# Neighborhood Extended Dynamic Graph Neural Network

Da Yu
Junli Wang
Changjun Jiang
1933025@tongji.edu.cn
junliwang@tongji.edu.cn
cjjiang@tongji.edu.cn
Key Laboratory of Embedded System and Service Computing(Tongji University), Ministry of Education
Shanghai, China

## ABSTRACT

Representation learning on dynamic graphs has drawn much attention due to its ability to learn hidden relationships as well as capture temporal patterns in graphs. It can be applied to represent a broad spectrum of graph-based data like social networks and further use the learned representations to solve the downstream tasks including link prediction and edge classification. Although some approaches have been proposed for dynamic graphs in recent years, most of them paid little attention to the evolution of the entire graph topology, leading to the lack of global information of what happened in nodes' neighborhoods during their update. We propose NEDGNN, a novel Neighborhood Extended Dynamic Graph Neural Network on dynamic graphs represented as sequences of time-stamped events. We introduce a temporal attention propagation module to generate messages for n-hop neighbors through a self-attention mechanism, which can help disseminate information among nodes' n-hop neighbors. Besides, a FIFO message box module is also applied to gain some time efficiency. Due to the introduction of these modules, NEDGNN outperforms many state-of-the-art baselines in several tasks. We also perform a detailed ablation study to test the effectiveness and time cost of each module.

## CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Learning latent representations**.

## KEYWORDS

dynamic graphs, graph neural networks, graph embedding, link prediction

**ACM Reference Format:**
Da Yu, Junli Wang, and Changjun Jiang. 2022. Neighborhood Extended Dynamic Graph Neural Network. In *2022 14th International Conference on Machine Learning and Computing (ICMLC) (ICMLC 2022), February 18–21, 2022, Guangzhou, China.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3529836.3529851

## 1 INTRODUCTION

A graph is a structure composed of a set of objects and relations between them. Many real-life data like user-item interactions, transaction records and social networks can be naturally represented as graphs. In recent years, graph data is drawing more and more attention in the field of machine learning. With the help of deep learning technology, there are many solutions to this problem [1, 22]. Message passing mechanism [8] is often used to aggregate the neighborhood information of the nodes in the graph and encode the nodes into low-dimensional vector embeddings. Taking these embeddings as input, node classification [12, 14], graph classification [7], link prediction [30] are then treated as the downstream tasks.

Representation learning on dynamic graphs is relatively recent, and most approaches model the dynamic graph as a discrete-time dynamic graph consisting of a sequence of snapshots of the graph [2, 6, 20]. Such methods are still unsuitable for the streaming scenario [10] (newly observed data are continuously streamed to the model.). However, in recent two years, many DGNN models [17, 23, 24, 30] have been proposed for the streaming scenario. These models first transform the graph data into continuous-time dynamic graphs, and then use the sequence models to learn the evolving nature of graphs. The emergence of these models confirms that the time feature in graph data can improve the performance of different tasks on graphs.

However, there are still some defects in the existing DGNN models for the streaming scenario. When an interaction happens, most DGNN models only updates the states of nodes that directly connected to it(see Fig 1 left). This implies that the receptive field of the message propagation module is zero-hop in terms of graph structure, limiting their ability to explore the correlation between graph structure information and the evolving nature of graph topology. We try to fix this problem by propagating the influence of an interaction through past interactions to its arbitrarily large neighborhood(see Fig 1 right).

In this paper, a neighborhood extended dynamic graph neural network (NEDGNN) model is proposed. In our model, the neighborhood of each interaction is extended so that the evolving topology information can be spread further while updating the local node vector embeddings. The contributions of this paper are summarized as follows:

- We propose a temporal attention message propagation mechanism, which combines temporal coding with a self-attention mechanism to spread an event's influence to its arbitrarily large neighborhood. To our limited knowledge, we believe

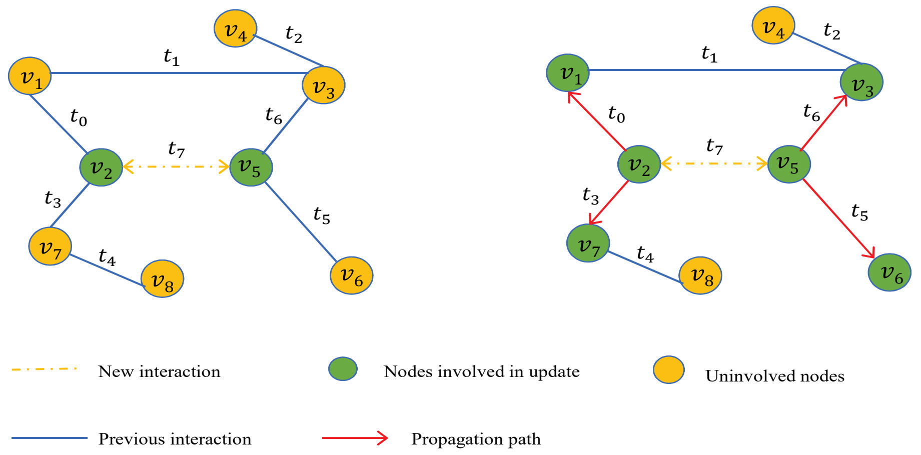

**Figure 1: Left: Without the propagation module, the interaction $(v_2, v_5, t_7)$ will only lead to the update of direct nodes $v_2$ and $v_5$. Right: With a one-layer propagation module, the information of the interaction $(v_2, v_5, t_7)$ can be propagated farther to their one-hop neighbors through the path in graph.**

our work is the first attempt to propagate the influence of an interaction to a larger neighborhood.

