Pre-Training on Dynamic Graph Neural Networks

Jiajun Zhang[1] , **Kejia Chen**[2*] , **Yunyun Wang**[3] ,

[123]Nanjing University of Posts and Telecommunications

hand_zjj@foxmail.com, {chenkj, wangyunyun}@njupt.edu.cn

Abstract

The pre-training on the graph neural network model can learn the general features of large-scale networks or networks of the same type by self-supervised methods, which allows the model to work even when node labels are missing. However, the existing pre-training methods do not take network evolution into consideration. This paper proposes a pre-training method on dynamic graph neural networks (PT-DGNN), which uses dynamic attributed graph generation tasks to simultaneously learn the structure, semantics, and evolution features of the graph. The method includes two steps: 1) dynamic sub-graph sampling, and 2) pre-training with dynamic attributed graph generation task. Comparative experiments on three realistic dynamic network datasets show that the proposed method achieves the best results on the link prediction fine-tuning task.

1 Introduction

Neural networks such as CNN and RNN have achieved great success in the fields of computer vision and natural language processing. In order to explore the learning ability of neural networks in structural data such as graphs and networks, graph neural networks (GNNs) have drawn increasing attention in recent years and have achieved breakthroughs in graph mining tasks [Kipf and Welling, 2016b; Veličković *et al.*, 2017; Kipf and Welling, 2016a]. The input of a GNN is usually the graph structure and node features, and after multiple layers of message passing, the model converts the input into node-level or graph-level representation and uses them to complete downstream tasks such as node classification, link prediction, and graph classification.

The existing GNN models are often designed according to specific tasks and need to be retrained if used for any new task and data, which will consume extra time and resources. Referring to the success of pre-training in the field of NLP [Devlin *et al.*, 2018] and CV [Chen *et al.*, 2020], it is believed that pre-training on graph data is also beneficial. Some starting methods have verified the advan-

tages of graph pre-training in the contexts of transfer learning [Hu *et al.*, 2019a], contrastive learning [Qiu *et al.*, 2020] and graph generation [Hu *et al.*, 2020]. Most pre-training methods are self-supervised, which can work on large-scale networks regardless of whether the nodes are labeled or not.

However, the above methods mainly focus more on the structural features at node level or graph level, rather than the temporal features of edge generation. The latter actually reveals the evolution of the network and can make node representations more accurate. Therefore, the incorporation of network dynamics into pre-training GNN models is a topic worthy of further research.

This paper proposes a pre-training method on dynamic graph neural networks (PT-DGNN) based on two tasks: dynamic edge generation task and node attribute generation task. In pre-training, historical edges are used to predict future edges, thus capturing the dynamic characteristics of the network.

The main contributions of this paper are: (1) In order to deal with large-scale networks, we propose a dynamic sub-graph sampling method, which can preserve the local structure and *freshness* of the original graph by sampling more newly updated nodes; (2) We propose a dynamic pre-training method based on graph generation tasks, which embeds temporal features into the capture of structural features and semantics (node attributes) of the graph for the first time; (3) The evaluation of the model after fine-tuning of the link prediction task shows that our method achieves the best prediction performance in different dynamic networks.

2 Related Work

This section reviews related work of pre-training on GNNs and dynamic network representation learning.

2.1 Pre-training on GNNs

The purpose of pre-training a GNN is to learn the parameters of the model according to different tasks, and then in a given fine-tuning task, the pre-trained model will be further optimized.

Hu W. et al. [Hu *et al.*, 2019a] proposed a pre-training method that combines the node-level and graph-level representations to train the model. At the node level, two self-supervised methods are used, namely context prediction and

