Handling Information Loss of Graph Neural Networks for Session-based Recommendation

Tianwen Chen
tchenaj@cse.ust.hk
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong

Raymond Chi-Wing Wong
raywong@cse.ust.hk
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong

ABSTRACT

Recently, graph neural networks (GNNs) have gained increasing popularity due to their convincing performance in various applications. Many previous studies also attempted to apply GNNs to session-based recommendation and obtained promising results. However, we spot that there are two information loss problems in these GNN-based methods for session-based recommendation, namely the *lossy session encoding* problem and the *ineffective long-range dependency capturing* problem. The first problem is the lossy session encoding problem. Some sequential information about item transitions is ignored because of the lossy encoding from sessions to graphs and the permutation-invariant aggregation during message passing. The second problem is the ineffective long-range dependency capturing problem. Some long-range dependencies within sessions cannot be captured due to the limited number of layers. To solve the first problem, we propose a lossless encoding scheme and an edge-order preserving aggregation layer based on GRU that is dedicatedly designed to process the losslessly encoded graphs. To solve the second problem, we propose a shortcut graph attention layer that effectively captures long-range dependencies by propagating information along shortcut connections. By combining the two kinds of layers, we are able to build a model that does not have the information loss problems and outperforms the state-of-the-art models on three public datasets.

CCS CONCEPTS

• **Information systems** → **Recommender systems**.

KEYWORDS

session-based recommendation, recommender system, information retrieval, graph neural networks

ACM Reference Format:
Tianwen Chen and Raymond Chi-Wing Wong. 2020. Handling Information Loss of Graph Neural Networks for Session-based Recommendation. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining USB Stick (KDD '20), August 23–27, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3394486.3403170

KDD '20, August 23–27, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403170

1 INTRODUCTION

In many online services, users' actions are naturally ordered by time. To predict users' future actions, a next-item recommender system learns users' preferences by mining sequential patterns from their historical actions. *Session-based recommendation* is a special case of next-item recommendation. Unlike a general next-item recommender system which predicts the next action using a fixed number of previous actions, a session-based recommender system groups user actions into disjoint sessions and only uses the previous actions in the active session to make recommendations. Here, a session is a sequence of items in close temporal proximity. The idea of session-based recommendation comes from the observation that the intra-session dependencies have a larger impact on the next item than the inter-session dependencies [3]. Specifically, user actions in the same session usually share a common objective such as buying some phone accessories, while user actions in different sessions have a relatively weak correlation. A user may buy phone accessories in one session but buy things that have little relation with phone accessories such as clothes in another session. Therefore, a general next-item recommender system may suffer from the problem of combining uncorrelated sessions and extracting incomplete sessions. A session-based recommender system does not have such a problem, and thus it can make more accurate recommendations and are deployed in many online services.

Due to the highly practical value, session-based recommendation attracted researchers' great attention and many effective methods were developed in the past few years. Most of the methods proposed earlier are based on Markov chains or recurrent neural networks (RNNs). Recently, GNNs have become increasingly popular and achieved state-of-the-art performance in many tasks. There are also some attempts to apply GNNs to session-based recommendation [16, 23–25]. Although these GNN-based methods obtained exciting results and offered a new and promising direction for session-based recommendation, we observe that there are two information loss problems in these methods.

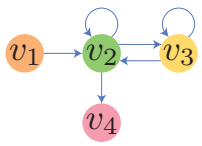

Figure 1: Two different sessions $[v_1, v_2, v_3, v_3, v_2, v_2, v_4]$ and $[v_1, v_2, v_2, v_3, v_3, v_2, v_4]$ are converted to the same graph.

The first information loss problem in the existing GNN-based methods [16, 23–25] is called the *lossy session encoding* problem. It is due to their lossy encoding schemes that convert sessions to

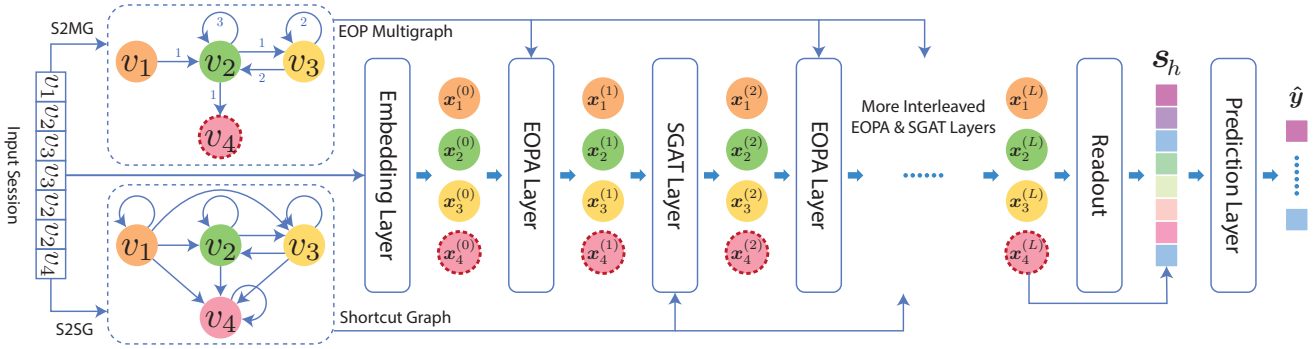

Figure 2: The overview of the proposed model LESSR. Given a session, an edge-order preserving (EOP) multigraph and a shortcut graph is computed. The initial node representations $x_i^{(0)}$ are the item embeddings. The graphs and the node representations are passed as input to multiple interleaved EOPA and SGAT layers. Each layer outputs the new node representations. The readout layer computes a graph-level representation, which is combined with the recent interests to form the session embedding s_h. Finally, the prediction layer computes the probability distribution of the next item $\hat{y}$.

graphs. To process sessions using a GNN, the sessions need to be converted to graphs first. In these methods, each session is converted to a directed graph whose nodes are the unique items in the session and the edges are the transitions between items. The edges can be either weighted or unweighted. For example, a session $[v_1, v_2, v_3, v_3, v_2, v_2, v_4]$ is converted to a graph as shown in Figure 1. However, such conversion is a lossy operation because it is not a one-to-one mapping. A different session $[v_1, v_2, v_2, v_3, v_3, v_2, v_4]$ is also converted to the same graph, and thus we cannot reconstruct the original session given the graph. Although in a particular dataset, the two sessions may produce the same next item, there may also exist a dataset in which the two sessions produce different next items. In the latter case, it is not possible for these GNN models to make correct recommendations for both sessions. Therefore, these models have a limitation in their modeling capacity.

