Cold-start Sequential Recommendation via Meta Learner

Yujia Zheng,[1] **Siyi Liu,** [1] **Zekun Li,** [2, 3] **Shu Wu** [4, 5]

[1] University of Electronic Science and Technology of China
[2] School of Cyber Security, University of Chinese Academy of Sciences
[3] Institute of Information Engineering, Chinese Academy of Sciences
[4] School of Artificial Intelligence, University of Chinese Academy of Sciences
[5] Institute of Automation and Artificial Intelligence Research, Chinese Academy of Sciences
{yjzheng19, ssui.liu1022, lizekunlee}@gmail.com, shu.wu@nlpr.ia.ac.cn

AAAI2021

[AAAI21]Cold-start Sequential Recommendation via Meta Learner

- alleviate the item cold-start problem in sequential scenarios given only sparse interactions
- No side information (e.g., user profile, item attributes, and cross-domain knowledge)
- We first design a sequence pair encoder to effectively extract user preference. Then a matching network is applied to match the query sequence pair to the support set corresponding to the candidate cold-start item. Moreover, we employ the meta-learning based gradient descent approach for parameter optimization.

Contribution

- First, we propose Mecos framework for sequential scenarios to alleviate the item cold-start problem. To the best of our knowledge, this is the first work to tackle this problem. The task is non-trivial as it aims at a classical problem in a novel and challenging context, where the only available data is the user-item interaction.
- Second, our framework can be easily integrated with existing neural network-based sequential recommendation models. And once trained, Mecos can be adapted to new items without further fine-tuning.
- Lastly, we compare Mecos with state-of-the-art methods on three public datasets. Results demonstrate the effectiveness of our framework. And a comprehensive ablation study is conducted to analyze the contributions of the key components.

Preliminary

$$\zeta_i = (v_{i,1}, v_{i,2}, ..., v_{i,n})$$

query sequence pair $(\zeta_i, ?^{[1]})$.

In our task, we have a set of cold-start items $\mathcal{Y} \subset \mathcal{V}$, that every single item in $\mathcal{Y}$ only has a few interactions with users

$\mathcal{Y}_{\text{train}}$ and $\mathcal{Y}_{\text{test}}$ to represent the training and testing instances, where $\mathcal{Y}_{\text{test}} \cap \mathcal{Y}_{\text{train}} = \phi$.

Meta-training

a training task $T_i \in \mathcal{T}_{\text{meta-train}}$

we first sample N next-click items from $\mathcal{Y}_{\text{train}}$. For each of those N items, we sample K sequence pairs, where each pair $(\zeta_i, v_{i,n+1})$ represents a sequence ζ_i and its ground-truth next-click item $v_{i,n+1}$, as the support set. As the cold-

data augmentation

2019) (e.g., a sequence (v_0, v_1, v_2, v_3) is divided into three successive sequences: (v_0, v_1), (v_0, v_1, v_2), (v_0, v_1, v_2, v_3)),

Preliminary

Meta-testing After training, the model can make predictions for the task with N new ground-truth items from $\mathcal{Y}_{\text{test}}$, which is the *meta-testing* step. These meta-testing ground-truth items are unseen from the meta-training. The same as meta-training, each meta-testing task also has its few-shot training data and testing data. And these tasks form the meta-test set $\mathcal{T}_{\text{meta-test}}$. Moreover, we randomly leave out a subset of labels in $\mathcal{Y}_{\text{train}}$ to generate the validation set $\mathcal{T}_{\text{meta-valid}}$.

Now we can define our total meta-learning task set as $\mathcal{T} = \mathcal{T}_{\text{meta-train}} \bigcup \mathcal{T}_{\text{meta-test}} \bigcup \mathcal{T}_{\text{meta-valid}}$.

