Structured Graph Convolutional Networks with Stochastic Masks for Recommender Systems

Huiyuan Chen
hchen@visa.com
Visa Research, Palo Alto, USA

Lan Wang
lwang4@visa.com
Visa Research, Palo Alto, USA

Yusan Lin
yusalin@visa.com
Visa Research, Palo Alto, USA

Chin-Chia Michael Yeh
miyeh@visa.com
Visa Research, Palo Alto, USA

Fei Wang
feiwang@visa.com
Visa Research, Palo Alto, USA

Hao Yang
haoyang@visa.com
Visa Research, Palo Alto, USA

ABSTRACT

Graph Convolutional Networks (GCNs) are powerful for collaborative filtering. The key component of GCNs is to explore neighborhood aggregation mechanisms to extract high-level representations of users and items. However, real-world user-item graphs are often incomplete and noisy. Aggregating misleading neighborhood information may lead to sub-optimal performance if GCNs are not regularized properly. Also, the real-world user-item graphs are often *sparse* and *low rank*. These two intrinsic graph properties are widely used in shallow matrix completion models, but far less studied in graph neural models.

Here we propose Structured Graph Convolutional Networks (SGCNs) to enhance the performance of GCNs by exploiting graph structural properties of sparsity and low rank. To achieve sparsity, we attach each layer of a GCN with a trainable stochastic binary mask to prune noisy and insignificant edges, resulting in a clean and sparsified graph. To preserve its low-rank property, the nuclear norm regularization is applied. We jointly learn the parameters of stochastic binary masks and original GCNs by solving a stochastic binary optimization problem. An unbiased gradient estimator is further proposed to better backpropagate the gradients of binary variables. Experimental results demonstrate that SGCNs achieve better performance compared with the state-of-the-art GCNs.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computer systems organization** → **Neural networks**.

KEYWORDS

Denoising, Graph Convolution Networks, Collaborative Filtering

ACM Reference Format:
Huiyuan Chen, Lan Wang, Yusan Lin, Chin-Chia Michael Yeh, Fei Wang, and Hao Yang. 2021. Structured Graph Convolutional Networks with Stochastic Masks for Recommender Systems. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '21), July 11–15, 2021, Virtual Event, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3404835.3462868

SIGIR '21, July 11–15, 2021, Virtual Event, Canada

ACM ISBN 978-1-4503-8037-9/21/07…$15.00
https://doi.org/10.1145/3404835.3462868

1 INTRODUCTION

Personalized recommender systems have been widely deployed in many online services to meet users' interests [47]. One of the most prominent techniques is *collaborative filtering*, which considers the users' historical interactions and assumes that users who share similar preferences in the past tend to make similar decisions in the future. Factorization Machines have achieved great success by using the inner product of a user embedding and an item embedding as a preference score [23]. Nevertheless, their recommendation performances are unsatisfactory due to the lack of strategies to learn high-order user-item feature interactions [5, 15, 18, 26, 27, 51]. Deep learning techniques thus have started to dominate the landscape of recommender systems [48].

Recently, Graph Convolutional Networks (GCNs) have become increasingly powerful in representation learning of graph-structured data [16, 22, 44]. GCNs use message passing mechanism over the input graph, which can be summarized into three steps: 1) Initialize node representations with their initial attributes or structural features like node degrees; 2) Update the representation of each node by recursively aggregating and transforming over the representations of its neighbors; 3) Readout the final representation of a single node or the entire graph as required by the downstream tasks. By regarding user-item interactions as a bipartite graph, researchers have attempted to adopt GCNs for recommendation due to their theoretical elegance and good performance [11, 17, 30, 42, 47]. For example, PinSage [47] combines efficient random walks and graph convolutions to generate item embeddings. NGCF [42] proposes an embedding propagation layer to investigate the high-order connectivities in the bipartite graphs. LightGCN [17] simplifies the design of GCNs to make it more concise for recommendation.

Although encouraging performance has been achieved, GCNs are still known to be vulnerable to the quality of the input graphs due to its recursive message passing schema [7, 52]. Unfortunately, real-world user-item graphs are often noisy. This is particularly true for implicit behaviors because they are not necessarily aligned with user preferences [37]. If GCNs are not regularized properly, aggregating misleading neighborhood information is likely to lead to sub-optimal performance. We use the following example to further explain the concerns mentioned above.

Figure 1 illustrates how the misleading information is propagated via noisy edges in the graph. Here we consider the embedding of

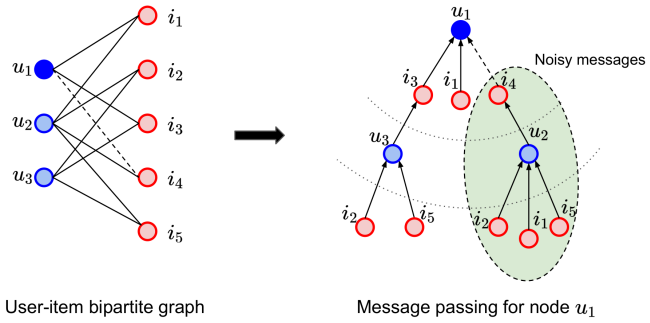

Figure 1: An illustration of how the noisy edge $u_1 - i_4$ will provide misleading information for the target user u_1 under the framework of GCNs.

the target node u_1 in a user-item graph, and there is a noisy edge between user u_1 and item i_4 as shown in the left subfigure. The right subfigure is the corresponding tree structure with node u_1 as the root. The key idea behind GCNs is to fully discover the high-order relations in the bipartite graph. As such, the representation of node i_2 can be aggregated to update the representation of target node u_1 through the path $u_1 \leftarrow i_3 \leftarrow u_3 \leftarrow i_2$, even though there is no explicit connection between u_1 and i_2. However, misleading messages, e.g., the embedding of the first-hop neighbor i_4 or the second-hop neighbor u_2, can be also passed to target node u_1 via the noisy edge $u_1 - i_4$, which likely degrades the performance. As the GCNs go deeper, these misleading messages would continue to propagate and contaminate the entire graph.

To this end, we argue that it is essential to remove the irrelevant neighbors during message passing. Otherwise, including less useful messages will complicate the model training, increase the risk of over-fitting, and even impair model effectiveness. The key challenge is then to decide the criteria to omit irrelevant neighbors during the training stage. Fortunately, real-world graphs are often *sparse* and *low-rank* [12]. Sparsity implies that only the most significant neighbors should be locally connected to the target node during the message passing; low-rank constraint indicates that the entire graph is globally structured and only a few factors contribute to a user's preferences. These two intrinsic graph properties are widely used in linear matrix completion models [4, 23, 32], e.g., l_p norm regularization or matrix rank minimization, but far less studied in graph neural models. A possible approach is to first create a clean k-nearest neighbor graph based on a certain similarity function. This is a common strategy used in shadow graph models such as LLE [36] and Isomap [39], and has been recently revisited in deep graph models [49]. The expressive power of k-nearest neighbor, however, is limited by the choice of k as well as the similarity function in the embedding space.

