

 Open access • Proceedings Article • DOI:10.1145/3357384.3358010

Rethinking the Item Order in Session-based Recommendation with Graph Neural Networks — [Source link](#)

Ruihong Qiu, Jingjing Li, Zi Huang, Hongzhi Yin

Institutions: University of Queensland, University of Electronic Science and Technology of China

Published on: 03 Nov 2019 - Conference on Information and Knowledge Management

Topics: Graph (abstract data type) and Recommender system

Related papers:

- [Session-Based Recommendation with Graph Neural Networks](#)
- [Factorizing personalized Markov chains for next-basket recommendation](#)
- [Neural Attentive Session-based Recommendation](#)
- [STAMP: Short-Term Attention/Memory Priority Model for Session-based Recommendation](#)
- [Graph contextualized self-attention network for session-based recommendation](#)

Share this paper:    

View more about this paper here: <https://typeset.io/papers/rethinking-the-item-order-in-session-based-recommendation-2ov3vzhy3l>

Rethinking the Item Order in Session-based Recommendation with Graph Neural Networks

Ruihong Qiu
The University of Queensland
Brisbane, Australia
r.qiu@uq.edu.au

Zi Huang
The University of Queensland
Brisbane, Australia
huang@itee.uq.edu.au

Jingjing Li
University of Electronic Science and Technology of China
Chengdu, China
lijin117@yeah.net

Hongzhi Yin*
The University of Queensland
Brisbane, Australia
h.yin1@uq.edu.au

ABSTRACT

Predicting a user's preference in a short anonymous interaction session instead of long-term history is a challenging problem in the real-life session-based recommendation, e.g., e-commerce and media stream. Recent research of the session-based recommender system mainly focuses on sequential patterns by utilizing the attention mechanism, which is straightforward for the session's natural sequence sorted by time. However, the user's preference is much more complicated than a solely consecutive time pattern in the transition of item choices. In this paper, therefore, we study the item transition pattern by constructing a session graph and propose a novel model which collaboratively considers the sequence order and the latent order in the session graph for a session-based recommender system. We formulate the next item recommendation within the session as a graph classification problem. Specifically, we propose a weighted attention graph layer and a Readout function to learn embeddings of items and sessions for the next item recommendation. Extensive experiments have been conducted on two benchmark E-commerce datasets, *Yoochoose* and *Diginetica*, and the experimental results show that our model outperforms other state-of-the-art methods.

CCS CONCEPTS

• Information systems → Recommender systems.

KEYWORDS

recommender system; session-based recommendation; graph neural networks

ACM Reference Format:

Ruihong Qiu, Jingjing Li, Zi Huang, and Hongzhi Yin. 2019. Rethinking the Item Order in Session-based Recommendation with Graph Neural Networks.

*Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

CIKM '19, November 3–7, 2019, Beijing, China

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-6976-3/19/11...\$15.00

<https://doi.org/10.1145/3357384.3358010>

In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19)*, November 3–7, 2019, Beijing, China. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3357384.3358010>

1 INTRODUCTION

Recommender system (RS) is an important tool to precisely advertise interested items to potential users in today's flood of web information. In recent years, the content-based RS [21] and the collaborative filtering RS [7, 27] are two widely used methods because they can effectively approximate the similarity between items while being simple and efficient. However, a clear drawback of these approaches is that a user's recent preference is ignored. In many scenarios, for example, e-commerce, customers may have their current prioritized choices of products over other products because of their recent need. Consequently, this shift of preference will only be shown in recent interactions between users and items [35]. In this case, the content-based RS and the collaborative filtering RS would fail to capture the important change of the user's preference, which leads to useless or even negative recommendations.

In contrast, a session-based RS can deal with the shift of the user's preference by taking a recent session of user-item interactions (e.g., clicks of items within 24 hours) into consideration. However, it is very common that for modern commercial online systems, they do not record the user's long-term history. In order to make use of the user's preference, the session can be regarded as the representation of the recent preference of the current anonymous user [35]. As a result, how to represent a user's preference by extracting representative information from interactions in the session is the essence of the session-based RS.

As a session is a slice of interactions divided by time, it can be naturally represented as a time series sequence. The sequence characteristic is considered as the most important information by recent methods. However, it has some challenging limitations:

- The user's preference does not completely depend on the chronology of the sequence. With the prevalence of RNN [10] applied on the sequence data, for instance, GRU4REC [9] and NARM [16] mainly model the time order of items and encode these items using a RNN like neural networks. After encoding the item, the representation of the session is a combination of the item features. Such a paradigm of dealing with the session is natural with the original order of items inside the session. However, as mentioned above, the shift of

the user’s preference within the session indicates that items should not be simply considered as time series. The item transition pattern is more complicated.

- The recent approaches [16, 19, 38], which divide the user’s preference into the long-term (global) and the short-term (local) preference, are too simple to capture the complicated item transition pattern. These methods choose the last item of the session to stand for the short-term (local) preference and the remaining items for the long-term (global) preference. This setting directly ignores the pattern of item choices, which introduces the bias to the model. NARM [16] applies a self-attention on the last item after encoding with a RNN. To alleviate the influence of time order, STAMP [19] only utilizes the self-attention mechanism without RNN. SR-GNN [38] proposes to use a single layer gated graph neural network [18] to learn the representation of items and again, a self-attention on the last item to extract a session level feature. Actually, the self-attention calculates the relative importance of the last single item, which ignores a specific item transition pattern within the session.

With the problems mentioned above, it is important to determine the intrinsic order of items within a session. This inherent order is neither the straightforward time order by RNN, nor the complete randomness by the self-attention. In this paper, a model named Full Graph Neural Network (FGNN) is proposed to learn the inherent order of the item transition pattern and compute a session level representation to generate recommendation. To utilize graph neural networks, we build a session graph for every session and formulate the recommendation as a graph classification problem. In order to capture the inherent order of the item transition pattern, which is vital for the item level feature representation, a multiple weighted graph attention layer (WGAT) network is proposed to compute the information flow between items within the session. After obtaining the item representations, the Readout function, which automatically learns to determine an appropriate order, is deployed to aggregate these features. Extensive experiments are conducted on two benchmark e-commerce datasets, the *Yoochoose* dataset from the RecSys Challenge 2015 and the *Diginetica* dataset from CIKM Cup 2016. The experimental results show the superiority of our method in the task of next item recommendation. Our main contributions are summarized as follows:

- To the best of our knowledge, we are the first to investigate the inherent order of item transition pattern in session-based recommendation. Specifically, we propose a novel FGNN model to perform the next item session-based recommendation based on the inherent order.
- A novel WGAT model is proposed to serve as the item feature encoder by learning to assign different weights to different neighbors. It helps to effectively convey the information between items.
- A Readout function is applied to generate appropriate graph level representation for item recommendation. The Readout function can learn the best order of the item transition pattern in the graph.
- We conduct extensive experiments on two benchmark e-commerce datasets and achieve state-of-the-art results.

