Dynamic Graph Collaborative Filtering

Xiaohan Li[*†], Mengqi Zhang[*‡§], Shu Wu[‡§||], Zheng Liu[†], Liang Wang[‡§], Philip S. Yu[†]

[†]University of Illinois at Chicago, Chicago, IL, USA

{xli241, zliu212, psyu}@uic.edu

[‡]School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing, China

mengqi.zhang@cripac.ia.ac.cn

[§]Institute of Automation, Chinese Academy of Sciences, Beijing, China

{shu.wu, wangliang}@nlpr.ia.ac.cn

Abstract—**Dynamic recommendation is essential for modern recommender systems to provide real-time predictions based on sequential data. In real-world scenarios, the popularity of items and interests of users change over time. Based on this assumption, many previous works focus on interaction sequences and learn evolutionary embeddings of users and items. However, we argue that sequence-based models are not able to capture collaborative information among users and items directly. Here we propose Dynamic Graph Collaborative Filtering (DGCF), a novel framework leveraging dynamic graphs to capture collaborative and sequential relations of both items and users at the same time. We propose three update mechanisms: zero-order 'inheritance', first-order 'propagation', and second-order 'aggregation', to represent the impact on a user or item when a new interaction occurs. Based on them, we update related user and item embeddings simultaneously when interactions occur in turn, and then use the latest embeddings to make recommendations. Extensive experiments conducted on three public datasets show that DGCF significantly outperforms the state-of-the-art dynamic recommendation methods up to 30%. Our approach achieves higher performance when the dataset contains less action repetition, indicating the effectiveness of integrating dynamic collaborative information.**

Index Terms—**recommender system, dynamic graph**

I. INTRODUCTION

Dynamic recommender systems have proved their effectiveness in many online applications, such as social media, online shopping, and streaming media. They leverage historical interaction sequences of users and items to predict the item that the user may interact with in the future. In real-world scenarios, both user interest and item popularity may shift and evolve along with time. Therefore, it is crucial for a dynamic recommendation model to accurately capture the dynamic changes in user and item perspectives to make accurate predictions. Besides, collaborative information is proved powerful in making recommendation [1]–[3]. Users that share common interacted items tend to have similar interests, and the methods that leverage this property to recommender system is what we call Collaborative Filtering (CF). Consequently, combining dynamic changes of users and items with collaborative information is one of the main tasks in dynamic recommender systems.
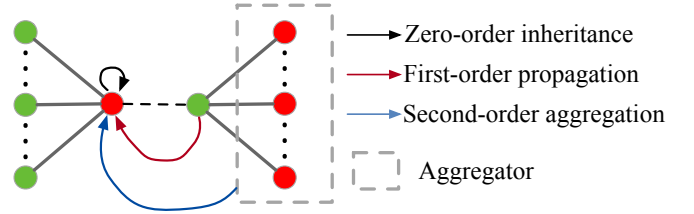
Fig. 1. User-Item graph. Red nodes denote users, and green nodes denote items. A solid line means the user has interacted with the item, a dash line means the user and the item are interacting. Arrows in different colors represent different relations when updating the embedding of a node.

To build a dynamic recommender system, an intuitive way is to model sequences of interactions. Previously, many kinds of sequence-based methods have been developed based on user-item interactions. For example, RNN-based sequential prediction models [4]–[7] use the recurrent architecture to capture the long-term dependency of the item sequences. Besides, to take user sequences into consideration as well, Jodie [8] and other dynamic evolution models [9]–[11] use double RNNs to simultaneously model the update of users and items based on the evolutionary processes of them. We argue that all the models above fail to utilize collaborative information directly. These sequence-based methods primarily model the transition relations between items but ignoring the similarity between users. To mitigate the problem of lacking collaborative information, the graph structure is an alternative instrument for dynamic recommender systems.

Several previous works have shown the advantage of using the graph structure in recommender systems, but they all have restrictions in different aspects. According to [12], [13], the graph structure is capable of incorporating collaborative information explicitly. By taking user-item interactions as a bipartite graph, these models exploit the high-order connectivity of users and items and encode collaborative information into the graph structure. However, all these models are only suitable for static scenarios. The advantages of sequential dependency and time information are wasted in them. Moreover, SR-GNN [14] proves the superiority of graph structures over sequences in the dynamic recommendation, but it fails to incorporate the evolving of items. To deal with these problems, we leverage

dynamic graphs to model the evolutionary process of dynamic recommender systems.

In our proposed dynamic graph, nodes are users and items, and edges are their interactions. In the beginning, the graph only contains isolated user and item nodes. With more users, items and their interactions join the dataset, the nodes and edges evolve and grow to a large graph. To model this process and learn the embeddings of users and items at different times, we develop three update mechanisms in the dynamic graph, which are shown in Figure 1. The first one is **zero-order inheritance**, in which each node inherits its previous embedding and incorporates the new features of it to update its embedding. Secondly, **first-order propagation** builds the connection between two sides of the interaction by propagating one's embedding to the other. It updates the embeddings of the user and item in the interaction simultaneously. Finally, **second-order aggregation** leverages aggregator functions to obtain an overall embedding for all neighbors of the node in the user side, and then pass the embedding to the node in the item side. It is a direct manner to utilize collaborative information.

Based on these three update mechanisms mentioned above, we propose Dynamic Graph Collaborative Filtering (DGCF) to employ all of them under a unified framework. Figure 2 illustrates the workflow of the DGCF model. There are three modules in the model, corresponding to the three update mechanisms. Each part produces an embedding, and then the embeddings generated by the three parts are fused to learn the embedding of the node. At the end of our model, we utilize an evolutionary loss to take the time information, i.e., time stamps, into our model to make recommendation. Details of the model will be introduced in section II. Overall, our proposed DGCF is capable of directly learning user and item embeddings and perform recommendation tasks in an end-to-end framework.

To summarize, our main contributions are listed as follows:

- To the best of our knowledge, our work is the first one to introduce the dynamic graph into dynamic recommendation scenarios to model the interactions and updates between users and items.
- We design a novel framework for the dynamic recommendation task with graph structure, which can effectively model the dynamic relationship between users and items.
- We conduct empirical experiments on three public datasets. Experimental results demonstrate that our DGCF model achieves state-of-the-art results on these datasets, especially for datasets with lower action repetition.

II. PROPOSED MODEL

In this section, we present the proposed Dynamic Graph Collaborative Filtering (DGCF) in detail. We first formulate the dynamic graph recommendation problem. Then we introduce the embedding update and recommendation modules of the DGCF. Finally, we illustrate the process of optimization and training.

