Streaming Graph Neural Networks

Yao Ma
Michigan State University
mayao4@msu.edu

Ziyi Guo
JD.com
guoziyi@jd.com

Zhaochun Ren
JD.com
renzhaocun@jd.com

Eric Zhao
JD.com
ericzhao@jd.com

Jiliang Tang
Michigan State University
tangjili@msu.edu

Dawei Yin
JD.com
yindawei@acm.org

ABSTRACT

Graphs are essential representations of many real-world data such as social networks. Recent years have witnessed the increasing efforts made to extend the neural network models to graph-structured data. These methods, which are usually known as the graph neural networks, have been applied to advance many graphs related tasks such as reasoning dynamics of the physical system, graph classification, and node classification. Most of the existing graph neural network models have been designed for static graphs, while many real-world graphs are inherently dynamic. For example, social networks are naturally evolving as new users joining and new relations being created. Current graph neural network models cannot utilize the dynamic information in dynamic graphs. However, the dynamic information has been proven to enhance the performance of many graph analytic tasks such as community detection and link prediction. Hence, it is necessary to design dedicated graph neural networks for dynamic graphs. In this paper, we propose DGNN, a new **D**ynamic **G**raph **N**eural **N**etwork model, which can model the dynamic information as the graph evolving. In particular, the proposed framework can keep updating node information by capturing the sequential information of edges (interactions), the time intervals between edges and information propagation coherently. Experimental results on various dynamic graphs demonstrate the effectiveness of the proposed framework.

ACM Reference Format:
Yao Ma, Ziyi Guo, Zhaochun Ren, Eric Zhao, Jiliang Tang, and Dawei Yin. 1997. Streaming Graph Neural Networks. In *Proceedings of ACM Woodstock conference (WOODSTOCK'97)*, Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, Article 4, 11 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

A graph describes a set of objects and their pairwise relations. Many real-life data such as social networks, transportation networks and e-commerce user-item graphs can naturally be represented in the form of graphs. Recent years have witnessed increasing efforts to generalize neural network models to graphs. These neural network models that operate on graphs are known as graph neural networks [12, 28]. Graph neural networks have been applied to perform the reasoning of dynamics of physical systems [1, 7, 27]. Graph convolutional neural networks, which extend the convolutional neural networks to graph structure data, have been shown to improve the performance of graph classification [5, 9] and node-level semi-supervised classification [15, 20]. A general framework of graph neural network is proposed in [2].

Most of these aforementioned neural network models have been designed for static graphs. Graphs in many real-world applications are inherently dynamic. For example, new users will join a social network and users in the social network will create new relations, users in e-commerce platform continue interacting with new items, and new connections are established in a communication network over time. To apply existing graph neural network models to dynamic graphs, we need to completely ignore their evolving structures by treating them as static graphs. However, the dynamic information has been proven to boost a variety of graph analytic tasks such as community detection [24], link prediction [13, 22] and network embedding [13, 22]. Therefore, it has great potential to advance graph neural networks by considering the dynamic nature of graphs, which calls for dedicated efforts.

Meanwhile, designing graph neural networks for dynamic graphs faces tremendous challenges. From the global perspective, structures of dynamic graphs continue evolving since new nodes and edges are constantly introduced. It is necessary to capture the evolving structures for graph neural networks. From the local perspective, a node can keep establishing new edges, the establishing order of these edges is important to understand the node properties. For example, in the e-commerce user-item graph, new interactions are more likely to represent the users' latest preferences. Moreover, the introduction of a new edge (interaction) would affect the properties of the node. It is necessary to keep the node information updated once a new interaction happened. In addition, these edges are unevenly introduced, i.e., the distribution of these edges in the time-line is uneven. For example, a user in social networks could create edges very frequently in certain periods while only establishing a few edges in others. The time intervals between interactions for a specific node can vary dramatically. It is important to consider these time intervals and the major reasons are two-fold. First, the time interval between interactions of specific node can impact our strategy to update the node information. For example, if a new interaction is distant from its previous interaction, we should focus more on the new interaction since the node properties could change. Second, a new interaction can not only affect the two nodes directly involved in the interaction, but also can influence other nodes that are "close" to the two nodes; and the time interval can impact our strategy to propagate the interaction information to the influenced nodes. For example, if the new interaction is distant from the latest interaction between the node and an influenced node, the effect of the new interaction on the influenced node could be little.

In this paper, we embrace the opportunities and challenges to study graph neural networks for dynamic graphs. In essence, we aim to answer the following questions – 1) how to constantly keep the node information updated when new interactions happen; 2) how to propagate the interaction information to the influenced nodes; and 3) how to incorporate time interval between interactions during update and propagation. We propose a dynamic graph neural networks (DGNN) to answer aforementioned three questions simultaneously. Our contributions can be summarized as follows:

- We provide a principled approach for the node information update and propagation when new edges are introduced;
- We propose a novel graph neural network for dynamic graphs (DGNN), which models establishing orders and time intervals of edges into a coherent framework; and
- We demonstrate the effectiveness of the proposed model with several graph related tasks on various real-world dynamic graphs.

The rest of this paper is organized as follows. In Section 2, we introduce the proposed framework with details about its update and propagation components and the approaches to learn model parameters. In Section 3, we present experimental results in two graph mining tasks including link prediction and node classification. We review related work in Section 4 and finally we conclude our work with future work in Section 5.

2 THE PROPOSED FRAMEWORK

In this section, we introduce the graph neural network framework designed for dynamic networks. We first provide an overview about the model and then describe the components of the framework in details. Before that, we first introduce notations and definitions we will use in this work.

A dynamic graph consists of a set of nodes and we assume that there are N nodes $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ introduced until our latest observation about the graph. The graph evolves when new edges and nodes emerge. An example of a dynamic graph is shown in the left side of Figure 1, where there are 8 nodes and 8 interactions (edges) emerge from time t_0 to t_7. Note that, in this work, we only consider the emerging of new edges and nodes while leaving the deletion of existing edges and nodes as one future work. A directed edge e can be represented as (v_s, v_g, t) describing an interaction from v_s to v_g at time t. For example, the interaction happened at t_0 in Figure 1 can be denoted as $\{v_2, v_1, t_0\}$. For convenience, we call the two nodes involved in the interaction as the "interacting nodes". As mentioned before, a new interaction can not only affect the two interacting nodes but also can influence other nodes that are "close" to the interacting nodes, which we call as the "influenced nodes". Thus, we need to update the information of this new interaction to the two interacting nodes and also propagate this information to the "influenced nodes".

To achieve this goal, a dynamic graph neural network (DGNN) is introduced and an overview about DGNN framework is demonstrated in Figure 1, which consists of two major components: 1) the update component and 2) the propagation component. We briefly describe the operations of the two components when introducing a new interaction $\{v_s, v_g, t\}$. The update component involves node v_s, node v_g and updates the interaction information to both of them. For example, in Figure 1, a new interaction $\{v_2, v_5, t_7\}$ happened at t_7, the two interacting nodes being involved in the update component are v_2 and v_5. The propagate component involves the two interacting nodes v_s, v_g and the "influenced nodes" as it propagates the information of the interaction $\{v_s, v_g, t\}$ to the "influenced nodes". The "influenced nodes" can be defined in different ways, which we will discuss in later subsections. In Figure 1, we define the "influenced nodes" as all the nodes that have interacted with the two "interacting nodes", which includes $\{v_1, v_7\}$—the 1-hop "neighors" of v_2, and $\{v_3, v_6\}$— the 1-hop "neighbors" of v_6. Next, we detail each component.