---

attribute prediction. At the graph level, supervised graph-level property prediction and structural similarity prediction are used. Combining four tasks enables GNN to learn useful local and global graph representations, and finally effectively improves the performance of graph classification tasks on molecular graphs. Hu Z. et al. [Hu *et al.*, 2019b] proposed a general pre-training framework that focuses on multiple subtasks, including denoising link reconstruction, centrality score ranking and cluster preserving. The model is expected to capture the characteristics of of the graph from multiple perspectives (nodes, edges, and subgraph structure). Later, they [Hu *et al.*, 2020] proposed another framework GPT-GNN, where pre-training is performed on a large-scale single graph using a graph generation task to learn node-level structural and semantic information. Specifically, a large-scale graph is sampled to get sub-graphs, which are sent to a general GNN model for graph generative pre-training. Qie et al. [Qiu *et al.*, 2020] proposed a self-supervised GNN pre-training framework GCC to capture the universal network topological properties across multiple networks. The model designs the pre-training task as subgraph-level instance discrimination in a graph or across graphs, using contrast learning to better learn the internal and transferable structural representation. The model is finally evaluated in three graph learning tasks (node classification, graph classification, and similarity search).

The above methods many focus on the design of different pre-training tasks to learn the general characteristics of graphs. Our work pays more attention to the role of network dynamics in the pre-training process.

2.2 Dynamic Network Representation Learning

Network representation learning (also known as graph embedding) aims to learn effective low-dimensional vector representations of nodes, edges, or graphs from the network structure for downstream classification or prediction tasks. The initial methods are based on network proximity [Tang *et al.*, 2015] or statistics derived from random walks [Perozzi *et al.*, 2014; Grover and Leskovec, 2016]. But they rely on specific networks and have poor adaptability. After GCN [Kipf and Welling, 2016b] was proposed, a series of GNN models emerged, such as GAT [Veličković *et al.*, 2017], GAE [Kipf and Welling, 2016a], Graph-SAGE [Hamilton *et al.*, 2017] and Graph Info-max [Velickovic *et al.*, 2019].

Note that most networks are constantly changing over time (especially in edge generation), a series of dynamic network representation learning methods are proposed to study the influence of network dynamics on graph embedding. In DANE [Li *et al.*, 2017], the network representation at the time t is updated based on the variation of the adjacency matrix and the attribute matrix over time. The parameters to initialize model at current timestep are directly derived from the previous timestep, thus speeding up the training. DynGEM [Goyal *et al.*, 2018] splits the dynamic network into different snapshots, and uses SDNE (Structural Deep Network Embedding) [Wang *et al.*, 2016] method for each snapshot. HTNE [Zuo *et al.*, 2018] introduces Hawkes

process theory into the dynamic model, considering that the impact of neighbors on the central nodes will weaken over time, thereby dynamically learning the node representations. DyRep [Trivedi *et al.*, 2018] divides the network into two dynamic processes: association process and communication process. The former focuses on the change of network topology, while the latter deals with the change of network dynamics. DySAT [Sankar *et al.*, 2018] combines attention [Vaswani *et al.*, 2017] and GNN to learn dynamic network representation from two aspects: structural attention and temporal attention, both using the past representation of nodes to update the current representation. Unlike the above methods, CTDNE [Nguyen *et al.*, 2018] proposes a temporal random walk sequence and then uses skip-gram to get node representation. The method effectively solves the accuracy loss caused by the snapshot-based method.

The above works have verified that the quality of node representation can been improved after learning from network dynamics.

3 Pre-training on Dynamic GNN

This section details the proposed pre-training method based on dynamic attributed graph generation, to better embed temporal features into the GNN model.

3.1 Problem Description

Definition 1 (Dynamic Networks). *In a dynamic network, the interaction between nodes occurs at a specific time, and the network structure changes over time. In this paper, the dynamic network is formalized as a graph $G = (V, \xi_t, X)$, where V represents the node set and $\xi_t \subseteq V \times V \times R^+$ represents the edge set with timestamp $t(t \in R^+)$. Any given edge $e = (u, v, t) \in \xi_t$ represents the interaction between node u and v at time t. X denotes the matrix of node attributes.*

Definition 2 (Dynamic Network Representation Learning). *For a dynamic network $G = (V, \xi_t, X)$, the task of dynamic network representation learning is to learn a mapping function $f_\theta : V \to R^d$, which outputs the node features in a low-dimensional space. The node representation may contain multitype of information, such as structure, semantics (node attributes) and evolution (time sequence of edges).*

With the success of self-supervised learning, the above multiple types of information can be better learned by pre-training a GNN model.

