Alexandria Engineering Journal (2023) 63, 359–369

HOSTED BY

Alexandria University

Alexandria Engineering Journal

www.elsevier.com/locate/aej
www.sciencedirect.com

A dynamic graph representation learning based on temporal graph transformer

Ying Zhong *, Chenze Huang

Research and Development Institute of Northwestern Polytechnical University, Shenzhen, Guangdong, China

Received 19 November 2021; revised 21 April 2022; accepted 7 August 2022
Available online 24 August 2022

KEYWORDS

Graph neural network;
Continuous-time dynamic graph;
Deep learning;
Complex networks

Abstract The graph neural network has received significant attention in recent years because of its unique role in mining graph-structure data and its ubiquitous application in various fields, such as social networking and recommendation systems. Although most work focuses on learning low-dimensional node representation in static graphs, the dynamic nature of real-world networks makes temporal graphs more practical and significant. In this paper, we propose a dynamic graph representation learning method based on a temporal graph transformer (TGT), which can efficiently preserve high-order information and temporally evolve structural properties by incorporating an update module, an aggregation module, and a propagation module in a single model. The experimental results on three real-world networks demonstrate that the TGT outperforms state-of-the-art baselines for dynamic link prediction and edge classification tasks in terms of both accuracy and efficiency.

1. Introduction

The graph neural network (GNN) has emerged as a keystone machine learning technique for graph-based data, due to its ability to learn from non-Euclidean space. The GNN focuses on preserving graph topological information, i.e., the proximity of nodes, node features, and edge features, and has been applied in multiple fields [1,2], such as e-commerce [3], social networking [4], and biology [5]. In e-commerce, users and items are both seen as nodes, and the edges between them denote shopping events or click events. The downstream mission (i.e., supervised learning tasks using a pre-trained model or component) contains node classification [6,7], link prediction [8], graph generation [9], and node clustering [10]. In addition, the GNN is also widely used in networking and communication [11–13].

Most aforementioned research [14–18] on GNNs has been done mostly on static graphs, which consist of fixed nodes and edges that do not change over time. However, networks usually have complicated temporal properties in reality, i.e., dynamic graphs [19,20]. A dynamic graph not only contains structural properties but also holds the network's evolving information, indicated by the timestamp in the edge. For social networks, nodes represent users, and edges show the interactions between two users. Each interaction has a timestamp, representing the time at which the edge emerges. If a static

* Corresponding author.
E-mail address: yxy_0713@qq.com (Y. Zhong).
Peer review under responsibility of Faculty of Engineering, Alexandria University.

https://doi.org/10.1016/j.aej.2022.08.010

graph neural network is directly applied on dynamic graphs, the temporal factor of the network is completely ignored.

The temporal graph embedding technique is classified into two categories [21], i.e., a discrete time dynamic graph (DTDG) and a continuous time dynamic graph (CTDG). A DTDG is represented as a series of snapshots of a graph obtained in different timestamps. Although temporal information between snapshots is preserved, the time factor is completely ignored in a snapshot. A CTDG is represented as a series of time-ordered edges that can maintain time information in fine granularity. A CTDG keeps more temporal information than a DTDG. Because of the inherent characteristics of a DTDG, a DTDG-based model has a serious problem, i.e., it is hard to apply in large graphs due to the high memory consumption. Therefore, most existing DTDG-based models are evaluated on relatively small benchmarks. A CTDG-based graph neural network [8,22] is a solution that has achieved remarkable performance in large dynamic networks. However, to meet the time requirement, these state-of-the-art models only aggregate a 1-hop neighbors information, since the time complexity exponential rises when the layers become deeper. High-order information is important in a graph-based structure, which cannot be preserved in these models owing to the requirement of a trade-off between time consumption and performance. To effectively learn node embedding from the high-order information of large graphs, it is urgent to design a CTDG-based graph neural network to effectively learn a high-order neighbor's information.

In this paper, we propose a temporal graph transformer (TGT) to efficiently learn information from 1-hop and 2-hop neighbors. The model consists of three modules, namely, an update module, an aggregation module, and a propagation module, and each node maintains two vectors, i.e., memory and embedding. The update module updates the memory of source nodes and target nodes within new emerging interactions by the GRU model, implicitly learning the temporal factor due to the input's being sorted in chronological order. The aggregation module is represented as a multi-layer transformer encoder, learns the attention of neighbors and aggregates its neighbors memory in a weighted way, thereby realizing node embedding. The propagation module propagates the embeddings of source and target nodes to their neighbors through the GRU, thereby re-updating the neighbor's memory and introducing 2-order information into the neighbor's memory. The propagation operation replaces the 2-hop aggregation that is generally performed in state-of-the-art baselines to dramatically reduce the time consumption of training and inference phase. Finally, we evaluate the performance of the proposed TGT model on three real-world networks and experimental results show that the proposed model remarkably outperforms all baselines in accuracy and efficiency. To facilitate readers' understanding, we list the acronyms and their corresponding full names in Table 1.

Overall, our contributions can be summarized as follows:

- We propose a novel continuous-time dynamic graph neural network, called a temporal graph transformer (TGT), which can efficiently learn information from 1-hop and 2-hop neighbors by modeling the interactive change sequential network and can learn node representation more accurately.
- We introduce an update module, an aggregation module and a propagation module. The update module learns temporal information implicitly, the aggregation module aggregates 1-hop temporal neighbors, and the propagation module embeds 2-order information into the node memory state. Aggregation and propagation operations can effectively improve the accuracy of dynamic graph representation and reduce running time.
- We conducted experiments on three real-word benchmarks to demonstrate the effectiveness and efficiency of our proposed model. Experimental results show that our model outperforms all state-of-the-art baselines.

Table 1 Acronyms and their corresponding full names.

Acronyms	Full Names
CTDG	Continuous Time Dynamic Graph
DTDG	Discrete Time Dynamic Graph
TGT	Temporal Graph Transformer
GRU	Gated Recurrent Unit
LSTM	Long Short-term Memory
GNN	Graph Neural Network
RNN	Recurrent Neural Network
AUC	Area Under Curve
MRR	Mean Reciprocal Rank

The remainder of this paper is organized as follows. In Section 2, work related to the temporal graph embedding technique is introduced; Section 3 describes the implementation of the temporal graph transformer for dynamic networks; Section 4 provides the experimental results; finally, the research is summarized in Section 5.