The second information loss problem is called the *ineffective long-range dependency capturing* problem where these GNN-based methods cannot effectively capture all long-range dependencies. In each layer of a GNN model, information carried by nodes are propagated along the edges for one step[1], so each layer can capture only 1-hop relation. By stacking multiple layers, the GNN model can capture up to L-hop relation where L is equal to the number of layers. Since stacking more layers do not necessarily increase performance due to the overfitting and over-smoothing problems [12, 26], the optimal number of layers for these GNN models is usually no larger than 3 [16, 23–25]. Therefore, the models can only capture up to 3-hop relation. However, in real world applications, the session length can easily be larger than 3. Thus, it is very likely that there are some important sequential patterns that are longer than 3. Nevertheless, due to the limitation of the network structure, these GNN-based model cannot capture such information.

To solve the above problems, we propose a novel GNN model called LESSR (Lossless Edge-order preserving aggregation and Shortcut graph attention for Session-based Recommendation)[2]. Figure 2 illustrates the workflow of LESSR. A given input session is first converted to a losslessly encoded graph called edge-order preserving (EOP) multigraph and a shortcut graph where the EOP multigraph could address the lossy session encoding problem and the shortcut graph could address the ineffective long-range dependency capturing problem. Then, the graphs along with the item embeddings are passed to multiple *edge-order preserving aggregation (EOPA)* and *shortcut graph attention (SGAT)* layers to generate latent features of all nodes. The EOPA layers capture local context information using the EOP multigraph and the SGAT layers effectively capture long-range dependencies using the shortcut graph. Then, a readout function with attention is applied to generate a graph-level embedding from all node embeddings. Finally, we combine the graph-level embedding with users' recent interests to make recommendations. We summarize our contributions as follows:

- We are the first to identify two information loss problems of the GNN-based methods for session-based recommendation, including the lossy session encoding problem and the ineffective long-range dependency capturing problem.
- To solve the lossy session encoding problem, we propose a lossless encoding scheme that transforms sessions into directed multigraphs, and an EOPA layer that aggregates propagated information using GRU.
- To solve the ineffective long-range dependency capturing problem, we propose a SGAT layer which effectively propagates information along shortcut connections using the attention mechanism.
- By combining the two solutions, we build a GNN model that does not have the information loss problems and outperforms the existing methods in three public datasets.

2 RELATED WORK

In this section, we review the related work of session-based recommendation.

Inspired by the prevalence of neighborhood models in traditional recommendation tasks where user identifiers are available, the recommender systems proposed in the early research work

[1]Some models such as [23] can propagate information for multiple steps in each layer, but they could still just capture up to k-hop relation where k is proportional to the total number of layers.

[2]The implementation is available at https://github.com/twchen/lessr

on session-based recommendation were mostly based on nearest-neighbors [4, 5, 15]. These methods need a similarity function to measure the similarity between items or sessions. Davidson et al. [4] proposed a method that computes item similarity from items' co-occurrence patterns and recommends the items that are most likely to co-occur with any item in the active session. Park et al. [15] proposed a model that first converts each session to a vector and then measures the cosine similarity between session vectors. Based on this work, Dias and Fonseca [5] proposed to transform the sparse session vectors to dense vectors using clustering methods before computing the cosine similarity. Though simple and efficient, the neighborhood-based methods suffer from the sparsity problem, and do not consider the relative order of items within sessions.

To better capture sequential properties, Markov chain-based methods can be adopted. The simplest Markov chain-based method computes the transition matrix heuristically using the transition frequencies in the training set [19]. However, this method cannot handle unobserved transitions. One solution is the FPMC method proposed in [18] which factorizes personalized transition matrices using a tensor decomposition technique. Another solution is called Latent Markov Embedding [1] which embeds items in a Euclidean space and estimates the transition probabilities between items by the Euclidean distance of their embeddings. Since the state size quickly becomes unmanageable when more previous items are considered, most Markov chain-based methods build the transition matrix using only first-order transitions, which makes them unable to capture more complex higher-order sequential patterns.

Recurrent neural networks (RNNs) are a natural solution to the above limitation of Markov chain-based methods due to their powerful sequence modeling capability. GRU4Rec [8] was the first RNN-based method for session-based recommendation which simply stacks multiple GRU layers. Inspired by the success of the attention mechanism in computer vision and natural language processing, Li et al. [11] employed a hybrid encoder with attention to model users' sequential behavior and main purpose in the active session, which was proved to be an effective approach for learning session representations. Following the work, almost all the subsequent RNN-based methods incorporate the attention mechanism [2, 17, 20].

Convolutional neural networks (CNNs) are also powerful sequence modeling tools. Tang and Wang [21] proposed a CNN-based method that embeds a sequence of items into a 2-dimensional latent matrix and performs both horizontal and vertical convolution on the matrix to extract the sequence representation. Yuan et al. [27] proposed to use dilated convolutional layers to efficiently increase the receptive fields without relying on the lossy pooling operation, resulting in a model that is more capable of capturing long-range dependencies compared with [8, 21].

In the last few years, graph neural networks (GNNs) have been increasingly popular and have achieved state-of-the-art performance in many tasks. There is also some effort to apply GNNs to session-based recommendation. SR-GNN [23] first encodes a session into an unweighted directed graph whose edges represent item transitions in the session and then propagates information between nodes along both directions of the edges using gated GNNs (GGNNs) [13]. Based on this work, Xu et al. [25] proposed a method that uses a GGNN to extract local context information and a self-attention network (SAN) to capture global dependencies between distant positions. FGNN [16] converts a session into a weighted directed graph where the edge weights are the counts of item transitions. An adapted multi-layered graph attention network (GAT) [22] is used to extract item features and a modified Set2Set pooling operator is applied to generate session representations. These GNN-based methods have shown a new and promising direction for session-based recommendation. However, as we will discuss in Section 4, these methods operate on a lossy encoding of sessions as graphs and cannot effectively capture long-range dependencies.

3 PRELIMINARIES

In this section, we introduce some preliminary knowledge about graph neural networks (GNNs).

GNNs are neural networks that directly operate on graph data. They are used for learning tasks such as graph classification, node classification and link prediction problems. In this paper, we just focus on the graph classification problem because session-based recommendation can be formulated as such a problem.

