Temporal Edge-Aware Hypergraph Convolutional Network for Dynamic Graph Embedding

Da Huang[1,2] and Fangyuan Lei[1,2](✉)

[1] School of Electronic and Information, Guangdong Polytechnic Normal University, Guangzhou 510665, China
leify@gpnu.edu.cn

[2] Guangdong Provincial Key Laboratory of Intellectual Property & Big Data, Guangzhou 510665, China

Abstract. Graph embedding is a critical aspect of network analysis that helps to advance various real-world applications such as social recommendation and protein structure prediction. Most of the existing graph embedding methods are designed for static graphs while many real-world graphs intrinsically behave as dynamic graphs. Recent works try to combine graph neural networks(GNN) with recurrent neural networks to address this issue. However, these methods can not independently utilize GNN models to cope with dynamic graphs and they ignore the inner edge-level correlations in dynamic graphs. To tackle these problems, we propose a novel dynamic graph embedding framework in this paper, called DynHyper. Specifically, we introduce a temporal hypergraph construction to capture the local structure information and temporal dynamics simultaneously. Then, we employ a hyperedge projection to obtain edge-level correlations. Further, we propose a temporal edge-aware hypergraph convolution to transmit and aggregate the messages in the temporal hypergraph. We conduct our experiments on seven real-world datasets to evaluate the effectiveness of DynHyper in both link prediction and node classification tasks. Experimental results show that DynHyper significantly outperforms all baselines, especially on the more complex datasets.

Keywords: Graph convolutional networks · Dynamic graph embedding · Hypergraph learning

1 Introduction

Graphs have a great capacity to model the relationship among entities, successfully applied in many fields, such as social network [10], finance analysis [15], and biological network [24]. Many academics are attempting to extend neural network models to graphs as a result of deep learning's extraordinary performance. These neural network models, also known as graph network embedding,

S. Khanna et al. (Eds.): PRICAI 2022, LNCS 13629, pp. 435–449, 2022.
https://doi.org/10.1007/978-3-031-20862-1_32

have emerged as a prominent method for graphs. The key idea of graph network embedding is to map node representation into a low-dimensional latent space, which preserves the similarity of nodes based on their local structure. These graph network embedding algorithms have been used by numerous academics for a variety of applications, including node classification, link prediction, and network visualization [1,3,5,7,22,23,27].

Although existing graph network embedding methods provide excellent performance, they are primarily developed for static graphs where nodes and edges remain unchanged over time. In most cases, however, networks behave as dynamic graphs in the actual world. For example, as new friendship contacts grow, new communication events such as emails and text messages are streamed on social networks. In e-commerce networks, new goods and ratings arise daily. In financial networks, transactions are streamed in computational finance, and supply chain relationships are always changing. In these dynamic graphs, nodes and edges are constantly evolving. The evolution trend of dynamic graphs can be recorded by a temporal sequence made up of a series of graph snapshots. Compared with static graphs, dynamic graphs have an additional dimension(i.e., the time dimension) that adds temporal dynamics to them. As a result, dynamic graph embedding is presented as a solution to the major issue of dynamic graphs, which is capturing temporal dynamics adequately.

Recently, several efforts, like DynmaicTraid [28], DynGEM [6], and TIMER [26], use some smoothness regularization to capture temporal dynamics. The premise behind these strategies is that dynamic graphs change slowly and thus they are unable to address dynamic graphs with abrupt changes. More recently, with the remarkable success of graph convolutional networks(GCN), some researchers focus on extending the GNNs to dynamic graphs by combining GCN with RNN components(e.g., LSTM or GRU), such as WD-GCN [14], EvolveGCN [17], and GANE [20]. However, these current GCN methods are designed for simple graphs that only represent pair-wise relationships among nodes. Thus, these GCN methods can not handle dynamic graphs composed of a series of simple graph snapshots independently, forcing them to resort to RNN components. Therefore, these approaches based on mixed architectures may break the internal link between topological information and temporal dynamics. Additionally, these methods only focus on capturing information from nodes and ignore edge information of graphs which is also an essential component of graph information.

To tackle the above issues, we propose a novel dynamic graph embedding framework, called DynHyper. First, we design a temporal hypergraph to model a dynamic graph that contains the characteristic of both local structure and temporal dynamics. Compared with simple graphs, hypergraphs can describe multiple relationships among nodes, and thereby we construct temporal hypergraphs to represent the correlation of nodes including local structure and temporal dynamics. To be specific, the main difference between simple graphs and hypergraphs lies in that hypergraphs contain hyperedges, which can connect an arbitrary number of nodes. Hence, we employ a hyperedge to connect nodes in

the same time step, which is enclosed with the local structure information. Interactions between distinct hyperedges can indicate temporal interactions between nodes, allowing us to capture temporal dynamics in our model. In addition, edge information is an indispensable part of the graph. Therefore, we introduce a hyperedge projection for temporal hypergraphs to capture edge-level correlations of hypergraphs. The hyperedge projection aims to convert hyperedges to nodes, which preserves edge-level relationships of temporal hypergraphs and can integrate message-passing schemes for nodes. Finally, we propose the temporal edge-aware hypergraph convolution to operate message aggregation and transmission to update node embeddings on the temporal hypergraph. DynHyper's effectiveness is demonstrated by experimental results on seven real-world datasets in link prediction and node classification tasks.