2. Related work

The temporal graph embedding technique is classified into two classes [21], i.e., a discrete-time dynamic graph (DTDG) and a continuous-time dynamic graph (CTDG). In Table 2, we list the representative algorithms based on these methods and the deep learning model used.

2.1. Discrete-Time Dynamic Embedding Approaches

DTDG approaches treat dynamic graph as a series of snapshots of graphs with different timestamps. Most discrete dynamic embedding approaches [30,31,18,20,32] focus on the learning representations of entire dynamic graphs rather than node representations. Goyal et al. [33] proposed a DTDG-based graph neural network based on a deep autoencoder to achieve stability, flexibility, and efficiency. In addition, Goyal et al. [19] proposed dyngraph2vec, combining an autoencoder and a recurrent neural network to introduce time dependency in inductive learning. Pareja et al. [23] proposed EvolveGCN, utilizing a recurrent neural network to update parameters in a graph neural network so as to induce temporal information and maintain a traditional GNN structure. Sankar et al. [24] introduced structural self-attention and position-aware

Table 2 Representative algorithms based on DTDG and CTDG, and their corresponding deep learning models.

	Algorithm	Deep Learning Model
Discrete-time Dynamic Embedding Approaches	dyngraph2vec [19]	Fully-Connected Autoencoders and LSTMs
	Evolvegcn [23]	Graph Convolutional Network and Evolving Graph Convolution Unit
	Dysat [24]	Temporal Self-attention
	EvolveGCN [25]	Temporal Attention
	Lime [26]	Recursive Neural Network
Continuous-time Dynamic Embedding Approaches	HTNE [6]	Attribute-Topology and Temporal Network
	DyRep [27]	Temporal Attention
	TGAT [28]	Transformer-Style Self-Attention and GCNs over time
	TGNs [22]	Recursive Neural Network
	JODIE [29]	Coupled Recurrent Neural Network and Graph Convolutional Network

temporal self-attention in a graph neural network to obtain weighted information.

Some approaches [25,34] are starting to focus on the dynamic representation at the node level. They encode each graph snapshot using static embedding approaches [18,20,32,26] to embed each node, and they then combine time-series models (e.g., LSTM [35] and RNNs [36]) for each node to model the discrete dynamic. Although DTDG-based algorithms maintain the time information between two adjacent snapshots, they cannot preserve the temporal property of a snapshot.

2.2. Continuous-Time Dynamic Embedding Approaches

A CTDG is represented as a series of interactions. Zuo et al. [6] firstly applied the Hawkes process to model temporal interactions in dynamic networks. Trivedi et al. [27] presented DyRep, a framework based on a temporal point process, which considers both association events and communication events to capture immediate information and long-term information at the same time. Kumar et al. [29] proposed JODIE to update node memory through messages from newly formed interactions. In these methods, topological information cannot be obtained since they only update node embedding through messages from 1-hop neighbors.

To learn from high-order neighbors, Xu et al. [28] proposed TGAT, a temporal version of GAT, to aggregate temporal neighbors layer by layer. The time information is explicitly introduced through a time encoding mechanism. Rossi et al. [22] combined the advantages of TGAT and JODIE and introduced a memory module to preserve the last node state. The memory of the nodes in an edge is updated before the memory of neighbors is aggregated. Ma et al. [8] used the propagation mechanism to replace the aggregation operation in [22] and applied two updater to deal with source node memory and destination node memory, respectively.

Instead of directly maintaining node embeddings, TGN [22] computes node embeddings at different times by introducing a message and a memory mechanism. The architecture of the flow graph neural network [8] consists of two parts: 1) an update model and 2) a propagation model. The update model is used to update the embeddings of the involved nodes, and the propagation model propagates the information to the neighbors of the involved nodes. The process of our proposed mechanism is similar to that in [22]; however, there are some significant differences.

- Aggregated information is different. The mechanism in [22] aggregates the historical information of the node itself, while we focus more on the information of aggregated neighbors.
- The propagation operation replaces the 2-hop aggregation that is generally performed in state-of-the-art baselines to dramatically reduce time consumption in the training and inference phases.
- We discuss the influence of different factors on the aggregated-diffusion mechanism in more detail.

Although CTDG-based algorithms apply an aggregation operation or a propagation operation, they do not integrate advantages of these two mechanisms into a single model.

3. The Proposed Method

In this section, we describe the structure of a temporal graph transformer for dynamic networks. We first introduce the overall structure of the proposed model and then describe each component in detail. Finally, we briefly describe the training procedure of the proposed model.

Given a dynamic graph, which consists of a set of nodes and a set of edges, the set of nodes is represented as $V = \{v_1, v_2, \ldots, v_n\}$, and the number of nodes is n. The set of edges is represented as $E = \{e_1, e_2, \ldots, e_m\}$, and the number of edges is m. One edge corresponds to a timestamp, which denotes the edge formation time. The network evolves when two nodes in the network establish an edge. A directed edge is represented as (v_s, v_d, t, e), which describes the interaction between v_s and v_d at timestamp t with the edge attribute e, and an undirected edge is seen as two directed edges. Following the terminology in [8], the two nodes that interact are referred to as interacting nodes, and their neighbors are called influenced nodes.

Fig. 1 shows an example of a dynamic network, The edge emerges at timestamp t from v_8 to v_6 and at $t + 1$ from v_2 to v_3. The overview of the structure of the TGT is demonstrated in Fig. 2, which consists of three modules: an update module, an aggregation module, and a propagation module. We briefly introduce the operation of modules when a new edge is formed between v_s and v_d. For example, when a new interaction from v_8 to v_6 occurs at timestamp t, the update module updates the memory of v_8 to v_6 according to the update message derived from the interaction.

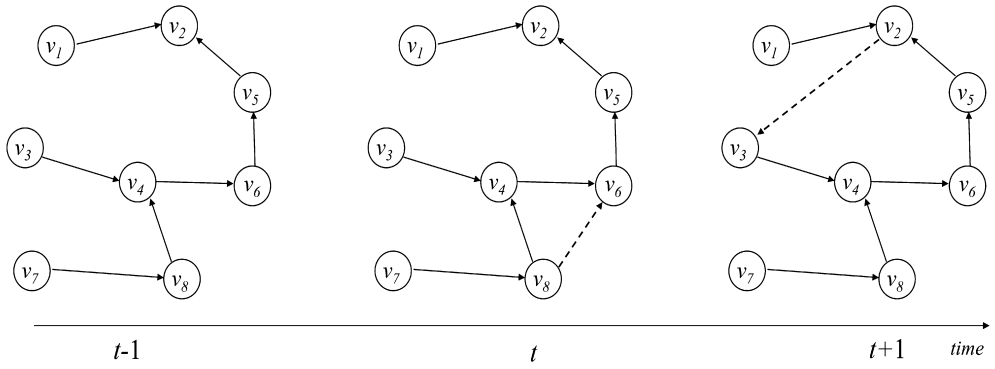

Fig. 1 The edge emerged at timestamp t from v_8 to v_6 and $t + 1$ from v_2 to v_3.

