AutoGSR: Neural Architecture Search for Graph-based Session Recommendation

Jingfan Chen
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China
jingfan.chen@smail.nju.edu.cn

Guanghui Zhu*
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China
zgh@nju.edu.cn

Haojun Hou
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China
haojunhou@smail.nju.edu.cn

Chunfeng Yuan
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China
cfyuan@nju.edu.cn

Yihua Huang
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China
yhuang@nju.edu.cn

ABSTRACT

Session-based recommendation aims to predict next click action (e.g., item) of anonymous users based on a fixed number of previous actions. Recently, Graph Neural Networks (GNNs) have shown superior performance in various applications. Inspired by the success of GNNs, tremendous endeavors have been devoted to introduce GNNs into session-based recommendation and have achieved significant results. Nevertheless, due to the highly diverse types of potential information in sessions, existing GNNs-based methods perform differently on different session datasets, leading to the need for efficient design of neural networks adapted to various session recommendation scenarios. To address this problem, we propose Automated neural architecture search for Graph-based Session Recommendation, namely AutoGSR, a framework that provides a practical and general solution to automatically find the optimal GNNs-based session recommendation model. In AutoGSR, we propose two novel GNN operations to build an expressive and compact search space. Building upon the search space, we employ a differentiable search algorithm to search for the optimal graph neural architecture. Furthermore, to consider all types of session information together, we propose to learn the item meta knowledge, which acts as a priori knowledge for guiding the optimization of final session representations. Comprehensive experiments on three real-world datasets demonstrate that AutoGSR is able to find effective neural architectures and achieve state-of-the-art results. To the best of our knowledge, we are the first to study the neural architecture search for the session-based recommendation.

*Corresponding author.

SIGIR '22, July 11–15, 2022, Madrid, Spain

ACM ISBN 978-1-4503-8732-3/22/07…$15.00
https://doi.org/10.1145/3477495.3531940

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

neural architecture search; session-based recommendation; graph neural networks; meta learning

ACM Reference Format:
Jingfan Chen, Guanghui Zhu, Haojun Hou, Chunfeng Yuan, and Yihua Huang. 2022. AutoGSR: Neural Architecture Search for Graph-based Session Recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22), July 11–15, 2022, Madrid, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3477495.3531940

1 INTRODUCTION

Recommender systems play an increasingly crucial role as they provide accurate and useful content to address the information overload problem. In many real-world scenes, the user's identity information may be unknown and only their ongoing click actions, namely session, is available. To recommend next click based on the limited interactions from the current session, session-based recommender systems have been proposed and have attracted researchers' great attention.

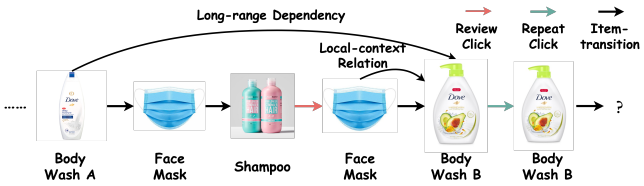


Figure 1: Different types of information within a session.

Many effective methods for session-based recommendation have been developed over the past few years. Earlier methods focused on modeling local item-transitions, most of which were based on Markov chains or Recurrent Neural Networks (RNNs) [13, 27, 29].

Recently, Graph Neural Networks (GNNs) have received increasing attention and have become state-of-the-art methods for many graph-based tasks [8–10, 36]. Inspired by the great success of GNNs, many works have been proposed to convert sessions into graphs and to apply GNNs to session-based recommendation [4, 23, 33]. The constructed session graphs and the proposed GNNs effectively capture certain types of information about the session, which indicate the underlying user preferences. Specifically, as shown in Figure 1, existing GNNs-based methods mainly consider three types of information in a session: (1) **local-context (i.e., sequential item-transitions)** [4] information records the correlation between temporally adjacent items, (2) **long-range dependency** [23] information records the correlation between distant items within a session, and (3) **relational semantics** [33] records different types of item-transition semantics, such as *repeat click* and *review click*.

However, existing methods only consider limited amount of the above types of session information, and do not consider all of them together. The problem is that different session data contain different proportions of the above information, and directly transferring the recommender system over different session data may degrade the recommendation performance. For instance, there may be a strong semantic correlation between two items, even though they are far apart in the session. Then, a recommender system that focuses only on local-context information may ignore such long-range dependency, especially when dealing with long session sequences, which leads to sub-optimal recommendation performance.

Thus, the disparity of session datasets makes it necessary to comprehensively design neural networks-based recommender systems for different session recommendation scenarios. However, designing such neural networks manually requires substantial domain knowledge and expert effort, which is time-consuming and does not guarantee the optimality. Inspired by the recent success of Neural Architecture Search (NAS) [11, 45] which aims to automatically find the optimal network architecture from a given search space, we consider the application of NAS techniques to GNNs-based session recommendation.

There are two major challenges from different levels when we apply NAS to the studied problem: (1) How to construct a compact and expressive search space (**network architecture level**). Existing GNNs-based models consider only a limited type of session information. Moreover, even the combination of these models still does not cover all possible types of session information. Thus, it is not sufficient to construct a search space that contains only these models. (2) How to train the searched network so that it can take into account all types of the potential information in a session (**network parameter level**). Using NAS techniques, a sparse network can be obtained by discretizing from a dense supernet that is adapted to the given session data. However, there may be deviations between the sparse network and the learned dense supernet. Training such a sparse network may result in item representations that loose or overemphasize certain types of session information without considering all types of session information together.

To address the above two challenges, we integrate our solution into a NAS framework, namely AutoGSR, for **G**raph-based **S**ession **R**ecommendation. The whole AutoGSR framework is designed as a two-stage training pipeline. In the first stage, AutoGSR aims to find an optimal architecture from the search space. To construct an expressive and compact search space covering all types of information (i.e., long-range dependencies, local-context relations, and various item-transition semantics), we propose two novel GNN operations to complement the search space. On the basis of the designed search space, we search for architectures by using a differentiable method. The learned supernet can be effectively adapted to the given session data. In the second stage, AutoGSR aims to learn item representations by retraining the obtained sparse network architectures. To prevent the final item representations from losing potential session information caused by the biases in the sparse architecture, we propose an item meta knowledge learner that learns item meta knowledge from session data beforehand. The item meta knowledge will be served as a priori knowledge for retraining sparse networks.

The main contributions of this paper are as follows:

- We propose an automatic architecture search framework, namely AutoGSR, for searching the optimal graph neural network architecture on session-based recommendation. To the best of our knowledge, it is the first searching framework for modeling GNNs-based session recommendation.
- We propose two novel GNN operations to complement and obtain a compact and expressive search space, which covers all types of session information.
- We propose an item meta knowledge learner to learn the item meta knowledge, which is served as the priori knowledge to guide the learning of the final session representations.
- Comprehensive experiments on three real-world datasets are conducted to evaluate the effectiveness of AutoGSR. The results demonstrate that our AutoGSR is able to find effective network architectures and achieves state-of-the-art session recommendation performance.

2 RELATED WORK

2.1 Session-based Recommendation

Session-based recommendation (SBR) aims to predict the next item based on anonymous behavior sequences in chronological order. Previously, Markov Chain-based methods and Recurrent Neural Networks (RNNs)-based SBRs have been adopted that the next item is inferred based on the previous one [27, 29]. For example, GRU4REC [13] is the first model that adapts RNNs to session-based recommendation. With the boom of attention modeling, many methods apply the attention mechanism to model the importance of items in a session [5, 15, 21, 39]. For example, STAMP [21] developed an attention-based short-term memory network that captures user preferences by exploiting both long and short-term memories of a session.