Present Work. Here we propose a Structured Graph Convolutional Network (SGCN) to enhance the performance of GCNs by exploiting graph structural properties of sparsity and low rank. To achieve sparsity, we attach each layer of a GCN with a stochastic binary mask to prune noisy and insignificant edges under the framework of GCNs. Intuitively, the stochastic binary masks (i.e., 1 is sampled and 0 is dropped) can be regarded as graph generators so as to support a high-quality sparse graph for each layer of GCNs. Our motivation is two-fold: 1) Noisy edges with parameterized masks can be learned to be dropped in a data-driven fashion. A sparse message passing strategy is thus less complicated and has better generalization ability; 2) Over-fitting and over-smoothing are two main bottlenecks of developing deeper GCNs [35]. These issues can be mitigated by sampling sub-graphs with our stochastic mechanism. Nevertheless, directly training the stochastic binary masks is intractable due to the combinatorial nature of discrete samples. To make them differentiable, we further reformulate the optimization problem from a discrete space to a continuous one via probabilistic reparameterization [20, 46]. An unbiased gradient estimator is further proposed to better backpropagate the gradients of binary variables. Inspired by the adversarial machine learning [9, 21], low-rank constraints are also imposed to the sparse adjacency matrices for each layer of GCNs . This regularization forces the graphs to be globally structured, which have been shown to be very successful in defending adversarial attacks [9, 21], not to mention defending the noise in recommendation. We conduct extensive experiments to evaluate the effectiveness and robustness of the proposed SGCN method. Our contributions are as follows:

- We propose SGCN, an approach that explicitly prunes the irrelevant neighbors in the message passing stage of GCNs, which largely reduce the negative impacts of the noise in the recommender systems.
- We develop stochastic binary masks with the goal of selecting the sparse and high-quality sub-graphs for each layer of GCNs. Low-rank constraints are also imposed to enhance the robustness and generalization of the GCNs.
- We propose an unbiased gradient estimator for stochastic binary optimization by casting it to an equivalent one in the continuous space. As such, we can jointly learn the parameters of stochastic binary masks as well as GCNs.
- We conduct extensive experiments on four public datasets. The results demonstrate the benefits of SGCN on the effectiveness of pruning noisy edges and the usage of low-rank constraints, resulting in 4.92% ~ 26.23% performance gains.

2 RELATED WORK

In this section, we briefly review the related work on recommender systems and graph convolutional networks. We also highlight the differences between the existing efforts and our SGCN.

2.1 Collaborative Filtering

Recommender systems often employ Collaborative Filtering (CF) to learn sophisticated feature interactions between users and items based on users' historical profiles. Matrix factorization is an early approach to learn the latent embeddings of users and items from user-item rating matrix and use inner product to predict the users' preference [23]. Motivated by the expressive power of deep neural networks, modern recommender systems are further improved with deep learning techniques to exploit more complex and nonlinear feature interactions between users and items [48]. Some representative models include Wide&Deep [6], NCF [18], DeepFM [15], xDeepFM [26], CDAE [27], etc. Nevertheless, these CF-based models are usually designed to approximate the first-order proximity,

e.g., direct connections between users and items. By revisiting user-item interactions as a bipartite graph, graph-based models are able to explore the implicit high-order proximity between nodes, which is helpful for discovering deeper connections between users and items in the personalized recommender systems [14, 45, 50].

2.2 Graph Convolutional Networks

Graph Convolutional Networks (GCNs), as a special instantiation of convolutional neural networks for structured data, have received a lot of attention for its great performance in graph embedding [16, 22, 44]. Recently, researchers have deployed GCNs in web-scale recommender systems [3, 11, 17, 30, 42, 47, 50]. For example, GC-MC [3] and RMGCNN [30] frame recommender systems as matrix completion and design GCNs on user-item bipartite graphs. SpectralCF [50] develops a spectral convolution to identify all possible connectivities between users and items in the spectral domain. PinSage [47] combines efficient random walks and graph convolutions to generate item embeddings in Pinterest. GraphRec [11] proposes a heterogeneous graph convolutional network for social recommendations. NGCF [42] proposes an embedding propagation layer to harvest the high-order collaborative signals in the bipartite graphs. LightGCN [17] simplifies the design of GCN to make it more concise for recommendation purpose.

Although the aforementioned methods have been proven to be very effective in generating embeddings of users and items, GCNs are known to be sensitive to the quality of the input graphs due to its recursive message passing schema [7, 52]. In other words, slight perturbations on the user-item bipartite graphs can mislead GCNs to output wrong predictions. In this study, we aim to learn a way to obtain the high-quality input graphs as well as the parameters of GCNs simultaneously.

2.3 Over-fitting and Over-smoothing

Two main obstacles encountered when developing deeper GCNs are over-fitting and over-smoothing [24, 25, 29]. Over-fitting comes from the case when an over-parameterized GCN is used to fit a distribution given limited training data. Over-smoothing, on the contrary, leads to features of graph nodes gradually converging to the same value when increasing the number of convolutional layers [25]. Both of the above two issues can be alleviated by using *dropout* tricks in the GCNs. For example, vanilla Dropout [38] randomly masks out the elements in the weight matrix to reduce the effect of over-fitting. However, Dropout does not prevent over-smoothing since it does not make any change of the graph adjacency matrix. DropNode [16] is a node-oriented method that randomly selects the nodes for the mini-batch training. DropEdge [35] is an edge-oriented method that randomly removes a certain number of edges from the input graphs, acting like a data augmenter. Message dropout [42] randomly drops the outgoing messages in each propagation layer to refine representations. DropoutNet [40] applies input dropout during training to address cold start issue in recommender systems. Nevertheless, these dropout techniques typically remove a certain fraction of nodes, edges, or features by *random*, which may lead to sub-optimal performance.

The mechanism of our stochastic binary masks is slightly different from the abovementioned dropout methods, but is more relevant to the recent developments on graph sparsification [13, 49]. We introduce an optimization algorithm, an alternative to random sampling, to determine which edge to be deleted in a data-driven way. As a result, the sparse graphs that best preserve desired properties, e.g., sparse and low-rank, can benefit GCNs in terms of better robustness and superior generalization.

3 THE PROPOSED MODEL

In this section, we present the proposed SGCN model in details. Our SGCN mainly consists of three components: a well-designed GCN, stochastic binary masks, and rank approximation. Finally, we introduce the loss function for model optimization.

3.1 Problem Formulation

In this paper, we focus on learning the user preferences from the implicit feedback. To be specific, the behavior data, e.g., click, comment, purchase, etc., involves a set of users $\mathcal{U} = \{u\}$ and items $\mathcal{I} = \{i\}$, such that the set $\mathcal{I}_u^+$ represents the items that user u has interacted with before, while $\mathcal{I}_u^- = \mathcal{I} - \mathcal{I}_u^+$ represents unobserved items. Unobserved interactions are not necessarily negative, rather that the user may simply be unaware of them.

When viewing user-item interactions as a bipartite graph $\mathcal{G}$, we can construct an implicit feedback matrix $\mathbf{R} \in \mathbb{R}^{|\mathcal{U}|\times|\mathcal{I}|}$, where $|\mathcal{U}|$ and $|\mathcal{I}|$ denote the number of users and items, respectively. Each entry $\mathbf{R}_{u,i} = 1$ if user u has interacted with item i , and $\mathbf{R}_{u,i} = 0$ otherwise. Its corresponding adjacency matrix $\mathbf{A}$ for the bipartite graph can be obtained as:

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{R} \\ \mathbf{R}^\top & \mathbf{0} \end{bmatrix}, \tag{1}$$

where the adjacency matrix $\mathbf{A}$ can be used as the input graph for the GCNs later. We aim to recommend a ranked list of items from $\mathcal{I}_u^-$ that are of interests to the user $u \in \mathcal{U}$, in the same sense that performing link prediction on the bipartite graph $\mathcal{G}$.

3.2 GCN for Recommendation

3.2.1 ***Embedding Layer.*** Following the mainstream graph convolutional recommender systems [17, 18, 42], we describe the representations of a user u and an item i via embedding lookup tables:

$$\mathbf{e}_u = \text{lookup}(u), \qquad \mathbf{e}_i = \text{lookup}(i), \tag{2}$$

where u and i denote the IDs of user and item; $\mathbf{e}_u \in \mathbb{R}^d$ and $\mathbf{e}_i \in \mathbb{R}^d$ are the embeddings of user u and item i, respectively, and d is the embedding size. These embeddings are expected to memorize the initial characteristics of items and users. We next introduce two state-of-the-art GCN-based recommender models.