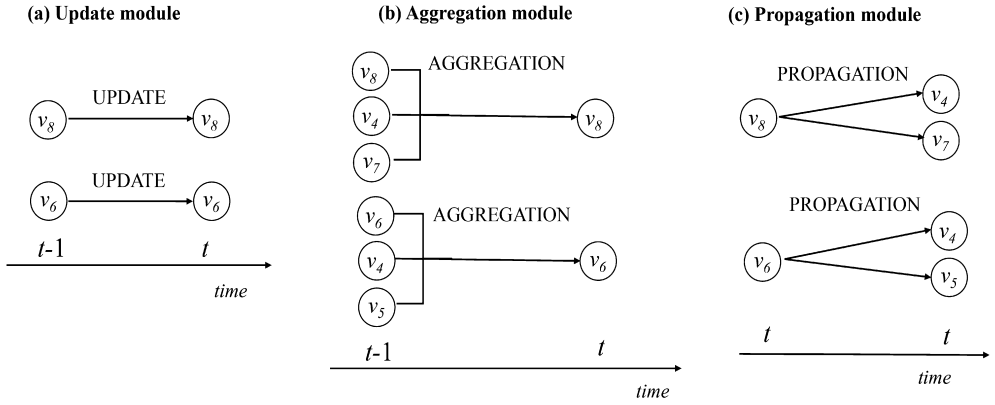

Fig. 2 Three modules (update module, aggregation module, and propagation module) and their corresponding execution process.

In the aggregation module, we propose a transformer-based embedding method to aggregate messages from influenced nodes. The aggregation module utilizes the aggregated messages sent from neighbors to calculate the embedding of interacting nodes. At timestamp t, influenced nodes set for v_8 is $\{v_7\}$ and those set for v_6 is $\{v_4,v_5\}$. After aggregation, the propagation module updates the memory of influenced nodes by propagating the new interaction information. The propagation mechanism introduces 2-order information into the memory of influenced nodes. For example, at timestamp t, the memory of the influenced node v_4 is updated according to the propagation message from the node v_4, which contains the information that a new edge between v_6 and v_8 appears at timestamp t. At the next timestamp $t + 1$, an edge between nodes v_3 and v_2 is formed, where v_4 is also an influenced node for v_3. Therefore, the aggregation module learns the information that an edge between v_8 and v_6 is formed at timestamp t by aggregating the aggregated messages from v_4. If the model does not have the propagation module, the memory of node v_4 can not contain edge generation information between v_8 and v_6 at timestamp t, so 2-order information cannot be delivered to the aggregation module at timestamp $t + 1$.

3.1. Update module

In this section, we describe the overview structure of the update module for the interacting nodes.We set a GRU (Gated Recurrent Unit) [37] as an updater to update the memory of the interacting nodes. Since the number of parameters in a GRU is lower, we do not use LSTM (Long Short-term Memory) [35]. The first operation in the update phase is to obtain update messages from newly formed edges. To reduce time consumption and to improve the efficiency of the model, we only used the updater to update the memory of the source and destination nodes. To make GRU distinguish between the source and destination nodes, we define different update messages for them. We adopted the following update message.

$$msg_s(t) = concat([m_s(t-), m_d(t-), e_{sd}(t)]) \tag{1}$$

$$msg_d(t) = concat([m_d(t-), m_s(t-), e_{sd}(t)]) \tag{2}$$

where s denotes the source node, d denotes the destination node, $m_s(t-)$ and $m_d(t-)$ represent the memory of the node v_s and the node v_d before the current timestamp, and e_{sd} is the edge attribute.

After receiving the update messages from the source and destination nodes, the memory is updated through the GRU. The node memory of the last state is treated as the hidden state of the recurrent neural network, and the update message is the input of the model. This updates the previous hidden state of the node through the update message, and obtains the new memory of the nodes. The formula of the GRU is shown as follows.

$$z_t = \sigma(W_z \cdot msg_i(t) + U_z \cdot m_i(t-)) \tag{3}$$

$$r_t = \sigma(W_r \cdot msg_i(t) + U_r \cdot m_i(t-)) \tag{4}$$

$$m_i' = \tanh\left(W_x \cdot msg_i(t)\right) + r_t \cdot U_m \cdot m_i(t-) \tag{5}$$

$$m_i(t) = z_t \cdot m_i(t-) + (1 - z_t) \cdot m_i'(t) \tag{6}$$

where z_t is the update gate, r_t is the reset gate, σ is the sigmoid function, and $\{W_z, W_r, W_x\}$ and $\{U_z, U_r, U_m\}$ are learnable parameters.

3.2. Aggregation module

In this section, we introduce the aggregation module, which aggregates aggregated messages from influenced nodes to calculate node embedding. The structure of the aggregation module is shown in Fig. 3. The core unit of the aggregation module is the transformer layer. The transformer is used because 1) the transformer can be stacked in multiple layers to increase the expressive ability of the model, and 2) the importance of neighbors is learned by its self-attention mechanism. The aggregation phase consists of three steps: 1) obtain the time encoding from the timestamp, 2) calculate the hidden state of each node through the transformer, and 3) aggregate the hidden state to obtain the node embedding.

3.2.1. Time Encoding

Since the self-attention mechanism in the transformer layer cannot preserve time information, temporal information should be appended to the unprocessed vector before importing into the transformer layer. We slightly modified the time encoding proposed in TGAT [28] to preserve the periodicity.

$$enc(t) = \sin\left(W_t \cdot t + b_t\right) \tag{7}$$

where W_t is the weight of the time encoding unit, b_t is the bias, and t represents a certain timestamp.

