.79	73.28 ± 4.19	64.14 ± 3.21	56.06 ± 2.90	63.59 ± 1.48	59.06 ± 1.21	75.35 ± 2.30	69.98 ± 1.84
EGCO+PN-SI	78.59 ± 2.79	74.32 ± 4.18	65.35 ± 3.31	56.68 ± 2.90	63.88 ± 1.41	59.25 ± 1.12	75.70 ± 2.39	70.43 ± 2.05
<b>EGCO+FN</b>	<b>92.65 ± 0.64</b>	<b>94.74 ± 0.42</b>	73.70 ± 2.41	<b>75.25 ± 1.80</b>	<b>72.30 ± 0.38</b>	<b>76.09 ± 0.57</b>	<b>87.72 ± 0.54</b>	<b>90.95 ± 0.35</b>
EGCH	89.27 ± 0.41	90.72 ± 0.49	<b>78.81 ± 1.95</b>	<b>77.16 ± 0.83</b>	69.65 ± 0.31	69.48 ± 0.49	82.06 ± 0.21	80.03 ± 0.14
EGCH+PN	83.64 ± 2.28	83.35 ± 3.69	68.20 ± 4.47	63.28 ± 3.32	65.75 ± 2.25	62.75 ± 2.03	80.19 ± 2.43	78.20 ± 2.67
EGCN+PN-SI	83.79 ± 2.35	83.68 ± 3.61	68.99 ± 4.85	63.92 ± 3.64	66.32 ± 2.32	63.75 ± 2.40	79.85 ± 2.52	77.38 ± 2.85
<b>EGCH+FN</b>	<b>92.78 ± 0.31</b>	<b>95.08 ± 0.13</b>	74.65 ± 2.91	76.41 ± 1.73	<b>74.34 ± 0.44</b>	<b>78.64 ± 0.21</b>	<b>87.72 ± 0.51</b>	<b>91.35 ± 0.32</b>

\* The best results of each model with different constraints are bold.

regarded as a plug-in module that is easy to extend related dynamic graph embedding models.

#### ACKNOWLEDGEMENTS

The work described in this paper was partially supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (CUHK 2410021, Research Impact Fund, No. R5034-18). We would like to thank the anonymous reviewers for their comments. Besides, we gratefully thank Xue Li for his helpful feedback and discussions.

#### APPENDIX

For simplicity and clarity, we assume the dimension of node feature is one and it can easily extend multidimensional case since  $D(\mathbf{X})/D(\tilde{\mathbf{X}})$  or  $D(\mathbf{H}_t)/D(\tilde{\mathbf{H}}_t)$  can be decomposed into the sum of one-dimension case. In the following, we use the corresponding lowercase in case of confusion.

##### A. Proof of Theorem 1

Here we rewrite the proof in line with the following corollary 1 for better understanding feature shrink in dynamic graph embedding.

**Theorem 1** [33] *Let the overall distance of node embeddings be  $D(\tilde{\mathbf{x}}) = \frac{1}{2} \sum_{v_i, v_j} \tilde{a}_{i,j} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$ . Then we have*

$$D(\tilde{\mathbf{x}}) \leq D(\mathbf{x}). \quad (16)$$

To prove the Theorem 1, two lemmas related to gradient of  $D(\mathbf{x})$  and Hessian matrix of  $D(\mathbf{x})$  will be introduced at first.

**Lemma 1** *Gradient of  $D(\mathbf{x})$  satisfies:  $\tilde{\mathbf{x}}_i - \frac{\partial D(\mathbf{x})}{\partial \mathbf{x}_i} = \mathbf{x}_i$*   
*Proof:*

$$\begin{aligned} \tilde{\mathbf{x}}_i - \frac{\partial D(\mathbf{x})}{\partial \mathbf{x}_i} &= \mathbf{x}_i - \sum_{v_j \in \mathcal{N}(v_i)} \tilde{a}_{ij} (\mathbf{x}_i - \mathbf{x}_j) \\ &= \sum_{v_j \in \mathcal{N}(v_i)} \tilde{a}_{ij} \mathbf{x}_j \\ &= \mathbf{x}_i. \end{aligned} \quad (17)$$