3.2.2 ***NGCF.*** Following the standard GCN [22], NGCF [42] leverages the user-item bipartite graph to perform embedding propagation and feature transformation as:

$$\begin{aligned} \mathbf{e}_u^{(k+1)} &= \sigma\left(\mathbf{W}_1\mathbf{e}_u^{(k)} + \sum_{i\in\mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}\left(\mathbf{W}_1\mathbf{e}_i^{(k)} + \mathbf{W}_2(\mathbf{e}_i^{(k)} \odot \mathbf{e}_u^{(k)})\right)\right), \\ \mathbf{e}_i^{(k+1)} &= \sigma\left(\mathbf{W}_1\mathbf{e}_i^{(k)} + \sum_{u\in\mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}}\left(\mathbf{W}_1\mathbf{e}_u^{(k)} + \mathbf{W}_2(\mathbf{e}_u^{(k)} \odot \mathbf{e}_i^{(k)})\right)\right), \end{aligned} \tag{3}$$

where $\mathbf{e}_u^{(k)}$ and $\mathbf{e}_i^{(k)}$, with initialization $\mathbf{e}_u^{(0)} = \mathbf{e}_u$ and $\mathbf{e}_i^{(0)} = \mathbf{e}_i$ as in Eq. (2), denote the refined representations of user u and item i in

the k-th layer of GCN, respectively; $\sigma(\cdot)$ is the nonlinear activation function and $\odot$ denotes the element-wise product; $\mathbf{W_1}$ and $\mathbf{W_2}$ are trainable weight matrices; $\mathcal{N}_u$ denotes the set of items that are directly interacted by user u, and $\mathcal{N}_i$ denotes the set of users that are connected to item i. As stacking more convolutional layers, the model is able to explore high-order collaborative signals between users and items.

3.2.3 ***LightGCN.*** Several studies have pointed out that simpler, sometimes linear GCNs are very effective for representation learnings [44]. Recently, LightGCN [17] aims to simplify the design of NGCF to make it more concise for recommendation.

In contrast to NGCF, LightGCN adopts weighted sum aggregators and abandon the use of feature transformation and nonlinear activation. As such, the propagation in Eq. (3) can be simplified as:

$$\begin{aligned} \mathbf{e}_u^{(k+1)} &= \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|\,|\mathcal{N}_i|}} \mathbf{e}_i^{(k)}, \\ \mathbf{e}_i^{(k+1)} &= \sum_{u \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i|\,|\mathcal{N}_u|}} \mathbf{e}^{(k)}. \end{aligned} \tag{4}$$

The above equation can be re-written in a compact matrix form. Let the 0-th layer embedding matrix be $\mathbf{E}^{(0)} \in \mathbb{R}^{(|\mathcal{U}|+|\mathcal{I}|)\times d}$, which collects all of the embeddings of users and items from Eq. (2). Then, we can obtain the equivalent matrix form of Eq. (4) as:

$$\mathbf{E}^{(k+1)} = \left(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right)\mathbf{E}^{(k)}, \tag{5}$$

where $\mathbf{A} \in \mathbb{R}^{(|\mathcal{U}|+|\mathcal{I}|)\times(|\mathcal{U}|+|\mathcal{I}|)}$ is the adjacency matrix of the use-item graph as shown in Eq. (1); $\mathbf{D}$ is the corresponding diagonal degree matrix, in which each entry $\mathbf{D}_{i,i}$ denotes the number of non-zeros in the i-th row of the matrix $\mathbf{A}$.

3.2.4 ***Model Optimization for NGCF and LightGCN.*** By propagating K layer, GCNs obtain $K+1$ embeddings to represent a user $(\mathbf{e}_u^{(0)}, \ldots, \mathbf{e}_u^{(K)})$ and an item $(\mathbf{e}_i^{(0)}, \ldots, \mathbf{e}_i^{(K)})$. An aggregation function is used to obtain the final representations:

$$\mathbf{e}_u^* = \mathrm{AGG}(\mathbf{e}_u^{(0)}, \ldots, \mathbf{e}_u^{(K)}), \quad \mathbf{e}_i^* = \mathrm{AGG}(\mathbf{e}_i^{(0)}, \ldots, \mathbf{e}_i^{(K)}),$$

NGCF implements $\mathrm{AGG}(\cdot)$ by concatenation while LightGCN uses weighted sum. The inner product is used to predict preference score:

$$\hat{y}_{ui} = \mathbf{e}_u^{*T}\mathbf{e}_i^*.$$

Both methods employ the Bayesian Personalized Ranking (BPR) loss [34] to optimize the model parameters, that is minimizing:

$$\mathcal{L}_{BPR}(\mathbf{\Theta}) = \sum_{(u,i,j)\in O} -\ln \sigma\left(\hat{y}_{ui} - \hat{y}_{uj}\right) + \alpha\|\mathbf{\Theta}\|_2^2, \tag{6}$$

where $O = \left\{(u,i,j) \mid u \in \mathcal{U} \wedge i \in I_u^+ \wedge j \in I_u^-\right\}$ denotes the pairwise training data; $\sigma(\cdot)$ is the sigmoid function; $\mathbf{\Theta}$ denotes model parameters, and α controls the L_2 norm to prevent over-fitting.

3.2.5 ***Limitations.*** Despite of the success of NGCF and LightGCN, we argue that they are insufficient to address the noise in the bipartite graphs. For example, LightGCN fully relies on the adjacency matrix $\mathbf{A}$ to refine the representations of users and items in Eq. (5). The matrix $\mathbf{A}$, however, may contain noisy edges as discussed in Section 1, those misleading messages continue to propagate as LightGCN goes deeper. The situations become much worse when the graph noisy signals contain low-frequency components. As such, GCNs have a high risk of over-fitting to the noise [33].

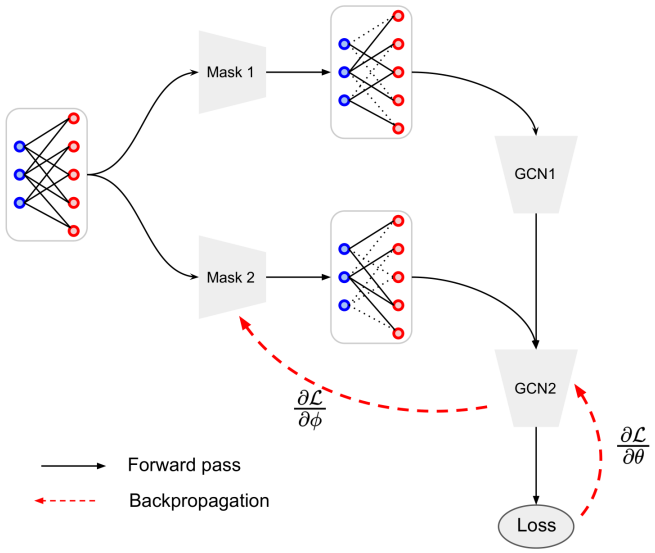

Figure 2: The overview of a two-layer GCN with stochastic binary masks. Here only partial gradients are shown in the backpropagation.

On the other hand, vanilla Dropout [38] randomly masks out the elements of weight matrix (e.g., $\mathbf{W}_1$ and $\mathbf{W}_2$ in Eq. (3)), which may have limited ability of preventing noise since it does not make any change to the adjacency matrix $\mathbf{A}$. NGCF somehow alleviates this issue by removing a fraction of messages or nodes *randomly*. Nevertheless, this weakens the interpretability and understanding of which edge should be kept or deleted in the training stage (see Section 4.3 for details). To address this challenge, we propose a simple yet effective data-driven principle, an alternative to random sampling, to mask out edges by using stochastic binary masks.

3.3 Stochastic Binary Masks

3.3.1 ***Graph Sparsification.*** To filter out the noise, we attach each layer of the GCNs with a stochastic binary mask to prune insignificant edges while simultaneously training the parameters of GCNs. The overall network architecture is shown in Figure 2. Formally, for each convolutional layer in Eq. (5), we introduce a binary matrix $\mathbf{Z}^{(k)} \in \{0,1\}$, where $\mathbf{Z}_{u,v}^{(k)}$ denotes whether the edge between node u and node v is included in the k-th layer. Instead of a *fixed* adjacency matrix in Eq. (5), the input graph adjacency matrix for the k-th layer becomes:

$$\mathbf{A}^{(k)} = \mathbf{A} \odot \mathbf{Z}^{(k)}, \tag{7}$$

where $\odot$ denotes the element-wise product. Intuitively, the stochastic binary masks $\mathbf{Z}^{(k)}$ (i.e., 1 is sampled and 0 is dropped) can be regarded as *graph generators* so as to support a high-quality sparse graph for each layer of GCNs. The sparse graphs enable a subset of neighbor aggregation instead of full aggregation during training, thus avoiding over-smoothing when GCNs go deeper. This idea of graph sparsification is not new. In fact, its original goal is

removing unnecessary edges for graph compressing while keeping primary information of the input graph [10], which has been recently revisited in deep graph models [13, 49].