3.2.2. The Transformer Layer

The transformer architecture consists of two main components: a self-attention layer and a feed forward network. We picked the top-k recently interacted neighbors as influenced nodes and concatenated the memory and edge features to form an aggregation message. Afterwards, the time encoding of the interaction time was added to the aggregation message, and the result was input into the transformer layer, shown as follows:

$$h_{ni}(t-) = (m_{ni}(t-)||e_{si}(t-)) + \lambda_{ni}(t-) \tag{8}$$

$$h_s(t-) = (m_s(t-))||mean\left(\{e_{si}(t-)\}_{i=o}^{n}\right) + \lambda_s(t-) \tag{9}$$

where $e_{si}(t-)$ represents the edge features between the influenced node and the interacting node, $||$ is the concatenating operation, $m_i(t-)$ is the memory of the node v_i before the current timestamp, and λ represents the time encoding.

In the self-attention layer, the inputs are mapped into three matrices, Q, K, and V, denoting the query, key, and value matrices, respectively. The output of the self-attention layer shows the following.

$$output = Softmax\left(\frac{QK^T}{d_k}\right)V \tag{10}$$

where $Q \in \mathbb{R}^{d_h \times d_Q}$, $Q \in \mathbb{K}^{d_h \times d_K}$, and $Q \in \mathbb{V}^{d_h \times d_V}$.

To avoid gradient disappearance and gradient explosion, the output of self-attention is added by the input of self-attention, and a layer normalization is the applied. Afterwards, the output of the self-attention layer is transformed through a feed forward neural network to obtain the output of the transformer layer, as follows:

$$h_i'(t) = \sigma(W_h \cdot h_i(t-) + b_h) \tag{11}$$

where $h_i(t)$ is the output of the self-attention layer.

Notice that the transformer layer can be stacked, where the input of the l-th layer is the output of the l-1-th layer. The output of the last transformer denotes the hidden state of nodes at timestamp t.

3.2.3. The Aggregator

After acquiring a hidden state of interacting node and influenced nodes, we needed to aggregate them to form node embedding. We adopted two widely applied aggregators, i.e., *SUM* and *MEAN*, in static graph neural network to achieve node embedding. Compared with the *MEAN* aggregator, the *SUM* aggregator preserves the number of influenced nodes because the addition result does not need to be divided by the number of neighbors. The equations of these two aggregators are as follows:

$$AGGR_{sum} = \sigma(W_{sum} \cdot (h_s + sum(\{h_{ni}\}))) \tag{12}$$

$$AGGR_{mean} = \sigma(W_{mean} \cdot (h_s||sum(\{h_{ni}\}))) \tag{13}$$

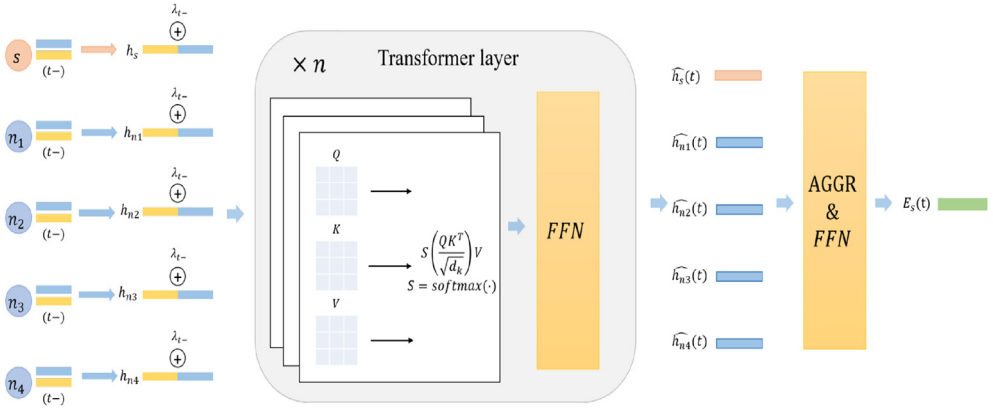

Fig. 3 The architecture of the aggregation module with three heads in self-attention layer.

3.3. The Propagation Module

In the previous section, we introduced an update module that updates the memory of interacting nodes and an aggregation module that aggregates the memory from influenced nodes. If the model only consists of these two modules, the training and inference phase will be time consuming. To overcome this, in this section, we introduce a propagation module that efficiently embeds the 2-order information into the memory of the influenced nodes.

In the directed graph, there are four types of interaction and influence: (1) the source node targets its influenced node, (2) the source node is targeted by its influenced node, (3) the destination node targets its influenced node, and 4) the destination node is targeted by its influenced node.

Similar to the update module, the GRU model was used as a propagator to update the memory of the influenced nodes. We used two GRUs to deal with the source node and destination node, respectively. One was used to update the memory of the influenced nodes related to the source node, i.e., Case (1) and Case (2), and the other was applied to the destination node, i.e., Case (3) and Case (4). The memory of the influenced nodes is seen as the hidden state of the GRU, and the propagation message is the input of that. We defined the propagation message as follows:

$$msg_{prop}(i,n) = concat([E_i(t), m_n(t-), e_{in}, \Delta t]) \tag{14}$$

where E_i is the embedding of the interacting node, $m_n(t-)$ is the memory of the influenced node before the current timestamp, e_in represents the edge feature between the interacting node v_i and the influenced node v_n, and Δt denotes the time encoding in the aggregation module.

3.4. Model Optimization

The proposed TGT model can be trained in supervised or self-supervised approaches. Here we introduce the self-supervised (dynamic link prediction) optimization procedure, i.e., to predict future interactions from a series of events observed in the past. For link prediction, all edges in the original graph are seen as positive instances, and negative samples should be manually introduced first. After negative sampling, the model concatenates a memory state, and edges feature in interactions to form update messages. Three modules are then sequentially performed to achieve node embedding and to update the memory state of the interacting nodes and influenced nodes. The probability of edge interaction is predicted by applying an MLP-based decoder through the concatenation of source node embedding and destination node embedding. Finally, a widely applied binary cross entropy is adopted to calculate the difference between the predicted probability and the ground truth. For an interaction (v_s, v_d, t), the probability of an interaction from node v_s to node v_d is represented as follows.

$$prob(v_s, v_d, t) = \sigma\big(W_p \cdot (E_s(t-)||E_d(t-)) + b_p\big) \tag{15}$$

where σ is the sigmoid function.