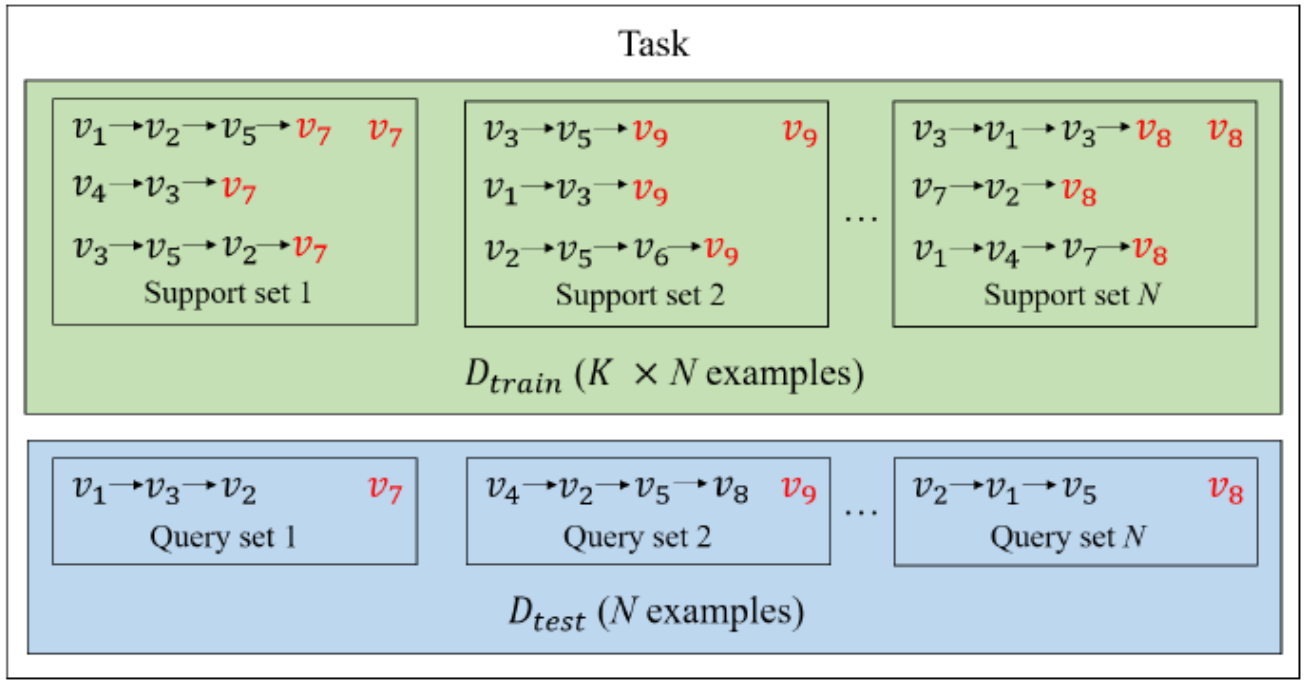

Figure 1: Single task with $K = 3$ and N different support sets. Each query set is the query sequence pair that needs to be matched with a candidate cold-start item (red) corresponding to the support set. When $K = 3$, we have three training instances for our target item. And N is the size of candidate cold-start items set.

Integrating with Existing Methods With the input of pre-trained item embeddings generated by existing sequential recommendation models, Mecos can be easily integrated with these methods to improve their performance in cold-start scenarios. Based on our setting, tasks that their ground-truth next-click items have rich interactions will be excluded from $\mathcal{T}$. And these data will be selected for the training of pre-trained item embeddings. To ensure those ground-truth items will not be seen before meta-learning tasks, sequences in pre-trained datasets will be removed when they contain the ground-truth item in $\mathcal{T}$.

It is noteworthy that we are not introducing more complexity or training with more data based on the baseline models. The purpose of pre-training is to make sure that both models with or without Mecos have a similar quality of embeddings for items with rich interactions. And all models are training with the same amount of data. Under these circumstances, we can make a fair comparison of recommendation performance toward cold-start items.

~~sampled from $\mathcal{Y}_{\text{test}}$.~~ All models are trained with the same data (pre-train data, all sequences in $\mathcal{T}_{\text{meta-train}}$ and support sets in $\mathcal{T}_{\text{meta-valid}}$/$\mathcal{T}_{\text{meta-test}}$) for fair comparisons. Moreover,

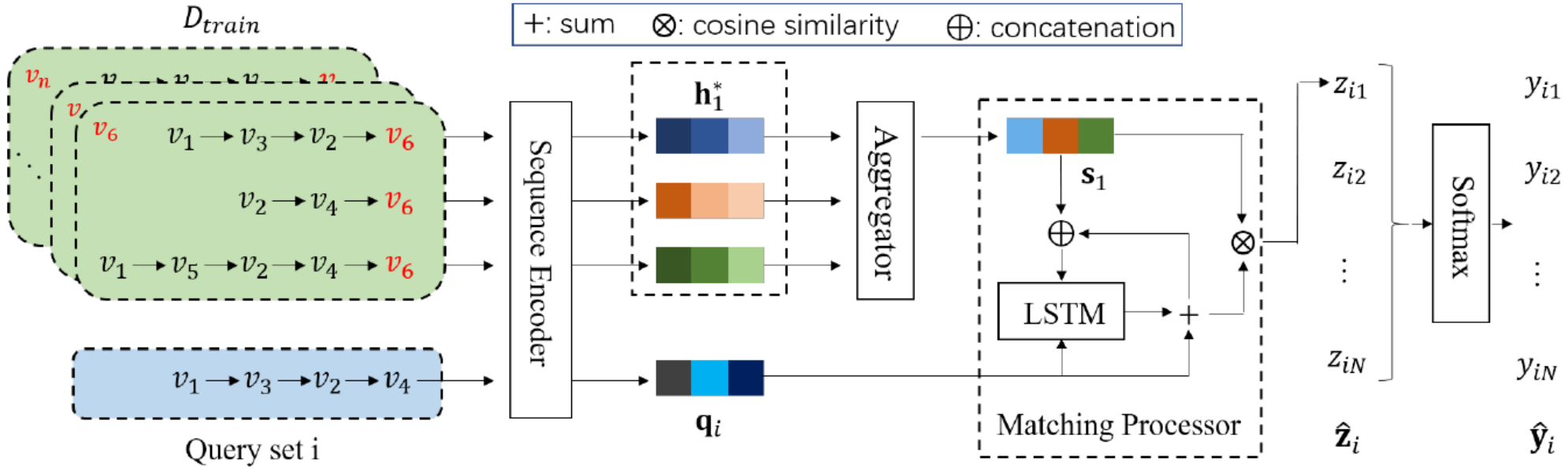

Figure 2: The framework of Mecos. The green square represents a support set that contains few-shot training sequences of a candidate cold-start item (red). Our goal is to recommend cold-start items to the query sequence (blue square). It first generates sequence pair representations (Eq. 1, 2), then aggregates few-shot sequence representations to generate support set representation (Eq. 3). Finally, it employs a matching network (Eq. 4) to compute similarity scores between the query set and all support sets (Eq. 5). After matching the i-th query set with N different support sets, we can get a vector of similarity scores $\hat{\mathbf{z}}_i$. Then a softmax function is applied to generate the i-th query set's final recommendation score of each candidate item, denoted as $\hat{\mathbf{y}}_i$.

$$
\begin{aligned}
\mathbf{e}_j &= \mathbf{p}^T \left(\mathbf{W}_a^1 \mathbf{v}_{i,n} + \mathbf{W}_a^2 \mathbf{v}_{i,j} + \mathbf{W}_a^3 \mathbf{v}_{i,avg} + \mathbf{b} \right), \\
\alpha_j &= \frac{exp(\mathbf{e}_j)}{\sum_{k=1}^{n} exp(\mathbf{e}_k)}, \\
\mathbf{R}(\zeta_i) &= \sum_{j=1}^{n} \alpha_j \mathbf{v}_{i,j},
\end{aligned}
\tag{1}
$$

$$
\begin{aligned}
\mathbf{h}_i^0 &= \left[\mathbf{R}(\zeta_i); \mathbf{v}_{i,n+1} \right], \\
\mathbf{h}_i^l &= \mathrm{ReLU} \left(\mathbf{W}^l \mathbf{h}_i^{l-1} + \mathbf{b}^l \right), \\
\mathbf{h}_i &= \mathbf{h}_i^0 + \mathbf{h}_i^l,
\end{aligned}
\tag{2}
$$

