Sequential Recommendation with Graph Neural Networks

Jianxin Chang[1], Chen Gao[1], Yu Zheng[1], Yiqun Hui[2], Yanan Niu[2], Yang Song[2], Depeng Jin[1], Yong Li[1]
[1]Beijing National Research Center for Information Science and Technology (BNRist),
Department of Electronic Engineering, Tsinghua University
[2]Beijing Kuaishou Technology Co., Ltd.
liyong07@tsinghua.edu.cn

ABSTRACT

Sequential recommendation aims to leverage users' historical behaviors to predict their next interaction. Existing works have not yet addressed two main challenges in sequential recommendation. First, user behaviors in their rich historical sequences are often implicit and noisy preference signals, they cannot sufficiently reflect users' actual preferences. In addition, users' dynamic preferences often change rapidly over time, and hence it is difficult to capture user patterns in their historical sequences. In this work, we propose a graph neural network model called SURGE (short for ***S**eq**U**ential **R**ecommendation with **G**raph neural n**E**tworks*) to address these two issues. Specifically, SURGE integrates different types of preferences in long-term user behaviors into clusters in the graph by re-constructing loose item sequences into tight item-item interest graphs based on metric learning. This helps explicitly distinguish users' core interests, by forming dense clusters in the interest graph. Then, we perform cluster-aware and query-aware graph convolutional propagation and graph pooling on the constructed graph. It dynamically fuses and extracts users' current activated core interests from noisy user behavior sequences. We conduct extensive experiments on both public and proprietary industrial datasets. Experimental results demonstrate significant performance gains of our proposed method compared to state-of-the-art methods. Further studies on sequence length confirm that our method can model long behavioral sequences effectively and efficiently.

KEYWORDS

Sequential Recommendation, Graph Neural Networks, Dynamic User Preferences

ACM Reference Format:
Jianxin Chang[1], Chen Gao[1], Yu Zheng[1], Yiqun Hui[2], Yanan Niu[2], Yang Song[2], Depeng Jin[1], Yong Li[1]. 2021. Sequential Recommendation with Graph Neural Networks. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21), July 11–15, 2021, Virtual Event, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3404835.3462968

SIGIR '21, July 11–15, 2021, Virtual Event, Canada

ACM ISBN 978-1-4503-8037-9/21/07…$15.00
https://doi.org/10.1145/3404835.3462968

1 INTRODUCTION

Sequential recommendation attempts to predict a user's next behavior by exploiting their historical behavior-sequences, which has been widely adopted in modern online information systems, such as news, video, advertisements, etc. Differ from traditional recommendation tasks that model user preferences in a static fashion, sequential recommendation is capable of capturing user's evolved and dynamic preferences. For example, a user may prefer to watch soccer news only during the period of World Cup, which can be regarded as a kind of *short-term* preference.

Existing works have realized the significance of modeling fast-changing short-term preferences, by approaching the problem from three perspectives. Specifically, early efforts [23, 46] adopt human-designed rules or attention mechanism to assign time-decaying weights to historically interacted items. The second category of works [10, 45] leverages recurrent neural networks to summarize the behavioral sequences, but they suffer from the short-term bottleneck in capturing users' dynamic interests due to the difficulty of modeling long-range dependencies. Recent solutions [39] jointly model long-term and short-term interests to avoid forgetting long-term interests, but the division and integration of long/short-term interests are still challenging. In short, the aforementioned works commonly concentrate more on user behaviors of recent times, and are not capable of fully mining older behavior-sequences to accurately estimate their current interests. As a result, there are two major challenges in sequential recommendation that have not been well-addressed so far as follows.

- **User behaviors in long sequences reflect implicit and noisy preference signals.** Users may interact with many items with implicit feedback, such as clicks and watches. Unlike explicit feedback that can infer user preferences, such as likes and favorites, single implicit feedback cannot reflect user preferences. The user may click on items that are not of their interest most of the time and will not choose similar items for interaction afterward. However, these records will serve as noises in the user's behavior history, worsening the modeling of their real interests.
- **User preferences are always drifting over time due to their diversity.** As we have mentioned, user preferences are changing, no matter slow or fast. Given a point of time, some preferences may be still activated and some others may have been deactivated. Thus, even if we have extracted user preferences from the implicit and noisy behaviors, it is still challenging to model how they change in the history and estimate the activated preferences at the current time, which is the core of recommendation models.

To address these two challenges, we propose a graph-based method with graph convolutional networks to extract implicit preference signals. The dynamic graph pooling is then used to capture

the dynamics of preferences. Specifically, we first convert the loose item sequence to a tight item-item graph and design a attentive graph convolutional network that gather weak signals to strong ones that can accurately reflect user preferences. We then propose a dynamic graph pooling technique that adaptively reserves activated core preferences for predicting the user's next behavior.

To summarize, the contributions of this paper are as follows,

- We approach sequential recommendation from a new perspective by taking into consideration the implicit-signal behaviors and fast-changing preferences.
- We propose to aggregate implicit signals into explicit ones from user behaviors by designing graph neural network-based models on constructed item-item interest graphs. Then we design dynamic-pooling for filtering and reserving activated core preferences for recommendation.
- We conduct extensive experiments on two large-scale datasets collected from real-world applications. The experimental results show significant performance improvements compared with the state-of-the-art methods of sequential recommendation. Further studies also verify that our method can model long behavioral sequences effectively and efficiently.

2 PROBLEM FORMULATION

Here we provide a formal definition of sequential recommendation. Assume we have a set of items, denoted by $\mathcal{X}$, where $x \in \mathcal{X}$ denotes an item. The number of items is denoted as $|\mathcal{X}|$. Generally, a user has a sequential interaction sequence with items: $\{x_1, x_2, \ldots, x_n\}$, where n is the number of interactions and x_i is the i-th item that the user has interacted with. Sequential recommendation aims to predict the next item x_{n+1} that matches the user's preferences. The user's preferences can be inferred from chronological user-item implicit feedback. Based on the above definition, the task of sequential recommendation can be formulated as follows:

Input: The interaction history for each user $\{x_1, x_2, \ldots, x_n\}$.

Output: A recommendation model that estimates the probability that a user with interaction history $\{x_1, x_2, \ldots, x_n\}$ will interact with the target item x_t at the $(n+1)$-th step.

3 METHODOLOGY

Figure 1 illustrates our proposed SURGE model which is made up of the following four parts, which we will elaborate on one by one.

- **Interest Graph Construction.** By re-constructing loose item sequences as tight item-item interest graphs based on metric learning, we explicitly integrate and distinguish different types of preferences in long-term user behaviors.
- **Interest-fusion Graph Convolutional Layer.** The graph convolution propagation on the constructed interest graph dynamically fuses the user's interests, strengthening important behaviors, and weakening noise behaviors.
- **Interest-extraction Graph Pooling Layer.** Considering users' different preferences at different moments, a dynamic graph pooling operation is conducted to adaptively reserve dynamically-activated core preferences.
- **Prediction Layer.** After the pooled graphs are flattened into reduced sequences, we model the evolution of the enhanced interest signals and predict the next item that the user has high probability to interact with.