In a nutshell, our key contributions can be summarized as follows:

- We introduce a temporal hypergraph construction to capture the local structure information and temporal dynamics simultaneously for dynamic graphs and a hyperedge projection to obtain edge-level relationships for temporal hypergraphs.
- We propose a temporal edge-aware hypergraph convolutional network that can execute message passing in dynamic graphs autonomously and effectively without the need for RNN components.
- We conduct our experiments on seven real-world datasets in link prediction and node classification tasks to evaluate the effectiveness of DynHyper. Our findings show superior predictive performance, compared to the state-of-the-art methods in dynamic graph embedding.

2 Related Work

Dynamic graph embedding plays a crucial role in network analysis, which aids in the advancement of many real-world applications such as social recommendation and protein structure prediction. Roughly, we classify them into three streams: random walk methods, autoencoder-based methods, and GNNs-based methods.

Random walk methods aim to apply random walks to generate node sequences and incrementally update the node embedding affected by temporal evolution [9,13,25]. For instance, dynnode2vec [13] employs the evolve random walks that only generate the walks for the changed nodes and proposes a dynamic skip-gram model, where the previous embedding is initialized as the weight for the next graph snapshot. For autoencoder-based methods, one seeks to minimize the reconstruct loss of a given graph snapshot. For example, DynGEM [6] proposes an incremental fully-connected network that can share the parameters between two consecutive networks to capture temporal evolution. The other aims to minimize the reconstruct loss between the previous graph snapshots and the future graph snapshot. For instance, dyngraph2vecAE [4] introduces the autoencoder network with the reconstruct loss between the adjacency mapped by the previous graph embeddings and the adjacency in the next time step.

Recently, the popular way to cope with dynamic graph embedding is to combine the GNNs model with temporal components(e.g., LSTM). GCRN-M1 [19] first employs graph convolution to obtain node embedding and then feed them into an RNN to capture temporal dynamics. The distinction between WD-GCN [14] and GCRM-M1 [19] lies in that WD-GCN utilizes the separate LSTM components per node. EvolveGCN [17] aims to use an RNN to evolve the parameter of the GNNs model, significantly reducing the mode size(i,e, the model parameters). DySAT [18] employs a self-attention mechanism with the GNNs model to joint learn representation along the dimensions of both local structure and temporal dynamics. GANE [20] utilizes tensor factorization to obtain temporal pattern similarity of nodes and incorporates it into the graph attention network for capturing temporal dynamics.

3 Preliminaries

Notations. A dynamic graph network is defined as a series of static graph network snapshots collected at each time step t, i.e., $\mathbb{G} = \{G^1, G^2, \ldots, G^T\}$, where T denotes the total number of time steps. Each graph snapshot $G^t = (V^t, E^t)$ is a weight undirected graph network made of a node-set V^t, an edge set E^t, and a weighted adjacency matrix A^t at each time step t.

Problem Formulation. In this subsection, we formally define the problem of dynamic graph embedding. Given a dynamic graph G, dynamic graph embedding aims to learn mappings $f^t : \{G^1, G^2, \ldots, G^t\} \longrightarrow R^{|V^t| \times d}$ so that they obtain the latent representation $Z^t = f^t(G^1, G^2, \ldots, G^t)$, where $Z^t \in R^{|V^t| \times d}$ and d denotes the embedding dimensionality. Here, each row vector $Z_v^t \in R^d$ is the low dimensional embedding of node v, which preserves local topological proximities and temporal evolutionary pattern information up to time step t.

4 Methodology

In this section, we present our proposed framework for dynamic graph embedding, as illustrated in Fig. 1. The proposed framework includes three major parts. First, we introduce the temporal hypergraph to capture both local structure information and temporal dynamics for dynamic graphs. Then, we use a hyperedge projection to obtain edge-level relationships. After that, we utilize the temporal edge-aware hypergraph convolution to aggregate information and pass on them among nodes to update nodes embedding, illustrated in the following sections.

