DynGCN: A Dynamic Graph Convolutional Network Based on Spatial-Temporal Modeling

Jing Li[1], Yu Liu[1], and Lei Zou[1,2]

[1] Peking University, China
[2] National Engineering Laboratory for Big Data Analysis Technology and Application (PKU), China

Abstract. Representation learning on graphs has recently attracted a lot of interest with graph convolutional networks (GCN) achieving state-of-the-art performance in many graph mining tasks. However, most of existing methods mainly focus on static graphs while ignoring the fact that real-world graphs may be dynamic in nature. Although a few recent studies have gone a step further to incorporate sequence modeling (e.g., RNN) with the GCN framework, they fail to capture the dynamism of graph structural (i.e., spatial) information over time. In this paper, we propose a **Dyn**amic **G**raph **C**onvolutional **N**etwork (**DynGCN**) that performs spatial and temporal convolutions in an interleaving manner along with a model adapting mechanism that updates model parameters to adapt to new graph snapshots. The model is able to extract both structural dynamism and temporal dynamism on dynamic graphs. We conduct extensive experiments on several real-world datasets for link prediction and edge classification tasks. Results show that DynGCN outperforms state-of-the-art methods.

Keywords: Dynamic graphs · GCN · Representation learning.

1 Introduction

Owing to the success of convolutional neural networks (CNN) in fields such as computer vision, natural language processing and speech recognition, researchers have showed a lot of interest in the topic of graph neural networks (GNN). However, compared with the Euclidean data structure of images, natural languages and speech signals, graphs are non-Euclidean data which makes it unsuitable for us to apply convolutions designed in CNN to graphs directly.

Therefore, graph convolutional networks (GCN) are specially designed to extract topological structures on graphs. They extend the concept of *convolution* to graph domain by designing operations that aggregate neighborhood information. Since topological structures and local information are extracted by the graph convolution operation, GCN is a powerful architecture in graph representation learning. Some existing studies [7, 9] have proved that GCN achieves state-of-the-art performance in a lot of graph mining tasks such as node classification, edge classification, link prediction, clustering, etc.

It is worthwhile to further investigate that most of existing graph convolutional networks are designed for static graphs. However, real-world graphs are dynamic in nature with insertion/deletion of nodes and edges or changing of properties all the time. For example, users of a financial network are constantly trading with each other, users of a social network may develop friendship over time and authors of a citation network are publishing new papers all the time, thus resulting in the dynamism of these networks. Under these dynamic scenarios, it requires dynamic models that aim at capturing not only structural information but also historical information on dynamic graphs.

A few recent studies have gone a step further by combining GCN with recurrent neural networks (RNN) since GCN is designed for structural information extraction and RNN is designed for sequence modeling which makes them a natural match in dynamic graph representation learning. Among them, some models use GCN to aggregate neighborhood information and then feed the output embeddings into RNN architectures to extract sequence information [11, 17]. While EvolveGCN [15] combines them in a different manner. It utilize RNN to evolve GCN parameters which results in a dynamic evolving GCN model for every snapshot along time axis. However, these existing works either model graph dynamism only on node embeddings that lacks adaptive learning or on model parameters that lacks structural dynamism extraction.

In this paper, we focus on both spatial and temporal dynamism on dynamic graphs and propose a **Dyn**amic **G**raph **C**onvolutional **N**etwork (**DynGCN**) that performs spatial-temporal convolutions in an interleaving manner with a model adapting mechanism. We conduct several experiments for DynGCN model on link prediction and edge classification tasks. DynGCN outperforms dynamic graph convolution baselines.

Our main contributions can be summarized as follows:

- We propose DynGCN, a self-adapting, spatial-temporal graph convolutional network that learns rich latent graph representations on dynamic graphs.
- We consider the problem of dynamic graph representation learning as a combination of spatial dynamism and temporal dynamism where either spatial or temporal dynamism is often ignored by existing methods.
- We give a comprehensive study for representation learning under extreme class imbalance situations. Our model shows superiority in identifying minority class with preserving total accuracy which indicates the necessity of spatial-temporal modeling.
- We conduct several experiments based on real-world scenarios. DynGCN outperforms state-of-the-art baselines thus demonstrates the efficacy of DynGCN in learning rich and high-level representations.

