Dynamic-GTN: Learning an Node Efficient Embedding in Dynamic Graph with Transformer

Anonymous

Anonymous
Anonymous

Abstract. Graph Transformer Networks (GTN) use an attention mechanism to learn the node representation in a static graph and achieves state-of-the-art results on several graph learning tasks. However, due to the computation complexity of the attention operation, GTNs are not applicable to dynamic graphs. In this paper, we propose the Dynamic-GTN model which is designed to learn the node embedding in a continous-time dynamic graph. The Dynamic-GTN extends the attention mechanism in a standard GTN to include temporal information of recent node interactions. Based on temporal patterns interaction between nodes, the Dynamic-GTN employs an node sampling step to reduce the number of attention operations in the dynamic graph. We evaluate our model on three benchmark datasets for learning node embedding in dynamic graphs. The results show that the Dynamic-GTN has better accuracy than the state-of-the-art of graph neural networks on both transductive and inductive graph learning tasks.

Keywords: Graph Transformer Network · Dynamic Graph · Node Sampling.

1 Introduction

In recent years, Graph Neural Networks (GNNs) have gained a lot of attention for learning in graph-based data such as social networks, author-papers in citation networks, user-item interactions in e-commerce and protein-protein interactions. The main idea of GNNs is to find a mapping of the nodes in the graph to a latent space, which preserves several key properties of the graphs. Given that every single node has a certain influence on its neighbors, node embedding is created by GNNs based on a message passing mechanism to aggregate information from the neighborhood nodes, which can be used for downstream tasks such as node classification, edge prediction, or graph classification.

The embedding learned by traditional GNN methods can describe the local and global structures on a static graph with the constraint that the graph's nodes and edges do not change over time. For online systems such as social networks or e-commerce, this assumption usually does not hold. In order to deal with dynamic graphs, one could employ a snapshot-based approach. More specifically, a GNN model such as Graph Convolution Network (GCN) [1], Graph Attention

Network (GAT) [2], or Graph Transformer [3] is trained to learn the graph representation at a specific timestamp. The drawback of this approach is that the learned representation at each snapshot ignores the temporal interactions because each models is trained separately. The trained embedding model in this case can only capture the graph specific structures at the end of a time interval. In addition to that, the snapshot-based approach is a time-consuming one because it has to retrain the model from scratch.

Dynamic-based graph learning methods overcome these issues by learning both temporal and structural properties of the dynamic graph. Recent works can be classified into discrete-time approaches and continuous-time approaches. Discrete-time methods improve the snapshot-based approach by adding the temporal relations to the node representation. Several architectures are proposed such as DynGEM [4] with regularized weights or DySAT [5] with structural attention layers. Discrete-time methods have issues in learning the fine-grained temporal structure of the dynamic graph. Continuous-time methods avoid the issues by seeing the dynamic graph as a sequence of nodes' interaction with a timestamp. Then, a sequence learning network is employed to extract the temporal pattern of interactions. For example, RNN is used in DeepCoevolve [6] and LSTM is used in Temporal Dependency Interaction Graph [7].

Although continuous-time approaches are more natural in learning temporal information in dynamic graphs than the discrete ones, they still have significant drawbacks. The usage of RNN-like architectures to aggregate information from temporal neighbors are unable to capture long-term dependencies. When the temporal information spreads over a long period of time, the learnt dynamic representations usually degrade. Secondly, these approaches usually compute dynamic embeddings of the two target interactions nodes independently without taking into account the semantic relatedness between their temporal regions (i.e. historical behaviors), which could be a causal element for the target interactions.

To address the above limitations, in this work, we extend the graph transformer network to capture the long-term dependencies of temporal interactions between nodes in the dynamic graphs. We introduce a Time Projection layer which is added after the standard transformer layer. Firstly, the multi-head attention layer is used to aggregate both time-based node interactions and local structures of the graph. Then, the projection layer uses node embedding with temporal interactions to predict the future node representation of the graph. In order to reduce the computing complexity of the multi-head attention layer, a node sampling component is added based on the dynamic embedding of the projection layer. The attention operation only includes similar nodes which are defined by a clustering process on the node embedding. We evaluate our model on three time-dynamic graph datasets: Wikipedia, Reddit, and MOOC [8]. The experiments show that our proposed Dynamic-GTN could improve the overall accuracy of downstream tasks, and also reduce the computational time of the model.

