IMPROVING DYNAMIC GRAPH CONVOLUTIONAL NETWORK WITH FINE-GRAINED ATTENTION MECHANISM

Bo Wu, Xun Liang, Xiangping Zheng, Yuhui Guo, Hui Tang

School of Information, Renmin University of China, Zhongguancun Street, Beijing

ABSTRACT

Graph convolutional network (GCN) is a novel framework that utilizes a pre-defined Laplacian matrix to learn graph data effectively. With its powerful nonlinear fitting ability, GCN can produce high-quality node embedding. However, generalized GCN can only handle static graphs, whereas a large number of graphs are dynamic and evolve over time, which limits the application field of GCN. Facing the challenge, GCN with recurrent neural network (e.g., RNN) is naturally combined to acquire dynamic graph changes through joint training. However, these methods must use the node information during the entire timeline and ignore two subtle factors: the influence of nodes change with time and are related to the frequency of events. Therefore, we propose a stable and scalable dynamic GCN method using a fine-grained attention mechanism named FADGC. We use GCN to obtain static node vectors at each timestep and integrate node influence factors with multi-head attention for graph time-series learning. Experiments on multiple datasets show that our approach can better capture the inherent special characteristics of different dynamic graphs and achieve higher performance compared with related approaches.

***Index Terms*—** Graph convolutional network, Dynamic Graph, Fine-grained Attention

1. INTRODUCTION

Graph, as a general and elegant data structure, is used to model the entities and the relationship between entities. The Internet, social network in a macro perspective, together with protein, compound molecules in micro perspective, can all be modeled by a graph structure. GCN, leveraging Fourier transform of Laplacian matrix [1, 2, 3, 4], can learn the inherent rules and deeper semantic features of nodes and edges in a graph. Owing to its strong nonlinear fitting ability to graph structure data, GCN shows higher accuracy and better robustness in graph-related problems in different fields [5, 6]. However, the graph is not static, whereas most of them are dynamic in real life. For example, in social networks, the relationship between people is constantly changing. Old and new friends will form a new social network relationship. Considering that the content attributes (i.e., node and edge features) and structure attributes (i.e., degree and number of nodes) of the graph have changed [7, 8], the original parameter matrices of GCN can no longer fit the graph data well. Therefore, relearning the graph and retraining the parameters are necessary. This procedure will not only lead to a large training cost but also cause difficulty in capturing certain time features of the graph under dynamic changes. In this case, some works combine GCN and RNN to learn dynamic graphs, that is, inputting graphs to GCN and using RNN for forming node embedding. This can simulate the evolution of graphs to some extent. However, the framework is insufficient to consider the process of graph change, ignoring two very important characteristics of the dynamic graph on the timeline. First, the influence of the latest events is greater than that of the previous nodes (e.g., the influence of new hot events is greater than that of old ones in the public opinion network). Second, the more events occurring in the unit time step, the greater the influence of nodes (e.g., if an individual interacts with others frequently, the social expert has a great influence on social networks). Therefore, we use an attention mechanism to include these two factors in a unified manner and build a novel dynamic graph convolutional network using a fine-grained attention mechanism called FADGC, which can effectively capture the characteristics of dynamic graph changes.

2. RELATED WORK

Most of the existing dynamic GCN framework can be expressed as GCN combined with RNN. GCN captures the graph information in each time segment to obtain node embedding, which is input into RNN for sequence learning. [9] proposed two different dynamic graph convolutional layers: WD-GCN and CD-GCN. The first is similar to the GCRN [10] model, which combines ChebNet with LSTM for dynamic graphs learning. [11] also started with the parameter weight of GCN and proposed EvolveGCN. The EvolveGCN, which is divided into EvolveGCN-H and EvolveGCN-O, two variants, utilizes RNN (typically LSTM and GRU) to evolve the weights of GCN. However, the input at each time step needs to wait for the processing of the previous weights of GCN, which leads to low time efficiency.

3. FADGC MODEL

This section elaborates on the proposed architecture of FADGC. FADGC is composed of two parts. One is the standard graph convolutional network, and the other is the node influence capture guided by a hierarchical attention mechanism. The details of each part are illustrated as follows.