3.1 Interest Graph Construction

To integrate and distinguish different types of preferences in users' rich historical behaviors, we can convert loose item sequences into tight item-item interest graphs. The co-occurrence relationship between two items is a reasonable construction criterion, but the challenge is that the sparseness of the co-occurrence relationship is not enough to generate a connected graph for each user. In this section, we propose a novel way based on metric learning to automatically construct graph structures for each interaction sequence to explore the distribution of its interests.

3.1.1 ***Raw graph construction.*** This novel module attempts to construct an undirected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, A\}$ for each interaction sequence, where $\mathcal{E}$ is the set of graph edges to learn and $A \in \mathbb{R}^{n \times n}$ denotes the corresponding adjacency matrix. Each vertex $v \in \mathcal{V}$ with $|\mathcal{V}| = n$ corresponds to an interacted item (and the associated embedding vector is denoted as $\vec{h} \in \mathbb{R}^d$). We aim to learn the adjacency matrix A, where each edge $(i, j, A_{i,j}) \in \mathcal{E}$ indicates whether item i is related to item j.

By representing each user's interaction history as a graph, it is easier to distinguish his/her core and peripheral interests. The core interest node has a higher degree than the peripheral interest node due to connecting more similar interests, and the higher frequency of similar interests results in a denser and larger subgraph. In this way, a priori framework is constructed, that is, neighbor nodes are similar, and dense subgraphs are the core interests of users.

3.1.2 ***Node similarity metric learning.*** Since we need a priori graph in which neighbor nodes are similar, the graph learning problem can be transformed into node similarity metric learning, which will be jointly trained with the downstream recommendation task. This graph construction method is general, easy to implement, and able to perfectly cope with inductive learning (with new items during testing). Metric learning can be classified into kernel-based and attention-based methods [48]. Common options for kernel-based methods include cosine distance [30, 41], Euclidean distance [38, 43] and Mahalanobis distance [18, 34]. A good similarity metric function is supposed to be learnable to improve expressiveness and have acceptable complexity. To balance expressiveness and complexity, we adopt weighted cosine similarity [5, 42] as our metric function formulated as follows,

$$M_{ij} = \cos(\vec{\mathbf{w}} \odot \vec{h}_i, \vec{\mathbf{w}} \odot \vec{h}_j), \tag{1}$$

where $\odot$ denotes the Hadamard product, and $\vec{w}$ is a trainable weight vector to adaptively highlight different dimensions of the item embeddings $\vec{h}_i$ and $\vec{h}_j$. Note that the learned graph structure changes continuously with the update of item embeddings.

To increase the expressive power and stabilize the learning process, the similarity metric function can be extended to the multi-head metric [5, 25]. Specifically, we use ϕ (the number of heads) weight vectors to compute ϕ independent similarity matrices (each one representing one perspective) using the above similarity metric

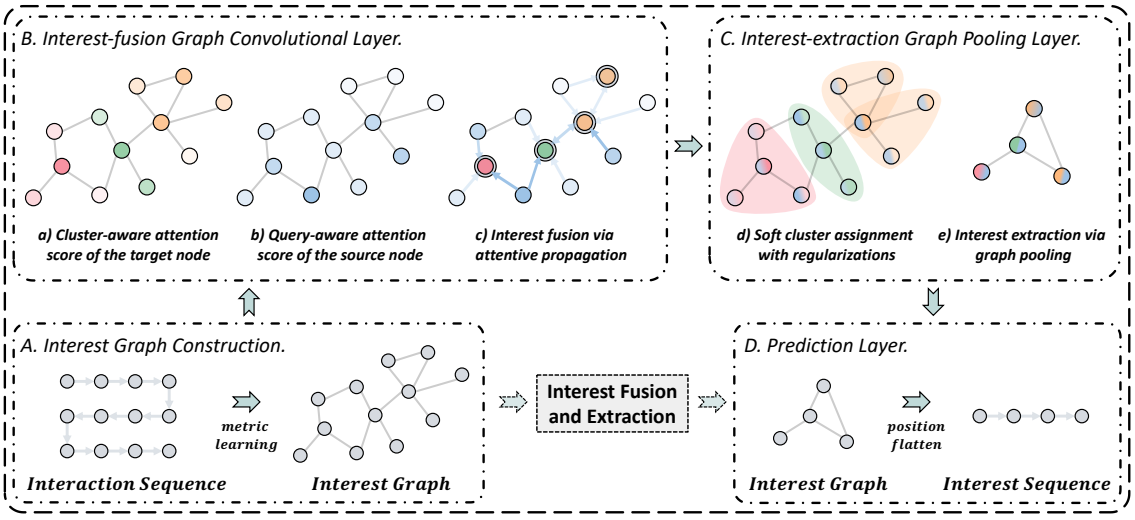

Figure 1: Illustration of the SURGE model. Each interaction sequence is re-constructed into an interest graph (A) based on metric learning, and interest fusion (B) and extraction (C) are dynamically performed on the graph. The currently activated core interest sequence (D) is obtained by flattening the pooled graph after interest fusing and extracting, which can be used for further modeling and prediction. Best viewed in color.

function and take their average as the final similarity:

$$M_{ij}^{\delta} = \cos(\vec{\mathbf{w}}_{\delta} \odot \vec{h}_i, \vec{\mathbf{w}}_{\delta} \odot \vec{h}_j), \quad M_{ij} = \frac{1}{\delta}\sum_{\delta=1}^{\phi} M_{ij}^{\delta}, \tag{2}$$

where M_{ij}^{δ} computes the similarity metric between the two item embeddings $\vec{h}_i$ and $\vec{h}_j$ for the δ-th head, and each head implicitly capture different perspective of semantics.

3.1.3 ***Graph sparsification via ε-sparseness.*** Typically, the adjacency matrix elements should be non-negative, but the cosine value M_{ij} calculated from the metric ranges between $[-1, 1]$. Simply normalizing it does not impose any constraints on the graph sparsity and can yield a fully connected adjacency matrix. This is computationally expensive and might introduce noise (i.e., unimportant edges), and it is not sparse enough that subsequent graph convolutions cannot focus on the most relevant aspects of the graph.

Therefore, we extract the symmetric sparse non-negative adjacency matrix A from M by considering only the node pair with the most vital connection. To make the hyperparameter of the extraction threshold insensitive and not destroy the graph's sparsity distribution, we adopt a *relative ranking strategy of the entire graph*. Specifically, we mask off (i.e., set to zero) those elements in M that are smaller than a non-negative threshold, which is obtained by ranking the metric value in M.

$$A_{ij} = \begin{cases} 1, & M_{ij} >= \mathbf{Rank}_{\varepsilon n^2}(M); \\ 0, & \text{otherwise}; \end{cases} \tag{3}$$

where $\mathbf{Rank}_{\varepsilon n^2}(M)$ returns the value of the εn^2-th largest value in the metric matrix M. n is the number of nodes and ε controls the overall sparsity of the generated graph.