4.1 Temporal Hypergraph Construction

In this subsection, we discuss temporal hypergraph construction. Note that, a dynamic graph contains a series of graph snapshots. The major challenge for

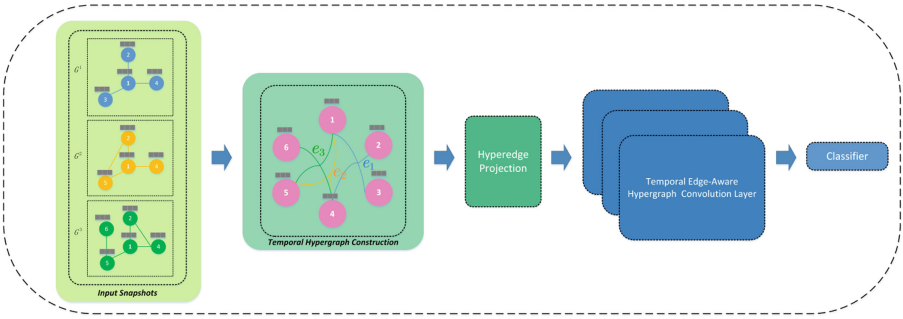


Fig. 1. An overview of our proposed framework. Given snapshot graphs $\{G^1, G^2, G^3\}$ as input, we first generate the temporal hypergraph based on time steps. To be specific, the temporal hypergraph contains all original nodes from input snapshots and hyperedges. A hyperedge is composed of the nodes from the same time step, i.e., e_1, e_2, e_3. Then, those hyperedges in the temporal hypergraph are operated by a hyperedge projection. After that, we utilize the temporal edge-aware hypergraph convolution to aggregate information and pass on them among nodes to update nodes embedding.

dynamic graph embedding is to capture temporal evolution among these graph snapshots. The prior works mainly focus on restoring to RNN or Transformer to capture temporal dynamics indirectly, which splits the internal connection between topological information and temporal dynamics. To address this issue, we aim to directly capture both temporal dynamics and topological information through the properties of the hypergraph.

For a given dynamic graph $\mathbb{G} = \{\mathbb{V}, \mathbb{E}\}$, where $\mathbb{V} = \{V^1, V^2, \ldots, V^t\}$ denotes a series of node sets and $\mathbb{E} = \{E^1, E^2, \ldots, E^t\}$ denotes a series of edge sets. We assume that historical observations start from time step 1 to time step τ. First, note that $V^1 \subseteq V^2 \subseteq \ldots \subseteq V^\tau$, V^τ contains all nodes in graph snapshots up to time step τ, so we define V^τ as a hypernode set of the temporal hypergraph. Second, we aim to construct hyperedges of the temporal hypergraph. More specifically, a hyperedge $e \in E^\tau$ is formed by linking a centroid node and its first-order neighbors at the same time step, where $E^\tau = \{e^m_{v_i} | m \in \{1, .., \tau\}, v_i \in V^\tau\}$ is the hyperedge set of the temporal hypergraph and m denotes a certain time step. For example, if a hyperedge connects v_1,v_2 and v_3, it can be denoted as $e^m_{v_1} = \{v_1, v_2, v_3 | v_2, v_3 \in N(v_1), v_1, v_2, v_3 \in G^m\}$, where v_1 is assigned as a centroid hypernode, and $N(v_1)$ is the first-order neighbors' set of hypernode v_1. Based on the discussion above, we define the temporal hypergraph as $H^\tau = (V^\tau, E^\tau, W)$, where W denotes weight matrix for hyperedge, $|V^\tau|$ is the number of hypernodes, and $|E^\tau|$ is the number of hyperedges. For simplicity, we use $|V|$ and $|E|$ to represent $|V^\tau|$ and $|E^\tau|$ respectively. Formally, the temporal hypergraph can be represented by an incidence matrix $H \in R^{|V| \times |E|}$:

$$H(v_i, e^m_{v_j}) = \begin{cases} 1, & \text{if } v_i \in e^m_{v_j} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

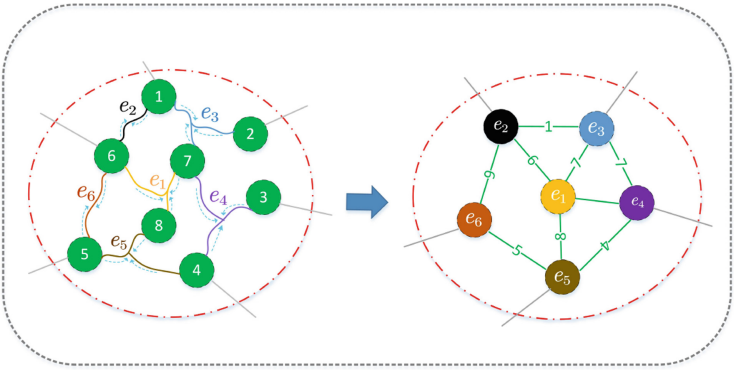


Fig. 2. An example of a hyperedge projection.