- Due to the introduction of the temporal attention message propagation mechanism, the time efficiency of our model has greatly reduced. We propose a FIFO message box structure to alleviate the time complexity problem to some extend.
- We demonstrate the feasibility and the effectiveness of the proposed model by experimenting on several graph datasets and tasks. Especially in inductive settings, NEDGNN outperforms many state-of-the-art methods.

## 2 BACKGROUND AND RELATED WORK

In this section, background, problem definition and some related work will be introduced. Here, some symbols and definitions are borrowed from TGN [17].

### 2.1 Background

*2.1.1 Learning on Static Graphs.* A static graph $G = (V, E)$ comprises node-set $V = 1, 2, ..., n$ and edge set $E = V \times V$ with features of nodes and edges, denoted by $v_i, e_{ij}$ for all $i, j = 1, 2, ..., n$, respectively. A typical spatial GNN learns vector embeddings of nodes by aggregating neighbor information in the following form:

$$h_i = \sum_{j \in N_i} f\left(m_{ij}, v_i\right), \quad m_{ij} = msg\left(v_i, v_j, e_{ij}\right)$$

where $m_{ij}$is a message passing to the neighbors $j$ of $i$, here $N_i = \{j \mid (i, j) \in E\}$, $f$ and $msg$ are learnable aggregation function and message generating function respectively. This form is called the message passing neural network [8].

*2.1.2 Dynamic Graphs.* Generally speaking, dynamic graphs can be represented in two forms [10]. *Discrete-time dynamic graphs* (DTDGs) are sequences of snapshots taken from static graphs sampled at fixed-spaced times. *Continuous-time dynamic graphs* (CTDGs) are lists of events or observations, including edge or node addition, edge or node deletion, and edge or node feature changes, which are more suitable for modeling data in the streaming scenario.

We adopt CTDGs to represent dynamic graphs. Hence we can represent our temporal graph as a sequence of timestamped events $O = \{o(t_0), o(t_1), ...\}$, where each $o(t_k)$ is addition or change of a node or an interaction (edge) between nodes at time $t_k$. A node-wise event $o(t_k) = v_i(t_k)$ represents the addition or deletion of a node, where $v_i(t_k)$ denotes the feature vector of node $i$. If the index $i$ appears for the first time, the event creates node $i$ along with feature $v_i$, otherwise it updates the features. An edge-wise(interaction) event $o(t_k) = e_{ij}(t_k)$ represents an interaction between two nodes, where $e_{ij}(t_k)$ denotes the feature vector of the interaction between node $i$ and $j$. If the index $i$ or $j$ appears for the first time, the event creates node $i$ or $j$ with an empty feature vector before the addition of interaction between nodes. A snapshot of the temporal graph $O$ at time t is the (multi-)graph $G(t) = \{V(t), E(t)\}$, where $V(t) = \{v_i(t_k) \mid \exists v_i(t_k) \in O, t_k \in [0, t]\}$, $E(t) = \left\{e_{ij}(t_k) \mid \exists e_{ij}(t_k) \in O, t_k \in [0, t]\right\}$ represents the temporal set of vertices and edges, respectively, and $N_i(t) = \{j \mid (i, j) \in E(t)\}$ represents the neighbor set of node $i$ at time $t$.

### 2.2 Related Work

**Representation Learning for Static Graphs.** GNN models for static graphs can be of spatial and spectral models. Spectral methods learn the vector embeddings of nodes by approximating, projecting, or extending the Laplacian matrix in the spectral domain [12, 15]. Because its calculation process depends on the specific Laplacian matrix, it is infeasible to generalize to dynamic graphs. Spatial methods gather the neighbor information to the current node through local neighborhood aggregation to learn the node vector embeddings [4, 14, 26]. This kind of methods does not directly learn the vector embeddings of nodes but learns the mapping functions of node embeddings, enabling them to inductively generate embeddings for unseen nodes based on their features and be easily extended to dynamic graph settings.

**Representation Learning for Dynamic Graphs.** Most early models for dynamic graphs are designed for DTDGs because many methods for static graphs can be naturally inherited into DTDGs [2, 16, 18]. There also exist some approaches that leverage or extend the random walk models for static graphs to dynamic graphs by constraining the random walks to respect time [6, 28]. Representation learning models for CTDGs based on sequence models have become popular in recent years. These models use recurrent neural networks (RNNs) to update the vector embeddings of source and target nodes every time new interactions appear. A general encoder-decoder framework for DGNNs is proposed in TGN [17], which is composed of a memory updater module, an embedding module, a message aggregator module and a message function module. Many previous models [23, 24, 30] can be regarded as variants of TGN model. Similar models can also be applied to dynamic knowledge graphs [5]. Such approaches for CTDGs update node embeddings only when new interactions appear, which can only help solve the very local problem of node evolution through interactions. However, little attention is paid to the global graph topology evolution, which leads to two major problems. First, the expression ability of the model is restricted. When dealing with data whose global topology evolves rapidly, the model may perform poorly. Second, it will also lead to a staleness problem (node embeddings become stale) [10].

## 3 NEIGHBORHOOD EXTENDED DYNAMIC GRAPH NEURAL NETWORK

We will generally follow the TGN framework and incorporate some additional modules like message boxes and the propagation module. The encoder of our model aims to encode a continuous-time dynamic graph into node vector embeddings $Z(t) = (z_1(t), z_2(t), \dots, z_{n(t)}(t))$, where $n(t)$ represents the number of nodes at time $t$.