2.1 The update component

In this subsection, we discuss the update component for the interacting nodes. We first give an overview of the operations of the update component with the focus on a single node v_2 of the dynamic graph illustrated in the left of Figure 1). There are three interactions involving node v_2, $\{v_2, v_1, t_0\}$, $\{v_7, v_2, t_3\}$ and $\{v_2, v_5, t_7\}$. It is natural that interactions between nodes will affect the properties of the nodes. For example, as suggested by homophily, users with similar interests are likely to create connections in social networks [26]. Thus, the update components should update the interaction information to the two interacting nodes. As shown in Figure 2, there are three update components, processing the three interactions involving node v_2 for node v_2. Each of the update components takes an interaction as input and update the interaction information to node v_2. Note that we only show the update component for node v_2 in Figure 2, while there is also another update component to update the interaction information to the other interacting node for each interaction. Furthermore, the order of the interactions is also important to understand the nodes' property. For example, in the e-commerce user-item graph, user's latest preference can be better captured by the recent interactions than the old ones. Thus, it is important to capture the order information. It is natural to view the interactions (involving the same node) as a "sequence" and recurrently apply the update component to the interactions. Note that, although the interactions can be viewed as a "sequence", we do not need to store all the information of this "sequence". We only store most recent information of the nodes. As shown in Figure 2, the three update components are connected in the sense that the next component takes the output of the previous component as input. Hence, we model the update component based on the long-short term memory (LSTM) unit [17]. As discussed before, the time interval information is also important, thus, we also incorporate it into the update component. As shown in Figure 2, a single update component consists of three units – the interact unit, the update unit and the merge unit. Next we describe these three units in details.

Before proceeding to the details of the units, we first introduce the information we store for each node v. Note that in a directed graph, a node could play the roles of both source node and target node. Thus, we introduce two sets of different cell memories and

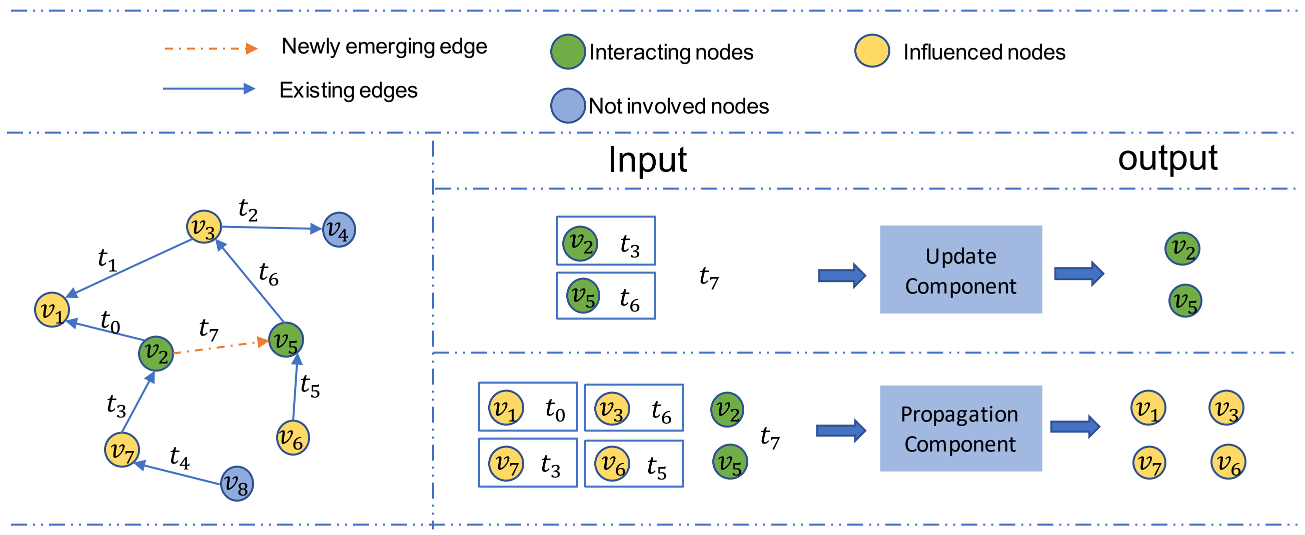


Figure 1: An overview of DGNN when a new interaction happened at time t_7 from v_2 to v_5. The two interacting nodes are v_2 and v_5. The nodes $\{v_1, v_3, v_6, v_7\}$ are assumed to be the influenced nodes.

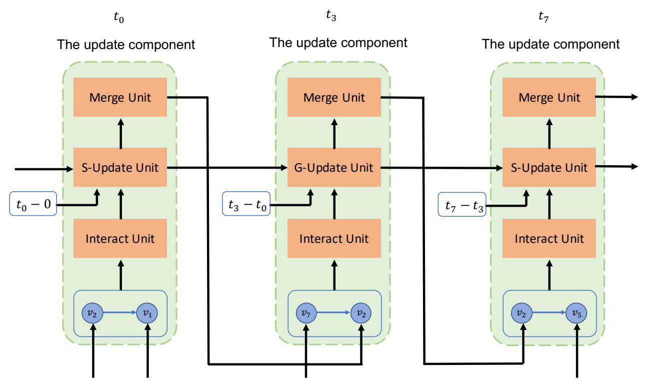


Figure 2: The operations of update components with the focus on node v_2 and all its interactions

hidden states for the two roles of each node, respectively. The cell memory and hidden state for the source role of node v right before time t are denoted as $C_c^s(t-)$ and $h_v^s(t-)$ respectively, while the cell memory and hidden state for the target role of node v right before time t are denoted as $C_c^g(t-)$ and $h_v^g(t-)$ separately. Here, the notation $t-$ means the time that is infinitely close to t, but prior to t, such that all the interactions before time t have been processed. For example, in Figure 2, at time t_7, for node v_2, $C_c^s(t_7-)$ is, in fact, equal to $C_c^s(t_3)$. Note that we do not consider the propagation component now, for the purpose of illustrating the update component. The source and target hidden states are merged with the merge unit, to generate the general features $u_v(t-)$ of the node v, which describes the general property of node v. These cell memories $C_c^s(t-)$, $C_c^g(t-)$, hidden states $h_v^s(t-)$, $h_v^g(t-)$ and general features $u_v(t-)$ are the information stored for each node v and needed to be updated when new interaction happens. For example, Figure 3 shows the operations of two update components performing update for node v_2 and v_5 when the interaction $\{v_2, v_5, t_7\}$ happens. The information stored for the two nodes right before time t_7 is shown in Figure 2 (a).