Let $G = (V, E)$ be a given graph, where V and E are the sets of nodes and edges, respectively. Each node $i \in V$ is associated with a node feature vector $\boldsymbol{x}_i$, which is passed to the first layer of GNN as the initial node representation. Most GNNs can be understood from the perspective of message passing. In each layer of a GNN, the node representations are updated by passing messages along the edges. The process can be formulated as follows:

$$\boldsymbol{x}_i^{(l+1)} = f_{upd}^{(l)}\left(\boldsymbol{x}_i^{(l)}, \mathrm{agg}_i^{(l)}\right) \tag{1}$$

$$\mathrm{agg}_i^{(l)} = f_{agg}^{(l)}\left(\left\{f_{msg}^{(l)}\left(\boldsymbol{x}_i^{(l)}, \boldsymbol{x}_j^{(l)}\right) : (j, i) \in E_{in}(i)\right\}\right) \tag{2}$$

where $\boldsymbol{x}_i^{(l)}$ is the representation of node i at layer l and $E_{in}(i)$ is the set of incoming edges of node i. The message function $f_{msg}^{(l)}$ computes the message to be propagated from a neighboring node to the target node. The aggregation function $f_{agg}^{(l)}$ aggregates the information passed to the target node. The update function $f_{upd}^{(l)}$ computes the new node representation from the original node representation and the aggregated information.

Let L be the number of layers in the GNN. After L steps of message passing in L layers, the final node representations capture the information about the graph structure and the features of nodes within a L-hop community. For the graph classification task, a readout function f_{out} is used to generate a graph level representation $\boldsymbol{h}_G$ by aggregating the representations of all nodes in the final layer:

$$\boldsymbol{h}_G = f_{out}(\{\boldsymbol{x}_i^{(L)} : i \in V\}) \tag{3}$$

4 METHODOLOGY

In this section, we first give a formal definition of session-based recommendation (Section 4.1), and then describe the proposed method which involves two major components. The first component is the module that converts each input session to an edge-order preserving (EOP) multigraph and a shortcut graph, since our GNN model requires two types of graphs as input (Section 4.2). The second component is the proposed GNN model LESSR (Section 4.3).

4.1 Problem Definition

Session-based recommendation is an instance of next-item recommendation. Its objective is to recommend the items that the user is most likely to click next given a sequence of items that are already clicked in the active session. Formally, let $I = \{v_1, v_2, \ldots, v_{|I|}\}$ denote the universal set of items. A session $s_i = [s_{i,1}, s_{i,2}, \ldots, s_{i,l_i}]$ is a sequence of items ordered by time, where $s_{i,t} \in I$ is the item at time step t and l_i is the length of s_i. The objective of the model is to predict the next item s_{i,l_i+1}. A typical session-based recommender system generates a probability distribution of the next item, i.e., $p(s_{i,l_i+1}|s_i)$. The items with the top-K probabilities are in the candidate set for recommendations.

Following [11, 16, 17, 21, 23, 25], we do not consider additional context information such as user IDs and item attributes in this paper. The item IDs are embedded in a d-dimensional space and served as the initial item features in the model. This is a common practice in the literature of session-based recommendation. However, it is easy to adapt our method to take additional context information into consideration. For example, the user ID embedding can be served as a graph level attribute and can be appended to item ID embeddings in each layer [24]. The item features can be combined with or replace the item ID embeddings [9].

4.2 Converting Sessions to Graphs

To process sessions using a GNN, sessions must be converted to graphs first. In this subsection, we first introduce a method called *S2MG* that converts sessions to EOP multigraphs, and then another method called *S2SG* that converts sessions to shortcut graphs.

4.2.1 S2MG: Session to EOP Multigraph. In the literature of session-based recommendation [16, 23], there are two common methods to convert sessions into graphs. The first method proposed in [23] converts a session to an unweighted directed graph $G = (V, E)$ where the node set V consists of the unique items in the session, and the edge set E contains an edge (u, v) if $u = s_{i,t}$ and $v = s_{i,t+1}$ for some $1 \le t < l_i$. The second method was proposed in [16]. Unlike the first method, the edges of the converted graph are weighted, where the weight of an edge (u, v) is the number of times that the transition $u \to v$ appears in the session. In the following, we will refer the first and second methods as S2G and S2WG, respectively.

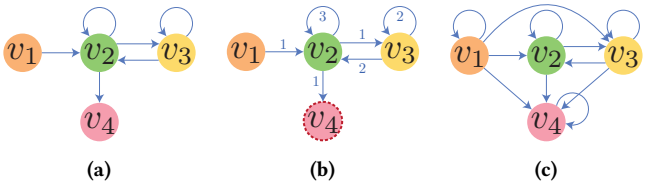

Figure 3: The weighted graph (a), EOP multigraph (b) and shortcut graph (c) of session $[v_1, v_2, v_3, v_3, v_2, v_2, v_4]$ converted by S2WG, S2MG and S2SG, respectively. Note that the weights in (a) are omitted because they are all 1.

We claim that both S2G and S2WG are lossy conversion methods because it is not always possible to reconstruct the original session given the converted graph. To prove this, we just need to prove S2WG is lossy because it captures more information than S2G, i.e., the number of occurrences of item transitions. Therefore, "S2WG is lossy" implies "S2G is lossy". To see why S2WG is lossy, we give a counter-example as follows. Two different sessions $s_1 = [v_1, v_2, v_3, v_3, v_2, v_2, v_4]$ and $s_2 = [v_1, v_2, v_2, v_3, v_3, v_2, v_4]$ are converted to the same graph as shown in Figure 3(a) by S2WG. Note that we omit the edge weights because they are all 1. Therefore, given the converted graph in Figure 3(a), it is not clear which of s_1 and s_2 is the original session.

Lossy conversion could be problematic because the information ignored may be important to determine the next item. We should let the model automatically learn to decide what information can be ignored instead of "blindly" making the decision using a lossy conversion method. Otherwise, the model is not flexible enough to fit complex datasets since its modeling capacity is limited by the lossy conversion method.

To handle this problem, we propose a method called *S2MG (session to EOP multigraph)* which converts a session to a directed multigraph that preserves the edge order. For each transition $u \to v$ in the original session, we create an edge (u, v). The graph is a multigraph because if there are multiple transitions from u to v, we will create multiple edges from u to v. Then, for each node v, the edges in $E_{in}(v)$ can be ordered by the time of their occurrences in the session. We record the order by giving each edge in $E_{in}(v)$ an integer attribute which indicates its relative order among the edges in $E_{in}(v)$. The edge occurs first in $E_{in}(v)$ is given 1, the next edge in $E_{in}(v)$ is given 2 and so on. For example, session $s_1 = [v_1, v_2, v_3, v_3, v_2, v_2, v_4]$ is converted to the graph in Figure 3(b), denoted by MG_{s_1}. In order for the conversion to be truly lossless, we also label the last item, e.g., node v_4 for session s_1, which is indicated by a dotted circle. We call this resulting graph as the *edge-order preserving (EOP) multigraph* of the given session.