2 RELATED WORK

In this section, we first review some related work about the general recommender system (RS) in Section 2.1 and the session-based recommender system (SBRS) in Section 2.2. At last, we will describe graph neural networks (GNN) for the node representation learning and graph classification problems in Section 2.3.

2.1 General Recommender System

The most popular method in recent years for the general recommender system is the collaborative filtering (CF), which represents the user interest based on the whole history. For example, the famous shallow method, Matrix Factorization (MF) [13] factorizes the whole user-item interaction matrix with latent representation for every user and item. With the prevalence of deep learning, neural networks are widely used. Neural collaborative filtering (NCF) [7] first proposes to use the multi layer perceptron to approximate the matrix factorization process. More subsequent work extends the incorporation of different deep learning tools, for instance, zero-shot learning and domain adaptation [14, 15]. These methods all depend on the identification of users and the whole record of interactions for every user. However, the user information is anonymous for many modern commercial online systems, which leads to the failure of these CF based algorithms.

2.2 Session-based Recommender System

The research on the session-based recommender system (SBRS) is a sub-field of RS. Compared with RS, SBRS takes the user’s recent user-item interactions into consideration rather than requiring all historical actions. SBRS is based on the assumption that the recent choice of items can be viewed as the recent preference of a user.

Sequential recommendation is based on the Markov chain model [28, 42], which learns the dependence of items of a sequence data to predict the next click. Using probabilistic decision-tree models, Zimdars et al. [42] proposed to encode the state of the item transition pattern. Shani et al. [28] made use of a Markov Decision Process (MDP) to compute the probability of recommendation with the transition probability between items.

Deep learning models are popular recently with the boom of recurrent neural networks [1, 2, 10, 29, 36], which is naturally designed for processing sequential data. Hidasi et al. [9] proposed the GRU4REC, which applies a multi layer GRU [2] to simply treat the data as time series. Based on the RNN model, some work makes improvements on the architectural choice and the objective function design [8, 30]. In addition to RNN, Jannach and Ludewig [11] proposed to use the neighborhood-based method to capture co-occurrence signals. Incorporating content features of items, Tuan and Phuong [31] utilized 3D convolutional neural networks to learn more accurate representations. Wu et al. [37] proposed a list-wise deep neural network model to train a ranking model. Some recent work uses the attention mechanism to avoid the time order. NARM [16] stacks GRU as the encoder to extract information and then a self-attention layer to assign weight to each hidden state to sum up as the session embedding. To further alleviate the bias introduced by time series, STAMP [19] entirely replaces the recurrent encoder with an attention layer. SR-GNN [38] applies a gated graph network [18] as the item feature encoder and a self-attention

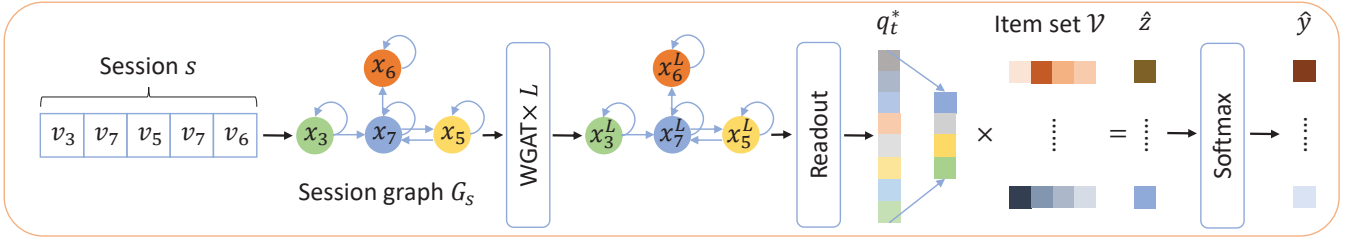


Figure 1: Pipeline of FGNN. The input to the model is organized as a session sequence s , which is then converted to a session graph G with node features x . L layers of WGAT serves as the encoder of node features for G . After being processed by WGAT, the session graph now contains L different semantic node representations x^L but with the same structure as the input session graph. The Readout function is applied to generate a session embedding based on the learned node features. Compared with other items in the item set \mathcal{V} , a recommendation score \hat{y}_i is finally generated.

layer to aggregate the item features together as the session feature. SSRM [5] considers a specific user’s history sessions and applies the attention mechanism to combine them. Though the attention mechanism can proactively ignore the bias introduced by the time order of interaction, it considers a session as a totally random set.

2.3 Graph Neural Networks

In recent years, GNN has attracted much interest in the deep learning society. Initially, GNN is applied to the simple situation on directed graphs [4, 26]. In recent years, many GNN methods [6, 12, 18, 33, 39] work under the mechanism that is similar to message passing network [3] to compute the information flow between nodes via edges. Additionally, the graph level feature representation learning is essential for graph level tasks, for example, graph classification and graph isomorphism [17, 39]. Set2Set [34] assigns each node in the graph a descriptor as the order feature and uses this re-defined order to process all nodes. SortPool [41] sorts the nodes based on their learned feature and uses a normal neural network layer to process the sorted nodes. DiffPool [40] designs two sets of GNN for every layer to learn a new dense adjacent matrix for a smaller size of the new densely connected graph.

3 PRELIMINARIES

In this section, we introduce how GNN works on the graph data. Let $G(V, E)$ denote a graph, where $v \in V$ is the node set with node feature vectors X_v and $e \in E$ is the edge set. There are two commonly popular tasks, e.g., *node classification* and *graph classification*. In this work, we focus on graph classification because our purpose is to learn a final embedding for the session rather than single items. For the *graph classification*, given a set of graphs $\{G_1, \dots, G_N\} \subseteq \mathcal{G}$ and the corresponding labels $\{y_1, \dots, y_N\} \subseteq \mathcal{Y}$, we aim to learn a representation of the graph \mathbf{h}_G to predict the graph label, $y_G = g(\mathbf{h}_G)$.

GNN makes use of the structure of the graph and the feature vectors of nodes to learn the representation of nodes or graphs. In recent years, most GNN work by aggregating information from neighboring nodes iteratively. After k iterations of update, the final representations of the nodes capture the structural information as well as the node information within k -hop neighbor. The procedure

can be formed as

$$\mathbf{a}_v^{(k)} = \text{Agg}(\{\mathbf{h}_u^{(k-1)}, u \in N(v)\}), \mathbf{h}_v^{(k)} = \text{Map}(\mathbf{h}_v^{(k-1)}, \mathbf{a}_v^{(k)}), \quad (1)$$

where $\mathbf{h}_v^{(k)}$ is the feature vector for node v in the k th layer. For the input \mathbf{h}_v^0 to the first layer, the feature vectors X_v are passed in. Agg and Map are two functions that can be defined in a different form. Agg serves as the aggregator to aggregate features of neighboring nodes. A typical characteristic of Agg is permutation invariant. Map is a mapping to transform the self information and the neighboring information to a new feature vector.