2.1.1 *The interact unit.*

The interact unit is designed to generate the interaction information for $\{v_s, v_g, t\}$ from node information. The generated interaction information is later used as the input of the update unit. We model the interact unit using a deep feed-forward neural network and the formulation is as follows:

$$e(t) = act(W_1 \cdot u_{v_s}(t-) + W_2 \cdot u_{v_g}(t-) + b_e) \quad (1)$$

where $u_{v_s}(t-)$ and $u_{v_g}(t-)$ are the general features of the nodes v_s and v_g right before time t. W_1, W_2 and b_e are the parameters of the neural network and $act(\cdot)$ is an activation function such as sigmoid or tanh. The output $e(t)$ contains the information of the interaction $\{v_s, v_g, t\}$. As an example, Figure 3 (b) shows how the interact unit works for the interaction $\{v_2, v_7, t_7\}$.

2.1.2 *The update unit.*

As mentioned before, the interactions (involving the same node) can be viewed as a "sequence". The information of this node gradually evolves as these interactions happens sequentially. Thus, to capture these interaction information for this node, we recurrently apply the update component to process the interaction information. The update unit is the part performing the operation to update the interaction information generated from the interact unit to the interacting nodes. Recall that the interactions do not emerge evenly in time. The time interval between interactions involving the same node can vary dramatically. The time interval impacts how the old information should be forgotten. It is intuitive that interactions happened in the far past should have less influence on the current information of node, thus they should be "heavily" forgotten. On the other hand, recent interactions should have more importance on the current information of node. Thus, it is desired to incorporate the time interval into the update component. Hence, to build the update unit, we modify the LSTM unit as similar in [3] to

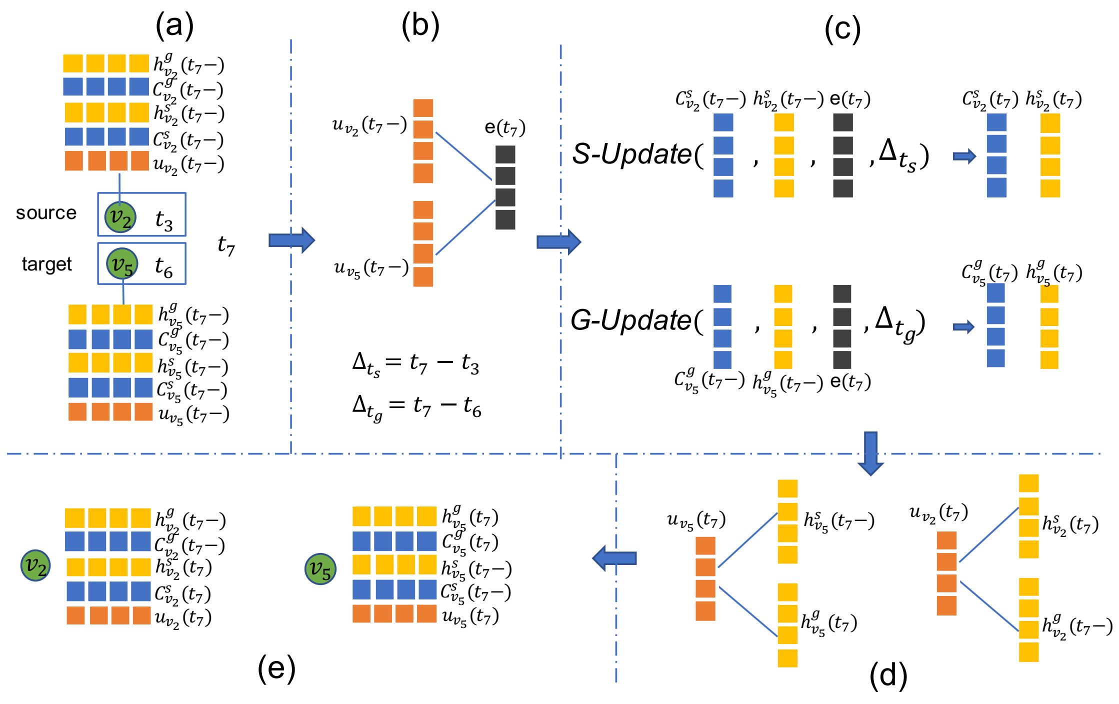


Figure 3: An example to illustrate an overview about the update components when an interaction v_2, v_5, t_7 happened.

incorporate the time interval information to control the "magtitude" of forgetting.

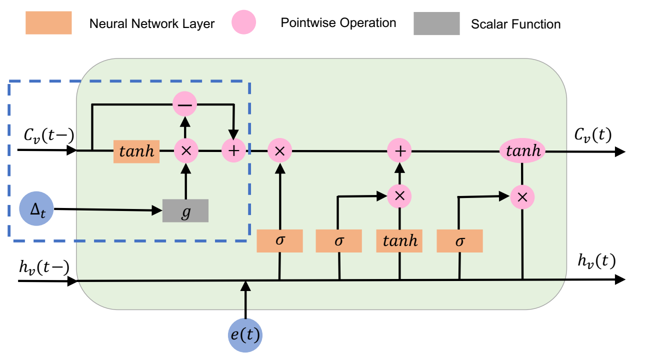


Figure 4: An illustration of the update unit

An update unit is shown in Figure 4, the input of this unit includes the most recent cell memory $C_v(t-)$, hidden states $h_v(t-)$, the time interval Δ_t and the interaction information $e(t)$ calculated by the interact unit. The output of the update unit are the updated cell memory $C_v(t)$ and hidden state $h_v(t)$. Note that, for illustration purpose, we do not differentiate the source and target cell memory and hidden states in Figure 4. In practice, we have two types of update units, the S-Update unit and the G-Update unit, which share the same structure but have different parameters. For an interaction $\{v_s, v_g, t\}$, we use the S-Update unit to update the information for the source node v_s and use the G-Update unit to update the information for the target node v_g. The update unit is based on an LSTM unit, the only difference between the update unit and a standard LSTM unit is in the blue dashed box part of Figure 4. The corresponding formulations for this part are as follows

$$C_v^I(t-1) = tanh(W_d \cdot C_v(t-1) + b_d) \tag{2}$$

$$\hat{C}_v^I(t-1) = C_v^I(t-1) * g(\Delta_t) \tag{3}$$

$$C_v^T(t-1) = C_v(t-1) - C_v^I(t-1) \tag{4}$$

$$C_v^*(t-1) = C_v^T(t-1) + \hat{C}_v^I(t-1) \tag{5}$$

In this part, the old cell memory $C_v(t-1)$ is adjusted according to the time interval to generate the adjusted old cell memory $C_v^*(t-1)$. It is first decomposed to two components, the short term memory $C_v^I(t-1)$ and the long term memory $C_v^T(t-1)$, where $C_v^I(t-1)$ is generated by a neural network and the long term memory $C_v^T(t-1) = C_v(t-1) - C_v^I(t-1)$. The long term memory is kept untouched while the short term memory is discounted (forgotten) according to the time interval Δ_t between the events with a discount function g. The discount function g is a decreasing function, which means the larger the time interval is, the less the short term memory is kept. Hence, we use this to model how we should forget the old information in our model. The discounted short term memory $\hat{C}_v^I(t-1)$ and the long term memory are then combined to generate the adjusted old cell memory $C_v^*(t-1) = \hat{C}_v^I(t-1) + C_v^T(t-1)$, which can be regarded as the output of the dashed box being input to the standard LSTM unit (the rest part of the update unit). The decomposition and recombination ensure that not the entire information of the old cell memory is lost during this procedure.