To encourage sparsity of $\mathbf{A}^{(k)}$, an L_0 regularizer is used to explicitly penalize the number of non-zero entries of $\mathbf{Z}^{(k)}$ by minimizing:

$$\mathcal{R}_s = \sum_{k=1}^{K} \left\| \mathbf{Z}^{(k)} \right\|_0 = \sum_{k=1}^{K} \sum_{(u,v)\in\mathcal{G}} \mathbb{I}\left[\mathbf{Z}_{u,v}^{(k)} \neq 0 \right], \tag{8}$$

where $\|\cdot\|_0$ denotes the L_0 norm that can drive insignificant edges to be *exact* zero. $\mathbb{I}[c]$ is an indicator function that is equal to 1 if the condition c holds, 0 otherwise. Optimization under this penalty, however, is computationally intractable due to the non-differentiability, discrete, and combinatorial nature of $2^{|\mathcal{G}|}$ possible states of the binary mask $\mathbf{Z}^{(k)}$. To address this challenge, we reparameterize these discrete variables as deterministic transformations of the underlying continuous variables and then apply antithetic sampling to produce low-variance and unbiased gradients. We next introduce an efficient algorithm to better backpropagate the gradients through stochastic binary layers.

3.3.2 ***Reparameterization and Gradients.*** Since $\mathbf{Z}^{(k)}$ is jointly optimized with the original GCNs (e.g., NGCF or LightGCN), we combine Eq. (6) and Eq. (8) as one unified objective:

$$\mathcal{L}(\mathbf{Z}, \mathbf{\Theta}) = \mathcal{L}_{BPR}(\{\mathbf{A} \odot \mathbf{Z}^{(k)}\}_{k=1}^{K}, \mathbf{\Theta}) + \beta \sum_{k=1}^{K} \sum_{(u,v)\in\mathcal{G}} \mathbb{I}\left[\mathbf{Z}_{u,v}^{(k)} \neq 0 \right], \tag{9}$$

where β controls the sparsity of graphs. As such, Eq. (9) involves stochastic gradient estimations, which require marginalization of $2^{|\mathcal{G}|}$ binary sequences. For this reason, we consider each $\mathbf{Z}_{u,v}^{(k)}$ is subject to a Bernoulli distribution with parameter $\mathbf{\Pi}_{u,v}^{(k)} \in [0, 1]$ such that $\mathbf{Z}_{u,v}^{(k)} \sim \text{Bern}(\mathbf{\Pi}_{u,v}^{(k)})$. The Eq. (9) can be reformulated as

$$\hat{\mathcal{L}}(\mathbf{Z}, \mathbf{\Theta}) = \mathbb{E}_{\mathbf{Z}\sim\prod_{k=1}^{K} \text{Bern}(\mathbf{Z}^{(k)};\mathbf{\Pi}^{(k)})} \left[\mathcal{L}_{BPR}(\mathbf{Z}, \mathbf{\Theta}) \right] + \beta \sum_{k=1}^{K} \sum_{(u,v)\in\mathcal{G}} \mathbf{\Pi}_{u,v}^{(k)}, \tag{10}$$

where $\mathbb{E}$ is the expectation, and objective $\hat{\mathcal{L}}$ in Eq. (10) is in fact a variational upper bound[1] for objective $\mathcal{L}$ in Eq. (9) over the parameters $\mathbf{\Pi}^{(k)}$. Now the second term is differentiable w.r.t. the new parameters $\mathbf{\Pi}^{(k)}$, while the first term is still problematic due to the discrete nature of $\mathbf{Z}^{(k)}$.

To efficiently compute gradients, we use the reparameterization trick [20], which reparameterizes $\mathbf{\Pi}_{u,v}^{(k)} \in [0, 1]$ to a deterministic function $g(\cdot)$ of the parameters $\mathbf{\Phi}_{u,v}^{(k)}$, so that

$$\mathbf{\Pi}_{u,v}^{(k)} = g(\mathbf{\Phi}_{u,v}^{(k)}),$$

since the deterministic function $g(\cdot)$ should be bounded within $[0, 1]$, the standard sigmoid function is thus a good candidate: $g(x) = 1/(1 + e^{-x})$. In addition, we adopt augment-REINFORCE-merge (ARM), a recently proposed unbiased gradient estimator, to solve the stochastic binary optimization [46]. We first introduce the key theorem as following:

Theorem 3.1 (ARM). *For a vector of N binary random variables $\boldsymbol{z} = (z_1, \ldots, z_N)^T$, and any function f, the gradient of*

$$\mathcal{E}(\boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{z}\sim\prod_{i=1}^{N} \text{Bern}(z_i;\sigma(\phi_i))} [f(\boldsymbol{z})]$$

with respect to $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_N)^T$, the logits of the Bernoulli probability parameters, can be expressed as:

$$\nabla_{\boldsymbol{\phi}} \mathcal{E}(\boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{u}\sim\prod_{i=1}^{N} \text{Uniform}(u_i;0,1)} \left[\left(f(\mathbb{I}[\boldsymbol{u} > \sigma(-\boldsymbol{\phi})]) - f(\mathbb{I}[\boldsymbol{u} < \sigma(\boldsymbol{\phi})]) \right) \left(\boldsymbol{u} - \frac{1}{2} \right) \right],$$

where $\mathbb{I}[\boldsymbol{u} > \sigma(-\phi)] := (\mathbb{I}[u_1 > \sigma(-\phi_1)], \ldots, \mathbb{I}[u_N > \sigma(-\phi_N)])^T$, and $\sigma(\cdot)$ is the sigmoid function.

Due to the linearity of expectations, ARM is able to directly optimize the Bernoulli variables without introducing any bias, which yields a highly competitive estimator. Moreover, the expectation can be estimated using only an antithetically coupled pair of samples, allowing to compute the gradient efficiently.

According to above Theorem, let $f(\cdot)$ be the BPR loss function: $f(\mathbf{Z}) = \mathcal{L}_{BPR}(\mathbf{Z}, \mathbf{\Theta})$, and the reparameterization[2]: $\mathbf{\Pi} = \sigma(\mathbf{\Phi})$, we can now compute the gradient of $\hat{\mathcal{L}}$ in Eq. (10) w.r.t. $\mathbf{\Phi}$ in the following matrix form:

$$\begin{aligned} \nabla_{\mathbf{\Phi}} \hat{\mathcal{L}}(\mathbf{\Phi}, \mathbf{\Theta}) = & \mathbb{E}_{\mathbf{U}\sim\prod_{k=1}^{K} \text{Uniform}(\mathbf{U}^{(k)};0,1)} [(f(\mathbb{I}[\mathbf{U} > \sigma(-\mathbf{\Phi})]) \\ & - f(\mathbb{I}[\mathbf{U} < \sigma(\mathbf{\Phi})]))(\mathbf{U} - \frac{1}{2})] + \beta \nabla_{\mathbf{\Phi}} \sigma(\mathbf{\Phi}), \end{aligned} \tag{11}$$

where $f(\mathbb{I}[\mathbf{U} > \sigma(-\mathbf{\Phi})])$ is the BPR loss obtained by setting the binary masks $\mathbf{Z}^{(k)}$ to 1 if $\mathbf{U}^{(k)} > \sigma(-\mathbf{\Phi}^{(k)})$ in the foward pass of GCNs, 0 otherwise. The same strategy is applied to $f(\mathbb{I}[\mathbf{U} < \sigma(\mathbf{\Phi})])$.

