A Framework for Recommending Accurate and Diverse Items Using Bayesian Graph Convolutional Neural Networks

Jianing Sun[1†], Wei Guo[2†], Dengcheng Zhang[3†],Yingxue Zhang[1*], Florence Regol [4*‡], Yaochen Hu[1], Huifeng Guo[2], Ruiming Tang[2], Han Yuan[3], Xiuqiang He[2], Mark Coates[4]
1. Huawei Noah's Ark Lab, Montreal Research Center, Montreal, QC, Canada
{jianing.sun, yingxue.zhang, yaochen.hu}@huawei.com
2. Huawei Noah's Ark Lab, Shenzhen, China
{guowei67, huifeng.guo, tangruiming, hexiuqiang1}@huawei.com
3. Huawei Distributed and Parallel Software Lab, Hangzhou, China
{zhangdengcheng, yuanhan3}@huawei.com
4. Department of Electrical and Computer Eningeering, McGill University, Montreal, QC, Canada
florence.robert-regol@mail.mcgill.ca, mark.coates@mcgill.ca

ABSTRACT

Personalized recommender systems are playing an increasingly important role for online consumption platforms. Because of the multitude of relationships existing in recommender systems, Graph Neural Networks (GNNs) based approaches have been proposed to better characterize the various relationships between a user and items while modeling a user's preferences. Previous graph-based recommendation approaches process the observed user-item interaction graph as a ground-truth depiction of the relationships between users and items. However, especially in the implicit recommendation setting, all the unobserved user-item interactions are usually assumed to be negative samples. There are missing links that represent a user's future actions. In addition, there may be spurious or misleading positive interactions. To alleviate the above issue, in this work, we take a first step to introduce a principled way to model the uncertainty in the user-item interaction graph using the Bayesian Graph Convolutional Neural Network framework. We discuss how inference can be performed under our framework and provide a concrete formulation using the Bayesian Probabilistic Ranking training loss. We demonstrate the effectiveness of our proposed framework on four benchmark recommendation datasets. The proposed method outperforms state-of-the-art graph-based recommendation models. Furthermore, we conducted an offline evaluation on one industrial large-scale dataset. It shows that our proposed method outperforms the baselines, with the potential gain being more significant for cold-start users. This illustrates the potential practical benefit in real-world recommender systems.

*Corresponding author
†Contribute to this work equally
‡Work done as intern at Huawei Noah's Ark Lab, Montreal Research Center.

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403254

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

Graph Convolution Networks, Recommendation, Collaborative Filtering, Bayesian Graph Neural Networks

ACM Reference Format:
Jianing Sun, Wei Guo, Dengcheng Zhang, Yingxue Zhang, Florence Regol, Yaochen Hu, Huifeng Guo, Ruiming Tang, Han Yuan, Xiuqiang He, Mark Coates. 2020. A Framework for Recommending Accurate and Diverse Items Using Bayesian Graph Convolutional Neural Networks. In *Proceedings of the 26rd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'20), August 23-27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397271.3401123

1 INTRODUCTION

Online consumption (e.g., online shopping, watching videos, reading news, etc) has become more and more popular with the rapid development of the Internet and mobile devices. In such online applications, it is difficult to meet users' diverse and personalized needs. Under these circumstances, recommender systems, which guide users to find the items of interest in a gigantic and rapidly expanding pool of candidates, have emerged. Recommender systems tend to recommend the same item to users of similar interests, which is known as collaborative filtering (CF). As one of the most successful implementations of model-based CF methods, matrix factorization (MF) [18] models achieved the best performance in Netflix contest. MF models (such as pLAS [15], MF [18] and SVD++ [17]) learn user and item embeddings by reconstructing the historical user-item interactions. The learned user and item embeddings are expected to characterize user preferences and item features. More recently, deep learning models ([10, 13]), which can learn more complex non-linear relationships between users and items, have been developed to enhance the performance of traditional MF models.

Despite the progress, most existing MF based methods struggle to address three major challenges: the *sparsity issue*, the *uncertainty issue* and the *diversity issue*. *First*, existing methods rely on the historical user-item interactions, so when the interactions are

sparse, it is hard to learn high-quality representations, resulting in performance deterioration. *Second*, existing methods recognize the provided data as ground truth without *uncertainty*, but in many practical settings, the user-item interactions are collected from noisy environments. On one hand, some spurious user-item positive interactions are present due to noisy information; on the other hand, some potential user-item positive interactions are missing because the item is never presented to the user. This is falsely indicated as a negative interaction. *Third*, most existing methods for top-N recommendation focus on the relevance of each individual item independently and overlook the *diversity* of the top-N recommended items (i.e., the mutual influence between items). As observed in [22], ignoring diversity of the recommended list leads to sub-optimal performance.