Problem 1 (Pre-Training on Dynamic GNN). *In this paper, pre-training on dynamic GNN uses dynamic graph generation tasks that take into account the edge generation time, to learn general features (including evolutionary information) from dynamic graphs. After pre-training, the parameter θ of the model f_θ is obtained.*

3.2 The PT-DGNN Framework

This paper proposes a new pre-training framework, named PT-DGNN (Figure 1), which includes dynamic sub-graph sampling, dynamic attributed graph generation and fine-tuning.
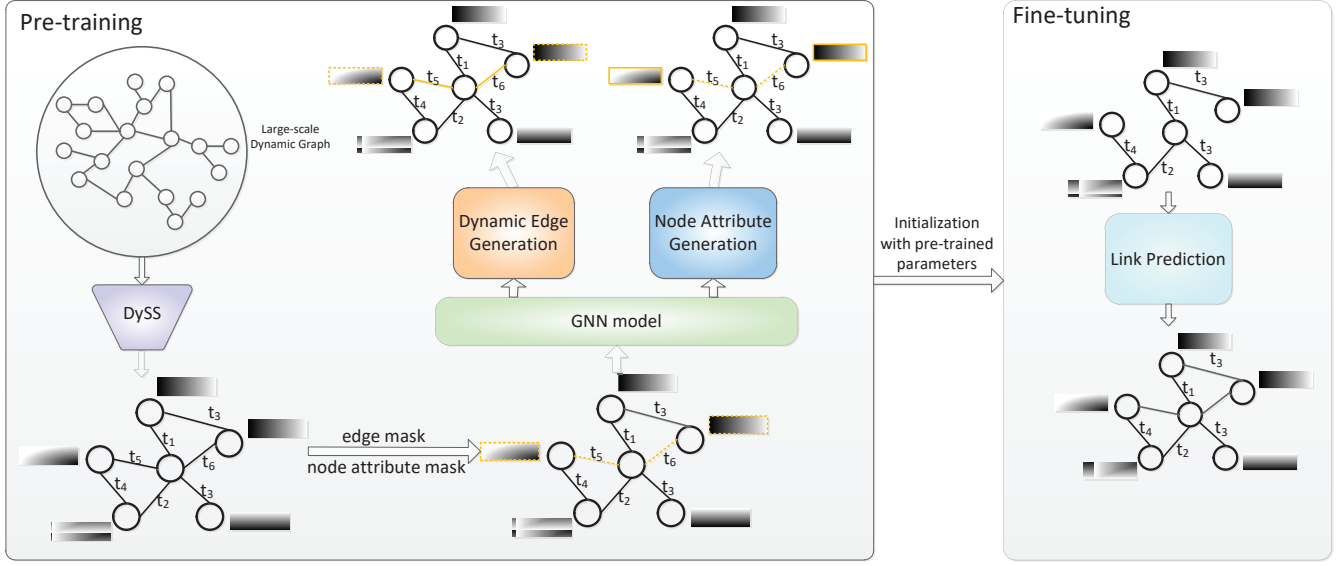
Figure 1: Overall framework of PT-DGNN

We use a single large-scale dynamic graph $G = (V, \xi_t, X)$ to train a general GNN model f_θ. After pre-training, f_θ can be fine-tuned in a specific task such as link prediction.

3.3 Dynamic Sub-graph Sampling

It is known that the GNN model is hard to be used on large-scale graphs. Inspired by LADIES [Zou *et al.*, 2019], we propose a dynamic subgraph sampling (DySS) to generate the input required for pre-training.

DySS mainly includes three steps: 1) Randomly select w nodes as the sampling initial point; 2) Put the first-order neighbors of nodes into the candidate pool and save their timestamp as weight to calculate sampling probability; 3) Select final w nodes according to sampling probability. The proposed samping method is described in Algorithm 1. The sampling depth refers to the number of sampling iterations, and sampling width refers to the number of sampling nodes in each iteration.