The formulations of the rest part of the update unit, which are the same as a standard LSTM unit, are as follows

$$f_t = \sigma(W_f \cdot e(t) + U_f \cdot h_v(t-1) + b_f) \tag{6}$$
$$i_t = \sigma(W_i \cdot e(t) + U_i \cdot h_v(t-1) + b_i) \tag{7}$$
$$o_t = \sigma(W_o \cdot e(t) + U_o \cdot h_v(t-1) + b_o) \tag{8}$$
$$\tilde{C}_v(t) = tanh(W_c \cdot e(t) + U_c \cdot h_v(t-1) + b_c) \tag{9}$$
$$C_v(t) = f_t * C_v^*(t-1) + i_t * \tilde{C}_v(t) \tag{10}$$
$$h_v(t) = o_t * tanh(C_v(t)). \tag{11}$$

For convenience, we summarize the procedure of the update unit in Figure 4 (eq. (2) to eq. (11)) as

$$C_v(t), h_v(t) = \text{Update}(C_v(t-1), h_v(t-1), \Delta_t, e(t)) \tag{12}$$

Examples of the operations of the update units are shown in Figure 3 (c), where, for interaction $\{v_2, v_7, t\}$, we use the S-Update unit to update the information for the source node v_2 and use the G-Update unit to update the information for the target node v_7. Note that the S-Update unit only updates the source information (cell memory and hidden state) of the source node but keeps the target information of the source node untouched. Similarly, the G-Update unit only updates the target information of the target node but keeps the source information untouched. In Figure 2, for the convenience of illustration, we "abuse" the output of the S-Update unit and G-Update unit a little bit by considering the untouched target information of source node as part of the output of the S-Update unit and the untouched source information of target node as part of the output of the G-Update unit.

2.1.3 The merge unit. The merge unit is to combine the source hidden state and target hidden state of a given node to generate the general features for this node. As we mentioned in last subsection, given an interaction $\{v_s, v_g, t\}$, the S-Update unit only updates the source information of the source node v_s and the G-update only updates the target information of the target node v_g. Hence, for node v_s, we have $h^s_{v_s}(t)$ and $h^g_{v_s}(t-)$ as the output of the S-Update unit. The merge unit takes these two hidden states as input and generates new general features $u_{v_s}(t)$ for the node v_s as follows:

$$u_{v_s}(t) = W^s \cdot h^s_{v_s}(t) + W^g \cdot h^g_{v_s}(t-) + b_u \tag{13}$$

Similarly, the merge unit generates the new general features $u_{v_g}(t)$ for node v_g as follows:

$$u_{v_g}(t) = W^s \cdot h^s_{v_g}(t-) + W^g \cdot h^g_{v_g}(t) + b_u \tag{14}$$

The two merge units to generate new general features for node v_2 and v_5 after the interaction $\{v_2, v_5, t_7\}$ are shown in Figure 3 (d).

Finally, the output of the update component is the updated information of the interacting nodes. For the source node v_s of the interaction $\{v_s, v_g, t\}$, the updated information includes $C^s_{v_s}(t)$, $h^s_{v_s}(t), C^g_{v_s}(t-), h^g_{v_s}(t-)$ and $u_{v_s}(t)$. For the target node v_g, the updated information includes $C^s_{v_g}(t-), h^s_{v_g}(t-), C^g_{v_g}(t), h^g_{v_g}(t)$ and $u_{v_g}(t)$. The operations of the two update components for the interaction $\{v_2, v_5, t_7\}$ are shown in Figure 3.

2.2 The propagation component

In the previous section, we introduced the component to update the two interacting nodes when a new interaction happens. The update component only considers the two nodes directly affected by the new interaction. However, the newly emerging interaction changes the existing local structure of the graph. Thus, the interaction can influence some other nodes. In this work, we choose the current neighbors of the two "interacting nodes" as the "influenced nodes". The major reasons are three-fold. First, as informed in mining streaming graphs, the impact of a new edge on the whole graph is often local [8]. Second, after we propagate information to the neighbors, the information will be further propagated, once the influenced nodes have interactions with other nodes. Third, we empirically found that when propagating more hops, the performance does not increase significantly or even decreases since we may also introduce noise during the propagation. To update the influenced nodes, the interaction information should be propagated to their cell memories. As the interaction does not directly influence the influenced nodes, we assume that the interaction does not disturb the history of the influenced nodes but only bring about new information. Thus, we do not need to decay or decrease the history information (cell memory) as what we do in the update component but only incrementally add new information to it. As similar with the intuition that older interactions should have less impact on the recent node information, an interaction should have less impact on the older influenced nodes. Thus, it is also desired to consider the time interval of the interactions in the propagation component. In addition, the influence can vary due to varied tie strengths (e.g., strong and weak ties are mixed together) [33]. Nodes are likely to influence others with strong ties than weak ties. Therefore, it is important to consider heterogeneous influence. With these intuitions, next we illustrate the operations of the propagation component.

The propagation component consists of three units – the interact unit, the prop unit and the merge unit. Note that the interact unit and the merge unit are the same as the ones in the update component. So, we mainly introduce the prop unit.

Let $\{v_s, v_g, t\}$ be the newly happened interaction, where v_s is the source node and v_g is the target node. The influenced nodes are the neighbors of these two nodes until time t, which can be denoted as $N(v_s)$ and $N(v_g)$. In a directed graph, we can further decompose the two sets of neighbors as $N(v_s) = N^s(v_s) \cup N^g(v_s)$ and $N(v_g) = N^s(v_g) \cup N^g(v_g)$, where $N^s(\cdot)$ denotes the set of source neighbors and $N^g(\cdot)$ denotes the set of target neighbors. Note that there are, in total, 4 types of different prop units with the same structure but different parameters. They are the prop unit to propagate interaction information 1) from the source node v_s to its source neighbors $N^s(v_s)$; 2) from the source node v_s to its target neighbors $N^g(v_s)$; 3) from the target node v_g to its source neighbors $N^s(v_g)$; and 4) from the target node v_g to its target neighbors $N^g(v_g)$. We only describe one of them, from source node to its source neighbors, as the others have the same structure. For each node $v_x \in N^s(v_s)$, we propagate the interaction information to them with the following formulations:

$$C^s_{v_x}(t) = C^s_{v_x}(t-) + f_a(u_{v_x}(t-), u_{v_s}(t-)) \cdot g(\Delta^s_t) \cdot h(\Delta^s_t) \cdot \hat{W}^s_s \cdot e(t) \tag{15}$$
$$h^s_{v_x}(t) = \tanh(C^s_{v_x}(t)) \tag{16}$$

where $\Delta^s_t = t - t_x$ is the time interval between the current time t and the last time t_x when the node v_x interacted with node v_s. $g(\Delta^s_t)$ is the same decay function as we defined for the update component.