2 Related works

The existing modeling approaches are roughly divided into two categories based on how the dynamic graph is constructed: discrete-time methods and continuous-time methods.

Discrete-time Methods: This category of methods deals with a sequence of discretized graph snapshots that coarsely approximate a time-evolving graph. DynGEM [4] is an auto-encoding method that minimizes reconstruction loss and learns incremental node embeddings from previous time steps. DySAT [5] computes dynamic embeddings by employing structural attention layers on each snapshot, followed by temporal attention layers to capture temporal variations among snapshots, as inspired by the self-attention mechanism. EvolveGCN [9] recently leverages RNNs to regulate the GCN model (i.e., network parameters) at each time step in order to capture the dynamism in the evolving network parameters. Regardless of progress, snapshot-based methods will always fail to capture fine-grained temporal and structural information due to the coarse approximation of continuous-time graphs. It is also difficult to specify an appropriate aggregation granularity.

Continuous-time Methods: Methods in this category operate directly on time-evolving graphs without time discretization and focus on designing various temporal aggregators to extract information. The dynamic graphs are represented as a series of chronological interactions with precise timestamps. DeepCoevolve [6] and it's variant JODIE [8] see two coupled RNNs to update dynamic node embeddings based on each interaction. They provide an implicit way to construct the dynamic graph in which only the historical interaction information of the two involved nodes of the interactions at time t is used. TDIG-MPNN [7] provides a graph creation approach called Temporal Dependency Interaction Graph (TDIG), which generalizes the above implicit construction and is formed from a sequence of cascaded interactions to explicitly leverage the topology structure of the temporal graph. To acquire the dynamic embeddings, they use a graph-informed Long Short Term Memory (LSTM) [10] based on the topology of TDIG.

Recent work such as TGAT [12] and TGNs [13] use a different graph creation technique, namely a time-recorded multi-graph, which allows for more than one interaction (edge) between two nodes. A single TGAT layer is used to collect one-hop neighborhoods, similar to the encoding process in static models (e.g.. GraphSAGE [18]). By stacking numerous layers, the TGAT model can capture high-order topological information. TGNs generalize TGAT's aggregation and use a node-wise memory to keep track of long-term dependencies.

Node sampling: Node sampling or graph pooling in GNN is often used to reduce the computing complexity in the aggregate. The idea to connect between graph learning and local node structures is not new. In [15], they arrange the nodes into a binary tree to fast pool adjacent nodes. The GraphSAGE [18] framework defines a neighborhood set with a fixed number of nodes to reduce the computational footprint. By exploiting the graph clustering structure, the authors propose a novel GCN training algorithm, namely Cluster_GCN [16].

The Cluster_GCN restricts the neighborhood search into a sub-graph in each learning batch. The sub-graphs are split from the original graph by a graph clustering algorithm. Our work is motivated by the work of Cluster_GCN. Instead of defining the learning batches for updating the graph cluster, we utilize the time step in a time-dynamic graph to define a learning batch.

3 Continuous-time Dynamic Graph

We define a dynamic continuous graph as $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ consists of node set $\mathcal{V}_t$ and an set of edges $\mathcal{E}_t$ ordered by time $t \in \mathbb{R}^+$ and described chronological interactions up to time t. An interaction appearing at time t is denoted as $e_{u,v,t}$, where nodes $u, v \in \mathcal{V}_t$ are two nodes involved in this interaction, $e_{u,v,t}$ has features extract from the interaction between two nodes. One node can have multiple interactions at different time points, we let $u(t)$ represent the node u at time t.

Since t can also indicate the order of interactions between two nodes, by recording the time or order of each edge, a dynamic graph can capture the evolution of the relationship between nodes. Given the topology of a graph $\mathcal{G}_t$, dynamic graph embedding aims to learn mapping function at time t:

$$f_t : \mathcal{V}_t \to \mathbb{R}^d,$$

where d is the number of node embedding dimensions. As long as the correctness of node representation in latent space, the downstream tasks such as node classification, and link prediction will more benefit from it. With interaction nodes $u(t)$ and $v(t)$, i.e., $h_{u(t)}$, $h_{v(t)}$ are node embedding of u, v at time t.