Most DGNN models have paid little attention to the evolution of the graph topology in a larger neighborhood. When new interactions emerge, only target and source nodes get updated, but nodes in their neighborhood are left behind. We try to solve this problem in this paper. Firstly, we combine the message generation module with the neighbor aggregation mechanism so that the information of other nodes in the neighborhood can be utilized when the message is generated. Secondly, we propose a new temporal attention message propagation module so that other nodes in the neighborhood can receive the messages and update their states accordingly. We will mainly focus on these two modules, but before detailing them, we first elaborate the information stored in each node, which is the memory module and message box structure. If we consider the message generation and propagation module as a black box temporarily, the overall framework of NEDGNN can be shown in Fig 2.

### 3.1 Message Box Module

In this section, we describe the information stored in each node and how each node's state is updated based on the information. For the memory part, we follow the design in TGN framework [17]. The memory vector $s_i(t)$ representing the state of node $i$ at time $t$ is stored in node $i$. We denote memory matrix for all nodes by $S(t) = [s_1(t), s_2(t), \dots, s_{n(t)}(t)]$, where $n(t)$ is the number of nodes at time $t$. Aside from memory, there is a message box for each node, which stores messages received from other nodes in a period of time. We denote $B_i(t) = \{m_i(t_1), m_i(t_4), \dots\}$ the message box for node $i$ at time $t$, where $m_i(t_k)$ represents the message vector received by node $i$ at time $t$ from another node. Due to memory constraints, each node's message box has a capacity. When a node's message box reaches its capacity limit, a part of messages will be discarded based on the principle of first in first out. Besides, we will also clear the messages we have already used. With memory and message box module, nodes can be updated in the following form:

$$s_i(t) = \text{Update}\left(s_i\left(t^-\right), \bar{m}_i(t)\right) \tag{1}$$

$$\bar{m}_i(t) = \text{Agg}\left(B_i(t)\right) = \text{Agg}\left(m_i\left(t_1\right), m_i\left(t_4\right), \dots\right) \tag{2}$$

Here, Update is a memory update function. An RNN such as LSTM [21] or GRU [13] is a good choice for the updater. $t^-$ means the latest updating time before time t. Agg is an aggregation function. Max or mean function is often chosen as aggregation function for the sake of simplicity. $\bar{m}_i(t)$ represents the aggregated message.

### 3.2 Message Generation Module

The message generation module uses the information of the new interactions and source and target nodes to generate the initial message for the message propagation module. Unlike the previous methods, we use the neighbor aggregation mechanism widely used in static graphs to make the generated initial message rich in the neighborhood information of the source and target nodes. For new interaction $(i, j, e_{ij}, t)$, the initial message for source and target nodes can be computed as follows:

$$m_i^0(t) = \left(z_i \left\| z_j \right\| e_{ij} \| \phi(t)\right), m_j^0(t) = \left(z_j \left\| z_i \right\| e_{ij} \| \phi(t)\right) \tag{3}$$

$$z_i = \text{Gat}\left(s_i\left(t^-\right), s_{N_i(t)}\left(t^-\right), t\right) \tag{4}$$

$$z_j = \text{Gat}\left(s_j\left(t^-\right), s_{N_j(t)}\left(t^-\right), t\right) \tag{5}$$

where $||$ is the concatenation operator, $\phi(\cdot)$ represents a generic time encoding proposed in [9] and used in TGAT [29], Gat$(\cdot)$ is a static neighbor aggregation mechanism [26]. $s_{N_i(t)}$ and $s_{N_j(t)}$ represent the memory vectors of neighbors of node $i$ and node $j$.

### 3.3 Temporal Attention Message Propagation Module

After the initial messages $m_i^0$ and $m_j^0$ are generated by the message generation module, the information of the new interactions will be propagated to not only the source and target nodes but also the nodes in their neighborhood through the message propagation module. It is necessary to transmit the information of new interactions to the neighborhood of the source nodes and the target nodes. The occurrence of some important interactions may have a profound impact, which requires the model to distinguish the importance of different interactions. Considering this, we propose a temporal attention message propagation module which is the core module of the NEDGNN model and absent in the TGN framework.

Here we present a single layer of temporal attention message propagation module. Considering a single node, it receives a message from one of its neighbors, modifies the message and passes it

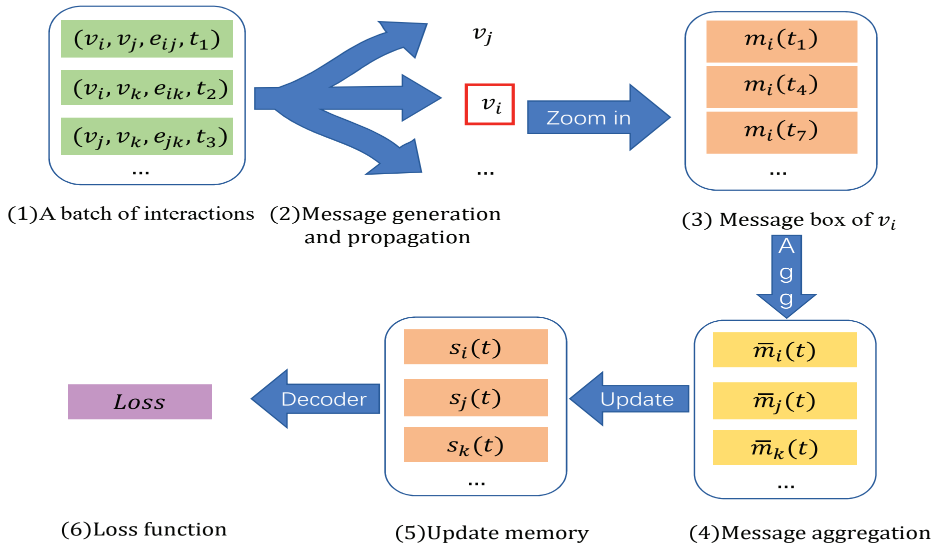

**Figure 2: For efficiency, we adopt the batch processing method, that is, processing multiple edges at a time. (1) shows a batch of new interactions from time $t_1$ till time $t$ (2) Messages generated by new interactions propagate to other nodes through message generation and propagation module (3) Node $v_i$ receives messages from other nodes and stores them in the message box temporarily. Here we focus on the message box of $v_i$. (4) Messages in the box of each node are aggregated to get the aggregated message vectors. (5) Each node updates itself using the aggregated message vector and its state before time $t$.**