For the graph classification, a Readout function aggregates all node features from the final layer of the graph to generate a graph level representation \mathbf{h}_G :

$$\mathbf{h}_G = \text{Readout}(\{\mathbf{h}_v^{(k)}, v \in V\}), \quad (2)$$

where the Readout function needs to be permutation invariant as well.

4 METHOD

In this section, we describe our FGNN model in detail. Above all, in Section 4.1, we define the problem and give out the notations used in this paper. The complete pipeline of the calculation is demonstrated in Figure 1. At first, an input session sequence is converted into a session graph (Section 4.2). After obtaining the session graph, an L weighted graph attentional layer (WGAT) model is applied to perform graph convolution among the nodes (Section 4.3). Once the node features are learned, a Readout function combines all these features to form a graph level representation q^* (Section 4.4). Based on the graph representation, FGNN makes a recommendation by comparing it with the whole item set \mathcal{V} (Section 4.5). Finally, we describe the way we train our model in Section 4.6.

4.1 Problem Definition and Notation

The purpose of a session-based recommender system is to predict the next item that matches the user’s preference based on the interactions within the session. In the following, we give out the definition of the SBRS problem.

In a SBRS, there is an item set $\mathcal{V} = \{v_1, v_2, v_3, \dots, v_m\}$, where all items are unique and m denotes the number of items. A session sequence from an anonymous user is defined as a sequential list $\mathcal{S} = [v_{s,1}, v_{s,2}, v_{s,3}, \dots, v_{s,n}]$, $v_{s,*} \in \mathcal{V}$. n is the length of the session

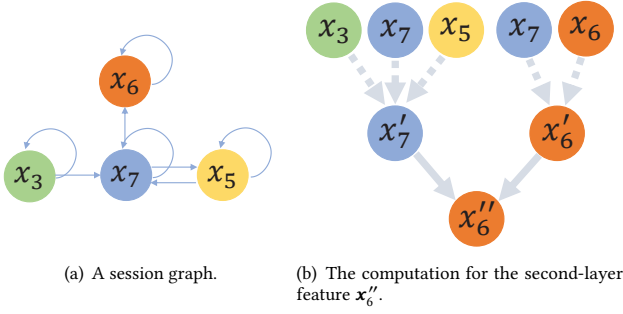


Figure 2: An example of how to compute a node representation of a two-layer GNN. The original session sequence is the same as the input in Figure 1. (a) The session graph is added with self-loop to every node. x_i is the input feature for the corresponding node v_i . (b) The computation of the second-layer feature x''_6 is based on all the first and second order neighboring nodes of v_6 . The first order neighbors are v_7 and v_6 itself. The second order neighbors of v_6 are the first order neighbors of v_7 and v_6 , i.e., v_3, v_7 and v_6 for v_7 , and v_7 and v_6 for v_6 .

S , which may contain duplicated items, $v_{s,p} = v_{s,q}$, $p, q < n$. The goal of our model is to predict the next item $v_{s,n}$ that matches the user’s preference the most.

During calculation, for every item $v \in \mathcal{V}$, our model learns a corresponding embedding vector $\mathbf{x} \in \mathbb{R}^d$, where d is the dimension of \mathbf{x} . For each training session, there is a label item v_{label} serving as the target to predict. In order to recommend items based on the given session and the whole item set, our model outputs a probability distribution $\hat{\mathbf{y}}$ over \mathcal{V} , where the items with top- K values in $\hat{\mathbf{y}}$ will be the candidates.

4.2 Session Graph

As shown in Figure 1, at the first stage, the session sequence is converted into a session graph for the purpose to process each session via GNN. Because of the natural order of the session sequence, we transform it into a weighted directed graph, $G_s = (V_s, E_s)$, $G_s \in \mathcal{G}$, where \mathcal{G} is the set of all session graphs. In the session graph G_s , the node set V_s represents all nodes, which are items $v_{s,n}$ from S . For every node v , the input feature is the initial embedding vector \mathbf{x} . The edge set E_s represents all directed edges $(v_{s,n-1}, v_{s,n}, w_{s,(n-1)n})$, where $v_{s,n}$ is the click of the item after $v_{s,n-1}$ in S , and $w_{s,(n-1)n}$ is the weight of the edge. The weight of the edge is defined as the frequency of the occurrence of the edge within the session. For the convenience, in the following, we use the nodes in the session graph to stand for the items in the session sequence. For the self-attention used in WGAT introduced in Section 4.3, if a node does not contain a self loop, it will be added with a self loop with a weight 1. Based on our observation of our daily life and the datasets, it is common for a user to click two consecutive items for a few times within the session. After converting the session into a graph, the final embedding of S is based on the calculation on this session graph G_s .

4.3 Weighted Graph Attentional Layer

After obtaining the session graph, a GNN is needed to learn embeddings for nodes in a graph, which is the $\text{WGAT} \times L$ part in Figure 1. In recent years, some baseline methods on GNN, for example, GCN [12] and GAT [33], are demonstrated to be capable of extracting features of the graph. However, most of them are only well-suited for unweighted and undirected graphs. For the session graph, weighted and directed, these baseline methods cannot be directly applied without losing the information carried by the weighted directed edge. Therefore, a suitable graph convolutional layer is needed to effectively convey information between the nodes in the graph.

In this paper, we propose a weighted graph attentional layer (WGAT), which simultaneously incorporates the edge weight when performing the attention aggregation on neighboring nodes. We describe the forward propagation of WGAT in the following. The information propagation procedures are shown in Figure 2. Figure 2(b) shows an example of how a two-layer GNN calculates the final representation of the node v_6 .

The input to a WGAT is a set of node initial features, the item embeddings, $\mathbf{x} = \{\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{n-1}\}$, $\mathbf{x}_i \in \mathbb{R}^d$, where n is the number of nodes in the graph, and d is the dimension of the embedding \mathbf{x}_i . After applying the WGAT, a new set of node features, $\mathbf{x}' = \{\mathbf{x}'_0, \mathbf{x}'_1, \mathbf{x}'_2, \dots, \mathbf{x}'_{n-1}\}$, $\mathbf{x}'_i \in \mathbb{R}^{d'}$, will be given out as the output. Specifically, the input feature vectors \mathbf{x}_i^0 of the first WGAT layer are generated from an embedding layer, whose input is the one-hot encoding of the item,

$$\mathbf{x}_i^0 = \text{Embed}(v_i), \quad (3)$$

where Embed is the embedding layer.