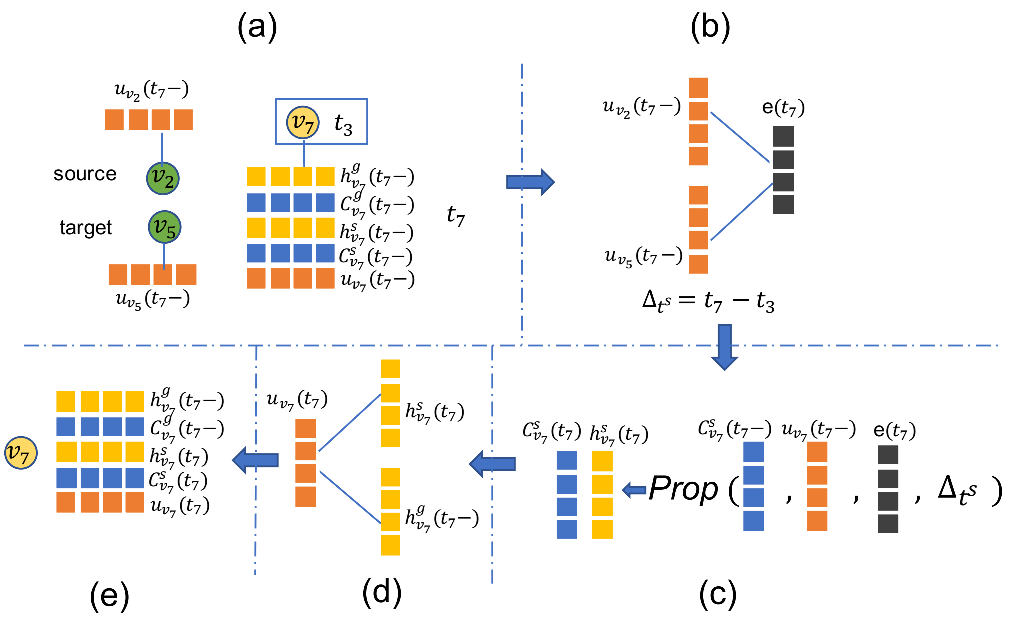


Figure 5: The propagation to the source neighbor v_7 of the source node v_2 when a new interaction $\{v_2, v_5, t_7\}$ happened.

Intuitively, propagating the interaction information to "extremely old neighbors" may introduce noise. Hence, we introduce a function $h(\Delta^s_t)$ to filter some "influenced nodes" as defined as follows:

$$h(\Delta^s_t) = \begin{cases} 1, & \Delta^s_t \leq \tau, \\ 0, & \text{otherwise.} \end{cases}$$

where τ is a predefined threshold. This means if the time interval is too large ($> \tau$), we will stop propagating information to such neighbors. *One advantage of this operation is to make the propagation step much more efficient.* We will demonstrate more details about this filtering step in the experiment section. $\hat{W}^s_s$ is a linear transformation to project the interaction information for propagating to source neighbors. We have different transformation matrix for the other types of prop units. The function $f_a(u_{v_x}(t_x-), u_{v_s}(t-))$ is an attention function to capture the tie strength between nodes v_x and v_s defined as:

$$f_a(u_{v_x}(t-), u_{v_s}(t-)) = \frac{\exp(u_{v_x}(t-)^T u_{v_s}(t-))}{\sum\limits_{v \in N^s(v_s)} \exp(u_v(t-)^T u_{v_s}(t-))} \tag{17}$$

Figure 5 illustrates an example of propagating the interaction information to the source neighbor v_7 of the source node v_2 when an interaction $\{v_2, v_5, t_7\}$ happens. The prop unit is shown in Figure 5 (c). Note that, for compactness of Figure 5, we do not include the attention mechanism in it. The interact unit is shown in Figure 5 (b), and the merge unit is shown in Figure 5 (d).

2.3 Parameter learning

In this section, we introduce the parameter learning procedure of the dynamic graph neural network model. The proposed framework DGNN is general and can be utilized for a variety of network analytic tasks. Next we will use link prediction and node classification as examples to illustrate how to use DGNN for network analysis and its corresponding algorithm for parameter learning.

2.3.1 Parameter learning for link prediction. To train the dynamic graph neural network model for the link prediction task, we design a specific training schedule. In DGNN, we only have one set of general features for each node, while each node can be either source node or target node. Thus, for the link prediction task, we first project the general features of the two interacting nodes to the corresponding role in the interaction with two projection matrix P^s and P^g. We then adapt a widely used graph-based loss function with temporal information. For an interaction (v_s, v_g, t), we project the most recent general features $u_{v_s}(t-)$, $u_{v_g}(t-)$ to $u^s_{v_s}(t-)$ and $u^g_{v_g}(t-)$ respectively as follows:

$$u^s_{v_s}(t-) = P^s \cdot u_{v_s}(t-)$$
$$u^g_{v_g}(t-) = P^g \cdot u_{v_g}(t-)$$

Then the probability of an interaction from v_s to v_g is modeled as $\sigma(u^s_{v_s}(t-)^T u^g_{v_g}(t-))$ where $\sigma(\cdot)$ is the sigmod function. Eventually the loss can be represented as

$$J((v_s, v_g, t)) = -\,log(\sigma(u^s_{v_s}(t-)^T u^g_{v_g}(t-))) \\ - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} log(\sigma(u^s_{v_s}(t-)^T u^g_{v_n}(t-))) \tag{18}$$

where Q is the number of negative samples and $P_n(v)$ is a negative sampling distribution. The total loss until time T can be represented as

$$\sum_{e \in \mathcal{E}(T)} J(e); \tag{19}$$

where $\mathcal{E}(T)$ denotes all the interactions until time T.

We then adopt a mini-batch gradient descent method to optimize the loss function. Note that in our case, the mini-batches of edges are not randomly sampled from the entire set of edges but sequences from the interaction sequence maintaining the temporal order. The loss of the mini-batch is calculated from all the interactions in the mini-batch. The negative sampling distribution $P_n(v)$ is a uniform distribution out of all the nodes involved in the mini-batch, which includes the interacting nodes and the influenced nodes of each interaction.

2.3.2 Learning parameters for node classification. To train the dynamic graph neural network model for node classification, we adopt the cross entropy loss. For a node v with general features $u_v(t)$ and label $y \in \{0,1\}^{N_c}$ immediately after time t, where N_c is the number of classes, we first project $u_v(t)$ to $u_v^c(t) \in \mathcal{R}^{N_c \times 1}$. Then, the loss corresponding for the node v at time t is defined as

$$J(v,t) = -\sum_{i=0}^{N_c-1} y[i] \log \left(\frac{\exp(u_v^c(t))[i]}{\sum_{j=0}^{N_c-1} \exp(u_v^c(t))[j]} \right);$$

where $y[i]$ and $u_v^c(t)[i]$ denote the i-th element of y and $u_v^c(t)$, respectively.

The training schedule is semi-supervised, only some nodes are labeled but the unlabeled nodes are also involved in the update and propagation component of the dynamic graph neural network model. We adopt a similar mini-batch (of edges) procedure as that in link prediction. Let T_m be the end time of the mini-batch, i.e. the time of the last interaction in the mini-batch. After the mini-batch of interactions is processed by the update and propagation components of DGNN, we collect all the nodes involved in the mini-batch and denote them as $\mathcal{V}_m$. Let $\mathcal{V}_{train}$ denote the set of all the nodes with labels. We then use $\mathcal{V}_{m-train} = \mathcal{V}_m \cap \mathcal{V}_{train}$ as the training samples of this mini-batch. We form the loss function for this mini-batch as

$$\sum_{v \in \mathcal{V}_{m-train}} J(v, T_m). \tag{20}$$

3 EXPERIMENTS

In this section, we perform two graph based tasks to demonstrate the effectiveness of the proposed dynamic graph network model. We first introduce three datasets we use in the experiments, then present the two tasks—link prediction and node classification—with experimental details and discussions and finally study the key model components of the proposed framework.