The remainder of the paper is organized as follows: we present related works for representation learning in section 2 and then give a detailed description of our proposed DynGCN model in section 3. In order to prove the efficacy of DynGCN, we show the experimental results and give a detailed analysis in section 4. Last but not least, we conclude our work in section 5.

2 Related Work

2.1 Static Graph Representation Learning

Representation learning aims to learn node embeddings into low dimensional vector space. A traditional way on static graphs is to perform Singular Vector Decomposition (SVD) on the similarity matrix computed from the adjacency matrix of the input graph [3, 14]. Despite their straightforward intuition, matrix factorization-based algorithms require more scalability. Inspired by Word2Vec [12] that maps words into low dimensional space, researchers developed some random walk-based approaches(e.g., DeepWalk [16] and Node2Vec[6]) that perform random walks [13] over the input graph and then feed these paths (regraded as *sentences*) into a SkipGram model [12] to learn node embeddings. What's more, with the success of deep learning, GNN has become an efficient tool in static graph representation learning [4, 5, 7, 9, 18]. One typical idea among these GNNs lies in graph convolution operation which adopts the concept of neighborhood aggregation to extract structural information. Yet, all methods mentioned above focus on static graphs and lack the ability to capture dynamism in some applications where graphs may change dynamically.

2.2 Dynamic Graph Representation Learning

It is a straightforward idea to extend static methods to dynamic ones with extra updating mechanism or sequence modeling architectures. Since GNN (e.g. GCN) is designed for structural extraction and RNN (e.g. LSTM or GRU) is designed for sequence modeling, it makes them a natural match to form a dynamic graph neural network.

Seo et al. proposed two GCRN [17] architectures that naturally use GCN to learn node embeddings and then feed them into LSTM to learn sequence dynamism along time axis. The only difference between these two architectures is that the second model uses a modified LSTM that replaces the Euclidean 2D convolution operator by graph convolution operator. Similarly, Manessia et al. proposed WD-GCN/CD-GCN [11] to learn dynamic graph embeddings by combining a modified LSTM and an extension of graph convolution operations to learn long short-term dependencies together with graph structures. WD-GCN takes as input a graph sequence while CD-GCN takes as input the corresponding ordered sequence of vertex features.

Furthermore, GCN-GAN [10] models the temporal link prediction tasks in weighted dynamic networks by combining GCN and RNN with generative adversarial networks (GAN). They use GCN to capture topological characteristics of each snapshot and employ LSTM to characterize the evolving features with a GAN followed to generate the next snapshot thus to learn a more effective representation. STAR [20] adds a dual attention mechanism on top of the GCN and RNN architectures. On the other side, Aldo Pareja et al. models dynamism is a different way, they proposed EvolveGCN [15] that employs RNN architecture (e.g. GRU or LSTM) to evolve the GCN weight parameters along time dimension to adapt models to graph dynamism.

Although these methods combine GCN with RNN from different aspects, they model dynamism either on output node embeddings that lacks adaptive learning or on the weight parameters that lacks the extraction of graph strctural dynamism.

3 Method

3.1 Problem Definition

A *graph* is denoted as $G = (V, E)$ where V is the set of vertices and E is the set of edges. The adjacency matrix of graph G is denoted as $A \in R^{N \times N}$, where $A_{i,j} = 1$ if $v_i, v_j \in V$ and $(v_i, v_j) \in E$ otherwise $A_{i,j} = 0$. Given a graph $G = (V, E)$, *graph representation learning* is a mapping function $f : V \to R^d$, where $d \ll N$. The mapping function f embeds every node into low-dimensional space while preserving original information at the same time, namely $y_v = f(v), \forall v \in V$. The more similar two nodes are in the graph G, the closer their representation vectors y_u and y_v are in the embedding space.