To learn the node representation via the higher order item transition pattern within the graph structure, a self-attention mechanism for every node i is used to aggregate information from its neighboring nodes $\mathcal{N}(i)$, which is defined as the nodes with edges towards the node i (may contain i itself if there is a self-loop edge). Because the size of the session graph is not huge, we can take the entire neighborhood of a node into consideration without any sampling. At the first stage, a self-attention coefficient e_{ij} , which determines how importantly the node j will influence the node i , is calculated based on $\mathbf{x}_i, \mathbf{x}_j$ and w_{ij} ,

$$e_{ij} = \text{Att}(\mathbf{W}\mathbf{x}_i, \mathbf{W}\mathbf{x}_j, w_{ij}), \quad (4)$$

where Att is a mapping $\text{Att} : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^1 \rightarrow \mathbb{R}^1$ and \mathbf{W} is a shared parameter which performs linear mapping across all nodes. As a matter of fact, the attention of a node i can extend to every node, which is a special case the same as how STAMP makes the attention of the last node of the sequence. Here we restrict the range of the attention within the first order neighborhood of the node i to make use of the inherent structure of the session graph S . To compare the importance of different nodes directly, a softmax function is applied to convert the coefficient into a probability form across the neighbors and itself,

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ik})}. \quad (5)$$

The choice of *att* can be diversified. In our experiments, we use an MLP layer with the parameter $\mathbf{W}_{att} \in \mathbb{R}^{2d+1}$, followed by a

LeakyRelu non-linearity unit with negative input slope $\alpha = 0.2$

$$\alpha_{ij} = \frac{\exp(\text{LeakyRelu}(\mathbf{W}_{att}[\mathbf{W}\mathbf{x}_i \parallel \mathbf{W}\mathbf{x}_j \parallel \mathbf{w}_{ij}]))}{\sum_{k \in \mathcal{N}(i)} \exp(\text{LeakyRelu}(\mathbf{W}_{att}[\mathbf{W}\mathbf{x}_i \parallel \mathbf{W}\mathbf{x}_k \parallel \mathbf{w}_{ik}]))}, \quad (6)$$

where \parallel means concatenation of two vectors.

For every node i in G_s , in a WGAT layer, all attention coefficients of their neighbors can be computed as (6). To utilize these attention coefficients, a linear combination for the corresponding neighbors is applied to update the features of the nodes.

$$\mathbf{x}'_i = \sigma \left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} \mathbf{W} \mathbf{x}_j \right), \quad (7)$$

where σ is a non-linearity unit and in our experiments, we use the ReLU [20].

As suggested in previous work [32, 33], the multi-head attention can help to stabilize the training of the self-attention layers. Therefore, we apply the multi-head setting for our WGAT.

$$\mathbf{x}'_i = \parallel_{k=1}^K \sigma \left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^k \mathbf{W}^k \mathbf{x}_j \right), \quad (8)$$

where K is the number of heads and for every head, there is a different set of parameters. \parallel in (8) stands for the concatenation of all heads. As a result, after the calculation of (8), $\mathbf{x}'_i \in \mathbb{R}^{Kd'}$.

Specifically, if we stack multiple WGAT layers, the final nodes feature will be shaped as $\mathbb{R}^{Kd'}$ as well. However, what we expect is $\mathbb{R}^{d'}$. Consequently, we calculate the mean over all heads of the attention results.

$$\mathbf{x}'_i = \sigma \left(\frac{1}{K} \sum_{k=1}^K \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^k \mathbf{W}^k \mathbf{x}_j \right). \quad (9)$$

Once the forward propagation of multiple WGAT layers has finished, we obtain the final feature vector of all nodes, which is the item level embeddings. These embeddings will serve as the input of the session embedding computation stage that we detail below.

4.4 Readout Function

A Readout function aims to give out a representation of the whole graph based on the node features after the forward computation of the GNN layers. As we introduce above, the Readout function needs to learn the order of the item transition pattern to avoid the bias of the time order and the inaccuracy of the self-attention on the last input item. For the convenience, some algorithms use simple permutation invariant operations for example, *Mean*, *Max* or *Sum* over all node features. Though clearly, the methods mentioned above are simple and perfectly do not violate the constraints of the permutation invariance, they can not provide a sufficient model capacity for learning a representative session embedding for the session graph. In contrast, Set2Set [34] is a graph level feature extractor which learns a query vector indicating the order of reading from the memory for an undirected graph. We modify this method to suit the setting of the session graphs. The computation procedures are as follows:

$$\mathbf{q}_t = \text{GRU}(\mathbf{q}_{t-1}^*), \quad (10)$$

$$e_{i,t} = f(\mathbf{x}_i, \mathbf{q}_t), \quad (11)$$

$$a_{i,t} = \frac{\exp(e_{i,t})}{\sum_j \exp(e_{j,t})}, \quad (12)$$

$$\mathbf{r}_t = \sum_i a_{i,t} \mathbf{x}_i, \quad (13)$$

$$\mathbf{q}_t^* = \mathbf{q}_t \parallel \mathbf{r}_t, \quad (14)$$

where i indexes node i in the session graph G_s , $\mathbf{q}_t, \mathbf{q}_t^* \in \mathbb{R}^d$, is a query vector which can be seen as the order to read $\mathbf{r}_t \in \mathbb{R}^d$ from the memory and GRU is the gated recurrent unit, which at the first step takes no input and at the following steps, takes the former output $\mathbf{q}_{t-1}^* \in \mathbb{R}^{2d}$. f calculates the attention coefficient $e_{i,t}$ between the embedding of every node \mathbf{x}_i and the query vector \mathbf{q}_t . $a_{i,t}$ is the probability form of $e_{i,t}$ after applying a softmax function over $e_{i,t}$, which is then used to a linear combination on the node embeddings \mathbf{x}_i . The final output \mathbf{q}_t^* of one forward computation of the Readout function is the concatenation of \mathbf{q}_t and \mathbf{r}_t .

Based on all node embeddings for a session graph, we use Equation 10~14 to obtain a graph level embedding which contains a query vector \mathbf{q}_t in addition to the semantic embedding vector \mathbf{r}_t . The query vector \mathbf{q}_t controls what to read from the node embeddings, which actually provides an order to process all nodes if we recursively apply the Readout function.

4.5 Recommendation

Once the graph level embedding \mathbf{q}_t^* is obtained, we can use it to make a recommendation by computing a score vector $\hat{\mathbf{z}}$ for every item over the whole item set \mathcal{V} with their initial embeddings in the matrix form,

$$\hat{\mathbf{z}} = (\mathbf{W}_{out} \mathbf{q}_t^*)^T \mathbf{X}^0, \quad (15)$$

where $\mathbf{W}_{out} \in \mathbb{R}^{d \times 2d}$ is a parameter that performs a linear mapping on the graph embedding \mathbf{q}_t^* , the T means the transformation on a matrix, and \mathbf{X}^0 is from Equation 3.