It is different from the *absolute threshold strategy of the entire graph* [5] and the *relative ranking strategy of the node neighborhood* [4, 19]. The former sets an absolute threshold to remove smaller elements in the adjacency matrix. When the hyperparameters are set improperly, as the embedding is continuously updated, the metric value distribution will also change, and it may not be possible to generate a graph or generate a complete graph. The latter returns the indices of a fixed number of maximum values of each row in the adjacency matrix, which will make each node of the generated graph have the same degree. Forcing a uniform sparse distribution will make the downstream GCN unable to fully utilize the graph's dense or sparse structure information.

3.2 Interest-fusion Graph Convolutional Layer

As mentioned above, we have learnable interest graphs which separate diverse interests. The core interests and peripheral interests form large clusters and small clusters respectively, and different types of interests form different clusters. Furthermore, to gather weak signals to strong ones that can accurately reflect user preferences, we need to aggregate information in the constructed graph.

3.2.1 ***Interest fusion via graph attentive convolution.*** We propose a *cluster- and query-aware graph attentive convolutional layer* that can perceive the user's core interest (i.e., the item located in the cluster center) and the interest related to query interest (i.e., current target item) during information aggregation. The input is a node embedding matrix $\{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_n\}$, $\vec{h}_i \in \mathbb{R}^d$, where n is the number of nodes (i.e., the length of the user interaction sequence), and d is the dimension of embeddings in each node. The layer produces a new node embedding matrix $\{\vec{h}'_1, \vec{h}'_2, \ldots, \vec{h}'_n\}$, $\vec{h}'_i \in \mathbb{R}^{d'}$, as its output with potentially different dimension d'.

An alignment score E_{ij} is computed to map the importance of target node v_i on it's neighbor node v_j. Once obtained, the normalized attention coefficients are used to perform a weighted combination of the embeddings corresponding to them, to serve as the refined output embeddings for every node after applying a

residual connection and a nonlinearity function σ:

$$\vec{h}'_i = \sigma\left(\mathbf{W_a} \cdot \textbf{Aggregate}\left(E_{ij} * \vec{h}_j | j \in \mathcal{N}_i\right) + \vec{h}_i\right). \quad (4)$$

Note that aggregation function can be a function such as `Mean`, `Sum`, `Max`, `GRU`, etc. We use the simple sum function here and leave other functions for future exploration. To stabilize the attention mechanism's learning process, we employ multi-head attention similar to [25, 26]. Precisely, ϕ independent attention mechanisms execute the above transformation, and then their embeddings are concatenated as the following output representation:

$$\vec{h}'_i = \Big\Vert_{\delta=1}^{\phi} \sigma\left(\mathbf{W_a}^{\delta} \cdot \textbf{Aggregate}\left(E_{ij}^{\delta} * \vec{h}_j | j \in \mathcal{N}_i\right) + \vec{h}_i\right), \quad (5)$$

where $\|$ represents concatenation operation, E_{ij}^{δ} are normalized attention coefficients obtained by the δ-th attention head, and $\mathbf{W_a}^{\delta}$ is the corresponding linear transformation's weight matrix. It is worth noting that the final returned output $\vec{h}'$ will correspond to $\phi d'$ dimension embeddings (rather than d') for each node.

3.2.2 ***Cluster- and query-aware attention.*** To strengthen important signals and weaken noise signals when integrating interests, we propose a cluster and query-aware attention mechanism. We uses the attention coefficients to redistribute weights on edge information in the process of message passing. The attention mechanism considers the following two aspects.

Firstly, we assume that the target node v_i's neighborhood will form a cluster and regard the target node v_i in the graph as a medoid of a cluster $c(v_i)$. We define the k-hop neighborhood of the target node v_i as the receptive field of the cluster. The average value of all nodes' embedding in the cluster $\vec{h}_{i_c}$ represents the cluster's average information. To identify whether the target node is the center of the cluster, the target node embedding and its cluster embedding are used to calculate the following attention score,

$$\alpha_i = \textbf{Attention}_c(\mathbf{W_c}\vec{h}_i \,\|\, \vec{h}_{i_c} \,\|\, \mathbf{W_c}\vec{h}_i \odot \vec{h}_{i_c}), \quad (6)$$

where $\mathbf{W_c}$ is a transformation matrix, $\|$ is the concatenation operator and $\odot$ denotes the Hadamard product. In our experiments, the attention mechanism $\textbf{Attention}_c$ is a two-layers feedforward neural network with the *LeakyReLU* as activation function.

Secondly, in order to serve the downstream dynamic pooling method and learn the user interest's independent evolution for different target interests, the correlation between the source node embedding $\vec{h}_j$ and the target item embedding $\vec{h}_t$ should also be considered. If the source node is more correlated with the query item, its weight in the aggregation towards the target node will be more significant, and vice versa. Since only relevant behaviors can play a role in the final prediction, we only keep relevant information, and irrelevant information will be discarded during aggregation.

$$\beta_j = \textbf{Attention}_q(\mathbf{W_q}\vec{h}_j \,\|\, \vec{h}_t \,\|\, \mathbf{W_q}\vec{h}_j \odot \vec{h}_t), \quad (7)$$

where $\mathbf{W_q}$ is a transformation matrix, $\|$ is the concatenation operator and $\odot$ denotes the Hadamard product. In our experiments, the attention mechanism $\textbf{Attention}_q$ is a two-layers feedforward neural network applying the LeakyReLU nonlinearity.

We follow the additive attention mechanism [1] to consider the factors of cluster and query simultaneously. We sum the target node's cluster score and the source node's query score as the update weight of the source node j to the target node i. To make coefficients easily comparable across different nodes, we employed the softmax function to normalize them across all choices of j. The *attention coefficients* E_{ij} is computed as:

$$E_{ij} = \text{softmax}_j(\alpha_i + \beta_j) = \frac{\exp(\alpha_i + \beta_j)}{\sum_{k \in \mathcal{N}_i} \exp(\alpha_i + \beta_k)}, \quad (8)$$

where neighborhood $\mathcal{N}_i$ of node i includes node i itself. In the context of containing self-loop propagation (when j equals i), α_i controls how much information the target node can receive, and β_j controls how much information the source node can send.

3.3 Interest-extraction Graph Pooling Layer

The fusion of implicit interest signals to explicit interest signals is completed by performing information aggregation on the interest graph. In this section, we use the graph pooling method [17, 22, 37] to further extract the fused information. Similar to the downsampling of feature maps in Pooling in CNN, graph pooling aims to downsize the graph reasonably. Through the coarsening of the constructed graph structure, loose interest is transformed into tight interest and its distribution is maintained.