The sampling probability of node i can be calculated as Eq. 1.

$$Prob_s[i] = \frac{Prob_s[i]^2}{\sum_{u \in CPool} Prob_s[u]^2} \tag{1}$$

3.4 Dynamic Attributed Graph Generation

In order to learn network dynamics, we use two pre-training tasks: dynamic edge generation and node attribute generation and each node in the network must participate in these two tasks. The former uses a time-based mask method to predict node $i's$ furture edges $\xi_{i,f}$ (masked edges) with node $i's$ history edges (reserved edges) and node $i's$ attributes X_i. The latter uses the attributes of node $i's$ neighbors to predict node $i's$ attributes X_i.

The autoregressive generation process can be defined in

Algorithm 1 DySS

Input: dynamic graph DG, sampling depth d, sampling width w
Output: dynamic subgraph SG

1: Randomly select w starting nodes to SN;
2: Create a candidate pool $CPool$;
3: $Prob_s \leftarrow \{0\}$; //sampling probability of all nodes
4: $SG \leftarrow SN$;
5: **for** $i = 1$ to d **do**
6:    **for** each v in SN **do**
7:       **for** each u in Neighbor(v) **do**
8:          $CPool \leftarrow CPool \cup \{v\}$;
9:          $Prob_s[u] \leftarrow$ Time(v,u);//initialized with timestamp
10:       **end for**
11:    **end for**
12:    Update $Prob_s$ with Eq.1 v_1';
13:    Select $\{v_1', ..., v_w'\}$ according to $Prob_s$;
14:    $SG \leftarrow SG \cup \{v_1', ..., v_w'\}$;
15:    $CPool \leftarrow CPool - \{v_1', ..., v_1'\}$;
16:    $SN \leftarrow \{v_1', ..., v_1'\}$;
17: **end for**
18: **return** SG

Eq. 2.

$$p_\theta\left(X_i, \xi_i | X_{<i}, \xi_{<i}\right) \tag{2}$$
$$= \sum_h p_\theta\left(X_i, \xi_{i,f} | \xi_{i,h}, X_{<i}, \xi_{<i}\right) \cdot p_\theta\left(\xi_{i,h} | X_{<i}, \xi_{<i}\right)$$
$$= \mathbb{E}_h\left[p_\theta\left(X_i, \xi_{i,f} | \xi_{i,h}, X_{<i}, \xi_{<i}\right)\right],$$

where p_θ indicates the likelihood that uses node $i's$ history neighbors and their attributes to predict node $i's$ attributes and its future neighbors.

Algorithm 2 Time-based edge mask

Input: subgraph SG, number of masked edges N
Output: set of masked edges ME

1: $Prob_t \leftarrow \{0\};$ // mask probability of all nodes.
2: **for** each v in SG **do**
3:     **for** each u in Neighbor(v) **do**
4:         $Prob_t[u] \leftarrow$ Time(v,u);
5:     **end for**
6:     $Prob_t \leftarrow Softmax(Prob_t);$
7:     Select $\{u_1, ..., u_N\}$ according to $Prob_t$;
8:     $ME \leftarrow ME \cup \{u_1, ..., u_N\};$
9: **end for**
10: **return** ME

Assuming that the two pre-training tasks are independent, $p_\theta\left(X_i, \xi_{i,f} | \xi_{i,h}, X_{<i}, \xi_{<i}\right)$ can be simplified to Eq. 3.

$$
\begin{aligned}
&\left[p_\theta\left(X_i, \xi_{i,f} | \xi_{i,h}, X_{<i}, \xi_{<i}\right)\right] \quad\quad (3)\\
&= \left[p_\theta\left(X_i | \xi_{i,h}, X_{<i}, \xi_{<i}\right) \cdot p_\theta\left(\xi_{i,f} \Big| \xi_{i,h}, X_{\leq i}, \xi_{\leq i}\right)\right],
\end{aligned}
$$

where the first term $p_\theta\left(X_i | \xi_{i,h}, X_{<i}, \xi_{<i}\right)$ denotes the generation of attributes for node i, given the attributes of node $i's$ history neighbors. The second term $p_\theta\left(\xi_{i,f} \Big| \xi_{i,h}, X_{\leq i}, \xi_{\leq i}\right)$ refers to the prediction of future edges based on its historical neighbors and the generated node attributes.

Dynamic Edge Generation

The time sequence of edge generation often implies the evolutionary law of the network. The generation of new edges is often related to that of old edges (See Example 1).