Recently, graphs have been used to represent the relational information present in recommendation datasets. The user-item interaction can be naturally viewed as a bipartite graph. Furthermore, the similarities between users and the commonalities of items can be explicitly modelled as user-user and item-item graphs, respectively. These graphs allow algorithms to exploit the relationships between the entities. Graph Convolutional (Neural) Networks (GCNs) [5, 11, 16] have proven to be among the best performing architectures for a variety of graph learning tasks. The key idea in GCNs is to learn how to iteratively aggregate feature information from local graph neighborhoods using neural networks. This aggregation step allows each node to learn a more general node representation from its local neighborhood. Incorporating the graph representation in recommendation systems has shown to be effective for alleviating the data sparsity and cold start problems and improves the recommendation relevance [19, 34, 38]. The proposed systems exploit user-item interaction graphs [31, 34, 38], user-user and (or) item-item co-occurrence graphs [19] and heterogeneous graphs [3, 6] coming from heterogeneous interaction types (search, guide, click, etc.) or interaction motives.

Although there has been progress, existing graph-based recommendation models totally neglect the *uncertainty issue* in the bipartite user-item interaction graph. The edges provided in the observed bipartite graph are only based on historical interactions from the perspective of the data collector. They do not represent a complete picture of a user's interactions (for example, many other items may have been purchased from a different store and are therefore not present in the record). In addition, there may be spurious or misleading positive interactions, where a user has inadvertently clicked or purchased something on a temporary whim. On the other front, existing graph-based models also fail to address the lack of recommendation diversity. The neighborhood aggregation step in the graph-based recommendation approach generally leads to a learned user embedding that is considerably closer to the embeddings of items that he/she has interacted with previously. As a result, there is the potential that graph-based approaches can further dampen diversity by recommending very similar items to those involved in historical interactions. Expanding the aggregation neighbourhood can alleviate this to some extent, but this has the adverse effect of incorporating more spurious edges and noisy relationships, leading to performance degradation.

To address the limitations of the current GNN based recommendation approaches, we propose a new training framework based on

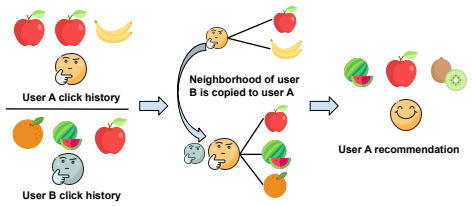


Figure 1: Motivation figure for node copying in the context of recommender system setting.

Bayesian Graph Neural Networks (BGNNs). The proposed BGNN incorporates a random graph generative model based on node-copying [24]. The node-copying model can be used to produce sample graphs that are similar to the observed graph, but they contain sufficient diversity in terms of edges to promote better learning. Importantly, the model is very efficient and scalable, allowing it to be applied to very large graphs. The BGNN allows us to address the uncertainty in the observed user-item interaction records and at the same time bring diversity into the recommendation results. Bayesian GNNs have not previously been used for the task of recommendation, but it has been shown that they can produce significant performance improvements in semi-supervised node classification when there are very few training labels [23, 24, 30, 39].

In this paper, we propose a novel Bayesian graph convolutional neural based recommender system framework, **BGCF** (Bayesian Graph Collaborative Filtering). There are two major contributions:

1) We introduce a principled way to address the uncertainty in the user-item bipartite graph in the recommendation training using Bayesian Graph Convolutional Neural Networks. We discuss how inference can be performed under our framework and provide a concrete formulation using the Bayesian Probabilistic Ranking training loss [26].

2) By performing thorough experiments on three commonly used recommendation datasets and one industrial large-scale dataset, we demonstrate that our proposed solution can achieve accurate and diverse recommendation results, and at the same time alleviate the data sparsity problem for users who do not have rich interaction history. The proposed method outperforms state-of-the-art graph-based recommender systems.