Dynamic graphs may be divided into two types according to the active time type for an edge, either time instances or time intervals, which results in continuous-time or discrete-time dynamic graphs [8, 19]. In our settings, we formulate dynamic graphs under discrete time and define *dynamic graph* and *dynamic graph representation learning* as follows:

Definition 1. *(**Dynamic Graph**) A dynamic graph is denoted as a set of graphs, i.e.* $\mathcal{G} = \{G_1, G_2, \ldots, G_T\}$*, where each* $G_t = (V_t, E_t)$ *is a snapshot at time step* $t, t \in \{1, 2, \ldots, T\}$*. For graph* $G_t, t \in \{1, 2, \ldots, T\}$*, the adjacency matrix* $A_t \in R^{N \times N}$ *is defined as follows:* $(A_t)_{i,j} = 1$ *if* $(v_t)_i, (v_t)_j \in V_t$ *and* $((v_t)_i, (v_t)_j) \in E_t$*, otherwise,* $(A_t)_{i,j} = 0$*.*

Definition 2. *(**Dynamic Graph representation learning**) Dynamic graph representation learning is to learn a set of mappings* $\mathcal{F} = \{f_1, f_2, \ldots, f_T\}$ *where each mapping function* $f_t, \forall t \in \{1, 2, \ldots, T\}$ *maps nodes of a dynamic graph* G_t *at time t into low-dimensional space while preserving origin information at the same time. The More similar two nodes are in the origin dynamic graph, the closer their representation vectors are in the embedding space.*

3.2 Architecture Overview

As shown in Fig.1, DynGCN performs spatial and temporal convolutions in an interleaving manner. The first layer of the model is a spatial convolution layer where the GCN model is updated by a GRU cell to self-adapt to graph dynamism. The spatial convolution layer aggregates neighborhood structural information with dynamism awareness. The second layer is a temporal convolution layer that aggregates information from current and historical time steps with dilated convolutions. After that, every node is represented by his current neighborhood information along with his historical neighborhood information. Then we add another spatial convolution layer on the top to aggregate spatial temporal information from neighborhoods.

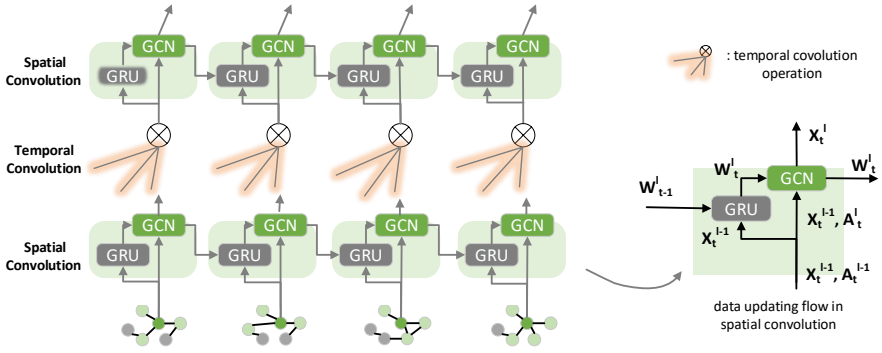


Fig. 1. An overview of DynGCN architecture

3.3 Spatial Convolution Layer

GCN has showed its superiority in learning graph topological structures, we utilize GCN unit to learn the structural information of every snapshot in dynamic graphs. Formally, given a graph $G_t = (V_t, E_t)$ at time step t, the adjacency matrix is denoted by $A_t \in R^{N \times N}$. At the l-th layer of GCN, it takes output node embedding vectors X_t^{l-1} of the $(l-1)$-th layer and the adjacency matrix A_t as input, and output the updated node embedding vectors X_t^l. We can write the operation of a GCN layer as follows:

$$X_t^l = \mathcal{F}(X_t^{l-1}, A_t, W_t^l) = \sigma(\hat{D_t}^{-1/2} \hat{A}_t \hat{D_t}^{-1/2} X_t^{l-1} W_t^l) \tag{1}$$

where $\hat{A}_t = A_t + I$ and I is an identity matrix. $\hat{D}_t = diag(\sum_{j=1}^{N} \hat{A}_{t(ij)})$ and $\hat{D_t}^{-1/2} \hat{A}_t \hat{D_t}^{-1/2}$ is a normalization of A_t served as a approximated graph convolution filter. W_t^l is the weight matrix of the l-th layer at time t and function σ is an activation function (e.g. ReLU). The initial input X_t^0 for the first layer is the node features matrix $Z_t \in R^{N \times K}$ where each row represents a K dimensional feature vector for each of N nodes. After L layers of graph convolution operations, the output matrix contains aggregated neighborhood information for every node in every single graph.

In consideration of graph dynamism, the spatial convolution layer further extends static GCN architecture with self-adapting mechanisms which is first introduced in EvolveGCN. The spatial convolution layer utilizes a recurrent network to update weight matrix of GCN unit. Although GRU and LSTM both work in updating GCN parameters, we choose GRU for our implementation since the input of GRU unit consists of hidden state and current embedding while LSTM only takes hidden state into consideration. By updating model parameters, GCN architecture is self-adapted to every time step.

For weight matrix W_t^l of l-th layer at time t, we obtain it by W_{t-1}^l and X_t^{l-1} through a GRU cell formulated as follows (the superscript l denotes for graph convolution layer, and the subscript t denotes for time step):

$$W_t^l = \mathcal{G}(X_t^{l-1}, W_{t-1}^l) = (1 - Z_t^l) \circ W_{t-1}^l + Z_t^l \circ \widetilde{W}_t^l \tag{2}$$

where:

$$Z_t^l = sigmoid(U_Z^l X_t^{l-1} + V_Z^l W_{t-1}^l + B_Z^l) \tag{3}$$

$$R_t^l = sigmoid(U_R^l X_t^{l-1} + V_R^l W_{t-1}^l + B_R^l) \tag{4}$$

$$\widetilde{W}_t^l = tanh(U_W^l X_t^{l-1} + V_W^l (R_t^l \circ W_{t-1}^l) + B_W^l) \tag{5}$$

The updating for weight matrix of the l-th layer is to apply the standard GRU operation on each column of the involved matrices independently since standard GRU maps vectors to vectors but we have matrices here. We treat W_{t-1}^l as the hidden state of GRU. Embedding vector X_t^{l-1} is chosen as the input of GRU at every time step to represent current information. Z_t^l, R_t^l, $\widetilde{W}_t^l$ are the update gate output, the reset gate output, and the pre-output, respectively. To deal with the inequality between the columns of weight matrix W_{t-1}^l and embedding matrix X_t^{l-1}, a summarization [2] on X_t^{l-1} is further added to the evolving graph convolution layer to transfer X_t^{l-1} to have the same columns as W_{t-1}^l .

GCN unit aggregates embedding vectors from $(l-1)$-th layer to l-th layer and GRU unit updates weight matrix from time step $t-1$ to t. The detailed data updating flow in spatial convolution layer is illustrated in Fig. 1. Formally, the spatial convolution architecture can be written as follows:

$$X_t^l = \mathcal{F}(X_t^{l-1}, A_t, \mathcal{G}(X_t^{l-1}, W_{t-1}^l)) \tag{6}$$

Where functions $\mathcal{F}$ and $\mathcal{G}$ are graph convolution operation and weight evolving operation respectively as declared above.

3.4 Temporal Convolution Layer

It is a key issue to capture temporal information along time dimension in dynamic graph embedding problems. A lot of existing models employ RNN architectures for sequence modeling. However, RNN-based methods are memory consuming and time consuming because of the complex gate mechanisms. Besides, standard RNN suffers from gradient disappear and can only obtain short-term memory.