Recent developments of Graph Neural Networks (GNNs) have also improved the performance of SBR [4, 23, 33, 37, 38]. GNNs-based methods first convert sessions into session graphs, and then learn high-order item-transitions on session graphs. For instance, SR-GNN [37] is the first work that adapts GNNs to SBR. It constructs directed session graphs and propose to apply gated GNN models to aggregate information among items. GC-SAN [38] constructs session graphs dynamically and employs self-attention networks on session graphs. LESSR [4] constructs edge-order preserving session graphs and shortcut session graphs, and applies GRU and graph

attention layers to aggregate item information, respectively. Unlike the previous methods, GCE-GNN [33] considers the heterogeneity between inter-item transitions and proposes to construct the global item graph and local session graphs to aggregate item representations based on heterogeneous graph convolutions as well as self-attention mechanism.

As the GNNs-based methods usually outperform other neural networks-based methods, in this paper, we construct the search space based on the GNNs-based models.

2.2 Neural Architecture Search

Neural Architecture Search (NAS) is a general solution in automating neural network design and has been widely used in the field of Computer Vision, Natural Language Processing and Graph Learning [26, 34, 40]. It can search better neural architectures comparing to experts-designed ones [24, 25]. The existing NAS literature can be divided into two categories: the query-based methods and the gradient-based methods. The former includes algorithms based on heuristic search such as reinforcement learning [24], Bayesian optimization [14] and evolutionary algorithms [7]. The latter includes gradient-based search strategies that first construct a hyperparameterized network (namely supernet) and then optimize the supernet using gradient descent based on the continuous relaxation of the search space, and finally discretize the supernet to obtain sparse architectures for downstream tasks. Since the dense supernet covers all candidate operations, the discretization process generates a performance gap between the obtained sparse network and the supernet.

The success in various domains has led to the use of NAS in recommendation systems [2, 16, 19, 35, 42–44, 46]. For instance, several works introduce the NAS to the CTR prediction task [19, 35, 46] to automate the interactions between various features. AdaRec [2] proposes to perform knowledge distillation to compress the deep sequential recommendation models based on NAS. Unlike the previous methods, our proposed searching framework aims to search the network architecture designed for session-based recommendation.

3 PRELIMINARIES

3.1 Problem Statement

Let $I = \{v_1, v_2, ..., v_N\}$ denotes all unique items in all sessions. Each session is represented as a sequence $s = [v_1^s, v_2^s, ..., v_t^s, ..., v_{l_s}^s]$ in a chronological order, where $v_t^s \in I$ $(1 \le t \le l_s)$ represents an interacted item of an anonymous user. l_s denotes the length of the session. Given a session s, the goal of session-based recommendation is to predict the item that will be clicked at the next timestamp (i.e., $v_{l_s+1}^s$). The output of session-based recommendation model is a list $\hat{\boldsymbol{y}} = [\hat{y}_1, \hat{y}_2, ..., \hat{y}_i, ..., \hat{y}_N]$, where $\hat{y}_i$ $(1 \le i \le N)$ is the corresponding predicted probability of item v_i to be clicked. The top-K items $(1 \le K \le N)$ with the highest probabilities in the prediction vector $\boldsymbol{y}$ will be recommended. The goal of NAS for graph-based session recommendation is to find the architecture with the best performance on validation dataset from the training dataset in a given search space:

$$\begin{aligned} \boldsymbol{\alpha}^* &= \arg\min_{\boldsymbol{\alpha} \in \mathcal{A}} \mathcal{L}_{val}(\boldsymbol{\alpha}, \boldsymbol{\omega}^*(\boldsymbol{\alpha})) \\ \text{s.t.} \quad \boldsymbol{\omega}^*(\boldsymbol{\alpha}) &= \arg\min_{\boldsymbol{\omega}} \mathcal{L}_{\text{train}}(\boldsymbol{\alpha}, \boldsymbol{\omega}), \end{aligned} \tag{1}$$

where $\mathcal{L}_{\text{train}}$ and $\mathcal{L}_{\text{val}}$ denote loss function on training dataset and validation dataset, respectively. $\boldsymbol{\omega}$ denotes network weight parameters and $\boldsymbol{\alpha}$ denotes the architecture parameters. $\mathcal{A}$ denotes the architecture search space.

3.2 Graph Neural Networks

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, where $\mathcal{V}, \mathcal{E}$ denote the nodes and the edges, respectively. Each node $x_i \in \mathcal{V}$ is associated with a node feature $\boldsymbol{x}_i$. In general, GNNs can be unified by a message passing schema. Each layer of GNNs performs an iteration of aggregation, such that the node receives the messages (i.e., feature representations) from its neighborhoods and then updates its representation by aggregating all messages. Given a l layer GNN, the node representation is able to aggregates the information within its l-hop neighborhoods. The $(l+1)$-th layer of GNN can be formulated as follows:

$$\begin{aligned} \boldsymbol{x}_i^{l+1} &= f_{\text{upd}}^l \left(\boldsymbol{x}_i^l, \text{agg}_i^l \right) \\ \text{agg}_i^l &= f_{\text{agg}}^l \left(\left\{ f_{\text{msg}}^l \left(\boldsymbol{x}_i^l, \boldsymbol{x}_j^l \right) : (j,i) \in \mathcal{E}_{\text{in}}(i) \right\} \right) \end{aligned} \tag{2}$$

where $\boldsymbol{x}_i^l$ is the representation of node x_i at layer l. $\mathcal{E}_{in}(i)$ denotes a set of incoming edges of node v_i. f_{msg} denotes the message function which computes the message representation along the edges. f_{agg} denotes the aggregation function which aggregates the representations passed to the target node. f_{upd} denotes the update function which updates the representation of the target node based on the ego and the aggregated representations.

To obtain the representation $\boldsymbol{x}_{\mathcal{G}}$ of the entire graph, given a L layer GNN, a readout function f_{out} aggregates the representations of all nodes in the L-th layer:

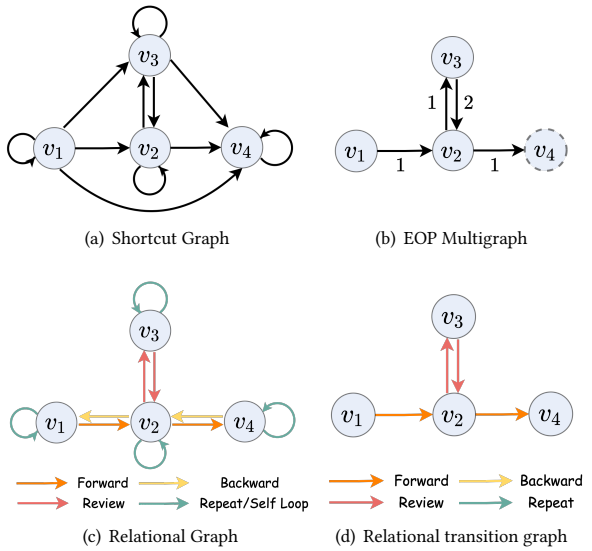
$$\boldsymbol{x}_{\mathcal{G}} = f_{\text{out}} \left(\left\{ \boldsymbol{x}_i^L : i \in \mathcal{V} \right\} \right) \tag{3}$$

3.3 Session Graph Models

In general, the GNNs-based session recommendation method first converts sessions into session graphs which record the relational dependencies between items, and then embeds each item $v \in I$ of the session s and the whole session into the same space for representation learning. Specifically, Each session $s = [v_1^s, v_2^s, ..., v_{l_s}^s]$ is represented as a session graph $\mathcal{G}_s = \{\mathcal{V}_s, \mathcal{E}_s\}$, where $\mathcal{V}_s = \{x_1^s, x_2^s, ..., x_m^s\} \subseteq I$ denotes the the set of nodes of the session graph and $\mathcal{E}_s$ denotes the set of edges of the session graph. $\boldsymbol{X}_s = \{\boldsymbol{x}_1^s, \boldsymbol{x}_2^s, ..., \boldsymbol{x}_m^s\} \subseteq \boldsymbol{X}$ denotes the set of item representations within the session s (we omit the session notation s for brevity if the context is clear). Note that $m \le l_s$ as there may be duplicate items in the session s. In the literatures of GNNs-based session recommendation, the methods that convert sessions into graphs can be divided into two categories: homogeneous and heterogeneous, depending on whether the semantic discrepancy of the item-transition is captured. The former applies the same message function to all edges as the edges are homogeneous. Such homogeneous session graphs differ

Table 1: Search space for session graph aggregators. The bold font represents the proposed session graphs and the corresponding aggregators. "Edge type" presents whether the model disentangles the various semantic types of the item-transitions.

