



Adaptive self-propagation graph convolutional network for recommendation

Zhuo Cai¹ · Guan Yuan¹ · Xiaobao Zhuang¹ · Senzhang Wang² · Shaojie Qiao³ · Mu Zhu⁴

Received: 22 September 2022 / Revised: 29 December 2022 / Accepted: 18 May 2023 /
Published online: 1 July 2023

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2023

Abstract

Graph Convolutional Networks (GCNs) have received a lot of attention in recommender systems due to their powerful representation learning ability on graph data. Depending on whether to combine ego embeddings after aggregating neighbor embeddings, GCNs can be divided into two categories: without self-propagation and with self-propagation. (1) The GCNs without self-propagation bring the loss of inherent information (e.g., income, age) because of discarding ego embeddings. (2) The existing GCNs with self-propagation treat all nodes (i.e., users and items) indistinguishably, so that the distinctive and diverse characteristics of users and items are overlooked. In light of these problems existed in two types of GCNs, we propose a novel GCN model, Adaptive Self-propagation Graph Convolutional Network (ASP-GCN), to retain inherent information and distinctive characteristics of users and items simultaneously. We first conduct pilot experiments to prove that existing GCNs actually suffer from the aforementioned problems. Then, we use Gumbel-Softmax trick to generate categorical distributions for each node in each layer between two types of embeddings: neighborhood embeddings and hybrid embeddings. Neighborhood embeddings are the aggregation of neighbor embeddings and hybrid embeddings consist of ego and neighborhood embeddings. Next, user and item embeddings are updated by aggregating these two types of embeddings proportionally according to corresponding categorical distributions. After obtaining node embeddings in the last convolution layer, the Bayesian Personalized Ranking loss optimized with a similarity term is used to refine the model parameters. Comprehensive experiments are conducted on three benchmark datasets to demonstrate the effectiveness of ASP-GCN over present state-of-the-art approaches.

Keywords Graph convolutional networks · Self-propagation · Gumbel-Softmax · Recommender systems

✉ Guan Yuan
yuanguan@cumt.edu.cn

Extended author information available on the last page of the article

1 Introduction

Graph Convolutional Networks (GCNs), which have the ability to explore the topological information of a user-item bipartite graph, have become frequently used in recommender systems in recent years [1–6]. Depending on whether to combine ego embeddings after aggregating neighbor embeddings, GCNs can be divided into two types: without self-propagation [3, 7] and with self-propagation [2, 8, 9].

The GCNs without self-propagation update user and item embeddings by aggregating embeddings of their neighbors only. Taking user u for example, the updated embedding of user u is $\sum_{i \in \mathcal{N}_u} \frac{1}{p_{u,i}} e_i$, where $p_{u,i}$ is the normalization constant which has several versions: $\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}$ (symmetric normalization), $|\mathcal{N}_u|$ (left normalization), $|\mathcal{N}_i|$ (right normalization) and other variations. \mathcal{N}_u and \mathcal{N}_i are neighbor sets of user u and item i respectively. e_i is the embedding of item i . The GCNs with self-propagation combine ego embeddings with neighbor embeddings in the stage of embedding propagation. For instance, NGCF [9] and LR-GCCF [2] add self-loop in the propagation matrix, PinSage [4] concatenates ego embeddings with neighborhood embeddings and performs feature transformation and non-linearity to update embeddings.

These two types of GCNs both have limitations: **(1)** The GCNs without self-propagation only aggregate neighbor embeddings while ignoring ego embeddings. This kind of embedding propagation mechanism discards ego embeddings, leading to the loss of nodes’ inherent information. As shown in Figure 1(a), user u_1 ’s income is in a relatively high level. Assuming that u_1 had purchased some inexpensive daily requirements like a pen (i_1) and a notebook (i_2). According to the principle of embedding propagation, the value of the income feature of u_1 is updated to 0.2 after performing GCN. As a result, u_1 ’s income feature is diminished to a low level, which is contrary to her inherent attributes. Under this circumstance, relatively

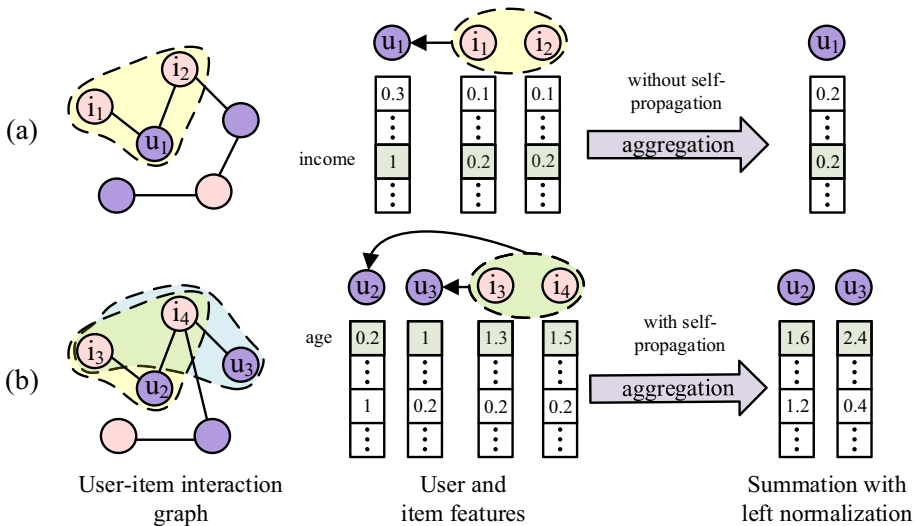


Fig. 1 A toy example of target users’ embedding update process in item recommendation, where user and item embeddings are described by a set of numerical features. (a) is the aggregation process without self-propagation and (b) is the aggregation process with self-propagation. In this example, we use left normalization as the coefficient of neighbor embeddings and choose summation as the combination way of ego embeddings and neighborhood embeddings in (b)

expensive items are in a low priority for u_1 in recommender systems, leading to the inability to meet the needs of target user. **(2)** Existing GCNs with self-propagation although combine ego embeddings with neighbor embeddings, they adopt a uniform manner treating all nodes. These kinds of GCNs neglect the individuality and differences of users and items. Taking Figure 1(b) for example, u_2 and u_3 both had read two history books (i_3 and i_4). As the age feature shows, u_2 and u_3 have a big gap about age and u_3 is much older than u_2 . Assuming that u_3 read history books for interest and u_2 just read them to finish the homework of history lessons. However, the relative gap between u_2 and u_3 narrows after the embedding propagation with uniform self-propagation. It obliterates the personalized differences between users in some extent. In this case, recommender systems will pretend to recommend items with similar styles to u_2 and u_3 , although they have different demands because of the age gap.

To tackle aforementioned issues, we propose a novel GCN model, Adaptive Self-propagation Graph Convolutional Network (ASP-GCN). Considering the importance of original features and the personalized properties and attributes of users and items, it's necessary to combine ego embeddings and neighborhood embeddings individually in the process of embedding propagation. A simple method is using a weight matrix and updating this matrix iteratively. However, the value of weight matrix is independent of ego embeddings and neighborhood embeddings, which reduces the model interpretability and recommendation performance. A solution is using a possibility vector generation module to generate aggregation proportions according to ego and neighborhood embeddings. Gumbel-Softmax is such a trick that can achieve this goal. Hence, ASP-GCN uses Gumbel-Softmax trick to generate categorical distributions between two types of embeddings: neighborhood embeddings and hybrid embeddings. Hybrid embeddings consist of ego and neighborhood embeddings. After obtaining the categorical distributions, ASP-GCN proportionally aggregates neighborhood embeddings and ego embeddings in an adaptive way. This kind of propagation rule can simultaneously retain nodes' inherent information and capture their distinctive characteristics. Note that, ASP-GCN is essentially different with some methods focusing on the residual connection [10, 11]. ASP-GCN can capture fine-granular user properties and item attributes layer by layer while methods focusing on residual connection only handle this in a monolithic or coarse grained way. ASP-GCN uses the embeddings from the last layer for prediction because each layer proportionally retains embeddings from previous layer, contributing to the unnecessary of layer combination. Moreover, we optimized the Bayesian Personalized Ranking (BPR) loss function [12] with a similarity term. Besides the role of BPR, similarity term makes the embeddings of connected nodes close to each other and that of disconnected nodes far to each other because connected nodes have better reflection to each other.

To summarize, the main contributions of our paper are as follows:

- We study existing GCN-based recommendation models and empirically divide them into two categories: GCNs without self-propagation and GCNs with self-propagation. We hold the view that both of them have corresponding limitations and we conduct a pilot experiment to prove our idea.
- We propose a novel GCN-based model, Adaptive Self-propagation Graph Convolutional Network (ASP-GCN). ASP-GCN updates node embeddings by aggregating ego embeddings and neighborhood embeddings proportionally according to the categorical distribution estimated by Gumbel-Softmax.
- Extensive experiments conducted on three publicly available datasets show that ASP-GCN outperforms several state-of-the-art approaches, which verifies the effectiveness of ASP-GCN.

2 Related work

In this section, we introduce the related work of two relevant technologies used in our work: Graph Convolutional Networks (GCNs) for recommendation and Gumbel-Softmax used to generate categorical distributions.

2.1 GCNs for recommendation

Graph Convolutional Networks (GCNs) generalize traditional Convolutional Neural Networks (CNNs) from Euclidean space to graph domain. GCNs have been widely used in node classification [13–15], link prediction [16–19], traffic flow forecasting [20–23] and other fields due to the remarkable capacity of GCNs in learning graph representations. In recommender systems, interactions between users and items can also be represented as graph structures, leading to the boost of GCNs in this field.