2 RELATED WORK

2.1 Model-based Collaborative Filtering Models

Collaborative filtering (CF) has been well studied for personalized recommender systems during the last decade. The basic assumption of CF is that users with similar preferences tend to like the same items, and items with similar audiences tend to have the same features. Model-based CF methods learn the similarities between items and users by fitting a model to the user-item interaction data. Latent factor models are common, such as probabilistic Latent Semantic Analysis (pLAS [15]), Matrix Factorization (MF [18]) and SVD++ [17]. They learn user and item embeddings by reconstructing the historical user-item interactions. The learned user and item embeddings are expected to characterize user preferences and item features. Predicting an unknown rating of a user-item pair relies on the learned user and item embeddings (the predicted score is the inner product of the corresponding user and item embeddings).

Despite their success, model-based CF methods suffer from two limitations: (1) they are confined to the inner-product mechanism to measure the similarity between users and items and (2) as a result of merely relying on user-item interactions they suffer from data sparsity. Many approaches have been proposed to address these two problems. Recently, neural networks have been incorporated into collaborative filtering architectures [8, 13]. These use a combination of fully-connected layers, convolution, inner-products and sub-nets to capture complex similarity relationships. An effective and common approach to alleviate the data sparsity problem is to leverage side information. For example, factorization machines [25] can provide a mechanism for incorporating side information such as user demographics and item attributes. Another line of research to tackle the sparsity problem is to exploit structure proximity in the user-item bipartite graph (or other graph information), which will be elaborated upon in the next section.

2.2 Graph Based Recommendation

There has been a considerable research effort devoted to the use of graph models in recommendation systems. Early works [9, 37] use traditional random walks and label propagation to model the similarity scores for user-item pairs. With the success of graph (convolutional) neural networks [5, 11, 16] on a wide range of applications, recent works have switched focus to applying GNNs in recommendation systems [31, 34, 38]. Graph Convolutional Matrix Completion (GCMC) [31] models the recommendation task as a matrix completion problem and uses a graph convolution autoencoder to learn user and item embeddings. Pinterest have proposed PinSage [38], a large-scale GNN-based recommendation model to learn the embeddings from the pin-board bipartite graph. This has been reported to achieve significant performance gains for the Pinterest recommendation system. Neural Graph Collaborative Filtering (NGCF) [34] designs a novel information propagation layer which enables explicit interactions between a user and its neighbor items to learn embeddings for users and items on the user-item interaction bipartite graph. GNNs have also been employed for the specific task of social recommendation in [7, 28, 35, 36] to better leverage the user's social relationships. It is worth mentioning another line of work which addresses the complex and heterogeneous interaction types between users and items in large-scale e-commerce networks [3, 6]. This problem setting is not in the scope of this paper since we address only the setting where there is a single type of interaction between user and item. However, with an extended graph generative model, our proposed approach has the potential to generalize to the case of multiple interaction types (the heterogeneous graph setting).

2.3 Bayesian Graph Neural Networks

Almost all GNNs approaches process a graph as a ground-truth depiction of the relationship between nodes, but often the graphs employed in applications are derived from noisy data. Addressing the uncertainty on the underlying graph was first considered in [39] for the problem of node classification. In this work, the authors target the inference of a generative graph model (using an MMSBM [20] as the random graph model) to address the uncertainty in the underlying observed graph and formulate the structure uncertainty exploration using a Bayesian framework. However, the approach is not flexible and does not utilize the node attributes or labels. These limitations were addressed in the follow-up works [23, 24], where [23] uses a non-parametric model for the graph generative model and [24] proposes a node copying model to achieve flexibility in the generative model and improve computational efficiency.

In the context of recommendation, [4, 14, 21] directly apply a Bayesian framework to model the users' preferences. Our approach is very different in that we use the Bayesian framework to model the uncertainty in the interaction graph within our graph neural network model. In the following sections, we address in depth how we can incorporate the Bayesian graph neural network framework into the recommendation system to better model the uncertainty in the user-item interaction graph.

3 PRELIMINARIES

We base our model on a Bayesian formulation that incorporates graph uncertainty, the generative graph model of node copying, and the Bayesian Personalized Ranking loss. We briefly review these three components in the preliminary section.