TABLE I
NOTATIONS

Notation	Explanation
t^-, t, t^+	Time points of previous, current, and future interaction
$\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$	Dynamic graph at time t
$S_i = (u_i, v_i, t_i, f_i)$	i-th interaction
u, v	User u and item v
$\mathbf{h}_u^t, \mathbf{h}_v^t$	The embedding of user and item at time t
θ, ϕ, ζ	Zero, first, and second-order functions
$\mathbf{W}$	Weight matrix
$\mathcal{H}_v^u = \{v_1, v_2, ..., v_m\}$	Second-order neighbors of item v
$\mathcal{H}_u^v = \{u_1, u_2, ..., u_n\}$	Second-order neighbors of user u
$\mathrm{F}(\cdot)$	Fusion function
MLP	Multi-Layer Perceptron
$\mathbf{f}$	feature vector

A. Preliminary

1) Dynamic recommendation: Let U, V represent the user and item sets, respectively. In a dynamic recommendation scenario, the i-th user-item interaction is represented in a tuple $S_i = (u_i, v_i, t_i, \mathbf{f}_i)$, where $i \in \{1, 2, \cdots, I\}$, and I is the total number of interactions. $u_i \in U$, $v_i \in V$ are the user and item in the interaction and t_i is the time stamp. $\mathbf{f}_i$ denotes features of the interaction, and it includes user features $\mathbf{f}_u$ and item features $\mathbf{f}_v$. The target of dynamic recommendation is to learn the representations of the user and item from current interaction and historical records, and then predict the most possible item that the user will interact with in the future.

2) Dynamic graph: The interactions between users and items at time stamp t construct a dynamic graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$, where $\mathcal{V}_t$, $\mathcal{E}_t$ are the sets of nodes and edges in $\mathcal{G}_t$ respectively. Under recommendation settings, $\mathcal{V}_t$ contains all user and item nodes, and $\mathcal{E}_t$ is the set of all interactions between users and items before time t. Essentially, the graph here is a bipartite graph, and all edges are between user and item nodes. We use $\mathbf{h}_u^t \in \mathbb{R}^d$ and $\mathbf{h}_v^t \in \mathbb{R}^d$ to represent the embedding of user node u and item node v at time t. The initial graph $\mathcal{G}_{t_0} = (\mathcal{V}_{t_0}, \mathcal{E}_{t_0})$ at time t_0 consists of isolated nodes or a snapshot of the dynamic graph, and the initial embeddings of users and items are initial feature vectors or random vectors. Then, while another interaction $S = (u, v, t, \mathbf{f})$ joins the graph, user and item embeddings $\mathbf{h}_u^t$, $\mathbf{h}_v^t$ are updated by our proposed DGCF. Besides, $\mathbf{h}_u^{t-}$ and $\mathbf{h}_v^{t-}$ represent the most recent embeddings of user node u and item node v before time stamp t. The notations used throughout this paper are summarized in Table I.

B. Overview

Figure 2 is an overview of our proposed framework. The interactions between users and items form a dynamic graph over time. When a new user-item interaction joins, we utilize the embedding update mechanism (Section II-C) to update both user and item nodes together. Then, we get the user and item embeddings in the future via the projection functions (Section II-D). Finally, we calculate the $L2$ distance between
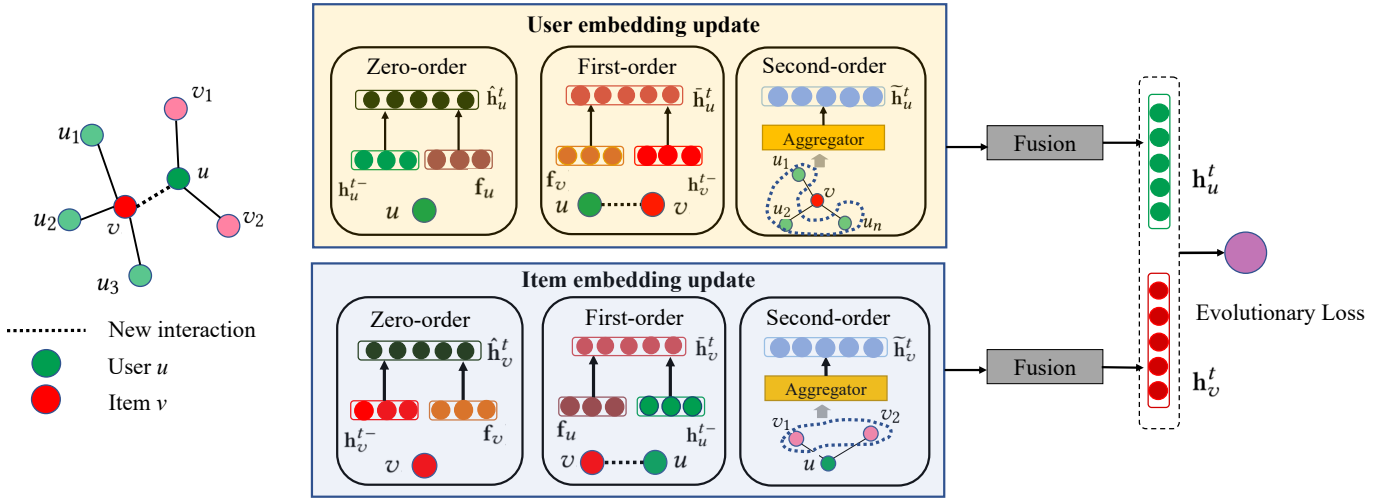
Fig. 2. Illustration of the Dynamic Graph Collaborative Filtering (DGCF). Left: A new interaction joins in the user-item graph. Right: Overall structure of DGCF. The pink and light green nodes means the neighbors of the target nodes. $\mathbf{h}_u^{t^-}$ and $\mathbf{h}_v^{t^-}$ denote the embedding of user u and item v before time t. Based on three update mechanisms, $\mathbf{h}_u^t$, $\mathbf{h}_v^t$ are produced by our DGCF at the same time. With $\mathbf{h}_u^t$, $\mathbf{h}_v^t$, and evolutionary loss function, we can get the recommendation in the end.

the predicted item embedding and all other item embeddings, and then recommend items with the smallest distance to the predicted item embedding.

C. Embedding Update

First, we discuss the embedding update mechanisms: (1) zero-order 'inheritance', which means users and items take the input of their previous embeddings and new features, (2) first-order 'propagation', which models current user-item iteration, and (3) second-order 'aggregation', which aggregates the previous users that have interacted with the item into the current user, and vice versa.

1) Zero-order 'inheritance': For a dynamic graph, the node to be updated firstly inherits the influence of the previous state and new features of itself. Similar to existing sequential prediction methods [4], [5], [8], we use the previous embedding as a part of the input to 'inherit' the previous state. Besides, we additionally encode the time interval between current and previous embeddings as a part of the features to learn the user and item embeddings. For user embedding $\mathbf{h}_u$ and item embedding $\mathbf{h}_v$, the forward formulas are

$$\hat{\mathbf{h}}_u^t = \theta_u(\mathbf{W}_0^u\mathbf{h}_u^{t^-} + \mathbf{w}_0\Delta t_u + \mathbf{W}_0^f\mathbf{f}_u), \quad (1)$$

$$\hat{\mathbf{h}}_v^t = \theta_v(\mathbf{W}_0^v\mathbf{h}_v^{t^-} + \mathbf{w}_0\Delta t_v + \mathbf{W}_0^f\mathbf{f}_v), \quad (2)$$

where Δt is the time interval between current time t and previous interaction time t^-, and $\mathbf{h}_u^{t^-}$, $\mathbf{h}_v^{t^-}$ are the most recent embeddings of user u and item v before t. $\mathbf{f}_u$ and $\mathbf{f}_v$ are the features of the user and the item respectively. $\mathbf{W}_0 \in \mathbb{R}^{d \times d}$ are parameter matrices and $\mathbf{w}_0 \in \mathbb{R}^d$ is the parameter vector to encode time interval Δt. θ_u and θ_v are activation functions. To improve the computational speed, our work here uses an identity map instead of a non-linear activation function.