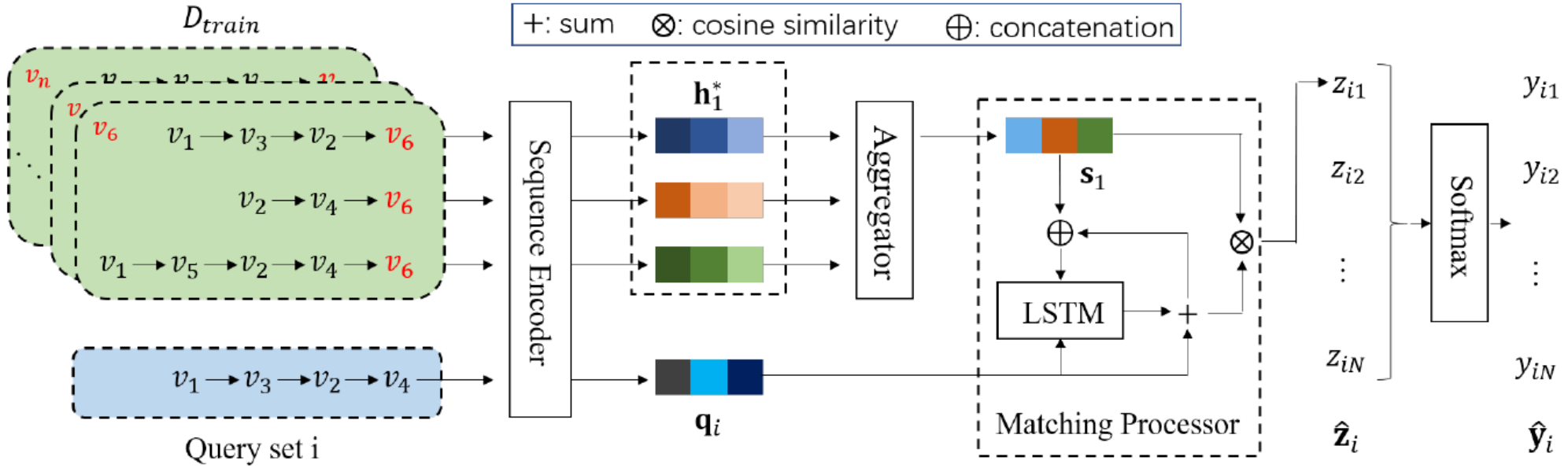

Figure 2: The framework of Mecos. The green square represents a support set that contains few-shot training sequences of a candidate cold-start item (red). Our goal is to recommend cold-start items to the query sequence (blue square). It first generates sequence pair representations (Eq. 1, 2), then aggregates few-shot sequence representations to generate support set representation (Eq. 3). Finally, it employs a matching network (Eq. 4) to compute similarity scores between the query set and all support sets (Eq. 5). After matching the i-th query set with N different support sets, we can get a vector of similarity scores $\hat{\mathbf{z}}_i$. Then a softmax function is applied to generate the i-th query set's final recommendation score of each candidate item, denoted as $\hat{\mathbf{y}}_i$.

After encoding K sequence pairs in the support set into $\mathbf{h}_i^* = \{\mathbf{h}_{i,1}, \mathbf{h}_{i,2}, ..., \mathbf{h}_{i,K}\}$, we aggregate them into the support set representation $\mathbf{s}_i$:

$$\mathbf{s}_i = Aggr(\mathbf{h}_i^*), \tag{3}$$

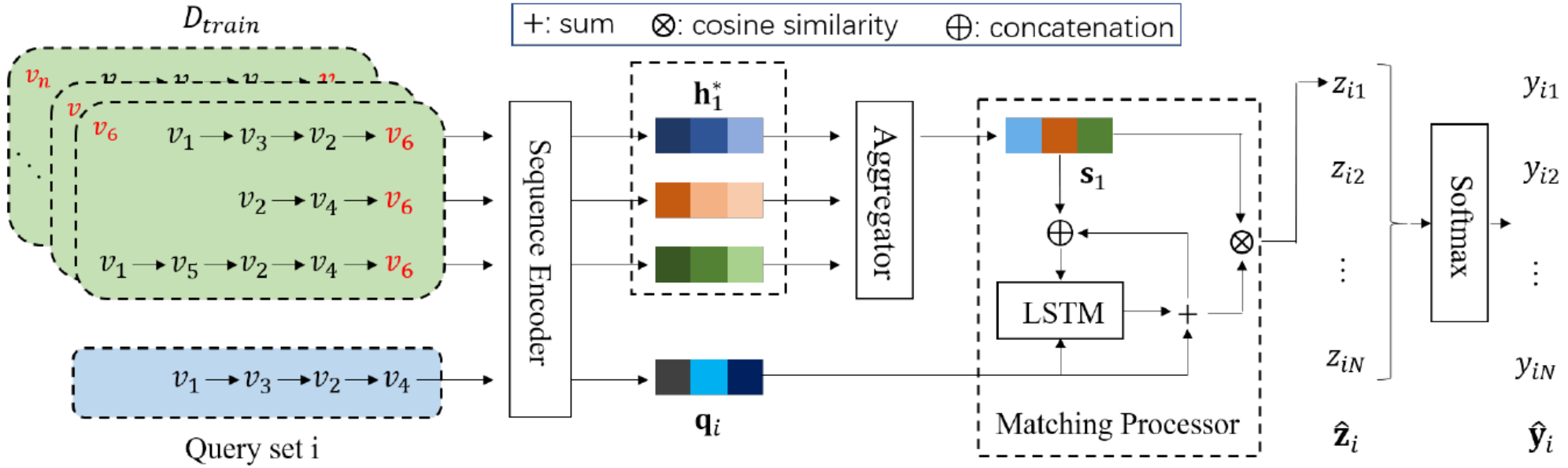

Figure 2: The framework of Mecos. The green square represents a support set that contains few-shot training sequences of a candidate cold-start item (red). Our goal is to recommend cold-start items to the query sequence (blue square). It first generates sequence pair representations (Eq. 1, 2), then aggregates few-shot sequence representations to generate support set representation (Eq. 3). Finally, it employs a matching network (Eq. 4) to compute similarity scores between the query set and all support sets (Eq. 5). After matching the i-th query set with N different support sets, we can get a vector of similarity scores $\hat{\mathbf{z}}_i$. Then a softmax function is applied to generate the i-th query set's final recommendation score of each candidate item, denoted as $\hat{\mathbf{y}}_i$.