on(like in 1 right). A single layer of propagation module is designed for a single node. It aims to utilize the received message to generate different new messages for different other neighbors. We think it is necessary to generate different messages since different neighbors have different memory vectors and time feature. The detailed computation process of a single layer is shown below.

Assume the current node is $i$, for each neighbor $k$ of $i$ at time t, the inputs to the $l$-th layer are the message of the current layer $m_i^l(t)$, the memory vectors of both the current node and neighbor $s_i(t), s_k(t)$, the timestamp $t_k$ of the interaction between current node $i$ and neighbor $k$ together with its feature $e_{ik}(t_k)$. The messages of the next layer can be calculated in the form:

$$M^{l+1}(t) = \text{MultiHeadAttention}^l \left(Q^l(t), K^l(t), V^l(t)\right) \quad (6)$$

$$Q^l(t) = K^l(t) = V^l(t) = F^l(t) \quad (7)$$

$$F^l(t) = \left[f_a^l(t), f_b^l(t), \dots\right] \forall a, b, \dots \in N_i(t) \quad (8)$$

$$f_k^l(t) = \text{mlp}\left(m_i^l(t) \,\|s_i(t)\|\, e_{ik}(t_k) \,\|s_k(t)\|\, \phi(t_k)\right), k \in N_i(t) \quad (9)$$

Here $a, b, \dots \in N_i(t)$ are neighbors of the current node $i$, $M$ represents the message matrix $(m_a, m_b, \dots)$, and an mlp is used to aggregate information and transform dimension. We perform self-multi-head-attention [25] where the queries $Q^l(t)$ are vectors after transformation, and the keys $K^l(t)$ and values $V^l(t)$ are the same as queries. Thus, each newly generated message can be regarded as a weighted sum of the old message combined with different information of different neighbors, in which the weights are also learned from the information. The above formulation is inspired by the temporal graph attention layer first proposed in TGAT [29]. Different from the TGAT layer, our method describes how a single node $i$ propagates messages to its neighborhood. For multiple nodes in the same layer, the above operation can be repeated several times to propagate the messages of all nodes from one layer to the next layer. We can stack this single layer temporal attention propagation module multiple times to propagation the information to further nodes.

The temporal attention propagation module makes it possible that not only target nodes and source nodes but also all nodes in the neighborhood can be aware of the information of current interactions. In other words, we extend the small neighborhood only containing source and target nodes to an arbitrarily large one. Besides, this message propagation module is different from the message passing mechanism proposed for static graphs. The message propagation module aims to spread the information to the neighbors of the interaction to update the representations of the influenced nodes while the message passing mechanism is designed for gathering the information from the neighborhood to generation better representations for nodes. However, it is worth noting that the propagation module increases the number of messages that each node receives by a large amount, increasing the space and time complexity of our model.

## 3.4 Training Algorithm

In this section, the training algorithm of link prediction and edge classification using NEDGNN will be described in detail. We will use a decoder to give the final results for these two problems. For interaction $(i, j, e_{ij}, t)$, the probability of edge can be computed:

$$p_{ij} = MLP\left(z_i \| z_j\right) \quad (10)$$

**Algorithm 1:** Training algorithm of link prediction

```
Input: training data, initial model parameters $\mathbb{W}$, model
       hyperparameters
Output: updated model parameters $\mathbb{W}^+$
1  $S = 0, B = \{\}$ //Initialize memory and message boxes;
2  for each batch $(i, j, f, t) \in$ training data do
3      $k \leftarrow$ negative sampling;
4      $\bar{m}_i(t) = Agg(m_i(t_1), m_i(t_4), \ldots)$ //Message
         Aggregation;
5      $s_i(t) = \text{Update}(s_i(t^-), \bar{m}_i(t))$ //Memory Update;
6      $z_i = \text{Gat}\left(s_i(t^-), s_{N_i(t)}(t^-), t\right)$ //Graph Aggregation;
7      Repeat the last 3 steps for nodes $j$ and $k$;
8      $p_{\text{pos}}, p_{\text{neg}} = \text{MLP}(z_i \| z_j), \text{MLP}(z_i \| z_k)$;
9      Backpropagate loss and update model parameters;
10     Clear messages in message boxes for nodes $i$ and $j$;
11     $m_i^0(t), m_j^0(t) = (z_i \| z_j \| e_{ij} \| \phi(t)), (z_j \| z_i \| e_{ij} \| \phi(t))$;
12     Propagate based on $m_i^0(t), m_j^0(t)$ through equations 6~9;
13     Store messages in corresponding nodes' message boxes
         during propagation;
14 end
```

where $z_i$ and $z_j$ can be calculated through equation 4 and 5. The aggregated vector embeddings of node $i$ and node $j$ are concatenated together to represent the edge $(i, j)$, so that the final prediction can be made by an MLP. For the loss function, binary cross-entropy loss (BCE) is used in both link prediction and binary edge classification tasks.