3.3.1 ***Interest extraction via graph pooling.*** To obtain the pooled graph, a cluster assignment matrix is necessary [22, 37]. Assuming that a soft cluster assignment matrix $S \in \mathbb{R}^{n \times m}$ exists, it can pool node information into cluster information. m is a pre-defined model hyperparameter that reflects the degree of pooling, where $m < n$. Given the node embeddings $\{\vec{h}'_1, \vec{h}'_2, \ldots, \vec{h}'_n\}$ and the node scores $\{\gamma_1, \gamma_2, \ldots, \gamma_n\}$ of the raw graph, the cluster embeddings and scores of the coarsened graph can be generated as follows,

$$\{\vec{h}^*_1, \vec{h}^*_2, \ldots, \vec{h}^*_m\} = S^T\{\vec{h}'_1, \vec{h}'_2, \ldots, \vec{h}'_n\}, \quad (9)$$

$$\{\gamma^*_1, \gamma^*_2, \ldots, \gamma^*_m\} = S^T\{\gamma_1, \gamma_2, \ldots, \gamma_n\}, \quad (10)$$

where γ_i obtained by applying softmax on β_i represents importance score of the i-th node. Each row of assignment matrix S corresponds to one of the n nodes, and each column corresponds to one of the m clusters. It provides a soft assignment of each node to the corresponding cluster. Above equations aggregate node embeddings and scores according to the cluster assignment S, thereby generating new embedding and score for each of the m clusters.

Next, we discuss how to learn differentiable soft clusters assignment S for nodes. We use the GNN architecture[37] to generate the assignment matrix. The probability matrix of the assignment mapping is obtained through standard message passing and the softmax function, based on the adjacency matrix and the node embedding.

$$S_{i:} = \text{softmax}\left(\mathbf{W_p} \cdot \textbf{Aggregate}\left(A_{ij} * \vec{h}'_j | j \in \mathcal{N}_i\right)\right), \quad (11)$$

where the output dimension of weight matrix $\mathbf{W_p}$ corresponds to the maximum number of clusters m. The softmax function is used to obtain the probability of the i-th node being divided into one of m clusters. It is worth noting that we can obtain the adjacency matrix A^* of the pooled graph by performing $S^T AS$, ensuring the connectivity between clusters. Then, the repetition of the above equations can perform multi-layer pooling to achieve hierarchical compression of interest.

3.3.2 ***Assignment regularization.*** However, it is difficult to train the cluster assignment matrix S using only the gradient signal from the downstream recommendation task. The non-convex optimization problem makes it easy to fall into the local optimum in the early training stage[37]. In addition, the relative position of each node embedding in $\{\vec{h}'_1, \vec{h}'_2, \ldots, \vec{h}'_n\}$ corresponds to the temporal order of the interaction. But in the pooled cluster embedding matrix $\{\vec{h}^*_1, \vec{h}^*_2, \ldots, \vec{h}^*_m\}$, the temporal order between the clusters reflecting the user's interest is difficult to be guaranteed. Therefore, we use three regularization terms to alleviate the above issue.

- **Same mapping regularization.** To make it easier for two nodes with greater connection strength to be mapped to the same cluster, the first regularization is used as follows,

$$L_{\mathrm{M}} = \|A, SS^T\|_F, \tag{12}$$

  where $\|\cdot\|_F$ denotes the Frobenius norm. Each element in adjacency matrix A represents the connection strength between two nodes, and each element in SS^T represents the probability that two nodes are divided to the same cluster.
- **Single affiliation regularization.** To clearly define the affiliation of each cluster, we make each row $S_{i:}$ in assignment matrix approach a one-hot vector by regularizing the entropy as follows,

$$L_{\mathrm{A}} = \frac{1}{n}\sum_{i=1}^{n} H\left(S_{i:}\right), \tag{13}$$

  where $H(\cdot)$ is the entropy function that can reduce the uncertainty of the mapping distribution. The optimal situation is that the i-th node is only mapped to one cluster, and the entropy $H(S_{i:})$ is 0 at this time.
- **Relative position regularization.** The temporal order of the user's interest before and after pooling needs to be maintained for downstream interest evolution modeling. However, the operation of swapping the index on the pooled cluster embedding matrix $\{\vec{h}^*_1, \vec{h}^*_2, \ldots, \vec{h}^*_m\}$ is not differentiable. Therefore, we design a position regularization to ensure the temporal order between clusters during pooling as follows,

$$L_{\mathrm{P}} = \|P_n S, P_m\|_2, \tag{14}$$

  where P_n is a position encoding vector $\{1, 2, ..., n\}$, and P_m is a position encoding vector $\{1, 2, ..., m\}$. Minimizing the L2 norm makes the position of the non-zero elements in S closer to the main diagonal elements. Intuitively, for the node with the front position in the original sequence, the position index of the cluster to which it is assigned tends to be in the front.

3.3.3 ***Graph readout.*** At this point, we have obtained a tightly coarsened graph $\mathcal{G}^*$ representing the user's stronger interest signal. At the same time, we perform a weighted readout on raw graph $\mathcal{G}$ to constrain each node's importance, which aggregates all node embeddings after the forward computation of the propagation layer to generate a graph-level representation $\vec{h}_g$:

$$\vec{h}_g = \mathbf{Readout}(\{\gamma_i * \vec{h}'_i, i \in \mathcal{G}\}), \tag{15}$$

where the weight is the score γ_i of each node before pooling, and the Readout function can be a function such as `Mean`, `Sum`, `Max`, etc. We use the simple sum function here to ensure permutation invariant and leave other functions for future exploration. We feed this graph-level representation into the final prediction layer to better extract each cluster's information in the pooling layer.

3.4 Prediction Layer

3.4.1 ***Interest evolution modeling.*** Under the joint influence of the external environment and internal cognition, the users' core interests are continually evolving. The user may become interested in various sports for a time and need books at another time. However, only using the readout operation mentioned above does not consider the evolution between core interests, which will undoubtedly cause the time order's bias. To supply the final representation of interest with more relative historical information, it is also necessary to consider the chronological relationship between interests.

Benefiting from the relative position regularization, the pooled cluster embedding matrix maintains the temporal order of the user's interest, which is equivalent to flattening the pooled graph into a reduced sequence with enhanced interest signals. Intuitively, we can use any known sequential recommendation method to model the concentrated interest sequence. For the sake of simplicity and to illustrate the effectiveness of the pooling method, we use a single sequential model to model the evolution of interest:

$$\vec{h}_s = \mathbf{AUGRU}(\{\vec{h}^*_1, \vec{h}^*_2, \ldots, \vec{h}^*_m\}). \tag{16}$$

As we know, GRU overcomes the vanishing gradients problem of RNN and is faster than LSTM [11]. Furthermore, to make better use of the importance weight γ^*_i of fused interest in the *interest extraction layer*, we adopt GRU with attentional update gate (AUGRU) [45] to combine attention mechanism and GRU seamlessly. AUGRU uses attention score γ^*_i to scale all dimensions of the update gate, which results that less related interest make fewer effects on the hidden state. It avoids the disturbance from interest drifting more effectively and pushes the relative interest to evolve smoothly.