$$\begin{aligned} \hat{\mathbf{q}}_i^t, \mathbf{c}^t &= \text{LSTM}\left(\mathbf{q}_i, \left[\mathbf{q}_i^{t-1}; \mathbf{s}_j\right], \mathbf{c}^{t-1}\right), \\ \mathbf{q}_i^t &= \hat{\mathbf{q}}_i^t + \mathbf{q}_i, \end{aligned} \tag{4}$$

$$z_{ij} = \frac{\mathbf{q}_i^t \mathbf{s}_j}{\|\mathbf{q}_i^t\| \times \|\mathbf{s}_j\|}, \tag{5}$$

$$\hat{\mathbf{y}}_i = \text{softmax}(\hat{\mathbf{z}}_i),$$

The training process is shown in Algorithm 1. For each task, we apply cross-entropy as the loss function:

$$\mathcal{L}_T = -\sum_{i=1}^{N}\sum_{j=1}^{N} y_{ij}\log\left(\hat{y}_{ij}\right) + \left(1-y_{ij}\right)\log\left(1-\hat{y}_{ij}\right), \tag{7}$$

where $\mathbf{y}_i$ denotes the one-hot vector of the ground truth item for i-th query pair. And y_{ij}, $\hat{y}_{ij}$ represent the j-th element in vector $\mathbf{y}_i$ and $\hat{\mathbf{y}}_i$, respectively. Finally, our model is trained by Back-Propagation Through Time (BPTT) algorithm.

Algorithm 1: Meta-training process

Input: Meta-training task set $\mathcal{T}_{\text{meta-train}}$, pre-trained item embedding $\mathbf{v}_i$, initial model parameters Θ
Output: Model's optimal parameters Θ^*

while *unfinished* **do**
  Shuffle tasks in $\mathcal{T}_{\text{meta-train}}$
  for $\mathbf{T}_i \in \mathcal{T}_{\text{meta-train}}$ **do**
    Sample support sets and query sets
    Generate support and query set representations $\mathbf{S}, \mathbf{Q}$
    for $\mathbf{q}_i \in \mathbf{Q}$ **do**
      Match with support set representations in $\mathbf{S}$
      Accumulate the loss by Eq. 7
    end for
    Update parameters Θ by Adam optimizer
  end for
end while
return Θ^*

Experiment

Table 1: Statistics of three datasets.

Datasets	Steam	Electronic	Tmall
# sequences	358, 228	443, 589	918, 358
# items	11, 969	63, 002	624, 221
# ground-truth items	9, 496	50, 331	219, 560
Propotion of meta-sequences	0.12	0.31	0.20

framework with following representative and state-of-the-art methods: (1) RNN-based methods including GRU4Rec (Hidasi et al. 2015) and NARM (Li et al. 2017). (2) CNN-based method Caser (Tang and Wang 2018). (3) Attention-based methods including STAMP (Liu et al. 2018) and SASRec (Kang and McAuley 2018). (4) Graph-based method SR-GNN (Wu et al. 2019).

Experiment

Table 2: Performance comparison of different sequential recommendation methods with (w) or without (w/o) Mecos. The best performing methods are **boldfaced** and the best baseline results are indicated by underline.