3.1 Bayesian Graph Convolutional Networks

In [39], to alleviate the effect of the potential noise in the observed graph, the authors view the graph as a random variable and consider a Bayesian approach. The framework they present can be generalized to target any prediction task and value of interest. To extend the model, we need to specify a probability function for the value of interest that depends on a graph $\mathcal{G}$, the parameters for the graph generation model or a vector of other intermediate random parameters $\boldsymbol{\lambda}$, and any available node attributes $\mathcal{D}$: $p(\cdot|\mathcal{D}, \mathcal{G}, \boldsymbol{\lambda})$. The posterior of interest $p(\cdot|\mathcal{D}, \mathcal{G}_{obs})$ is obtained by marginalizing over the random variables $\mathcal{G}$ and $\boldsymbol{\lambda}$:

$$p(\cdot|\mathcal{D}, \mathcal{G}_{obs}) = \int p(\cdot|\mathcal{D}, \mathcal{G}, \boldsymbol{\lambda}) p(\boldsymbol{\lambda}|\mathcal{D}, \mathcal{G}, \mathcal{G}_{obs}) \, p(\mathcal{G}|\mathcal{D}, \mathcal{G}_{obs}) \, d\mathcal{G} \, d\boldsymbol{\lambda} \,. \quad (1)$$

Zhang et al. presented this model in [39] for the node classification task. In their case, the value of interest is the node labels. They used a stochastic block model for $p(\mathcal{G}|\mathcal{G}_{obs})$, which is an appropriate choice for node classification. However the above Bayesian Graph Convolutional Network (BGCN) formulation has two limitations. First, it ignores the possible dependence of the graph $\mathcal{G}$ on $\mathcal{D}$. Second, it requires an appropriate parametric random graph model (such as a Mixed-Membership Stochastic Block Model (MMSBM) model [20]). Whether a model is appropriate depends very much on the encountered graph structure, which can vary greatly for different problem settings.

Since there is no inherent block structure in a recommender system bipartite graph, the MMSBM is not an applicable graph model. As an alternative, we use a more general generative model for graphs based on copying nodes, as introduced in [24]. We demonstrate in the following sections that this model can be adapted naturally to the recommender system setting.

3.2 Node Copying

In [24], Pal et al. introduce the node copying model for $p(\mathcal{G})$. Samples from this model are generated by probabilistically rearranging (with replacement) the adjacency matrix rows of the observed graph A^{obs}. The i^{th}-row in the adjacency matrix $A^{obs}_{i,:}$ encodes the neighborhood of node i, so if we have a sampled graph with $A_{i,:} = A^{obs}_{j,:}$, the neighborhood of node j was *copied* to node i.

The copying operation for the whole graph can be expressed using an auxiliary random vector $\boldsymbol{\zeta} = [\zeta^1, \zeta^2, ...\zeta^N]^\top \in \{1, 2, ...N\}^N$, where each entry ζ_i denotes the row $A^{obs}_{\zeta_i,:}$ that will be placed in $A_{i,:}$ of the sampled $\mathcal{G}$. The distribution of $p(\boldsymbol{\zeta})$ should be proportional to some node similarity that can be derived from the observed graph and data $(\mathcal{G}_{obs}, \mathcal{D})$. A suitable node similarity is task dependent and should be specified accordingly.

The complete graph sampling process involves two phases. First we sample $\boldsymbol{\zeta}$. The entries are assumed to be mutually independent so it can be factorized as follows:

$$p(\boldsymbol{\zeta}|\mathcal{G}_{obs}, \mathcal{D}) = \prod_{i=1}^{N} p(\zeta^i|\mathcal{G}_{obs}, \mathcal{D}) \,. \tag{2}$$

Once a realization of $\boldsymbol{\zeta}$ is obtained, a second layer of randomness is added by performing the copying for each node with some probability $0 \ll \epsilon \leq 1$. The event of copying node j to node q is denoted by the indicator function $\mathbb{1}_{\{\mathcal{G}_q = \mathcal{G}_{obs,j}\}}$, so the generative model can be written as:

$$p(\mathcal{G}|\mathcal{G}_{obs}, \boldsymbol{\zeta}) = \prod_{i=1}^{N} \epsilon^{\mathbb{1}_{\{\mathcal{G}_i = \mathcal{G}_{obs,\zeta^i}\}}} (1-\epsilon)^{\mathbb{1}_{\{\mathcal{G}_i = \mathcal{G}_{obs,i}\}}} \,. \tag{3}$$

3.3 Bayesian Personalized Ranking loss for Implicit Recommendation

In [26], Rendle et al. introduce a ranking loss for recommendation systems based on a Bayesian model. We build on that model in this work, extending it to take into account the multiple graphs of the node-copying BGNN, so we now provide a brief review.