Although variant architectures like LSTM and GRU fix these problems to some extent, they are still at a disadvantage compared to CNN-based architectures (specially, TCN[1]) which require less memory and training time along with a flexible receptive field size. What's more, RNN-based methods model temporal dynamism by gate mechanism where historical information is considered only in hidden state of every time step. While in CNN-based methods, information of every historical time steps are aggregated by convolution operations which guarantees rich information and at the same time, unifies the idea of spatial and temporal convolutions.

We employ TCN architecture to capture historical information in the proposed DynGCN model. The temporal convolution layer uses a 1D fully convolutional unit and a causal convolutional network unit. The 1D fully-convolutional

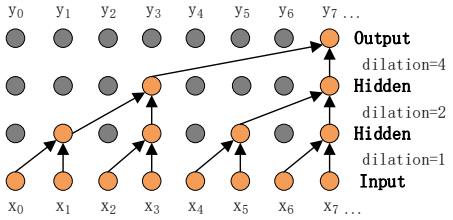


Fig. 2. An example of dilated convolution in temporal convolution layer with $dilation = 1, 2, 4$ and $filter\_size = 2$. This guarantees that the aggregated information covers all historical time steps.

unit guarantees that the output has the same length as the input and the causal convolutional unit guarantees that an output at time step t_k is convoluted with input of time steps $t \leq t_k$, which means aggregating information from current and historical time steps. Further, the causal convolution is equipped with dilation to enable a larger and more flexible receptive field. An example of dilated convolution is shown in Fig.2. Formally, given $X^l \in R^{T \times M}$, a length-T sequence of node embedding vectors in the l-th layer with M channels, and a filter $f : \{0, 1, \ldots k-1\}$, the temporal convolution operation $\mathcal{H}$ on element x of X^l is formalized as follow:

$$\mathcal{H}(x) = (X^l *_d f)(x) = \sum_{i=0}^{k-1} f(i) \cdot X^l_{x-d \cdot i} \tag{7}$$

d is the dilation factor, k is the filter size and $x - d \cdot i$ represents the direction of the past. Benefit from this, we can obtain a larger receptive field by either increasing filter size k or increasing the dilation factor d. What's more, convolution operations provide parallelism in processing a sequence thus is efficient.

Node embedding vectors of the l-th layer are aggregations of neighborhood information. By performing temporal convolution operations on node embedding vectors, we can then aggregate historical information along time axis. With another evolving graph convolution layer at the top, we get representations with current and historical information for both a node and its neighborhood. Therefore, performing spatial and temporal convolutions in an interleaving manner ensures both topological and historical information to be included in high-level node embedding vectors learned from our DynGCN architecture.

3.5 Tasks and Model Training

In order to show the capacity in representation learning of DynGCN, we further train the model based on two specific downstream tasks: *link prediction* and *edge classification*.

Link prediction. The task of link prediction aims to predict the existence of an edge (u, v) at a future time step t.

Edge classification. The task of edge classification is to classify the label of an edge (u, v) at time step t.

Given the output embedding vectors of DynGCN for every node, we form the prediction of an edge label or edge existence based on concatenating embedding vectors of two nodes of this edge. Assuming that the output embedding vector of node u at time t is denoted by X_t^u, and P is the parameter matrix, the prediction can be written as $y_t^{uv} = softmax(P[X_t^u; X_t^v])$. The loss function is defined as $L = -\sum_{t=1}^{T}\sum_{(u,v)}\alpha_{uv}\sum_{i=1}^{N}(z_t^{uv})_i log(y_t^{uv})_i$, where z_t^{uv} is the ground-truth one-hot label vector and the nonuniform weight α_{uv} serves as hyperparameter to balance class distribution since datasets we use in our experiments suffer from serious class imbalance problems.