Session graph	Aggregation function	Edge type	Local-context	Long-range dependencies
EOP Multigraph	EOPA [4]	Homogeneous	✓	
Shortcut Graph	SGAT [4]	Homogeneous		✓
Relational Graph	Relational GAT [33]	Heterogeneous		✓
Relational transition Graph	**Relational GGNN**	Heterogeneous	✓	
Mixup Graph	**Mixup**	Heterogeneous	✓	✓



(a) Shortcut Graph (b) EOP Multigraph (c) Relational Graph (d) Relational transition graph

Figure 2: Session graphs for session $[v_1, v_2, v_3, v_2, v_4]$. The dotted node represents the last item in the session and the number represents the edge order.

in edge direction or edge weight, which encodes different inter-item relation information of the session. For instance, as shown in Figure 2(b), LESSR [4] constructs the edge-order preserving (EOP) multigraph which records lossless item-transition information. In addition, they also consider the long-range dependencies and propose the shortcut graph as shown in Figure 2(a). In contrast, the latter explicitly disentangles the semantics of sequential transitions between items and depicts different transition types for them. They apply different message functions to different edge types as the edges are heterogeneous. For instance, as shown in Figure 2(c), the session graph constructed by GCE-GNN [33] encodes four types of item-transitions including repeat clicks, review clicks, forward clicks, and backward clicks. Finally, we list the above three session graphs from recent advanced models in Table 1.

Given the session graphs, GNNs will be applied on them to learn the item representations $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N]$ and session representations $\boldsymbol{x}_s$. To predict the item to be clicked in a session s, following [12], we utilize the *softmax* function to obtain the probabilities distribution vector $\hat{\boldsymbol{y}}$ between the session representation $\boldsymbol{x}_s$ with all normalized item representations in $\boldsymbol{X}$. To train the model, the cross-entropy loss is adopted as the optimization objective:

$$\mathcal{L} = \sum_{i=1} \boldsymbol{y}_i \log(\hat{\boldsymbol{y}}_i) \tag{4}$$

where $\boldsymbol{y}$ denotes the one-hot vector of the ground truth item.

4 METHODOLOGIES

4.1 Item Representation Learning Layer

To capture sequential information in a session, the existing method LESSR [4] proposes to convert a session into a directed EOP multigraph that preserves the *strict* item-transition order and then performs GRU on it. However, it ignores the semantic discrepancy between different item-transition types. Moreover, strict order may be so detailed that it may capture noisy item-transition information [3], e.g., a user simply compares two items in a session without any sequential intention.

To address this problem, we first explicitly distinguish between the types of item-transitions in a session. Then, to capture the high-level ordering information and mitigate the strict sequential noise, we propose to convert the session into a semantically disentangled directed session graph, namely *relational transition graph* (see Figure 2(d)). Motivated by SR-GNN [37], we propose to utilize the relational GGNN which aggregates the disentangled information passed from different edge types using GGNN [18]. Specifically,

$$\begin{aligned} \boldsymbol{h}_i^l &= \sum_{j \in \mathcal{N}(i)} \boldsymbol{W}_{r_{ij}} \boldsymbol{x}_j^l \\ \boldsymbol{x}_i^{l+1} &= \text{GRU}\left(\boldsymbol{h}_i^l, \boldsymbol{x}_i^l\right) \end{aligned} \tag{5}$$

where $\boldsymbol{x}_i^l$ denotes the embedding of node x_i in the session graph at layer l. $\mathcal{N}(i)$ denotes the neighbors of the node x_i within the session graph $\mathcal{G}_s$. $\boldsymbol{W}_{r_{ij}}$ denotes the trainable weights associated with the relation type r_{ij} of edge (i, j).

Furthermore, to obtain both the sequential information and the long-range inter-item dependencies, we propose the Mixup model, which combines the relational GGNN with the relational GAT [31]. Formally,

$$\begin{aligned} \boldsymbol{x}_{i,\text{Mixup}}^l &= \boldsymbol{g} \odot \boldsymbol{x}_{i,1}^l + (1 - \boldsymbol{g}) \odot \boldsymbol{x}_{i,2}^l \\ \boldsymbol{g} &= \sigma\left(\boldsymbol{W}_g \left[\boldsymbol{x}_{i,1}^l; \boldsymbol{x}_{i,2}^l\right]\right) \end{aligned} \tag{6}$$

where $\boldsymbol{x}_{i,1}^l$ and $\boldsymbol{x}_{i,2}^l$ denote the node embedding of l-th layer of relation GGNN and relation GAT, respectively. σ is the sigmoid function. [;] is the concatenation operation. $\boldsymbol{W}_g$ is the trainable weights. $\boldsymbol{g}$ is used to control the weights of two models.

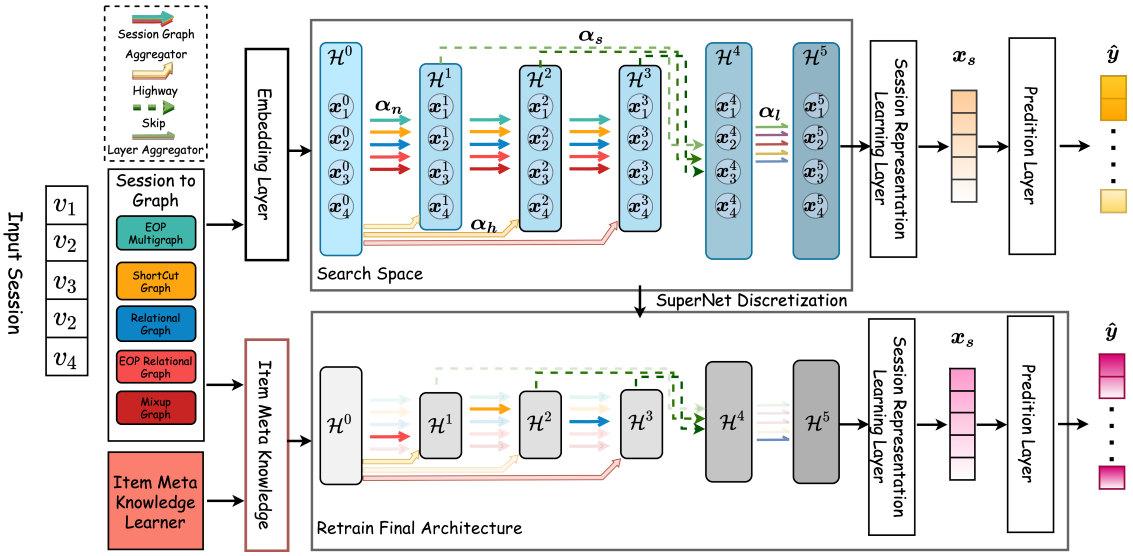


Figure 3: Framework Overview of AutoGSR. The session representation learning layer is consistent with GCE-GNN [33].