4.2 Hyperedge Projection

In this subsection, we further explore the edge-level correlations in hypergraphs. The temporal hypergraph is designed to obtain temporal dynamics of dynamic graphs, but it cannot well reflect the edge-level correlations in dynamic graphs. Thus, we introduce a hyperedge projection to extract edge-level correlations for dynamic graphs. The key idea of hyperedge projection is to capture edge-edge correlations of the temporal hypergraph. Figure 2 shows an example of a hyperedge projection. Specifically, the hypernodes connected to the same hyperedge are uniformly mapped into an edge that is defined as a node in the new graph. The hypernode projection can be formally written as follows:

$$P = D_e^{-1} H^T X \tag{2}$$

where $P \in R^{|V| \times M}$ is the hyperedge projection embedding of the original hypernode representation X, H^T is the transpose matrix of the incidence matrix H, and $D_e \in R^{|E| \times |E|}$ denotes the hyperedge degree matrix. Then, these new nodes are connected if they contain the same hypernode. For example, in Fig. 2, e_5 connects e_6 by the green line with the number 5, denoting that they contain the same hypernode v_5. In other words, these nodes connected are neighbors if they share the same hypernodes. Compared to hyperedges, edges only connect two nodes in this new graph. In a way, we convert the temporal hypergraph to a simple graph by the hyperedge projection and can easily integrate message-passing schemes for nodes, which preserves the original edge-level correlations of the temporal hypergraph.

4.3 Temporal Edge-Aware Hypergraph Convolution

In this section, we introduce the details of the message passing process via temporal edge-aware hypergraph convolution. In our work, if nodes have not initial feature, each node v is initialized by a one-hot vector $x_v \in R^M$, where M is the number of nodes in G^τ. Then, an update operation for each node v is conducted in the temporal hypergraph, which contains intra-hyperedge aggregation and inter-hyperedge aggregation. The intra-hyperedge aggregation can be formulated as:

$$z_e = \sum_{v \in e} \frac{x_v}{\delta(e)} \tag{3}$$

where z_e is the hyperedge representation through the intra-hyperedge aggregation, x_v is the initial representation of the node v, and $\delta(e)$ denotes the degree of the hyperedge $|e|$. Afterward, the inter-hyperedge aggregation can be expressed as:

$$z_v = \sum_{e \in S} \frac{z_e}{d(v)} \tag{4}$$

where z_v is the embedding of the node v through the inter-hyperedge aggregation, S is the set of hyperedges containing the node v, and $d(v)$ denotes the number of hyperedges containing the node v. These two steps can merge and be rewritten as:

$$Z = D_e^{-1} H^T D_v^{-1} H X \tag{5}$$

where $Z \in R^{|E| \times M}$ is the embedding of hypernodes, $H \in R^{|V| \times |E|}$ denotes the incident matrix, the original hypernode representation $X \in R^{|V| \times M}$,and $D_v \in R^{|V| \times |E|}$ denotes hypernode degree matrix. We observe that this update operation is equal to the simplified hypergraph convolution [2]. Then, we further extend this operation to capture the edge-level correlations in the temporal hypergraph. According to the hypernode projection mentioned in Sect. 4.2, we utilize the hypernode projection to replace the original hypernode representation X with P in the convolution rule as follows:

$$Z_{\text{edge}} = D_e^{-1} H^T D_v^{-1} H P \tag{6}$$

where $Z_{edge} \in R^{|E| \times M}$ is the edge-level embedding of hypernodes. After capturing the edge-level correlations of the temporal hypergraph, we remap the edge-level embedding into the node-level embedding which is assigned to each node in the dynamic graph:

$$Z_{node} = H Z_{edge} \tag{7}$$

where $Z_{node} \in R^{|V| \times M}$ is the node-level embedding of hypernodes. Then, we utilize the renormalization trick introduced by [11] and employ a learnbale matrix. The complete temporal hypergraph propagation rule can be written as follows:

$$\begin{aligned} \tilde{Z} &= \sigma\left(D_v^{-1/2} Z_{node} D_v^{-1/2} \Theta\right) \\ &= \sigma\left(D_v^{-1/2} H D_e^{-1} H^T D_v^{-1} H P D_v^{-1/2} \Theta\right) \end{aligned} \tag{8}$$

where $\tilde{Z} \in R^{|E| \times d}$ is the final embedding of hypernodes, $\Theta \in R^{|E| \times d}$ denotes the model parameters matrix, d denotes the embedding dimensionality, and $\sigma(\cdot)$ denotes the activation function (e.g. ELU).

Table 1. The statistics of datasets

Dataset	#nodes	#edges	#features	#labels	#time steps
UCI	1,809	16,822	–	–	13
Enron	143	2,347	–	–	12
Yelp	6,569	95,361	–		12
ML-10M	20,537	43,760	–	–	13
Alibaba	5,640	53,049	19	–	11
Epinions	9,368	231,537	44	–	9
Primary School	242	20,009	–	11	40

4.4 Loss Function

In this subsection, we introduce the objective function that enables node representations to capture dynamic topological evolution during training our model. Inspired by DySAT [18], our model encourages nodes sampled in the fixed-length random walk to obtain similar representations. Formally, we use a binary cross-entropy loss to optimize the model parameters as follows:

$$L = \sum_{v \in V} \Big(\sum_{u \in N_{walk}(v)} -\log\left(\sigma\left(< \tilde{z}_w, \tilde{z}_v >\right)\right) - \beta \cdot \sum_{u' \in P_n(v)} \log\left(1 - \sigma\left(< \tilde{z}_{u'}, \tilde{z}_v >\right)\right)\Big) \tag{9}$$

where $\tilde{z}_v$ is the final embedding of a node v, $\sigma(\cdot)$ denotes the sigmoid function, $< \cdot >$ denotes the inner product. $N_{walk}(v)$ is the positive nodes' set of a node v sampled by the random walk, and $P_n(v)$ is the negative nodes' set of a node v sampled by a negative sampling function based on the degree of nodes. β is the negative sample value to balance positive and negative samples.