Example 1 (Dynamic Edge Generation Example). *As shown in Figure 2, taking the collaboration network as an example, suppose that A is a scholar in the computer science field, B and C are scholars in the cross-field of computer science and music, D and E are scholars in the music field. Generally, A is unlikely to cooperate with scholars D and E in the music field unless he/she cooperates with B and C first. Therefore, the order of edge generation in the local graph should be $\{(A, B), (A, C)\} \rightarrow \{(A, D), (A, E)\}$ instead of $\{(A, D), (A, E)\} \rightarrow \{(A, B), (A, C)\}$.*

Instead of randomly masking edges in previous methods, we propose a time-sensitive edge mask method, which is based on an intuitive idea that for any node u, the edges between itself and its neighbors, if marked with newer timestamp, should be masked. The method follows the assumption that historical neighbor information should be used to predict new neighbors, not vice versa. The method is described in Algorithm 2. The edges between nodes are weighted by time probability defined in Eq. 4.

The GNN model uses the masked graph as input and generates node embedding r^E by learning from dynamic edge generation. To optimize the model, L^E is defined, where a score function $Sim\left(r_i^E, r_j^E\right), \forall i, j \in V$ is used to calculate the similarity of (r_i^E, r_j^E).
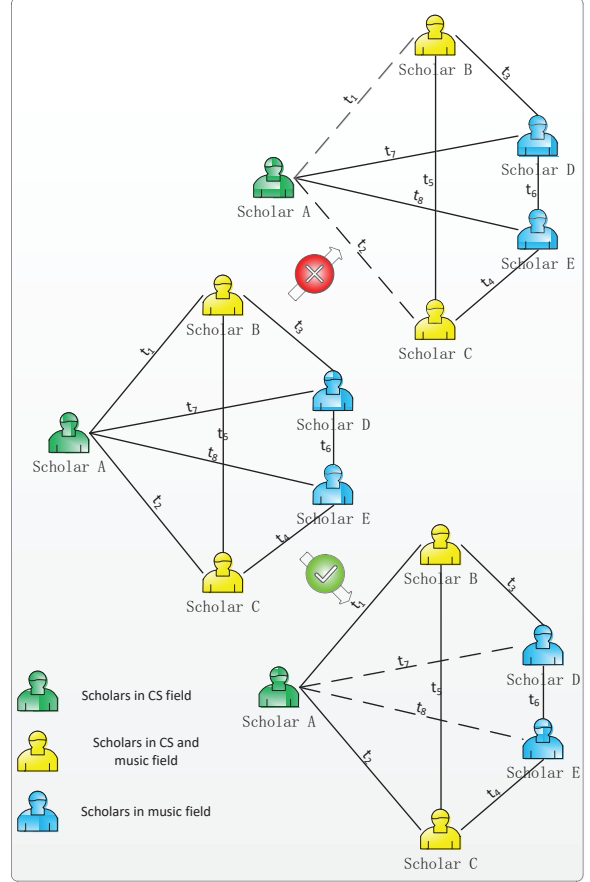

Figure 2: An illustrative example of time-based mask method

$$L^E = - \sum_{i \in V} \sum_{j^+ \in \xi_{i,f}} log \frac{exp\left(Sim\left(r_i^E, r_{j^+}^E\right)\right)}{\sum_{j \in V} exp\left(Sim\left(r_i^E, r_j^E\right)\right)} \quad (4)$$

Node Attribute Generation

In the node attribute generation task, node attributes are masked and replaced with a learnable vector X'. The GNN model uses the masked graph to generate node embedding r^A.

To optimiz the model, L^A is defined, which is L2 distance function between the generated attributes r^A and initial attributes X. Here, MLP (Multilayer Perceptron) is used as a decoder.

$$L^A = \sum_{i \in V} \left| MLP\left(r_i^A\right) - X_i \right|^2 \quad (5)$$

By minimizing L^A, the GNN model can capture the semantic information in the graph.

Only r^E is used to conduct GNN information propagation between GNN layers as the real node attributes are masked in r^A.