For example, Fig 1 shows a graph evolve with time, which describes interactions between users and items. Given an ordered sequence of temporal node interactions at time $0 < t_1 < t_2 < t_3 < t$, the target is learning embedding of node u at time t: $u(t)$ (square symbol). And uses the previous observed state $u(t)$ and the elapsed time Δt to predict the future embedding of the node at $t + \Delta t$. For each node, its dynamic associated nodes and their neighbor from a graph structure, which includes more time/order information than conventional static graphs. It is not trivial to encode the preference of each user from this dynamic graph.

4 Graph Transformer Network for Continuous-time Dynamic Graph

Our proposed model, Dynamic-GTN, works on the chronological interactions between two nodes in the continuous-time dynamic graph. It includes three major components as illustrated in Fig. 2:

- *Node sampling*: A sampled subgraph of an original graph G should obtain a good sample quality. The goal of this component is to find a better way to

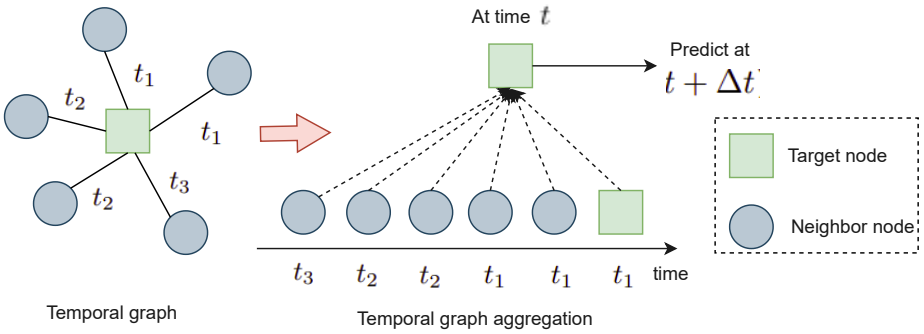

Fig. 1. Illustration of the temporal graph aggregation and label prediction with continuous time event

evaluate the entire sample clustering process which integrates node sampling with clustering. Node sampling base on cluster can remove the edges with high similarity centrality and then optimize the calculation of multi-head attention steps in Graph Transformer.

– *Graph Transformer Network and Time Projection layer*: the Graph Transformer Network (GTN) layer is used to aggregate both continuous-time embedding and structural information of the graph. Output embedding from the GTN layer is used to project the self-node to the future embedding by the Time Projection layer. The resulting embedding are used for improving the node sampling and representing as dynamic embedding for the Prediction Layer.
– *Output layer*: it utilizes output embedding from the Time Projection layer to calculate the target values. In Fig 2, the link prediction task is computed by concatenating the output of two related nodes. In the node classification task, we could omit the Concatenation layer and feed the embedding into the feed-forward layer directly.

4.1 Node sampling

At the first block, we employ a node sampling method based on cluster with dynamic information to extract relevant nodes based on the latent space of the graph. This component allows the Graph Transformer to learn different graph attention kernels for different regions based on a gradient-based self-clustering assignment such that different regions are treated differently in spatial dependency modeling.

First, a vertex-level soft-assignment to M clusters is learnt from the temporal pattern of each vertex. To partition the graph, we employ graph clustering methods. Node sampling component try to build partitions over the vertices in the graph such that within-cluster ties are significantly more than between-cluster links in order to better represent the graph's clustering and community