5 Experiments and Analysis

5.1 Experimental Setup

Datasets. To evaluate the performance of our model, we use seven public real-world datasets in our experiments. The datasets are summarized in Table 1. UCI [16]is an online social network. Links of this network denote the massage sent between peer users, i.e., nodes. Enron [12] contains a set of email messages

concerning the Enron corporation, which is represented as an email communication network. Nodes of network denote the addresses and links denote there's an interaction between these email addresses. Yelp[1]: is a rating network of users and businesses where links connect users and businesses if users score the businesses. ML-10M [8] consists of users and tags that users applied to certain movies. The links of this network denote there's an interaction between users and movies. Alibaba[2] is an e-commerce network, consisting of users and items. The edge between user and item denotes the click interaction. Epinions[3] denotes a trusted network between users. The edge of the network indicates the trust correlation between users. Primary School [21] represents the contact network. The link of this network is constructed from the interactions between teachers and students.

Baselines. We compare our proposed model with the following state-of-the-art dynamic graph embedding methods: (1) DynAE [4] utilizes an autoencoder framework based on dense layers; (2) DynAERNN [4] is based on DynAE, which uses recurrent neural networks to capture temporal dynamics of dynamic graphs; (3) DynGEM [6] adopts an incremental autoencoder framework for a dynamic graph based on the last graph snapshot; (4) DySAT [18]: DySAT aims to simultaneously capture the local structure information and temporal dynamics based on the self-attention mechanism; (5) EvolveGCN [17] uses the GCN to learn local structure information for each graph snapshot and employs the GRU or LSTM to update parameters of GCN to capture temporal evolution based on different graph snapshots; (6) GAEN [20] incorporates node temporal pattern similarities based on the tensor factorization technique and neighborhood attention to learn the node embedding for dynamic graphs.

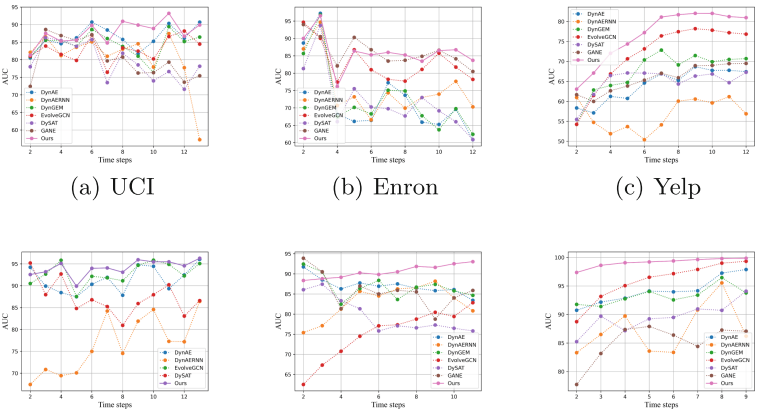
Settings. In our experiments, we evaluate the performance of our model in both link prediction and node classification tasks. For link prediction, we train a logistic regression classifier to predict the existence of links at the time step $t+1$ based on the embeddings learned from previous networks up to time step t. We randomly sample 60% of nodes for training. We utilize 20% of nodes to tune hyperparameters of our model and the remaining 20% of nodes for testing. We utilize the Mean Accuracy(ACC) and the Mean Area Under the ROC Curve (AUC) as our evaluation metrics of link prediction. For node classification, we randomly sample 20% of nodes as a validation set. Then, we use 30%, 50%, and 70% of nodes as train sets respectively, the corresponding remaining nodes are used as test sets. We also train a logistic regression classifier to map nodes into different categories based on the embeddings learned from previous networks up to time step t. We employ the Mean Accuracy(ACC) as our evaluation metrics of node classification. We use mini-batch gradient descent with Adam. For hyperparameters, we set batch size as 512, the embedding dimensionality d as 128, the

[1] https://www.yelp.com/dataset/.
[2] https://tianchi.aliyun.com/competition/entrance/231719/information/.
[3] https://cse.msu.edu/~tangjili/trust.html.

learning rate as 10^{-3}, the weight decay as 5×10^{-4}, the max epoch as 20, and negative sample ratio β as 0.01. We conduct our experiments on a machine with the Intel Core i9-9960X (3.10GHz) CPU, 128 Gb of RAM, and four NVIDIA 2080Ti GPU cards, which are implemented in Python 3.6 with Tensorflow.

Table 2. The predictive performance of link prediction task in terms of AUC and ACC on UCI, Enron, Yelp, ML-10M, Alibaba, and Epinions. The results are the mean and standard deviation of 5 different runs. OOM denotes running out of memory on our machine.