3.4.2 ***Prediction.*** We take the graph-level representation of the *interest extraction layer* and evolution output of the *interest evolution layer* as the user's current interest, and concatenate them with the target item embedding. Given the concatenated dense representation vector, fully connected layers are used to automatically learn the combination of embeddings. We use two-layer feedforward neural network as the prediction function to estimate the probability of the user interacting with the item at the next moment, and all compared models in the experimental part will share this popular design [39, 45, 46],

$$\hat{y} = \mathbf{Predict}(\vec{h}_s \| \vec{h}_g \| \vec{h}_t \| \vec{h}_g \odot \vec{h}_t). \tag{17}$$

Following the CTR (click-through rate) prediction in the real-world industry [45, 46], we use the negative log-likelihood function as the loss function and share this setting with all compared models. The optimization process is to minimize the loss function together with a $L2$ regularization term to prevent over-fitting,

$$L = -\frac{1}{|O|}\sum_{o \in O}\left(y_o \log \hat{y}_o + (1 - y_o)\log(1 - \hat{y}_o)\right) + \lambda\|\Theta\|_2, \tag{18}$$

where O is the training set and $|O|$ is the number of training instances. Θ denotes the set of trainable parameters and λ controls the penalty strength. The label $y_o = 1$ indicates a positive instance and $y_o = 0$ indicates a negative instance. And $\hat{y}_o$ stands for the

Table 1: Statistics of the Datasets.

Dataset	Users	Items	Instances	Average Length
Taobao	36,915	64,138	1,471,155	39.85
Kuaishou	60,813	292,286	14,952,659	245.88

network's output after the softmax layer, representing the predicted probability of the next item being clicked. Besides, the three regularization terms in Section 3.3.2 are added to the final recommendation objective function to obtain the better performance and more interpretable cluster assignments.

4 EXPERIMENT

In this section, we conduct experiments on two real-world datasets for sequential recommendation to evaluate our proposed method, with the purpose of answering the following three questions.

- **RQ1:** How does the proposed method perform compared with state-of-the-art sequential recommenders?
- **RQ2:** Can the proposed method be able to handle sequences with various length effectively and efficiently?
- **RQ3:** What is the effect of different components in the method?

4.1 Experimental Settings

4.1.1 ***Dataset.*** We evaluate the recommendation performance on a public e-commerce dataset and an industrial short-video dataset. Table 1 summarizes the basic statistics of the two datasets. Average Length represents the average of users' history length, which indicates that the scale of the industry dataset we adopt is much larger than the public dataset.

- **Taobao**[*]**.** This dataset is widely used for recommendation research [20, 47], which is collected from the largest e-commerce platform in China. We use the click data from November 25 to December 3, 2017 and filter out users with less than 10 interactions. We use the first 7 days as training set, the 8th day as validation set, and the last day as test set.
- **Kuaishou**[†]**.** This is an industrial dataset collected from one of the largest short-video platforms in China. Users can upload short-videos and browse other users' short-videos. We downsample the logs from October 22 to October 28, 2020. User behaviors such as click, like, follow (subscribe) and forward are recorded in the dataset. Click data is used to conduct experiments, and the 10-core setting is also adopted to filter out inactive users and videos. Behaviors of the first 6 days are used to train recommendation models. Behaviors during the before 12 pm of the last day is used as validation set, and we keep the instances after 12 pm of the last day to evaluate the final recommendation performance.

4.1.2 ***Evaluation Metrics.*** To evaluate the performance of each model, we use two widely adopted accuracy metrics including AUC and GAUC [46], as well as two ranking metrics MRR and NDCG. They are defined as follows,

- **AUC** signifies the probability that the positive item sample's score is higher than the negative item sample's score, reflecting the classification model's ability to rank samples.
- **GAUC** performs a weighted average of each user's AUC, where the weight is his number of clicks. It eliminates the bias between users and evaluates model performance with a finer granularity.
- **MRR** is the mean reciprocal rank, which is the mean value of the inverse of the ranking of the first hit item.
- **NDCG@K** assigns higher scores to hits at higher positions in the top-K ranking list, which emphasizes that test items should be ranked as higher as possible. In our experiments, we set K to 2, a widely-used setting in existing works.

4.1.3 ***Baselines.*** To demonstrate the effectiveness of our SURGE model, we compare it with competitive sequential recommenders. The baselines are classified into two categories: non-sequential model that only captures user's static interest, and sequential models that consider dynamic interest patterns.

Non-sequential Models:

- **NCF** [9]: This method combines matrix factorization and multi-layer perceptrons to predict user interactions, and it is the state-of-the-art general recommender.
- **DIN** [46]: This method uses attention mechanism with the target item as the query vector. Representation of the user is obtained by aggregating the history interaction with the attention weights.
- **LightGCN** [8]: This is the state-of-the-art model which uses graph neural network to extract higher-order connectivity for the recommendation.

Sequential Models:

- **Caser** [24]: This method embeds a set of recent item sequences in time and latent space into an image feature and uses convolution filters to learn the its sequence patterns.
- **GRU4REC** [10]: This method uses GRU to model user session sequences and encode user interest into a final state.
- **DIEN** [45]: This method uses a two-layer GRU composed of interest extraction layer and interest evolution layer to model the user's behavior sequence.
- **SLi-Rec** [39]: This is the state-of-the-art method that jointly models long and short-term interests based on an attention framework and an improved time-aware LSTM.

It is worth noting that *session recommendation* is another recommendation task similar to sequential recommendation, which aims to predict the next item based on **only** the user's current session data without utilizing the long-term preference profile. Recently, graph-based models [21, 31, 33, 35] achieve successes on this task. The complex transitions between repeated behaviors in each session are modeled through the small item graphs for each user. However, users rarely produce repetitive behaviors over a long time, making relevant work impossible to apply to the task of sequential recommendation.

4.1.4 ***Hyper-parameter Settings.*** We implement all the models with the Microsoft Recommenders framework[‡] based on TensorFlow[§]. We use Adam [14] for optimization with the initial learning rate as 0.001. The batch size is set as 500 and embedding size is fixed to 40 for all models. Xavier initialization [7] is used here to initialize the parameters. All methods use a two-layer feedforward neural network with hidden sizes of [100, 64] for interaction estimation.

[*]https://tianchi.aliyun.com/dataset/dataDetail?dataId=649
[†]https://www.kuaishou.com/en
[‡]https://github.com/microsoft/recommenders
[§]https://www.tensorflow.org

Table 2: Performance comparisons (bold means p-value < 0.05, bold* means p-value < 0.01, and bold** means p-value < 0.001.)