Dataset	Metric	GRU4Rec		NARM		Caser		STAMP		SASRec		SR-GNN		Avg.
		w/o	w	w/o	w	w/o	w	w/o	w	w/o	w	w/o	w	Improv.
Steam	HR@5	0.0860	0.2018	0.0862	0.1850	0.0839	**0.2029**	0.0780	0.1861	0.0621	0.1431	0.1119	0.1878	121.33%
	HR@10	0.1498	0.3066	0.1578	0.3002	0.1508	**0.3136**	0.1392	0.2969	0.1113	0.2358	0.1885	0.3085	98.61%
	HR@20	0.2567	0.4544	0.2785	**0.4583**	0.2634	0.4500	0.2441	0.4539	0.2074	0.3751	0.3137	0.4561	70.70%
	NDCG@5	0.0533	**0.1308**	0.0531	0.1206	0.0518	0.1304	0.0486	0.1184	0.0400	0.0955	0.0701	0.1183	129.23%
	NDCG@10	0.0737	0.1639	0.0760	0.1561	0.0732	**0.1651**	0.0681	0.1529	0.0557	0.1245	0.0947	0.1650	112.60%
	NDCG@20	0.1005	0.1991	0.1062	0.1951	0.1014	**0.1997**	0.0944	0.1920	0.0798	0.1588	0.1261	0.1945	89.23%
	MRR	0.0720	0.1412	0.0543	0.1380	0.0728	**0.1456**	0.0689	0.1353	0.0676	0.1138	0.0936	0.1454	95.05%
Electronic	HR@5	0.0745	0.1389	0.0843	0.1479	0.0438	0.1300	0.0464	0.0888	0.0394	0.1029	0.1215	**0.1619**	107.41%
	HR@10	0.1357	0.2431	0.1604	0.2546	0.0870	0.2285	0.0924	0.1652	0.0796	0.1858	0.1935	**0.2591**	91.10%
	HR@20	0.2470	0.3970	0.2949	**0.4158**	0.1740	0.3878	0.1794	0.2995	0.1573	0.3217	0.3236	0.4138	70.65%
	NDCG@5	0.0457	0.0856	0.0507	0.0931	0.0255	0.0807	0.0286	0.0546	0.0230	0.0618	0.0655	**0.0975**	115.98%
	NDCG@10	0.0652	0.1180	0.0751	0.1277	0.0393	0.0975	0.0432	0.0817	0.0359	0.0890	0.0950	**0.1324**	95.92%
	NDCG@20	0.0930	0.1565	0.1088	**0.1682**	0.0610	0.1522	0.0650	0.1122	0.0553	0.1231	0.1202	0.1678	84.53%
	MRR	0.0684	0.1061	0.0699	0.1131	0.0440	0.0903	0.0515	0.0736	0.0514	0.0915	0.0921	**0.1243**	63.01%
Tmall	HR@5	0.2863	0.3947	0.3105	0.4005	0.1006	0.3384	0.2791	0.3719	0.0862	0.3215	0.3263	**0.4030**	105.49%
	HR@10	0.3437	0.4666	0.3860	**0.4727**	0.1724	0.4023	0.3373	0.4386	0.1307	0.3694	0.4081	0.4624	69.59%
	HR@20	0.4371	0.5632	0.4832	**0.5703**	0.2966	0.5070	0.4252	0.5348	0.2086	0.4487	0.5133	0.5616	44.68%
	NDCG@5	0.2470	0.3409	0.2747	**0.3450**	0.0640	0.2914	0.2409	0.3245	0.0578	0.2883	0.2548	0.3375	147.48%
	NDCG@10	0.2655	0.3641	0.2957	**0.3680**	0.0870	0.3107	0.2597	0.3456	0.0721	0.3040	0.2921	0.3609	116.16%
	NDCG@20	0.2889	0.3883	0.3201	**0.3927**	0.1181	0.3339	0.2817	0.3697	0.0915	0.3232	0.3264	0.3848	90.36%
	MRR	0.2580	0.3477	0.2848	0.3510	0.0799	0.2996	0.2520	0.3327	0.0659	0.2984	0.2956	**0.3635**	123.46%

Experiment

Table 3: Performance of ablation on different components.

Dataset	Metric	Mecos_R	Variant_1	Variant_2	Variant_3
Steam	HR@10	**0.1606**	0.0933	0.1163	0.0716
	NDCG@10	**0.0805**	0.0445	0.0555	0.0339
	MRR	**0.0551**	0.0405	0.0476	0.0294
Electronic	HR@10	**0.1382**	0.0832	0.1207	0.0770
	NDCG@10	**0.0665**	0.0382	0.0572	0.0329
	MRR	**0.0658**	0.0355	0.0505	0.0314
Tmall	HR@10	**0.3481**	0.3135	0.3230	0.2921
	NDCG@10	**0.2915**	0.2787	0.2737	0.2552
	MRR	**0.2891**	0.2461	0.2503	0.2324

Ablation Study (RQ2). We conduct an ablation study to investigate the contribution of each component. To remove the influence of pre-trained embedding models, we conduct our framework based on randomly initialized item embeddings (Mecos_R). And the following variants of it are tested on all datasets, where the results are reported in Table 3:

(Variant_1) Mecos_R without pair encoder.

(Variant_2) Mecos_R without matching processor.

(Variant_3) Mecos_R without pair encoder and matching

Experiment

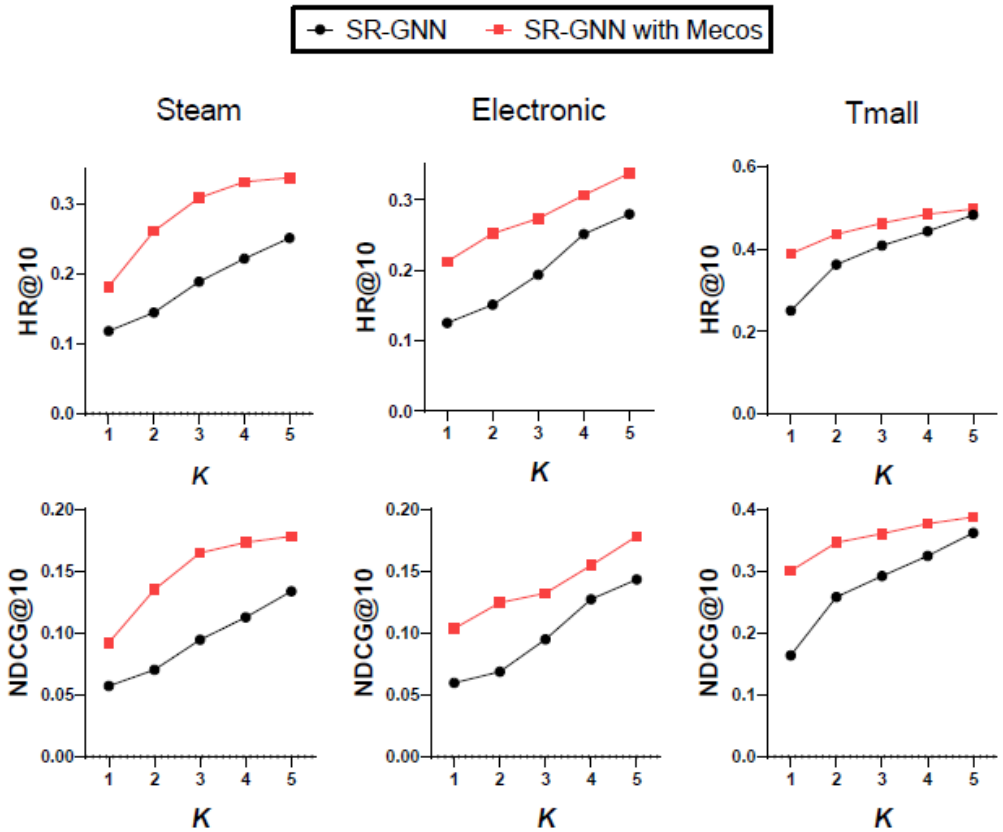

Figure 4: Performance w.r.t. the value of few-shot size K. The performance of *SR-GNN with Mecos* consistently outperforms SR-GNN.