Algorithm 1 shows the pseudocode of link prediction, and the procedure of edge classification will be discussed later.

Because there are no negative samples in the link prediction task, negative samples need to be generated by sampling. Besides, we need to ensure that the information of current interactions will not leak out so that our model can make a fair prediction. As in TGN [17], we reverse the order of computing the message and computing the loss. Message vectors are computed at the end of every batch, which will be used to compute loss in the next batch.

The most significant difference between these two tasks lies in whether to use the information from the current batch. When we predict the links between nodes, we should prevent information leakage. We need to use the information from last batch to make predictions. The order for link prediction task is prediction (based on previous information), then back propagation and storing the information of current batch. But in edge classification task, we can use the information in the current batch to predict the property of the edges. The order for edge classification task is storing information, prediction and back propagation. The details of the edge classification algorithm are shown in the Appendix A.

## 4 EXPERIMENTS

In this section, we design and conduct two graph-based experiments to demonstrate the feasibility and effectiveness of NEDGNN. First, all variants we experiment with are reported in Table 1. We introduce three datasets used in experiments in section 4.1, then present link prediction and edge classification with experimental details and results in section 4.2 and 4.3 respectively. For hyperparameters, we primarily tune those closely realted to the propagation module. Besides, an ablation study to show the effectiveness and time cost of each module will be shown together with hyperparameters in section 4.4.

### 4.1 Datasets

**Reddit post dataset [23].** In the Reddit dataset, users and sub-reddits are two kinds of nodes, which means Reddit dataset is a bipartite graph. If a user sends a post to the sub-reddit, an interaction will occur. The interactions are represented by text features (of a post), and labels represent whether a user is banned.

**Wikipedia edits dataset [23].** This dataset is one month of edits made by edits on Wikipedia pages. It is also a bipartite graph whose nodes are users and pages. An interaction means an edit by a user on a page.

**Transaction dataset.** The transaction dataset contains online credit card transactions spanning from April 2017 to June 2017, which is provided by a financial company in China. Users and merchants are two kinds of nodes. The interactions are represented by transaction features (amount, area, transaction type, ...), and labels represent whether a transaction is a fraud. Due to the privacy policy, this dataset will be private.

The statistics of the three datasets are reported in Appendix B.

### 4.2 Link prediction

In this experiment, the prediction task is: given all interactions till time t, and the node pairs at time $t$, will interactions between these node pairs occur? Reddit post dataset and Wikipedia edits dataset will be used in this experiment.

**Experimental setting.** The first 70% data is used for training, the next 15% is for validation, and the last 15% is for testing. We measure the algorithms' performance in terms of average precision (AP) and area under curve (AUC). Higher values for both are better. In transductive settings, we predict future interactions between nodes that have been observed during training, whereas in the inductive settings we predict future interactions between nodes never appeared before.

**Baselines.** Our baselines for link prediction task are state-of–the-art approaches for CTDGs (CTDNE [6], Jodie [23], DyRep [24], TGAT [29] and TGN [17]) as well as state-of-the-art models for static graphs (GAE [11], DeepWalk [3], Node2Vec [1], GAT [26] and GraphSAGE [14]).

**Experimental results.** The link prediction results on two datasets are shown in Table 2. The results except NEDGNN's are taken directly from the TGN [17] paper. What we can conclude from them are:

- In general, dynamic graph neural network methods outperform static neural graph methods.
- NEDGNN outperforms TGN, DyRep, and other dynamic graph neural network methods, demonstrating the propagation module's effectiveness.
- NEDGNN performs better as the model goes deeper, which means a larger neighborhood is conducive to the dissemination of information and further boosts the performance.

**Table 1: We follow the TGN framework proposed in [17] to describe different deep learning models on CTDGs. Agg. refers to message aggregation function, Mess.Func. refers to message function, Prop. refers to propagation function and Cap. refers to capacity of message box, id represents the identity function and attn (l, n) refers to attention methods having l layers and sampling n neighbors.**

| | Updater | Embedding | Agg. | Mess. Func. | Prop. | Cap. |
|---|---|---|---|---|---|---|
| Jodie | RNN | time | — | id | — | — |
| TGAT | — | attn (2l, 20n) | — | — | — | — |
| DyRep | RNN | id | — | attn | — | — |
| TGN | GRU | attn (1l, 10n) | last | id | — | — |
| TGN-mean | GRU | attn (1l, 10n) | mean | id | — | — |
| NEDGNN-1 | GRU | attn (1l, 10n) | mean | attn (1l, 10n) | attn (1l, 10n) | 20 |
| NEDGNN-2 | GRU | attn (2l, 10n) | mean | attn (2l, 10n) | attn (2l, 10n) | 20 |
| NEDGNN-3 | GRU | attn (3l, 10n) | mean | attn (3l, 10n) | attn (2l, 10n) | 20 |
| NEDGNN-nop | GRU | attn (1l, 10n) | mean | attn (1l, 10n) | — | 20 |
| NEDGNN-max | GRU | attn (1l, 10n) | max | attn (1l, 10n) | attn (1l, 10n) | 20 |
| NEDGNN-last | GRU | attn (1l, 10n) | last | attn (1l, 10n) | attn (1l, 10n) | 20 |
| NEDGNN-20n | GRU | attn (1l, 20n) | mean | attn (1l, 20n) | attn (1l, 20n) | 20 |
| NEDGNN-10c | GRU | attn (1l, 10n) | mean | attn (1l, 10n) | attn (1l, 10n) | 10 |
| NEDGNN-40c | GRU | attn (1l, 10n) | mean | attn (1l, 10n) | attn (1l, 10n) | 40 |
| NEDGNN-3-40c | GRU | attn (3l, 10n) | mean | attn (3l, 10n) | attn (3l, 10n) | 40 |