Method	Taobao				Kuaishou			
	AUC	GAUC	MRR	NDCG@2	AUC	GAUC	MRR	NDCG@2
NCF	0.7128	0.7221	0.1446	0.0829	0.5559	0.5531	0.7734	0.8327
DIN	0.7637	0.8524	0.3091	0.2352	0.6160	0.7483	0.8863	0.9160
LightGCN	0.7483	0.7513	0.1669	0.1012	0.6403	0.6407	0.8175	0.8653
Caser	0.8312	0.8499	0.3508	0.2890	0.7795	0.8097	0.9100	0.9336
GRU4REC	0.8635	0.8680	0.3993	0.3422	0.8156	0.8333	0.9174	0.9391
DIEN	0.8477	0.8745	0.4011	0.3404	0.7037	0.7800	0.9030	0.9284
SLi-Rec	0.8664	0.8669	0.3617	0.2971	0.7978	0.8128	0.9075	0.9318
SURGE	**0.8906****	**0.8888**	**0.4228***	**0.3625****	**0.8525****	**0.8610****	**0.9316****	**0.9495***

The maximum length for user interaction sequences is 50 for the Taobao dataset and 250 for the Kuaishou dataset. We apply careful grid-search to find the best hyper-parameters. All regularization coefficients are searched in $[1e^{-7}, 1e^{-5}, 1e^{-3}]$. The pooling length of the user interaction sequence is searched in $[10, 20, 30, 40, 50]$ for Taobao dataset and $[50, 100, 150, 200, 250]$ for Kuaishou dataset.

4.2 Overall Performance (RQ1)

Table 2 illustrates the results on the two datasets. From the results, we have the following observations:

- **Our proposed method consistently achieves the best performance.** We can observe that our model SURGE significantly outperforms all baselines in terms of both classification and ranking metrics. Specifically, our model improves AUC by around 0.03 (p-value < 0.001) on Taobao dataset and 0.04 (p-value < 0.001) on Kuaishou dataset. The improvement is more obvious on the Kuaishou dataset with longer interaction history, which verifies that our method can handle long sequences more effectively and significantly reduces the difficulty of modeling user interests.
- **Sequential models are effective but have a short-term bottleneck.** Compared with NCF, DIN and LightGCN, the better performance of Caser, DIEN and GRU4Rec verifies the necessity of capturing sequential patterns for modeling user interests. On the Taobao dataset, RNN-based models (GRU4Rec and DIEN) with more powerful ability to capture sequential patterns outperformed the CNN-based model (Caser). The max pooling scheme in CNN that is commonly used in computer vision omits important position and recurrent signals when modeling long-range sequence data. But on the Kuaishou dataset, since RNNs tend to forget long-term interest when processing longer sequences, the performance of DIEN and GRU4Rec are in par with Caser in most metrics. This result indicates that even powerful recurrent neural networks have a short-term memory bottleneck. In addition, since long sequences tend to contain more noise, DIEN's performance on the two datasets is unstable compared to GRU4REC. This shows that the even though two-layer GRU structure is often more effective, the performance is more likely to be impacted by noise on datasets with long sequences, therefore justifying our motivation to summarize the sequences with metric learning.
- **Joint modeling long and short-term interests does not always add up to better performance.** SLi-Rec, which joint models long and short-term interests, is the best baseline on Taobao in terms of the AUC metric, but exhibits poor performance according to ranking metrics. In addition, on Kuaishou with longer interaction sequences, SLi-Rec's performance is worse than GRU4Rec for all metrics, even though GRU4REC does not explicitly model long and short-term interests. This indicates that although SLi-Rec utilizes two separate components to model users' long and short-term interests, it still fails to effectively integrate them into a single model, in particular for long sequences. Moreover, SLi-Rec leverages timestamp information to improve modeling long and short-term interests. However, our method shows better performance by compressing information with metric learning, without the need to explicitly model timestamp.

4.3 Study on Sequence Length and Efficiency Comparison (RQ2)

4.3.1 ***Study on Sequence Length.*** In real-world applications, a user may have very long interaction sequences. Long historical sequences often have more patterns that can reflect user interests, but the accompanying increased noise signals will mislead the modeling of real interests. Thus, whether to effectively model the user's long-term history is a significant issue for sequential recommendation. We study how SURGE improves the recommendation for those users with long behavior records. Specifically, we divide all users of the two datasets into five groups based on the length of the interaction history. For each group, we compare the performance of our method with the baseline methods and present the GAUC metric of the two datasets, as shown in Figure 3.

From the results, we can observe that all models are challenging to capture users' real interest when the sequence length is short due to data sparsity. As the length of the sequence increases and the difficulty of modeling decreases, most models' performance improves and reaches a peak. But as the length continues to increase, most models' performance will decline with the introduction of a large number of noise signals. Among them, DIN and DIEN declined most significantly. It is difficult for DIN to focus on the most critical parts in a long sequence. The item with the greatest attention may occur in the early part of the sequence, and it may be very different from the user's current interest. When the two-layer GRU structure in DIEN models user interests, the next GRU input depends on the previous GRU output, making it easier to be disturbed by the noise in long sequences. Due to the short-term bottleneck of a single GRU, GRU4REC will only focus on the recent history and

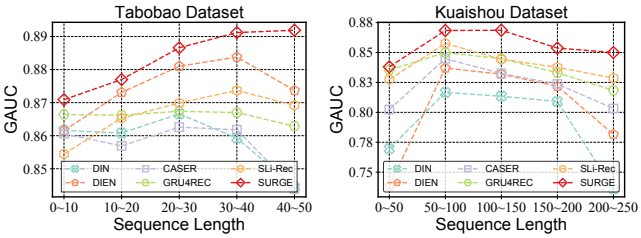

Figure 2: Performance breakdown by sequence lengths on the two datasets. Best viewed in color.

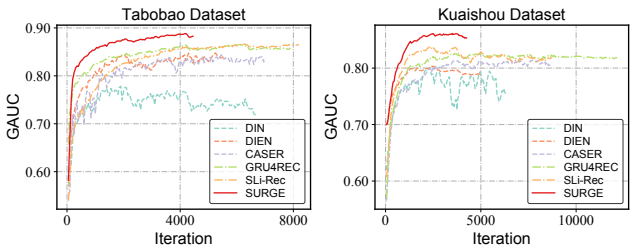

Figure 3: Test performance of the baselines by iterations on two datasets. Best viewed in color.

ignore the sequence length, and its performance in each length group is relatively stable. Although SLi-Rec jointly considers users' long-term and short-term interests, it still models for noise-filled sequences, so it is inevitable to suffer performance degradation on long sequences.

However, the performance gap between SURGE and other methods becomes larger as the sequence length increases. Furthermore, even in the user group with the longest historical sequence, SURGE still keeps the excellent performance of 0.8919 and 0.8502 on Taobao and Kuaishou datasets, respectively. Since the SURGE model merges implicit signals into explicit signals and filters out noise, it can achieve good performance for users with a long history. In summary, we conclude that the SURGE model we proposed can more effectively model users' long-term historical sequence.

4.3.2 ***Efficiency Comparison.*** For sequential recommendation systems, it is challenging to efficiently model user behavior history. The differences and diversity of items in the user's historical sequence lead to slow model convergence. Besides, long historical sequences often correspond to more complex calculations and more time-consuming training. To study whether SURGE can alleviate the issue, we visualize the training process of SURGE and baseline models and compare the convergence speed and training time of each model. Specifically, we plot the performance changes of the proposed method and the baseline methods on the validation set during the training process and reported the GAUC metric. We use early stop to detect whether the training is over, that is, if the GAUC on the validation set does not increase within five epochs, the training process will stop. For the two datasets' performance change curves, we use smoothing rates of 0.2 and 0.6 to smooth them to see the trend better.