AAAI2021

Self-Supervised Hypergraph Convolutional Networks for Session-based Recommendation

Xin Xia[1], Hongzhi Yin[1]*, Junliang Yu[1], Qinyong Wang[1], Lizhen Cui[2], Xiangliang Zhang[3]

[1]The University of Queensland, [2]Shandong University,
[3]King Abdullah University of Science and Technology
xin.xia1@uqconnect.edu.au, {h.yin1, jl.yu, qinyong.wang}@uq.edu.au, clz@sdu.edu.cn, xiangliang.zhang@kaust.edu.sa

motivation

- In real scenarios, an item transition is often triggered by the joint effect of previous item clicks, and many-to-many and high-order relations exist among items. Obviously, simple graphs are incapable of depicting such set-like relations.
- a hypergraph is composed of a vertex set and a hyperedge set, where a hyperedge can connect any numbers of vertices, which can be used to encode high-order data correlations.
- we first model each sessionas a hyperedge in which all the items are connected with each other, and different hyperedges, which are connected via shared items, constitute the hypergraph that contains the item-level high-order correlations. Then a line graph is built based on the hypergraph by modeling each hyperedge as a node and focuses on the connectivity of hyperedges, which depicts the session-level relations.
- the two channels in our network can be seen as two different views that describe the intra- and inter-information of sessions, while each of them knows little information of the other. By maximizing the mutual information between the session representations learned via the two channels through self-supervised learning, the two channels can acquire new information from each other to improve their own performance in item/session feature extraction.

contribution

- We propose a novel dual channel hypergraph convolutional network for SBR, which can capture the beyondpairwise relations among items and the cross-session information through hypergraph modeling.
- We innovatively integrate a self-supervised task into the training of our network to enhance hypergraph modeling and improve the recommendation task.
- Extensive experiments show that our proposed model has overwhelming superiority over the state-of-the-art baselines and achieves statistically significant improvements on benchmark datasets.

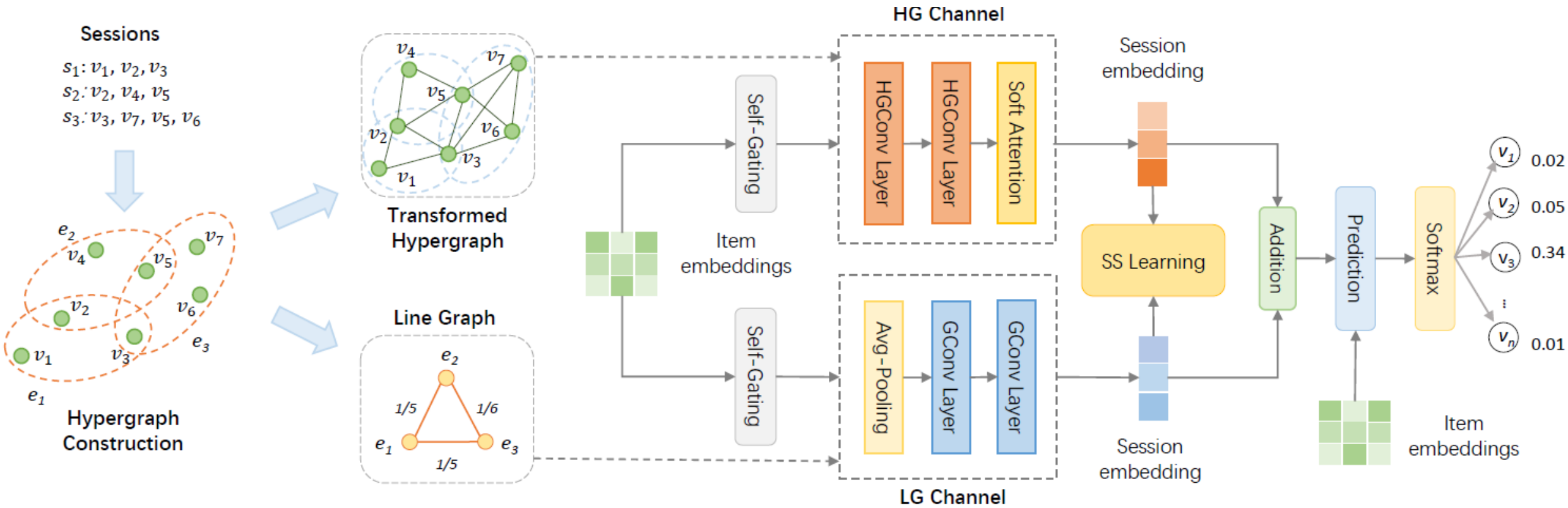

Figure 1: The construction of hypergraph and the pipeline of the proposed DHCN model.

The Proposed Method

Notations and Definitions

Let $I = \{i_1, i_2, i_3, ..., i_N\}$ denote the set of items, where N is the number of items. Each session is represented as a set $s = [i_{s,1}, i_{s,2}, i_{s,3}, ..., i_{s,m}]$ and $i_{s,k} \in I (1 \leq k \leq m)$ represents an interacted item of an anonymous user within the session s. We embed each item $i \in I$ into the same space and let $\mathbf{x}_i^{(l)} \in \mathbb{R}^{d^{(l)}}$ denote the vector representation of item i of dimension d^l in the l-th layer of a deep neural network. The representation of the whole item set is denoted as $\mathbf{X}^{(l)} \in \mathbb{R}^{N \times d^{(l)}}$. Each session s is represented by a vector $\mathbf{s}$. The task of SBR is to predict the next item, namely $i_{s,m+1}$, for any given session s.

- It should be noted that we transform the session sequences into an undirected graph, which is in line with our intuition that items in a session are temporally related instead of sequentially dependent.