Let us denote the set of items that are neighbours in the observed graph for user u as $I^+_u := \{i \in I : (u, i) \in \mathcal{G}_{obs}\}$, where I is the set of all items. The training set can then be written as:

$$D_S := \left\{(u, i, j) | i \in I^+_u \wedge j \in I \setminus I^+_u\right\} \,. \tag{4}$$

In other words, the training set is all triples (u, i, j) such that user u interacted with i but did not interact with j. The test set, denoted $\overline{D_S}$, consists of all triples (u, i, j) such that neither edge (u, i) nor (u, j) appears in $\mathcal{G}_{obs}$. The goal of the recommender system is to generate a total ranking $>_u$ of all items for each user u. The binary relation $>_u$ is required to be a total order on the set of items I. The relation $i >_u j$ specifies that user u prefers item i to item j.

In the Bayesian personalized ranking framework of [26], our task is to maximize:

$$p\left(\Theta|\{>_u\}_{D_S}\right) \propto p\left(\{>_u\}_{D_S}|\Theta\right) p(\Theta) \,. \tag{5}$$

Here Θ are the parameters of the model, and $\{>_u\}_{D_S}$ are the observed preferences in the training data. We aim to identify the parameters Θ that maximize this posterior over all users and all pairs of items. We assume that users act independently, so:

$$p\left(\{>_u\}_{D_S}|\Theta\right) = \prod_{(u,i,j)\in D_S} p\left(i >_u j|\Theta\right) \tag{6}$$

We define the probability that a user prefers item i over j as

$$p\left(i >_u j|\Theta\right) := \sigma\left(\hat{x}_{uij}(\Theta)\right) \,. \tag{7}$$

Here $\hat{x}_{uij}$ is a function of the model parameters Θ and the observed graph for each triple (u, i, j). In our case, we use the difference between the dot products of the user and item embeddings, so $\hat{x}_{uij}(\Theta) = h_u(\Theta) \cdot h_i(\Theta) - h_u(\Theta) \cdot h_j(\Theta)$.

If we adopt a normal distribution as the prior for $p(\Theta)$ then we can formulate the optimization objective as:

$$\begin{aligned} \text{BPR-OPT} &:= \ln p\left(\Theta|\{>_u\}_{D_S}\right) \\ &= \sum_{(u,i,j)\in D_S} \ln \sigma\left(\hat{x}_{uij}\right) - \lambda_\Theta ||\Theta||^2 \end{aligned} \tag{8}$$

We can optimize this via stochastic gradient descent by repeatedly drawing triples (u, i, j) randomly from the training set and updating the model parameters Θ.

4 METHODOLOGY

4.1 Bayesian Graph Neural Networks for Personalized Ranking

In this section we build on the Bayesian Personalized Ranking framework of [26] to develop an approach that incorporates the strengths of Bayesian GNNs and thus takes into account the uncertainty in the graph observations. We develop two strategies; our experimental approach is based on the second.

4.1.1 Marginalizing over sampled graphs. In this approach, we focus on the predictive posterior:

$$\begin{aligned} &p\left(\{>_u\}_{\overline{D_S}}|\{>_u\}_{D_S}\right) \propto \\ &\int_{\mathcal{G}} p(\mathcal{G}|\mathcal{G}_{obs}) \left[\int_\Theta p\left(\{>_u\}_{\overline{D_S}}|\Theta, \mathcal{G}\right) p(\Theta|\{>_u\}_{D_S}, \mathcal{G})\, d\Theta\right] d\mathcal{G} \,. \end{aligned} \tag{9}$$

Note that $\mathcal{G}_{obs}$ encodes the same information as $\{>_u\}_{D_S}$. A triple $(u, i, j) \in D_S$ indicates an edge (u, i) in $\mathcal{G}_{obs}$ (the bipartite user-item interaction graph) and the absence of the edge (u, j). We choose to write $p(\mathcal{G}|\mathcal{G}_{obs})$ instead of $p(\mathcal{G}|\{>_u\}_{D_S})$ to emphase the relationship between the sampled $\mathcal{G}$ and $\mathcal{G}_{obs}$. In this expression, we highlight that the probabilities are dependent on $\mathcal{G}$ because the sampled graph structure directly affects the models used to form the embeddings.