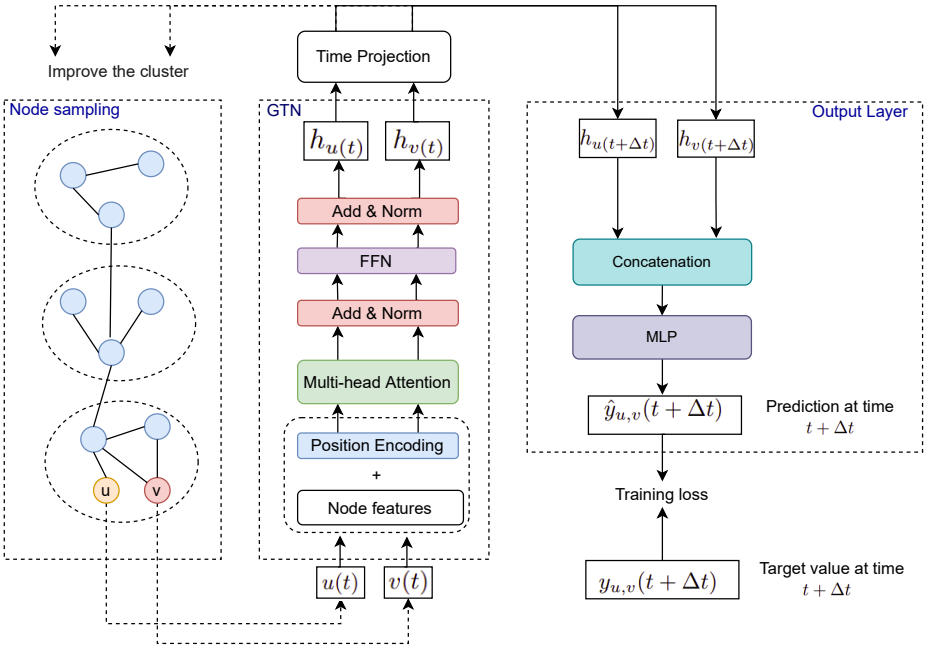

Fig. 2. Illustration of the architecture of the proposed model

structure. This is precisely what we require because: As previously stated, the embedding usage for each batch is equal to within-cluster linkages. Intuitively, each node and its neighbors are usually in the same cluster, hence neighborhood nodes with a high chance of being in the same cluster after a few hops are still in the same cluster.

$$C = \sigma_s \left(\sigma_r \left(h_{i(t)} \mathbf{W}_f \right)_t \mathbf{W}_t \right), \tag{1}$$

where C is the cluster assignment score for each vertex to M clusters. $\mathbf{h}_{i(t)}$ represent embedding of node i at time t and $\mathbf{W}_t$ is parameters for linear layers on the feature mode and temporal mode, respectively, and σ_r and σ_s represent the relu and softmax activation functions. The feature dimension of input tensor $h_{i(t)}$ is first squeezed to 1 using $\mathbf{W}_f$, in order to provide a summarized temporal pattern at each vertex. The $\mathbf{W}_t$ is further applied to the temporal pattern to calculate a M-dimensional cluster assignment score.

At the beginning, i.e at time $t = 0$, the output embedding from the Time Projection layer is not available. Therefore, the Dynamic-GTN uses the default node embedding PE for clustering the nodes as the initial clusters.

4.2 Graph Transformer Network

Observing the benefits of the Transformer in capturing long-term dependencies and in computational effort, we propose to extract temporal and structural infor-

mation of dynamic graph by Transformer type of architecture. Thus, We use the Graph Transformer to aggregate information from neighbor nodes, and it will derive information from both spatial as well as temporal features. An importance of using Transformer in graph is that we need to have position encoding (PE) to feed as an input in Transformer Encoder layer. Several works introduce PE to Transformer-based GNNs to help model capture the node position information. We use Laplacian PE is employed in [23], the authors prove that it performs better than other PE. To enhance node's positional information, we also employ time intervals that usually convey important behavior information.

Dynamic node embedding: Firstly, we update the hidden feature h of the i ́th node in a graph from layer l to layer $l+1$ at time t when there is a interaction of node i as follows:

$$h_i^{\ell+1}(t) = \sum_{j \in \mathcal{N}(i)} w_{ij} \left(V^\ell h_j^\ell \right) \tag{2}$$

where

$$w_{ij} = \text{softmax}_j \left(\frac{Q^\ell h_i^\ell \cdot K^\ell h_j^\ell}{\sqrt{d}} \right) \tag{3}$$

and $j \in \mathcal{N}(i)$ denotes the set of neighbor nodes of node i in graph and Q^ℓ, K^ℓ, V^ℓ are learnable linear weights (denoting the Query, Key and Value for the attention computation, respectively). $\mathcal{N}(i)$ is neighborhood of node i evolve by time and after node or edge event such as create a new node, delete/edit edge $\mathcal{N}(i)$ can be change, also have many version of interactions, thus we formulate neighbor of node i at time t as $\mathcal{N}_t(i)$, which describes in Fig 2. The method uses for sampling neighborhoods is cluster-based sampling as we introduced in the previous section. The attention mechanism is performed parallelly for each node in the neighbor nodes to obtain their updated features in one shot—another plus point for Transformers over RNNs, which update features node-by-node.