4.2 Search Space for Architecture Construction

Existing empirical studies have shown that designing a compact and expressive search space is crucial for NAS methods [41]. Specifically, for GNNs-based session recommendation, the expressiveness depends on various aggregation functions. Thus, we investigate two key important components to construct the search space: session graph aggregators and layer aggregators.

4.2.1 Session Graph Aggregators. To make the search space of the session graph aggregators cover all of the session information, we select five graph aggregators based on the following two criteria: (1) the reported recommendation performance in the literature, and (2) the inter-item dependencies captured in the sessions. The set of session graph aggregators is denoted by O_n and is presented in Table 1.

Specifically, we categorize session graphs and their corresponding aggregators in three dimensions (i.e., relational semantics, local-context information, and long-range dependencies). Session graph aggregators can be heterogeneous or homogeneous depending on whether the session graph distinguishes semantic discrepancy at the edges. For example, Relational GAT [33], Relational GGNN, and Mixup operation explicitly disentangle different types of item-transition semantics. Besides, some operations focus on capturing local-context information (i.e., sequential information in a session), such as EOPA [4] and our proposed Relational GGNN. Some operations capture long-range dependencies explicitly or implicitly. For example, SGAT [4] captures long-range dependencies by explicitly establish edges between distant items, while Relational GAT implicitly captures higher-order item interactions by stacking multi-layer networks. Finally, our proposed Mixup aggregator considers all types of information together.

4.2.2 Layer Aggregators. Layer aggregators aim to summarize the item representations from each layer of GNN. Similar to SANE [41], we choose 5 layer aggregators including *Max*, *Concat*, *Sum*, *Mean*, and *Last*. In addition, we search for the existence of the skip-connections between each intermediate layer and the last layer. We choose *Identity* to pass the output of the intermediate layer to the layer aggregators, and we choose *ZeroIdentity* otherwise. The set of layer aggregators and the set of skip operations are denoted by O_l and O_s, respectively.

Moreover, motivated by [23], we employ highway network as the shortcut connections between the initial node representation $\boldsymbol{x}_i^0$ and each intermediate layer output $\boldsymbol{x}_i^l (1 \le l \le L)$ to improve the learning capability of a L layer GNN. To use the highway network, we choose *Highway*, and *ZeroHighway* otherwise. We denote the set of highway network operations by O_h.

4.2.3 The Scale of the Search Space. The scale of the search space is determined by the number of GNN layers L. Precisely, according to the proposed session graph aggregators and layer aggregators, the size of search space can be formulated as $O(5^L \times 2^L \times 2^L \times 5)$. In our experiments, we set $L = 2$. Then, the possible number of architectures in the search space is 2000.

4.2.4 Continuous Relaxation of the Search Space. To utilize the differentiable search algorithm to find the optimal architecture, we perform the continuous relaxation of the search space. As shown in Figure 3, suppose we employ a L layer GNN, then the search space becomes a direct acyclic graph consisting of $L+3$ vertices by following SANE framework [41]. Each vertex denotes the latent representation of the items within a session, i.e., $\boldsymbol{H}^l \in \mathbb{R}^{b \times m \times d}$, where b denotes the mini-batch size, m denotes the length of the padded session, and d denotes the dimension size of latent representations. In addition to the L vertex $\{\boldsymbol{H}^1, ..., \boldsymbol{H}^L\}$ which associate with the outputs of L intermediate layers of GNN, we also have the input vertex $\boldsymbol{H}^0$, output vertex $\boldsymbol{H}^{L+2}$, and vertex $\boldsymbol{H}^{L+1}$ which collects the representations of the intermediate layers. Each vertex pair $(l, l+1)$ is associated with a candidate operation set $O = \{o_1, o_2, ...\}$ weighted by architecture parameters $\boldsymbol{\alpha}^{(l,l+1)} = \{\alpha_o^{(l,l+1)} | o \in O\}$

that transforms $\boldsymbol{H}^l$. Then, the categorical choice of a particular operation can be relaxed as:

$$\bar{o}^{(l,l+1)}(\boldsymbol{H}^l) = \sum_{o \in O} \frac{\exp\left(\alpha_o^{(l,l+1)}\right)}{\sum_{o' \in O} \exp\left(\alpha_{o'}^{(l,l+1)}\right)} o(\boldsymbol{H}^l) \tag{7}$$

As shown in Figure 3, for each vertex pair, the O can be any one of four operation sets O_n, O_l, O_s, O_h associated with architecture parameters $\boldsymbol{\alpha}_n, \boldsymbol{\alpha}_l, \boldsymbol{\alpha}_s, \boldsymbol{\alpha}_h$.

Specifically, given the mixed operation $\bar{o}_n$ collected from the operation set O_n, the continuous relaxed session graph aggregators aim to update the node representation:

$$\boldsymbol{x}_i^{l+1} = \bar{o}_n\left(\boldsymbol{x}_i^l, \boldsymbol{x}_j^l : (j,i) \in \mathcal{E}_{\text{in}}(i)\right) \tag{8}$$

Similarly, given the continuous relaxed operations $\bar{o}_l, \bar{o}_s, \bar{o}_h$ collected from O_l, O_s, O_h, the layer aggregator summarizes the item representations of intermediate layers and gives the final representations:

$$\begin{aligned} \boldsymbol{H}^{L+1} &= [\bar{o}_s(\bar{o}_h(\boldsymbol{H}^0, \boldsymbol{H}^1)); ...; \bar{o}_s(\bar{o}_h(\boldsymbol{H}^0, \boldsymbol{H}^L))] \\ \boldsymbol{H}^{L+2} &= \bar{o}_l(\boldsymbol{H}^{L+1}) \end{aligned} \tag{9}$$

where $[;]$ denotes the concatenation operation, which stacks the item representations of all the intermediate layers.

To this end, given the continuous relaxed dense architecture (i.e., supernet), we aim to find a dense graph neural network for session-based recommendation by solving the following bi-level optimization:

$$\begin{aligned} \boldsymbol{\alpha}^* = & \quad \arg\min_{\boldsymbol{\alpha} \in \{\boldsymbol{\alpha}_n, \boldsymbol{\alpha}_l, \boldsymbol{\alpha}_s, \boldsymbol{\alpha}_h\}} \mathcal{L}_{\text{val}}(\boldsymbol{\alpha}, \boldsymbol{\omega}^*(\boldsymbol{\alpha})) \\ \text{s.t.} \quad & \boldsymbol{\omega}^*(\boldsymbol{\alpha}) = \arg\min_{\boldsymbol{\omega}} \mathcal{L}_{\text{train}}(\boldsymbol{\alpha}, \boldsymbol{\omega}), \end{aligned} \tag{10}$$

where $\boldsymbol{\omega}$, $\boldsymbol{\alpha}$ denote the network weight parameters and architecture parameters, respectively.

4.3 Item Meta Knowledge Learner

Following the existing differentiable NAS methods [6, 20], our proposed AutoGSR employs the two-stage strategy. As mentioned before, in the architecture search stage, a dense supernet is trained by weighting the sum of candidate operations associated with different types of session information. In the second stage, a sparse architecture will be obtained by discritizing the weighted operations (i.e., keep the operation with the largest weight). As a result, there exists the correlation gap between dense supernet and sparse architecture, where the obtained sparse architecture may be biased (e.g., loose or overemphasize some types of session information).