**Table 2: Average Precision % (AP) for link prediction task in transductive and inductive settings. – does not support inductive setting. Means and standard deviations were computed over 10 runs.**

| | Wikipedia | | Reddit | |
|---|---|---|---|---|
| | Transductive | Inductive | Transductive | Inductive |
| GAE | 91.44±0.1 | – | 93.23±0.3 | – |
| DeepWalk | 91.34±0.6 | – | 92.92±0.5 | – |
| Node2Vec | 91.48±0.3 | – | 84.58±0.5 | – |
| GAT | 94.73±0.2 | 91.27±0.4 | 97.33±0.2 | 95.37±0.3 |
| GraphSAGE | 93.56±0.3 | 91.90±0.3 | 97.65±0.2 | 96.27±0.2 |
| Jodie | 94.62±0.4 | 93.11±0.5 | 97.11±0.3 | 94.36±1.1 |
| TGAT | 95.34±0.1 | 93.99±0.3 | 98.12±0.2 | 96.62±0.3 |
| DyRep | 94.59±0.2 | 92.05±0.3 | 97.98±0.1 | 95.68±0.2 |
| TGN | 98.46±0.1 | 97.81±0.1 | 98.70±0.1 | 97.55±0.1 |
| NEDGNN-1 | 98.79±0.06 | 98.31±0.06 | 98.79±0.06 | 98.03±0.08 |
| NEDGNN-2 | 98.83±0.08 | 98.37±0.11 | 98.87±0.04 | 98.23±0.03 |
| NEDGNN-3 | 98.94±0.04 | 98.48±0.04 | 98.97±0.04 | 98.27±0.04 |

- NEDGNN performs better than existing models by a larger margin in inductive settings, which may imply that our model has a stronger inductive ability.

## 4.3 Edge classification

In this experiment, based on all interactions till time t and the target interaction at time *t*, we will predict the property of the interaction. The Transaction dataset will be used in this experiment.

**Experimental setting.** Since the transaction dataset contains three months of data, we use data of the first two months to train, next half month to validate and last half month to test. We measure the performance of the algorithms in terms of average precision (AP) and F1 score. Higher values for both are better.

**Baselines.** Our baselines for this experiment include some traditional methods for fraud detection (SVM, Random Forest, CNN,

**Table 3: Average Precision (%) (AP) and F1 score for the edge classification task. Means were computed over 10 runs.**

| | Transaction | |
|---|---|---|
| | AP | F1 |
| SVM | — | 72.30 |
| RF | — | 78.28 |
| CNN | — | 78.09 |
| CapsNet | — | 82.41 |
| Jodie | 75.70 | 81.72 |
| TGN | 76.42 | 82.53 |
| NEDGNN-1 | 76.88 | 83.39 |
| NEDGNN-2 | 77.16 | 83.81 |

Capsule Network (CapsNet) [19]) and some state-of-the-art approaches for CTDGs (Jodie [23], TGN [17]).

**Experimental Results.** The edge classification results on two datasets are shown in Table 3. The results of traditional methods are taken directly from the capsule network [27] paper. From the results, we can make the following observations:

- The overall results of dynamic graph methods are better than those of traditional methods, indicating that graph topology and temporal patterns are necessary for detecting frauds.
- NEDGNN-2 outperforms TGN and Jodie, which may imply the information of fraudulent activities in the neighborhood is also useful for fraud detection.

## 4.4 Hyperparameters and Ablation study

In this subsection, we extensively tune some major hyperparameters including the number of layers l, the number of sample neighbors, the choice of aggregation function and leave others the best configuration found in TGN [17]. Detailed settings about other hyperparameters can be found in Appendix C. We also perform a detailed ablation study comparing different variants of our NEDGNN model to test the effectiveness and time complexity of each module. We choose link prediction as the target task and the experiments are conducted on Wikipedia datasets. The details of the NEDGNN variants are reported in Table 4, and the results are shown in Table 4. More detailed complexity evaluation is provided in Appendix D.

From the result, we can make the following observations:

- The propagation module contributes to the improvement. NEDGNN-1 performs better than NEDGNN-nop whose propagation module is absent and the time cost is affordable.
- Appropriate choice of aggregation function has a great influence on the results. The NEDGNN-last performs much worse than NEDGNN-1. It is because that the propagation module increases the number of messages by a considerable amount, but the last aggregation function only uses the latest message in the message box, which makes the propagation module useless.
- NEDGNN-20n performs slightly better than NEDGNN-1, but it is slower than NEDGNN-1. When the model goes deeper, the number of neighbors will have a more significant impact on the time cost. Hence, we suppose that 10 neighbors in each layer are enough for Wikipedia and Reddit datasets.
- The impact of the capacity seems marginal on both the performance and the time cost. It should be noted that the average number of messages each node receive is determined by the number of layers and the batch size. Therefore, when the number of layers is shallow and the batch is small, a small capacity is enough for messages. By comparing the time cost of NEDGNN-3 and NEDGNN-3-40c, we can find that the capacity has an impact on the time efficiency of deep models.
- It can be found that NEDGNN is much slower than TGN, but it's unfair to compare NEDGNN with TGN. The message aggregation function of TGN is last instead of mean function, which greatly affects the time complexity. As mentioned above, last function is unsuitable for NEDGNN, so it is more reasonable to compare TGN-mean with NEDGNN-1, showing that the time complexity of propagation module is acceptable.