**Lemma 2** *Hessian matrix of  $D(\mathbf{x})$  satisfies:  $\nabla^2 D(\mathbf{x}) \preceq 2I$*   
*Proof:* Hessian matrix of  $D(\mathbf{x}) = \frac{1}{2} \sum_{v_i, v_j} \tilde{a}_{ij} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$  can be written as

$$\begin{aligned} \nabla^2 D(\mathbf{x}) &= \begin{bmatrix} 1 - \tilde{a}_{11} & -\tilde{a}_{12} & \cdots & -\tilde{a}_{1n} \\ -\tilde{a}_{21} & 1 - \tilde{a}_{22} & \cdots & -\tilde{a}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\tilde{a}_{n1} & -\tilde{a}_{n2} & \cdots & 1 - \tilde{a}_{nn} \end{bmatrix} \\ &= I - D^{-1}A, \end{aligned} \quad (18)$$

and we then can obtain  $2I - \nabla^2 D(\mathbf{x}) = I + D^{-1}A$ . Here,  $D^{-1}A$  is transition matrix, where each entry is non-negative, the sum of each row is one and its eigenvalues are within the range  $[-1, 1]$ , so the eigenvalues of  $I + D^{-1}A$  are within the range  $[0, 2]$ . That is,  $I + D^{-1}A$  is a positive semidefinite matrix, and thus  $\nabla^2 D(\mathbf{x}) \preceq 2I$  holds on.

*Proof of Theorem 1.*  $D$  is a quadratic function, by second-order Taylor expansion of  $D$  around  $\mathbf{x}$ , we have the following inequality:

$$\begin{aligned} D(\tilde{\mathbf{x}}) &= D(\mathbf{x}) + \nabla D(\mathbf{x})^\top (\tilde{\mathbf{x}} - \mathbf{x}) + \frac{1}{2} (\tilde{\mathbf{x}} - \mathbf{x})^\top \nabla^2 D(\mathbf{x}) (\tilde{\mathbf{x}} - \mathbf{x}) \\ &= D(\mathbf{x}) - \nabla D(\mathbf{x})^\top \nabla D(\mathbf{x}) + \frac{1}{2} \nabla D(\mathbf{x})^\top \nabla^2 D(\mathbf{x}) \nabla D(\mathbf{x}) \\ &\leq D(\mathbf{x}) - \nabla D(\mathbf{x})^\top \nabla D(\mathbf{x}) + \nabla D(\mathbf{x})^\top \nabla D(\mathbf{x}) = D(\mathbf{x}). \end{aligned} \quad (19)$$

## B. Proof of Corollary 1

**Corollary 1.** *In dynamic graph embedding, we use  $\tilde{\mathbf{h}}_t$  and  $\mathbf{h}_t$  to denote two cases: applying graph convolution to  $\mathbf{x}_t$ ; not applying graph convolution to  $\mathbf{x}_t$ , respectively. Similarly, let the overall distance of pairwise nodes state at time step  $t$  be  $D(\tilde{\mathbf{h}}_t) = \frac{1}{2} \sum_{v_t^i, v_t^j} \tilde{a}_t \|\mathbf{h}_t^i - \mathbf{h}_t^j\|_2^2$ . Then we have:*

$$D(\tilde{\mathbf{h}}_t) \leq D(\mathbf{h}_t), \quad (20)$$

and

$$\sum_t D(\tilde{\mathbf{h}}_t) \leq \sum_t D(\mathbf{h}_t). \quad (21)$$

*Proof:* (1) In temporal-smoothness learning fashion,  $\tilde{\mathbf{h}}_t = \tilde{\mathbf{x}}_t = \text{GCN}(\mathbf{x}_t, \mathbf{A}_t)$  if using graph convolution, and  $\mathbf{h}_t = \mathbf{x}_t$  if no graph convolution, thus we can know that  $D(\tilde{\mathbf{h}}_t) \leq D(\mathbf{h}_t)$  from Theorem 1, then  $\sum_t (D(\tilde{\mathbf{h}}_t) \leq D(\mathbf{h}_t))$ . (2) In recurrent learning manner, we use the vanilla RNN, ignoring bias and activation, to show it. Here,  $\tilde{\mathbf{h}}_t = W\tilde{\mathbf{x}}_t + U\mathbf{h}_{t-1}$  (using graph convolution) and  $\mathbf{h}_t = W\mathbf{x}_t + U\mathbf{h}_{t-1}$  (no graph convolution), then we have  $\mathbf{h}_t - \tilde{\mathbf{h}}_t = W(\mathbf{x}_t - \tilde{\mathbf{x}}_t)$ . We can attribute the difference of  $\mathbf{h}_t$  and  $\tilde{\mathbf{h}}_t$  to  $\mathbf{x}_t$  and  $\tilde{\mathbf{x}}_t$ , where  $W$  is a transformation vector/matrix, which can be incorporated in transformation step in graph convolution. In Theorem 1, we know  $\tilde{\mathbf{x}}_t$  is shrunk from  $\mathbf{x}_t$ , therefore we can easy to know that  $\tilde{\mathbf{h}}_t$  will be shrunk from  $\mathbf{h}_t$ . With time increasing, the shrink will be amplified gradually if we initialize  $\mathbf{x}_t$  from the last time output  $\mathbf{h}_{t-1}$ , and the shrink appears in each time step independently which is still accumulated if we keep using the original feature of  $\mathbf{x}_t$  at each time step. Together, we have  $D(\tilde{\mathbf{h}}_t) \leq D(\mathbf{h}_t)$  and  $\sum_t (D(\tilde{\mathbf{h}}_t) \leq D(\mathbf{h}_t))$ .