To this end, we can efficiently backpropagate the gradients through stochastic binary masks since: 1) Sampling from a Bernoulli distribution is simply replaced by sampling from a Uniform distribution between 0 and 1; 2) The first term of Eq. (11) only involves forward pass of GCNs to compute the gradients; 3) The second term $\nabla_{\mathbf{\Phi}} \sigma(\mathbf{\Phi})$ is differentiable and easy to compute. These are very appealing properties, meaning that we can compute the gradients from a discrete space to a continuous one.

In the inference stage, we can use the expectation of $\mathbf{Z}_{u,v}^{(k)} \sim \text{Bern}(\mathbf{\Pi}_{u,v}^{(k)})$ as the mask in Eq. (7), i.e., $\mathbb{E}(\mathbf{Z}_{u,v}^{(k)}) = \mathbf{\Pi}_{u,v}^{(k)} = g(\mathbf{\Phi}_{u,v}^{(k)})$. Nevertheless, this will not yield a sparse graph $\mathbf{A}^{(k)}$ since the sigmoid function in Theorem 3.1 is smooth and none of the element of masks is exact zero (unless the hard sigmoid function is used). Here we simply clip those values $g(\mathbf{\Phi}_{u,v}^{(k)}) \leq 0.5$ to zeros such that a sparse graph is guaranteed and the corresponding noisy edges are effectively eliminated.

Discussion: It is worth mentioning that several recent studies have been proposed to estimate the gradients for discrete variables in Eq. (10), such as REINFORCE [43], Gumbel-Softmax [20], Straight Through Estimator [2], and Hard Concrete Estimator [28]. These approaches, however, suffer from either biased gradients or high variance, while the ARM estimator is unbiased, exhibits low variance, and has low computational complexity as shown in [46].

The ARM estimator for Eq. (11) is remarkably simple but requires two-forward pass of the GCNs to compute the BPR loss. In the original paper [46], the authors also introduce its variant, namely Augment-Reinforce (AR), to overcome the issue of double forward pass, but it leads to a higher variance. Fortunately, unlike Convolutional Neural Networks (CNNs), the number of layers in

[1] This can be derived by the *Jensen's Inequality.*

[2] Here we denote $\mathbf{Z} := \{\mathbf{Z}^{(1)}, \ldots, \mathbf{Z}^{(K)}\}$, $\mathbf{\Pi} := \{\mathbf{\Pi}^{(1)}, \ldots, \mathbf{\Pi}^{(K)}\}$, and $\mathbf{\Phi} := \{\mathbf{\Phi}^{(1)}, \ldots, \mathbf{\Phi}^{(K)}\}$ for K layers of GCNs.

GCNs is often very small (e.g., $K \leq 4$ in NGCF and LightGCN), the complexity of double forward pass is acceptable. We thus stick to the standard ARM in the experiments. We are aware that other advanced techniques can be incorporated to further improve the training of our stochastic binary masks, such as DisARM [8]. We leave this extension in the future.

3.4 Low-rank Approximation

In addition to achieving sparse graphs via binary masks, the GCNs themselves suffer from vulnerabilities against small perturbations [7]. Changes to one node can affect other nodes that are in the same local community. Recently, several studies show that graphs with low-rank constraints are more robust to perturbations [9, 21]. In this work, we further impose the low-rank constraints on the adjacency matrix $\mathbf{A}^{(k)}$, $0 \leq k \leq K$, by minimizing:

$$\mathcal{R}_l = \sum_{k=1}^{K} \left\| \mathbf{A}^{(k)} \right\|_* = \sum_{k=1}^{K} \sum_{i} \lambda_i(\mathbf{A}^{(k)}), \tag{12}$$

where $\|\cdot\|_*$ denotes the nuclear norm that is the convex surrogate for rank minimization. $\lambda_i(\mathbf{A}^{(k)})$ denotes the i-th largest singular values of $\mathbf{A}^{(k)}$. Singular value decomposition (SVD) is often required to optimize the nuclear norm [9].

SVD can be easily implemented but is often numerically unstable during backpropagation [19, 41]. This is because the partial derivatives of the nuclear norm depend on a matrix $\mathbf{K}$ with elements [19]:

$$\mathbf{K}_{i,j} = \begin{cases} \frac{1}{\lambda_i^2 - \lambda_j^2}, & i \neq j \\ 0, & i = j. \end{cases}$$

when two singular values are close, the partial derivatives become very large, causing arithmetic overflow. This is particularly true for large matrices, in which the possibility of two singular values being almost equal is much higher than for small ones. The Power Iteration (PI) method is one way to solve this problem. PI relies on an iterative procedure to approximate the dominant eigenvalues and eigenvectors. Nonetheless, PI is sensitive to how the singular vectors are initialized at the start of each deflation step [41].

To address these problems, we turn our attention to the recent algorithm that friendly combines SVD and PI [41]. For nuclear norm, the top-n singular values are much more informative, we thus adopt the truncated SVD to approximate the Eq. (12) as $\mathcal{R}_l \approx \sum_{k=1}^{K} \sum_{i}^{n} \lambda_i(\mathbf{A}^{(k)})$. As suggested by [41], the hybrid strategy is follows: 1) In the forward pass, we use the truncated SVD to compute $[\mathbf{V}^{(k)}, \mathbf{\Lambda}^{(k)}, \mathbf{V}^{(k)}] = \text{SVD}(\mathbf{A}^{(k)})$ for each adjacency matrix and compute the nuclear norm based on $\mathbf{\Lambda}^{(k)}$; 2) In the backpropagation, we compute the gradients from the PI derivations, but using the SVD-computed vectors $\mathbf{V}^{(k)}$ for initialization purposes. The overall computational graph is demonstrated in Figure 3.

In conclusion, SVD is not involved the backpropagation (routine 2), SVD is only involved the forward pass to compute the nuclear norm loss $\mathcal{R}_l$ for Eq. (12) (routine 1) and initializes the states of PI. In contrast, the PI is not involved in the forward pass (routine 1), it is only used to compute the gradients during backpropagation (routine 2).The result computational graph is both numerically stable and differentiable for imposing low-rank constraints in GCNs.

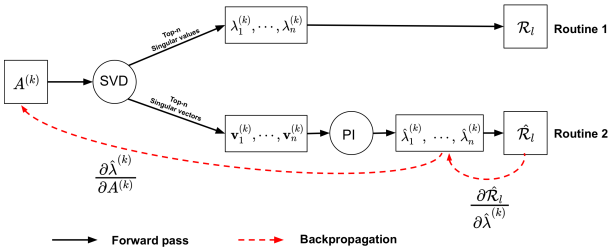

Figure 3: Computational graph for the nuclear norm loss $\mathcal{R}_l$ in Eq. (12). The shadow loss $\hat{\mathcal{R}}_l$ is only used in backpropagation.

3.5 Joint Training

3.5.1 Hybrid Loss. To this end, we jointly learn the graph structure and the GCN model for recommendation tasks. Combining the loss in Eq. (6), Eq. (8), and Eq. (12), the overall objective function of SGCN is given as:

$$\mathcal{L}_{SGCN} = \mathcal{L}_{BPR} + \beta \cdot \mathcal{R}_s + \gamma \cdot \mathcal{R}_l \tag{13}$$

where β and γ are the hyper-parameters to control the degree of sparsity and low-rank constraints, respectively. The overall training of SGCN is summarized in Algorithm 1.

Algorithm 1: SGCN

Input: The training graph $\mathbf{A}$, the number of GCN layers K, and the regularization coefficients α, β, and γ.

```
for each mini-batch do
    for k ← 1 to K do
        Generate a subgraph A^(k) via the stochastic binary mask in Eq. (7);
        Feed A^(k) into the k-th layer of GCN;
    end
    Compute the loss L_SGCN in Eq. (13);
    Update the parameters of GCN and stochastic binary masks;
end
```

Output: A well-trained SGCN to predict $\hat{y}_{ui}$.