3.5 Fine-tuning in Link Prediction

In the fine-tuning stage, f_θ obtained after pretraining is used to initialize the GNN model and a new graph is provided to optimize the model. For link prediction task, we use a loss function L^F based on dynamic graph.

$$L^F = \sum_{i \in V} -log\left(\sigma\left(r_i^T r_j\right)\right) - Q \cdot E_{j_n \sim P_n(j)} log\left(\sigma\left(r_i^T r_{r_n}\right)\right),$$
$$(6)$$

where j is a node that co-occurs near i on a fixed-length temporal random walk [Nguyen *et al.*, 2018], σ is sigmoid function, P_n is the negative sampling distribution, and Q defines the number of negative samples.

By optimization, the connected nodes are closer in the latent space while unconnected nodes can be more distinguishable.

4 Experiments

Comparative experiments were conducted on three dynamic network datasets and the link prediction task is used to evaluate the performance of different models.

4.1 Datasets

Three realistic dynamic network datasets (HepPh, Math overflow and Superuser) were selected from the SNAP website[1].

HepPh. It is a citation network dataset in high energy physics phenomenology constructed from the e-print arXiv. It covers papers from January 1993 to April 2003. If paper i cited paper j in a certain time t, there is a directed edge with time t from node i to node j.

[1]http://snap.stanford.edu/

Math Overflow. It is a temporal network of interactions on the stack exchange website Math Overflow. A directed edge (u, v, t) indicates that user u and user v interacted at time t. The interactions include answering questions, commenting question and commenting answer.

Super User. It is another temporal network of interactions on the stack exchange web site Super User, It has the same interaction types with Math Overflow.

The statistics of the datasets are shown in Table 1.

Datasets	Nodes	Edges	Time span (days)	Types
HepPh	34,546	421,578	3,720	Directed
Math Overflow	24,818	239,978	2,350	Directed
Super User	194,095	924,886	2,773	Directed

Table 1: Statistics of the datasets

4.2 Comparison Methods

In the experiment, a total of 9 models are compared, including several static GNNs without pre-training, two dynamic graph embedding methods without pre-training, a static GNN with pre-training and a dynamic GNN with pre-training (i.e. ours).

node2vec [Grover and Leskovec, 2016]. This method uses biased random walk sequences as the input of Skip-Gram to learn the representation of nodes in the graph. It is a benchmark method for comparison.

GAE and VGAE [Kipf and Welling, 2016a]. Both are graph auto-encoders. The former uses GCN as the encoder, the latter use GCN encoder to learn the distribution of hidden node representations and both use the inner product as the decoder to reconstruct the graph.

GraphSAGE [Hamilton *et al.*, 2017]. In this model, both first-order and second-order neighbor information of nodes is used during graph convolution. The optimization goal is to make neighboring nodes have more similar representations.

GraphInfomax [Velickovic *et al.*, 2019]. This model combines the mutual information theory and tries to maximize the local-global mutual information between node embedding and graph embedding.

CTDNE [Nguyen *et al.*, 2018]. It is a dynamic network embedding model also based on random walk and Skip-Gram. But the model uses the temporal random walk, where the timestamp in the sequence is increasing.

HTNE [Zuo *et al.*, 2018]. It is also a dynamic network embedding model, introducing the Hawkes process theory. It learns the node representations based on the influence of neighbors on the central nodes will decay over time.

GPT-GNN [Hu *et al.*, 2020]. It is a pre-training model for static networks, which uses edge generation task and node attribute generation task to pre-train a single graph. Without considering the network evolution, a randomly mask method is used in edge generation task.

4.3 Experimental Setup

According to the time sequence of the edges, we split each dataset into four parts, which are used for pre-training, training, verification and testing. The data ratio of each part is 7:1:1:1. For models without pre-training, the first two parts of the data are merged as the training set.

Due to the lack of node attributes in public dynamic network datasets, we choose node2vec to generate initialize virtual attributes of the node features in the graph of the entire time span. The initialization method is commonly used in dynamic network representation learning [Singer *et al.*, 2019; Gong *et al.*, 2020].

For node2vec, CTDNE and HTNE, we use default parameters given in their paper and set the embedding dimensions to 400.