Now, we prove that S2MG is a lossless conversion method from sessions to graphs by showing how to reconstruct the original session given an EOP multigraph (i.e., a graph converted using S2MG). The idea is to recover the items in the reverse order of their occurrences in the session. Let's use MG_{s_1} as an example. The last item is the labelled node v_4. Since we know the last item, and we know the order of incoming edges of v_4, we can determine the last edge is (v_2, v_4). Then, the second last item is simply the source node of the last edge, i.e., v_2. We can do this iteratively and determine the third last item and so on. In this way, we can recover session s_1 from graph MG_{s_1}. The same procedure can be applied to reconstruct the original session given any graph. Therefore, S2MG is a lossless conversion method from sessions to graphs.

4.2.2 S2SG: Session to Shortcut Graph. To handle the ineffective long-range dependency problem in existing GNN-based models for session-based recommendation, we propose the shortcut graph attention (SGAT) layer to be described in Section 4.3.2. The SGAT layer requires an input graph that is different from the above EOP multigraph. The input graph is converted from the input session using the following method called *S2SG (session to shortcut graph)*.

Given a session s_i, we create a graph whose nodes are the unique items in s_i. For each ordered pair of nodes (u, v), we create an edge from u to v if and only if there exists a pair (s_{i,t_1}, s_{i,t_2}), such

that $s_{i,t_1} = u$, $s_{i,t_2} = v$ and $t_1 < t_2$. The graph is called a shortcut graph because it connects items without going through intermediate items. We also add self-loops to the graph so that later when the SGAT layer performs message passing, the update function and aggregate function can be combined, which is a common practice in GAT models [16, 22]. Therefore, the shortcut graph of $s_1 = [v_1, v_2, v_3, v_3, v_2, v_2, v_4]$ is the graph shown in Figure 3(c).

In Section 4.3.2, we will show how the SGAT layer could solve the ineffective long-range dependency problem by performing message passing on the shortcut graph.

4.3 Our GNN Model: LESSR

In this subsection, we describe the details of our GNN model, LESSR, whose overview is shown in Figure 2. Firstly, we introduce the EOPA layer in Section 4.3.1 and the SGAT layer in Section 4.3.2. Next, we describe how to stack these two types of layers in Section 4.3.3. Next, we show how to obtain the session embedding in Section 4.3.4. Lastly, we give how we do the prediction and training in Section 4.3.5.

4.3.1 Edge-Order Preserving Aggregation (EOPA) Layer. Given EOP multigraphs that are converted losslessly from sessions, the GNN still needs to properly process the graphs so that different sessions can be mapped to different representations. This is not possible in the existing GNN-based models for session-based recommendation, because they use permutation-invariant aggregation functions which ignore the relative order of edges. Therefore, these models are only suitable for datasets where the ordering information between edges is not important, which means that there is a limitation in their modeling capability. To fill this gap, we propose the edge-order preserving aggregation (EOPA) layer which aggregates the information passed from neighboring nodes using a GRU. To be specific, let $OE_{in}(i) = [(j_1, i), (j_2, i), \cdots, (j_{d_i}, i)]$ be an ordered list of the edges in $E_{in}(i)$, where d_i is the in-degree of node i. $OE_{in}(i)$ can be obtained from $E_{in}(i)$ using the integer attributes of the edges in an EOP multigraph. The aggregated information from neighbors is defined as follows:

$$\text{agg}_i^{(l)} = \boldsymbol{h}_{d_i}^{(l)} \tag{4}$$

$$\boldsymbol{h}_k^{(l)} = \text{GRU}^{(l)}\left(f_{msg}^{(l)}\left(\boldsymbol{x}_i^{(l)}, \boldsymbol{x}_{j_k}^{(l)}\right), \boldsymbol{h}_{k-1}^{(l)}\right) \tag{5}$$

where $\{\boldsymbol{h}_k^{(l)} : 0 \le k \le d_i\}$ are the hidden states of GRU and $f_{msg}^{(l)}$ can be any valid message function that computes the message passed from node j_k to node i. The initial state $\boldsymbol{h}_0^{(l)}$ is set to a zero vector.

The GRU aggregator is a type of RNN aggregators. We choose GRU instead of LSTM because it was shown that GRU outperforms LSTM for the session-based recommendation task [8, 11]. Although RNN aggregators have been proposed in existing work, e.g., an LSTM aggregator was proposed in [7], it should be noted that these RNN aggregators work differently from ours. Specifically, existing RNN aggregators are employed to deliberately ignore the relative order of incoming edges by performing aggregation on a random permutation of these edges, whereas our GRU aggregator performs aggregation in a fixed order. This is because in the setting of session-based recommendation, the edges are naturally ordered by time. However, we need to emphasize that the GRU aggregator is not our primary contribution. Our primary contribution is applying the GRU aggregator to solve the information loss problem described in Section 4.2.

The same message and update functions from the existing GNN models can be used together with the GRU aggregator, but, considering the powerful expressive capacity of GRU, we simply use linear transformation for both the message and update functions. Thus, putting everything together, the EOPA layer of our model is defined as follows:

$$\boldsymbol{x}_i^{(l+1)} = \boldsymbol{W}_{upd}^{(l)}\left(\boldsymbol{x}_i^{(l)} \| \boldsymbol{h}_{d_i}^{(l)}\right) \tag{6}$$

$$\boldsymbol{h}_k^{(l)} = \text{GRU}^{(l)}\left(\boldsymbol{W}_{msg}^{(l)}\boldsymbol{x}_{j_k}^{(l)}, \boldsymbol{h}_{k-1}^{(l)}\right) \tag{7}$$

where $\boldsymbol{W}_{upd}^{(l)} \in \mathbb{R}^{d\times 2d}, \boldsymbol{W}_{msg}^{(l)} \in \mathbb{R}^{d\times d}$ are learnable parameters, and $\|$ denotes concatenation.

As for the information about the last node, it will be used in the readout function to be described in Section 4.3.4, so that different sessions can be mapped to different representations.