Multi-head Attention: Getting this straightforward dot-product attention mechanism to work proves to be tricky. Bad random initializations of the learnable weights can destabilize the training process. We can overcome this by parallelly performing multiple 'heads' of attention and concatenating the result (with each head now having separate learnable weights):

$$\hat{h}_i^{\ell+1}(t) = O_h^\ell \|_{k=1}^H \left(\sum_{j \in \mathcal{N}_i} w_{ij}^{k,\ell} V^{k,\ell} h_j^\ell \right), \tag{4}$$

where,

$$w_{ij}^{k,\ell} = \text{softmax}_j \left(\frac{Q^{k,\ell} h_i^\ell \cdot K^{k,\ell} h_j^\ell}{\sqrt{d_k}} \right), \tag{5}$$

and $Q^{k,\ell}, K^{k,\ell}, V^{k,\ell}$ are the learnable weights of the kth attention head and O^ℓ is a down-projection to match the dimensions of $h_i^{\ell+1}$ and h_i^ℓ across layers. The

attention outputs $\hat{h}_i^{\ell+1}(t)$ are then passed to a Feed Forward Network (FFN) preceded and succeeded by residual connections and normalization layers, as:

$$z_i^{\ell+1}(t) = \text{Norm}\left(h_i^{\ell}(t) + \hat{h}_i^{\ell+1}(t)\right), \tag{6}$$

$$\hat{z}_i^{\ell+1}(t) = W_2^{\ell}\,\text{ReLU}\left(W_1^{\ell} z_i^{\ell+1}(t)\right), \tag{7}$$

$$h_i^{\ell+1}(t) = \text{Norm}\left(z_i^{\ell+1}(t) + \hat{z}_i^{\ell+1}(t)\right), \tag{8}$$

where $W_1^{\ell}, W_2^{\ell}, z_i^{\ell+1}(t), \hat{z}_i^{\ell+1}(t)$ denote intermediate representations, and Norm can either be LayerNorm or BatchNorm.

Time Projection: Our proposed model projects the embedding to capture temporal information, and predicts the future embedding at a time. After a short duration Δ_t the node i's projected embedding is update to as follow:

$$h_{i(t+\Delta t)} = (1 + w) * h_{i(t)} \tag{9}$$

where w is time-context vector is converted from Δt by using a linear layer: $w = W_p \Delta t$. The vector $(1 + w)$ works as a temporal attention vector to scale the past node embedding.

4.3 Output layer

In the link prediction task, The interaction of two nodes u and v at time $t + \Delta t$ for link prediction task represent by:

$$\hat{y}_{u,v}(t + \Delta t) = W * (h_{u(t+\Delta t)} \parallel h_{v(t+\Delta t)}) + b \tag{10}$$

To learn model parameters, we optimize the cross entropy loss. The objective function $\mathcal{L}$ is defined follows:

$$\mathcal{L} = -\sum_{\mathcal{S}} \mathbf{y}_{uv} \log\left(\hat{\mathbf{y}}_{uv}\right) + (1 - \mathbf{y}_{uv}) \log\left(1 - \hat{\mathbf{y}}_{uv}\right) + \lambda \|\Theta\|_2 \tag{11}$$

where $\mathcal{S}$ denotes the training samples, $\mathbf{y}_{uv}$ is input interaction of node u and node v and $\hat{\mathbf{y}}_{uv}$ is the predicted interaction of node u and node v from the classification layer of the model.
In the node classification task, we could directly use the embedding in 9 without the concatenation layer for predicting the label of a specific node at time $t + \Delta t$.

5 Experiments

5.1 Datasets

For testing our proposed Dynamic-GTN model, we use three popular time-continous dynamic graph datasets: Wikipedia, Reddit, and MOOC, these datasets

public in [8]. These datasets consist of one month of interaction between user and item (i.e., MOOC: MOOC online course, Reddit: post, Wekipedia: page). The detail statistics of each dataset is described in Table 1. We evaluate the efficiency of our model output embedding on both transductive and inductive settings. Our experiments follow the setting in [13] in continuous-time graph learning.

More specifically, we split the data by time for training, validating and testing. We use the first 70% interaction to train, next 15% to evaluate, and the final 15% to test. For example, on Reddit dataset consist of four weeks of posts created by users on subreddits, in a week the models take the first 5 days data of week to train, the next day to evaluate, and the last day to test. The fixed evaluation period is selected at one week duration. Because our proposed model can learn continuously, the duration could be changed freely.