Our goal is to minimize the loss function that follows.

$$\sum_{e \in E(T)} L(e) \tag{16}$$

where e is a tuple of an edge, i.e., (v_s, v_d, t). The loss function $L(e)$ can be represented as follows.

$$\begin{aligned} L((v_s, v_d, t)) &= -\log\left(prob(v_s, v_d, t)\right) \\ &\quad -N \cdot E_{v_n \sim P_n(v)} \log\left(prob(v_s, v_d, t)\right) \end{aligned} \tag{17}$$

4. Experiments

In this section, we evaluate the proposed model by comparing our model with eleven baselines, including four static methods and seven start-of-the-art CTDG-based models. Link prediction and edge classification tasks were performed on different networks. The time complexity, parameter selection, and ablation study of the algorithm are discussed and analyzed.

4.1. Datasets

We used three datasets in our experiments: UCI, MovieLens-10 M, and Math-overflow.

UCI[1] [38] is a directed social network, that contains 1899 nodes and 59,835 edges. Each node denotes a student from University of California, Irvine, and an interaction represents a message communication between students where the timestamp corresponds to the time at which a message is delivered.

MovieLens-10 M[2] [39] is network of dynamic user-tag interactions, which consists of nearly 100,000 tags applied to 10,000 movies by 72,000 users. We picked a subset of the benchmark with about 1500 nodes and 50,000 interactions.

Math-overflow[3] [38] is a temporal network of interactions on Math Overflow. There are three types of interaction in the graph, i.e., a2q, c2q, and c2a, representing an answer to a question, a comment in response to a question, and a comment in response to an answer, respectively. The dataset contains nearly 25,000 nodes and 506,550 edges.

4.2. Baseline

In this section, we mainly introduce the comparison methods. These methods are state-of-the-art approaches for continuous time dynamic graphs (JODIE [29], DyRep [27], TGAT [28], TGN [22], APAN [20] and SGSketch [40]) as well as state-of-the-art models for static graphs (GCN [41], GAT [42], GraphSage [16], and the Graph transformer [42]). These models are described as follows.

- GCN [41] is a classic graph neural network designed for static graphs, and ignores temporal information.
- GAT [42] is widely applied on static graph and the attention mechanism is used to calculate the weight of a nodes neighbors.
- GraphSage [16] is similar to GAT, in that it obtains node representation by aggregating information from neighbors.
- The graph transformer [42] is a novel message passing algorithm that incorporates feature embedding and label embedding as the input of the transformer.
- JODIE [29] is a CTCG-based model where the graph can be represented as a series of interaction events, and a recurrent neural network is applied to learn node embedding.

[1] https://snap.stanford.edu/data/CollegeMsg.html
[2] https://grouplens.org/datasets/movielens/10 m/
[3] https://snap.stanford.edu/data/sx-mathoverflow.html

- DyRep [27] treats representation learning for dynamic graphs as a latent mediation process from topological evolution and activities between nodes simultaneously.
- TGAT [28] introduces time encoding into a CTDG-based model to explicitly learn from time information in the aggregation phase.
- TGN [22] is a state-of-the-art CTCG-based embedding method, which generalizes the concept of RNN-based dynamic graph embedding method.
- TGT is our proposed model, which is composed of an update module, an aggregation module, and a propagation module. The suffix shows the type of aggregator.
- APAN [20] is an asynchronous CTDG-based model framework for real-time temporal graph embedding. It aims at transforming traditional CTDG algorithms to adapt to online deployment and real-time inference on Internet platforms.
- SGSketch [40] is a highly efficient streaming graph embedding technique via incremental neighborhood sketching. It can not only generate high-quality node embeddings from a streaming graph by gradually forgetting outdated streaming edges, but also efficiently update the generated node embeddings via an incremental embedding updating mechanism.

4.3. Link Prediction

4.3.1. Experimental Setting

The dynamic link prediction task was performed to predict when a new interaction will emerge given a sequence of interaction events as history. In the experiment, we use 70% of the total edges as the training set, 15% as the validation set, and the last 15% as the testing set. We tuned model parameters on the validation set, and the early stop was 5 epochs. The number of negative sampling was the same as that for the training edges. For the dynamic graph neural network, the batch size was set to 20 for the UCI and MoiveLen-10 M datasets and to 50 for the Math-overflow dataset. We measured the performance of the baselines in terms of AUC, F1, and MRR.

4.3.2. Results

Table 3 lists the results of various models running on these three datasets. For the UCI dataset, all dynamic graph embedding methods outperformed the static graph embedding methods that ignore dynamic information in most evaluation metrics. Graphsage had the worst performance, and TGT-sum had the best. In terms of AUC, F1, and MRR, our algorithm improves performance by 47.2%, 36.4%, and 92.4%, respectively. The performance of TGT-mean was closest to TGT-sum. Experimental results show the representation learning ability of the proposed model in small graphs.

For the MovieLen-10 M dataset, APAN, GCN, GAT, and TGT performed best, which is consistent with the experimental results of the UCI dataset. The dynamic network representation was better than the static network representation. The TGT model proposed in this paper had the best performance in terms of AUC and F1. Notably, the average aggregator outperformed the sum aggregator. For the Math-overflow dataset, the transformer had the worst performance and TGT-mean had the best. In terms of AUC, F1 and MRR, our algorithm improves performance by 46.5%, 24.1% and 97.2%, respectively.

In summary, dynamic graph embedding achieve a better performance than static graph embedding. There are two main reasons: 1) static graph embedding methods face difficulties in large graphs, and 2) dynamic graph embedding methods learn the time information in fine-grained contexts.

4.4. Edge Classification

4.4.1. Experimental Setting

To evaluate the generalization of the proposed model, we conducted experiments at different training ratios; 40% of the edges were regarded as the training set, and the remaining edges were equally divided into validation and test sets.

4.4.2. Results

Fig. 4 shows the performance of the proposed TGT model against four state-of-the-art static and dynamic graph embedding methods on the Math-overflow dataset. We observed the following:

The TGT model greatly outperformed all baselines in the edge classification task. With the increase in the training ratio, the performance of the proposed model remained high, indicating the generalization ability of the proposed model.

When training ratio was low, the static graph embedding methods were better than the dynamic graph embedding

Table 3 Performance comparison of link prediction in terms of AUC, F1, and MRR.