3.1 Datasets

We conduct the experiments on the following three datasets. Some important statics of the three datasets can be found in Table 1.

Table 1: Statistics of datasets

	UCI	DNC	Epinions
number of nodes	1,899	2,029	6,224
number of edges	59,835	39,264	19496
time duration	194 days	982 days	936 days
number of labels	\	\	15

- **UCI** [21] is a directed graph which denotes the message communications between the users of an online community of students from the University of California, Irvine. A node in this graph represents a user. There are edges between users if they have message communications where the time associated with each edge indicates when they communicated. In this dataset, the graph structure and edge creation time are available; hence we use this dataset to evaluate link prediction performance.
- **DNC** [21] is a directed graph of email communications in the 2016 Democratic National Committee email leak. Nodes in this graph represents persons. A directed edge in this graph represents that an email is sent from one person to another. In this dataset, the graph structure and the time information when edges are established are available; thus we use this dataset for the link prediction task.
- **Epinions** [29] is a directed graph which denotes trust relations between users in the product review platform Epinions. A node in this graph represents a user. A directed edge represents a trust relation among users. We have 15 labels in this dataset. The label of each user is assigned according to the category of the majority of the user's reviewed products. In this dataset, we have graph structure, edge creation time and node labels; therefore, we use this dataset for both link prediction and node classification tasks.

3.2 Link prediction

In this section, we conduct the link prediction experiments to evaluate the performance of the DGNN model. We first introduce the baselines. We then describe the experimental setting of the link prediction task and the evaluation metrics. Finally, we present the experimental results with discussions.

3.2.1 Baselines. We carefully choose representative baselines from two groups. One group includes existing neural graph network models. The other contains state of art graph representation learning methods given their promising performance in link prediction. Details about baselines are introduced as follows:

- **GCN** [20] is a state of art graph convolutional network model, it tries to learn better node features by aggregating information from the node's neighbors. The method cannot use temporal information; thus we treat the dynamic graphs as static graphs for this method by ignoring the edge creation time information.
- **GraphSage** [15] also aggregates information from neighbors, but it samples neighbors instead of using all neighbors. It cannot use temporal information neither; thus we treat the dynamic graphs as static graphs for this method similar to **GCN**.
- **node2vec** [14] is a state of art graph representation learning method. It utilizes random walk to capture the proximity in the network and maps all the nodes into a low-dimensional

representation space which preserves the proximity. It cannot utilize the temporal information and we convert the dynamic graphs into static graphs for node2vec.

- **DynGEM** [13] is a graph representation learning method designed for dynamic graphs. However, it can only be applied to discrete time data with snapshots, thus in our experiments, we split each dataset into snapshots for this baseline.
- **CPTM** [10] is a tensor-based model. It treats the dynamic graph as 3-dimension tensor, where two dimensions describe the interactions of nodes and the third dimension is time. It decomposes the tensor to get the features of the nodes. It can only be applied to discrete time data with snapshots. Hence, as for DyGEM, we split each dataset into snapshots.
- **DANE** [23] is a recent proposed eigendocompoation based node representation learning algorithm for attributed dynamic graphs. It updates the node representations over time by perturbation analysis of eigenvectors. It can only be applied to discrete time data with snapshots. Hence, as for DyGEM, we split each dataset into snapshots. Note that since the focus in this work is not attributed networks, we use a variant of **DANE**, which only considers the structural information of dynamic networks.
- **DynamicTriad** [36] is a recent proposed node representation learning algorithm for dynamic graphs. As suggested by its name, it is based on modelling the triangle closure between snapshots of the dynamic graphs. It can only be applied to discrete time data with snapshots. Hence, as for DyGEM, we split each dataset into snapshots.

As we can see, our baselines include representative graph neural network models, i.e., **GCN** and **GraphSage**, one state of the art static node embedding method **node2vec**, three recent dynamic network embedding methods **DynGEM**, **DANE**, **DynamicTriad** and one traditional dynamic network embedding method **CPTM**.

3.2.2 Experimental setting. In the link prediction task, we are given a fraction of interactions in the graph as the history and supposed to predict which new edges will emerge in the future. In this experiment, we use the first 80% of the edges as the history (training set) to train the dynamic graph neural network model, 10% of the edges as the validation set and the next 10% edges as the testing set. All the baselines and our model return node features after training. We use the node features learned with the 80% training set as the node features for link prediction. For each edge (v_s, v_g, t) in the testing set, we first fix v_s and replace v_g with all nodes in the graph and then we use the cosine similarity to measure the similarity and rank the nodes. We then fix v_g and replace v_s with all the nodes in the graph and rank the nodes in a similar way. For all the models we tune the parameters on the validation set. For our model, to calculate the cosine similarity, we use the projected features for UCI and DNC dataset and use the original features for Epinions dataset according to the performance on the validation dataset. In the link prediction task, we randomly initialize the cell memories, hidden states and general features for all nodes.

3.2.3 Evaluation metrics. We use two different metrics to evaluate the performance of the link prediction task. One of them is mean reciprocal rank (**MRR**) [32], which is defined as

$$\mathbf{MRR} = \frac{1}{|H|}\sum_{i=1}^{H}\frac{1}{rank_i} \tag{21}$$

where H is the number of testing pairs. Note that one edge is corresponding to two testing pairs: one for the source node and the other one for the target node. $rank_i$ is the rank of the ground truth node out of all the nodes. The **MRR** metric calculates the mean of the reciprocal ranking of the ground truth nodes in the testing set. It is higher when there are more ground truth node ranked top out of all the nodes.

The other metric we use is **Recall@k**, which is defined as:

$$\mathbf{Recall@k} = \frac{1}{|H|}\sum_{i}^{|H|}\mathbf{1}\{rank_i \leq k\} \tag{22}$$

where $\mathbf{1}\{rank_i \leq k\} = 1$ only when $rank_i \leq k$, otherwise 0. The **recall@k** calculates how many of the ground truth nodes are ranked in top k out of all the nodes in their own testing pairs. The larger it is, the better the performance is. In this work, we use **Recall@20** and **Recall@50**.

3.2.4 Experimental results. In this section, we present the experimental results. The link prediction results on the three datasets are shown in Table 2. From results, we can make the following observations

- DANE does not perform well as expected since it has been originally designed for attributed networks.
- DynGEM and DynamicTriad outperforms node2vec in most cases. All the three methods are embedding algorithms – node2vec is for static networks while DynGEM and DynamicTriad capture dynamics. These results suggest the importance of the dynamic information in graphs.
- The proposed dynamic graph neural network model outperforms two representative existing GNNs, i.e., GCN and GraphSage. Our model is for dynamic networks while GCN and GraphSage ignore the dynamic information, which further support the importance to capture dynamics.
- The proposed model DGNN outperforms all the baselines in most of the cases on all the three datasets. DGNN provides model components to capture time interval, propagation and tie strength. In the following subsections, we will study the impact of these model components on the performance of the proposed framework.