**Table 4: Average Precision % and time (seconds per epoch) for ablation study. Means and standard deviations of AP are computed over 10 runs and means and standard deviations of time are computed over 10 epoches.**

| | Trans. | Induc. | Time |
|---|---|---|---|
| TGN | 98.46±0.10 | 97.81±0.10 | 45.75±3.1 |
| TGN-mean | 98.59±0.08 | 98.09±0.11 | 160.40±3.6 |
| NEDGNN-1 | 98.79±0.06 | 98.31±0.06 | 235.52±5.7 |
| NEDGNN-3 | 98.94±0.04 | 98.48±0.04 | 1198.83±38.2 |
| NEDGNN-nop | 98.56±0.12 | 98.04±0.13 | 157.66±6.6 |
| NEDGNN-max | 98.73±0.03 | 98.20±0.05 | 244.62±10.6 |
| NEDGNN-last | 98.58±0.11 | 98.05±0.15 | 77.72±3.7 |
| NEDGNN-20n | 98.80±0.06 | 98.27±0.09 | 277.01±10.6 |
| NEDGNN-10c | 98.76±0.09 | 98.24±0.08 | 232.25±8.3 |
| NEDGNN-40c | 98.78±0.11 | 98.27±0.12 | 244.89±7.1 |
| NEDGNN-3-40c | 98.96±0.07 | 98.50±0.05 | 1351.62±42.3 |

## 5 CONCLUSION AND FUTURE WORK

We introduce NEDGNN, a novel dynamic graph neural network model for learning on continuous-time dynamic graphs. We gain performance improvement on several tasks and datasets comparing with state-of-the-art baselines by proposing a new temporal attention propagation module. Detailed ablation studies show the importance of the propagation module and its related modules to generate meaningful information, as well as the importance of the appropriate message aggregation module. Due to the memory and time limitation, our model can not be very deep. We will leave how to reduce the time and space complexity as future work. Still, we envision that NEDGNN can be further advanced and applied in many interesting fields.

## ACKNOWLEDGMENTS

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB2100801 and in part by the Shanghai Science and Technology Commission Project under Grant 19511101300.

## REFERENCES

[1] Grover Aditya and Leskovec Jure. 2016. Node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. 855–864.

[2] Pareja Aldo, Domeniconi Giacomo, Chen Jie, Ma Tengfei, Suzumura Toyotaro, Kanezashi Hiroki, Kaler Tim, Schardl Tao, and Leiserson Charles. 2020. EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 04 (Apr. 2020), 5363–5370.

[3] Perozzi Bryan, Al-Rfou Rami, and Skiena Steven. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '14)*. 701–710.

[4] Zhuang Chenyi and Ma Qiang. 2018. Dual Graph Convolutional Networks for Graph-Based Semi-Supervised Classification. In *Proceedings of the 2018 World Wide Web Conference (WWW '18)*. 499–508.

[5] Rishab Goel, Seyed Mehran Kazemi, Marcus Brubaker, and Pascal Poupart. 2020. Diachronic Embedding for Temporal Knowledge Graph Completion. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 04 (Apr. 2020), 3988–3995.

[6] Nguyen Giang Hoang, Lee John Boaz, Rossi Ryan A., Ahmed Nesreen K., Koh Eunyee, and Kim Sungchul. 2018. Continuous-Time Dynamic Network Embeddings. In *Companion Proceedings of the The Web Conference 2018 (WWW '18)*. 969–976.

[7] Bruna Joan, Zaremba Wojciech, Szlam Arthur, and LeCun Yann. 2014. Spectral networks and deep locally connected networks on graphs. In *2nd International Conference on Learning Representations ICLR 2014*. 1–14.

[8] Gilmer Justin, Schoenholz Samuel S., Riley Patrick F., Vinyals Oriol, and Dahl George E. 2017. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70 (ICML'17)*. 1263–1272.

[9] S. Kazemi, R. Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, S. Wu, Cathal Smyth, P. Poupart, and Marcus A. Brubaker. 2019. Time2Vec: Learning a Vector Representation of Time. *arXiv preprint arXiv:1907.05321* (2019), 1–16.

[10] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. 2020. Representation Learning for Dynamic Graphs: A Survey. *Journal of Machine Learning Research* 21, 70 (2020), 1–73.

[11] Thomas N. Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. In *NIPS Workshop on Bayesian Deep Learning*. 1–14. arXiv:1611.07308

[12] Thomas N. Kipf and Max Welling. 2017. 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*.

[13] Cho Kyunghyun, van Merriënboer Bart, Gulcehre Caglar, Bahdanau Dzmitry, Bougares Fethi, Schwenk Holger, and Bengio Yoshua. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1724–1734.

[14] Hamilton William L., Ying Rex, and Leskovec Jure. 2017. Inductive Representation Learning on Large Graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*. 1025–1035.

[15] Defferrard Michaël, Bresson Xavier, and Vandergheynst Pierre. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*. 3844–3852.

[16] Ibrahim Nahla Mohamed and Chen Ling. 2015. Link Prediction in Dynamic Social Networks by Integrating Different Types of Information. *Applied Intelligence* 42, 4 (June 2015), 738–750.

[17] E. Rossi, Ben Chamberlain, F. Frasca, D. Eynard, Federico Monti, and M. Bronstein. [n.d.]. Temporal Graph Networks for Deep Learning on Dynamic Graphs. *arXiv preprint arXiv:2006.10637* ([n. d.]), 1–16.