Method	Metric	UCI	Enron	Yelp	ML-10M	Alibaba	Epinion
DynAE	ACC	69.05 ± 0.08	68.22 ± 1.09	60.00 ± 0.07	64.75 ± 0.31	75.84 ± 0.03	74.53 ± 0.21
	AUC	85.94 ± 0.26	72.64 ± 0.16	64.17 ± 0.16	91.14 ± 0.21	87.51 ± 0.20	94.17 ± 0.03
DynAERNN	ACC	68.92 ± 1.34	67.36 ± 0.40	54.77 ± 0.67	74.55 ± 0.32	76.21 ± 0.39	74.71 ± 0.94
	AUC	82.73 ± 0.58	75.56 ± 0.14	56.74 ± 0.87	76.57 ± 0.86	82.74 ± 0.28	82.03 ± 1.52
DynGEM	ACC	71.01 ± 1.19	66.99 ± 0.34	60.98 ± 0.06	OOM	75.42 ± 0.60	83.82 ± 0.61
	AUC	84.29 ± 1.88	72.90 ± 1.36	67.33 ± 0.04	OOM	87.31 ± 0.43	91.88 ± 1.04
EvolveGCN	ACC	72.26 ± 0.45	66.14 ± 1.09	61.03 ± 0.29	79.35 ± 0.32	72.63 ± 0.08	81.01 ± 0.57
	AUC	79.58 ± 0.28	72.12 ± 1.08	64.94 ± 0.23	87.28 ± 0.79	79.56 ± 0.11	89.07 ± 0.38
DynSAT	ACC	68.47 ± 0.05	74.17 ± 1.03	65.76 ± 0.23	82.40 ± 0.66	67.52 ± 0.15	89.13 ± 0.14
	AUC	82.77 ± 0.08	82.97 ± 1.03	71.84 ± 0.65	92.86 ± 0.15	75.34 ± 0.15	96.14 ± 0.52
GANE	ACC	72.86 ± 0.94	**78.99±0.66**	62.38 ± 0.11	OOM	77.00 ± 0.02	74.85 ± 0.55
	AUC	80.59 ± 0.74	86.08 ± 0.50	65.76 ± 0.18	OOM	85.45 ± 0.23	82.20 ± 2.09
DynHyper(ours)	ACC	**75.04±0.16**	76.21 ± 0.38	**69.90±0.71**	**82.86±0.61**	**83.08±0.04**	**91.58±0.29**
	AUC	**87.81±0.13**	**87.26±0.39**	**76.61±0.04**	**94.14±0.15**	**90.58±0.03**	**99.10±0.01**



(a) UCI (b) Enron (c) Yelp

(d) ML-10M (e) Alibaba (f) Epinions

Fig. 3. The results of six datasets in link prediction task in terms of AUC at various time steps.

Table 3. The predictive performance of the node prediction task in terms of ACC on Primary School at different train ratios. The results are the mean and standard deviation of 5 different runs.

Dataset	Primary School		
Train ratios	30%	50%	70%
DynAE	47.19 ± 0.02	47.25 ± 0.05	46.05 ± 0.04
DynAERNN	46.85 ± 0.14	46.99 ± 0.18	46.99 ± 0.04
DynGEM	50.01 ± 0.08	51.05 ± 0.11	50.84 ± 0.26
EvolveGCN	42.75 ± 0.30	45.16 ± 0.23	46.20 ± 0.24
DySAT	60.72 ± 0.38	63.72 ± 0.24	64.85 ± 0.39
GANE	55.17 ± 0.19	57.82 ± 0.09	59.26 ± 0.10
DynHyper(ours)	**63.39±0.09**	**65.86±0.17**	**67.01±0.13**

5.2 Experimental Results

Link Prediction. In this subsection, we discuss the performance of our model in the link prediction task compared with state-of-the-art methods. Experimental results are illustrated in Table 2. Table 2 shows that DynHyper consistently outperforms baselines in all datasets except that GANE outperforms DynHyper on the Enron dataset under ACC. These results indicate the effectiveness of DynHyper in link prediction. For example, as compared to the best approach of baselines(i.e., DySAT) on the Yelp dataset, we get roughly a 5% improvement in both AUC and ACC. Note that GANE gains better performance than DynHyper on the Enron dataset. GANE obtains node temporal patterns via tensor factorization to improve performance, which may be more successful on tiny datasets like Enron having only 143 nodes. However, DynHyper tries to capture the edge-level correlations on datasets, which may perform better in large datasets rather than small ones. As the result shows, DynHyper obtains about 94% AUC and 99% AUC on ML-10M and Epinions datasets respectively, which are much larger datasets than the Enron dataset. Based on the abovementioned, this might be the reason why our approach on Enron is inferior to GANE. Besides, DynGEM employs the smoothness regularization to capture temporal dynamics that can not address the network with abrupt change. Users' communications on UCI typically span longer periods, showing that the network is smooth. However, rating behaviors on Yelp, tend to be erratic and connected with events like restaurant openings and discounted promotions, indicating a network with abrupt change. Thus, we observe that DynGEM obtain a relatively better performance on UCI than the performance on Yelp. The predictive results of DynHyper are consistently superior to DynGEM on all the datasets, especially Yelp, demonstrating that DynHyper performs well in both smooth and abrupt networks.