Table 1. Statistics of the datasets used in our experiments

Information	**MOOC**	**Reddit**	**Wikipedia**
#Nodes	7,144	10,984	9,227
#Edges	411,749	672,447	157,474
#Dynamic Nodes	4,066	366	217
Nodes' Label Type	course dropout	posting ban	editing ban

5.2 Baseline

In the transductive edge prediction and inductive node classification, we use the state-of-the-art algorithms for representation learning on temporal graphs as baselines: Discrete-Time Methods: EvolveGCN [9] and DySAT [5]; Continuous-Time Methods: JODIE [8] , TGAT [12], DyRep [11], and TGN [13] for comparison.

Evaluation metric: With future link prediction task, given an interaction $e_{u,v,t}$ each method outputs calculate the node u's preference score over node v at time t in test set. This score is used to classify if there is a connection between two nodes at time t. To evaluate the performance of the proposed method and baseline we use average precision for future edge prediction task in transductive setting. In the node classification task, we aim to represent a node u at time t as $u(t)$, and base on this representation these model prediction status of node u at time t. Accuracy is used to measure the achievement of methods.

5.3 Performance

We implement our method in PyTorch. For the other methods, we use all the original papers' code from their github pages. For all the methods we use the

Adam optimizer with learning rate as 0.01, dropout rate as 20%, weight decay as zero. The mean aggregator proposed by TGN is adopted and the number of hidden units is the same for all methods. All the results were averaged over 10 runs. For Dynamic-GTN, the number of partitions and clusters per batch for each dataset are listed in Table 4 and we show that graph clustering only takes a small portion of preprocessing time. Note that clustering is seen as a preprocessing step and its running time is not taken into account in training.

Table 2 and Table 3 shows the performance results on dynamic node classification task and future link prediction task, respectively. In general, the continuous-time methods perform better than the discrete-time methods. This can be explained by the fact that continuous-time methods can access to a more fine-gained temporal and structural information. Built on continuous-time approach, our model Dynamic-GTN outperforms all the competitors on all the datasets. The improvements are stable across the two down stream tasks. The nearest competitor to our model is the TGN architecture. By combining the time-based embedding with the self-attention operation, our model likely captures more interaction information than the compared baselines without the need to retrain the models.

Table 2. The performance of our model and base line on node classification task

Method	Model	MOOC	Wikipedia	Reddit
Discrete-time	**EvolveGCN**	70.26 ± 0.5	63.41 ± 0.3	81.77 ± 1.2
	DySAT	72.11 ± 0.5	61.79 ± 0.3	74.82 ± 1.2
Continuous-time	**Jodie**	73.39 ± 2.1	61.23 ± 2.5	84.35 ± 1.2
	TGAT	74.23 ± 1.2	65.43 ± 0.7	83.12 ± 0.7
	DyRep	75.12 ± 0.7	62.79 ± 2.3	84.82 ± 2.2
	TGN	77.47 ± 0.8	67.11 ± 0.9	87.41 ± 0.3
	Dynamic-GTN (ours)	**78.13 ± 0.9**	**69.74 ± 1.3**	**89.03 ± 0.3**

Table 3. The performance of our model and base line on link prediction task

Method	Model	MOOC	Wikipedia	Reddit
Discrete-time	**EvolveGCN**	78.33 ± 0.3	89.71 ± 0.5	80.79 ± 0.4
	DySAT	74.05 ± 0.4	88.13 ± 0.5	87.23 ± 0.4
Continuous-time	**Jodie**	76.34 ± 0.5	90.74 ± 0.3	79.11 ± 0.4
	TGAT	75.36 ± 0.5	92.87 ± 0.3	87.42 ± 0.2
	DyRep	73.45 ± 0.4	92.21 ± 0.3	86.89 ± 0.4
	TGN	81.20 ± 0.6	92.37 ± 0.2	88.17 ± 0.2
	Dynamic-GTN (ours)	**84.42 ± 0.5**	**93.71 ± 0.3**	**89.69 ± 0.2**

5.4 Discussion

We perform further experiments to highlight different components of our propose Dynamic-GTN for learning an efficient node representation in dynamic graphs.