For every item in the item set \mathcal{V} , we can calculate a recommendation score and combine them together, we obtain a score vector $\hat{\mathbf{z}}$. Furthermore, we apply a softmax function over $\hat{\mathbf{z}}$ to transform it into the probability distribution form $\hat{\mathbf{y}}$,

$$\hat{\mathbf{y}} = \text{softmax}(\hat{\mathbf{z}}). \quad (16)$$

For the top-K recommendation, it is simple to choose the highest K probabilities over all items based on $\hat{\mathbf{y}}$.

4.6 Objective Function

Since we already have the recommendation probability of a session, we can use the label item v_{label} to train our model with the supervised learning method.

As mentioned above, we formulate the recommendation as a graph level classification problem. Consequently, we apply the multi-class cross entropy loss between $\hat{\mathbf{y}}$ and the one-hot encoding of v_{label} as the objective function. For a batch of training sessions, we can have

$$L = - \sum_{i=1}^l \text{one-hot}(v_{label,i}) \log(\hat{\mathbf{y}}_i), \quad (17)$$

where l is the batch size we use in the optimizer.

In the end, we use the Back-Propagation Through Time (BPTT) algorithm to train the whole FGNN model.

Table 1: Statistic details of datasets.

Dataset	all the clicks	train sessions	test sessions	all the items	avg.length
Yoochoose1/64	557248	369859	55898	16766	6.16
Yoochoose1/4	8326407	5917746	55898	29618	5.71
Diginetica	982961	719470	60858	43097	5.12

5 EXPERIMENTS

In this section, we conduct experiments with the purpose to prove the efficacy of our proposed FGNN model by answering the following research questions:

- **RQ1** Does the FGNN outperform other state-of-the-art SBRS methods? (in Section 5.5)
- **RQ2** How does the WGAT work for the session-based recommendation problem? (in Section 5.6)
- **RQ3** How does the Readout function work differently from other graph level embedding methods? (in Section 5.7)

In the following, we first describe the details of the basic setting of the experiments and afterwards, we answer the questions above by showing the results.

5.1 Datasets

We choose two representative benchmark e-commerce datasets, i.e., *Yoochoose* and *Diginetica*, to evaluate our model.

- *Yoochoose* is used as a challenge dataset for RecSys Challenge 2015¹. It is obtained by recording click-streams from an E-commerce website within 6 months.
- *Diginetica* is used as a challenge dataset for CIKM cup 2016². It contains the transaction data which is suitable for session-based recommendation.

For the fairness and the convenience of comparison, we follow [16, 19, 38] to filter out sessions of length 1 and items which occur less than 5 times in each dataset respectively. After the preprocessing step, there are 7,981,580 sessions and 37,483 items remaining in *Yoochoose* dataset, while 204,771 sessions and 43097 items in *Diginetica* dataset. Similar to [30, 38], we split a session of length n into $n - 1$ partial sessions of length ranging from 2 to n to augment the datasets. For the partial session of length i in the session S , it is defined as $[v_{s,0}, \dots, v_{s,i-1}]$ with the last item $v_{s,i-1}$ as v_{label} . Following [16, 19, 38], for the *Yoochoose* dataset, the most recent portions 1/64 and 1/4 of the training sequence are used as two split datasets respectively. Statistical details of all datasets are shown in Table 5.1.

5.2 Baselines

In order to prove the advantage of our proposed FGNN model, we compare FGNN with the following representative methods:

- **POP** always recommends the most popular items in the whole training set, which serves as a strong baseline in some situations although it is simple.
- **S-POP** always recommends the most popular items for the current session.

- **Item-KNN** [24] computes the similarity of items by the cosine distance of two item vectors in sessions. Regularization is also introduced to avoid the rare high similarities for unvisited items.
- **BPR-MF** [22] proposes a BPR objective function which calculates a pairwise ranking loss. Following [16], Matrix Factorization is modified to session-based recommendation by using mean latent vectors of items in a session.
- **FPMC** [23] is a hybrid model for the next-basket recommendation and it achieves state-of-the-art results. For anonymous session-based recommendation, following [16], we omit the user feature directly because of the unavailability.
- **GRU4REC** [9] stacks multiple GRU layers to encode the session sequence into a final state. It also applies a ranking loss to train the model.
- **NARM** [16] extends to use an attention layer to combine encoded states of RNN, which enables the model to explicitly emphasize on the more important parts of the input.
- **STAMP** [19] uses attention layers to replace all RNN encoders in previous work to even make the model more powerful by fully relying on the self-attention of the last item in a sequence.
- **SR-GNN** [38] applies a gated graph convolutional layer [18] to obtain item embeddings, followed by a self-attention of the last item as **STAMP** does to compute the sequence level embeddings.

5.3 Evaluation Metrics

At a time, a recommender system can give out a few recommended items and a user would choose the first few of them. To keep the same setting as previous baselines, we mainly choose to use top-20 items to evaluate a recommender system and specifically, two metrics, i.e., **P@20** and **MRR@20**. For more detailed comparison, top-5 and top-10 results are considered as well.

- **P@K** (Precision calculated over top-K items). The P@K score is the primary metric that calculates the proportion of test cases which recommend the correct item in a top K position in a ranking list,

$$P@K = \frac{n_{hit}}{N}, \quad (18)$$

where N represents the number of test sequences S_{test} in the dataset and n_{hit} counts the number that the desired items are in the top K position in the ranking list, which is named the *hit*.

- **MRR@K** (Mean Reciprocal Rank calculated over top-K items). The reciprocal is set to 0 when the desired items are not in

¹<https://2015.recsyschallenge.com/challenge.html>

²<http://cikm2016.cs.iupui.edu/cikm-cup/>

the top K position and the calculation is as follows,

$$\text{MRR@K} = \frac{1}{N} \sum_{v_{label} \in S_{test}} \frac{1}{\text{Rank}(v_{label})}. \quad (19)$$

The MRR is a normalized ranking of *hit*, the higher the score, the better the quality of the recommendation because it indicates a higher ranking position of the desired item.

5.4 Experiments Setting

We apply a three layer WGAT and each with eight heads as our node representation encoder and a three processing steps of our Readout function. The size of the feature vectors of the items are set to 100 for every layer including the initial embedding layer. All parameters of the FGNN are initialized using a Gaussian distribution with a mean of 0 and a standard deviation of 0.1 except for the GRU unit in the Readout function, which is initialized using the orthogonal initialization [25] because of its performance on RNN-like units. We use the Adam optimizer with the initial learning rate $1e-3$ and the linear schedule decay rate of 0.1 for every 3 epochs. The batch size for mini-batch optimization is 100 and we set an L2 regularization to $1e-5$ to avoid overfitting.