3.3 Node classification

In this subsection, we conduct the node classification task to evaluate the performance of the dynamic graph neural network model. We first introduce the baselines. We then describe the experimental setting and the evaluation metrics. Finally, we present the experimental results.

3.3.1 Baselines. The node classification task is a semi-supervised learning task, where some nodes are labeled and we aim to infer the labels of unlabeled nodes in the graph. Therefore, we carefully choose two groups of baselines. One is about GNNs for semi-supervised learning including GCN and GraphSage. The other is

Table 2: Performance comparison of link prediction.

Baselines	UCI			DNC			Epinions		
	MRR	Recall@20	Recall@50	MRR	Recall@20	Recall@50	MRR	Recall@20	Recall@50
DGNN	**0.0342**	**0.1284**	**0.2547**	**0.0536**	0.1852	**0.3884**	**0.0204**	**0.0848**	**0.1894**
GCN	0.0138	0.0632	0.1176	0.0447	**0.2032**	0.3291	0.0045	0.0071	0.0119
GraphSage	0.0060	0.0161	0.0578	0.0167	0.0576	0.1781	0.0035	0.0072	0.0108
node2vec	0.0056	0.0184	0.0309	0.0202	0.0719	0.178	0.0135	0.0571	0.1240
DynGEM	0.0146	0.0773	0.1455	0.0271	0.0971	0.2356	0.0150	0.0657	0.1233
CPTM	0.0138	0.0921	0.1082	0.0109	0.0072	0.0108	0.0036	0.0060	0.0125
DANE	0.0040	0.0110	0.0233	0.0128	0.0270	0.0432	0.0040	0.0100	0.0120
DynamicTriad	0.0150	0.0610	0.1236	0.0146	0.0414	0.0665	0.0170	0.0729	0.1629

traditional semi-supervised learning methods and we choose a start-of-the-art traditional semi-supervised method LP based on Label Propagation [37]. Note that for a fair comparison, we do not choose node embedding algorithms such as node2vec as baselines since they are designed under the unsupervised setting.

3.3.2 Experimental setting. In the node classification task, we randomly sample a fraction of nodes and hide their labels. These nodes with labels hidden will be treated as validation and testing sets. The remaining nodes are treated as the training set. In this work, we randomly sample 20% of all the nodes and hide their labels. We use 10% of them as validation set and the other 10% as testing set. For the rest 80% of nodes with labels, we choose $x\%$ as labeled nodes and others as unlabeled nodes. In this experiment, we vary x as $\{100, 80, 60\}$. We use F_1-micro and F_1-macro as the metrics to measure the performance of the node classification task.

3.3.3 Experimental results. Among three datasets, only Epinions dataset has label information. Hence we conduct the node classification task on it and the results are presented in Figure 6. We can make the following observations:

- With the increase of the number of labeled nodes, the classification performance tends to increase.
- GraphSage, GCN and DGNN outperforms LP in all settings, which indicates the power of GNNs in semi-supervised learning.
- DGNN outperforms GraphSage and GCN under all the three settings, which shows the importance of temporal information in node classification.

3.4 Model Component analysis

In the last two sections, we have demonstrated the effectiveness of the proposed framework in two graph mining tasks – link prediction and node classification. In this subsection, we conduct experiments to understand the effect of the key components on our proposed model. More specifically, we form the following variants of our model by removing some components in the model:

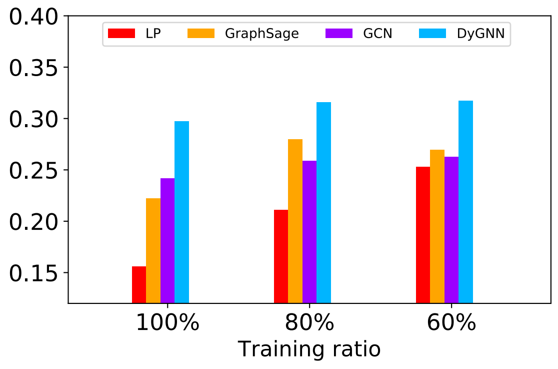
- **DGNN-prop:** In this variant, we remove the entire propagation component from the model. This variant only does the update procedure when new edge emerges.
- **DGNN-ti:** In this variant, we do not use the time interval information in both update component and propagation component. Thus, we treat the interactions as a sequence with no temporal information.
- **DGNN-att:** In this variant, we remove the attention mechanism in the propagation component and consider equal influence.

We will use the task of link prediction to illustrate the impact of model components. The performance of these variants on link prediction task are shown in Table 3. As we can observe from the results, all the three components are important to our model, as removing them will reduce the performance of link prediction. Via this study, we can conclude that (1) it is necessary to propagate interaction information to influenced nodes; (2) it is important to consider the time interval information; and (3) capturing varied influence can improve the performance.

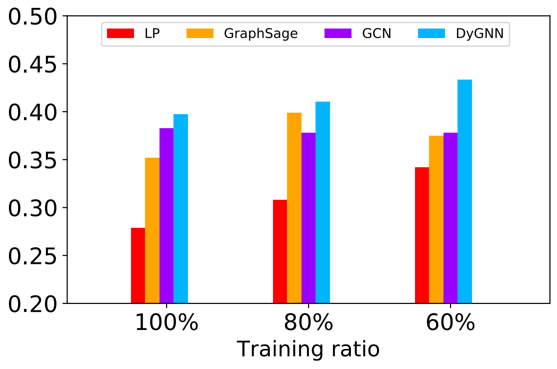
3.5 Parameter Analysis

The proposed framework introduces one parameter τ in the propagation component to filter some "influenced nodes". In this subsection, we analyze how different values of τ in the propagation component affect the performance of the DGNN model. We perform the analysis for the link prediction task on the UCI dataset with the *MRR* measure since we have similar observations with other settings and on other datasets.

As shown in Table 1, the duration of this dataset is 194 days and we set the threshold τ to 1, 7 days, and $10 - 100$ days with a step size of 10. The performance in terms of *MRR* is shown in Figure 7. The performance of DGNN first increases as the threshold τ gets larger. A large τ allows the interaction information to be propagated to more influenced nodes. After τ hits 50, the performance becomes stable or even slightly decreases. These observations suggest that 1) the propagation procedure does help to broadcast necessary information to the "influenced nodes" as the performance first gets improved when the threshold increases; and 2) propagating the interaction information to "extremely old neighbors" may not be helpful or even may bring noise. These observations have practical significance since we can choose a proper τ in the propagation component, which can remarkably boost the efficiency of the proposed framework since we only need to perform the propagation with a small number of "influenced nodes".



(a) Epinions: F_1-macro



(b) Epinions: F_1-micro

Figure 6: Performance Comparison of Node classification on Epinions dataset

Table 3: Comparison of variants on the link prediction task.