According to the domain that the convolution operations were applied on, existing GCN-based approaches can be divided into two categories: spectral GCNs [24–27] and spatial GCNs [3, 9, 28, 29]. Spectral GCNs perform convolution operations on the spectral domain to refine eigenvectors. They are computationally expensive because of the consumptive operations like Laplacian eigen-decomposition [27] and Chebyshev polynomials [24]. To tackle this problem, spatial GCNs have been proposed, which refine node (users and items) embeddings by aggregating neighbor embeddings, GC-MC [1] applies graph convolutional network on recommender systems to exploit information from user-item interaction graph structure. However, it stacks only one convolution layer, which means that only the first-order connectivities can be captured. This kind of graph structure is insufficient to model the high-order similarities of users and items, leading to the loss of useful information. To solve this problem, recent studies shine a light on stacking multiple convolution layers to exploit high-hop similarities of users and items from user-item interaction graph [2, 3, 9, 30, 31]. NGCF [9] and PinSage [4] propose to use multi-layer graph structure to capture high-order collaborative filtering signals to update user and item embeddings. Despite success, the feature transformation and non-linearity operations involved in the convolution process lead to the rise of time and space consumption. To tackle this problem, LR-GCCF [2] removes the non-linearity operation. Making a further step, LightGCN [3] removes feature transformation and non-linearity simultaneously, considering these two operations are not only consumptive but also make model difficult to train.

Since ASP-GCN belongs to the domain of spatial GCNs, we compare ASP-GCN with GC-MC, PinSage, NGCF, LR-GCCF and LightGCN from the following perspectives: (1) whether uses feature transformation; (2) whether uses non-linearity; (3) whether uses a residual network structure to make prediction; (4) whether has self-propagation. The comparison details are shown in Table 1.

Table 1 The comparison between ASP-GCN and other state-of-the-art GCN-based methods

	GC-MC	PinSage	NGCF	LR-GCCF	LightGCN	ASP-GCN
Feature Transformation	O	O	O	O	X	O
Non-linearity	O	O	O	X	X	O
Residual Prediction	X	O	O	O	O	X
Self-propagation	X	O	O	O	X	O

2.2 Gumbel-Softmax

The Gumbel-max trick [32–35] provides a view to generate a one-hot vector over a categorical distribution, which can be used to select features. However, the argmax operation involved is non-differentiable so that the gradient flow is not allowed for Gumbel-max. For this reason, Gumbel-max’s application in neural networks is limited. To this end, Gumbel-Softmax [36, 37] is proposed to handle the non-differentiability problem. Gumbel-Softmax distribution is a continuous distribution over the simplex distribution that appropriates the categorical distribution via reparameterization trick. It has been widely used in learning optimal categorical distributions. One such application is network architecture search (NAS) [38–42]. In these methods, Gumbel-Softmax is used to generate optimal distributions between several pre-defined operations so that to optimize the neural architecture. Another example is graph sparsification [43, 44], in which Gumbel-Softmax is utilized to judge whether an edge is preserved in the sparsified graph. By doing this, the time and space complexity of Graph-based methods can be reduced and the model robustness can be improved. Moreover, Kong et al. [45] explore to investigate which of the linear and non-linear propagation is better in recommender system. Hence, it utilizes Gumbel-Softmax to generate categorical distributions between two modes of nodes in the propagation stage: the linear and non-linear characteristics.

3 Motivation

To demonstrate the motivation of our research, we conduct a pilot experiment to verify the innovation rationality of ASP-GCN. Specifically, we design two variants based on LightGCN and compare the performances of the three methods.

- **LightGCN**, which is a typical GCN-based method without self-propagation in the embedding propagation stage. The convolution operation (a.k.a, propagation rule) is: $\sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} e_i$ (taking user u for example), where \mathcal{N}_u and \mathcal{N}_i are neighbor sets of user u and item i ; e_i is embedding of item i .
- **LightGCN-s**, which adds uniform self-propagation in the embedding propagation stage. The convolution operation (a.k.a, propagation rule) is: $mean(e_u, \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} e_i)$. Because of the self-propagation mechanism, we only use the user and item embeddings from the last convolution layer to estimate user preferences over items.
- **LightGCN-ws**, which adds weighted self-propagation in the embedding propagation stage. The convolution operation (a.k.a, propagation rule) is: $w_1 e_u + w_2 \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} e_i$, where w_1 and w_2 are trainable weights processed by *softmax* function. Here, we also use the user and item embeddings from the last convolution layer to estimate user preferences over items.

For these three methods, we maintain the same value for each hyperparameter. Specifically, the embedding size is fixed to 16 and the number of layers is set to 3. Besides, other hyperparameters (e.g., learning rate, regularization coefficient, dropout ratio) are all set to the same values. We report the performances of these three methods on Gowalla and Yelp datasets in Figure 2.

As shown in Figure 2, LightGCN-s has similar tendency with LightGCN in terms of training loss, but has higher testing Recall and NDCG values on both Gowalla and Yelp datasets.

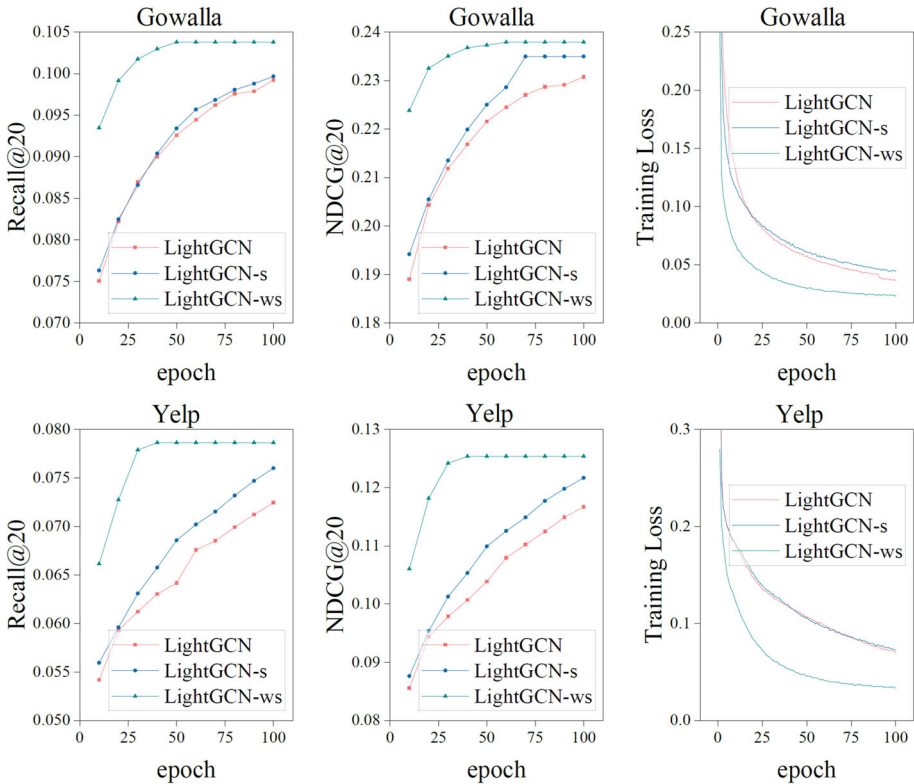


Fig. 2 Training curves (testing Recall, NDCG and training loss) of LightGCN and its two variants on Gowalla and Yelp datasets

It indicates that self-propagation plays positive effect in embedding propagation stage, which can improve recommendation performance to some extent. Further, LightGCN-ws consistently outperforms LightGCN and LightGCN-s in terms of testing Recall and NDCG on two datasets significantly. Besides, as can be seen, LightGCN-ws accelerates the convergence speed of GCNs to a great extent. From these evidences, we can infer that both GCNs without self-propagation and GCNs with a uniform self-propagation manner are insufficient to model user preferences and provide satisfying recommendation. The propagation rule with weighted self-propagation is an effective and efficient way compared with the aforementioned two kinds of GCNs.

Although effective, LightGCN-ws just resorts to a simple way to aggregate neighbor embedding and ego embedding in a weighted way. It uses the trainable weights to determine the proportion between neighbor embedding and ego embedding. By doing so, it can mitigate the problem of both GCNs without self-propagation and uniform self-propagation. However, the weights are independent to the user and item embeddings, leading to the high randomness of the proportion in practice, which also limits the recommendation performance to some extent. Therefore, it is urged to take user and item embeddings into account to estimate the proportion between neighbor embedding and ego embedding using an adaptive way. This inspires us to design ASP-GCN.

4 Methodology

In this section, we first present the overall workflow of ASP-GCN. Then, we illustrate how to use Gumbel-Softmax trick to generate the categorical distributions between two types of embeddings: neighborhood embeddings and hybrid embeddings. Next, we introduce the multi-layer convolution operations of ASP-GCN, which are used to update user and item embeddings. Finally, optimized BPR loss function used to refine model parameters is demonstrated. The overall workflow of ASP-GCN is shown in Figure 3.

4.1 Category distribution generation module

In some existing GCN-based methods, the embeddings involved in embedding aggregation procedure are only neighbor embeddings, while ego embeddings are ignored. For example, the graph convolution operation (a.k.a., propagation rule) in LightGCN is defined as (taking user u for example):

$$[e_u^{(1)}]_N = \sum_{i \in N_u} \frac{1}{\sqrt{|N_u||N_i|}} e_i^{(0)} \tag{1}$$

“ N ” denotes that there are only neighbor embeddings used in the update stage. $e_i^{(0)}$ is the initial embedding of item i and $e_u^{(1)}$ is the updated embedding of user u . N_u and N_i are the neighbor sets of user u and item i .