Furthermore, we seek to analyze the detailed performance of these methods at each time step. The results are reported in Fig. 3. First, we note that DynHyper is inferior to some baselines at the initial time step on some datasets, such as

Enron and Alibaba. The potential reason is that these datasets do not form a lot of edge-level relationships at the initial time step. Additionally, we find that as the time step is increased, DynHyper's performance improves. Moreover, DyperHyper is consistently superior to all baselines at each time step on some datasets, such as Yelp and Epinions. This finding might be caused by these datasets containing more edge-level correlations. It is worth noticing that Yelp and Epinions have more links than other datasets.

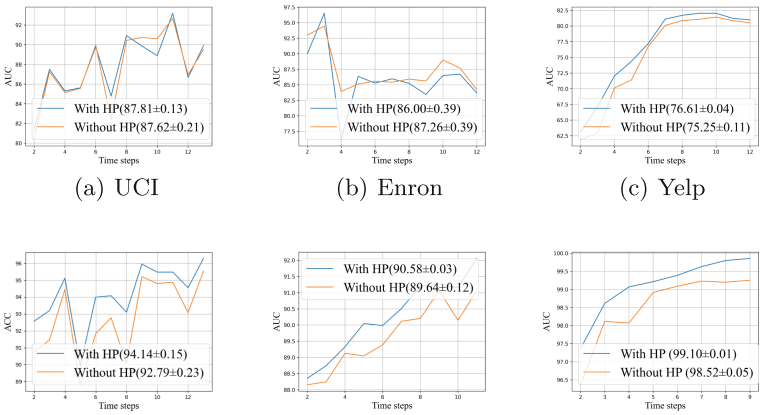


Fig. 4. Experimental results for ablations

Node Classification. In this subsection, we compare DynHyper's performance to that of state-of-the-art approaches in the node classification task. Due to the lack of dynamic graphs datasets with node labels, we use the Primary School dataset with different train ratios to fully use this dataset for evaluation. Table 3 shows the results of the experiments. DynHyper achieves a consistent 2%∼3% ACC improvement on Primary School at different train ratios, demonstrating DynHyper's effectiveness in node classification. In addition, approaches with the RNN component, such as DynAERNN and EvolveGCN, perform poorly in node classification. DynAERNN is even superior to DynAE, suggesting that Combination with the RNN component is ineffective at capturing temporal dynamics in the node classification task.

Ablation Study. In this subsection, we conduct ablation studies to evaluate the contribution of the hyperedge projection(HP) of our model. HP aims to capture edge-level relationships of datasets to improve performance. To better demonstrate this, we compare the performance between our model with HP and our model without HP at various time steps. The compared results are shown in

Fig. 4. According to Fig. 4, DynHyper with HP outperforms DynHyper without HP on most datasets. As discussed above, the Enron dataset is a small dataset having 143 nodes while HP is much more effective with big datasets. As a result, we note that DynHyper without HP is superior to DynHyper without HP on the Enron dataset.

6 Conclusion

In this paper, we propose a dynamic embedding framework to address dynamic graphs, named DynHyper. We introduce temporal hypergraph construction to capture effectively temporal dynamics for dynamic graphs. Additionally, we propose a hyperedge projection to obtain edge-level relationships of temporal hypergraphs. Furthermore, We propose a temporal edge-aware hypergraph convolutional network to independently and effectively conduct the message passing in dynamic graphs without any RNN components. Experimental results confirm that DynHyper has great performance in both link prediction and node classification tasks, especially on the more complex datasets. Our future work aims to extend our work to address more complex dynamic graphs, such as those with changeable attributed nodes.

Acknowledgments. This work was partly supported by the Guangdong Provincial Key Laboratory of Intellectual Property and Big Data (2018B030322016), the National Natural Science Foundation of China (U1701266), Special Projects for Key Fields in Higher Education of Guangdong, China(2021ZDZX1042), the Natural Science Foundation of Guangdong Province, China(2022A1515011146), Key Field R&D Plan Project of Guanzhou(202206070003).