4 Experiments

4.1 Datasets

We verify the effectiveness of the DynGCN model on three data sets. All statistics about these data sets are summarized in Tabel 1.

Table 1. Dataset statistics

Datasets	Nodes	Edges	Positive Edges	Train / Test / Val (Edge classification)	Train / Test / Val (Link prediction)
OTC	5881	35592	89%	96 / 21 / 21	96 / 14 / 28
Alpha	3783	24186	93%	96 / 22 / 22	96 / 13 / 28
AS	6474	13895	—	—	70 / 10 / 20

- **OTC.**[3] This network of bitcoin users rate each other in a scale from -10 (total distrust) to +10 (trust) and every rating comes with a time stamp. We generate a sequence of 138 time steps and split it into training, testing and validation sets as shown in Table 1. Specially, this data set suffers from class imbalance probelms since there are 89% of positive edges.
- **Alpha.**[4] Alpha is similar to OTC but extracted from a different bitcoin platform. We extract a sequence of 140 time steps. Alpha has a higher ratio of 93% positive ratings than OTC.
- **Autonomous Systems (AS).**[5] The graph of routers comprising the Internet are organized into sub-graphs called autonomous systems and each exchanges traffic flows with some neighbors. AS dataset is a who-talks-to-whom network constructed from the BGP (Border Gateway Protocol) logs. The dataset spans an interval of 785 days we form a sequence of 100 time steps.

[3] http://snap.standford.edu/data/soc-sign-bitcoin-otc.html
[4] http://snap.standford.edu/data/soc-sign-bitcoin-alpha.html
[5] http://snap.stanford.edu/data/as-733.html

4.2 Baselines

We compare DynGCN with both static and dynamic baselines where dynamism is considered in different manners. By conducting these comprehensive comparisons, we verify that our architecture design makes sense and has the ability to model rich information.

- **GCN[9]:** This is a classical method for static graph representation learning that learns node embeddings by aggregating neighborhood information. For dynamic graph, we use the same GCN model for all snapshots.
- **GCN-GRU:** This baseline is a combination of graph convolution and sequence modeling. The output node embeddings of GCN unit are feed into a GRU architecture to model dynamism on node representations. This method comes from the Method 1 proposed by Seo et al. [17] and we further adjust their GNN from ChebNet to GCN and their RNN from LSTM to GRU for a better comparison.
- **EvolveGCN[15]:** Different from the above mentioned GCN-GRU method, EvolveGCN uses RNN to model dynamism on GCN parameters. The authors proposed two types of EvolveGCN model, namely EvolveGCN-H and EvolveGCN-O. The -H version is implemented by using GRU architectures to evolve GCN parameters, and the -O version is implemented by using LSTM architectures to evolve GCN parameters.

4.3 Experimental Results

Link Prediction. The evaluation metric for link prediction is Mean Average Precision (MAP). MAP averages the predictions for all nodes and higher MAP shows that the model can predict well for more nodes. Fig. 3. shows the performance comparisons of all methods in link prediction tasks. Although all methods perform badly because of the data distribution and class imbalance, DynGCN still outperforms all other baselines in all three datasets.

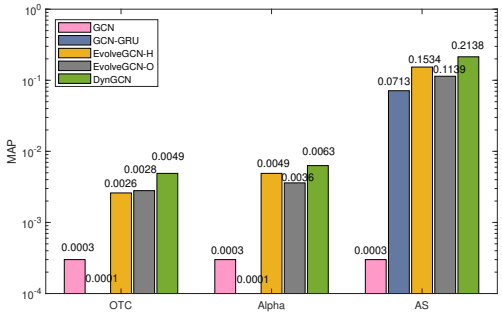
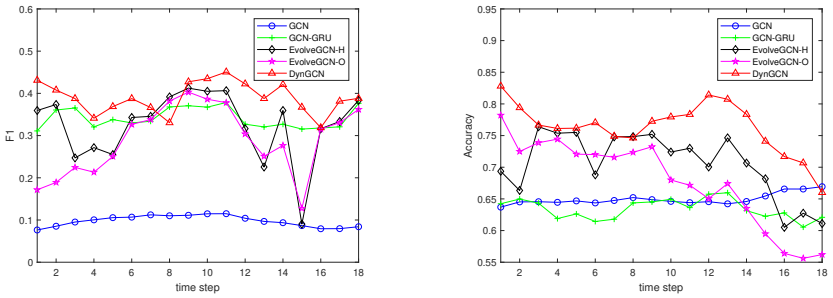