To address this problem, we propose to utilize the item meta knowledge as the priori knowledge to guide the learning of the item representations by retraining the obtained sparse architecture. Specifically, we propose an item meta knowledge learner to learn the item meta knowledge (i.e., initial item representations). To enable the item meta knowledge to take into account all the information in the session, we design the item meta learner based on the self-attention techniques. More specifically, given a session, the item meta knowledge learner explicitly models inter-item dependencies by encoding the semantic relations from both adjacent and non-adjacent items in a session.

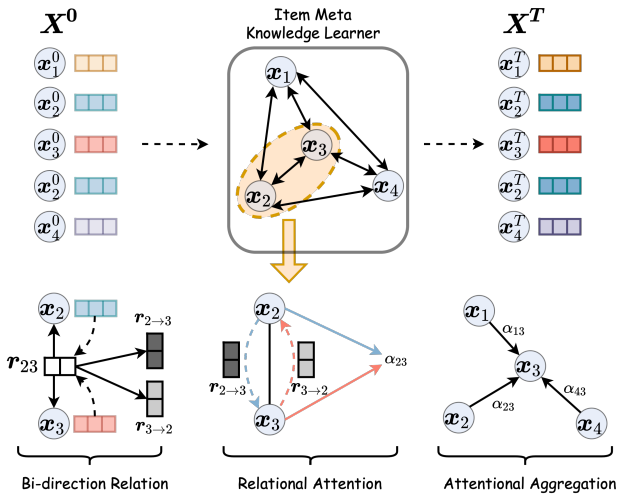


Figure 4: A detailed illustration of item meta knowledge learner.

Furthermore, to obtain complete item meta knowledge, the item meta knowledge learner should also be aware of the item-transition semantic discrepancy. Thus, to take all the session information into consideration, we propose an item meta knowledge learner, namely *Relation-Aware Self-Attention*. As shown in Figure 4, we learn the item representations in a session by item meta knowledge learner for T epochs. The inputs are initial item representations $\mathbf{X}^0$ and session data, while the output is the item meta knowledge $\mathbf{X}^T$. The details of the meta knowledge learner are as follows (we omit the layer notation l for brevity if the context is clear):

4.3.1 Relation-Aware Self-Attention. To apply self-attention on all item-pairs within a session, the challenge is how to distinguish the various item-transition semantics and maintain the sequential information. To overcome these challenges, we first propose to give each edge an explicit relation representation which aims to learn to disentangle the edge semantics automatically. Specifically, given representations $\boldsymbol{x}_i, \boldsymbol{x}_j$ of any two items in a session, the relation embedding is computed as follows:

$$\boldsymbol{r}_{ij} = \text{LeakyReLU}(\boldsymbol{W}_r[\boldsymbol{x}_i; \boldsymbol{x}_j]) \tag{11}$$

To encode the session sequential information (i.e., the direction of item-transition), following [1], we split the relation representation $\boldsymbol{r}_{ij}$ into forward relation representation $\boldsymbol{r}_{i \to j}$ and backward relation representation $\boldsymbol{r}_{j \to i}$:

$$[\boldsymbol{r}_{i \to j}; \boldsymbol{r}_{j \to i}] = \boldsymbol{W}_d \boldsymbol{r}_{ij} \tag{12}$$

Then, the attention score a_{ij} is computed based on both the item representations and the bi-directional relation representations:

$$\begin{aligned} e_{ij} &= (\boldsymbol{x}_i + \boldsymbol{r}_{i \to j}) \boldsymbol{W}_q^T \boldsymbol{W}_k (\boldsymbol{x}_j + \boldsymbol{r}_{j \to i}) \\ a_{ij} &= \text{softmax}_j\left(\frac{\exp(e_{ij})}{\sum_{j'=1}^{m} \exp(e_{ij'})}\right) \end{aligned} \tag{13}$$

In the end, the item meta knowledge can be computed as:

$$x_i^{l+1} = \sum_{j=1}^{m} a_{ij} W_v^l x_j^l \tag{14}$$

To obtain the final meta knowledge learner, following the transformer [30] architecture, we employ multi-head attention technique and stack multiple attention layers followed by a MLP layer with residual connections.

4.4 Searching Algorithm

To solve the bi-level optimization in Equation 10, following [41], we employ the gradient-based approximation to update the architecture parameters:

$$\nabla_{\boldsymbol{\alpha}} \mathcal{L}_{\text{val}} \left(\boldsymbol{\omega}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}\right) \approx \nabla_{\boldsymbol{\alpha}} \mathcal{L}_{\text{val}} \left(\boldsymbol{\omega} - \xi \nabla_{\boldsymbol{\omega}} \mathcal{L}_{\text{train}}(\boldsymbol{\omega}, \boldsymbol{\alpha}), \boldsymbol{\alpha}\right) \tag{15}$$

where $\boldsymbol{\omega}$ is the network weight parameters at current gradient step, ξ is the learning rate for inner optimization. After training, the operation with the highest architecture weight is kept as the operation in the final sparse architecture. Then the final architecture will be retrained on the training data and tuned on the validation data. We summarize the optimization process of AutoGSR in Algorithm 1.

Algorithm 1 Optimization algorithm of AutoGSR

Require: The search space $\mathcal{A}$, the epochs T_{search} for architecture search, the epochs T for training item meta learner
Ensure: Well-trained graph neural networks
1: **for** $t \leftarrow 1, 2, ..., T_{\text{search}}$ **do**
2: Compute the validation loss $\mathcal{L}_{val}$
3: Update architecture parameters $\boldsymbol{\alpha}$ based on Equation 15
4: Compute the training loss $\mathcal{L}_{\text{train}}$
5: Update weight parameters $\boldsymbol{\omega}$ based on $\nabla_{\boldsymbol{\omega}} \mathcal{L}_{\text{train}}(\boldsymbol{\omega}, \boldsymbol{\alpha})$
6: **end for**
7: Derive the final architecture $\{\boldsymbol{\alpha}_n^*, \boldsymbol{\alpha}_l^*, \boldsymbol{\alpha}_s^*, \boldsymbol{\alpha}_h^*\}$
8: Train the item meta knowledge learner for T epochs
9: Re-train the final architecture with the item meta knowledge $\mathbf{X}^T$
10: **return** Searched architecture with the trained parameters.

5 EXPERIMENTS

In this section, we conduct extensive experiments to demonstrate the effectiveness of our proposed AutoGSR.

5.1 Experimental Settings

5.1.1 Datasets. We utilize the following three real-world datasets to verify the effectiveness of our AutoGSR:

- **Diginetica**[1] records the typical e-commerce transaction data for five month. The dataset is from CIKM Cup 2016.
- **Tmall**[2] records the anonymized users' shopping logs on Tmall online shopping platform. The dataset is from IJCAI-15 competition.
- **Yoochoose1_64**[3] records users' clicks from an e-commerce website. Since the training set of Yoochoose is large, we use the most recent proportion 1/64 of the dataset as the training data by following [21].

In dataset splitting, we follow the common practice [33, 37]. For Diginetica dataset, the sessions of last week are set as the test data, and the remaining historical data are set for training. For Tmall, the sessions of last 100 seconds are set as the test data, and the remaining historical data are set for training. For Yoochoose1_64, the sessions of last day are set as the test data, and the remaining historical data are set for training. In dataset preprocessing, we also follow the common practice [33, 37]. Specifically, for all datasets, we filter out the sessions with length 1 and remove items appearing less than 5 times. For Tmall dataset, we also filter out sessions with length longer than 40. Besides, following [33], we augment a session $s = [s_1, s_2, ..., s_{l_s}]$ to generate labeled sequences $([s_1], s_2), ([s_1, s_2], s_3), ..., ([s_1, s_2, ..., s_{l_s-1}], s_{l_s})$ for both training and testing data. The statistics of three pre-processed datasets are summarized in Table 2.