4.3.2 Shortcut Graph Attention (SGAT) Layer. In general, each layer propagates information for one step, and thus one layer can only capture 1-hop relationship between nodes. To capture multi-hop relationship, one could stack multiple GNN layers. Unfortunately, this would introduce the over-smooth problem [12, 26], which states that the node representations converge to the same value. Since the over-smooth problem usually occurs when the number of layers is larger than 3, stacking multiple layers is not a good way to capture multi-hop relationship. Besides, even if one could stack multiple layers, the GNN can only capture up to k-hop relationship where k is equal to the number of layers. However, in real world applications such as e-commerce websites, it is very common that a session has a length larger than k. Thus, existing GNN models for session-based recommendation cannot effectively capture dependencies at very long ranges. To solve this problem, we propose the shortcut graph attention (SGAT) layer, which essentially uses edges in the shortcut graph obtained by S2SG for fast information propagation.

Specifically, the SGAT layer propagates information along edges in the shortcut graph using the following attention mechanism.

$$\boldsymbol{x}_i^{(l+1)} = \sum_{(j,i)\in E_{in}(i)} \alpha_{ij}^{(l)} \boldsymbol{W}_{val}^{(l)} \boldsymbol{x}_j^{(l)} \tag{8}$$

$$\boldsymbol{\alpha}_i^{(l)} = \text{softmax}(\boldsymbol{e}_i^{(l)}) \tag{9}$$

$$e_{ij}^{(l)} = \left(\boldsymbol{p}^{(l)}\right)^T \sigma\left(\boldsymbol{W}_{key}^{(l)}\boldsymbol{x}_i^{(l)} + \boldsymbol{W}_{qry}^{(l)}\boldsymbol{x}_j^{(l)} + \boldsymbol{b}^{(l)}\right) \tag{10}$$

where $E_{in}(i)$ is the set of incoming edges of node i in the shortcut graph, $\boldsymbol{p}^{(l)}, \boldsymbol{b}^{(l)} \in \mathbb{R}^d$ and $\boldsymbol{W}_{key}^{(l)}, \boldsymbol{W}_{qry}^{(l)}, \boldsymbol{W}_{val}^{(l)} \in \mathbb{R}^{d\times d}$ are learnable parameters.

The edges in the shortcut graph directly connect each item to all of its subsequent items without going through intermediate items. Therefore, the edges in the shortcut graph can be viewed as "shortcut connections" between items. The SGAT layer can effectively capture long-term dependencies of any length because it propagates information along the shortcut connections between items in one step without going through intermediate items. It can be combined with the original layers in existing GNN-based methods to increase their capability of capturing long-range dependencies.

4.3.3 Stacking Multiple Layers. The EOPA layer and the SGAT layer are proposed to solve two information loss problems of previous GNN-based methods for session-based recommendation. To build a GNN model that does not have the information loss problems, we stack multiple EOPA and SGAT layers. Instead of putting all of the EOPA layers after all of the SGAT layers or vice versa, we interleave EOPA layers with SGAT layers for the following reasons:

- The shortcut graph is a lossy conversion of the original session, so continuously stacking multiple SGAT layers will introduce the lossy session encoding problem. The problem is more severe with more SGAT layers because the amount of information loss accumulates. By interleaving the EOPA and SGAT layers, the lost information can be retained in the subsequent EOPA layers and the SGAT layers can just focus on capturing long-range dependencies.
- Another advantage of interleaving two kinds of layers is that each kind of layers can effectively utilize the features captured by the other kind of layers. Since the EOPA layers are more capable of capturing local context information and the SGAT layers are more capable of capture global dependencies, interleaving the two kinds of layers can effectively combine the advantages of both and improve the model's capabilities in learning more complex dependencies.

To further facilitate feature reuse, we introduce the *dense connections* proposed in [10]. The input to each layer consists of the output features of all previous layers. To be specific, originally, the input to layer l is $\{\boldsymbol{x}_i^{(l-1)} : i \in V\}$. With the dense connections, the input to layer l is changed to $\{\boldsymbol{x}_i^{(0)} \| \boldsymbol{x}_i^{(1)} \| \cdots \| \boldsymbol{x}_i^{(l-1)} : i \in V\}$, which is the concatenation of the output of all previous layers. It is shown that a deep learning model with dense connections is more parameter-efficient, i.e., achieving the same performance with much fewer parameters, because each of the higher layers can use not only the abstract features at its previous layer but also the low-level features at lower layers [10].

4.3.4 Generating Session Embedding. After the message passing of all layers is finished, we obtain the final representations of all nodes. To represent the current session as an embedding vector, we apply a readout function proposed in [23], which computes a graph-level representation by aggregating node representations using the attention mechanism. Let $\boldsymbol{x}_{last}^{(L)}$ denote the final node representation of the last item in the session. The graph-level representation $\boldsymbol{h}_G$ is defined as follows:

$$\boldsymbol{h}_G = \sum_{i \in V} \beta_i \boldsymbol{x}_i^{(L)} \tag{11}$$

$$\boldsymbol{\beta} = \text{softmax}(\boldsymbol{\epsilon}) \tag{12}$$

$$\epsilon_i = \boldsymbol{q}^T \sigma(\boldsymbol{W}_1 \boldsymbol{x}_i^{(L)} + \boldsymbol{W}_2 \boldsymbol{x}_{last}^{(L)} + \boldsymbol{r}) \tag{13}$$

where $\boldsymbol{q}, \boldsymbol{r} \in \mathbb{R}^d$ and $\boldsymbol{W}_1, \boldsymbol{W}_2 \in \mathbb{R}^{d \times d}$ are learnable parameters.

The graph-level representation captures the *global preferences* of the current session, denoted by $\boldsymbol{s}_g = \boldsymbol{h}_G$. Since previous studies [11, 17, 23] showed that it is also important to explicitly consider users' recent interests, we define a *local preference vector* by $\boldsymbol{s}_l = \boldsymbol{x}_{last}^{(L)}$. Then, we compute the *session embedding* as a linear transformation of both the global and local session preferences:

$$\boldsymbol{s}_h = \boldsymbol{W}_h(\boldsymbol{s}_g \| \boldsymbol{s}_l) \tag{14}$$

where $\boldsymbol{W}_h \in \mathbb{R}^{d \times 2d}$ is a learnable matrix parameter.