When attempting to maximize this predictive posterior, we can choose to approximate the inner integral with respect to Θ by $p\left(\{>_u\}_{\overline{D_S}}|\widehat{\Theta}_{\mathcal{G}}, \mathcal{G}\right)$, where $\widehat{\Theta}_{\mathcal{G}}$ are the model parameters that maximize $p(\Theta|\{>_u\}_{D_S}, \mathcal{G})$. This latter maximization is the same as BPR-OPT in (13), as outlined in Section 3.3, but the inclusion of $\mathcal{G}$ indicates that $\hat{x}_{uij}(\Theta, \mathcal{G}, \mathcal{G}_{obs})$ now takes into account the embeddings derived using $\mathcal{G}$ (and $\mathcal{G}_{obs}$), as expressed in (19).

Adopting a Monte-Carlo approximation to the integral, and drawing N_G graphs using the copying model from $p(\mathcal{G}|\mathcal{G}_{obs})$, we have:

$$p(\{>u\}_{\overline{D_S}}|\{>u\}_{D_S}) \approx \frac{1}{N_G}\sum_{l=1}^{N_G} p(\{>u\}_{\overline{D_S}}|\widehat{\Theta}_l, \mathcal{G}_l), \quad (10)$$

where $\widehat{\Theta}_l = \arg\max p(\Theta|\{>u\}_{D_S}, \mathcal{G}_l)$. The maximization to identify $\widehat{\Theta}_l$ is performed using stochastic gradient descent for each graph $\mathcal{G}_l$. The procedure thus involves sampling graphs and performing training of the weights for each graph.

4.1.2 Redefining the probability mapping to address uncertainty. The procedure described above involves increased computation, because we must train multiple models, one for each sampled graph. There are also more parameters in the collective models, so there is a potential for overfitting. We can alleviate these concerns by assuming a common model for all of the graphs. In this case, we return to the original BPR framework, but change our choice of $\hat{x}_{uij}(\Theta, \mathcal{G}_{obs})$. Let $h_{u,\mathcal{G}}$ denote the embedding in (19) for user u under sampled graph $\mathcal{G}$, and let $h_{i,\mathcal{G}}$ be the corresponding embedding for item i. Then we define:

$$\begin{aligned}\hat{x}_{uij}(\Theta, \mathcal{G}_{obs}) &:= \int_{\mathcal{G}} \left[h_{u,\mathcal{G}} \cdot h_{i,\mathcal{G}} - h_{u,\mathcal{G}} \cdot h_{j,\mathcal{G}}\right] p(\mathcal{G}|\mathcal{G}_{obs})\, d\mathcal{G} \\ &\approx \frac{1}{N_G}\sum_{l=1}^{N_G} \left[h_{u,\mathcal{G}_l} \cdot h_{i,\mathcal{G}_l} - h_{u,\mathcal{G}_l} \cdot h_{j,\mathcal{G}_l}\right], \quad (11)\end{aligned}$$

where the $\mathcal{G}_l$ are drawn from $p(\mathcal{G}|\mathcal{G}_{obs})$. With this formulation, we can identify the model parameters using stochastic gradient descent as for BPR-OPT, but instead of drawing just a triple (u, i, j), we first draw a graph $\mathcal{G}$ and then a triple $(u, i, j) \in D_S$. The final estimates of rankings are derived from

$$p\,(i >_u j|\Theta, \mathcal{G}_{obs}) := \sigma\left(\hat{x}_{uij}(\widehat{\Theta}, \mathcal{G}_{obs})\right). \quad (12)$$

where $\widehat{\Theta}$ is the solution to the maximization of:

$$\text{BPR–OPT}_{\mathcal{G}_{obs}} := \sum_{(u,i,j)\in D_S} \ln\sigma\left(\hat{x}_{uij}(\Theta, \mathcal{G}_{obs})\right) - \lambda_\Theta||\Theta||^2 \quad (13)$$

Equation (11), in conjunction with (7), illustrates that our final prediction of a ranking involves averaging the scores $\hat{x}_{uij}(\widehat{\Theta}, \mathcal{G}_{obs}, \mathcal{G})$ over all sampled graphs and then applying the sigmoid function.