Table 2: Basic statistics of three datasets. "Avg. lengths" denotes the average length of all sessions.

Dataset	Diginetica	Tmall	Yoochoose1_ 64
# Train sessions	719,470	351,268	369,859
# Test sessions	60,858	25,898	55,898
# Items	43,097	40,728	16,766
Avg. lengths	5.12	6.69	6.16

5.1.2 Evaluation Metrics. Following [33], to evaluate the recommendation results, we adopt two widely-used ranking metrics: Precision (P@K) and Mean Reciprocal Rank (MRR@K). We set K as 20 for both two metrics.

5.1.3 Baselines. We first compare AutoGSR with the following temporal-based and GNNs-based models for session recommendation:

- **POP** recommends the top-N items based on the popularity information in the training data.
- **Item-KNN [28]** recommends items based on the cosine similarity between items in the current session and others.
- **FPMC [27]** is a hybrid method combining both Markov chains and matrix factorization to capture users' interests.
- **GRU4Rec [13]** is an RNN-based method which makes prediction by stacking multiple GRU layers.
- **NARM [17]** employs RNNs to make prediction and utilizes attention mechanism to capture the users' interests.
- **STAMP [21]** applies an attention mechanism to model the users' general interests and the current interests simultaneously.
- **CSRM [32]** incorporates the neighbor sessions to model the current session by utilizing the memory networks.
- **CoSAN [22]** employs multi-head attention to capture the dependencies between items.

[1] http://cikm2016.cs.iupui.edu/cikm-cup
[2] https://tianchi.aliyun.com/dataset/dataDetail?dataId=42
[3] http://2015.recsyschallenge.com/challege. html

Table 3: Performance comparison between the baselines and AutoGSR on three datasets. We use bold font to label the best performance.

Method	Diginetica		Tmall		Yoochoose1_ 64	
	P@20	MRR@20	P@20	MRR@20	P@20	MRR@20
POP	1.18	0.28	2.00	0.90	6.71	0.58
Item-KNN	35.75	11.57	9.15	3.31	51.60	21.81
FPMC	22.14	6.66	16.06	7.32	45.62	15.01
GRU4Rec	30.79	8.22	10.93	5.89	60.64	22.89
NARM	48.32	16.00	23.30	10.70	68.32	28.63
STAMP	46.62	15.13	26.47	13.36	68.74	29.67
CSRM	48.49	17.13	29.46	13.96	-	-
CoSAN	51.97	17.92	32.68	14.09	-	-
SR-GNN	50.73	17.59	27.57	13.72	70.57	30.94
GC-SAN	50.90	17.63	-	-	70.66	30.04
LESSR	51.71	18.15	23.53	9.56	70.05	30.59
GCE-GNN	54.22	19.04	33.42	15.42	70.91	30.63
MetaNet	53.59	18.40	32.13	15.82	71.39	30.48
AutoGSR	**54.56 (0.12)**	**19.20 (0.04)**	**33.71 (0.47)**	**15.87 (0.23)**	**71.77 (0.27)**	**31.02 (0.42)**

- **SR-GNN [37]** converts the session into a graph and utilizes GGNN to learn item embeddings. An attention network is used to get the final session representation.
- **GC-SAN [38]** utilizes GGNN to extract local-context information and then utilizes the multi-head attention to capture the global dependencies between items.
- **LESSR [4]** converts a session to a losslessly encoded graph and a shortcut graph, and applies two different GNN layers to learn the item embeddings.
- **GCE-GNN [33]** applies GNNs in both the session graph and global graph to learn the item embeddings. Reversed position embeddings are incorporated to get the final session representations and make predictions.

In addition, we compare AutoGSR with two basic NAS methods:

- **RANDOM** samples network architectures from the search space randomly, and trains the sampled network from scratch.
- **SANE [41]** is a recent proposed differentiable NAS framework for GNNs. The basic SANE framework without item meta knowledge is employed as the baseline.

5.1.4 Framework Settings and Implementation Details. We run all the experiments with Pytorch (version 1.8) on a Nvidia A4000 GPU (Memory: 16GB, Cuda version: 11.1). We implement AutoGSR on top of the building code provided by SANE[4], DGL (version 9.6)[5] and PyG (version 2.0.2)[6]. We use first-order approximation as introduced in [20] since it is efficient and effective in our experiments (i.e., $\xi = 0$ in Equation 15). We run the search process for 20 times with different random seeds on the train set, and retrieve top-5 architectures. For each obtained architecture. we retrain it on the train set, fine-tune the hyperparameters on the validation set and report the recommendation performance on the test set. The final mean accuracy with standard deviations are reported.

Following the settings in [33, 36], for a fair comparison, we keep the hyper-parameters of each model consistent. Specifically, the size of latent embedding is set to 100. The size of mini-batch is set to 100. We use the Adam optimizer with the initial learning rate 0.001, which will decay by 0.1 after every 3 epoch. The L_2 penalty is set to 10^{-5}. All the hyperparameters are searched on the validation set which is a random 10% subset of the train set.

5.2 Overall Comparison

To demonstrate the effectiveness of the proposed AutoGSR, we compare it with the state-of-the-art session-based recommendation models. The results are shown in Table 3. We retrain and test the obtained top-5 architectures and calculate the mean and standard deviation on two metrics. From the results we observe that the neural network-based methods outperform the traditional methods significantly, which demonstrates their powerful capability of capturing the local-context sequential information within a session. Besides, the methods utilizing self-attention mechanism such as STAMP and CoSAN perform better than traditional neural network-based methods, which demonstrates the importance of long-range dependencies for session recommendation. Generally, GNNs-based methods outperform other methods, indicating that GNNs are very effective for the session recommendation and justifying our use of the GNNs as the search space. GCE-GNN achieves the competitive performance among other traditional baselines which explicitly models the inter-item relation semantics in a session. This indicates that disentangling the relation semantics of a session is a promising direction for GNNs-based session recommendation.

To capture the information from the session data in a comprehensive way, MetaNet stacks our proposed item meta knowledge learner to learn item representations, and achieves very competitive

[4]https://github.com/AutoML-4Paradigm/SANE
[5]https://github.com/dmlc/dgl
[6]https://github.com/rusty1s/pytorch geometric

performance. Compared to the expert-designed baseline methods, based on our designed search space, AutoGSR is able to automatically learn neural architectures that capture various session information and achieve the best results on all three datasets.

Table 4: Comparison between our framework and its variants. Bold font denotes the best performance.

Models	Diginetica		Tmall		Yoochoose1_64	
	P@20	MRR@20	P@20	MRR@20	P@20	MRR@20
RANDOM	50.23	18.02	24.72	9.86	59.16	25.93
SANE	54.51	19.17	33.65	15.76	71.75	31.06
w/o highway	54.30	19.13	31.37	14.45	71.53	30.92
w/o layer aggregator	54.42	19.06	**33.99**	15.75	71.54	**31.07**
AutoGSR	**54.56**	**19.20**	33.71	**15.87**	**71.77**	31.02

5.3 Ablation Study

In this subsection, we conduct the ablation study to study the effectiveness of different components of AutoGSR. We compare our model with its two variants: (1) **RANDOM**: this method randomly samples architectures from the search space. (2) **SANE**: this method does not employ the item meta knowledge to retrain the obtained sparse architecture. In addition, we also conduct the ablation to study the effectiveness of operations: (3) **w/o highway**: this method excludes the highway network operation set O_h from the search space. (4) **w/o layer aggregator**: this method excludes the layer aggregator operation set O_l from the search space. The comparison results are shown in Table 4. We report the mean values for two metrics.