4.3.5 Prediction and Training. After obtaining the session embedding, we can use it to make recommendations by computing a probability distribution of the next item. For each item $i \in I$, we first compute a score using its embedding $\boldsymbol{v}_i$ and the session embedding as follows:

$$z_i = \boldsymbol{s}_h^T \boldsymbol{v}_i \tag{15}$$

Then, the predicted probability of the next item being item i, $\hat{y}_i$, can be computed by:

$$\hat{y}_i = \frac{\exp(z_i)}{\sum_{j \in I} \exp(z_j)} \tag{16}$$

For top-K recommendation, we can just recommend the items with top K probabilities.

Let $\boldsymbol{y}$ denote the ground-truth probability distribution of next item, which is a one-hot vector. The loss function is defined to be the cross-entropy of the prediction and the ground truth:

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\boldsymbol{y}^T \log \hat{\boldsymbol{y}}$$

Then, all parameters as well as the item embeddings are randomly initialized and jointly learned in an end-to-end back-propagation training paradigm.

5 EXPERIMENTS

In this section, we first describe the experimental settings, including the datasets, compared methods and evaluation metrics. Then, we make detailed analysis on the experimental results.

5.1 Datasets

We conducted our experiments on the following three public real-world datasets which are commonly used in the literature of session-based recommendation [11, 16, 17, 23, 27] :

- *Diginetica*[3] is a dataset that comes from CIKM Cup 2016. Its transaction data is suitable for session-based recommendation. Fowllowing [11, 14, 17, 23], we used the sessions in the last week as the test set.
- *Gowalla*[4] is a check-in dataset that is widely used for point-of-interest recommendation. Following [6, 21], we kept the top 30,000 most popular locations, and grouped users' check-in records into disjoint sessions by splitting at intervals between adjacent records that are longer than 1 day. The last 20% of the sessions were used as the test set.
- *Last.fm*[5] is a dataset that is widely used in many recommendation tasks. We used this dataset for music artist recommendation. Following [6, 17], we kept the top 40,000 most popular artists and set the splitting interval to 8 hours. Similar to *Gowalla*, the most recent 20% of the sessions were used as the test set.

[3] http://cikm2016.cs.iupui.edu/cikm-cup
[4] https://snap.stanford.edu/data/loc-gowalla.html
[5] http://ocelma.net/MusicRecommendationDataset/lastfm-1K.html

Following [11, 14, 16, 17, 23], we first filtered short sessions and infrequent items and then applied a data augmentation technique described in [11, 14, 23]. Some statistics of the datasets after prepossessing are shown in Table 1.

Table 1: Statistics of datasets used in the experiments

Statistic	Diginetica	Gowalla	Last.fm
No. of Clicks	981,620	1,122,788	3,835,706
No. of Sessions	777,029	830,893	3,510,163
No. of Items	42,596	29,510	38,615
Average length	4.80	3.85	11.78

5.2 Baselines and Evaluation Metrics

To evaluate the performance of the proposed model, we compared it with the following representative methods.

- **Item-KNN** [4] is a neighborhood method that recommends items that are similar to the previous items in the current session, where the similarity between two items is defined by their cosine similarity.
- **FPMC** [18] is a Markov-chain based method for next-basket recommendation. To adapt it for session-based recommendation, we consider the next item as the next basket.
- **NARM** [11] employs RNNs with attention to capture user's main purposes and sequential behaviors.
- **NextItNet** [27] is a CNN-based method for next-item recommendation. It uses dilated convolution to increase the receptive fields without using the lossy pooling operations.
- **SR-GNN** [23] transforms sessions into directed unweighted graphs and extracts item features by propagating information along both directions of the edges using a GGNN [13].
- **FGNN** [16] converts sessions into directed weighted graphs and uses an adapted GAT [22] to learn item representations.
- **GC-SAN** [25] first uses GGNN to extract local context information and then employs a self-attention network (SAN) to capture global dependencies.

Following [14, 21], for each method, grid search is applied to find the optimal hyper-parameters using the last 20% of the training set as the validation set. The ranges of the hyper-parameters are: $\{16,\ 32,\ 64,\ 96,\ 128\}$ for embedding dimension d, and $\{10^{-4}, 10^{-3}, \cdots, 10^{-1}\}$ for learning rate η. For the GNN-based models, we also search the total number of layers L in $\{1, 2, 3, 4, 5\}$. We use the Adam optimizer to train the models and the batch size is set to 512. We report the result of each model under its optimal hyper-parameter settings.

Following previous studies [11, 16, 17, 23], we adopt the commonly used HR@20 (Hit Rate)[6] and MRR@20 (Mean Reciprocal Rank) as our evaluation metrics.

5.3 Performance Comparisons

To demonstrate the overall performance of the proposed model, we compared it with the state-of-the-art recommendation methods.

[6]Note that [11, 16, 17, 23] used different metric names for HR@20 (e.g., P@20 and Recall@20). But, they used the same formula to obtain this measurement (i.e., the proportion of cases when the desired item is among the top-20 items in all test cases).

The experimental results of all compared methods are shown in Table 2, from which we could have the following observations.

Table 2: Experimental results (%) on three datasets

Method	Diginetica		Gowalla		Last.fm	
	HR@20	MRR@20	HR@20	MRR@20	HR@20	MRR@20
Item-KNN	39.51	11.22	38.60	16.66	14.90	4.04
FPMC	28.50	7.67	29.91	11.45	12.86	3.78
NextItNet	45.41	15.19	45.15	21.26	20.12	7.08
NARM	49.80	16.57	50.07	23.92	21.83	7.59
FGNN	50.03	17.01	50.06	24.12	22.20	8.02
SR-GNN	50.81	17.31	50.32	24.25	22.33	8.23
GC-SAN	50.90	17.63	50.68	24.67	22.64	8.42
LESSR	**51.71**	**18.15**	**51.34**	**25.49**	**23.37**	**9.01**
Improv.	1.59%	2.95%	1.30%	3.32%	3.22%	7.01%

The performance of the conventional methods, including Item-KNN and FPMC, are not competitive. These methods make recommendations merely based on item similarities or transitions, without considering other important sequential information such as the relative order among previous items in the active session.

All of the neural network-based models outperform the conventional methods by a large margin, proving the effectiveness of deep learning technology in session-based recommendation. The neural network-based models are capable of capturing complex sequential patterns for making recommendations. However, their sequential modeling capabilities are not equally powerful. The CNN-based method, NextItNet, is less performant than other RNN-based and GNN-based methods. One possible reason could be that CNNs are only good at capturing consecutive sequential patterns but not long-term dependencies. NARM achieves competitive performance because it uses RNN to capture sequential behaviors and the attention mechanism to capture global preferences.