5.5 Comparison with Baseline Methods (RQ1)

To demonstrate the overall performance of FGNN, we compare it with the baseline methods mentioned in Section 5.2 by evaluating the P@20 and MRR@20 scores. The overall results are presented in Table 2 with respect to all baseline methods and our proposed FGNN model. Due to the insufficient memory of hardware, we can not initialize FPMC on *Yoochoose1/4* as [16], which is not reported in Table 2. For more detailed comparisons, in Table 3, we present the results of the most recent state-of-the-art methods for the dataset *Yoochoose1/64* when $K = 5$ and 10.

5.5.1 General Comparison by P20 and MRR20. FGNN utilizes the multi layers of WGAT to easily convey the semantic and structural information between items within the session graph and applies the Readout function to decide the relative significance as the order of nodes in the graph to make the recommendation. According to the results reported in Table 2, obviously, the proposed FGNN model outperforms all the baseline methods on all three datasets for both metrics, P@20 and MRR@20. It is proved that our method achieves state-of-the-art performance on benchmark datasets. We also substitute the two key components, WGAT and the Readout function, with gated graph networks (FGNN-Gated) and the self-attention (FGNN-ATT-OUT) used by previous methods. Both of the variants perform better than previous models, which demonstrates the efficacy of the proposed WGAT and the Readout function respectively.

Compared with those traditional algorithms, e.g., POP and S-POP, because of their simple intuition to recommend items based on the frequency of appearance, they perform far worse than FGNN. They tend to recommend fixed items, which leads to the failure of capturing the characteristics of different items and sessions. Taking BPR-MF and FPMC into consideration, which omits the session setting when recommending items, we can see that S-POP can defeat these methods as well because S-POP makes use of the session context information. Item-KNN achieves the best results among

Table 2: Performance compared with other baselines.

Method	<i>Yoochoose1/64</i>		<i>Yoochoose1/4</i>		<i>Diginetica</i>	
	P@20	MRR@20	P@20	MRR@20	P@20	MRR@20
POP	6.71	1.65	1.33	0.30	0.89	0.20
S-POP	30.44	18.35	27.08	17.75	21.06	13.68
Item-KNN	51.60	21.81	52.31	21.70	35.75	11.57
BPR-MF	31.31	12.08	3.40	1.57	5.24	1.98
FPMC	45.62	15.01	-	-	26.53	6.95
GRU4REC	60.64	22.89	59.53	22.60	29.45	8.33
NARM	68.32	28.63	69.73	29.23	49.70	16.17
STAMP	68.74	29.67	70.44	30.00	45.64	14.32
SR-GNN	70.57	30.94	71.36	31.89	50.73	17.59
(ours)						
FGNN-Gated	70.85	31.05	71.5	32.17	51.03	17.86
FGNN-ATT-OUT	70.74	31.16	71.68	32.26	50.97	18.02
FGNN	71.12	31.68	71.97	32.54	51.36	18.47

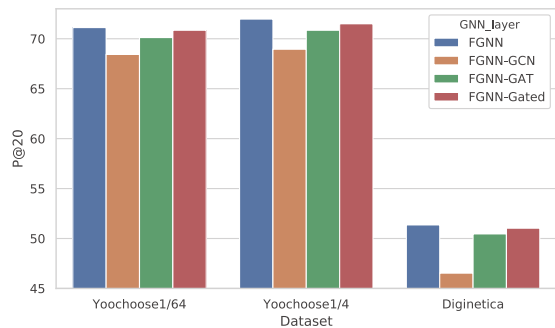
Table 3: Performance when $K = 5$ and 10 for *Yoochoose1/64*.

Method	<i>Yoochoose1/64</i>			
	P@5	MRR@5	P@10	MRR@10
NARM	44.34	26.21	57.50	27.97
STAMP	45.69	27.26	58.07	28.92
SR-GNN	47.42	28.41	60.21	30.13
FGNN	48.23	29.16	60.97	30.85

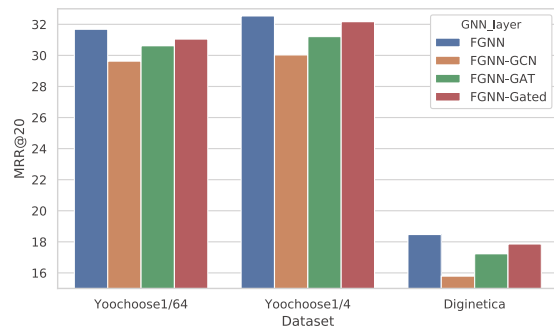
the traditional methods, though it only calculates the similarity between items without considering sequential information. At the even worse situation when the dataset is large, methods relying on the whole item set undoubtedly fail to scale well. All methods above achieve relatively poor results compared with the recent neural-network-based methods, which fully model the user’s preference in the session sequence.

Different from the traditional methods mentioned above, all baselines using neural networks achieve a large performance margin. GRU4REC is the first to apply RNN-like units to encode the session sequence. It sets the baseline of neural-network-based methods. Though RNN is perfectly matched for sequence modeling, session-based recommendation problems are not merely a sequence modeling task because the user’s preference is even changing within the session. RNN takes every input item equally importantly, which introduces bias to the model during training. For the subsequent methods, NARM and STAMP, which both incorporate a self-attention over the last input item of a session, they both outperform GRU4REC in a large margin. They both use the last input item as the representation of short-term user interest. It proves that assigning different attention on different inputs is a more accurate modeling method for session encoding. Looking into the comparison between NARM, combining RNN and attention mechanism, and STAMP, the complete attention setting, there is a conspicuous gap of performance that STAMP outperforms NARM. This further demonstrates that directly using RNN to encode the session sequence can inevitably introduce bias to the model, which the attention can completely avoid.

SR-GNN uses a session graph to represent the session sequence, followed by a gated graph layer to encode items. In the final stage, it again uses a self-attention the same as STAMP to output a session



(a) P@20 index.



(b) MRR@20 index.

Figure 3: Results with different GNN layers.

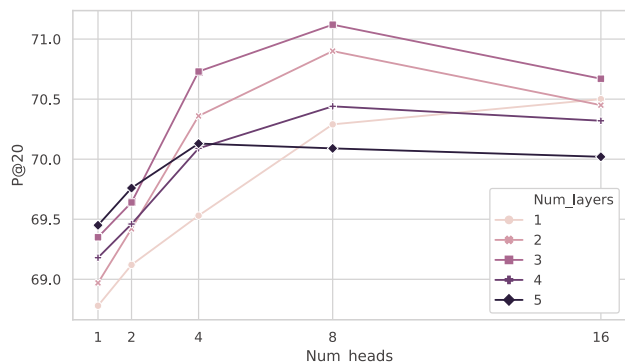


Figure 4: P@20 index for different number of layers and heads for WGAT.

embedding. It achieves the best result compared to all methods mentioned above. The graph structure is shown to be more suitable than the sequence structure, the RNN modeling, or a set structure, the attention modeling.