For GraphSAGE and Graph Infomax, the output dimensions of the two convolutional layers are 800 and 400 to get the best result, respectively. However, GAE and VGAE have poor performance on the above convolutional layer size settings, so we use their default settings. The other parameters of these four methods are all set by default.

For PT-DGNN and GPT-GNN, we stacked 3 layers of GCN as the basic model, and the dimension of each layer is 400. The depth and width of the sub-graph sampling are set to 6 and 128, respectively. The number of masked edges in each subgraph accounts for 50% of the total number of edges. For comparison, the fine-tuning task of GPT-GNN is also set to link prediction instead of node classification. Both pre-training models are optimized via AdamW [Loshchilov and Hutter, 2017], and 20 epochs are conducted for both pre-training and fine-tuning.

The PT-DGNN[2] is implemented by PyTorch Geometric package in NVIDIA GeForce GTX 1080Ti GPU.

4.4 Experimental Results

Comparison of Performance on Link Prediction

The link prediction task is widely used to evaluate the performance of dynamic graph embedding [Nguyen *et al.*, 2018; Gong *et al.*, 2020]. Each method was run ten times on each dataset and the average AUC values are reported in Table 2. For CTDNE, it is difficult to train with a large-scale dynamic network, so the result is missing for the Super User dataset. PT-DGNN$_{Attr}$ and PT-DGNN$_{Edge}$ are variants of PT-DGNN, using one of the two pre-training tasks. DGNN is a model without pre-training, but directly trained on sampled subgraphs.

The experimental results show that the average performance of the models after pre-training is significantly higher than those without pre-training. When comparing models without pre-training, the performance of CTDNE and HTNE is slightly higher than static GNN models, but their performance is not stable on different datasets. The dynamic pre-training method in this paper further improves the performance of the pre-trained GNN model GPT-GNN (2-3% improvement on all three datasets), which fully shows that the dynamic subgraph sampling and the time-based mask method are effective.

---

[2]https://github.com/Mobzhang/PT-DGNN/

Datasets	HepPh	Math Overflow	Super User
Method	Link Prediction (AUC)		
node2vec	0.6235	0.6250	0.5841
GAE	0.7134	0.6879	0.6985
VGAE	0.7189	0.6935	0.7064
GraphSAGE	0.5998	0.6696	0.6582
GraphInfomax	0.6203	0.6730	0.6849
CTDNE	0.7429	0.7432	—
HTNE	0.6948	0.7493	0.6965
GPT-GNN(Attr)	0.8492	0.8902	0.8373
GPT-GNN(Edge)	0.8536	0.8773	0.8413
GPT-GNN	0.8564	0.8910	0.8455
DGNN$_{np}$	**0.8586**	**0.8963**	**0.8524**
PT-DGNN(Attr)	**0.8617**	**0.9037**	**0.8671**
PT-DGNN(Edge)	**0.8684**	**0.9064**	**0.8642**
PT-DGNN	**0.8762**	**0.9196**	**0.8750**

Table 2: Comparative results in Link Prediction

Comparison of Sub-graph Sampling Methods

In PT-DGNN, two sub-graph sampling methods LADIES [Zou *et al.*, 2019] and DySS (proposed in this paper) are compared. The former only uses neighborhood structure to sample nodes, while ours also uses the timestamp of neighborhood edges. As shown in Figure 3, PT-DGNN using DySS performs superior to that using LADIES in link prediction.
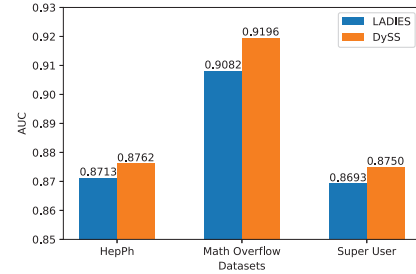

Figure 3: Performance of PT-DGNN with different sampling methods

Impact of Pre-training Data Size

We further examine the impact of the data size (the proportion of the entire dataset) of the pre-training on the final result. The results of PT-DGNN and GPT-GNN on the three datasets are shown in Figure 4(a). It shows that as the amount of pre-training data increases, the model can learn more robust parameters from it, so as to better serve the fine-tuning task.