This kind of GCNs discard ego embeddings while aggregating neighbor embeddings, leading to the loss of inherent information of users and items. Considering the importance of original features, some other GCN-based methods combine ego embeddings when updating embeddings of target node. For instance, NGCF [9] adds self-loop in the propagation matrix and PinSage [4] concatenates ego embeddings after neighborhood aggregation. Although ego embeddings are involved in these methods, they adopt a uniform way to handle these two types of embeddings. Specifically, LR-GCCF’s propagation rule for user u is

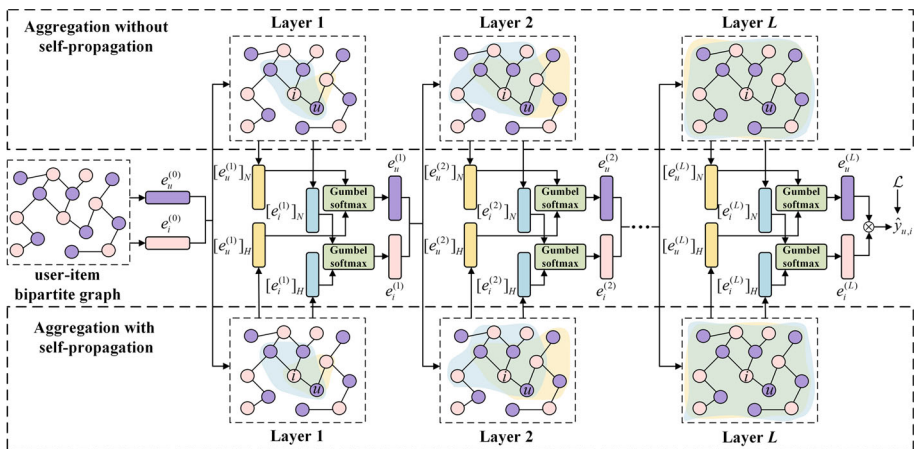


Fig. 3 The overall workflow of ASP-GCN. The upper part is the neighborhood embeddings generation process. The lower part is the hybrid embeddings generation process. The middle part is the adaptive aggregation process, in which neighborhood embeddings and hybrid embeddings aggregated proportionally

$(\frac{1}{|\mathcal{N}_u|} \mathbf{e}_u + \sum_{i \in \mathcal{N}_u} \frac{1}{|\mathcal{N}_u||\mathcal{N}_i|} \mathbf{e}_i) \mathbf{W}$ [2], where \mathbf{W} is the trainable feature transformation matrix.

The proportions between ego embeddings and neighbor embeddings are determined by the neighbor numbers of target node and its neighbor node. In other words, once the graph structure is fixed, the proportions between ego embeddings and neighbor embeddings are fixed in the embedding aggregation stage. PinSage updates the embedding of target node by concatenating ego embedding and aggregated neighbor embeddings: $\sigma((\mathbf{e}_u || \mathbf{e}_{\mathcal{N}_u}) \mathbf{W})$. It makes ego embeddings and neighborhood embeddings share the same rule in the embedding propagation process. These kinds of methods treat ego embeddings and neighbor embeddings in a uniform way, while neglecting the different rules of ego embedding and neighbor embedding in the embedding update stage for each node. To tackle the aforementioned problems, we design another kind of convolution operation that combines ego embeddings and neighborhood embeddings:

$$[\mathbf{e}_u^{(1)}]_H = \mathbf{e}_u^{(0)} + \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} \mathbf{e}_i^{(0)} \tag{2}$$

“ H ” denotes that there are both neighbor embeddings and ego embeddings used in the update stage. After obtaining two kinds of update signals for user u as shown in Eqs. 1 and 2, it is necessary to estimate the proportion of each of them in the process of embedding aggregation. Once obtaining corresponding proportions of each kind of update signal for each user and item, the ego embeddings and neighbor embeddings can be aggregated proportionally. By doing so, the problems illustrated before can be mitigated. To achieve this goal, one simple method is assign two weights for each user and item to represent the aggregation proportion of each kind of update signal in the embedding update process. In other words, model initial a weight matrix and refine this weight matrix to learn the proportions. However, this kind of processing is not optimal because the weights are independent of ego embeddings and neighbor embeddings. This brings a large randomness to the model, which would reduce the model interpretability and further limit the recommendation performance. Gumbel-Softmax trick is good solution to handle this problem, which can generate a categorical distribution according to the provided ego and neighbor embeddings. Hence, in ASP-GCN, Gumbel-Softmax trick is resorted to generate a categorical distribution between $[\mathbf{e}_u^{(1)}]_N$ and $[\mathbf{e}_u^{(1)}]_H$ adaptively:

$$\alpha_{u,k} = \frac{\exp((\log \pi_{u,k} + g_{u,k})/\tau)}{\sum_{j=1}^{|\pi_u|} \exp((\log \pi_{u,j} + g_{u,j})/\tau)} \tag{3}$$

where $\alpha_{u,k}$ is the k -th element of user u 's categorical distribution. $g_{u,1}, \dots, g_{u,|\pi|}$ are independently identical distribution (i.i.d) samples drawn from Gumbel(0,1) distribution, which can be sampled via inverse transform sampling as: $g_{u,k} = -\log(-\log(a))$, $a \sim \text{Uniform}(0, 1)$. τ is the temperature factor of Gumbel-softmax. π_u is a two dimension vector, in which $\pi_{u,1}$ and $\pi_{u,2}$ are the class possibilities of $[\mathbf{e}_u^{(1)}]_N$ and $[\mathbf{e}_u^{(1)}]_H$ respectively. $\pi_{u,1}$ and $\pi_{u,2}$ are deduced by a multi-layer perceptron (MLP):

$$\begin{cases} \pi_{u,1} = \sigma_2(\sigma_1([\mathbf{e}_u^{(1)}]_N \mathbf{W}_1 + \mathbf{b}_1) \mathbf{W}_2 + \mathbf{b}_2) \\ \pi_{u,2} = \sigma_2(\sigma_1([\mathbf{e}_u^{(1)}]_H \mathbf{W}_1 + \mathbf{b}_1) \mathbf{W}_2 + \mathbf{b}_2) \end{cases} \tag{4}$$

where $\mathbf{W}_1 \in \mathbb{R}^{d \times d'}$, $\mathbf{W}_2 \in \mathbb{R}^{d' \times 1}$ and $\mathbf{b}_1 \in \mathbb{R}^{1 \times d'}$, $\mathbf{b}_2 \in \mathbb{R}^{1 \times 1}$ are trainable weight matrixes and biases. $\sigma_1(\cdot)$ and $\sigma_2(\cdot)$ are the *SELU* [46] and *Sigmoid* functions respectively.

It's worth noting that we use *SELU* as the activation function of the first layer of MLP, In the second layer of MLP, we use *Sigmoid* instead of *SELU* as the output of MLP to

generate a possibility. It is because the output of $SELU$ is not a possibility. $SELU$ is proved to have a better performance than $tanh$, $ReLU$ and $LeakyReLU$ for two reasons: a) it can achieve internal normalization which converges faster than external normalization; b) it can avoid the problem of gradient vanishing and gradient explosion. The definition of $SELU$ is as:

$$SELU(x) = \gamma \begin{cases} x & \text{if } x > 0 \\ \eta(e^x - 1) & \text{if } x \leq 0 \end{cases}, \quad (5)$$

where γ and η are hyperparameters, predefined as 1.67326 and 1.05070 respectively.

4.2 Adaptive self-propagation

Intuitively, the interacted items can directly reflect a user's preference; analogously, the users that observed an item can be modeled to describe item features. According to this natural principle, it is necessary to aggregate neighbor embeddings to update the embeddings of target nodes (shown in Eq. 2). Meanwhile, considering the importance of original features users and items, we design another propagation rule which considers ego embeddings and neighbor embeddings simultaneously (shown in Eq. 2). Besides, Gumbel-Softmax trick is utilized to generate categorical distributions between neighborhood embeddings and hybrid embeddings. After obtaining the categorical distributions generated by Gumbel-Softmax, the embeddings are propagated by performing these two types of propagation rules proportionally. By doing this, the node embedding can be propagated to the node itself, which is called self-propagation.

We first introduce the first-order embedding propagation of ASP-GCN that aggregates embeddings of the first-order neighbors, and then generalize it to multiple convolution layers.

First-order propagation. The basic idea of ASP-GCN is to aggregate ego embeddings and neighbor embeddings proportionally. Specifically, we perform the propagation rules as shown in Eqs. 1 and 2 proportionally according to the categorical distributions estimated by the categorical distribution generation module. So, the first-order propagation rule (taking user u for example) in ASP-GCN is:

$$\begin{aligned} e_u^{(1)} &= \alpha_{u,1}^{(1)}[e_u^{(1)}]_N + \alpha_{u,2}^{(1)}[e_u^{(1)}]_H \\ &= \alpha_{u,1}^{(1)} \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} e_i^{(0)} + \alpha_{u,2}^{(1)}(e_u^{(0)} + \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} e_i^{(0)}) \\ &= \alpha_{u,2}^{(1)} e_u^{(0)} + (\alpha_{u,1}^{(1)} + \alpha_{u,2}^{(1)}) \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} e_i^{(0)} \\ &= \alpha_{u,2}^{(1)} e_u^{(0)} + \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} e_i^{(0)}, \end{aligned} \quad (6)$$

where $e_u^{(0)}$ and $e_i^{(0)}$ are initial embeddings of user u and item i ; $e_u^{(1)}$ is user u 's updated embedding from the first layer. $[\alpha_{u,1}^{(1)}, \alpha_{u,2}^{(1)}]$ is the categorical distribution of user u in the first layer. To note that, the categorical distribution generation module should be operated in each layer for each node. It is because embedding propagation process in different convolution layers involves different nodes, making it inappropriate to perform categorical distribution

generation module only once for all layers. Operating categorical distribution generation module in each layer for each node is beneficial to capture finer-grained user preferences and item attributes.