Baselines	UCI			DNC			Epinions		
	MRR	Recall@20	Recall@50	MRR	Recall@20	Recall@50	MRR	Recall@20	Recall@50
DGNN	0.0342	0.1284	0.2547	0.0536	0.185	0.3884	0.0204	0.0848	0.1894
DGNN-prop	0.0103	0.0444	0.1087	0.0046	0	0	0.0171	0.0633	0.1514
DGNN-ti	0.0174	0.0918	0.2118	0.0050	0	0.0054	0.0157	0.0591	0.1589
DGNN-att	0.0200	0.0844	0.2235	0.0562	0.1547	0.3219	0.0177	0.0651	0.1655

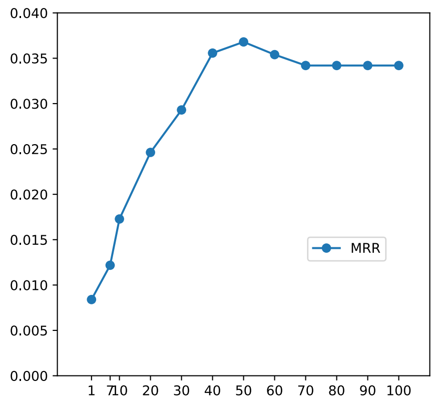


Figure 7: Impact of τ

4 RELATED WORK

In this section, we briefly review two streams of research related to our work: graph neural networks and dynamic graph analysis.

In recent years, many efforts have been made to extend deep neural network models to graph structured data. These neural network models that are applied to graphs are known as graph neural network models [12, 28]. They have been applied to various tasks in many areas. Various graph neural network models have been designed to reason dynamics of physical systems where previous states of the nodes are given as history to predict future states of the nodes [1, 7, 27]. Neural message passing networks have been designed to predict the properties of molecules [11]. Graph convolutional neural networks, which try to perform convolution operations on graph structure data, have been shown to advance many tasks such as graph classification [5, 9], node classification [15, 20, 31] and recommendation [34]. A comprehensive survey on graph convolutional neural networks can be found in [4]. A general framework of graph neural networks was proposed in [2] recently.

Most of the current graph neural network models are designed for static graphs where nodes and edges are fixed. However, many real-world graphs are evolving. For example, social networks are naturally evolving as new nodes joining the graph and new edges being created. It has been of great interest to study the properties of dynamic graphs [6, 16, 18, 35]. Many graph-based tasks such as community detection [24], link prediction [13, 22], node classification [19], knowledge graph mining [30] and network embedding [23, 25, 36] haven been shown to be facilitated by including and modeling the temporal information in dynamic graphs. In this work, we propose a dynamic graph neural network model, which incorporates and models the temporal information in the dynamic graphs.

5 CONCLUSION

In this paper, we propose a novel graph neural graph DGNN for dynamic graphs. It provides two key components – the update component and the propagation component. When a new edge is introduced, the update component can keep node information being updated by capturing the creation sequential information of edges and the time intervals between interactions. The propagation component will propagate new interaction information to the influenced nodes by considering influence strengths. We use link prediction and node classification as examples to illustrate how to leverage DGCN to advance graph mining tasks. We conduct experiments on three real-world dynamic graphs and the experimental results in terms of link prediction and node classification suggest the important of dynamic information and the effectiveness of the proposed update and propagation components in capturing dynamic information.

In the current model, we choose one's neighbors as the set of influenced nodes. Though that choice is reasonable and it works well in practice, we would like to provide some theoretical analysis about this choice and also investigate alternative approaches. Now we illustrate how to use the proposed framework for link prediction and node classification. We also want to investigate how to use the framework for other graph mining tasks especially these under the unsupervised settings such as community detection.

REFERENCES

[1] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. 2016. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*. 4502–4510.

[2] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261* (2018).

[3] Inci M Baytas, Cao Xiao, Xi Zhang, Fei Wang, Anil K Jain, and Jiayu Zhou. 2017. Patient subtyping via time-aware LSTM networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 65–74.

[4] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.

[5] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* (2013).

[6] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. 2012. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems* 27, 5 (2012), 387–408.

[7] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. 2016. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341* (2016).

[8] Shiyu Chang, Yang Zhang, Jiliang Tang, Dawei Yin, Yi Chang, Mark A Hasegawa-Johnson, and Thomas S Huang. 2017. Streaming recommender systems. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 381–389.

[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*. 3844–3852.

[10] Daniel M Dunlavy, Tamara G Kolda, and Evrim Acar. 2011. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 5, 2 (2011), 10.

[11] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212* (2017).

[12] Marco Gori, Gabriele Monfardini, and Franco Scarselli. [n. d.]. A new model for learning in graph domains. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, Vol. 2. IEEE, 729–734.

[13] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2018. DynGEM: Deep Embedding Method for Dynamic Graphs. *arXiv preprint arXiv:1805.11273* (2018).

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

[15] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. 1024–1034.

[16] Frank Harary and Gopal Gupta. 1997. Dynamic graph models. *Mathematical and Computer Modelling* 25, 7 (1997), 79–87.

[17] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[18] Petter Holme and Jari Saramäki. 2012. Temporal networks. *Physics reports* 519, 3 (2012), 97–125.

[19] Ling Jian, Jundong Li, and Huan Liu. 2018. Toward online node classification on streaming networks. *Data Mining and Knowledge Discovery* 32, 1 (2018), 231–257.

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

[21] Jérôme Kunegis. 2013. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 1343–1350.

[22] Jundong Li, Kewei Cheng, Liang Wu, and Huan Liu. 2018. Streaming link prediction on dynamic attributed networks. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 369–377.

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

[24] Yu-Ru Lin, Yun Chi, Shenghuo Zhu, Hari Sundaram, and Belle L Tseng. 2008. Facetnet: a framework for analyzing communities and their evolutions in dynamic networks. In *Proceedings of the 17th international conference on World Wide Web*. ACM, 685–694.

[25] Jianxin Ma, Peng Cui, and Wenwu Zhu. 2018. DepthLGP: Learning Embeddings of Out-of-Sample Nodes in Dynamic Networks. AAAI.

[26] Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a feather: Homophily in social networks. *Annual review of sociology* 27, 1 (2001), 415–444.

[27] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. 2018. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242* (2018).

[28] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80.

[29] J. Tang, H. Gao, and H. Liu. 2012. mTrust: Discerning multi-faceted trust in a connected world. In *Proceedings of the fifth ACM international conference on Web search and data mining*. ACM, 93–102.

[30] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. 2017. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. *arXiv preprint arXiv:1705.05742* (2017).

[31] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2017. Graph Attention Networks. *arXiv preprint arXiv:1710.10903* (2017).

[32] Ellen M Voorhees et al. 1999. The TREC-8 Question Answering Track Report.. In *Trec*, Vol. 99. 77–82.

[33] Rongjing Xiang, Jennifer Neville, and Monica Rogati. 2010. Modeling relationship strength in online social networks. In *Proceedings of the 19th international conference on World wide web*. ACM, 981–990.

[34] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. *arXiv preprint arXiv:1806.01973* (2018).

[35] Ziwei Zhang, Peng Cui, Jian Pei, Xiao Wang, and Wenwu Zhu. 2017. TIMERS: Error-Bounded SVD Restart on Dynamic Networks. *arXiv preprint arXiv:1711.09541* (2017).

[36] Le-kui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic Network Embedding by Modeling Triadic Closure Process.

[37] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*. 912–919.