Impact of Mask Edge Ratio

In the dynamic edge generation module of PT-DGNN, the softmax function is used to calculate the probability of the edge being masked. Moreover, the liner function is used for comparsion. The results in Figure 4(b) show that the softmax function is slightly better than the linear function in the mask strategy. We also found that as the number of masked edges increase the performance of the PT-DGNN model gradually improves. But when the ratio continues to increase, isolated

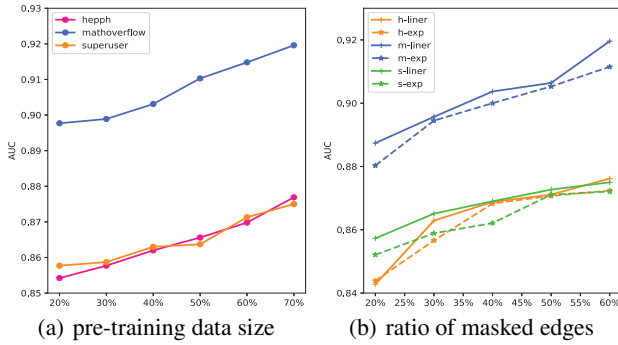nodes may be generated making it diffcult to train the model.

(a) pre-training data size   (b) ratio of masked edges

Figure 4: The impact of pre-training data size and ratio of masked edges on the results.

5 Conclusion

In this paper, a pre-training method for dynamic GNNs is proposed. Different from other graph-based pre-training methods, our method takes into account the dynamics of network evolution and combines the structural features with the temporal features of the edges in the graph generation task to learn more accurate low-dimensional representation of nodes. For this purpose, we propose a time-based edge mask method and a time-based sub-graph sampling strategy. We fine-tune the pre-trained model in link prediction on multiple datasets, and the experimental results verify the superiority of our method.

References

[Chen *et al.*, 2020] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

[Devlin *et al.*, 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[Gong *et al.*, 2020] Maoguo Gong, Shunfei Ji, Yu Xie, Yuan Gao, and AK Qin. Exploring temporal information for dynamic network embedding. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[Goyal *et al.*, 2018] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. Dyngem: Deep embedding method for dynamic graphs. *arXiv: Social and Information Networks*, 2018.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[Hu *et al.*, 2019a] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

[Hu *et al.*, 2019b] Ziniu Hu, Changjun Fan, Ting Chen, Kai-Wei Chang, and Yizhou Sun. Pre-training graph neural networks for generic structural feature extraction. *arXiv preprint arXiv:1905.13728*, 2019.

[Hu *et al.*, 2020] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1857–1867, 2020.

[Kipf and Welling, 2016a] N. Thomas Kipf and Max Welling. Variational graph auto-encoders. *CoRR*, 2016.

[Kipf and Welling, 2016b] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[Li *et al.*, 2017] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. Attributed network embedding for learning in a dynamic environment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 387–396, 2017.

[Nguyen *et al.*, 2018] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *Companion Proceedings of the The Web Conference 2018*, pages 969–976, 2018.

[Perozzi *et al.*, 2014] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[Qiu *et al.*, 2020] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1150–1160, 2020.

[Sankar *et al.*, 2018] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dynamic graph representation learning via self-attention networks. *arXiv preprint arXiv:1812.09430*, 2018.

[Singer *et al.*, 2019] Uriel Singer, Ido Guy, and Kira Radinsky. Node embedding over temporal graphs. *arXiv preprint arXiv:1903.08889*, 2019.

[Tang *et al.*, 2015] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings*

of the 24th international conference on world wide web, pages 1067–1077, 2015.

[Trivedi *et al.*, 2018] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Representation learning over dynamic graphs. *arXiv preprint arXiv:1803.04051*, 2018.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[Veličković *et al.*, 2017] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[Velickovic *et al.*, 2019] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. 2019.

[Wang *et al.*, 2016] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234, 2016.

[Zou *et al.*, 2019] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. In *Advances in Neural Information Processing Systems*, pages 11249–11259, 2019.

[Zuo *et al.*, 2018] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. Embedding temporal network via neighborhood formation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2857–2866, 2018.