What mentioned above is the embedding update process of a single node. To show the holistic process of embedding update and facilitate the implementation, we provide the matrix form of embedding propagation of ASP-GCN:

$$E^{(1)} = \alpha_1^{(1)} \hat{A}E^{(0)} + \alpha_2^{(1)} (E^{(0)} + \hat{A}E^{(0)}) = \alpha_2^{(1)} E^{(0)} + \hat{A}E^{(0)}, \tag{7}$$

where $E^{(0)} \in \mathbb{R}^{(|\mathcal{U}|+|\mathcal{I}|) \times d}$ and $E^{(1)} \in \mathbb{R}^{(|\mathcal{U}|+|\mathcal{I}|) \times d}$ are the initial embedding matrix and updated embedding matrix of all users and items in the first layer respectively. d is the embedding size $[\alpha_1^{(1)} \in \mathbb{R}^{(|\mathcal{U}|+|\mathcal{I}|) \times 1}, \alpha_2^{(1)} \in \mathbb{R}^{(|\mathcal{U}|+|\mathcal{I}|) \times 1}]$ is the categorical distributions for all users and items in the first layer. $\hat{A} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ is the symmetric normalized adjacency matrix, where D is diagonal degree matrix and $A \in \mathbb{R}^{(|\mathcal{U}|+|\mathcal{I}|) \times (|\mathcal{U}|+|\mathcal{I}|)}$ is the adjacency matrix. Single layer graph convolutional network updates user and item embeddings by aggregating first-order neighbors' embeddings. It captures the direct connectivities of users and items to model users' preferences and items' attributes. This can be seen as the first-order similarity.

High-order propagation. However, aggregating the embeddings of first-hop neighbors is insufficient to capture complex collaborative information implied in the user-item interaction graph. To exploit the high-order similarity of users and items, we stack multiple convolution layers to aggregate the embeddings of high-order neighbors. The recurrence formula is as:

$$E^{(l)} = \alpha_2^{(l)} E^{(l-1)} + \hat{A}E^{(l-1)}, l = 1, 2, \dots, L, \tag{8}$$

$E^{(l)}$ and $E^{(l-1)}$ are embedding matrixes of all users and items in layer l and $(l - 1)$ respectively. By implementing the multi-layer matrix-form propagation rule, we can update all user and item embeddings for each convolution layer in an efficient way.

4.3 Prediction

After L layers' embedding propagation, user and item embeddings from each layer can be obtained. Most existing GCN-based methods adopt residual connection to get the final embeddings of users and items. Taking user u for example, NGCF concatenates the embeddings from each layer: $e_u^{(0)} \parallel \dots \parallel e_u^{(L)}$; LightGCN uses the average of embeddings from each layer as the final embeddings: $\frac{1}{L+1} \sum_{l=0}^L e_u^{(l)}$. Different from these methods, ASP-GCN regards the embeddings from the last layer as the final embeddings. It is because the embeddings from previous layer are proportionally retains in current layer, contributes to the unnecessary of layer combination. If ASP-GCN also adopts a residual connection manner, the ego embeddings would occupy too much proportion in the final embeddings, which would lead to the deterioration of the representation ability of user and item embeddings. Therefore, ASP-GCN regards the user and item embeddings from the last layer as the final embeddings. Taking user u and item i for example, the final embeddings are:

$$e_u^* = e_u^{(L)}, e_i^* = e_i^{(L)} \tag{9}$$

After getting the final embeddings, we conduct inner product operation to estimate the preference score of user u over item i :

$$\hat{y}_{u,i} = e_u^* \otimes e_i^{*T} \tag{10}$$

In this work, we only use inner product to compute the preference score of users over target items. The inner product operation is proved to be efficient yet simple. Some other operations can also be performed to replace inner product, such as neural network [47]. We leave it to the future work since it is not the focus of our work.

4.4 Model optimization

To refine model parameters, we employ the pairwise BPR loss [12], which has been widely used in recommender systems. The BPR loss function is written as:

$$\mathcal{L}_{\text{BPR}} = - \sum_{(u, i^+, i^-) \in \mathcal{O}} \ln \sigma(\hat{y}_{u, i^+} - \hat{y}_{u, i^-}) + \lambda \|\Theta\|_2^2, \quad (11)$$

where $\mathcal{O} = \{(u, i^+, i^-) | (u, i^+) \in \mathbf{R}^+, (u, i^-) \in \mathbf{R}^-\}$ is the pairwise training data, \mathbf{R}^+ denotes the positive user-item pair set (observed interactions) and \mathbf{R}^- denotes the negative user-item pair set (unobserved interactions). $\sigma(\cdot)$ is the sigmoid function and Θ denotes all model parameters. \hat{y}_{u, i^+} and \hat{y}_{u, i^-} represent the preference scores of user u over item i^+ and i^- respectively. λ is the coefficient to control the strength of regularization term. BPR loss assumes that positive items (namely the observed items) can better reflect user preference than negative items (namely the unobserved items). Hence, it assigns higher predicted value to the positive items than to the negative items. The regularization term is used to avoid overfitting.

In addition to the regularization term, we add a similarity term. The similarity term makes the embeddings of connected nodes close to each other and that of disconnected nodes far to each other because connected nodes have better reflects to each other. The similarity loss function is as follows:

$$\mathcal{L}_S = - \sum_{(u, i^+, i^-) \in \mathcal{O}} (s(\mathbf{e}_u^*, \mathbf{e}_{i^+}^*) - s(\mathbf{e}_u^*, \mathbf{e}_{i^-}^*)) \quad (12)$$

where $s(\cdot, \cdot)$ is cosine similarity function. \mathbf{e}_u^* is the final embedding of user u . $\mathbf{e}_{i^+}^*$ and $\mathbf{e}_{i^-}^*$ are final embeddings of item i^+ and i^- respectively. The goal of similarity term is to enlarge the gap between the similarity of positive interactions and negative interactions. But if we use $s(\mathbf{e}_u^*, \mathbf{e}_{i^+}^*) - s(\mathbf{e}_u^*, \mathbf{e}_{i^-}^*)$ as the similarity term, the gradient would refine the model weights into a reverse direction, in which the similarity gap between positive and negative interactions would be narrowed. Considering this, it's natural to add a negative operation in the similarity term. Finally, we combine the BPR loss and similarity loss in a multi-task learning way:

$$\mathcal{L} = \mathcal{L}_{\text{BPR}} + \beta \mathcal{L}_S \quad (13)$$

where β is the coefficient to control the strength of similarity term. By introducing the similarity term to the loss function, the overall loss function can guide the model to refine the model parameter to cater to the two tasks simultaneously. We think this kind of combination practice has a certain degree of universality. The similarity term can be combined with not only BPR loss, but also many other loss functions such as Binary Cross-Entropy Loss [47] and Square Loss [48, 49] with a small adjustment. Besides, this kind of practice can also be performed in many other fields, including node classification and link prediction.

4.5 Time complexity analysis

Assuming that the number of convolution layer is L and the dimension of MLP is $\mathbb{R}^{d_0 \times d_1}$ and $\mathbb{R}^{d_1 \times 1}$. The main operation of ASP-GCN is matrix multiplication and categorical distribution generation. The time complexity of categorical distribution generation is $O(\sum_{l=0}^L (|\mathcal{U}| + |\mathcal{I}|)(d + 1)d')$, where $|\mathcal{U}| + |\mathcal{I}|$ is the number of all nodes (users and items) and d, d' are the dimensions of node embedding and the first hidden layer of MLP. The time complexity of matrix multiplication is $O(\sum_{l=0}^L |R^+|d)$, where $|R^+|$ denotes the number of nonzero entities the adjacent matrix. Therefore, the overall time complexity of ASP-GCN is $O(\sum_{l=0}^L |R^+|d + (|\mathcal{U}| + |\mathcal{I}|)(d + 1)d')$.

4.6 Model analysis

Relation with NGCF. NGCF [9] is a neural graph collaborative filtering model, which exploits user-item graph structure by propagating embeddings on it. NGCF adds self-loop in the adjacent matrix (i.e., $A_{vv} = 1$, where v is the index of a user or an item), so that ego embedding can be propagated to node itself in the embedding propagation process. However, it's insufficient because ego embedding only occupies a very small proportion in the embedding update procedure. Meanwhile, once the graph structure is fixed, the proportions of ego embeddings and neighbor embeddings are fixed during the process of embedding propagation. ASP-GCN adopts an adaptive self-propagation mechanism, in which ego and neighbor embeddings can be aggregated proportionally according to the categorical distributions generated by Gumbel-Softmax trick. By doing so, ASP-GCN can mitigate the problem of NGCF: the individual difference of user properties and item attributes is not fully captured.

Relation with LightGCN. LightGCN [3] removes the feature transformation matrix and non-linear activation function in the embedding propagation process, considering these two operations are burdensome and even make model hard to train. However, ego embeddings are not involved in the embedding propagation process of LightGCN, resulting in the loss of inherent information of users and items. Although LightGCN adopts a residual connection mechanism to combine embeddings from each layer, the connection is in a layer-wise manner, which means that the difference of different nodes are not distinguished. ASP-GCN generates categorical distributions for each user or item to update its embedding, which pays special attention to the distinction between different users or items.

Relation with DGCF. DGCF [50] is a disentangled graph collaborative filtering model, which pays attention to the finer-grained user intents. However, DGCF is similar with LightGCN essentially since they are all GCNs without self-propagation. Although DGCF yields disentangled representations for users and items, the operations of DGCF are time consumptive, which limits its generalization. ASP-GCN is essentially different from DGCF. DGCF learns the finer granularity of user intents by dividing user and item embeddings into several intents-aware trunks, and the update process of each trunk is independent. ASP-GCN generates individual categorical distributions for users and items to guide the process of embedding propagation, which does not increase too much time and space consumption for model, so that own a higher generalization.