Methods	UCI			MovieLen-10 M			Math-overflow		
	AUC	F1	MRR	AUC	F1	MRR	AUC	F1	MRR
GCN	0.9321	0.3427	0.0138	0.7324	0.3361	0.0167	0.9232	0.5941	0.0083
GAT	0.8556	0.6934	0.0132	0.7301	0.4304	0.0088	0.7629	0.7242	0.0014
GraphSAGE	0.5121	0.5815	0.0060	0.5332	0.4911	0.0020	0.5098	0.6622	0.0007
Transformer	0.5509	0.6174	0.0024	0.5515	0.4910	0.0022	0.5058	0.6699	0.0007
JODIE	0.8846	0.7820	0.0374	0.5915	0.5524	0.0019	0.9142	0.8158	0.2514
DyRep	0.8782	0.7439	0.0565	0.5573	0.5160	0.0032	0.8986	0.7646	0.2580
APAN	0.7925	0.7660	0.0412	0.5823	0.4782	0.0012	0.8652	0.7824	0.2107
TGN	0.9279	0.7587	0.0130	0.7509	0.6449	0.0010	0.9054	0.6337	0.1715
TGT-mean	0.9689	0.9092	0.0435	0.8466	0.7393	0.0005	0.9461	0.8830	0.2634
TGT-sum	0.9707	0.9175	0.0797	0.8358	0.7368	0.0010	0.9496	0.8821	0.2640
APAN	0.9015	0.8275	0.0547	0.7387	0.6154	0.0007	0.7949	0.7312	0.2174
SGSketch	0.7174	0.8109	0.0407	0.6417	0.5987	0.0005	0.7891	0.6008	0.1573

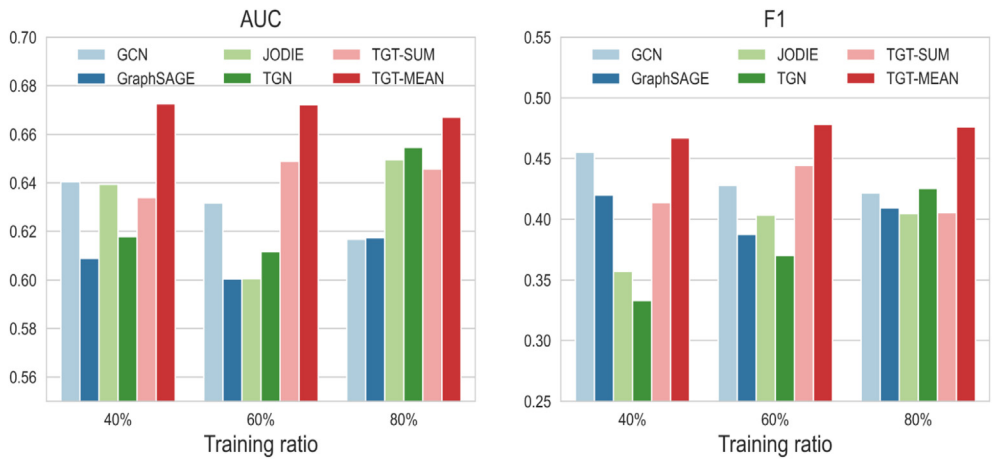

Fig. 4 Performance comparison of edge classification on math-overflow dataset in terms of AUC and F1.

methods on the Math-overflow dataset. However, the dynamic methods outperformed the static techniques with the increase in the training ratio, showing that, if a dynamic graph neural network observes more instances with a timestamp in the training phase, the performance will be enhanced.

4.5. Time Complexity Analysis

We compared the efficiency of TGT with that of state-of-the-art baselines on the UCI dataset, and AUC was used as the metric. Fig. 5(a) and (b) show the time consumption of the training phase and inference phase, respectively. Notice that DyRep and JODIE cannot be expanded to multiple layers, i.e., these two models have only one layer.

We observed that the proposed TGT model outperformed all baselines in terms of AUC. Because of the existence of the propagation mechanism, which propagates the 2-order information to neighbors, a desirable performance and a low time consumption were achieved. In the inference phase, the TGT was the fastest model, and meets the requirements of real-time inference. For baselines, the performance of TGN and TGAT was improved when models became deeper. Although performance was improved, time consumption was greatly increased, especially in the inference phase. In online environments, inference time is more important than training time. For example, users cannot tolerate waiting for too long time for recommendations in e-commence.

4.6. Module Analysis

To analyze the effect of the main modules, we performed an ablation study on the proposed model. In particular, we defined three variants by removing specific modules in the proposed model:

TGT-prop: This model only consists of an update module and an aggregation module. The propagation module was removed. The aggregator used in the model is a MEAN aggregator.

TGT-update: This model ignores the update module that implicitly learns the time information in the model. The aggregator is a MEAN aggregator.

TGT-aggr: The transformer here has the ability to aggregate the information of influenced nodes. In this model, we evaluate the impact of the aggregator by removing it.

We performed the link prediction task to evaluate the performance of each variant; the experimental results are shown in Table 4. We conclude that all components are important

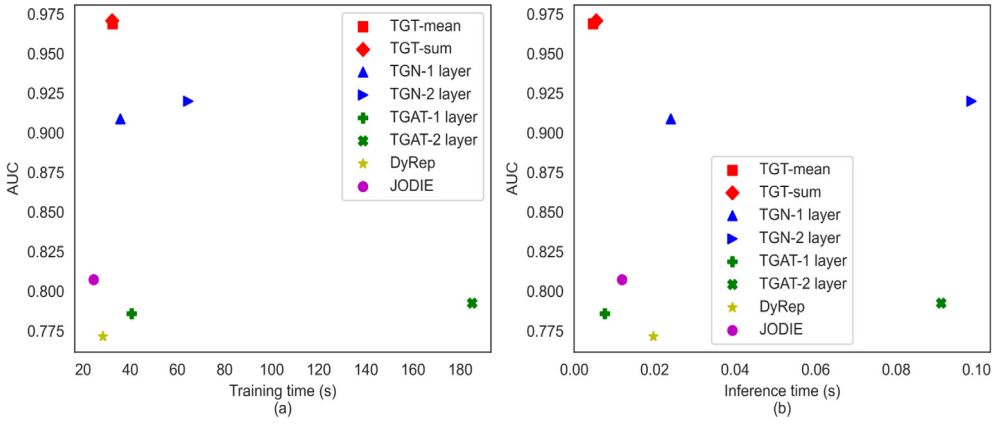

Fig. 5 The time complexty analysis on UCI dataset. (a) Training time per epoch. (b) Inference time per batch.

Table 4 Ablation study of link prediction in terms of AUC, AP, and F1.