3.1. Graph Convolutional Network (GCN)

To model a static graph at each time step, we use the graph convolutional network [4], which consists of multiple layers. It is similar to a fully connected layer but utilizes an adjacent matrix and feature matrix to forward information transmission, which produces the node embedding. Its mathematical formula can be expressed as follows:

$$\mathbf{H}_t^l = \sigma\left(\hat{\mathbf{A}}\mathbf{H}_t^{l-1}\mathbf{W}^l\right) \tag{1}$$

$$\hat{\mathbf{A}} = \widetilde{\mathbf{D}}^{-1/2}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-1/2}, \widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}, \widetilde{\mathbf{D}} = \operatorname{diag}\sum_j \widetilde{\mathbf{A}}_{ij} \tag{2}$$

$\mathbf{H}_t^l$ is the output of l current layer at time t calculated by the adjacency matrix $\hat{\mathbf{A}}$ and the node embedding matrix $\mathbf{H}_t^{l-1}$ of the preceding layer. Then the weight layer $\mathbf{W}^l$ is used to update the product of the above two. σ is the activation function (typically ReLU) for all middle layers except the output layer. $\mathbf{I}$ is the identity matrix. The original adjacency matrix of the graph is $\mathbf{A}$, which can be used for summing each row to obtain the degree matrix $\widetilde{\mathbf{D}}$. The initial input part of $\mathbf{H}_t^{l-1}$ comes from the feature matrix of graph nodes, which means $\mathbf{H}_t^0 = \mathbf{X}_t$. Finally, the embedding vector matrix of all nodes at each time step in the last layer can be represented as $\mathbf{H}_t^{\text{out}}$, of which are composed the embedding of a single node $\mathbf{h}_t^{\text{out}}$.

3.2. Fine-grained Attention Mechanism

This section discusses two factors that graph convolutional network must consider in adapting to the dynamic graph. *(1) The influence of the nodes with the latest events is higher than that of the previous nodes. (2) The more events are occurring in a unit time step, the greater the influence of the nodes.* Figure 2 illustrates the architecture of the FADGC model.

We defined a time factor variable ζ to obtain an accurate time factor influence change for the first factor. ζ is the time decay function with a node-dependent and learnable decay rate. $u > 0$ is a super parameter greater than 0, used to measure the influence factor of the current time step from the initial time step. The time window is used to reflect the change range of the graph over a period in history, expressed as $\mathcal{T} \in [t_s, t_f]$, where t_s and t_f denotes the start and end time steps respectively. For the influence factor at each time step in time window $\mathcal{T}$, we use ζ_k as presentation:

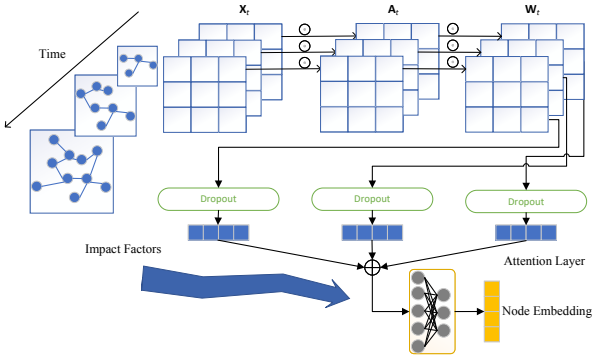

Fig. 1. The overall architecture of FADGC. Each graph snapshot forms a feature matrix and adjacency matrix. The feature matrix and adjacency matrix are multiplied by the weight matrix at each step. We add a dropout layer to prevent overfitting. After adding the influence factors, We concatenat the output of the attention layer together and finally input into the full connection layer to obtain the final node embedding, where $\odot$ and $\oplus$ represent dot product and concatenation, respectively.

$\zeta_k = f(u(t_k - t_s))$. $f(\cdot)$ stands for the function, which indicates how to enlarge the time influence factor. It can be exponential, linear, or other functions. We use exponential functions here to amplify this effect as much as possible.

After obtaining the time influence factor, we apply it to the node embedding. Following the highly effective attention-based models for natural language processing [12], we use the self-attention mechanism to calculate the coefficient through a single-layer neural network. The attention coefficient can be expressed as follows:

$$\alpha_k = \frac{\exp\left(\sigma\left(\zeta_k \mathbf{a}^\top\left[\mathbf{P}\mathbf{h}_{t_f} \oplus \mathbf{P}\mathbf{h}_k\right]\right)\right)}{\sum_{k\in\mathcal{T}}\exp\left(\left(\sigma\left(\zeta_k \mathbf{a}^\top\left[\mathbf{P}\mathbf{h}_{t_f} \oplus \mathbf{P}\mathbf{h}_k\right]\right)\right)\right.} \tag{3}$$

where $\mathbf{h}_{t_f}$ and $\mathbf{h}_k$ represent the node representation of the final node step and the node representation of the intermediate time step, respectively. $\alpha_k \in \mathbb{R}^{2d}$ serves as the attention weight vector and $\mathbf{P} \in \mathbb{R}^{d\times d}$ is the attention weight matrix. $\cdot^\top$ represents the transposition. We perform attention calculation for all nodes in the time window to obtain sufficient expression and transform the input features into higher-level features. At least a learnable linear transformation is needed. Therefore, as an initial step, the shared linear transformation is parameterized by the weight matrix $\mathbf{P}$. $\oplus$ stands for the concatenation operation, σ is the activation function, and we use sigmoid here.