Fig. 3. MAP results for link prediction (the y-axis is in log scale).

Edge Classification. Table 2 shows the performance comparisons between DynGCN and all baselines on edge classification task. The accuracy and weighted F1 score are for all classes and F1 score and the corresponding precision and recall are for the negative edges. Since both datasets suffer from serious class imbalance problems, classification tasks on them are of great challenges. Our proposed DynGCN model achieves the best performance by a significant margin.

Table 2. Experimental results for edge classification tasks

Datasets	Methods	Accuracy	Weighted F1	F1 (Precision/ Recall)
OTC	GCN	0.5535	0.6155	0.3186 (0.2050/ 0.7148)
	GCN-GRU	0.5981	0.6551	0.3386 (0.2256/ 0.6788)
	EvolveGCN-H	0.6839	0.7118	0.2171 (0.1754/ 0.2848)
	EvolveGCN-O	0.6873	0.7168	0.2446 (0.1962/ 0.3246)
	DynGCN	**0.7949**	**0.8019**	**0.3583** (0.3309/ 0.3905)
Alpha	GCN	0.6495	0.6814	0.0992 (0.0802/ 0.1303)
	GCN-GRU	0.6340	0.6865	0.3433 (0.2353/ 0.6347)
	EvolveGCN-H	0.7569	0.7668	0.3160 (0.2738/ 0.3735)
	EvolveGCN-O	0.6773	0.7126	0.2971 (0.2288/ 0.4234)
	DynGCN	**0.8011**	**0.8027**	**0.3519** (0.3549/ 0.3489)



(a) F1 score over time (b) Accuracy score over time

Fig. 4. Performance over time for dataset Alpha on edge classification.

Advantage of dynamic modeling. We further plot the accuracy score and F1 score for the minority class over time for Alpha on edge classification. Although the same results are also observed on OTC, we omit figures due to space limitation. As shown in Fig. 4 (a), there is an obvious gap between GCN method and all other methods while DynGCN still outperforms all baselines along time steps. Besides, the accuracy score for GCN method is also relatively small as shown in Fig. 4 (b). Intuitively, GCN is designed for static graphs and no dynamism is considered in GCN model. The superior performance of other methods against GCN indicates the advantage of dynamic modeling.

Advantage of spatial-temporal modeling. By further analyzing Fig. 4, we can observe that DynGCN outperforms all baselines in F1 score of minority class and accuracy score of all classes over time. Especially, at time step 15, all baselines show extremely low performance, but DynGCN performs relatively stable and still keeps an absolute advantage in F1 score. This is benefit from the special architecture of DynGCN designed to capture both spatial and temporal information from current and the past.

5 Conclusion

In this paper, we proposed DynGCN for representation learning on dynamic graphs. By conducting spatial and temporal convolutions in an interleaving manner with a model adapting architecture, we obtain rich information aggregated along both time dimension and graph dimension. The extended experiments show that DynGCN outperforms all baselines. The study of dynamic graph representation learning opens up a lot of future works. For example, the model scalability remains more improvements and we plan to extend this work to other graph mining tasks like node classification, clustering. Besides, it is also an interesting research topic on modeling continuous time for dynamic graphs.

Acknowledgements. This work was supported by NSFC under grant 61932001, 61961130390. This work was also supported by Beijing Academy of Artificial Intelligence (BAAI). Lei Zou is the corresponding author of this work.