GNN-based methods generally outperform other methods, suggesting that GNNs offer a promising direction for session-based recommendation. Although FGNN uses a conversion method that captures more information than the method used by SR-GNN, FGNN does not outperform SR-GNN, suggesting the importance of choosing a powerful message passing scheme. GC-SAN outperforms SR-GNN because it uses SAN to capture global dependencies between distant items, showing the effectiveness of explicitly considering long-range dependencies. The proposed method LESSR uses the powerful EOPA to extract local contextual information and SGAT to capture long-range dependencies of any length, and thus it can significantly outperform all compared methods, proving the efficacy and the validity of the proposed method. Among the three datasets, LESSR obtains the largest performance improvement in *Last.fm* because this dataset has the largest average length, and thus LESSR benefits most from preserving the edge order and capturing the long-term dependencies.

5.4 Ablation Studies

In this section, we performed some ablation studies to show the effectiveness of the EOPA and SGAT layers.

5.4.1 Effectiveness of EOPA Layer. To evaluate the effectiveness of the EOPA layer, we compared it with the message passing (MP)

layers from other GNN-based methods, including GGNN from SR-GNN [23], WGAT from FGNN [16], and SAN from GC-SAN [25]. We used the same readout function described in Section 4.3.4. The embedding size was set to 96 because most models achieved the best performance with this embedding size. For GGNN and WGAT, the number of layers was set to 1 and they were compared with a model with 1 EOPA layer. To show the importance of EOPA, we also compared the EOPA layer with its modified variant which performs aggregation on a random permutation of the incoming edges. In other words, the modified EOPA layer ignores the order attributes of edges in the EOP multigraph. We denote the variant "EOPA (rand)". For SAN, since it is designed to work with GGNN, we compared a model consisting of GGNN followed by SAN, with a model consisting of GGNN followed by EOPA. The results are shown in Table 3. Each model is denoted by the name(s) of the MP layer(s) that it contains.

Table 3: The performance of different message passing layers

Layer(s)	Diginetica		Gowalla		Last.fm	
	HR@20	MRR@20	HR@20	MRR@20	HR@20	MRR@20
WGAT	49.71	16.46	50.03	24.02	21.89	7.89
GGNN	49.85	16.59	50.24	24.23	22.08	8.02
EOPA (rand)	49.81	16.56	50.18	24.11	22.05	8.06
EOPA	**50.30**	**16.93**	**50.86**	**24.89**	**22.31**	**8.36**
GGNN+SAN	50.06	16.72	50.37	24.44	22.22	8.18
GGNN+EOPA	**50.28**	**16.91**	**50.76**	**24.96**	**22.41**	**8.40**

We can see that EOPA consistently outperforms all other types of MP layers, proving the effectiveness of EOPA. The other types of MP layers perform worse because they use lossy encoding schemes to convert sessions to graphs, and their aggregation schemes do not consider the relative order of edges. Note that although "EOPA" and "EOPA (rand)" use the same encoding scheme, i.e., S2MG, "EOPA" outperforms "EOPA (rand)" because "EOPA (rand)" randomly permutes the incoming edges when performing aggregation. Therefore, it is not enough to just use a conversion method that preserves the edge order. The aggregation scheme also needs to preserve the edge order.

5.4.2 Effectiveness of SGAT Layer. To show the effectiveness of SGAT, we replaced the last MP layer with a SGAT layer for each existing GNN-based method and checked if the replacement improved the performance. Since SR-GNN and FGNN have only one kind of MP layer, the number of layers was set to 2 so that the modified model had one original MP layer followed by one SGAT layer. For GC-SAN, the number of layers was set to 3 because it has two kinds of MP layers. For LESSR, we compared a model with two EOPA layers and a model with one EOPA layer followed by a SGAT layer. The embedding size was set to 96. Due to the space limitation, we only reported the performance in terms of *HR@20* on the *Diginetica* dataset since it is more commonly used in session-based recommendation. The results are shown in Table 4.

We can see that the replacement helps improving the models performance. This is because the original layers are only good at capturing local contextual information but not complex long-range dependencies. By replacing an original layer with a SGAT layer, the model's capability of capturing long-range dependencies is improved. The SAN layers in GC-SAN also have the ability to capture global dependencies between distant nodes. However, they propagate information between every pair of nodes, which completely discard the connectivity information in the original session. In contrast, our SGAT layer propagates information from u to v only if u appears before v, which preserves some connectivity information. Therefore, it is reasonable to see a performance gain when a SAN layer is replaced by our SGAT layer.

Table 4: Performance differences in terms of HR@20

Model	LESSR	FGNN	SR-GNN	GC-SAN
Original	50.29	49.51	50.02	50.01
Modified	50.56	50.08	50.18	50.36
Improv.	0.54%	1.15%	0.32%	0.70%

Table 5: The performance of different orders of layers

Model	Diginetica		Gowalla		Last.fm	
	HR@20	MRR@20	HR@20	MRR@20	HR@20	MRR@20
SSEE	50.17	16.75	50.77	24.70	21.98	8.19
EESS	50.34	16.83	50.71	24.83	22.11	8.25
SESE	50.36	16.81	50.74	24.75	22.07	8.20
ESES	**50.63**	**17.08**	**50.82**	**25.04**	**22.48**	**8.41**

5.4.3 Order of EOPA and SGAT Layers. . To show the advantages of the proposed order of layers, we compared four models, including EESS, SSEE, ESES and SESE. Each model contains two EOPA layers and two SGAT layers, where EOPA and SGAT are abbreviated as E and S, respectively. The order of characters (i.e., E and S) denotes the order of layers. For example, EESS has two EOPA layers followed by two SGAT layers. The embedding size was set to 96 and residual connections were used. Table 5 shows the results. We can see that SSEE performs the worst because it cannot use the advantages of either kinds of layers. SESE and EESS have similar performance, suggesting the benefits of interleaving two kinds of layers and putting EOPA before SGAT. ESES outperforms all other models, proving that the proposed order is the most effective way to utilize the advantages of both kinds of layers.

5.5 Hyper-parameter Study

In this section, we study how the embedding size and the number of layers affect the performance of the proposed method. Due to the limited space, we only show the results in terms of HR@20. The results are shown in Figure 4.