Methods	UCI			MovieLen-10 M			Math-overflow		
	AUC	AP	F1	AUC	AP	F1	AUC	AP	F1
TGT-mean	0.9689	0.9708	0.9092	0.8466	0.8820	0.7393	0.9461	0.9450	0.8830
TGT-prop	0.7297	0.6996	0.7290	0.5974	0.6276	0.6599	0.7835	0.7577	0.6939
TGT-update	0.8726	0.8632	0.8082	0.5868	0.6416	0.6675	0.7691	0.7012	0.7602
TGT-aggr	0.9677	0.9702	0.9131	0.8382	0.8730	0.7485	0.9124	0.9043	0.8304

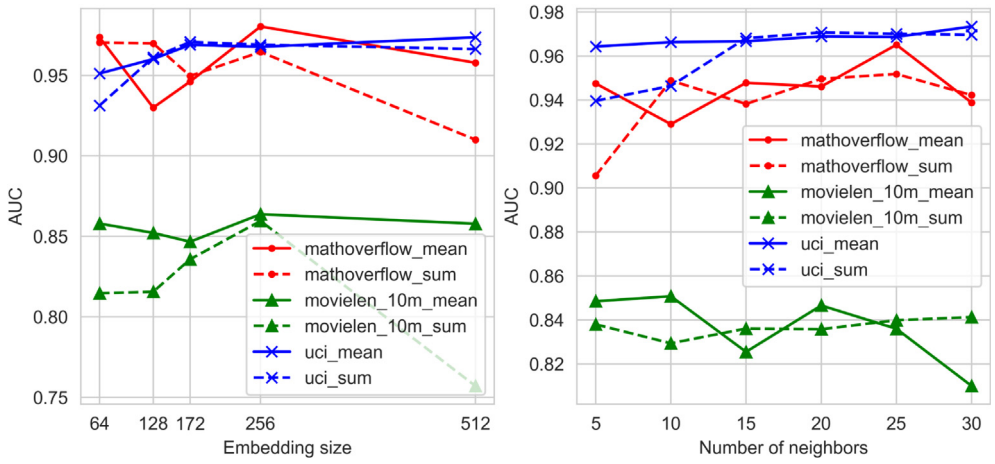

Fig. 6 The time complexty analysis on UCI dataset. (a) Training time per epoch. (b) Inference time per batch.

to the model, and we can draw the following points: 1) introducing 2-order information (i.e., propagation) is necessary, 2) the information from neighbors should be aggregated, and 3) temporal information plays a vital role.

4.7. Hyperparameter Analysis

The proposed model has two significant hyperparameters, i.e., embedding size and the number of neighbors. We analyze the impact of these two hyperparameters in this subsection. We conduct the analysis for the dynamic link prediction task on three benchmarks and two aggregators with the AUC measurements.

The analysis of embedding size is shown in Fig. 6 (a), where the embedding sizes are set to 64, 128, 172, 256. for the MovieLen-10 M and Math-overflow dataset, the best performance appears when the embedding size is 256; for the UCI dataset, the best result is achieved when the embedding size is 172. This shows that a large embedding size does not always lead to an improved performance, since redundant information may be stored. When the embedding size is 512, the performance of the model drops, which also proves that redundant information contained in the embedding vector hinders performance.

Fig. 6 (b) shows the performance of the proposed model with different number of neighbors. For different datasets, the best number of neighbors varies. If the number is too large, useless neighbors that introduce noises may be aggregated, leading to a drop in performance. If the number is too small, important neighbors may not be aggregated.

The SUM aggregator, unlike the MEAN aggregator, introduces the number of neighbors into node embedding. For different types of dataset, the influence of these neighbors differ. For the UCI dataset, the edge represents the behavior of sending a message. Popular users generally have more messages to send or receive. In such a situation, the performance of a SUM aggregator that explicitly aggregates the number of neighbors in node embedding is dramatically influenced by this hyperparameter. However, in the MovieLen-10 M dataset, the impact of the SUM aggregator is lower than that of the MEAN aggregator because the embedding of a movie is more influenced by the type of person that tags the movie instead of the number of tags.

5. Discussion and Conclusion

In this paper, we propose a novel graph neural network framework, called a temporal graph transformer (TGT), that learns dynamic node representation from a sequence of time-stamped events. TGT consists of three main modules, i.e., an update module, an aggregation module, and a propagation module. When a new interaction is formed, the update module updates the interacting node memory on the basis of the last memory of the node. Moreover, using the memory in conjunction with sampling the most recent neighbors reduces the number of neighbors needed to achieve the best performance when used. The aggregation module then aggregates the 1-hop temporal neighbors memory to obtain node embedding. This indicates that the ability to obtain more recent information through the graph and the ability to select which neighbors are the most important are critical factors in the performance of a model. Finally, the propagation module updates the neighbors memory through the representation of the source node or destination node obtained from the aggregation module to propagate 2-order information.

We performed link prediction and edge classification on three real-world benchmarks to demonstrate the effectiveness of the proposed model. The experimental result shows that the model effectively improves performance and dramatically reduces the time consumption when learning 2-hop neighbors information. Detailed ablation studies demonstrated the importance of the memory and its associated modules for storing long-term information, as well as the importance of the graph-based embedding module for generating up-to-date node embeddings.

In future work, we will try to explain how the aggregation diffusion mechanism works from a theoretical perspective. In addition, the representation of interactive nodes and affected nodes can be updated more effectively if they are combined with the influence weight, thus improving the representation ability of the model. Our goal is to apply TGT in fields such as social sciences, recommendation systems, and biological interaction networks, opening up future research directions for exploring areas where our model may be suitable for more advanced ideas and methods.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by Shenzhen Fundamental Research Program under Grant No. 20210317191843003 and Shaanxi Provincial Key R&D Program under Grant No. 2021ZDLGY05-01.

References

[1] Usman Ahmed, Gautam Srivastava, Youcef Djenouri, Jerry Chun-Wei Lin, Knowledge graph based trajectory outlier detection in sustainable smart cities, Sustain. Cities Soc. 78 (2022) 103580.

[2] Youcef Djenouri, Gautam Srivastava, Asma Belhadi, Jerry Chun-Wei Lin, Intelligent blockchain management for distributed knowledge graphs in iot 5g environments, Trans. Emerg. Telecommun. Technol., page e4332, 2021.

[3] Xu. Da, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, Kannan Achan, Product knowledge graph embedding for e-commerce, in: Proceedings of the 13th international conference on web search and data mining, 2020, pp. 672–680.

[4] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, Dawei Yin, Graph neural networks for social recommendation, in: The World Wide Web Conference, pages 417–426, 2019.