## REFERENCES

- [1] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*, 2014, pp. 701–710.
- [2] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "LINE: Large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.
- [3] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.
- [4] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.
- [5] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *AAAI*, 2018.
- [6] Z. Lingxiao and A. Leman, "PairNorm: Tackling oversmoothing in GNNs," in *ICLR*, 2020.
- [7] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-I. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," *arXiv preprint arXiv:1806.03536*, 2018.
- [8] J. Klicpera, A. Bojchevski, and S. Günnemann, "Combining neural networks with personalized pagerank for classification on graphs," in *ICLR*, 2019.
- [9] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.
- [10] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *NeurIPS*, 2002, pp. 585–591.
- [11] J. B. Tenenbaum, V. De Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.
- [12] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu, "Attributed network embedding for learning in a dynamic environment," in *CIKM*, 2017, pp. 387–396.
- [13] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, "Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks," in *KDD*, 2018, pp. 2672–2681.
- [14] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NeurIPS*, 2016, pp. 3844–3852.
- [15] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1024–1034.
- [16] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.
- [17] A. Sankar, Y. Wu, L. Gou, W. Zhang, and H. Yang, "Dynamic graph representation learning via self-attention networks," *arXiv preprint arXiv:1812.09430*, 2018.
- [18] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, and C. E. Leiserson, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," *arXiv preprint arXiv:1902.10191*, 2019.
- [19] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.
- [20] X. da, r. chuanwei, k. evren, k. sushant, and a. kannan, "Inductive representation learning on temporal graphs," in *ICLR*, 2020.
- [21] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, "T-gcn: A temporal graph convolutional network for traffic prediction," *IEEE Transactions on Intelligent Transportation Systems (TITS)*, 2019.
- [22] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, "Gaan: Gated attention networks for learning on large and spatiotemporal graphs," *arXiv preprint arXiv:1803.07294*, 2018.
- [23] L. Zhu, D. Guo, J. Yin, G. Ver Steeg, and A. Galstyan, "Scalable temporal latent space inference for link prediction in dynamic social networks," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 28, no. 10, pp. 2765–2777, 2016.
- [24] P. Goyal, N. Kamra, X. He, and Y. Liu, "Dyngem: Deep embedding method for dynamic graphs," *arXiv preprint arXiv:1805.11273*, 2018.
- [25] G. Li, M. Muller, A. Thabet, and B. Ghanem, "DeepGCNs: Can gcns go as deep as cnns?" in *ICCV*, 2019, pp. 9267–9276.
- [26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.
- [27] R. Yu, H. Wenbing, X. Tingyang, and H. Junzhou, "DropEdge: Towards deep graph convolutional networks on node classification," in *ICLR*, 2020.
- [28] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *ACL*, 2014.
- [29] F. A. Gers, J. A. Schmidhuber, and F. A. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Comput.*, p. 2451–2471, Oct. 2000.
- [30] A. Ziat, E. Delasalles, L. Denoyer, and P. Gallinari, "Spatio-temporal neural networks for space-time series forecasting and relations discovery," in *ICDM*, 2017, pp. 705–714.
- [31] U. Singer, I. Guy, and K. Radinsky, "Node embedding over temporal graphs," *arXiv preprint arXiv:1903.08889*, 2019.
- [32] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.
- [33] H. Wang and J. Leskovec, "Unifying graph convolutional neural networks and label propagation," *arXiv preprint arXiv:2002.06755*, 2020.
- [34] P. Goyal, S. R. Chhetri, and A. Canedo, "Dyngraph2vec: Capturing network dynamics using dynamic graph representation learning," *Knowledge-Based Systems*, vol. 187, p. 104816, 2020.
- [35] L. Zheng, Z. Li, J. Li, Z. Li, and J. Gao, "AddGraph: anomaly detection in dynamic graph using attention-based temporal gcn," in *IJCAI*, 2019, pp. 4419–4425.
- [36] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AAAI*, 2010, pp. 249–256.
- [37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ICLR*, 2015.
- [38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *JMLR*, vol. 15, no. 1, pp. 1929–1958, 2014.