Impact of Dynamic-GTN in long period: We test the accuracy of our proposed model by varying the time projecting window Δt. The node classification task results on Reddit dataset of our model and other baselines are shown in Table 4. In general, it is more difficult to predict for a long period updating time Δt than the short one. While all of the tested models drop accuracy, our model still achieve the best accuracies. At the longest $\Delta t = 7$, the proposed Dynamic-GTN achieves around 85.36% accuracy. The second highest accuracy is the TGN with 82.53% accuracy. This demonstrates that our architectures is more stable on learning node representation in dynamic graphs.

Table 4. The accuracy of node classification task on Reddit dataset by varying the time projection $\Delta t(days)$ of different models

Model	$\Delta t = 1$	$\Delta t = 3$	$\Delta t = 5$	$\Delta t = 7$
EvolveGCN	81.77 ± 1.2	70.39 ± 0.7	71.22 ± 0.5	74.07 ± 0.5
DySAT	82.32 ± 0.7	75.13 ± 0.5	74.05 ± 0.4	71.39 ± 0.5
Jodie	84.35 ± 1.2	81.71 ± 0.8	81.13 ± 0.5	79.38 ± 0.7
TGAT	83.12 ± 0.7	84.46 ± 0.5	83.18 ± 0.7	78.59 ± 1.2
DyRep	84.82 ± 2.2	80.33 ± 0.5	81.05 ± 0.5	79.77 ± 1.1
TGN	87.41 ± 0.3	87.58 ± 0.5	86.11 ± 0.3	82.53 ± 0.5
Dynamic-GTN (ours)	**89.03 ± 0.3**	**88.11 ± 0.2**	**86.43 ± 0.5**	**85.36 ± 0.7**

Impact of node sampling: To evaluate the effects of node sampling step with temporal information, we iterate the number of clustering components and compare the accuracy and run time performance against the baseline architecture. Table 5 compares three different node partitioning and model without clustering. The usage of clustering could improve both accuracy and training time. From our experimental results, the optimal number of clusters depend heavily on the temporal and local structures of the graph. More investigation should be done in future works to have a more accurate estimation of the number.

Table 5. The training time and the Accuracy (%) of node sampling component in Dynamic-GTN, testing on node classification task with Reddit dataset. The average time is reported per epoch with lower is better.

Model	Avg. time (s)	Accuracy
Dynamic-GTN (10 cluster)	**50.23**	90.67
Dynamic-GTN (15 clusters)	52.37	**90.81**
Dynamic-GTN (20 clusters)	52.58	90.08
Dynamic-GTN (w/o node sampling)	75.83	89.72

Impact of the number of attention head number: As the number of attention head plays an important role in projecting between consecutive latent spaces, we perform further experiments to test how it affects the performance on down stream tasks. We plot the test accuracy on MOOC dataset with different number of heads in Fig 3. It follows from [24] that the best performance can be achieved with 3 layers and 2 heads (6 effective heads). We observe that on MOOC dataset, the performance improves when the head number increases from 1-4, which demonstrates the effectiveness of multi-head attention. However, when the number of head turn over 5, there are downturn in accuracy due to the possible over-fitting problem.

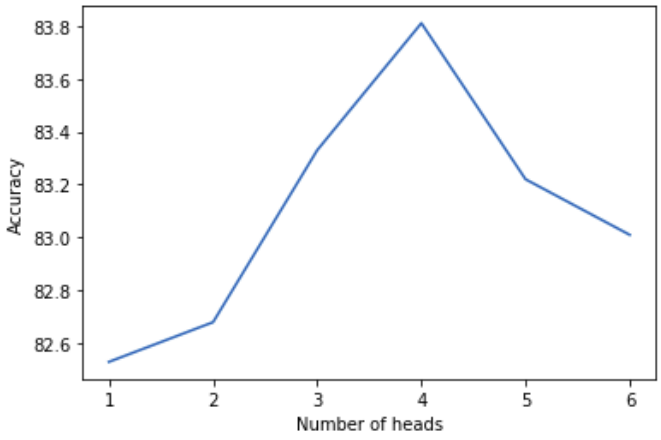

Fig. 3. The comparison of number head attention in Dynamic-GTN on MOOC's node classification

6 Conclusion

In this paper, we propose a continuous-time dynamic graph representation learning method, called Dynamic-GTN. Dynamic-GTN generalizes the Graph Transformer Network (GTN) to extract temporal-based local structure information on dynamic graphs via node embedding projection. Due to the cost computation in sampling graph in the temporal network, we utilize a cluster-based sampling to help model to train faster both in inductive and transductive learning. Several experiments are made to evaluate the characteristics of our proposed architecture. The overall results on three benchmark datasets show that our model achieves better performance than previous state-of-the-art GCN models on continuous-time graphs.