5.5.2 Higher Standard Recommendation with $K = 5, 10$. For more detailed results in Table 3, FGNN also achieves the best results with a higher standard of the top-5 and top-10 recommendation. The proposed FGNN model outperforms all baseline methods above. It has a more accurate node-level encoding tool, WGAT, to learn more representative features and a Readout function, to learn an inherent order of nodes in the graph to avoid the entire random order of items. According to the result, it is demonstrated that a more accurate session embedding is obtained by FGNN to make effective recommendations, which proves the efficacy of the proposed FGNN.

5.6 Comparison with Other GNN Layers (RQ2)

To efficiently convey information between items in a session graph, we propose to use WGAT, which suits the situation of the session better. As mentioned above, there are many different GNN layers that can be used to generate node embeddings, e.g., GCN [12], GAT [33] and gated graph networks [18, 38]. To prove the usefulness of WGAT, we substitute all three WGAT layers with GCN, GAT

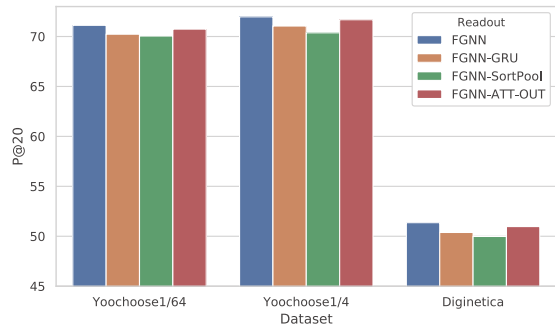
and gated graph networks respectively in our model. For GCN and GAT, they both initially work for unweighted and undirected graph, which is not the same setting as the proposed session graph. To make both of them work on the session graph, we directly convert the session graph into undirected by replacing the original directed edges with undirected ones, i.e., reverse the source node and target node of edges. And we simply omit the original weight of edges and set all connections between nodes with the same weight 1. For the other one, the Gated graph networks, it can work with the session graph setting in its original form without any modification on the session graph.

In Figure 3(a) and Figure 3(b), results of different GNN layers are shown with P@20 and MRR@20 indices. FGNN is the model proposed in this work, which achieves the best performance. WGAT is more powerful than other GNN layers in session-based recommendation. GCN and GAT are not able to capture the direction and the explicit weight of edges, resulting in performing worse than WGAT and gated graph networks, which holds the ability to capture these information. Between WGAT and gated graph networks, WGAT performs better because of the stronger ability of representation learning.

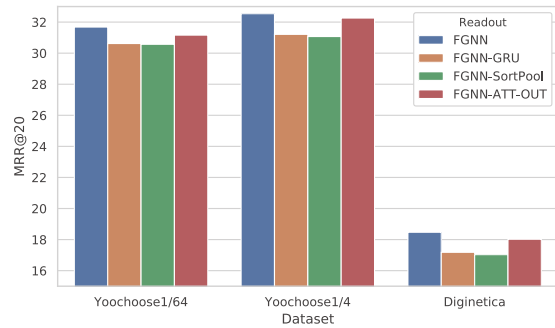
For the study of WGAT, we test how the number of layers and heads affect the P@20 index performance on *Yoochoose1/64*. In Figure 4, we report the experiment results of different number of layers ranging in $\{1, 2, 3, 4, 5\}$ and heads ranging in $\{1, 2, 4, 8, 16\}$. It shows that stacking three WGAT layers with eight heads performs the best. Lower results are shown for smaller models for the reason that the capacity of them is too low to represent the complexity of the item transition pattern. According to the tendency of results for larger models, it shows that it is difficult to train these models and the overfitting is harmful to the final performance.

5.7 Comparison with Other Graph Embedding Methods (RQ3)

Different approaches for generating the session embedding after obtaining the node embedding stand for different emphasis of the input item. The Readout function proposed in this work learns an inherent order of the nodes by the query vector, which indicates the relatively different impact on the user's preference along with the item transition. To prove the superiority of our Readout function,



(a) P@20 index.



(b) MRR@20 index.

Figure 5: Results with different aggregation functions.

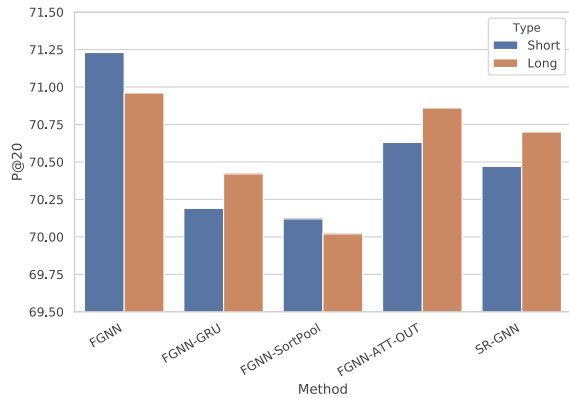


Figure 6: P@20 for Short and Long sessions with different aggregation functions and baselines.

we replace the Readout function with other session embedding generators:

- **FGNN-ATT-OUT** We apply the widely-used self-attention of the last input item. It directly considers the last input item as the short-term reference and all other items as the long-term reference.
- **FGNN-GRU** To compare the inherent order learned by our Readout function, we use GRU to directly make use of the input session sequence order.
- **FGNN-SortPool** SortPooling is introduced by Zhang et. al. [41] to perform a pooling on graph level by sorting the features of the nodes. This sorting can be viewed as a kind of order as well.

In Figure 5(a) and Figure 5(b), results of different methods for graph level embedding generation are presented for all the datasets with the P@20 and MRR@20 indices. It is obvious that the proposed Readout function achieves the best result. For FGNN-GRU and FGNN-SortPool, they both contain an order but which is too simple to capture the item transition pattern. FGNN-GRU uses GRU to encode the session sequence with the input order. Such a setting is similar to the RNN-based method. As a consequence, it performs worse than attention-based method FGNN-ATT-OUT, which takes

both the short-term and the long-term preference into consideration. As for FGNN-SortPool, it sorts the nodes based on WL colors from previous multiple layers of computations. Though it does not simply rely on the input order of the session sequence, the order used for the node is set according to the relative scale of the features. For the best performance, our Readout function learns the order of the item transition pattern, which is different from using the time order or the hand-crafted split of long-term and short-term preference. The results prove that there is a more accurate order for the model to make a more accurate recommendation.