Table 2 Statistics of datasets

Datasets	#user	#item	#interactions	Sparsity ¹
Gowalla	18,737	32,510	741,906	99.878%
Yelp	30,887	18,995	745,197	99.873%
Movielens-100K	943	1,682	100,000	93.695%

¹The ratio of elements without interaction history to the whole matrix space

5 Experimental analysis

In this section, we conduct extensive experiments on three benchmark datasets to evaluate ASP-GCN and answer the following research questions:

- **RQ1:** Does ASP-GCN has a better performance compared with present state-of-the-art methods?
- **RQ2:** Is adaptive self-propagation mechanism of ASP-GCN effective?
- **RQ3:** How does ASP-GCN perform under different aggregation mechanisms?
- **RQ4:** Why we regard the embeddings from the last layer as the final embeddings to conduct prediction?
- **RQ5:** How different hyper parameters (including temperature factor τ , the coefficient β of similarity term and model depth L) affect the result of ASP-GCN?

5.1 Datasets and evaluation metrics

We use three publicly available datasets: Gowalla, Yelp, Movielens-100K to conduct our experiments and the statistics of them are shown in Table 2.

To evaluate the Top- N recommendation, we select Precision@ N , Recall@ N , HR@ N and NDCG@ N as the evaluation metrics. The calculation formulas are as follows:

$$\text{Recall@}N = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|S_{u,N} \cap T_u|}{|T_u|}, \quad (14)$$

$$\text{Precision@}N = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|S_{u,N} \cap T_u|}{N}, \quad (15)$$

$$\text{F1@}N = 2 \times \frac{\text{Recall@}N \times \text{Precision@}N}{\text{Recall@}N + \text{Precision@}N} \quad (16)$$

$$\text{NDCG@}N = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \sum_{p=1}^N \frac{I(S_{u,N}(p) \cap T(p))}{Z_u \cdot \log_2(p+1)}, \quad (17)$$

where $S_{u,N}$ is the set of top N items recommended to user u and T_u is the ground-truth item set of user u in the testing set. $S_{u,N}(p)$ denotes the item set of p -th element in $S_{u,N}$. $I(\cdot)$ is an indicator function, that equals to 1 if the set is not empty, otherwise 0. Z_u is the ideal discount cumulative gain so that a perfect recommended list obtains $\text{NDCG}_u=1$.

5.2 Baselines

- **BPR-MF** [12]: BPR-MF is a matrix factorization model optimized by Bayesian Personalized Ranking (BPR) loss, which exploits the user-item interactions to compute the loss and refine model parameters.
- **GMF** [47]: GMF is a generalized matrix factorization recommendation model, which uses a linear kernel to calculate users' preference over items.
- **MLP** [47]: MLP is a Multi-Layer Perceptron based model, which uses a non-linear kernel to estimate users' preference over items.
- **NCF** [47]: NCF is the combination of GMF and MLP, which can simultaneously capture the linear and non-linear feature of users and items.
- **GC-MC** [1]: GC-MC is a GCN-based model, which stacks only one convolution layer to update node embeddings by aggregating the embeddings of first-order neighbors.
- **PinSage** [4]: PinSage is the implement of GraphSage in large Web-scale recommender systems.
- **NGCF** [9]: NGCF is a GCN-based collaborative model, that stacks multiple convolution layer to update user and item embeddings. It additionally encodes the interactions of target node and neighbor node, compared with convolutional GCNs.
- **LR-GCCF** [2]: LR-GCCF is a linear GCN-based collaborative filtering model, whose embedding propagation is linear. It only uses feature transformation and removes the non-linearity in the aggregation stage.
- **LightGCN** [41]: LightGCN is a simplifying and powering GCN model that removes feature transformation and non-linearity simultaneously. It improves the recommendation performance, while reducing the memory and time consumption.
- **DGCF** [50]: DGCF is a disentangled graph collaborative filtering method, which pay special attention to user-item relationships at the finer granularity of user intents. It disentangles user intents and yield disentangled representations so as to improve the robustness and interpretability of recommendation model.
- **SGL** [51]: SGL is the implement of self-supervised learning on user-item interaction graph with the aim of improving the accuracy and robustness of GCNs for recommendation. We select Edge Dropout (SGL-ED) and Random Walk (SGL-RW) as the competitive models since they generally show better performance.
- **CIGCN** [52]: CIGCN is an embedding disentanglement model, which designs a channel independent graph convolutional network to disentangle user and item embeddings. It assigns different dimension of embeddings with different importance to update embedding dimension independently. To notice, we don't use the item-item relations since there are no item-item relation data in the datasets we used.

5.3 Experimental environment and parameter settings

Experimental environment: ASP-GCN is implemented using pytorch framework and speed up by a NVIDIA 2080Ti GPU.

Parameter settings: For the sake of fairness, the embedding sizes of all methods are fixed to 64. For all multi-layer GCN-based methods, we search the layer number in $\{1, 2, 3, 4, 5\}$ for the best performance and the size of each layer is set to 64. The batch size is selected in range $\{256, 512, 1024, 2048\}$ and the learning rate is searched in $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$ to tune the convergence speed. The temperature factor τ is selected in range $\{1, 5, 10, 50, 100\}$. Moreover, the coefficient of similarity term and regularization term is searched in

range $\{0.01, 0.05, 1\}$ and $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ respectively. Meanwhile, we use a dropout strategy for all models and search it in $\{0.1, 0.2, \dots, 0.8\}$. For every methods, one negative sample is selected to match each positive sample. Xavier [53] and Adam [54] are used to initialize and optimize the model parameters of ASP-GCN respectively.

5.4 Performance comparison (RQ1)

From the results shown in Tables 3, 4 and 5 (The boldfaced values mean the best performance in each column, and the underlined values indicate the best performance of baselines), we have the following observations:

- ASP-GCN has the best recommendation performance over other present baselines on both three datasets, which indicates the effectiveness of ASP-GCN proposed in this paper. By generating categorical distributions between neighborhood embeddings and hybrid embeddings and aggregating these two kinds of update signals according to the generated categorical distribution, ASP-GCN can benefit the GCN model for retaining nodes' (users' and items') inherent information and capturing the users' personalized needs.
- BPR has poor performance on Gowalla and Yelp datasets consistently. It samples the user-item interactions to calculate the loss and then refine user and item embeddings using backward propagation. This indicates that simply perform inner product between users and items is insufficient.
- GCN-based methods (GC-MC, PinSage, NGCF, LR-GCCF, LightGCN, DGCF, CIGCN) have better performance than BPR, GMF, MLP and NCF. They use graph convolutional network to exploit high-order connectivities of users and items. In particular, GC-MC's performance is worse than multi-layer based methods such as LR-GCCF and LightGCN. This is because GC-MC stacks only one convolution layer, which can only exploit the first-order connectivities and fails to capture the high-order similarity of users or items.

Table 3 The comparison of overall performance with baseline methods on Gowalla dataset

<i>N</i> Metrics	10				20			
	Recall	Precision	F1	NDCG	Recall	Precision	F1	NDCG
BPR-MF	0.05818	0.04282	0.04933	0.16885	0.08806	0.03298	0.04799	0.20934
GMF	0.05996	0.04432	0.05097	0.18015	0.09202	0.03440	0.05008	0.22451
MLP	0.04592	0.03400	0.03907	0.13690	0.07374	0.02755	0.04011	0.17722
NCF	0.05738	0.04201	0.04851	0.16377	0.09020	0.03367	0.04904	0.20983
GC-MC	0.06531	0.04820	0.05547	0.18182	0.10071	0.03775	0.05492	0.22819
PinSage	0.06074	0.04471	0.05151	0.16912	0.09600	0.03576	0.05211	0.21573
NGCF	0.06488	0.04769	0.05497	0.17723	0.10128	0.03769	0.05494	0.22428
LR-GCCF	0.06450	0.04733	0.05460	0.18253	0.09915	0.03710	0.05400	0.22853
LightGCN	0.07114	0.05198	0.06007	0.19239	0.11017	0.04095	0.05971	0.24170
DGCF	0.06582	0.04848	0.05583	0.18162	0.10175	0.03789	0.05522	0.22767
SGL-ED	0.07178	0.05255	0.06068	0.19412	0.11049	0.04101	0.05982	0.24205
SGL-RW	0.07226	0.05274	0.06098	<u>0.19530</u>	0.11221	<u>0.04167</u>	<u>0.06077</u>	<u>0.24594</u>
CIGCN	<u>0.07263</u>	<u>0.05302</u>	<u>0.06129</u>	0.19425	<u>0.11236</u>	0.04163	0.06075	0.24372
ASP-GCN	0.07586	0.05547	0.06408	0.20061	0.11690	0.04341	0.06331	0.25220

Table 4 The comparison of overall performance with baseline methods on Yelp dataset

N Metrics	10				20			
	Recall	Precision	F1	NDCG	Recall	Precision	F1	NDCG
BPR-MF	0.04074	0.02086	0.02759	0.08186	0.06985	0.01808	0.02872	0.11210
GMF	0.03316	0.01720	0.02265	0.07081	0.05589	0.01474	0.02333	0.09644
MLP	0.03091	0.01545	0.02060	0.06242	0.05420	0.01372	0.02190	0.08787
NCF	0.03328	0.01737	0.02283	0.07021	0.05696	0.01506	0.02382	0.09667
GC-MC	0.04431	0.02231	0.02968	0.08649	0.07277	0.01881	0.02989	0.11660
PinSage	0.04363	0.02221	0.02944	0.08728	0.07336	0.01887	0.03002	0.11817
NGCF	0.04577	0.02295	0.03057	0.08980	0.07630	0.01943	0.03097	0.12136
LR-GCCF	0.04349	0.02236	0.02953	0.08704	0.07183	0.01880	0.02980	0.11718
LightGCN	0.04722	0.02388	0.03172	0.09430	0.07993	0.02035	0.03244	0.12750
DGCF	0.04571	0.02323	0.03080	0.09093	0.07644	0.01968	0.03130	0.12277
SGL-ED	0.05007	0.02582	0.03407	0.09867	<u>0.08225</u>	<u>0.02149</u>	<u>0.03408</u>	<u>0.13464</u>
SGL-RW	<u>0.05016</u>	<u>0.02584</u>	<u>0.03411</u>	<u>0.09902</u>	0.08150	0.02145	0.03396	0.13440
CIGCN	0.04838	0.02439	0.03243	0.09499	0.08031	0.02043	0.03257	0.12747
ASP-GCN	0.05426	0.02736	0.03638	0.10486	0.08955	0.02297	0.03656	0.14032