From the results we observe that, AutoGSR performs significantly better than method that randomly samples architectures. In addition, the learned item meta knowledge can be used as effective initial item representations to improve the recommendation performance. Although the sparse architecture obtained in the second stage may suffer from bias as analyzed before, training this architecture without any prior knowledge at the parameter level can still effectively capture the session information. Furthermore, we observe that introducing the prior knowledge can complement the learned session information and improve the final recommendation performance. The results demonstrate the effectiveness of these components. For operations, the results show that both highway network and layer aggregator are able to improve the recommendation performance. In particular, highway network makes a greater contribution as it effectively mitigates the overfitting problem.

5.4 Parameter Analysis

This subsection studies the impact of the model hyperparameters. Specifically, for architecture hyperparameters, we study the influence of the number of GNN layers and the architecture sample ratio ϵ. Here, the architecture sample ratio ϵ is the probability that we randomly sample an operation from a certain type of operation set. When $\epsilon = 1$, it is equivalent to random search, and when $\epsilon = 0$, it is equivalent to our AutoGSR.

Besides, we also study the hyperparameters of item meta knowledge learner. In particular, we study the number of model heads and the size of relational embedding. For each hyperparameter, we present the recommendation performance under different choices of it by fixing other hyperparameters.

Table 5: The results of MRR@20 w.r.t. different architecture sample ratio ϵ.

ϵ	0.0	0.2	0.4	0.6	0.8	1.0
Diginetica	**19.20**	19.01	18.72	18.90	18.87	18.02
Tmall	**15.87**	15.02	14.9	13.67	13.74	9.86
Yoochoose1_64	**31.02**	30.50	30.29	30.22	30.03	25.93

Table 6: The results of MRR@20 w.r.t. different GNN layers.

L	1	2	3
Diginetica	**19.31**	19.20	19.12
Tmall	15.68	**15.87**	15.83
Yoochoose1_64	30.73	**31.02**	30.83

Table 7: The results of MRR@20 w.r.t. different head number.

H	1	2	4
Diginetica	19.18	**19.20**	19.17
Tmall	15.79	**15.87**	15.76
Yoochoose1_64	30.93	**31.02**	30.84

Table 8: The results of MRR@20 w.r.t. different relational embedding size.

E	32	64	128	256
Diginetica	18.97	19.09	19.20	**19.21**
Tmall	15.68	15.74	**15.87**	15.87
Yoochoose1_64	30.56	30.79	**31.02**	30.92

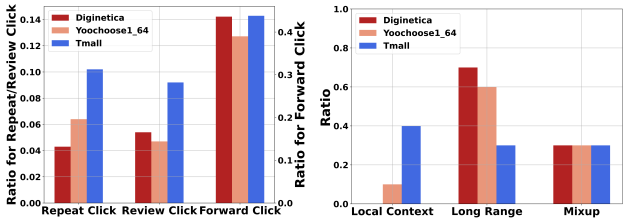
5.4.1 Effect of Architecture Sample Ratio ϵ. Here, we investigate the recommendation performance of the searched network architecture by varying the sample ratio ϵ.

The results in Table 5 show that when the ratio ϵ is increased, the recommendation performance degrades. The performance gets worst when $\epsilon = 1$. The results indicate the effectiveness of the searching algorithm compared to the random-based method.

5.4.2 Effect of GNNs Layers. We vary the GNN layers to investigate the robustness of our AutoGSR. The results in Table 6 show that increasing the number of layers improves the performance. Nevertheless, when the number of layer increases too much, the model still faces the overfitting problem.

5.4.3 Effect of Item Meta Knowledge Learner. We first vary the number of heads of the item meta knowledge learner to investigate the performance. The results in Table 7 illustrate that too many heads will degrade the final recommendation performance. A similar phenomenon is observed for the size of relational embedding as shown in Table 8, where a small embedding size affects the learned initial item representation, which further reduces the final recommendation performance. This is because these parameters affect the learning capability of the item meta knowledge learner and influence the learned item meta knowledge. Therefore, these hyperparameters should be carefully tuned for real-world session data.

5.5 Empirical Studies of Learned Architectures



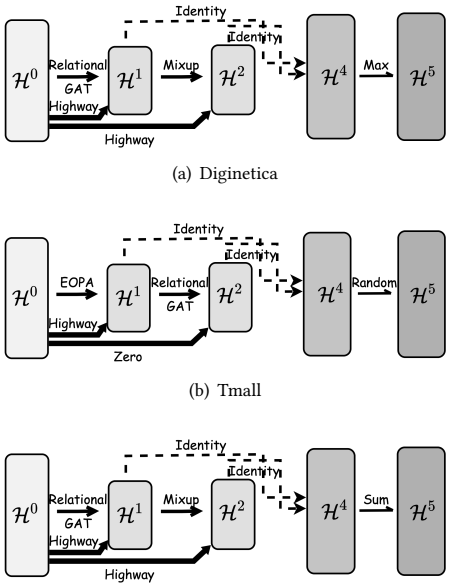
(a) Edge Type Distribution (b) Operation Distribution

Figure 5: (a) Distribution of the edge types on three datasets. (b) Distribution of the operations of the learned network architectures on three datasets.

In this subsection, we investigate the correlation between the searched architectures and the corresponding dataset properties. We first study the properties of three datasets (i.e., relational semantic distribution). Specifically, we calculate the proportion of each type of item-transition in a session and calculate its average over all sessions. As shown in Figure 5(a), *Repeat Click* and *Review Click* appear in higher proportions in the Tmall dataset and in a much lower proportions in the other two datasets. Then, we study the distribution of different operations in the learned architectures. We calculate the proportion of occurrences of GNN operations in top-5 architectures. As shown in Figure 5(b), for Tmall dataset, the learned architectures contain more operations which focus on local-context information (i.e., sequential information), while for the other two datasets, the searched architectures prefer operations which focus on long-range dependencies. Moreover, for all datasets, the *Mixup* operation is favored. The above results demonstrate that the effectiveness of the proposed *Mixup* operator and AutoGSR can find the architectures that adapt various properties of the dataset.

5.6 Visualization

We visualize one of top-5 searched architectures by AutoGSR on three datasets in Figure 6. From the results we observe that, the search algorithm yields neural network architectures that are very different from the existing literature. The generated architectures are strongly related to the session data. For instance, different layer aggregators are employed for different session data. In addition, the search algorithm selects the different session graph aggregators which capture different types of session information. We also observe that Relatoinal GAT is a popular choice, which illustrates the importance of the rich semantics corresponding to item-transitions.



(a) Diginetica

(b) Tmall

(c) Yoochoose1_64

Figure 6: Visualization of the one of top-5 architectures searched by AutoGSR on three datasets.

6 CONCLUSION

In this paper, we propose a NAS framework for session-based recommendation, namely AutoGSR, to automatically search graph neural network architectures. In addtion to the existing operations, we propose two novel GNN operations (i.e., Relational GGNN and Mixup) to construct a complete and expressive search space. We use the differentiable search algorithm to find the optimal graph neural network architecture. Besides, we propose the item meta knowledge learner to learn item meta knowledge which contain comprehensive session information. Substantial experiments show that AutoGSR outperforms previous expert-designed networks for session recommendation. The results demonstrated that our AutoGSR can find effective network architectures and achieve state-of-the-art recommendation performance.