3.5.2 Model Complexity. The complexity of SGCN comes from three components: a basic GCN (either NGCF or LightGCN), stochastic binary masks, and low-rank constraints. The basic GCN has the same complexity as NGCF or LightGCN, which denotes as $O(T)$. The complexity of the stochastic binary masks is mainly from ARM in Eq. (11), which requires two-forward pass of the GCN. As discussed before, the number of layers in GCNs is often very small. As such, the complexity of ARM is roughly $O(2T)$, which is much less expensive than the standard gradient backpropagation [46]. In addition, the major complexity of low-rank constraints is the SVD computation. Recently, a few breakthroughs have been proposed for k-SVD, such as Block Krylov method [31] or LazySVD [1]. As the matrix $\mathbf{A}^{(k)}$ is naturally sparse, it only requires $O(\text{nnz}(\mathbf{A}^{(k)}))$ to compute the top-n singular values and their corresponding singular vectors. Although SGCN incorporates the sparse and low-rank information, the computational complexity remains the same order as state-of-the-art GCNs for recommendations.

Table 1: Dataset statistics.

Dataset	#User	#Items	#Interactions	Sparsity
MovieLens	6,040	3,900	1,000,209	4.190%
Gowalla	29,858	40,981	1, 027, 370	0.084%
Yelp	31, 668	38, 048	1, 561, 406	0.130%
Amazon	52, 643	91, 599	2, 984, 108	0.062%

4 EXPERIMENTS

Here we conduct experiments to answer the following questions:

- **RQ1:** How effective is the proposed SGCN compared to state-of-the-art baselines?
- **RQ2:** How can SGCN alleviate the problem of noisy edges?
- **RQ3:** How do different components (e.g., stochastic binary masks and low-rank constraints) affect the performance of SGCN?

4.1 Experimental Settings

4.1.1 Datasets. We use four public benchmark datasets for evaluating recommendation performance:

- **Movielens-1M**[3] is a widely used benchmark for recommendations. The dataset contains one million user-movie ratings.
- **Gowalla**[4] is a check-in dataset obtained from the location-based social website *Gowalla*.
- **Yelp**[5] is released by the *Yelp* challenge. The *Yelp2018* version is used in the experiments.
- **Amazon**[6] contains a large corpus of user reviews, ratings, and product metadata, collected from *Amazon.com*. We use the largest category *Books*.

For MovieLens, we treat all ratings as implicit feedback, e.g., each rating score is transformed to either 1 or 0 indicating whether a user rates a movie. For sparse datasets: Gowalla, Yelp, and Amazon, we use the 10-core setting of the graphs to ensure that all users and items have at least 10 interactions [17, 42]. We summarize the statistics of the datasets in Table 1.

For each dataset, we randomly select 80% of historical interactions of each user to construct the training set, and treat the remaining as the test set. From the training set, we randomly select 10% of interactions as validation set to tune hyper-parameters. For each observed user-item interaction, we treat it as a positive instance, and then conduct the ranking triplets by sampling from negative items that the user did not consume before. We repeat five random splits independently and report the averaged results.

4.1.2 Baselines. We compare with the following baselines:

- **BPR-MF** [34]: A classic model that seeks to optimize the Bayesian personalized ranking loss. We employ matrix factorization as its preference predictor.
- **NeuMF** [18]: NeuMF learns nonlinear interactions between user and item embeddings via a multi-layer perceptron as well as a generalized matrix factorization.
- **GC-MC** [3]: GC-MC employs a graph auto-encoder approach to learn the embeddings of users and items. A bilinear decoder is then used to predict the preference scores.
- **HOP-Rec** [45]: HOP-Rec discovers high-order indirect information of neighborhood items for each user from the bipartite graph by conducting random surfing on the graph.
- **BiNE** [14]: BiNE learns both explicit and implicit user-item relationships by performing biased random walks on the bipartite graph.
- **NGCF** [42] and **LightGCN** [17]: Two state-of-the-art GCN-based collaborative filtering models. They are briefly introduced in the Section 3.2.
- **S-NGCF**: Our SGCN model is a general framework that is compatible with diverse GCN models. With NGCF as the basic backbone, the structured NGCF (S-NGCF) aims to improve its performance and robustness.
- **S-LightGCN**: Similarly, we choose the LightGCN as the backbone under our SGCN framework.

4.1.3 Implementation Details. We implement our SGCNs in the TensorFlow. For all models, the embedding dimension d of users and items (e.g., in Eq. (2)) is searched among $\{16, 32, 64, 128\}$. For baselines BPR-MF, NeuMF, GC-MC, HOP-Rec, and BiNE, their hyper-parameters are initialized as in their original papers and are then carefully tuned to achieve the optimal performance. For the GCN components inside the proposed SGCNs, we use the same hyper-parameters as the original NGCF and LightGCN, such as batch size, stopping criteria, learning rate in Adam optimizer, etc. In addition, the SGCNs have two hyper-parameters β and γ to control the degree of sparsity and low-rank structure, respectively. We tune both β and γ within $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$ to investigate the parameter sensitivity of our models.

To evaluate the performance of top-n recommendation, we adopt two widely used evaluation metrics [17, 42]: *Recall* and *Normalized Discounted Cumulative Gain (NDCG)* over varying numbers of top ranking items.

4.2 Performance Comparison (RQ1)

In this section, we compare the proposed SGCNs with baselines in terms of *Recall@n* and *NDCG@n* on all four datasets, where n is set to 50 and 100. The performance for different top-n values is similar in the experiments, we omit them for space limitation. The results for top-n recommendation are summarized in Table 2. Our proposed models consistently yield the best performance across all cases. From Table 2, we have the following observations:

- Compared with CF-based methods (e.g., BPR-MF, NeuMF, and GC-MC), graph-based methods consistently achieve better performance in most cases. This demonstrates the effectiveness of exploiting high-order proximity between users and items in the bipartite graph. As a result, a user is capable of receiving broader messages from items that beyond the user's line of sight.
- Among graph-based methods, GCN-based methods (e.g., NGCF, LightGCN, and SGCNs) perform better than HOP-Rec and BiNE for all the datasets. This is because GCN-based methods allow end-to-end gradient-based training, and they can directly accept the original graph as input without the need of any preprocessing. In contrast, both HOP-Rec and BiNE first require random walks

[3]https://grouplens.org/datasets/movielens/20m/
[4]https://github.com/kuandeng/LightGCN/tree/master/Data
[5]https://www.yelp.com/dataset
[6]https://jmcauley.ucsd.edu/data/amazon/

Table 2: Recommendation performance comparison for different models. Note that R and N are short for Recall and NDCG, respectively. %Improv denotes the relative improvement of SGCNs over their corresponding GCNs. The best results are highlighted in bold and the second best ones are underlined.

	MovieLens				Gowalla				Yelp				Amazon			
Metric	R@50	N@50	R@100	N@100	R@50	N@50	R@100	N@100	R@50	N@50	R@100	N@100	R@50	N@50	R@100	N@100
BPR-MF	0.282	0.243	0.371	0.354	0.129	0.118	0.346	0.156	0.093	0.038	0.140	0.047	0.069	0.041	0.122	0.059
NeuMF	0.297	0.251	0.378	0.368	0.143	0.124	0.350	0.169	0.103	0.040	0.151	0.050	0.074	0.047	0.135	0.061
GC-MC	0.291	0.247	0.375	0.360	0.137	0.122	0.347	0.163	0.098	0.036	0.146	0.044	0.070	0.044	0.128	0.064
HOP-Rec	0.314	0.260	0.373	0.367	0.135	0.125	0.352	0.182	0.111	0.048	0.163	0.053	0.080	0.059	0.143	0.074
BiNE	0.312	0.253	0.381	0.371	0.141	0.126	0.354	0.188	0.110	0.042	0.155	0.049	0.076	0.052	0.134	0.069
NGCF	0.325	0.289	0.393	0.382	0.160	0.132	0.356	0.197	0.114	0.054	0.172	0.061	0.092	0.065	0.157	0.076
S-NGCF	0.341	0.311	0.417	**0.408**	0.177	0.156	0.384	0.218	0.127	0.068	0.194	0.077	0.107	0.074	0.170	0.087
%Improv.	4.92%	7.61%	6.11%	6.81%	10.63%	18.18%	7.87%	10.66%	11.40%	25.93%	12.79%	26.23%	16.30%	13.85%	8.28%	14.47%
LightGCN	0.328	0.294	0.399	0.384	0.163	0.134	0.360	0.205	0.117	0.059	0.181	0.067	0.098	0.071	0.162	0.083
S-LightGCN	**0.347**	**0.313**	**0.424**	0.406	**0.178**	**0.159**	**0.387**	**0.223**	**0.134**	**0.073**	**0.199**	**0.081**	**0.114**	**0.078**	**0.177**	**0.092**
%Improv.	5.79%	6.46%	6.27%	5.73%	9.20%	18.66%	7.50%	8.78%	14.53%	23.73%	9.94%	20.90%	16.33%	9.86%	9.56%	10.84%