2) First-order 'propagation': In our model, we build a dynamic bipartite graph to model the interactions between user and item nodes, which means a user node's first-order neighbors are the items that he or she has interacted with, and vice versa.

In dynamic recommendation scenarios, the item that a user interacts with, to some extent, reflects his or her recent intention and interest. Correspondingly, users who are interested in a specific item can be regarded as a part of the item's properties. Therefore, it is essential to exploit the first-order neighbor information to learn user and item embeddings. Note that our model only incorporates the current interacted node as the input instead of taking all first-order neighbors as inputs. To be specific, when an interaction involving user u and item v occurs, on the one hand, item v's current embedding and features like reviews or descriptions are incorporated into user u's embedding. On the other hand, the user u's current embedding and features, e.g., the user's profile, are injected into item v's embedding. Formally,

$$\bar{\mathbf{h}}_u^t = \phi_u(\mathbf{W}_1^u\mathbf{h}_v^{t^-} + \mathbf{W}_1^f\mathbf{f}_v), \quad (3)$$

$$\bar{\mathbf{h}}_v^t = \phi_v(\mathbf{W}_1^v\mathbf{h}_u^{t^-} + \mathbf{W}_1^f\mathbf{f}_u), \quad (4)$$

where $\mathbf{W}_1 \in \mathbb{R}^{d \times d}$ are parameter matrices. In this way, the features of the current interaction are propagated to update user and item embeddings. Similar to zero-order inheritance, we still use identity map function for ϕ_u and ϕ_v.

3) Second-order 'aggregation': In the spirit of modeling collaborative filtering in a dynamic graph, the update between nodes considers not only the historical sequence of the node and the information between interacted nodes, but also the structural information among the nodes. The rationale behind second-order aggregation is to model the collaborative relationship between users and items. In a dynamic graph, it

captures the influence of the nodes at second-order proximity passing through the other node participating in the interaction.

Specifically, for a user, he or she may have bought a bunch of items before the current interaction, and now this user buys a new item, so we assume the newly purchased item has a collaborative relation with the previously purchased items of the user. To model this relation, we build a direct connection, which denoted as $v_u \rightarrow u \rightarrow v$. Here $v_u \in \mathcal{H}_v^u$, where $\mathcal{H}_v^u = \{v_1, v_2, ..., v_m\}$ is the set of previously purchased items of user u, and v is the item in current interaction. Node u serves as a bridge passing information from v_u to node v so that v receives the aggregated second-order information through u. For the user u, the relation is $u_v \rightarrow v \rightarrow u$, where $u_v \in \mathcal{H}_u^v$, $\mathcal{H}_u^v = \{u_1, u_2, ..., u_n\}$ is the set of users who previously purchased item v.

As shown in Figure 2, to make the second-order information flow from the neighbors of v to u, we take u as the anchor node and $u_i \in \mathcal{H}_u^v$ as second-order neighbor nodes in the graph. Then we use aggregator functions to transmit neighborhood information to the anchor node. This process is formulated as

$$\widetilde{\mathbf{h}}_u^t = \zeta_u(\mathbf{h}_u^{t^-}, \mathbf{h}_{u_1}^{t^-}, \mathbf{h}_{u_2}^{t^-}, ..., \mathbf{h}_{u_n}^{t^-}), \tag{5}$$

$$\widetilde{\mathbf{h}}_v^t = \zeta_v(\mathbf{h}_v^{t^-}, \mathbf{h}_{v_1}^{t^-}, \mathbf{h}_{v_2}^{t^-}, ..., \mathbf{h}_{v_m}^{t^-}), \tag{6}$$

where ζ_u and ζ_v are aggregator functions. In the dynamic graph, users and items are two different kinds of nodes with different properties. For example, the neighbors of a user are the items that he or she has interacted with, which are time-dependent. The neighbors of an item are the users who have interacted with it, which tend to be similar. Besides, the number of users that have interacted with a popular item could be significantly large-scale. Thus, the aggregator functions we use also should be different for user and item nodes. The following part provides some candidate aggregator functions we can use in second-order update:

- **Mean aggregator** is a straightforward operator to aggregate the neighbor information of user u and item v. According to [15], **Mean aggregator** can be viewed as an inductive variant of the GCN [16] approach. So, the formulas of the aggregator can be written as follows:

$$\widetilde{\mathbf{h}}_u^t = \mathbf{h}_u^{t^-} + \frac{1}{|\mathcal{H}_u^v|} \sum_{u_i \in \mathcal{H}_u^v} \mathbf{W}_u^m \mathbf{h}_{u_i}^{t^-}, \tag{7}$$

$$\widetilde{\mathbf{h}}_v^t = \mathbf{h}_v^{t^-} + \frac{1}{|\mathcal{H}_v^u|} \sum_{v_i \in \mathcal{H}_v^u} \mathbf{W}_v^m \mathbf{h}_{v_i}^{t^-}, \tag{8}$$

where $\mathbf{W}^m \in \mathbb{R}^{d \times d}$ are aggregation parameters.
- **LSTM aggregator** is a complex aggregator function based on LSTM [17] architecture. By taking sequential data as inputs, LSTM has strong non-linear memory expression capability to keep track of long term memory. From user's perspective, previous items that users have interacted have explicit sequential dependency, so we feed all of the user node's neighbors into the aggregator function in a chronological order. Besides, from the item's perspective, we order

its connected users by time and input them to LSTM. The LSTM aggregator can be formulated as:

$$\widetilde{\mathbf{h}}_u^t = \mathbf{h}_u^{t^-} + \text{LSTM}_u(\mathbf{h}_{u_1}^{t^-}, \mathbf{h}_{u_2}^{t^-}, ..., \mathbf{h}_{u_n}^{t^-}), \tag{9}$$

$$\widetilde{\mathbf{h}}_v^t = \mathbf{h}_v^{t^-} + \text{LSTM}_v(\mathbf{h}_{v_1}^{t^-}, \mathbf{h}_{v_2}^{t^-}, ..., \mathbf{h}_{v_m}^{t^-}). \tag{10}$$