- Compared with NGCF and LR-GCCF, LighGCN consistently shows better performances on two datasets. It suggests that non-linear activation and feature transformation are two burdensome operations for recommendation tasks. They not only rise the time and memory consumption, but also make the model difficult to train.
- SGL outperforms LightGCN since it performs self-supervised learning on user-item interaction graph. SGL mitigates two problems existed in the GCN-based recommenda-

Table 5 The comparison of overall performance with baseline methods on Movielens-100K dataset

N Metrics	10				20			
	Recall	Precision	F1	NDCG	Recall	Precision	F1	NDCG
BPR-MF	0.22895	0.35016	0.27687	0.52037	0.34394	0.27996	0.30867	0.62982
GMF	0.21044	0.32344	0.25498	0.52341	0.31349	0.26357	0.28637	<u>0.64558</u>
MLP	0.19705	0.31113	0.24129	0.50863	0.30481	0.25286	0.27642	0.63910
NCF	0.20511	0.31220	0.24757	0.52002	0.31284	0.25403	0.28038	0.64048
GC-MC	0.22317	0.34751	0.27179	0.52396	0.34207	0.27953	0.30765	0.63419
PinSage	0.21924	0.33245	0.26423	0.51015	0.33659	0.27169	0.30068	0.62551
NGCF	0.22073	0.34305	0.26862	0.51514	0.34164	0.27842	0.30681	0.63192
LR-GCCF	0.22247	0.34390	0.27017	0.50866	0.33749	0.27497	0.30304	0.61873
LightGCN	0.23271	0.35578	0.28138	0.52655	<u>0.35534</u>	<u>0.28823</u>	<u>0.31829</u>	0.64037
DGCF	0.23175	0.35558	0.28061	0.53092	0.34758	0.28250	0.31168	0.63677
SGL-ED	0.23334	0.35524	0.28167	0.52797	0.34782	0.28335	0.31229	0.63680
SGL-RW	<u>0.23456</u>	<u>0.35790</u>	<u>0.28339</u>	<u>0.52969</u>	0.35235	0.28616	0.31582	0.63822
CIGCN	0.23336	0.35302	0.28098	0.52503	0.34797	0.28351	0.31245	0.63433
ASP-GCN	0.23892	0.36766	0.28963	0.54537	0.36056	0.29290	0.32323	0.65433

tion model: long-tail and noisy interaction problems. By doing so, SGL can improve the robustness of GCN-based recommendation model.

5.5 Ablation experiments

5.5.1 Effectiveness of adaptive self-propagation mechanism (RQ2)

To investigate whether the adaptive self-propagation mechanism of ASP-GCN is helpful to improve recommendation performance, we remove the similarity term in the loss function (denoted by ASP-GCN-s). ASP-GCN-s is the pure adaptive self-propagation mechanism based GCN as compared with other GCN models. From Figure 4, we can find that ASP-GCN-s outperforms all nine comparative methods. It verifies that the adaptive self-propagation mechanism designed in this paper is effective. Concretely, ASP-GCN utilizes the Gumbel-Softmax trick to generate the category distribution between neighborhood embeddings and hybrid embeddings, and aggregates these embeddings proportionally according to the generated categorical distributions. This kind of embedding propagation mechanism captures both ego and neighbor features, which can further improve the recommendation performance.

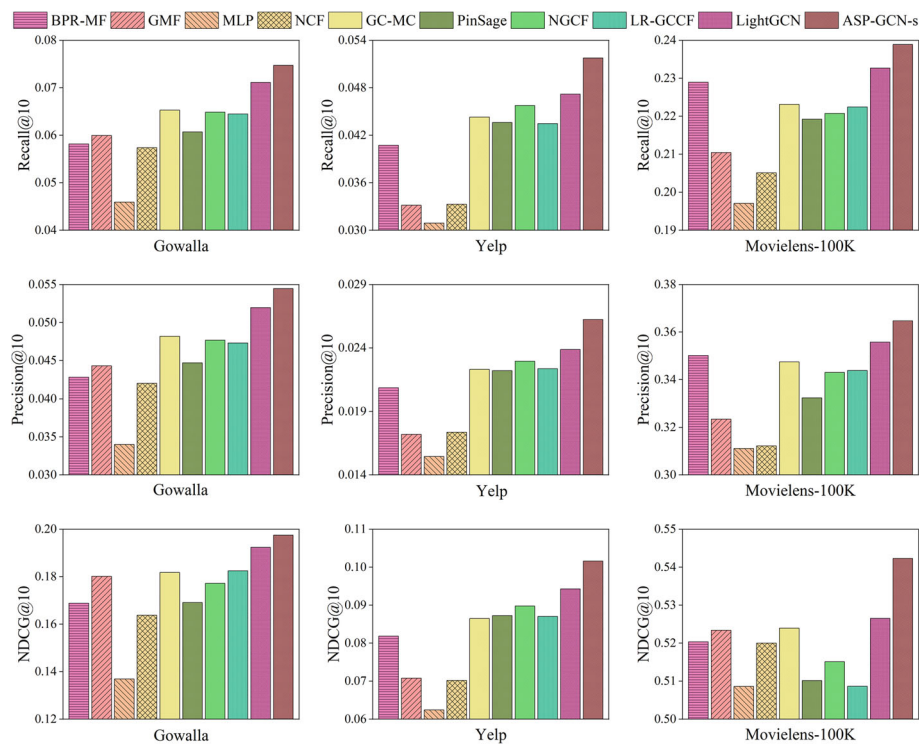


Fig. 4 Performance comparison of ASP-GCN-s (i.e., the variant of ASP-GCN, which removes the similarity term of loss function) and other recommendation models

5.5.2 Impact of different aggregation mechanisms (RQ3)

In ASP-GCN, we employ proportional summation between neighborhood embeddings and hybrid embeddings in embedding aggregation stage. To study its rationality, we also design three variants of ASP-GCN: ASP-GCN-max (performing proportionally max pooling between two types of embeddings), ASP-GCN-concat (proportionally concatenating two types of embeddings) and ASP-GCN-mean (performing proportionally mean pooling between two types of embeddings). From the results shown in Table 6 (The boldfaced values indicate the best performances), we can observe that the best performance in general is ASP-GCN. On Yelp dataset, ASP-GCN-mean has a better performance than the others. But we found that ASP-GCN-mean is hard to train and time consumptive. It is particularly obvious on Movielens-100K dataset, not only rising the time cost, but also deteriorating overall performance.

5.5.3 Impact of layer combination (RQ4)

ASP-GCN regards the embeddings from the last layer as the final embeddings to predict preference scores of users over items. To show why we do this, we design a variant ASP-GCN_{all-layer} that sums embeddings from each layer as the final embeddings of users and items. As shown in Figure 5, ASP-GCN consistently has better recommendation performance than ASP-GCN_{all-layer} on both Gowalla, Yelp and Movielens-100K datasets, which can be explained by ASP-GCN's adaptive self-propagation mechanism. In each layer, ASP-GCN aggregates a certain proportion of hybrid embeddings which consist of ego embeddings and neighbor embeddings. It means that a certain proportion of embeddings from the previous layer can be retained. Hence, it is unnecessary for ASP-GCN to perform layer combination, which would affect the recommendation performance adversely.

5.6 Hyper-parameter analysis (RQ5)

5.6.1 Impact of temperature factor and coefficient of similarity term

The temperature factor τ and the coefficient β of similarity term are two crucial hyper-parameters of ASP-GCN. Performances of ASP-GCN under different τ and β settings are shown in Figure 6.