References

1. Hamilton, William L., Rex Ying, and Jure Leskovec. "Inductive representation learning on large graphs." In Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 1025-1035. 2017.
2. Veličković, Petar, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. "Graph attention networks." arXiv preprint arXiv:1710.10903 (2017).
3. Cai, Deng, and Wai Lam. "Graph transformer for graph-to-sequence learning." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 05, pp. 7464-7471. 2020.
4. Goyal, Palash, Nitin Kamra, Xinran He, and Yan Liu. "Dyngem: Deep embedding method for dynamic graphs." arXiv preprint arXiv:1805.11273 (2018).
5. Sankar, Aravind, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. "Dysat: Deep neural representation learning on dynamic graphs via self-attention networks." In Proceedings of the 13th International Conference on Web Search and Data Mining, pp. 519-527. 2020.
6. Dai, Hanjun, Yichen Wang, Rakshit Trivedi, and Le Song. "Deep coevolutionary network: Embedding user and item features for recommendation." arXiv preprint arXiv:1609.03675 (2016).
7. Chang, Xiaofu, Xuqin Liu, Jianfeng Wen, Shuang Li, Yanming Fang, Le Song, and Yuan Qi. "Continuous-Time Dynamic Graph Learning via Neural Interaction Processes." In Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp. 145-154. 2020.
8. Kumar, Srijan, Xikun Zhang, and Jure Leskovec. "Predicting dynamic embedding trajectory in temporal interaction networks." In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1269-1278. 2019.
9. Pareja, Aldo, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. "Evolvegcn: Evolving graph convolutional networks for dynamic graphs." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 04, pp. 5363-5370. 2020.
10. Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9, no. 8 (1997): 1735-1780.
11. Trivedi, Rakshit, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. "Dyrep: Learning representations over dynamic graphs." In International conference on learning representations. 2019.
12. Xu, Da, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. "Inductive representation learning on temporal graphs." arXiv preprint arXiv:2002.07962 (2020).
13. Rossi, Emanuele, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. "Temporal graph networks for deep learning on dynamic graphs." arXiv preprint arXiv:2006.10637 (2020).
14. Ying, Rex, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. "Hierarchical graph representation learning with differentiable pooling." arXiv preprint arXiv:1806.08804 (2018).
15. Defferrard, Michaël, Xavier Bresson, and Pierre Vandergheynst. "Convolutional neural networks on graphs with fast localized spectral filtering." Advances in neural information processing systems 29 (2016).

16. Chiang, Wei-Lin, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks." In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 257-266. 2019.
17. Karypis, George, and Vipin Kumar. "A fast and high quality multilevel scheme for partitioning irregular graphs." SIAM Journal on scientific Computing 20, no. 1 (1998): 359-392.
18. Hamilton, Will, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs." Advances in neural information processing systems 30 (2017).
19. He, Xiangnan, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. "Lightgcn: Simplifying and powering graph convolution network for recommendation." In Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, pp. 639-648. 2020.
20. Fan, Wenqi, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. "Graph neural networks for social recommendation." In The world wide web conference, pp. 417-426. 2019.
21. You, Jiaxuan, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. "Graph convolutional policy network for goal-directed molecular graph generation." Advances in neural information processing systems 31 (2018).
22. Jiang, Mingjian, Zhen Li, Shugang Zhang, Shuang Wang, Xiaofeng Wang, Qing Yuan, and Zhiqiang Wei. "Drug–target affinity prediction using graph neural network and contact maps." RSC Advances 10, no. 35 (2020): 20701-20712.
23. Dwivedi, Vijay Prakash, and Xavier Bresson. "A generalization of transformer networks to graphs." arXiv preprint arXiv:2012.09699 (2020).
24. Ma, Xutai, Juan Pino, James Cross, Liezl Puzon, and Jiatao Gu. "Monotonic multihead attention." arXiv preprint arXiv:1909.12406 (2019).