- **Graph Attention aggregator** can compute attention weights between the central node and the neighbor nodes, which indicate the importance of each neighbor node to the central node. Inspired by the GAT [18] model, we define graph attention aggregator as:

$$\widetilde{\mathbf{h}}_u^t = \sum_{u_i \in \mathcal{H}_u^v} \alpha_{ui} \mathbf{h}_{u_i}^{t^-}, \tag{11}$$

$$\alpha_{ui} = \frac{\exp(\text{LeakyRelu}(\mathbf{W}_w[\mathbf{h}_u^{t^-} \parallel \mathbf{h}_{u_i}^{t^-}]))}{\sum_{u_i \in \mathcal{H}_u^v} \exp(\text{LeakyRelu}(\mathbf{W}_w[h_u^{t^-} \parallel \mathbf{h}_{u_i}^{t^-}]))}, \tag{12}$$

$$\widetilde{\mathbf{h}}_v^t = \sum_{u_i \in \mathcal{H}_v^u} \alpha_{vi} \mathbf{h}_{v_i}^{t^-}, \tag{13}$$

$$\alpha_{vi} = \frac{\exp(\text{LeakyRelu}(\mathbf{W}_w[\mathbf{h}_v^{t^-} \parallel \mathbf{h}_{v_i}^{t^-}]))}{\sum_{v_i \in \mathcal{H}_v^u} \exp(\text{LeakyRelu}(\mathbf{W}_w[\mathbf{h}_v^{t^-} \parallel \mathbf{h}_{v_i}^{t^-}]))}, \tag{14}$$

where $\mathbf{W}_w \in \mathbb{R}^{2d}$ is a weight matrix, and $\parallel$ is the concatenation operation.

In practical recommendation scenarios, second-order aggregation may face enormous computational costs due to the large scale of data. Therefore, when performing second-order aggregation, we select a fixed number of neighbors for aggregation. We call the number of neighbor nodes selected as aggregator size.

For higher-order information, we choose not to use it because of the following reasons. Firstly, higher-order information may lead to over-smoothing problem [19], which makes the node embeddings prone to be similar. Besides, using higher-order information increases the computational complexity in power law. As efficiency is a vital issue in recommender system, we try to control the computational complexity to be acceptable in most scenarios and only use up to second-order information.

4) Fusion information: To combine the three kinds of update mechanisms in learning node embeddings in the dynamic graph, we fuse the above mentioned three representations to obtain the final update formula:

$$\mathbf{h}_u^t = \text{F}_u(\mathbf{W}_u^{zero}\hat{\mathbf{h}}_u^t + \mathbf{W}_u^{first}\bar{\mathbf{h}}_u^t + \mathbf{W}_u^{second}\widetilde{\mathbf{h}}_{\mathbf{u}}^{\mathbf{t}}), \tag{15}$$

$$\mathbf{h}_v^t = \text{F}_v(\mathbf{W}_v^{zero}\hat{\mathbf{h}}_v^t + \mathbf{W}_v^{first}\bar{\mathbf{h}}_v^t + \mathbf{W}_v^{second}\widetilde{\mathbf{h}}_{\mathbf{v}}^{\mathbf{t}}), \tag{16}$$

where $\mathbf{h}_u^t, \mathbf{h}_v^t \in \mathbb{R}^d$ are the node embeddings updated after the user u interact with item v at time t. F_u, F_u are fusion functions of user and item respectively. Here we generally choose sigmod $\sigma(\cdot)$ as activation function. $\mathbf{W}^{zero}$, $\mathbf{W}^{first}$, and $\mathbf{W}^{second} \in \mathbb{R}^{d \times d}$ are parameters to control the influence of three update mechanisms.

D. Recommendation

In dynamic recommendation, the goal is to predict the item that the user is most likely to interact with at time t according to his or her historical interaction sequence before time t. Intuitively, this is an analogy to link prediction tasks in dynamic graphs. To be specific, our target is to predict the item node v in the dynamic graph that the user node u is most likely to link to at time t. Based on [8], we propose an evolutionary loss for dynamic graph.

1) Evolution formula: Different from traditional collaborative filtering methods, our model is designed for predicting future interaction. Specifically, given a future time point, we can leverage our model to predict the future embeddings and then make recommendation. It is a more flexible setting, because the predicted results do not rely on the sequences. Instead, they are based on the embeddings learned by the dynamic graph structure.

Since $\mathbf{h}_u^t$ means the predicted embedding of the user's future, we need an estimated future embeddings to measure whether the predicted embedding is accurate. Motivated by [8], we assume the growth of users is smooth, so the embedding vector of the user node evolves in a contiguous space. Therefore, we set a projection function to estimate the future embedding based on element-wise product of the previous embedding and time interval. We define the embedding projection formula of user u after current time t to the future time t^+ as follows:

$$\widehat{\mathbf{h}}_u^{t^+} = \mathrm{MLP}_u(\mathbf{h}_u^t \odot (\mathbf{1} + \mathbf{w}_t(t^+ - t))), \quad (17)$$

where $\mathbf{w}_t \in \mathbb{R}^d$ is time-context parameter to convert the time interval to vector, $\mathbf{1} \in \mathbb{R}^d$ is a vector with all elements 1. MLP here means Multi-Layer Perceptron. t^+ is the future time that the user interacts with the next item. With this projection function, the future item embedding grows in a smooth trajectory w.r.t. the time interval.

After obtaining the projected embedding $\widehat{\mathbf{h}}_u^{t^+}$ of the user u, we learn the future embedding of the item v denoted as $\widehat{\mathbf{h}}_v^{t^+}$ by setting another projection function. The projected item embedding is based on three parts: the user that currently interacts with, the update features of the user and the item itself, which are all we have already known. So, we define the projection formula of item v as:

$$\widehat{\mathbf{h}}_v^{t^+} = \mathrm{MLP}_v(\mathbf{W}_2\widehat{\mathbf{h}}_u^{t^+} + \mathbf{W}_3\mathbf{f}_u + \mathbf{W}_4\mathbf{f}_v)), \quad (18)$$

where $\mathbf{W}_2, \mathbf{W}_3$ and $\mathbf{W}_4$ denote the weight matrix.

2) Loss function: When we have the estimated future embeddings by projection functions, we take them as ground truth embeddings in our loss function. In order to train our model, the loss function is composed of Mean Square Error (MSE) between model-generated embeddings $\mathbf{h}_v^t$, $\mathbf{h}_u^t$ and estimated ground truth embeddings $\widehat{\mathbf{h}}_v^{t^+}$, $\widehat{\mathbf{h}}_u^{t^+}$ at each interaction time t. Besides, we need another constraint for the item embedding to avoid overfitting. We constrain the distance between model-generated $\mathbf{h}_v^t$ and mostly recently embedding $\mathbf{h}_v^{t^-}$ of item v, and between $\mathbf{h}_u^t$ and $\mathbf{h}_u^{t^-}$, respectively, to make

the node embedding more consistent with the previous one. The assumption behind this constraint is that items' and users' properties tend to be stable in a short time. The loss function is written as follows:

$$\mathcal{L} = \sum_{(u,v,t,f)\in\{S_i\}_{i=0}^I} \|\widehat{\mathbf{h}}_v^{t^+} - \mathbf{h}_v^t\|_2 + \lambda_u\|\mathbf{h}_u^t - \mathbf{h}_u^{t^-}\|_2 + \tag{19}$$
$$\alpha_v\|\mathbf{h}_v^t - \mathbf{h}_v^{t^-}\|_2,$$

where $\{S_t\}_{i=0}^I$ denotes the interaction events sorted by chronological order, and λ_u and α_v are smooth coefficients, which are used to prevent the embedding of user and item from deviating too much during the update process.