References

1. Bai, S., Kolter, J.Z., Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. CoRR **abs/1803.01271** (2018), http://arxiv.org/abs/1803.01271
2. Cangea, C., Veličković, P., Jovanović, N., Kipf, T., Liò, P.: Towards sparse hierarchical graph classifiers (2018)
3. Cao, S., Lu, W., Xu, Q.: Grarep: Learning graph representations with global structural information. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. pp. 891–900. CIKM '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2806416.2806512, http://doi.acm.org/10.1145/2806416.2806512
4. Chen, J., Ma, T., Xiao, C.: Fastgcn: Fast learning with graph convolutional networks via importance sampling (2018)
5. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems 29, pp. 3844–3852. Curran Associates, Inc. (2016), http://papers.nips.cc/paper/6081-convolutional-neural-networks-on-graphs-with-fast-localized-spectral-filtering.pdf
6. Grover, A., Leskovec, J.: Node2vec: Scalable feature learning for networks. In: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 855–864. KDD '16,

ACM, New York, NY, USA (2016). https://doi.org/10.1145/2939672.2939754, http://doi.acm.org/10.1145/2939672.2939754
7. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 1024–1034. Curran Associates, Inc. (2017), http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.pdf
8. Kazemi, S.M., Goel, R., Jain, K., Kobyzev, I., Sethi, A., Forsyth, P., Poupart, P.: Relational representation learning for dynamic (knowledge) graphs: A survey. CoRR **abs/1905.11485** (2019), http://arxiv.org/abs/1905.11485
9. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. CoRR **abs/1609.02907** (2016), http://arxiv.org/abs/1609.02907
10. Lei, K., Qin, M., Bai, B., Zhang, G., Yang, M.: GCN-GAN: A non-linear temporal link prediction model for weighted dynamic networks. CoRR **abs/1901.09165** (2019), http://arxiv.org/abs/1901.09165
11. Manessi, F., Rozza, A., Manzo, M.: Dynamic graph convolutional networks. Pattern Recognition **97**, 107000 (2020). https://doi.org/https://doi.org/10.1016/j.patcog.2019.107000, http://www.sciencedirect.com/science/article/pii/S0031320319303036
12. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space (2013)
13. Noh, J.D., Rieger, H.: Random walks on complex networks. Physical Review Letters **92**(11) (Mar 2004). https://doi.org/10.1103/physrevlett.92.118701, http://dx.doi.org/10.1103/PhysRevLett.92.118701
14. Ou, M., Cui, P., Pei, J., Zhang, Z., Zhu, W.: Asymmetric transitivity preserving graph embedding. In: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1105–1114. KDD '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2939672.2939751, http://doi.acm.org/10.1145/2939672.2939751
15. Pareja, A., Domeniconi, G., Chen, J., Ma, T., Suzumura, T., Kanezashi, H., Kaler, T., Leisersen, C.E.: Evolvegcn: Evolving graph convolutional networks for dynamic graphs. CoRR **abs/1902.10191** (2019), http://arxiv.org/abs/1902.10191
16. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 701–710. KDD '14, ACM, New York, NY, USA (2014). https://doi.org/10.1145/2623330.2623732, http://doi.acm.org/10.1145/2623330.2623732
17. Seo, Y., Defferrard, M., Vandergheynst, P., Bresson, X.: Structured sequence modeling with graph convolutional recurrent networks (2016)
18. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks (2017)
19. Wang, Y., Yuan, Y., Ma, Y., Wang, G.: Time-dependent graphs: Definitions, applications, and algorithms. Data Science and Engineering **4**(4), 352–366 (2019). https://doi.org/10.1007/s41019-019-00105-0, https://doi.org/10.1007/s41019-019-00105-0
20. Xu, D., Cheng, W., Luo, D., Liu, X., Zhang, X.: Spatio-temporal attentive rnn for node classification in temporal attributed graphs. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence. pp. 3947–3953. IJCAI'19, AAAI Press (2019)