From the results, we can see that increasing the embedding size or the number of layers does not always result in a better performance. For the smaller dataset *Diginetica*, the optimal embedding size is 32. The performance decreases quickly when the embedding size exceeds this optimal value because the model overfits. For the other two larger datasets, increasing embedding size generally improves

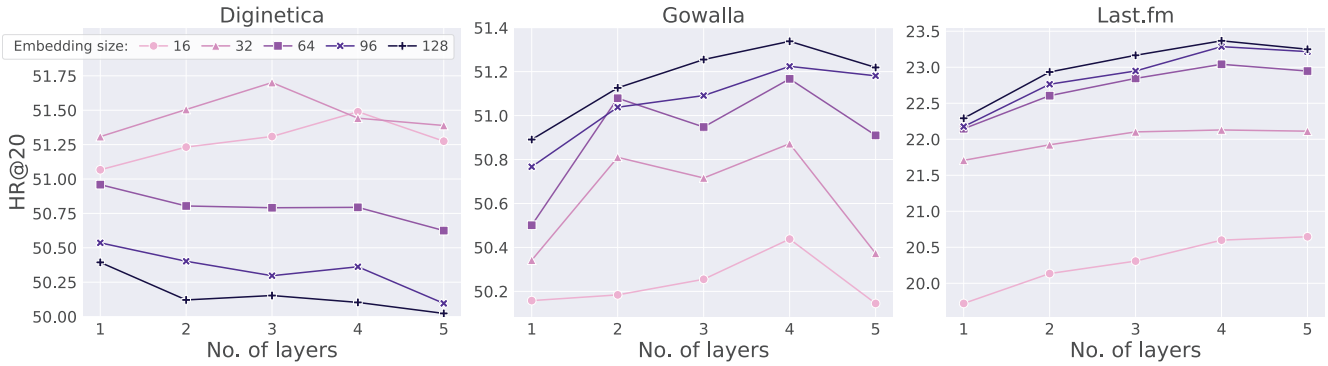

Figure 4: The performance of LESSR

the performance because a larger embedding size increases the model's learning capacity.

If the embedding size is smaller than the optimal value, as the number of layers increases, the performance first improves and then drops when the number of layers is 3 or 4. This is because the model's learning capacity increases when the number of layers becomes larger, but too many layers do not help because the over-smooth problem occurs, even if the model has not overfitted. Therefore, stacking more layers is not an effective method to capture long-range dependencies, and it is meaningful to apply alternative ways such as using SGAT layers to improve the model's capability in capturing long-range dependencies.

6 CONCLUSION

In this paper, we identify two information loss problems in the existing GNN models for session-based recommendation, namely the lossy session encoding and the ineffective long-range dependency capturing problems. To solve the two problems, we propose the EOPA and the SGAT layers, which rely on the two conversion methods that convert sessions to graphs, including S2MG and S2SG. By combining the two kinds of layers, we build a model called LESSR that does not have the two information loss problems and the experimental results show that LESSR outperforms state-of-the-art methods on three public datasets. For future work, we are interested in applying LESSR to personalized and streaming session-based recommendation.

Acknowledgements: We are grateful to the anonymous reviewers for their constructive comments on this paper. The research is supported by 16214017.

REFERENCES

[1] Shuo Chen, Josh L. Moore, Douglas Turnbull, and Thorsten Joachims. 2012. Playlist Prediction via Metric Embedding. In *KDD*. 714–722.
[2] Tianwen Chen and Raymond Chi-Wing Wong. 2019. Session-based Recommendation with Local Invariance. In *ICDM*. 994–999.
[3] Zhiyong Cheng, Jialie Shen, Lei Zhu, Mohan S. Kankanhalli, and Liqiang Nie. 2017. Exploiting Music Play Sequence for Music Recommendation. In *IJCAI*. 3654–3660.
[4] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. 2010. The YouTube Video Recommendation System. In *RecSys*. 293–296.
[5] Ricardo Dias and Manuel J. Fonseca. 2013. Improving Music Recommendation in Session-based Collaborative Filtering by Using Temporal Context. In *ICTAI*. 783–788.
[6] Lei Guo, Hongzhi Yin, Qinyong Wang, Tong Chen, Alexander Zhou, and Nguyen Quoc Viet Hung. 2019. Streaming Session-based Recommendation. In *KDD*. 1569–1577.
[7] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*. 1024–1034.
[8] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. In *ICLR*.
[9] Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Domonkos Tikk. 2016. Parallel Recurrent Neural Network Architectures for Feature-rich Session-based Recommendations. In *RecSys*. 241–248.
[10] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2017. Densely Connected Convolutional Networks. In *CVPR*. 2261–2269.
[11] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural Attentive Session-based Recommendation. In *CIKM*. 1419–1428.
[12] Qimai Li, Zhichao Han, and Xiaoming Wu. 2018. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *AAAI*. 3538–3545.
[13] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. 2016. Gated Graph Sequence Neural Networks.. In *ICLR*.
[14] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. 2018. STAMP: Short-Term Attention/Memory Priority Model for Session-based Recommendation. In *KDD*. 1831–1839.
[15] Sung Eun Park, Sangkeun Lee, and Sang goo Lee. 2011. Session-based Collaborative Filtering for Predicting the Next Song. In *CNSI*. 353–358.
[16] 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.
[17] Pengjie Ren, Zhumin Chen, Jing Li, Zhaochun Ren, Jun Ma, and Maarten de Rijke. 2019. RepeatNet: A Repeat Aware Neural Recommendation Machine for Session-based Recommendation. In *AAAI*. 4806–4813.
[18] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing Personalized Markov Chains for Next-basket Recommendation. In *WWW*. 811–820.
[19] Guy Shani, David Heckerman, and Ronen I. Brafman. 2005. An MDP-Based Recommender System. *JMLR* 6 (2005), 1265–1295.
[20] Jing Song, Hong Shen, Zijing Ou, Junyi Zhang, Teng Xiao, and Shangsong Liang. 2019. ISLF: Interest Shift and Latent Factors Combination Model for Session-based Recommendation. In *IJCAI*. 5765–5771.
[21] Jiaxi Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. In *WSDM*. 565–573.
[22] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.
[23] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. 2019. Session-based Recommendation with Graph Neural Network. In *AAAI*. 346–353.
[24] Shu Wu, Mengqi Zhang, Xin Jiang, Ke Xu, and Liang Wang. 2019. Personalizing Graph Neural Networks with Attention Mechanism for Session-based Recommendation. *TKDE* 31 (2019).
[25] 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.
[26] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*. 5449–5458.
[27] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M. Jose, and Xiangnan He. 2019. A Simple Convolutional Generative Network for Next Item Recommendation. In *WSDM*. 582–590.