The second factor is that the node with relatively many changes in a unit time step tends to have a strong influence. The impact factor describes the activity of nodes. If the node change frequency is significant, they interact with the surrounding nodes frequently for some time in the past. This finding means that these nodes are highly active, and their identity is essential. By contrast, if a node does not contact

the surrounding nodes in the previous period, its activity remains low and is supposed to be inactive. The nodes falling into this category possess a weak influence on other nodes in the present situation. This kind of influence factor can be measured by the degrees change of the nodes in a unit time step. If a node may be subject to alteration, again and again, the change frequency of its degree will be more pronounced, expressed as follows:

$$\mathcal{S} = f(\frac{\widetilde{\mathbf{D}}}{t_k - t_s}) \tag{4}$$

Similar to Formula 3, the function f here is also used to amplify the influence of the frequency of the event. Similarly, we also use the exponential function to enlarge the influence factor.

Until now, we have defined two influence factors. In addition, the two factors ought to work together and create a linkage effect. The closer a node is to the current time step in the dynamic graphs, the more significant the impact. Simultaneously, the more frequent a node changes per unit time, the more significant the impact on the surrounding nodes. They are superimposed together to exert influence. Given that each influence factor causes different effects, we intend to reconcile the influence changes of the two factors. It is represented by a joint influence factor λ_k:

$$\lambda_k = \beta\alpha_k + (1-\beta)\mathcal{S} \tag{5}$$

where $\beta \in [0,1]$ is the weight of the constraint of α_k and $\mathcal{S}$. Combining the two parts of influence factors, we can preserve the subtle changes of structural and temporal properties in dynamic graphs.

After obtaining the joint influence factors of coefficient, we multiply them to every node in the time window by the node vector at each time step; we obtain the aggregated features from the node vectors with attention weight matrix $\mathbf{P}$ in the time window as follows:

$$\mathbf{z}_t = \sigma(\sum_{k\in\mathcal{T}} \lambda_k \mathbf{P}\mathbf{h}_k) \tag{6}$$

To obtain a more stable representation, we extend the self-attention mechanism to the multi-head attention mechanism according to the description [13]. K independent attention heads are executed, and the node vectors are concatenated together. Simultaneously, the value of K exactly corresponds to the total number of time steps in the time window $\mathcal{T}$. Therefore, the equation (6) can be reformulated as:

$$\mathbf{z}_t = \sigma(\frac{1}{K}(\mathop{\Big\Vert}_{k=1}^{K} \sum_{k\in\mathcal{T}} \lambda_k \mathbf{P}_k \mathbf{h}_k)) \tag{7}$$

The adjacency matrix determines the graph's structure, so we reconstruct the adjacency matrix of the graph to represent the topological structure of the graph after a period of evolution.

Table 1. Statistics of the datasets.

Datasets	Nodes	Edges	Time Steps (Train/Val/Test)
SBM	1,000	4,870,863	35 / 5 / 10
BC-OTC	5,881	35,588	95 / 14 / 28
BC-Alpha	3,777	24,173	95 / 13 / 28
UCI	1,899	59,835	62 / 9 / 17
AS	6,474	13,895	70 / 10 / 20
Reddit	55,863	858,490	122 / 18 / 34
Elliptic	203,769	234,355	31 / 5 / 13

According to [14], the generated node vector matrix and its transposed inner product are used as the reconstructed adjacency matrix, and they should be as similar as the original adjacency matrix as possible. We measure it by KL divergence [15]. After combining the loss of the reconstructed graph and the loss of nodes classification, the final loss function can be defined as:

$$\mathcal{L} = D_{KL}(\sigma(\mathbf{Z}_t\mathbf{Z}_t^\top)||\mathbf{A}_t) - \sum^{n_t} ylog(p(\mathbf{z}))) + \gamma\mathcal{L}_{reg} \tag{8}$$

where regularization term $\mathcal{L}_{reg} = \|\mathbf{W}\|_2^2 + \|\mathbf{P}\|_2^2$ and we set $\gamma = 0.01$ here. The time complexity of FADGC is $O(K \cdot \mathcal{T}(|V|Fd + |E|Fd))$, where K is the number of multiple independent attention heads. $|V|$ and $|E|$ are the number of nodes and edges. F is the number of input features, and d is the final embedding dimension.