The training process on the two datasets is shown in Figure 3. From the results, we can observe that DIN fails to focus on critical interests on long sequences, so it continually fluctuates on the kuaishou dataset and it is difficult to converge. Since GRU4REC is more likely to forget long-term interests, only the item embeddings at the end of the sequence will be updated in each training instance. Therefore, its training curve is steady and slow, and the continuous slight increase makes it hard to stop early. Because SLIREC specifically considers the long-term interests of users, it converges quickly on the kuaishou dataset, but it is the slowest method on the Taobao dataset with a shorter sequence.

Table 3 shows each model's training time on the two datasets. We can observe that, except for the non-sequential model of DIN on the kuaishou dataset, our method's efficiency improvement compared with all baselines is more than 20%. The reason is that SURGE performs a pooling operation on the sequence before feeding the embedded sequence into the recurrent neural network, which greatly reduces the number of recurrent steps. Besides, since most of the noise is filtered, the pooled sequence only contains the core interest, which will undoubtedly help speed up the model's convergence. Therefore, we concluded that the SURGE model we proposed can more efficiently model users' long-term historical sequence.

Table 3: Total training time until convergence of baselines on two real-world datasets, where m indicates minutes.

Dataset	DIN	Caser	GRU4REC	DIEN	SLi-Rec	SURGE
Taobao	22.65m	23.66m	26.78m	18.74m	27.82m	14.96m
Kuaishou	20.59m	120.26m	73.35m	28.47m	28.84m	22.86m

4.4 Ablation and Hyper-parameter Study (RQ3)

4.4.1 ***Effectiveness of interest fusion.*** We propose to perform message passing on the interest graph based on similarity to merge weak signals into strong signals. We now investigate whether this fusion design that strengthens core interests and activates target interests is necessary. To be specific, Specifically, we compare the no propagation, cluster-aware propagation, query-aware propagation, cluster- and query-aware propagation.

The results on the two datasets in Table 4 show the effectiveness of fusing weak signals into strong signals through graph convolution. The enhancement of core interest and the activation of target interest respectively bring further performance improvements.

4.4.2 ***Effectiveness of interest extraction.*** To evaluate the impact of interest extraction through pooling strategy on interest modeling. We compared no graph pooling, graph pooling without assignment regularization, graph pooling without weighted readout, and complete graph pooling.

The results are shown in Table 4. We can observe that interest extraction can help filter irrelevant noise so that the model focuses on the most critical part of modeling. Especially when the assignment regularization and the graph readout are injected into the model, the user's interest can be better compressed to improve the recommendation performance.

4.4.3 ***Design choices for interest evolution.*** Our framework is agnostic to the selection of the prediction layer after the pooling sequence. We can use any known sequential recommendation method to model the concentrated interest sequence. We compared

Table 4: Ablation study of the key designs

	Model	Taobao		Kuaishou	
		AUC	MRR	AUC	MRR
Interest Fusion	w/o Fusion	0.8307	0.0317	0.7149	0.9076
	w/o Query-aware	0.8720	0.3929	0.7641	0.9069
	w/o Cluster-aware	0.8764	0.3973	0.8213	0.9186
	w/ Fusion	0.8906	0.4228	0.8525	0.9316
Interest Extraction	w/o Extraction	0.8513	0.3605	0.8240	0.9182
	w/o Readout	0.8578	0.3720	0.8422	0.9257
	w/o Regularization	0.8815	0.0343	0.8487	0.9291
	w/ Extraction	0.8906	0.4228	0.8525	0.9316

the effects of using different prediction layers on the pooled sequence, including Attention (DIN), GRU (GRU4Rec), AUGRU (DIEN) and TIME4LSTM (SLi-Rec), and the results are shown in Figure 4.

The first observation is that the performance of sequential models other than DIN are less different, and AUGRU, which can utilize the cluster score in the interest extraction layer, is slightly better. The second observation is that modeling on the pooled sequence can bring benefits to all existing methods. It shows that our pooling strategy will significantly reduce the difficulty of modeling user interests and obtain better performance.

In conclusion, we conduct extensive experiments on two real-world datasets, which verifies that our proposed SURGE model outperform existing recommendation methods. Further studies demonstrate our model can effectively and efficiently alleviate the problem that long sequences are difficult to model.

5 RELATED WORK

Sequential Recommendation. Sequential recommendation is defined as leveraging historical sequential behaviors for predicting next behavior. The earliest work, FPMC [23], used Markov chain to model the transition in behavior sequences. For stronger ability to learn the complex bahaviors, deep learning-based methods [10, 13, 24, 45, 46] were proposed, including recurrent neural network-based ones [10, 45] and attention network-based ones [13, 46]. These works pay more attention to users' recent behaviors and largely ignore the users' long-term behaviors. Considering this point, there are some recent works [39, 44] combining sequential recommendation model and a normal recommendation model, such as matrix factorization [16], to capture long and short-term interest.

However, roughly dividing user interest into long-term part and short-term part is not reasonable. Compared with these works, in our work, we approach the problem of sequential recommendation from a new perspective: we argue the sequential behaviors reflect weak preference signals, and some part of user preferences may be deactivated at a given time point.

Graph Neural Networks for Recommendation. In recommendation scenarios, the input data can be represented in a graph structure. Recently, with strong ability of learning from graph-structure data, graph neural networks [15, 26] have become popular means for recommender systems. PinSage [36] applied GCN to pin-board graphs, which was the first work for applying GCN into industrial recommender systems. The standard GCN [15] was adopted

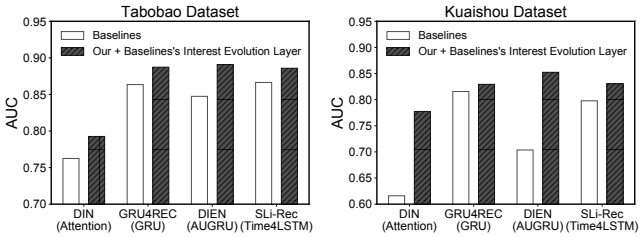

Figure 4: Performance comparison of the proposed method using different interest evolution layers.

to factorizing user-item rating matrices into user and item embedding matrices for recommendation [2] and Wang *et al.* [29] further proposed the general solution for implicit recommendation task. GCN-based methods have achieved the state-of-the-art performance in other recommendation problems, such as social recommendation [6, 32, 40], knowledge graph-based recommendation [27, 28], multi-behavior recommendation [12], bundle recommendation [3], etc. There are some works [33] utilizing graph neural networks for session-based recommendation, a problem similar with sequential recommendation. In session-based recommendation, one of most important goals is to capture the seasonal repetitive behaviors, making it has big difference with sequential recommendation, which is a more general and important problem in the research area.