To make recommendations for a user, we calculated the $L2$ distances between the predicted item embedding that we obtain from the loss function and all other item embeddings. Then the nearest Top-k items are what we predict for the user.

Compared with traditional BPR loss [3], the evolutionary loss is more suitable for dynamic recommendation, because it takes time into account. As a result, the changing trajectories for users and items are modeled by this loss [8], and it can make more precise recommendation for the next item.

E. Optimization and Training

Similar with Recurrent Neural Networks (RNNs), we apply the back-propagation through time (BPTT) algorithm for model training. The model parameters are optimized by Adam optimizer [20].

To speed up the training process, we use the same method of constructing batches as [8]. As mentioned in [8], the training algorithm needs to follow two critical criteria: (1) it should process the interactions in each batch simultaneously, and (2) the batching process should maintain the original temporal ordering of the interactions and keep the sequential dependency in the generated embeddings. In practice, we arrange the interaction events S_i in chronological order to get an event sequence $\{S_1, S_2, \cdots, S_I\}$ numbered by integer, and I is the total number of interactions. We traverse through the temporally sorted sequence of interactions iteratively and put each interaction to a $Batch_k$, where $k \in [1, I]$. In the initial stage of constructing the $Batch$ sequence: each $Batch$ set is empty at first, and the $Batch$ index is -1. We define as $B_{init}(u) = -1$, $B_{init}(v) = -1$. After each interaction (u, v, t, f_r) is added to $Batch$, we update the batch index of the user u and item v. For each interaction, the index of the added $Batch$ is $\max\{B(u) + 1, B(v) + 1)\}$. When the interaction is added to the $Batch$, we update the index of the added u and v. This mechanism ensures that the embeddings of users and items in the same $Batch$ are updated simultaneously in the training and testing process.

III. EXPERIMENTS

All source codes and datasets are provided in this link[1]. In this section, we design the experiments to answer the following questions:

---

[1] https://github.com/CRIPAC-DIG/DGCF

Data	Users	Items	Interactions	Action Repetition
Reddit	10000	1000	672447	79%
Wikipedia	8227	1000	157474	61%
LastFM	1000	1000	1293103	8.6%

Q1: How does DGCF perform compared with other state-of-the-art dynamic or sequential models?

Q2: What is the influence of three types of embedding update mechanisms (zero-order inheritance, first-order propagation, second-order aggregation) in DGCF?

Q3: What is the effect of different aggregator functions in second-order aggregation on model performance?

Q4: How do different hyper-parameters (e.g. aggregation size) affect the performance of DGCF?

A. Datasets Description

To evaluate the proposed model, we conduct experiments on three real-world datasets. The amounts of users, items and interactions for datasets and their action repetitions are listed in Table II. It is worth emphasizing that the three datasets differ significantly in terms of users' repetitive behaviors. Here is the details of the datasets:

Reddit: This dataset contains one month of posts made by users on subreddits [21]. The 10,000 most active users and the 1,000 most active subreddits are selected and treated as users and items respectively, and they have 672,447 interactions. Besides, each post's text is converted into a feature vector to represent their LIWC categories [22].

Wikipedia edits: This dataset contains one month of edits on Wikipedia [23]. The editors who made at least 5 edits and the 1,000 most edited pages are filtered out as users and items for recommendation. This dataset contains 157,474 interactions in total. Similarly, the edit texts are converted into an LIWC-feature vector.

LastFM: This dataset contains the listening records of users within a month [24]. 1000 users and the 1000 most listened songs are selected as users and items, and 1,293,103 interactions are in this dataset. Note that interactions do not have features in this dataset.

All user-item interactions are arranged in chronological order. Then we split the training, validation and test set in a proportion of 80%, 10%, 10% for each dataset. For each interaction $(u, v, t, \mathbf{f})$ in the test set, our goal is to use the given u and v to predict the item that the user is most likely to interact with at time t.

B. Compared Methods

To evaluate the performance of DGCF, we compare it with the following baseline methods:

- **LSTM** [17]: It is a variant of RNN whose name is Long Short-Term Memory (LSTM). LSTM updates user (session) embedding by inputting a sequence of historical interacted items of the user into the LSTM cell, which could capture the long-term dependence of the item sequence.
- **Time-LSTM** [25]: It uses time gates in LSTM to model time intervals in the interaction sequences.
- **RRN** [7]: Recurrent Recommender Network (RRN) predicts future trajectories to learn user and item embeddings based on LSTM.
- **CTDNE** [10]: It is a state-of-the-art model in generating embeddings from temporal networks, but it only produces static embeddings.
- **DeepCoevolve** [26]: It is based on co-evolutionary point process algorithms. We use 10 negative samples per interaction following the setting of [8].
- **Jodie** [8] : It is a state-of-the-art model in dynamic recommendation problem. It defines a projection operation to predict dynamic embedding trajectory.

C. Experimental Settings

1) **Evaluation Metrics:** Two evaluation metrics are used to measure the performance of our DGCF framework:

(1) **Mean Reciprocal Rank (MRR)** supposes the model produces a list of recommended items to a user, and the list is ordered by confidence of the prediction. With MRR, we can measure the performance of the model with respect to the ranking list of items. Higher MRR score means target items tend to have higher rank positions in the predicted item lists. Formally, MRR is defined as:

$$MRR = \frac{1}{|N|} \sum_{i=1}^{|N|} \frac{1}{rank_u},\qquad(20)$$

where $rank_u$ represents the rank position of the target item for user u.

(2) **Recall@10** means the number of target items that are in the top-10 recommendation lists. To calculate recall@10, we use equation

$$Recall@10 = \frac{n_{hit}}{n_{test}},\qquad(21)$$

where n_{hit} is the number of target items that are among the top-10 recommendation list and n_{test} is the number of all test cases.

2) **Parameter Settings:** We implement DGCF framework in *Pytorch*[2]. The dimensionality of embeddings is 128 for all attempts. We use randomly sampled vectors in Gaussian distribution with a mean of 0 and a variance of 1 to initialize embeddings of the users and items. Features of users and items are one-hot vectors. Adam optimizer with learning rate $1e-3$, $L2$ penalty $1e-3$ is adopted in our model. The smoothing coefficients λ and α in loss function are set to 1. We run 50 epochs each time and select the best attempt based on the validation set for testing. For comparison methods, we mostly use the default hyperparameters in their paper.