to generate K-step node sequences and then optimize the node embeddings with the downstream tasks. However, the random walk algorithms can not be trained end-to-end, which may lead to sub-optimal performance.

- By comparing the S-NGCF and NGCF, S-NGCF has on average 9.79% improvement with respect to *Recall* and over 15.47% improvements with respect to *NDCG*. Analogously, S-LightGCN outperforms the best baseline LightGCN by average 9.85% in *Recall* and 13.12% in *NDCG*. From the results, we can see that the SGCNs perform much better than their vanilla GCNs. In real-world applications, users are possible to implicitly interact with millions of items, the implicit feedback may be not perfectly matched with user preferences. The original GCNs are thus incapable of dealing with the noisy interactions (e.g., false positive interactions). On the contrary, SGCNs jointly learn a sparse and low-rank graph structure under the architecture of the GCNs, which have the ability of denoising the users' implicit feedback.

It is common to assume the observations contain some noise, we next explore their resilience to noisy edges and provide some insights on the design of SGCNs.

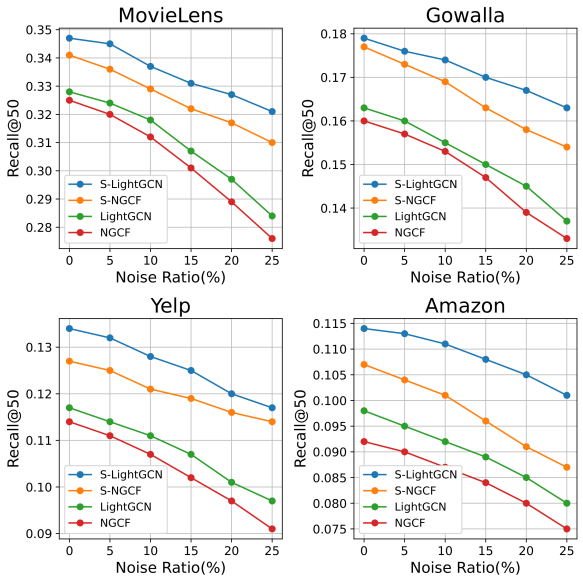

Figure 4: The model robustness of SGCNs and GCNs to the different levels of noisy edges.

4.3 Robustness Analysis (RQ2)

Noisy Edges Injection. As discussed before, the performance of GCNs is sensitive to noise since the misleading information can be massively propagated from node to node via noisy edges. In this part, we further conduct simulated experiments to investigate the robustness of SGCNs to noisy edges on the graphs. For each dataset, we randomly connect the unobserved edges that serve as false positive interactions for each user in the training set. We then evaluate how different models behave on the simulated graphs with different ratios of noisy edges from 0% to 25%. For better comparison, we mainly focus on SGCNs and GCNs, and the results of the rest baselines are omitted due to their inferior performance. All the simulated experiments are repeated five times and the average results are shown in Figure 4.

From Figure 4, we observe SGCNs consistently outperform GCNs under different ratios of noise on all datasets. The margins achieved by SGCNs over GCNs become larger as the number of noisy edges increases. For example, S-LightGCN achieves over 13% improvement over LightGCN in the setting of 25% noise rate on MovieLens dataset. These comparisons demonstrate that the random messages/nodes sampling strategies used in NGCF/LightGCN are still vulnerable to noisy edges. Figure 5 shows the training curves of training loss and the testing recall for MovieLens dataset with 25% noise. Clearly, the original GCNs have a risk of over-fitting to the noise. For example, GCNs attain stable training errors but produce large uncertainty in the stage of validation, i.e., the performance of GCNs slightly decreases with more training epochs. Conversely, SGCNs work well for both training and validation.

SGCNs address the noise by introducing the trainable stochastic binary masks and low-rank constraint. The stochastic binary masks have the potential to serve as L_0 regularization, which drives the insignificant or noisy edges to be *exact* zero. By sparsifying

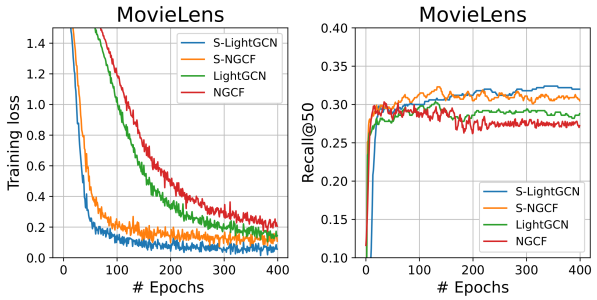

Figure 5: Analysis of over-fitting for SGCNs and GCNs, which are evaluated by training loss and testing recall for noisy MovieLens dataset.

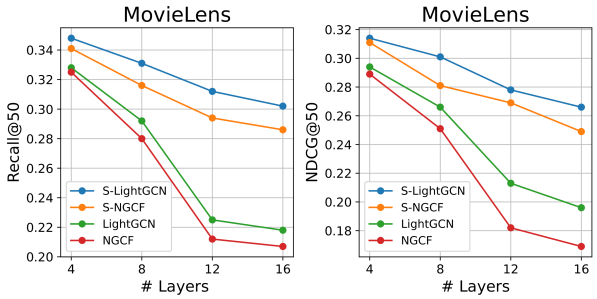

Figure 6: Analysis of over-smoothing for SGCNs and GCNs.

the graph, we can avoid unnecessary computation in the stage of message passing, thus alleviating over-fitting and improving the generalization ability. Meanwhile, the low-rank constraint guarantees that the structure information of graph is well preserved by optimizing its principal singular values. As a result, SGCNs can highly reduce the impact of the noisy edges and thus improve the robustness of vanilla GCNs.

4.4 Parameter Sensitivity (RQ3)

We further investigate the parameter sensitivity of SGCNs with respect to the following hyper-parameters: the number of layers K, two regularizer parameters $\{\beta, \gamma\}$ in Eq. (13), and the number of top-n singular values to approximate nuclear norm in Eq. (12). We mainly use the MovieLens for hyper-parameter studies, the results show the same trend for other datasets which are omitted for saving space.

4.4.1 ***Over-smoothing.*** The over-smoothing phenomenon exists when training deeper GCNs [25]. To illustrate its influence, we conduct experiments with varying number of GCN layers K within $\{4, 8, 16, 32\}$. The results are presented in Figure 6. We can observe a significant performance drop for both NGCF and LightGCN by increasing the number of layers. Our SGCNs successfully alleviate the over-smoothing issue. The reason is that the stochastic binary masks enable a subset of neighbor aggregation instead of

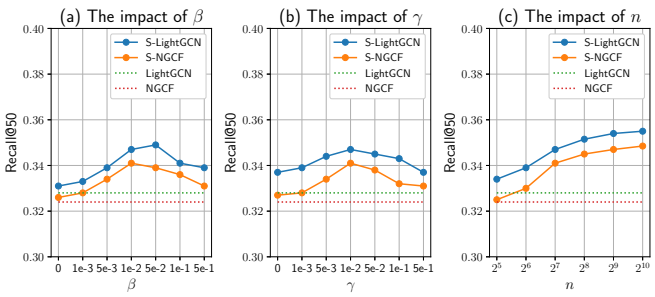

Figure 7: Results of parameter sensitivity on MovieLens.

full aggregation during training. This strategy prevents all node representations converging to the same value as the GCNs go deeper, which improves the generalization ability in the testing phase.