[18] Hisano Ryohei. 2018. Semi-supervised Graph Embedding Approach to Dynamic Link Prediction. In *Complex Networks IX*. 109–121.

[19] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. 2017. Dynamic Routing Between Capsules. In *Advances in Neural Information Processing Systems*, Vol. 30.

[20] Mahdavi Sedigheh, Khoshraftar Shima, and An Aijun. 2018. dynnode2vec: Scalable dynamic network embedding. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 3762–3765.

[21] Hochreiter Sepp and Schmidhuber Jürgen. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (Nov. 1997), 1735–1780.

[22] Cao Shaosheng, Lu Wei, and Xu Qiongkai. 2015. GraRep: Learning Graph Representations with Global Structural Information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM '15)*. 891–900.

[23] Kumar Srijan, Zhang Xikun, and Leskovec Jure. 2019. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery; Data Mining (KDD '19)*. 1269–1278.

[24] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning Representations over Dynamic Graphs. In *International Conference on Learning Representations*. 1–25.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, Vol. 30. 1–22.

[26] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*. 1–16. https://openreview.net/forum?id=rJXMpikCZ

[27] S. Wang, G. Liu, Z. Li, S. Xuan, C. Yan, and C. Jiang. 2018. Credit Card Fraud Detection Using Capsule Network. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 3679–3684.

[28] Yu Wenchao, Cheng Wei, Aggarwal Charu C., Zhang Kai, Chen Haifeng, and Wang Wei. 2018. 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 (KDD '18)*. 2672–2681.

[29] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations*. 1–19.

[30] Ma Yao, Guo Ziyi, Ren Zhaocun, Tang Jiliang, and Yin Dawei. 2020. Streaming Graph Neural Networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20)*. 719–728.

**Algorithm 2:** Training algorithm of edge classification

```
Input: training data, initial model parameters $\mathbb{W}$, model hyperparameters
Output: updated model parameters $\mathbb{W}^+$
1  $S = 0, B = \{\}$ //Initialize memory and message boxes
2  for each batch $(i, j, f, t, label) \in$ training data do
3      $z_i = \mathrm{Gat}\left(s_i(t^-), s_{N_i(t)}(t^-), t\right)$ //Graph Aggregation;
4      $m_i^0(t), m_j^0(t) = \left(z_i \| z_j \| e_{ij} \| \phi(t)\right), \left(z_j \| z_i \| e_{ij} \| \phi(t)\right)$;
5      Propagate based on $m_i^0(t), m_j^0(t)$ through equations 6~9;
6      Store messages in corresponding nodes' message boxes during propagation;
7      $\bar{m}_i(t) = Agg\left(m_i(t_1), m_i(t_4), \ldots\right)$ //Message Aggregation;
8      $s_i(t) = \mathrm{Update}\left(s_i(t^-), \bar{m}_i(t)\right)$ //Memory Update;
9      $z_i = \mathrm{Gat}\left(s_i(t), s_{N_i(t)}(t), t\right)$ //Graph Aggregation;
10     Repeat the last 3 steps for nodes $j$;
11     $p_{\mathrm{pos}}, p_{\mathrm{neg}} = \mathrm{MLP}\left(z_i \| z_j\right), \mathrm{MLP}\left(z_i \| z_k\right)$;
12     $loss = BCE\left(p_{\mathrm{pos}}, label\right)$;
13     Backpropagate loss and update model parameters;
14     Clear messages in message boxes for nodes $i$ and $j$;
```

## A ALGORITHM

Algorithm for edge classification is shown in algorithm 2.

## B DATASETS

The statistics of the three datasets are reported in Table 5.

## C HYPERPARAMETERS

The hyperparameters we used in link prediction and edge classification task is reported in Table 6, which roughly follows the setting in TGN paper.

## D COMPLEXITY EVALUATION

Extensive experiments tell us that the main factor affecting the time complexity of NEDGNN model is the depth of the model, the aggregation function and the size of the dataset. In terms of dataset size, the time complexity is roughly linear to it. The model depth and the aggregation function contributes to the time complexity in a more complicated way and we will leave the theoretical analysis as a future work. From the experimental results, the time complexity is at least exponential to the model depth and the mean aggregation will bring extra complexity when model goes deeper. We also find some tricks that can alleviate the time cost. For example, a shallow embedding module can be used together with a deep propagation module since a deep embedding module contributes little to the performance. Besides, the number of neighbors can be different at each layer. A neighborhood structure like 16-8-2 seems more

**Table 5: Training algorithm of edge classification**

| | Reddit | Wikipedia | Transaction |
|---|---|---|---|
| # Nodes | 9,227 | 11,000 | 96,211 |
| # Interactions | 157,474 | 672,447 | 3500,000 |
| # Feature | 172 | 172 | 48 |
| Time Span | 30 days | 30 days | 3 months |
| Chronological Split | 70%-15%-15% | 70%-15%-15% | 67%-16%-16% |
| # Nodes with dynamic labels | 217 | 366 | - |

**Table 6: Model Hyperparameters**

| | Reddit&Wikipedia | Transaction |
|---|---|---|
| Memory Dimension | 172 | 100 |
| Node Embedding Dimension | 172 | 100 |
| Time Embedding Dimension | 172 | 100 |
| #Attention Heads | 2 | 2 |
| Dropout | 0.1 | 0.2 |
| Batch Size | 200 | 200 |
| Learning Rate | 0.0001 | 0.001 |

reasonable and efficient than a 10-10-10 structure. However, both tricks need further exploration and theoretical analysis.