---

[2]https://pytorch.org/

Models	LastFM		Wikipedia		Reddit	
	MRR	R@10	MRR	R@10	MRR	R@10
LSTM	0.081	0.127	0.332	0.459	0.367	0.573
Time-LSTM	0.088	0.146	0.251	0.353	0.398	0.601
RRN	0.093	0.199	0.530	0.628	0.605	0.751
CTDNE	0.010	0.010	0.035	0.056	0.165	0.257
DeepCoevolve	0.021	0.042	0.515	0.563	0.243	0.305
Jodie	_0.239_	_0.387_	_0.746_	_0.821_	_0.724_	_0.851_
DGCF	**0.321**	**0.456**	**0.786**	**0.852**	**0.726**	**0.856**
Improvement	34.3%	27.7%	5.4%	3.6%	0.2%	0.5%

D. Performance Comparison (Q1)

To prove the superiority of our proposed DGCF model, we conduct experiments on three datasets and compare our model with six baseline methods. Table III shows that DGCF significantly outperforms all six baselines on the three datasets according to both MRR and Recall@10. Especially on LastFM, the improvements are 34.3% on MRR and 27.7% on Recall@10. Based on the experimental results shown in Table III, we find three following facts:

- DGCF yields significant improvements on LastFM. Besides, compared with Jodie, the improvements on Reddit, Wikipedia, and LastFM are in an increasing order, which are consistent with the repetitive action pattern in the datasets. The main reason for the improvements might be that DGCF explicitly considers the collaborative information in the graph. LastFM dataset includes users and their listened songs. Intuitively, users tend to listen to different songs that belong to similar genres, and that is why LastFM dataset has a low action repetition. In this case, it is not easy to make recommendations only based on sequences. However, with collaborative information we utilize in DGCF, we can find similar sequences from other users and make recommendations. It can be proof that DGCF can deal with low action repetition situation.
- In DGCF, the improvements on MRR are greater than on Recall@10. It may result from projection functions and the evolutionary loss. Because our loss considers the future time, the predicted results corresponding to different time stamps tend to be different. In this case, our model has a better performance on the ranking of recommended items.

E. Ablation Study (Q2)

As the three update mechanisms are crucial in DGCF, we measure their effectiveness respectively by conducting experiments. We compare our model with different ablation models: (1) DGCF-0: DGCF without zero-order inheritance. (2) DGCF-1: DGCF without first-order propagation. (3) DGCF-2: DGCF without second-order aggregation. The DGCF here uses
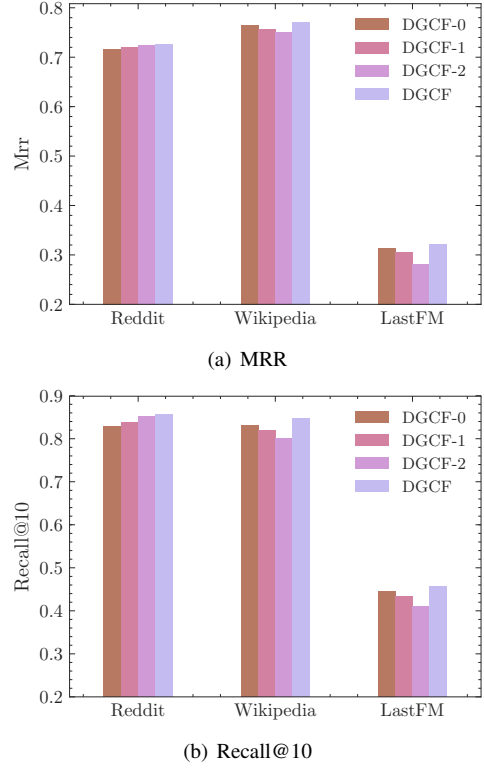

(a) MRR

(b) Recall@10

Fig. 3. Ablation Study. It shows different node relations affect the performance of the model. Due to the different characteristics of datasets, The results are different on several datasets.

graph attention as its aggregator function. Figure 3 summarizes the experimental results.

Compared with ablation models, DGCF achieves the best on all datasets, which proves the effectiveness of each module of our model. The following observations are further derived from the results:

- On the LastFM dataset, we find that DGCF-2 shows evidently lower performance than others. Since the second-order update is capable of finding the users who like the same songs, it is plausible that second-order aggregation is advantageous on the LastFM dataset. For example, when a user does not have many repetitive actions to indicate his/her interests, our model is still capable of modeling the user based on the similarity with other users. As mentioned in Q1, DGCF is able to deal with low action repetition datasets. Without the second-order update, our model cannot achieve similar results on LastFM. As a result, it proves the correctness of this assumption.
- Compared to DGCF, DGCF-0 and DGCF-1 tend to be stable. It shows that when the model lacks sequential information or current interaction information, second-order aggregation can still have a decent performance by taking advantage of collaborative information.

F. Aggregator Function (Q3)

In this subsection, we test the effectiveness of Mean, LSTM, and Graph attention aggregator functions respectively. The

Aggregator	LastFM		Reddit		Wikipedia	
	MRR	R@10	MRR	R@10	MRR	R@10
Mean	0.296	0.419	0.721	0.844	0.770	0.836
LSTM	0.291	0.425	0.721	0.841	0.755	0.815
Attention	0.321	0.456	0.726	0.856	0.786	0.852

experimental results are shown in Table IV.

In Table IV, graph attention outperforms the other two aggregator functions on all datasets. It may because that attention mechanism can learn the strength of the relations between the central node and its neighbors. In other words, graph attention is able to explicitly select effective collaborative information among many second-order neighbor nodes to update user or item embeddings. Compared with graph attention, the drawback of mean and LSTM is that neighbors of the central node are equally important when performing aggregation, so that the most influential nodes might be ignored. Different from the other two datasets, low action repetition dataset LastFM depends more on second-order aggregation, so the attention mechanism improves more on it.

G. Hyper-Parameter Study (Q4)

Appropriate aggregator size not only guarantees model performance but also speeds up training and inference. In this section, we test the performance of DGCF when applying different aggregation sizes. We select graph attention aggregator and choose the aggregation size as $20, 40, 60, 80, 100, 120$. Figure 4 shows the results. According to results, a smaller T can lead to higher performance. The degradation of performances when aggregation sizes increase maybe because of the redundancy of second-order collaborative information. Therefore, we can reduce the aggregator size to improve training and inference speed.

IV. DISCUSSION

In this section, we compare the DGCF with two representative dynamic models, which are RNN and Jodie, to highlight the main differences and innovative parts of our model.

A. Difference between RNN and DGCF

Recurrent Neural Network (RNN) based models, such as GRU4Rec [4], NARM [5], are widely used in sequential recommendation problems. These models take the user's historical item sequences as input to model the user's interest. However, RNN is only a special case of our DGCF when first and second-order relations are removed. It makes RNN fail to consider the variations of items as well as the interactions between users and items.

B. Difference between Jodie and DGCF

Joint Dynamic User-Item Embeddings (Jodie) [8] is a representative method of co-evolving based models. It uses two
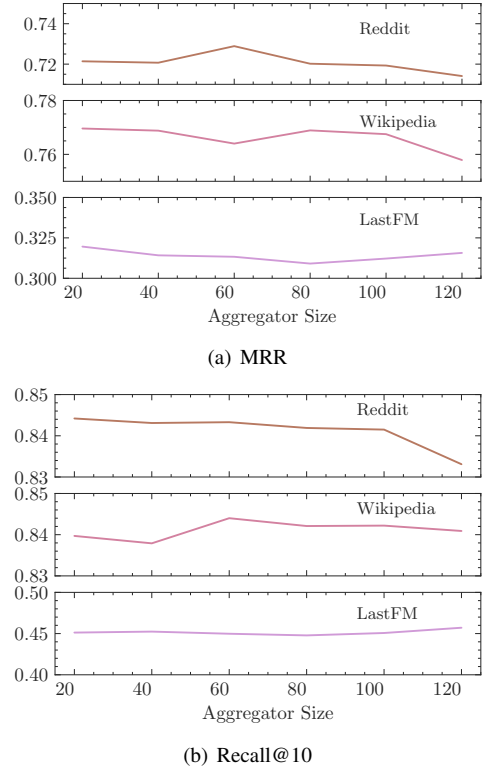

(a) MRR

(b) Recall@10

Fig. 4. Aggregation size. As the aggregator size changes, the performance fluctuation of the model on the datasets is small, indicating that the model is generally robust to aggregator sizes in second-order aggregation.