τ influences the categorical distributions generation process. For low temperatures, the categorical distributions approximate one-hot distributions. For high temperatures, the categorical distributions gradually become uniform distributions. These two extremes will lead

Table 6 Performance of 3-layer ASP-GCN with different choices of aggregation mechanisms in graph convolution

	Gowalla		Yelp		Movielens-100K	
	Recall	NDCG	Recall	NDCG	Recall	NDCG
ASP-GCN-max	0.11072	0.24311	0.08060	0.12862	0.34108	0.63511
ASP-GCN-concat	0.11521	0.24911	0.08576	0.13587	0.35511	0.65310
ASP-GCN-mean	0.11510	0.24982	0.08778	0.13831	0.29763	0.59434
ASP-GCN	0.11573	0.25033	0.08617	0.13606	0.36056	0.65433

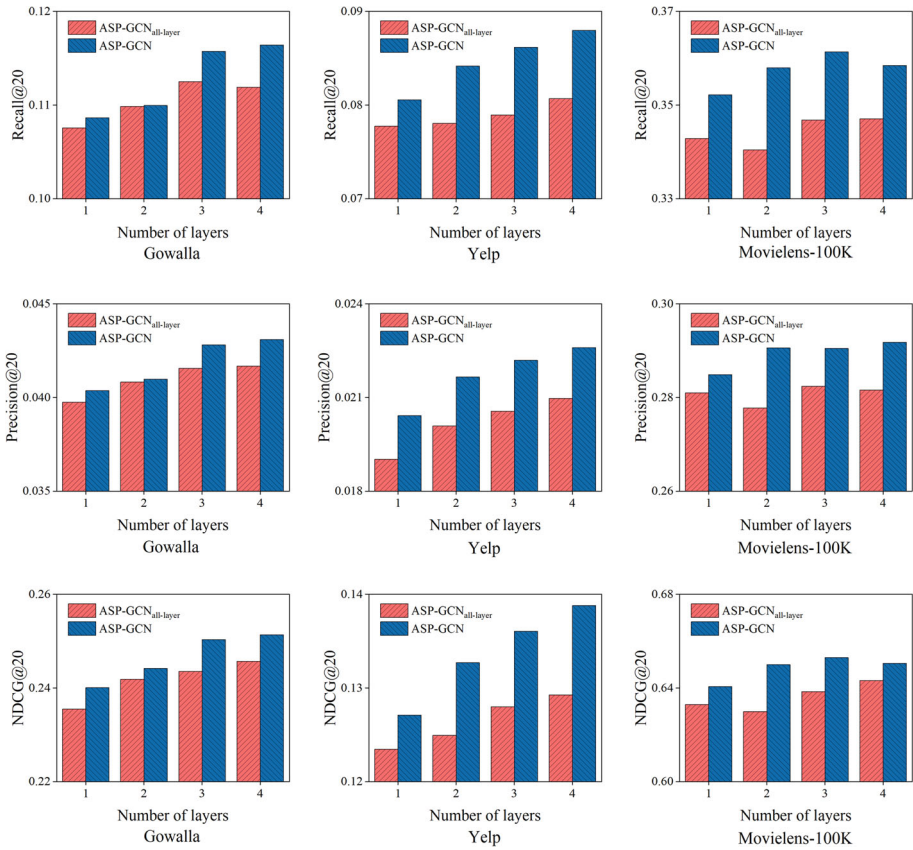


Fig. 5 Results of ASP-GCN and the variant that sums embeddings from each layer under different number of layers on Gowalla and Yelp datasets

to two problems: (1) The first one causes the imbalance of ego embeddings and neighbor embeddings; (2) The second one stifles the differences between nodes. Hence, as shown in Figure 6, the Recall@20 and Precision@20 have showed a pattern that the curve increases early then decreases later as the temperature increases.

β influences the model training process. As shown in Figure 6, Recall@20 decreases after the increasing as β rises. As the coefficient of similarity term increases, the embeddings of connected nodes become closer and disconnected nodes become further iteratively. It is beneficial because connected nodes can better reflect the features of target nodes. This proves the effectiveness of our optimized loss function. But as β increases into a relatively higher level, Recall@20 shows a downward trend. This is because a larger proportion of similarity term will weaken the rule of main loss (BPR), causing ASP-GCN deviate from the main goal.

5.6.2 Impact of model depth

To investigate how model depth affects the performance of ASP-GCN, we search the number of convolution layer in the range {1, 2, ..., 7}. As summarized in Table 7 (The boldfaced values indicate the best performances), we can have the following observation:

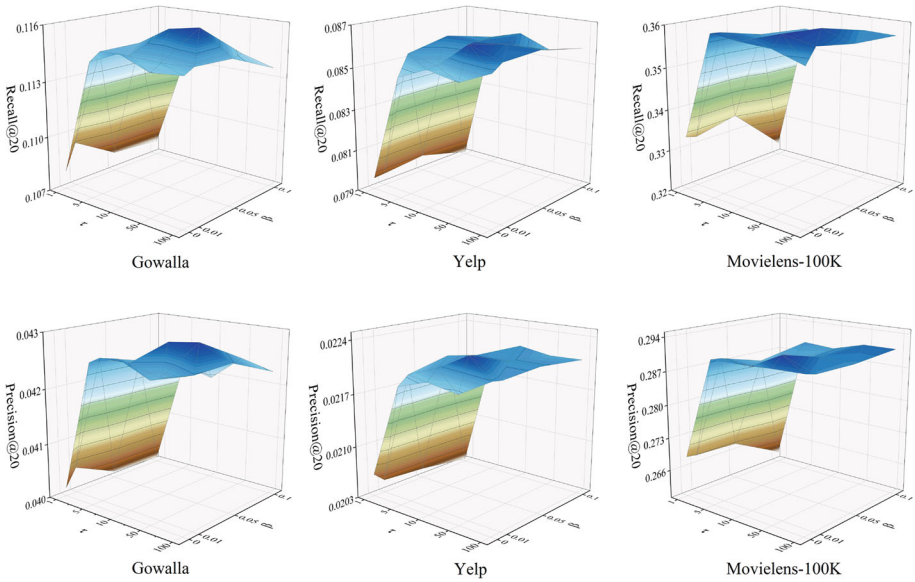


Fig. 6 Performance of 3-layer ASP-GCN under different temperature factors and coefficients of similarity term

- ASP-GCN benefits from the multi-layer graph structure. In particular, the recommendation performance has consistent improvement on Movielens-100K when layer number increases from 1 to 4, on Gowalla when layer number increases from 1 to 5 and on Yelp when layer number increases from 1 to 6. This can be attributed to the exploitation of high-order collaborative signals. By aggregating the embeddings of high-hop neighbors, the high-order collaborative signals can be captured to model the high-order similarity of users and items. Therefore, increasing the number of convolution layers can enhance the model’s representational ability, further improving the recommendation performance.

Table 7 The performance of ASP-GCN under different number of layers

Model depth		1	2	3	4	5	6	7
Gowalla	Recall@10	0.07053	0.07122	0.07499	0.07476	0.07586	0.07550	0.06256
	Precision@10	0.05147	0.05234	0.05465	0.05460	0.05547	0.05516	0.04666
	F1@10	0.05951	0.06034	0.06322	0.06346	0.06408	0.06375	0.05345
	NDCG@10	0.19108	0.19455	0.19919	0.19886	0.20062	0.19878	0.18274
Yelp	Recall@10	0.04831	0.05062	0.05214	0.05323	0.05426	0.05521	0.05502
	Precision@10	0.02566	0.02636	0.02692	0.02736	0.02773	0.02775	0.02607
	F1@10	0.03352	0.03467	0.03551	0.03614	0.03670	0.03694	0.03538
	NDCG@10	0.09427	0.09863	0.10111	0.10375	0.10486	0.10671	0.10509
Movielens -100K	Recall@10	0.23396	0.23956	0.23825	0.24105	0.23624	0.23003	0.22613
	Precision@10	0.35695	0.36394	0.36352	0.36702	0.36108	0.35133	0.34634
	F1@10	0.28266	0.28893	0.28785	0.29099	0.28561	0.27803	0.27361
	NDCG@10	0.53036	0.53947	0.54039	0.54344	0.53815	0.52758	0.52958

- When model depth increases from 5 to 7 on Gowalla, from 6 to 7 on Yelp and from 4 to 7 on MoiveLens-100K, the model witnesses the deterioration. It is because the problem of over-smoothing, which is the reason why GCNs cannot get satisfying performance when increasing the model depth continuously. Deepening the model causes the exponential increase of the number of nodes that involved in the embedding update process of target nodes. Hence, these nodes smooth to each other through the process of embedding propagation. This contributes to the slight discrimination of embeddings, and further limits the recommendation performance.

6 Conclusion

In this work, we hold the view that neither discarding ego embeddings nor combining ego embeddings in a uniform way in the process of embedding propagation is efficient to update node embeddings (user and item embeddings). We conduct a pilot experiment to verify our observation. Considering this issue, we propose an Adaptive Self-propagation Graph Convolutional Network (ASP-GCN) to proportionally aggregate neighbor embeddings and hybrid embeddings composed of ego embeddings and neighborhood embeddings. Specifically, we resort to Gumbel-Softmax trick to generate categorical distributions between aforementioned two types of embeddings. Generated categorical distributions are used as the weight of each type of embeddings in the representation propagation process. In back propagation stage, we optimize BPR loss with a similarity term that forces embeddings of connected nodes close to each other and that of disconnected nodes far to each other. Finally, comprehensive experiments on three publicly available datasets are conducted to prove the effectiveness and efficiency of ASP-GCN.

Acknowledgements We acknowledge the editorial committee's support and all anonymous reviewers for their insightful comments and suggestions, which can improve the content and presentation of this manuscript.

Author Contributions Zhuo Cai wrote the main manuscript text and designed the experiments, Guan Yuan provided the main idea of this manuscript and organized the full text, Xiaobao Zhuang prepared Figures 1-4 and formatted the references. Senzhang Wang checked the English grammar and smoothed the representation of main text, Shaojie Qiao gave many suggestions and instructions in experiments, and Mu Zhu sorted out experimental data. All authors have proofread and approved the final manuscript.

Funding This work was supported in part by the National Natural Science Foundation of China under Grants 71774159 and 62272066, China Postdoctoral Science Foundation under Grants 2021T140707, Jiangsu Postdoctoral Science Foundation under Grants 2021K565C, and State Key Laboratory of NBC Protection for Civilian under grant SKLNBC2020-23.

Availability of data and materials All datasets used in this paper are open datasets.

Declarations

Competing interests The authors declare no competing interests.