References

1. Cai, H., Zheng, V.W., Chang, K.C.C.J.I.T.O.K., Engineering, D.: A comprehensive survey of graph embedding: problems, techniques, and applications. IEEE Trans. Knowl. Data Eng. **30**(9), 1616–1637 (2018)
2. Feng, Y., You, H., Zhang, Z., Ji, R., Gao, Y.: Hypergraph neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 3558–3565 (2019)
3. Fu, S., Liu, W., Zhang, K., Zhou, Y.: Example-feature graph convolutional networks for semi-supervised classification. Neurocomputing **461**, 63–76 (2021)
4. Goyal, P., Chhetri, S.R., Canedo, A.J.K.B.S.: dyngraph2vec: capturing network dynamics using dynamic graph representation learning. Knowl.-Based Syst. **187**, 104816 (2020)
5. Goyal, P., Ferrara, E.J.K.B.S.: Graph embedding techniques, applications, and performance: a survey. Knowl.-Based Syst. **151**, 78–94 (2018)
6. Goyal, P., Kamra, N., He, X., Liu, Y.: Dyngem: Deep embedding method for dynamic graphs. arXiv preprint arXiv:1805.11273 (2018)
7. Hamilton, W.L., Ying, R., Leskovec, J.: Representation learning on graphs: methods and applications. IEEE Data Eng. Bull. **40**(3), 52–74 (2017)

8. Harper, F.M., Konstan, J.A.: The movielens datasets: History and context. ACM Trans. Interact. Intell. syst. (TIIS) **5**(4), 1–19 (2015)
9. Heidari, F., Papagelis, M.: EvoNRL: evolving network representation learning based on random walks. In: Aiello, L.M., Cherifi, C., Cherifi, H., Lambiotte, R., Lió, P., Rocha, L.M. (eds.) COMPLEX NETWORKS 2018. SCI, vol. 812, pp. 457–469. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05411-3_37
10. Jiang, Y., Ma, H., Liu, Y., Li, Z., Chang, L.: Enhancing social recommendation via two-level graph attentional networks. Neurocomputing **449**, 71–84 (2021)
11. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings. OpenReview.net (2017)
12. Klimt, B., Yang, Y.: The Enron corpus: a new dataset for email classification research. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) ECML 2004. LNCS (LNAI), vol. 3201, pp. 217–226. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30115-8_22
13. Mahdavi, S., Khoshraftar, S., An, A.: dynnode2vec: scalable dynamic network embedding. In: 2018 IEEE International Conference on Big Data (Big Data), pp. 3762–3765. IEEE (2018)
14. Manessi, F., Rozza, A., Manzo, M.J.P.R.: Dynamic graph convolutional networks. Pattern Recogn. **97**, 107000 (2020)
15. Mizerka, J., Stróżyńska-Szajek, A., Mizerka, P.J.F.R.L.: The role of bitcoin on developed and emerging markets-on the basis of a bitcoin users graph analysis. Finan. Res. Lett. 35, 101489 (2020)
16. Panzarasa, P., Opsahl, T., Carley, K.M.: Patterns and dynamics of users' behavior and interaction: network analysis of an online community. J. Am. Soc. Inform. Sci. Technol. **60**(5), 911–932 (2009)
17. Pareja, A., et al.: Evolvegcn: evolving graph convolutional networks for dynamic graphs. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 5363–5370 (2020)
18. Sankar, A., Wu, Y., Gou, L., Zhang, W., Yang, H.: 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)
19. Seo, Y., Defferrard, M., Vandergheynst, P., Bresson, X.: Structured sequence modeling with graph convolutional recurrent networks. In: Cheng, L., Leung, A.C.S., Ozawa, S. (eds.) ICONIP 2018. LNCS, vol. 11301, pp. 362–373. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-04167-0_33
20. Shi, M., Huang, Y., Zhu, X., Tang, Y., Zhuang, Y., Liu, J.: GAEN: graph attention evolving networks. In: Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI) (2021)
21. Stehlé, J., et al.: High-resolution measurements of face-to-face contact patterns in a primary school. PLoS ONE **6**(8), e23176 (2011)
22. Tian, X., Ding, C.H., Chen, S., Luo, B., Wang, X.: Regularization graph convolutional networks with data augmentation. Neurocomputing **436**, 92–102 (2021)
23. Wang, S., Fu, K., Sun, X., Zhang, Z., Li, S., Jin, L.: Hierarchical-aware relation rotational knowledge graph embedding for link prediction. Neurocomputing **458**, 259–270 (2021)
24. Wang, Y., You, Z.H., Yang, S., Li, X., Jiang, T.H., Zhou, X.J.C.: A high efficient biological language model for predicting protein-protein interactions. Cells **8**(2), 122 (2019)

25. Yu, W., Cheng, W., Aggarwal, C.C., Zhang, K., Chen, H., Wang, W.: Netwalk: a flexible deep embedding approach for anomaly detection in dynamic networks. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2672–2681 (2018)
26. Zhang, Z., Cui, P., Pei, J., Wang, X., Zhu, W.: Timers: error-bounded SVD restart on dynamic networks. In: Thirty-Second AAAI conference on artificial intelligence (2018)
27. Zheng, W., Qian, F., Zhao, S., Zhang, Y.: M-GWNN: multi-granularity graph wavelet neural networks for semi-supervised node classification. Neurocomputing **453**, 524–537 (2021)
28. Zhou, L., Yang, Y., Ren, X., Wu, F., Zhuang, Y.: Dynamic network embedding by modeling triadic closure process. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32 (2018)