RNNs, which are user-RNN and item-RNN and projection operation to predict the embedding of users and items. Our DGCF model shares similar motivation with Jodie, but compared to DGCF, Jodie is also a special case with no second-order aggregation. This makes it difficult to explicitly model the collaborative relation between users and items. Therefore, when users have few repetitive actions, RNN-based model cannot achieve satisfactory results, and collaborative relations play an important role in this case. This difference results in our significant improvement over Jodie on LastFM dataset, which has the lowest action repetition rate.

V. RELATED WORKS

In this section, we review related works on dynamic recommendation models, graph-based recommendation and dynamic graph representation learning. We also include the comparison between our model and previous methods in the end of each subsection.

A. Dynamic recommendation

Distinct from static recommendation models like Matrix Factorization (MF) and Bayesian Personal Ranking (BPR), the main task of dynamic recommendation models is to capture the variations of users and items. Recently, Recurrent Neural Network (RNN) and its variants (LSTM and GRU) are widely used in the dynamic recommendation problems. Hidasi et al. [4] apply RNN in session-based recommendation, which is a

sub-problem of dynamic recommendation, to model the fluctuation of users' interests. Time-LSTM [25] combines LSTM with time gates to model time differences between interactions. Although the RNN model has made great progress in the sequential or dynamic recommendation, it has shortcomings in long-range dependence. To solve the problem of long-distance dependence, NARM, Stamp, and SasRec [5], [27], [28] utilizes attention mechanism to capture users' main purposes, and also improve the training speed. Besides, CNN models [29] are also introduced to dynamic recommendation. However, all these methods only utilize the item trajectory of a user to model variations in the user's interest, while ignoring the evolution of items.

To deal with this problem, some methods that jointly learn representations of users and items by using the point process model [26] and RNN model [7]. Jodie [8] predicts user and item embedding with RNN and projection operation. All methods above are recommendation models based on RNNs from different perspectives. They mainly deal with the item sequences or user sequences in the temporal order. However, collaborative signal, which means indirect connections between user-user or item-item, is not used among these works. Therefore, in the DGCF, we not only consider both user and item sequence, but also exploit high-order neighbors in the user-item graph to enrich the training data.

B. Graph-based recommendation

Users, items, and their interactions can be seen as two types of nodes and edges in a bipartite graph. The advantages of modeling user-item interactions as a graph are: 1) graph-based algorithms like random walk and Graph Neural Networks (GNNs) can be applied to predict links between users and items. 2) High-order connectivity can be explored to enrich training data. Because of the ability to reach high-order neighbors, random walk is tried in making recommendations on the interaction graph. HOP-Rec [30] performs random walks on the graph to consider high-order neighbors. RecWalk [31] is also a random walk-based method that leverages the spectral properties of nearly uncoupled Markov chains for the top-N recommendation. However, random walk-based methods have an issue of lacking robustness.

Nowadays, GNNs show remarkable success in many applications [32]–[38]. The effectiveness of GNNs is also proved on recommendation problems. GCMC [39] applies Graph Convolutional Networks (GCN) [16] in completing the user-item matrix. PinSage [40] introduces GraphSAGE [15] into recommender system on item-item graph. Spectral CF [13] leverage spectral convolution over the user-item bipartite graph to discover possible connections in the spectral domain. SR-GNN [14] and A-PGNN [41] use GNNs on session-based recommendation. BasConv [42] leverages graph convolution to user-basket-item graph embedding. NGCF [12] propagates user and item embeddings hierarchically to model high-order connectivity. Although these methods achieve significant performances on static recommender system, all of them fail to make use of the influence of time, and the dynamics in

users and items are not well considered among these methods. Therefore, we take the graph-based recommendation models under a dynamic graph framework to combine graph structure with time series.

C. Dynamic graph representation learning

Representation learning over graph-structured data has received wide attention [15], [43]–[45]. It aims to encode the high-dimensional graph information into low-dimensional vectors. However, in real-world graph data like social networks and citation networks, the graphs are always evolving. To deal with this problem, the graph encoding methods should also consider the dynamics of data, and we call this kind of methods as dynamic graph representation learning.

Previously, some preliminary methods take the evolving graph as snapshots during the discrete-time. DANE [46] proposes an embedding method on dynamic attributed network. It models the variations of the adjacency matrix and attribute matrix based on matrix perturbation. DynGEM [47] trains a deep autoencoder based model across snapshots of the graph to learn stable graph embeddings over time. TIMERS [48] propose an incremental method based on Singular Value Decomposition (SVD) for dynamic representation learning. DyRep [49] defines association and communication events on dynamic graphs and learning the representation based on graph attention neural networks. Our DGCF model is motivated by these works, especially DyRep, to learn dynamic representations of users and items under recommendation scenarios. However, the recommendation scenario is different from social networks because we have two types of nodes and different influences for dynamic events. Under these circumstances, we develop our DGCF model to capture dynamics on both user and item level.

VI. Conclusion

In this paper, we associate the dynamic graph with the dynamic recommendation scenarios and propose a novel framework based on dynamic graph for dynamic recommendation: Dynamic Graph Collaborative Filtering, abbreviated as DGCF. In DGCF, we design three dynamic node update mechanisms for learning node embedding and making recommendations. Experimental results show that our model outperforms all seven baselines.

The proposed DGCF is an initial trial of combining dynamic graph with recommender system. Apart from user-item bipartite graph, many other kinds of graph structure can be explored with dynamic graph, e.g., knowledge graph, social network, and attributed graph.

VII. Acknowledgement

REFERENCES

[1] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized markov chains for next-basket recommendation," in *WWW*. ACM, 2010, pp. 811–820.

[2] Y. Koren, "Collaborative filtering with temporal dynamics," in *KDD*. ACM, 2009, pp. 447–456.

[3] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," in *UAI*. AUAI Press, 2009, pp. 452–461.

[4] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," in *ICLR*, 2016.

[5] J. Li, P. Ren, Z. Chen, Z. Ren, T. Lian, and J. Ma, "Neural attentive session-based recommendation," in *CIKM*. ACM, 2017, pp. 1419–1428.

[6] Q. Liu, S. Wu, D. Wang, Z. Li, and L. Wang, "Context-aware sequential recommendation," in *ICDM*. IEEE, 2016, pp. 1053–1058.