References

1. Berg, R.v.d., Kipf, T.N., Welling, M.: Graph convolutional matrix completion. arXiv preprint [arXiv:1706.02263](https://arxiv.org/abs/1706.02263) (2017)

2. Chen, L., Wu, L., Hong, R., Zhang, K., Wang, M.: Revisiting graph based collaborative filtering: A linear residual graph convolutional network approach. In: Proceedings of the AAAI conference on artificial intelligence, vol. 34, pp. 27–34. (2020)
3. He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., Wang, M.: Lightgcn: Simplifying and powering graph convolution network for recommendation. In: Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, pp. 639–648. (2020)
4. Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W.L., Leskovec, J.: Graph convolutional neural networks for web-scale recommender systems. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 974–983. (2018)
5. Liu, M., Li, J., Li, G., Pan, P.: Cross domain recommendation via bi-directional transfer graph collaborative filtering networks. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp. 885–894. (2020)
6. Wang, H., Lian, D., Zhang, Y., Qin, L., He, X., Lin, Y., Lin, X.: Binarized graph neural network. *World Wide Web* **24**(3), 825–848 (2021)
7. Wang, L., Hu, F., Wu, S., Wang, L.: Fully hyperbolic graph convolution network for recommendation. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pp. 3483–3487. (2021)
8. Mao, K., Zhu, J., Xiao, X., Lu, B., Wang, Z., He, X.: Ultragcn: Ultra simplification of graph convolutional networks for recommendation. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pp. 1253–1262. (2021)
9. Wang, X., He, X., Wang, M., Feng, F., Chua, T.S.: Neural graph collaborative filtering. In: Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval, pp. 165–174. (2019)
10. Liu, M., Gao, H., Ji, S.: Towards deeper graph neural networks. In: Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 338–348. (2020)
11. Liu, X., Ding, J., Jin, W., Xu, H., Ma, Y., Liu, Z., Tang, J.: Graph neural networks with adaptive residual. *Adv. Neural Inf. Process. Syst.* **34**, 9720–9733 (2021)
12. Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L.: Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint [arXiv:1205.2618](https://arxiv.org/abs/1205.2618)* (2012)
13. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. *arXiv preprint [arXiv:1609.02907](https://arxiv.org/abs/1609.02907)* (2016)
14. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 1025–1035. (2017)
15. Zhuang, C., Ma, Q.: Dual graph convolutional networks for graph-based semi-supervised classification. In: Proceedings of the 2018 World Wide Web Conference, pp. 499–508. (2018)
16. Qu, L., Zhu, H., Duan, Q., Shi, Y.: Continuous-time link prediction via temporal dependent graph neural network. In: Proceedings of The Web Conference 2020, pp. 3026–3032. (2020)
17. Chen, H., Yin, H., Sun, X., Chen, T., Gabrys, B., Musial, K.: Multi-level graph convolutional networks for cross-platform anchor link prediction. In: Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 1503–1511. (2020)
18. Yadati, N., Nitin, V., Nimishakavi, M., Yadav, P., Louis, A., Talukdar, P.: Nhp: Neural hypergraph link prediction. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp. 1705–1714. (2020)
19. Ranganathan, V., Barbosa, D.: Hoplop: multi-hop link prediction over knowledge graph embeddings. *World Wide Web* **25**(2), 1037–1065 (2022)
20. Guo, S., Lin, Y., Feng, N., Song, C., Wan, H.: Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In: Proceedings of the AAAI conference on artificial intelligence, vol. 33, pp. 922–929. (2019)
21. Lu, B., Gan, X., Jin, H., Fu, L., Zhang, H.: Spatiotemporal adaptive gated graph convolution network for urban traffic flow forecasting. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp. 1025–1034. (2020)
22. Lu, B., Gan, X., Jin, H., Fu, L., Wang, X., Zhang, H.: Make more connections: Urban traffic flow forecasting with spatiotemporal adaptive gated graph convolution network. *ACM Trans. Intell. Syst. Technol. (TIST)* **13**(2), 1–25 (2022)
23. Xu, M., Li, X., Wang, F., Shang, J.S., Chong, T., Cheng, W., Xu, J.: Learning to effectively model spatial-temporal heterogeneity for traffic flow forecasting. *World Wide Web* **26**(3), 849–865 (2023)
24. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Proceedings of the 30th International Conference on Neural Information Processing Systems, pp. 3844–3852. (2016)

25. Yu, W., Qin, Z.: Graph convolutional network for recommendation with low-pass collaborative filters. In: International Conference on Machine Learning, pp. 10936–10945. (2020)
26. Liao, R., Zhao, Z., Urtasun, R., Zemel, R.S.: Lanczosnet: Multi-scale deep graph convolutional networks. In: 7th International Conference on Learning Representations, ICLR 2019. (2019)
27. Estrach, J.B., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and deep locally connected networks on graphs. In: 2nd international conference on learning representations, ICLR, vol. 2014. (2014)
28. Zhu, H., Feng, F., He, X., Wang, X., Li, Y., Zheng, K., Zhang, Y.: Bilinear graph neural network with neighbor interactions. In: Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence, pp. 1452–1458. (2021)
29. Huang, T., Dong, Y., Ding, M., Yang, Z., Feng, W., Wang, X., Tang, J.: Mixgcn: An improved training method for graph neural network-based recommender systems. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pp. 665–674. (2021)
30. Zheng, Y., Gao, C., Chen, L., Jin, D., Li, Y.: Dgcn: Diversified recommendation with graph convolutional networks. In: Proceedings of the Web Conference 2021, pp. 401–412. (2021)
31. Liu, F., Cheng, Z., Zhu, L., Gao, Z., Nie, L.: Interest-aware message-passing gcn for recommendation. In: Proceedings of the Web Conference 2021, pp. 1296–1305. (2021)
32. Huijben, I.A., Kool, W., Paulus, M.B., Van Sloun, R.J.: A review of the gumbel-max trick and its extensions for discrete stochasticity in machine learning. *IEEE Trans. Pattern. Anal. Mach. Intell.* **45**(2), 1353–1371 (2022)
33. Maddison, C.J., Tarlow, D., Minka, T.: A* sampling. In: Proceedings of the 27th International Conference on Neural Information Processing Systems-Volume 2, pp. 3086–3094. (2014)
34. Oberst, M., Sontag, D.: Counterfactual off-policy evaluation with gumbel-max structural causal models. In: International Conference on Machine Learning, pp. 4881–4890. (2019)
35. Lorberbom, G., Johnson, D., Maddison, C.J., Tarlow, D., Hazan, T.: Learning generalized gumbel-max causal mechanisms. *Adv. Neural Inf. Syst.* **34** (2021)
36. Jang, E., Gu, S., Poole, B.: Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016)
37. Potapczynski, A., Loaiza-Ganem, G., Cunningham, J.P.: Invertible gaussian reparameterization: Revisiting the gumbel-softmax. *Adv. Neural Inf. Process Syst.* **33**, 12311–12321 (2020)
38. Hu, S., Xie, S., Zheng, H., Liu, C., Shi, J., Liu, X., Lin, D.: Dsnas: Direct neural architecture search without parameter retraining. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 12084–12092. (2020)
39. Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., Keutzer, K.: Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10734–10742. (2019)
40. Xiao, T., Chen, Z., Wang, D., Wang, S.: Learning how to propagate messages in graph neural networks. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pp. 1894–1903. (2021)
41. He, C., Ye, H., Shen, L., Zhang, T.: Milenas: Efficient neural architecture search via mixed-level reformulation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11993–12002. (2020)
42. Li, Y., Dong, M., Wang, Y., Xu, C.: Neural architecture search in a proxy validation loss landscape. In: International Conference on Machine Learning, pp. 5853–5862. (2020)
43. Li, D., Yang, T., Du, L., He, Z., Jiang, L.: Adaptivegcn: Efficient gcn through adaptively sparsifying graphs. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pp. 3206–3210. (2021)
44. Zheng, C., Zong, B., Cheng, W., Song, D., Ni, J., Yu, W., Chen, H., Wang, W.: Robust graph representation learning via neural sparsification. In: International Conference on Machine Learning, pp. 11458–11468. (2020)
45. Kong, T., Kim, T., Jeon, J., Choi, J., Lee, Y.C., Park, N., Kim, S.W.: Linear, or non-linear, that is the question! In: Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining, pp. 517–525. (2022)
46. Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S.: Self-normalizing neural networks. In: Proceedings of the 31st international conference on neural information processing systems, pp. 972–981. (2017)
47. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.S.: Neural collaborative filtering. In: Proceedings of the 26th international conference on world wide web, pp. 173–182. (2017)
48. He, X., Zhang, H., Kan, M.Y., Chua, T.S.: Fast matrix factorization for online recommendation with implicit feedback. In: Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, pp. 549–558. (2016)

49. Hu, Y., Koren, Y., Volinsky, C.: Collaborative filtering for implicit feedback datasets. In: 2008 Eighth IEEE international conference on data mining, pp. 263–272. IEEE (2008)
50. Wang, X., Jin, H., Zhang, A., He, X., Xu, T., Chua, T.S.: Disentangled graph collaborative filtering. In: Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval, pp. 1001–1010. (2020)
51. Wu, J., Wang, X., Feng, F., He, X., Chen, L., Lian, J., Xie, X.: Self-supervised graph learning for recommendation. In: Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval, pp. 726–735. (2021)
52. Zhu, T., Sun, L., Chen, G.: Embedding disentanglement in graph convolutional networks for recommendation. IEEE Trans. Knowl. Data Eng. (2021)
53. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp. 249–256. (2010)
54. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (Poster). (2015)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

Authors and Affiliations

Zhuo Cai¹ · Guan Yuan¹ · Xiaobao Zhuang¹ · Senzhang Wang² · Shaojie Qiao³ · Mu Zhu⁴

Zhuo Cai
czhuo@cumt.edu.cn

Xiaobao Zhuang
zhuangxb@cumt.edu.cn

Senzhang Wang
szwang@csu.edu.cn

Shaojie Qiao
sjqiao@cuit.edu.cn

Mu Zhu
zhumubest@163.com

¹ School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221000, China

² School of Computer, Central South University, Changsha 410083, China

³ School of Software Engineering, Chengdu University of Information Technology, Chengdu 610225, China

⁴ State Key Laboratory of NBC Protection for Civilian, Beijing 100000, China