[5] Kathryn Tunyasuvunakool, Jonas Adler, Wu. Zachary, Tim Green, Michal Zielinski, Augustin Žídek, Alex Bridgland, Andrew Cowie, Clemens Meyer, Agata Laydon, et al, Highly accurate protein structure prediction for the human proteome, Nature 596 (7873) (2021) 590–596.

[6] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, 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.

[7] Junshan Wang, Guojie Song, Wu. Yi, Liang Wang, Streaming graph neural networks via continual learning, in: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, 2020, pp. 1515–1524.

[8] Yao Ma, Ziyi Guo, Zhaocun Ren, Jiliang Tang, Dawei Yin, Streaming graph neural networks, in: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2020, pp. 719–728.

[9] Dawei Zhou, Lecheng Zheng, Jiawei Han, Jingrui He, A data-driven graph generative model for temporal interaction networks, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 401–411.

[10] Ranran Bian, Yun Sing Koh, Gillian Dobbie, Anna Divoli, Network embedding and change modeling in dynamic heterogeneous networks, in: Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2019, pp. 861–864.

[11] Mengyuan Lee, Yu. Guanding, Huaiyu Dai, Decentralized inference with graph neural networks in wireless communication systems, IEEE Trans. Mob. Comput. (2021).

[12] Martin Happ, Matthias Herlich, Christian Maier, Jia Lei Du, Peter Dorfinger, Graph-neural-network-based delay estimation for communication networks with heterogeneous scheduling policies, ITU J. Future Evol. Technol., 2(4), 2021.

[13] Hu. Ting-Kuei, Fernando Gama, Tianlong Chen, Wenqing Zheng, Atlas Wang, Alejandro R. Ribeiro, Brian M. Sadler, Scalable perception-action-communication loops with convolutional and graph neural networks, IEEE Transactions on Signal and Information Processing over Networks, 2021.

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

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

[16] William L. Hamilton, Rex Ying, Jure Leskovec, Inductive representation learning on large graphs, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 1025–1035.

[17] Benedek Rozemberczki, Carl Allen, Rik Sarkar, Multi-scale attributed node embedding, J. Complex Networks 9 (2) (2021) cnab014.

[18] Guotong Xue, Ming Zhong, Jianxin Li, Jia Chen, Chengshuai Zhai, Ruochen Kong, Dynamic network embedding survey, Neurocomputing 472 (2022) 212–223.

[19] Palash Goyal, Sujit Rokka Chhetri, Arquimedes Canedo, dyngraph2vec: Capturing network dynamics using dynamic graph representation learning, Knowledge-Based Syst., 187:104816, 2020.

[20] Xuhong Wang, Ding Lyu, Mengjian Li, Yang Xia, Qi Yang, Xinwen Wang, Xinguang Wang, Ping Cui, Yupu Yang, Bowen Sun, et al., Apan: Asynchronous propagation attention network for real-time temporal graph embedding, in: Proceedings of the 2021 International Conference on Management of Data, pages 2628–2638, 2021.

[21] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. J. Mach. Learn. Res., 21(70):1–73, 2020.

[22] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, M. Bronstein, Temporal graph networks for deep learning on dynamic graphs, 2020.

[23] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, Charles Leiserson, Evolvegcn: Evolving graph convolutional networks for dynamic graphs, in: Proceedings of

the AAAI Conference on Artificial Intelligence, volume 34, pages 5363–5370, 2020.
[24] Aravind Sankar, Wu. Yanhong, Liang Gou, Wei Zhang, 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, 2020, pp. 519–527.
[25] Mingyi Liu, Zhiying Tu, Xiaofei Xu, Zhongjie Wang, Learning representation over dynamic graph using aggregation-diffusion mechanism. arXiv preprint arXiv:2106.01678, 2021.
[26] Hao Peng, Renyu Yang, Zheng Wang, Jianxin Li, Lifang He, Philip Yu, Albert Zomaya, Raj Ranjan, Lime: Low-cost incremental learning for dynamic heterogeneous information networks, IEEE Trans. Comput., 2021.
[27] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, Hongyuan Zha, Dyrep: Learning representations over dynamic graphs, International conference on learning representations, 2019.
[28] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, Kannan Achan, Inductive representation learning on temporal graphs. arXiv preprint arXiv:2002.07962, 2020.
[29] Srijan Kumar, Xikun Zhang, Jure Leskovec, Predicting dynamic embedding trajectory in temporal interaction networks, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 1269–1278.
[30] Shi Dong, Ping Wang, Khushnood Abbas, A survey on deep learning and its applications, Comput. Sci. Rev. 40 (2021) 100379.
[31] Apurva Narayan, Peter HO'N Roe, Learning graph dynamics using deep neural networks, IFAC-PapersOnLine 51 (2) (2018) 433–438.
[32] Xuebin Zheng, Bingxin Zhou, Ming Li, Yu Guang Wang, Junbin Gao, Mathnet: Haar-like wavelet multiresolution-analysis for graph representation and learning. arXiv preprint arXiv:2007.11202, 2020.
[33] Palash Goyal, Nitin Kamra, Xinran He, Yan Liu, Dyngem: Deep embedding method for dynamic graphs. arXiv preprint arXiv:1805.11273, 2018.
[34] Tianyu Zhu, Leilei Sun, Guoqing Chen, Embedding disentanglement in graph convolutional networks for recommendation, IEEE Trans. Knowl. Data Eng. (2021).
[35] Sepp Hochreiter, Jürgen Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[36] Yicong Huang, Yu. Zhuliang, Representation learning for dynamic functional connectivities via variational dynamic graph latent variable models, Entropy 24 (2) (2022) 152.
[37] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555, 2014.
[38] Pietro Panzarasa, Tore Opsahl, Kathleen M. Carley, 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.
[39] F. Maxwell Harper, Joseph A Konstan, The movielens datasets: History and context, Acm transactions on interactive intelligent systems (tiis), 5, 2015, pp. 1–19.
[40] Dingqi Yang, Bingqing Qu, Jie Yang, Liang Wang, Philipe Cudre-Mauroux, Streaming graph embeddings via incremental neighborhood sketching, IEEE Trans. Knowl. Data Eng., 2022.
[41] Thomas N. Kipf, Max Welling, Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.
[42] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, Yu Sun, Masked label prediction: Unified message passing model for semi-supervised classification. arXiv preprint arXiv:2009.03509, 2020.