[7] C.-Y. Wu, A. Ahmed, A. Beutel, A. J. Smola, and H. Jing, "Recurrent recommender networks," in *WSDM*. ACM, 2017, pp. 495–503.

[8] S. Kumar, X. Zhang, and J. Leskovec, "Predicting dynamic embedding trajectory in temporal interaction networks," in *KDD*. ACM, 2019, pp. 1269–1278.

[9] J. You, Y. Wang, A. Pal, P. Eksombatchai, C. Rosenburg, and J. Leskovec, "Hierarchical temporal convolutional networks for dynamic recommender systems," in *WWW*. ACM, 2019, pp. 2236–2246.

[10] Y. Wang, N. Du, R. Trivedi, and L. Song, "Coevolutionary latent feature processes for continuous-time user-item interactions," in *NeurIPS*, 2016, pp. 4547–4555.

[11] Q. Wu, Y. Gao, X. Gao, P. Weng, and G. Chen, "Dual sequential prediction models linking sequential recommendation and information dissemination," in *KDD*. ACM, 2019, pp. 447–457.

[12] X. Wang, X. He, M. Wang, F. Feng, and T.-S. Chua, "Neural graph collaborative filtering," in *SIGIR*, 2019, pp. 165–174.

[13] L. Zheng, C.-T. Lu, F. Jiang, J. Zhang, and P. S. Yu, "Spectral collaborative filtering," in *Proceedings of the 12th ACM Conference on Recommender Systems*. ACM, 2018, pp. 311–319.

[14] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan, "Session-based recommendation with graph neural networks," in *AAAI*, vol. 33, 2019, pp. 346–353.

[15] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017, pp. 1024–1034.

[16] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *ICLR*, 2017.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *ICLR*, 2018.

[19] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," in *AAAI*, 2018.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ICLR*, 2015.

[21] J. Baumgarten, "Reddit data dump," *URL https://files. pushshift. io/reddit*.

[22] J. W. Pennebaker, M. E. Francis, and R. J. Booth, "Linguistic inquiry and word count: Liwc 2001," *Mahway: Lawrence Erlbaum Associates*, vol. 71, 2001.

[23] G. Kossinets, "Processed wikipedia edit history," 2012.

[24] B. Hidasi and D. Tikk, "Fast als-based tensor factorization for context-aware recommendation from implicit feedback," in *ECML-PKDD*. Springer, 2012, pp. 67–82.

[25] Y. Zhu, H. Li, Y. Liao, B. Wang, Z. Guan, H. Liu, and D. Cai, "What to do next: Modeling user behaviors by time-lstm." in *IJCAI*, 2017, pp. 3602–3608.

[26] H. Dai, Y. Wang, R. Trivedi, and L. Song, "Deep coevolutionary network: Embedding user and item features for recommendation," *arXiv preprint arXiv:1609.03675*, 2016.

[27] Q. Liu, Y. Zeng, R. Mokhosi, and H. Zhang, "Stamp: short-term attention/memory priority model for session-based recommendation," in *KDD*. ACM, 2018, pp. 1831–1839.

[28] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *ICDM*. IEEE, 2018, pp. 197–206.

[29] J. Tang and K. Wang, "Personalized top-n sequential recommendation via convolutional sequence embedding," in *WSDM*. ACM, 2018, pp. 565–573.

[30] J.-H. Yang, C.-M. Chen, C.-J. Wang, and M.-F. Tsai, "Hop-rec: high-order proximity for implicit recommendation," in *Proceedings of the 12th ACM Conference on Recommender Systems*. ACM, 2018, pp. 140–144.

[31] A. N. Nikolakopoulos and G. Karypis, "Recwalk: Nearly uncoupled random walks for top-n recommendation," in *WSDM*. ACM, 2019, pp. 150–158.

[32] H. Peng, J. Li, Q. Gong, Y. Song, Y. Ning, K. Lai, and P. S. Yu, "Fine-grained event categorization with heterogeneous graph convolutional networks," in *IJCAI*. AAAI Press, 2019, pp. 3238–3245.

[33] Y. Dou, Z. Liu, L. Sun, Y. Deng, H. Peng, and P. S. Yu, "Enhancing graph neural network-based fraud detectors against camouflaged fraudsters," *arXiv preprint arXiv:2008.08692*, 2020.

[34] Z. Li, Z. Cui, S. Wu, X. Zhang, and L. Wang, "Fi-gnn: Modeling feature interactions via graph neural networks for ctr prediction," in *CIKM*. ACM, 2019, pp. 539–548.

[35] Y. Zhang, X. Yu, Z. Cui, S. Wu, Z. Wen, and L. Wang, "Every document owns its structure: Inductive text classification via graph neural networks," *arXiv preprint arXiv:2004.13826*, 2020.

[36] H. Peng, H. Wang, B. Du, M. Z. A. Bhuiyan, H. Ma, J. Liu, L. Wang, Z. Yang, L. Du, S. Wang *et al.*, "Spatial temporal incidence dynamic graph neural networks for traffic flow forecasting," *Information Sciences*, vol. 521, pp. 277–290, 2020.

[37] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Deep Graph Contrastive Representation Learning," in *ICML Workshop on Graph Representation Learning and Beyond*, 2020. [Online]. Available: http://arxiv.org/abs/2006.04131

[38] F. Yu, Y. Zhu, Q. Liu, S. Wu, L. Wang, and T. Tan, "Tagnn: Target attentive graph neural networks for session-based recommendation," *arXiv preprint arXiv:2005.02844*, 2020.

[39] R. v. d. Berg, T. N. Kipf, and M. Welling, "Graph convolutional matrix completion," *KDD Deep Learning Day*, 2018.

[40] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *KDD*. ACM, 2018, pp. 974–983.

[41] S. Wu, M. Zhang, X. Jiang, X. Ke, and L. Wang, "Personalizing graph neural networks with attention mechanism for session-based recommendation," *arXiv preprint arXiv:1910.08887*, 2019.

[42] Z. Liu, M. Wan, S. Guo, K. Achan, and P. S. Yu, "Basconv: Aggregating heterogeneous interactions for basket recommendation with graph convolutional neural network," in *Proceedings of the 2020 SIAM International Conference on Data Mining*. SIAM, 2020, pp. 64–72.

[43] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*. ACM, 2014, pp. 701–710.

[44] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.

[45] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.

[46] J. Li, H. Dani, X. Hu, J. Tang, Y. Chang, and H. Liu, "Attributed network embedding for learning in a dynamic environment," in *CIKM*. ACM, 2017, pp. 387–396.

[47] P. Goyal, N. Kamra, X. He, and Y. Liu, "Dyngem: Deep embedding method for dynamic graphs," *arXiv preprint arXiv:1805.11273*, 2018.

[48] Z. Zhang, P. Cui, J. Pei, X. Wang, and W. Zhu, "Timers: Error-bounded svd restart on dynamic networks," in *AAAI*, 2018.

[49] R. Trivedi, M. Farajtabar, P. Biswal, and H. Zha, "Dyrep: Learning representations over dynamic graphs," in *ICLR*, 2019.