Our findings are consistent with the recent work DropEdge [35]. In fact, if we set $\beta = \gamma = 0$ and allow the stochastic masks to randomly drop certain rate of edges (e.g., simply detach the masks from the computational graph), our SGCNs can be then degraded to DropEdge. In our experiments, we find that non-zero settings of β and γ in SGCNs generally outperform DropEdge. We argue that DropEdge, a random dropping method, cannot discern between true or noisy edges, while SGCNs can precisely remove the noisy edges with the parameterized masks.

4.4.2 ***Regularizers.*** There are two major regularization parameters β and γ for sparsity $\mathcal{R}_s$ and low-rank constraints $\mathcal{R}_l$. Fig. 7(a) and 7(b) show the performance by changing one parameter while fixing the other as 0.01. As can be seen, the non-zero choices of β and γ demonstrate the importance of the regularization terms in our models. Even in the worst settings of $\beta = 0$ or $\gamma = 0$, SGCNs are still better than the baselines. In the extreme case, i.e., setting $\beta = \gamma = 0$ and turning on all masks to be all-ones matrices, our SGCNs exactly become GCNs.

Figure 7(c) also shows the effect of the number of top-n singular values to approximate the nuclear norm. We observe that the performance increases with a larger n. Nevertheless, larger n leads to more running time. It is reasonable to set n within $[2^7, 2^8]$ in our experiments.

5 CONCLUSION AND FUTURE WORK

In this paper, we propose Structured Graph Convolutional Networks to reduce the negative effects of noise in user-item bipartite graphs. In particular, we enforce sparsity and low-rank structure of the input graph while simultaneously training the parameters of GCNs. Our proposed SGCNs are compatible with various GCNs, such as NGCF and LightGCN, which can improve their robustness and generalization. The extensive experiments with real-world datasets show that SGCNs outperform the existing baselines.

For future work, we plan to improve our SGCNs by incorporating semantic information of graphs, such as users' social information or items' knowledge graph. Moreover, we are interested in applying pre-training strategies to enhance the embeddings of users and items. Lastly, we will also explore the explainability of GCNs, e.g., discovering a compact subgraph structure for a target user.

REFERENCES

[1] Zeyuan Allen-Zhu and Yuanzhi Li. 2016. LazySVD: Even faster SVD decomposition yet without agonizing pain. In *NeurIPS*. 974–982.

[2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[3] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. In *KDD Workshop on Deep Learning Day*.

[4] Huiyuan Chen and Jing Li. 2017. Learning multiple similarities of users and items in recommender systems. In *ICDM*. IEEE, 811–816.

[5] Huiyuan Chen and Jing Li. 2020. Neural Tensor Model for Learning Multi-Aspect Factors in Recommender Systems. In *IJCAI*.

[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *DLRS*. 7–10.

[7] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial Attack on Graph Structured Data. In *ICML*. 1115–1124.

[8] Zhe Dong, Andriy Mnih, and George Tucker. 2020. DisARM: An Antithetic Gradient Estimator for Binary Latent Variables. In *NeurIPS*.

[9] Negin Entezari, Saba A Al-Sayouri, Amirali Darvishzadeh, and Evangelos E Papalexakis. 2020. All You Need Is Low (Rank) Defending Against Adversarial Attacks on Graphs. In *WSDM*. 169–177.

[10] David Eppstein, Zvi Galil, Giuseppe F Italiano, and Amnon Nissenzweig. 1997. Sparsification—a technique for speeding up dynamic graph algorithms. *Journal of the ACM (JACM)* (1997), 669–696.

[11] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *WWW*. 417–426.

[12] Santo Fortunato. 2010. Community detection in graphs. *Physics reports* 486, 3-5 (2010), 75–174.

[13] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. 2019. Learning Discrete Structures for Graph Neural Networks. In *ICML*. 1972–1982.

[14] Ming Gao, Leihui Chen, Xiangnan He, and Aoying Zhou. 2018. Bine: Bipartite network embedding. In *SIGIR*. 715–724.

[15] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In *IJCAI*.

[16] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1024–1034.

[17] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, YongDong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR*. 639–648.

[18] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*. 173–182.

[19] Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. 2015. Matrix backpropagation for deep networks with structured layers. In *CVPR*. 2965–2973.

[20] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical Reparameterization with Gumbel-Softmax. In *ICLR*.

[21] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020. Graph Structure Learning for Robust Graph Neural Networks. In *KDD*. 66–74.

[22] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[23] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* (2009), 30–37.

[24] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns?. In *CVPR*. 9267–9276.

[25] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*.

[26] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *KDD*. 1754–1763.

[27] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. 2018. Variational autoencoders for collaborative filtering. In *WWW*. 689–698.

[28] Christos Louizos, Max Welling, and Diederik P Kingma. 2019. Learning Sparse Neural Networks through L_0 Regularization. In *ICLR*.

[29] Dongsheng Luo, Wei Cheng, Wenchao Yu, Bo Zong, Jingchao Ni, Haifeng Chen, and Xiang Zhang. 2021. Learning to Drop: Robust Graph Neural Network via Topological Denoising. In *WSDM*.

[30] Federico Monti, Michael Bronstein, and Xavier Bresson. 2017. Geometric matrix completion with recurrent multi-graph neural networks. In *NeurIPS*. 3697–3707.

[31] Cameron Musco and Christopher Musco. 2015. Randomized block krylov methods for stronger and faster approximate singular value decomposition. *NeurIPS* (2015), 1396–1404.

[32] Xia Ning and George Karypis. 2011. Slim: Sparse linear methods for top-n recommender systems. In *ICDM*. 497–506.

[33] Hoang NT and Takanori Maehara. 2019. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550* (2019).

[34] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*. 452–461.

[35] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2019. Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*.

[36] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290, 5500 (2000), 2323–2326.

[37] Yuta Saito, Suguru Yaginuma, Yuta Nishino, Hayato Sakata, and Kazuhide Nakata. 2020. Unbiased recommender learning from missing-not-at-random implicit feedback. In *WSDM*. 501–509.

[38] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* (2014), 1929–1958.

[39] Joshua B Tenenbaum, Vin De Silva, and John C Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *science* 290, 5500 (2000), 2319–2323.

[40] Maksims Volkovs, Guangwei Yu, and Tomi Poutanen. 2017. Dropoutnet: Addressing cold start in recommender systems. In *NeurIPS*. 4957–4966.

[41] Wei Wang, Zheng Dang, Yinlin Hu, Pascal Fua, and Mathieu Salzmann. 2019. Backpropagation-friendly eigendecomposition. In *NeurIPS*. 3162–3170.

[42] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *SIGIR*. 165–174.

[43] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[44] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *ICML*. 6861–6871.

[45] Jheng-Hong Yang, Chih-Ming Chen, Chuan-Ju Wang, and Ming-Feng Tsai. 2018. HOP-rec: high-order proximity for implicit recommendation. In *RecSys*. 140–144.

[46] Mingzhang Yin and Mingyuan Zhou. 2019. ARM: Augment-REINFORCE-merge gradient for stochastic binary networks. In *ICLR*.

[47] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*. 974–983.

[48] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)* (2019), 1–38.

[49] Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, and Wei Wang. 2020. Robust graph representation learning via neural sparsification. In *ICML*. 11458–11468.

[50] Lei Zheng, Chun-Ta Lu, Fei Jiang, Jiawei Zhang, and Philip S Yu. 2018. Spectral collaborative filtering. In *RecSys*. 311–319.

[51] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *KDD*. 1059–1068.

[52] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2019. Robust graph convolutional networks against adversarial attacks. In *KDD*. 1399–1407.