For different session embedding generators, it is also important to look deep into how they perform on sessions with different lengths because the length varies greatly within one dataset. Following previous work [19, 38], sessions in *Yoochoose 1/64* are separated into two groups, i.e., **Short** and **Long**. **Short** indicates that the length of sessions is less than or equal to 5, while sessions longer than 5 are categorized as **Long**. Length 5 is the closest to the average length of total sessions. 70.1% of *Yoochoose1/64* are **Short** sessions and 29.9% are **Long**. In addition to different session embedding generators, we take the previous GNN-based baseline method SR-GNN into comparison. For each method, we report the results evaluated in terms of P@20 in Figure 6. In the aspect of both **Short** and **Long** sessions, FGNN achieves the best performance compared with other graph embedding generators and SR-GNN. The proposed Readout function shows superiority to other methods. The order introduced by the Readout function is demonstrated to convey more accurate information of item transition pattern. For the comparison among RNN-based and attention-based methods, it is shown that the performance is relatively better for longer sessions than shorter ones. This indicates that the item transition pattern relies on the latter input items of a sequence more heavily for long sessions, which is more suitable for these methods.

6 CONCLUSION

Session-based recommender system is a challenging problem because the user history is unavailable for predicting the user’s preference. This work proposes to use WGAT layers to learn item embeddings for items in a session, which are then processed by the Readout function to obtain the session embedding to represent

the user’s preference for this session. It is demonstrated by experiments that our proposed method achieves state-of-the-art results on benchmark e-commerce datasets. In the future, it is important and promising to make use of inter-session information to more accurately represent the user’s preference.

7 ACKNOWLEDGMENTS

This work is supported by ARC DP190101985, DP170103954, NSFC 61628206 and NSFC 61806039.

REFERENCES

- [1] Tong Chen, Hongzhi Yin, Hongxu Chen, Rui Yan, Quoc Viet Hung Nguyen, and Xue Li. 2019. AIR: Attentional Intention-Aware Recommender Systems. In *Proceedings of the 35th IEEE International Conference on Data Engineering, ICDE 2019*.
- [2] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014 workshop on Deep Learning, NIPS 2014*.
- [3] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*.
- [4] Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, Vol. 2. IEEE.
- [5] Lei Guo, Hongzhi Yin, Qinyong Wang, Tong Chen, Alexander Zhou, and Nguyen Quoc Viet Hung. 2019. Streaming Session-based Recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019*. ACM.
- [6] William L. Hamilton, Zitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, NIPS 2017*.
- [7] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017*.
- [8] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent Neural Networks with Top-k Gains for Session-based Recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018*.
- [9] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. In *4th International Conference on Learning Representations, ICLR 2016*.
- [10] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997).
- [11] Dietmar Jannach and Malte Ludewig. 2017. When Recurrent Neural Networks meet the Neighborhood for Session-Based Recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys 2017*.
- [12] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017*.
- [13] Yehuda Koren, Robert M. Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *IEEE Computer* 42, 8 (2009).
- [14] Jingjing Li, Mengmeng Jing, Ke Lu, Lei Zhu, Yang Yang, and Zi Huang. 2019. From Zero-Shot Learning to Cold-Start Recommendation. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*.
- [15] Jingjing Li, Ke Lu, Zi Huang, and Heng Tao Shen. 2019. On both Cold-Start and Long-Tail Recommendation with Social Data. *IEEE Transactions on Knowledge and Data Engineering* (2019).
- [16] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural Attentive Session-based Recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017*.
- [17] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. 2019. Graph Matching Networks for Learning the Similarity of Graph Structured Objects. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*.
- [18] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. 2016. Gated Graph Sequence Neural Networks. In *4th International Conference on Learning Representations, ICLR 2016*.
- [19] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. 2018. STAMP: Short-Term Attention/Memory Priority Model for Session-based Recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018*.
- [20] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning, ICML 2010*.
- [21] Michael J. Pazzani and Daniel Billsus. 2007. Content-Based Recommendation Systems. In *The Adaptive Web, Methods and Strategies of Web Personalization*.
- [22] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2009*.
- [23] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized Markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web, WWW 2010*.
- [24] Badrul Munir Sarwar, George Karypis, Joseph A. Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the Tenth International World Wide Web Conference, WWW 2010*.
- [25] Andrew M. Saxe, James L. McClelland, and Surya Ganguli. 2014. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *2nd International Conference on Learning Representations, ICLR 2014*.
- [26] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The Graph Neural Network Model. *IEEE Trans. Neural Networks* 20, 1 (2009).
- [27] J. Ben Schafer, Dan Frankowski, Jonathan L. Herlocker, and Shilad Sen. 2007. Collaborative Filtering Recommender Systems. In *The Adaptive Web, Methods and Strategies of Web Personalization*.
- [28] Guy Shani, Ronen I. Brafman, and David Heckerman. 2002. An MDP-based Recommender System. In *Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence, UAI 2002*.
- [29] Ke Sun, Tiejun Qian, Hongzhi Yin, Tong Chen, Yiqi Chen, and Ling Chen. 2019. What Can History Tell Us? Identifying Relevant Sessions for Next-Item Recommendation. In *Proceedings of the 29th ACM on Conference on Information and Knowledge Management, CIKM 2019*.
- [30] Yong Kiam Tan, Xinxing Xu, and Yong Liu. 2016. Improved Recurrent Neural Networks for Session-based Recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, DLRS@RecSys 2016*.
- [31] Trinh Xuan Tuan and Tu Minh Phuong. 2017. 3D Convolutional Networks for Session-based Recommendation with Content Features. In *Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys 2017*.
- [32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, NIPS 2017*.
- [33] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *6th International Conference on Learning Representations, ICLR 2018*.
- [34] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. 2016. Order Matters: Sequence to sequence for sets. In *4th International Conference on Learning Representations, ICLR 2016*.
- [35] Shoujin Wang, Longbing Cao, and Yan Wang. 2019. A Survey on Session-based Recommender Systems. *CoRR* abs/1902.04864 (2019).
- [36] Weiqing Wang, Hongzhi Yin, Shazia Wasim Sadiq, Ling Chen, Min Xie, and Xiaofang Zhou. 2016. SPORE: A sequential personalized spatial item recommender system. In *Proceedings of the 32nd IEEE International Conference on Data Engineering, ICDE 2016*. 954–965.
- [37] Chen Wu and Ming Yan. 2017. Session-aware Information Embedding for E-commerce Product Recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017*.
- [38] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. 2019. Session-based Recommendation with Graph Neural Networks. In *Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*.
- [39] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *5th International Conference on Learning Representations, ICLR 2019*.
- [40] Zitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018*.
- [41] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An End-to-End Deep Learning Architecture for Graph Classification. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), and the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18)*.
- [42] Andrew Zimdars, David Maxwell Chickering, and Christopher Meek. 2001. Using Temporal Data for Making Recommendations. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, UAI 2001*.