4. EXPERIMENTS

This section verifies the effectiveness of the method through experiments. We focus on the experimental results and explain why our approach is superior to other methods. We selected a total of seven dynamic graph datasets. Table 1 shows the detailed statistics of datasets. We compare the performance of FADGC against the following methods, including static GCN, three dynamic GCN, and two unsupervised dynamic network embedding methods. To explore the contribution of different influencing factors to the model, we conducted ablation experiments simultaneously. We record the model only using the first influence factor as FADGC-c, the model only using the second influence factor as FADGC-h, the model using both as FADGC. Here are the experimental results of different tasks.

Link Prediction. Table 2 shows the results of MAP and MRR evaluation metrics of FADGC and baseline in the link prediction task. FADGC achieves the best performance on SBM, BC-OTC, BC-Alpha, and AS datasets, and FADGC-c and FADGC-h also achieved good results. The second-best performance on UCI datasets is only 0.002 lower than EvolveGCN-O. We think that FADGC does not reach the best results in UCI datasets because some noise points exist in the

Table 2. Results for link prediction. Significantly outperforms Line at the: ** 0.01 and * 0.05 level, paired t-test.

Model	MAP					MRR				
	SBM	BC-OTC	BC-Alpha	UCI	AS	SBM	BC-OTC	BC-Alpha	UCI	AS
Static GCN [4]	0.1987	0.0003	0.0003	0.0251	0.0003	0.0138	0.0025	0.0031	0.1141	0.0555
GCN-LSTM [10]	0.1897	0.0001	0.0002	0.0103	0.0503	0.0121	0.0001	0.0003	0.0973	0.3217
GCN-GRU [10]	0.1898	0.0001	0.0001	0.0114	0.0713	0.0119	0.0003	0.0004	0.0985	0.3388
DynGEM [16]	0.1680	0.0134	0.0525	0.0209	0.0529	0.0139	0.0921	0.1287	0.1055	0.1028
dyngraph2vecAE [17]	0.0983	0.0090	0.0507	0.0044	0.0331	0.0079	0.0916	0.1478	0.0540	0.0698
dyngraph2vecAERNN [17]	0.1593	0.0220	0.1100	0.0205	0.0711	0.0120	0.1268	0.1945	0.0713	0.0493
EvolveGCN-H [11]	0.1947	0.0026	0.0049	0.0126	0.1534	0.0141	0.0690	0.1104	0.0899	0.3632
EvolveGCN-O [11]	0.1989	0.0328	0.0036	**0.0270**	0.1139	0.0138	0.0968	0.1185	**0.1379**	0.2746
FADGC-c	0.2001**	0.0215**	0.1306*	0.0210**	0.1564*	0.0112**	0.0987**	0.2516*	0.0764**	0.3955**
FADGC-h	0.1934**	0.0216**	0.1456*	0.0195**	0.1601*	0.0131**	0.1115**	0.3016*	0.1058**	0.4106**
FADGC	**0.2133****	**0.0246****	**0.1783***	0.0250**	**0.1711***	**0.0152****	**0.1323****	**0.3221***	0.1156**	**0.4610****

Table 3. Results for edge classification.

Model	Micro F1			Macro F1		
	BC-OTC	BC-Alpha	Reddit	BC-OTC	BC-Alpha	Reddit
Static GCN	77.54	76.48	67.41	46.13	47.10	43.15
GCN-LSTM	78.80	70.23	68.43	47.86	48.16	45.56
GCN-GRU	79.12	69.10	69.12	45.61	50.54	49.15
EvolveGCN-H	86.89	**85.11**	77.55	59.64	52.65	51.89
EvolveGCN-O	87.54	80.64	75.13	59.10	51.49	50.16
FADGC-c	87.11	84.42	76.84	58.59	57.46	52.55
FADGC-h	88.10	83.47	78.80	58.81	56.44	54.22
FADGC	**88.13**	85.10	**78.84**	**60.55**	**58.18**	**55.13**

Table 4. Results for node classification.

Model	Precision	Recall	F1
Static GCN	60.15	30.19	40.20
GCN-LSTM	78.45	59.16	67.45
GCN-GRU	77.94	**61.25**	68.59
EvolveGCN-H	41.84	33.56	37.24
EvolveGCN-O	75.57	54.48	63.31
FADGC-c	77.89	58.94	67.43
FADGC-h	79.47	59.45	68.99
FADGC	**80.99**	60.89	**69.52**

UCI datasets, which cause the disorder of node change trend in the process of graph change.