Differ from aforementioned works, we take advantage of graph convolutional propagation to fuse the weak preference signals to strong ones and propose graph pooling to extract the dynamically-activated core preference in the long behavior sequences.

6 CONCLUSIONS AND FUTURE WORK

In this work, we studies the task of sequential recommender systems. We propose a graph-based solution that re-constructs loose item sequences into tight item-item interest graphs. The model utilizes graph neural network's powerful ability to dynamically fuse and extract users' activated core interests from noisy user behavior sequences. Extensive experiments on both public and proprietary industrial datasets demonstrate the effectiveness of our proposal. Further studies on sequence length confirm that our method can model long behavioral sequences effectively and efficiently.

As for future work, we plan to conduct A/B tests on the online system to further evaluate our proposed solution's recommendation performance. We also plan to consider using multiple types of behaviors, such as clicks and favorites, to explore fine-grained multiple interactions from noisy historical sequences.

ACKNOWLEDGMENTS

This work was supported in part by The National Key Research and Development Program of China under grant 2020AAA0106000, the National Natural Science Foundation of China under U1936217, 61971267, 61972223, 61941117, 61861136003, Beijing Natural Science Foundation under L182038, Beijing National Research Center for Information Science and Technology under 20031887521, and research fund of Tsinghua University - Tencent Joint Laboratory for Internet Innovation Technology.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

[2] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. In *KDD workshop*.

[3] Jianxin Chang, Chen Gao, Xiangnan He, Depeng Jin, and Yong Li. 2020. Bundle recommendation with graph convolutional networks. In *SIGIR*. 1673–1676.

[4] Yu Chen, Lingfei Wu, and Mohammed J Zaki. 2019. Reinforcement learning based graph-to-sequence model for natural question generation. In *ICLR*.

[5] Yu Chen, Lingfei Wu, and Mohammed J. Zaki. 2020. Iterative Deep Graph Learning for Graph Neural Networks: Better and Robust Node Embeddings. In *NeurIPS*.

[6] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *WWW*. 417–426.

[7] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*. 249–256.

[8] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR*.

[9] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*. 173–182.

[10] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. In *ICLR*.

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

[12] Bowen Jin, Chen Gao, Xiangnan He, Depeng Jin, and Yong Li. 2020. Multi-behavior recommendation with graph convolutional networks. In *SIGIR*.

[13] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *ICDM*. 197–206.

[14] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

[15] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[16] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009).

[17] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-Attention Graph Pooling. In *ICML*.

[18] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. 2018. Adaptive Graph Convolutional Neural Networks. In *AAAI*.

[19] Will Norcliffe-Brown, Stathis Vafeias, and Sarah Parisot. 2018. Learning Conditioned Graph Structures for Interpretable Visual Question Answering. In *NeurIPS*.

[20] Qi Pi, Weijie Bian, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Practice on long sequential user behavior modeling for click-through rate prediction. In *KDD*. 2671–2679.

[21] Ruihong Qiu, Jingjing Li, Zi Huang, and Hongzhi Yin. 2019. Rethinking the item order in session-based recommendation with graph neural networks. In *CIKM*. 579–588.

[22] Ekagra Ranjan, Soumya Sanyal, and Partha Talukdar. 2020. Asap: Adaptive structure aware pooling for learning hierarchical graph representations. In *AAAI*. 5470–5477.

[23] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *WWW*. 811–820.

[24] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *WWW*. 565–573.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*. 5998–6008.

[26] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[27] Hongwei Wang, Miao Zhao, Xing Xie, Wenjie Li, and Minyi Guo. 2019. Knowledge graph convolutional networks for recommender systems. In *WWW*. 3307–3313.

[28] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. 2019. KGAT: Knowledge Graph Attention Network for Recommendation. In *KDD*. 950–958.

[29] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *SIGIR*. 165–174.

[30] Xiao Wang, Meiqi Zhu, Deyu Bo, Peng Cui, Chuan Shi, and Jian Pei. 2020. AM-GCN: Adaptive Multi-channel Graph Convolutional Networks. In *KDD*.

[31] Ziyang Wang, Wei Wei, Gao Cong, Xiao-Li Li, Xian-Ling Mao, and Minghui Qiu. 2020. Global context enhanced graph neural networks for session-based recommendation. In *SIGIR*. 169–178.

[32] Le Wu, Peijie Sun, Yanjie Fu, Richang Hong, Xiting Wang, and Meng Wang. 2019. A neural influence diffusion model for social recommendation. In *SIGIR*. 235–244.

[33] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. 2019. Session-based recommendation with graph neural networks. In *AAAI*. 346–353.

[34] Xuan Wu, Lingxiao Zhao, and Leman Akoglu. 2018. A Quest for Structure: Jointly Learning the Graph Structure and Semi-Supervised Classification. In *CIKM*.

[35] Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Victor S Sheng, Jiajie Xu, Fuzhen Zhuang, Junhua Fang, and Xiaofang Zhou. 2019. Graph Contextualized Self-Attention Network for Session-based Recommendation. In *IJCAI*. 3940–3946.

[36] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*. 974–983.

[37] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *NeurIPS*.

[38] Donghan Yu, Ruohong Zhang, Zhengbao Jiang, Yuexin Wu, and Yiming Yang. 2020. Graph-Revised Convolutional Network. In *ECML PKDD*.

[39] Zeping Yu, Jianxun Lian, Ahmad Mahmoody, Gongshen Liu, and Xing Xie. 2019. Adaptive User Modeling with Long and Short-Term Preferences for Personalized Recommendation. In *IJCAI*. 4213–4219.

[40] Jun Zhang, Chen Gao, Depeng Jin, and Yong Li. 2021. Group-Buying Recommendation for Social E-Commerce. In *ICDE*.

[41] Xiang Zhang and Marinka Zitnik. 2020. GNNGuard: Defending Graph Neural Networks against Adversarial Attacks. In *NeurIPS*.

[42] Jianan Zhao, Xiao Wang, Chuan Shi, Binbin Hu, Guojie Song, and Yanfang Ye. 2021. Heterogeneous Graph Structure Learning for Graph Neural Networks. In *AAAI*.

[43] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. 2021. Data Augmentation for Graph Neural Networks. In *AAAI*.

[44] Wei Zhao, Benyou Wang, Jianbo Ye, Yongqiang Gao, Min Yang, and Xiaojun Chen. 2018. PLASTIC: Prioritize Long and Short-term Information in Top-n Recommendation using Adversarial Training. In *IJCAI*. 3676–3682.

[45] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep interest evolution network for click-through rate prediction. In *AAAI*. 5941–5948.

[46] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *KDD*. 1059–1068.

[47] Han Zhu, Daqing Chang, Ziru Xu, Pengye Zhang, Xiang Li, Jie He, Han Li, Jian Xu, and Kun Gai. 2019. Joint optimization of tree-based index and deep model for recommender systems. In *NeurIPS*. 3971–3980.

[48] Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Qiang Liu, Shu Wu, and Liang Wang. 2021. Deep Graph Structure Learning for Robust Representations: A Survey. In *IJCAI*.