Definition 1. Hypergraph. Let $G = (V, E)$ denote a hypergraph, where V is a set containing N unique vertices and E is a set containing M hyperedges. Each hyperedge $\epsilon \in E$ contains two or more vertices and is assigned a positive weight $W_{\epsilon\epsilon}$, and all the weights formulate a diagonal matrix $\mathbf{W} \in \mathbb{R}^{M \times M}$. The hypergraph can be represented by an incidence matrix $\mathbf{H} \in \mathbb{R}^{N \times M}$ where $H_{i\epsilon} = 1$ if the hyperedge $\epsilon \in E$ contains a vertex $v_i \in V$, otherwise 0. For each vertex and hyperedge, their degree D_{ii} and $B_{\epsilon\epsilon}$ are respectively defined as $D_{ii} = \sum_{\epsilon=1}^{M} W_{\epsilon\epsilon} H_{i\epsilon}$; $B_{\epsilon\epsilon} = \sum_{i=1}^{N} H_{i\epsilon}$. $\mathbf{D}$ and $\mathbf{B}$ are diagonal matrices.

Definition 2. Line graph of hypergraph. Given the hypergraph $G = (V, E)$, the line graph of the hypergraph $L(G)$ is a graph where each node of $L(G)$ is a hyperedge in G and two nodes of $L(G)$ are connected if their corresponding hyperedges in G share at least one common node (Whitney 1992). Formally, $L(G) = (V_L, E_L)$ where $V_L = \{v_e : v_e \in E\}$, and $E_L = \{(v_{e_p}, v_{e_q}) : e_p, e_q \in E, |e_p \cap e_q| \geq 1\}$. We assign each edge (v_{e_p}, v_{e_q}) a weight $W_{p,q}$, where $W_{p,q} = |e_p \cap e_q| / |e_p \cup e_q|$.

Hypergraph Channel and Convolution.

channel encodes the hypergraph. As there are two channels, directly feeding the full base item embeddings $\mathbf{X}^{(0)} \in \mathbb{R}^{N\times d}$ to two channels is unwise. To control the magnitude of embedding flowing to each channel, we employ a pre-filter with *self-gating units* (SGUs), which is defined as:

$$\mathbf{X}_c^{(0)} = f_{\text{gate}}^c(\mathbf{X}^{(0)}) = \mathbf{X}^{(0)} \odot \sigma(\mathbf{X}^{(0)}\mathbf{W}_g^c + \mathbf{b}_g^c), \quad (1)$$

where $\mathbf{W}_g^c \in \mathbb{R}^{d\times d}, \mathbf{b}_g^c \in \mathbb{R}^d$ are gating parameters to be learned, $c \in \{h, l\}$ represents the channel, $\odot$ denotes the element-wise product and σ is the sigmoid function.

lution proposed in (Feng et al. 2019), we define our hypergraph convolution as:

$$\mathbf{x}_i^{(l+1)} = \sum_{j=1}^{N}\sum_{\epsilon=1}^{M} H_{i\epsilon}H_{j\epsilon}W_{\epsilon\epsilon}\mathbf{x}_j^{(l)}\mathbf{P}^{(l)}, \quad (2)$$

where $\mathbf{P}^{(l)} \in \mathbb{R}^{d\times d}$ is the learnable parameter matrix between two convolutional layers. Following the suggestions in (Wu et al. 2019a), we do not use nonlinear activation function. For $W_{\epsilon\epsilon}$, we assign each hyperedge the same weight 1. The matrix form of Eq. (1) with row normalization is:

$$\mathbf{X}_h^{(l+1)} = \mathbf{D}^{-1}\mathbf{H}\mathbf{W}\mathbf{B}^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{X}_h^{(l)}\mathbf{P}^{(l)}. \quad (3)$$

The hypergraph convolution can be viewed as a two-stage refinement performing 'node-hyperedge-node' feature transformation upon hypergraph structure. The multiplication operation $\mathbf{H}^\top\mathbf{X}_h^{(l)}$ defines the information aggregation from nodes to hyperedges and then premultiplying $\mathbf{H}$ is viewed to aggregate information from hyperedges to nodes.

Hypergraph Channel and Convolution.

the final item embeddings $\mathbf{X}_h^* = \frac{1}{L+1}\sum_{l=0}^{L}\mathbf{X}_h^{(l)}$.

the embedding of session $s = [i_{s,1}, i_{s,2}, i_{s,3}, ..., i_{s,m}]$:

$$\begin{aligned} \alpha_t &= \mathbf{f}^\top \sigma\left(\mathbf{W}_1\mathbf{x}_m^* + \mathbf{W}_2\mathbf{x}_t^* + \mathbf{c}\right), \\ \mathbf{s}_g &= \sum_{t=1}^{m}\alpha_t\mathbf{x}_t^*, \quad \theta_h = \mathbf{W}_3[\mathbf{x}_m^*; \mathbf{s}_g], \end{aligned} \tag{4}$$

where $\mathbf{x}_m^*$ is the embedding of the last item in session s, and $\mathbf{x}_t^*$ is the embedding of the t-th item in session s. Let $\mathbf{x}_m^*$ denote the current user intent, and the user's general interest embedding $\mathbf{s}_g$ across this session is represented by aggregating item embeddings through a soft-attention mechanism where items have different levels of priorities. $\mathbf{f} \in \mathbb{R}^d$, $\mathbf{W}_1 \in \mathbb{R}^{d\times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{d\times d}$ are attention parameters