Edge Classifications. As reported in Table 3, dynamic GCN is better than static GCN in general. On the micro F1 evaluation index, in BC-OTC and Reddit datasets, the optimal results are 88.13 and 78.84 respectively made by FADGC, which are 0.67% and 1.67% higher than the suboptimal results. Comparatively, EvolveGCN-H achieves the best performance in BC-Alpha. On the macro F1 evaluation index, FADGC achieves very competitive results and is significantly ahead of other methods.

Node Classifications. In the Elliptic dataset, illegal and legal classes are extremely unbalanced. A total of 42,019 valid node samples are labeled, and ten percent are illegal class nodes. According to [11], the micro F1 value is higher than 0.95. Therefore, showing the F1 value of the legal class is not necessary. For bitcoin transactions, we are concerned about illegal transaction samples and evaluate the F1 value of the illegal class. As reported in Table 4, the precision of FADGC is 80.99, the recall rate is 60.89, and the final F1 score is 69.52, which is the best result surpassing baselines.

5. CONCLUSIONS

In this paper, we proposed the FADGC model based on fine-grained attention mechanism to an embedding dynamic graph. FADGC focuses on two factors that are easily ignored in the dynamic graph convolutional network. The first is that the closer the node is to the current time step, the greater the influence of the node. The second is that the more frequent the node changes in the unit time step, the higher the influence of the node on other nodes. We summarize these two elements with an attention mechanism to better capture the temporal characteristics of a dynamic graph. Experimental results on several dynamic graph datasets demonstrate the effectiveness of our method. Our future direction is to consider more time characteristic factors and extend the model to larger dynamic graphs.

6. ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (62072463, 71531012), and the National Social Science Foundation of China (18ZDA309). Xun Liang is the corresponding author of this paper.

7. REFERENCES

[1] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun, "Spectral networks and deep locally connected networks on graphs," in *Proceedings of 2nd International Conference on Learning Representations*, Banff, Canada, April 2014.

[2] Ron Levie, Federico Monti, Xavier Bresson, and Michael M. Bronstein, "Cayleynets: graph convolutional neural networks with complex rational spectral filters," *IEEE Trans. Signal Process.*, vol. 67, no. 1, pp. 97–109, 2019.

[3] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proceedings of the 30th Advances in Neural Information Processing Systems*, Barcelona, Spain, December 2016, ACM.

[4] Thomas N. Kipf and Max Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the 5th International Conference on Learning Representations*, Toulon, France, April 2017.

[5] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 32, no. 1, pp. 4–24, 2021.

[6] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun, "Graph neural networks: a review of methods and applications," *arXiv preprint*, vol. arXiv:1812.08434, 2018.

[7] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang, "Dynamic network embedding : an extended approach for skip-gram based network embedding," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, July 2018, pp. 2086–2092.

[8] Yu Xie, Chunyi Li, Bin Yu, Chen Zhang, and Zhouhua Tang, "A survey on dynamic network embedding," *arXiv preprint*, vol. arXiv:2006.08093, 2020.

[9] Franco Manessi, Alessandro Rozza, and Mario Manzo, "Dynamic graph convolutional networks," *Pattern Recognit.*, vol. 97, 2020.

[10] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *The proceedings of 25th ICONIP*, Siem Reap, Cambodia, December 2018, pp. 362–373, Springer.

[11] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leiserson, "Evolvegcn: evolving graph convolutional networks for dynamic graphs," in *The Proceedings of the 34th Conference on Artificial Intelligence*, New York, USA, February 2020, pp. 5363–5370, AAAI Press.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, "Attention is all you need," in *Proceedings of the 31st Conference on Neural Information Processing Systems*, Long Beach, USA, December 2017, pp. 5998–6008.

[13] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio, "Graph attention networks," in *Proceedings of the 6th International Conference on Learning Representations*, Vancouver, Canada, April 2018.

[14] Thomas N. Kipf and Max Welling, "Variational graph auto-encoders," *CoRR*, vol. abs/1611.07308, 2016.

[15] S. V. Sai Santosh and Sumit Jagdish Darak, "Intelligent and reconfigurable architecture for KL divergence-based multi-armed bandit algorithms," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 68, no. 3, pp. 1008–1012, 2021.

[16] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu, "Dyngem: deep embedding method for dynamic graphs," *arXiv preprint*, vol. arXiv:1805.11273, 2018.

[17] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo, "Dyngraph2vec: capturing network dynamics using dynamic graph representation learning," *Knowl. Based Syst.*, vol. 187, 2020.