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China (#62102177 and #U1811461), the Natural Science Foundation of Jiangsu Province (#BK20210181), the Key R&D Program of Jiangsu Province (#BE2021729), Open Research Projects of Zhejiang Lab (#2022PG0AB07), and the Collaborative Innovation Center of Novel Software Technology and Industrialization, Jiangsu, China.

REFERENCES

[1] Deng Cai and Wai Lam. 2020. Graph transformer for graph-to-sequence learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 7464–7471.

[2] Lei Chen, Fajie Yuan, Jiaxi Yang, Min Yang, and Chengming Li. 2021. Scene-adaptive Knowledge Distillation for Sequential Recommendation via Differentiable Architecture Search. *arXiv preprint arXiv:2107.07173* (2021).

[3] Tianwen Chen and Raymond Chi-Wing Wong. 2019. Session-Based Recommendation with Local Invariance. In *2019 IEEE International Conference on Data Mining (ICDM)*. 994–999. https://doi.org/10.1109/ICDM.2019.00113

[4] 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 International Conference on Knowledge Discovery & Data Mining*. 1172–1180.

[5] Wanyu Chen, Fei Cai, Honghui Chen, and Maarten de Rijke. 2019. A dynamic co-attention network for session-based recommendation. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1461–1470.

[6] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. 2019. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1294–1303.

[7] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2018. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081* (2018).

[8] Wenqi Fan, Xiaorui Liu, Wei Jin, Xiangyu Zhao, Jiliang Tang, and Qing Li. 2022. Graph Trend Networks for Recommendationsn. In *Proceedings of the 45rd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

[9] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *Proceedings of the 30th The Web Conference, WWW'19*. ACM, CA, 417–426.

[10] Chen Gao, Yu Zheng, Nian Li, Yinfeng Li, Yingrong Qin, Jinghua Piao, Yuhan Quan, Jianxin Chang, Depeng Jin, Xiangnan He, et al. 2021. Graph neural networks for recommender systems: Challenges, methods, and directions. *CoRR* (2021).

[11] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. 2020. Graph Neural Architecture Search.. In *IJCAI*, Vol. 20. 1403–1409.

[12] Priyanka Gupta, Diksha Garg, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff. 2019. NISER: Normalized item and session representations to handle popularity bias. *arXiv preprint arXiv:1909.04276* (2019).

[13] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[14] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric Xing. 2018. Neural architecture search with bayesian optimisation and optimal transport. *arXiv preprint arXiv:1802.07191* (2018).

[15] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 197–206.

[16] Farhan Khawar, Xu Hang, Ruiming Tang, Bin Liu, Zhenguo Li, and Xiuqiang He. 2020. AutoFeature: Searching for Feature Interactions and Their Architectures for Click-through Rate Prediction. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 625–634.

[17] 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*. 1419–1428.

[18] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* (2015).

[19] Bin Liu, Niannan Xue, Huifeng Guo, Ruiming Tang, Stefanos Zafeiriou, Xiuqiang He, and Zhenguo Li. 2020. AutoGroup: Automatic feature grouping for modelling explicit high-order feature interactions in CTR prediction. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 199–208.

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[21] 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*. 1831–1839.

[22] Anjing Luo, Pengpeng Zhao, Yanchi Liu, Fuzhen Zhuang, Deqing Wang, Jiajie Xu, Junhua Fang, and Victor S Sheng. 2020. Collaborative Self-Attention Network for Session-based Recommendation.. In *IJCAI*. 2591–2597.

[23] Zhiqiang Pan, Fei Cai, Wanyu Chen, Honghui Chen, and Maarten de Rijke. 2020. Star graph neural networks for session-based recommendation. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1195–1204.

[24] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*. PMLR, 4095–4104.

[25] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, Vol. 33. 4780–4789.

[26] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 2020. A comprehensive survey of neural architecture search: Challenges and solutions. *arXiv preprint arXiv:2006.02903* (2020).

[27] 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*. 811–820.

[28] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-Based Collaborative Filtering Recommendation Algorithms. In *Proceedings of the 10th International Conference on World Wide Web* (Hong Kong, Hong Kong) *(WWW '01)*. Association for Computing Machinery, New York, NY, USA, 285–295.

[29] Guy Shani, David Heckerman, Ronen I Brafman, and Craig Boutilier. 2005. An MDP-based recommender system. *Journal of Machine Learning Research* 6, 9 (2005).

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

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

[32] Meirui Wang, Pengjie Ren, Lei Mei, Zhumin Chen, Jun Ma, and Maarten de Rijke. 2019. A collaborative session-based recommendation approach with parallel memory modules. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 345–354.

[33] Ziyang Wang, Wei Wei, Gao Cong, Xiao-Li Li, Xian-Ling Mao, and Minghui Qiu. 2020. Global context enhanced graph neural networks for session-based recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 169–178.

[34] Lanning Wei, Huan Zhao, Quanming Yao, and Zhiqiang He. 2021. Pooling architecture search for graph classification. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2091–2100.

[35] Zhikun Wei, Xin Wang, and Wenwu Zhu. 2021. AutoIAS: Automatic Integrated Architecture Searcher for Click-Trough Rate Prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2101–2110.

[36] Qitian Wu, Hengrui Zhang, Xiaofeng Gao, Peng He, Paul Weng, Han Gao, and Guihai Chen. 2019. Dual graph attention networks for deep latent representation of multifaceted social effects in recommender systems. In *Proceedings of the 30th The Web Conference, WWW'19*. ACM, CA, 2091–2102.

[37] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. 2019. Session-based recommendation with graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 346–353.

[38] 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*, Vol. 19. 3940–3946.

[39] Haochao Ying, Fuzhen Zhuang, Fuzheng Zhang, Yanchi Liu, Guandong Xu, Xing Xie, Hui Xiong, and Jian Wu. 2018. Sequential recommender system based on hierarchical attention network. In *IJCAI International Joint Conference on Artificial Intelligence*.

[40] Ziwei Zhang, Xin Wang, and Wenwu Zhu. 2021. Automated Machine Learning on Graphs: A Survey. *arXiv preprint arXiv:2103.00742* (2021).

[41] Huan Zhao, Quanming Yao, and Weiwei Tu. 2021. Search to aggregate neighborhood for graph neural network. *arXiv preprint arXiv:2104.06608* (2021).

[42] Xiangyu Zhao, Haochen Liu, Wenqi Fan, Hui Liu, Jiliang Tang, and Chong Wang. 2021. Autoloss: Automated loss function search in recommendations. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3959–3967.

[43] Xiangyu Zhao, Haochen Liu, Hui Liu, Jiliang Tang, Weiwei Guo, Jun Shi, Sida Wang, Huiji Gao, and Bo Long. 2021. Autodim: Field-aware embedding dimension searchin recommender systems. In *Proceedings of the Web Conference 2021*. 3015–3022.

[44] Xiangyu Zhaok, Haochen Liu, Wenqi Fan, Hui Liu, Jiliang Tang, Chong Wang, Ming Chen, Xudong Zheng, Xiaobing Liu, and Xiwang Yang. 2021. Autoemb: Automated embedding dimensionality search in streaming recommendations. In *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE, 896–905.

[45] Kaixiong Zhou, Qingquan Song, Xiao Huang, and Xia Hu. 2019. Auto-gnn: Neural architecture search of graph neural networks. *arXiv preprint arXiv:1909.03184* (2019).

[46] Chenxu Zhu, Bo Chen, Weinan Zhang, Jincai Lai, Ruiming Tang, Xiuqiang He, Zhenguo Li, and Yong Yu. 2021. AIM: Automatic Interaction Machine for Click-Through Rate Prediction. *IEEE Transactions on Knowledge and Data Engineering* (2021).