Line Graph Channel and Convolution

tivity of hyperedges. Before the convolution operation, analogously, we pass $\mathbf{X}^{(0)}$ through the SGU to obtain the line graph channel-specific item embeddings $\mathbf{X}_l^{(0)}$. As there are no item involved in the line graph channel, we first initialize the channel-specific session embeddings $\mathbf{\Theta}_l^{(0)}$ by looking up the items belonged to each session and then averaging the corresponding items embeddings in $\mathbf{X}_l^{(0)}$. An incidence matrix for $L(G)$ is defined as $\mathbf{A} \in \mathbb{R}^{M\times M}$ where M is the number of nodes in the line graph and $A_{p,q} = W_{p,q}$ according to Definition 2. Let $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ where $\mathbf{I}$ is an identity matrix. $\hat{\mathbf{D}} \in \mathbb{R}^{M\times M}$ is a diagonal degree matrix where $\hat{\mathbf{D}}_{p,p} = \sum_{q=1}^{m}\hat{\mathbf{A}}_{p,q}$. The line graph convolution is then defined as:

$$\mathbf{\Theta}_l^{(l+1)} = \hat{\mathbf{D}}^{-1}\hat{\mathbf{A}}\mathbf{\Theta}^{(l)}\mathbf{Q}^{(l)}, \tag{5}$$

where $\mathbf{Q}^{(l)} \in \mathbb{R}^{d\times d}$ is the weight matrix. In each convolution, the sessions gather information from their neighbors. By doing so, the learned $\mathbf{\Theta}$ can capture the cross-session information. Likewise, we pass $\mathbf{\Theta}_l^{(0)}$ through L graph convolutional layer, and then average the session embeddings obtained at each layer to get the final session embeddings $\mathbf{\Theta}_l = \frac{1}{L+1}\sum_{l=0}^{L}\mathbf{\Theta}_l^{(l)}$.

Model Optimization and Recommendation Generation Given a session s, we compute scores $\hat{\mathbf{z}}$ for all the candidate items $i \in I$ by doing inner product between the base item embedding $\mathbf{X}^0$ and θ_s^h and θ_s^l, respectively. Then we add up the two predicted scores to get the final predictions:

$$\hat{\mathbf{z}}_i = (\theta_s^h + \theta_s^l)^T \mathbf{x}_i. \tag{6}$$

After that, a softmax function is applied to compute the probabilities of each item being the next one in the session:

$$\hat{\mathbf{y}} = \mathrm{softmax}(\hat{\mathbf{z}}). \tag{7}$$

We formulate the learning objective as a cross entropy loss function, which has been extensively used in recommender systems and defined as:

$$\mathcal{L}_r = -\sum_{i=1}^{N} \mathbf{y}_i \log\left(\hat{\mathbf{y}}_i\right) + \left(1 - \mathbf{y}_i\right) \log\left(1 - \hat{\mathbf{y}}_i\right), \tag{8}$$

where $\mathbf{y}$ is the one-hot encoding vector of the ground truth. For simplicity, we leave out the L_2 regularization terms. By minimizing $\mathcal{L}_r$ with Adam, we can get high-quality session-based recommendations.

Enhancing DHCN with Self-Supervised Learning

(1) Creating self-supervision signals.

augmentation. If two session embeddings both denote the same session in two views, we label this pair as the ground-truth, otherwise we label it as the negative.

(2) Contrastive learning.

cross-entropy loss between the samples from the ground-truth (positive) and the corrupted samples (negative) as our learning objective and defined it as:

$$\mathcal{L}_s = -\log \sigma(f_{\mathrm{D}}(\theta_i^h, \theta_i^l)) - \log \sigma(1 - f_{\mathrm{D}}(\tilde{\theta}_i^h, \theta_i^l)), \quad (9)$$

where $\tilde{\theta}_i^h$ (or $\tilde{\theta}_i^l$) is the negative sample obtained by corrupting Θ_h (Θ_l) with row-wise and column-wise shuffling, and $f_{\mathrm{D}}(\cdot) : \mathbb{R}^d \times \mathbb{R}^d \longmapsto \mathbb{R}$ is the discriminator function that takes two vectors as the input and then scores the agreement between them. We simply implement the discriminator as the dot product between two vectors. This learning objective

Finally, we unify the recommendation task and this self-supervised task into a *primary&auxiliary* learning framework, where the former is the primary task and the latter is the auxiliary task. Formally, the joint learning objective is defined as:

$$\mathcal{L} = \mathcal{L}_r + \beta \mathcal{L}_s, \quad (10)$$

Experiment

Table 1: Dataset Statistics

Dataset	Yooch1/64	Yooch1/4	Diginetica
training sessions	369,859	5,917,745	719,470
test sessions	55,898	55,898	60,858
# of items	16,766	29,618	43,097
average lengths	6.16	5.71	5.12

Table 2: Performances of all comparison methods on three datasets.

Method	Yoochoose1/64				Yoochoose1/4				Diginetica			
	P@10	M@10	P@20	M@20	P@10	M@10	P@20	M@20	P@10	M@10	P@20	M@20
Item-KNN	48.69	24.94	51.60	21.81	50.50	25.78	52.31	21.70	25.07	10.77	35.75	11.57
FPMC	37.44	20.05	45.62	15.01	-	-	-	-	15.43	6.20	26.53	6.95
GRU4REC	52.43	24.53	60.64	22.89	55.49	26.05	59.53	22.60	17.93	7.33	29.45	8.33
NARM	57.83	27.42	68.32	28.63	57.98	28.51	69.73	29.23	35.44	15.13	49.70	16.17
STAMP	58.07	28.92	68.74	29.67	59.62	29.24	70.44	30.00	33.98	14.26	45.64	14.32
SR-GNN	60.21	30.12	70.57	30.94	61.06	31.08	71.36	31.89	36.86	15.52	50.73	17.59
FGNN	60.97	30.85	71.12	31.68	61.98	32.43	71.97	32.54	37.72	15.95	51.36	18.47
DHCN	**64.02**	**36.75**	**73.43**	**37.34**	**68.72**	**36.96**	**77.61**	**36.72**	**51.45**	**30.05**	**61.62**	**30.58**
S^2-DHCN	**64.33**	**37.32**	**74.10**	**37.89**	**70.01**	**39.50**	**78.96**	**40.13**	**52.52**	**30.42**	**63.20**	**30.94**
Improv. (%)	5.51	20.97	4.19	19.60	12.95	21.80	9.71	23.32	39.23	90.72	23.06	67.51