4.2 Neighborhood Copying Graph Generative Model

In order to sample graphs $\mathcal{G}$ with the node copying graph generative model, we first need to form a suitable copying distribution

An intuitive way to measure similarity between users is to look at preferences over items they share in the historical interaction record. The Jaccard index is a statistic used for gauging the similarity between finite sample sets. It is defined as the size of the intersection divided by the size of the union of the sample sets. We specify the similarity between every pair of users $u_i, u_j \in \mathcal{U}$ as the Jaccard index for the interaction records of the two users:

$$sim(u_i, u_j) = \frac{\mathcal{N}(u_i) \cap \mathcal{N}(u_j)}{\mathcal{N}(u_i) \cup \mathcal{N}(u_j)}. \quad (14)$$

Here $\mathcal{N}(\cdot)$ denotes the neighborhood of a node in the observed user-item interaction graph $\mathcal{G}_{obs}$. To define $p(\zeta)$, we simply normalize the metric over the pairs of users and set it to 0 otherwise. As the distribution is not dependent on any additional information $\mathcal{D}$, it is only conditioned on $\mathcal{G}_{obs}$:

$$p(\zeta^j = m|\mathcal{G}_{obs}) = \begin{cases} \frac{sim(j,m)}{\sum_{i=1}^{|\mathcal{U}|} sim(j,i)} & \text{if } j, m \in \mathcal{U} \\ 0 & \text{otherwise} \end{cases} \quad (15)$$

Having specified this distribution, we can then adopt the graph sampling strategy from the copying model $p(\mathcal{G}|\zeta)$ according to equation (3). By sampling multiple graphs from our graph generative model, the observed graph, which is usually very sparse, can be directly augmented with likely neighbors discovered from the neighborhood of nodes with high similarities. In practice, we set the probability of copying nodes ϵ to be high to gain more additional edges.

4.3 Joint Training on Observed Graph and Sampled Graphs

From our proposed neighborhood copying graph generative model, we obtain l graph samples $\mathcal{G}_1, ..., \mathcal{G}_l \sim p(\mathcal{G}|\mathcal{G}_{obs}, \zeta)$. The next step is how to learn useful information from these sampled graphs. In this section, we first present our representation learning model, specifying how we learn user and item embeddings from both the observed graph and each individual graph sample.

4.3.1 Learning on Sampled Graphs $\mathcal{G}$ with Attention. Unlike the observed graph $\mathcal{G}_{obs}$ whose edges are all true labels, the graph samples introduce some spurious edges while attempting to find potential links. The graph attention network (GAT) [33] applies a self attention strategy operating on groups of spatially close neighbors to implicitly specify different weights for different nodes in a neighborhood. This follows the intuition that higher importance should be given to the neighbors that are more similar with the central node. GAT applied a single-layer feedforward neural network with multiple weights on sets of node feature vectors and on the transformation function to map from $\mathbb{R}^{2d} \to \mathbb{R}$ as the attention score.

To avoid the high complexity of GAT, we apply a simplified attention mechanism by simply taking the dot product between each neighbor with the central node as the attention coefficient. This reduces the influence of noisy edges in the neighborhood and emphasizes the discovered nodes with potentially positive effects. More specifically, the attention coefficient is calculated as follows:

$$\alpha_{jk} = \frac{\exp(\boldsymbol{e_j} \cdot \boldsymbol{e_k})}{\sum_{i\in N(j)} \exp(\boldsymbol{e_j} \cdot \boldsymbol{e_i})} \quad (16)$$

α_{jk} represents the importance weight of each node $k \in \mathcal{N}_{\mathcal{G}}(j)$ on the target node j, where $\mathcal{N}_{\mathcal{G}}(j)$ denotes the neighborhood of node j in the sampled graph $\mathcal{G}$. $\boldsymbol{e}_*$ are input node embeddings.

Once obtained, the attention coefficients are used to compute a linear combination over sets of neighbor nodes, together with a mean aggregator, to serve as the learned representation from a sampled graph $\mathcal{G}$:

$$\tilde{\boldsymbol{h}}_j^{\mathcal{G}} = \left[\left(\boldsymbol{W}_{\mathcal{G}}^1 \sum_{k\in\mathcal{N}_{\mathcal{G}}(j)} \alpha_{jk}\boldsymbol{e_k}\right) \| \left(\boldsymbol{W}_{\mathcal{G}}^2 \cdot \frac{1}{n_j}\sum_{k\in\mathcal{N}_{\mathcal{G}}(j)} \boldsymbol{e}_k\right)\right], \quad (17)$$

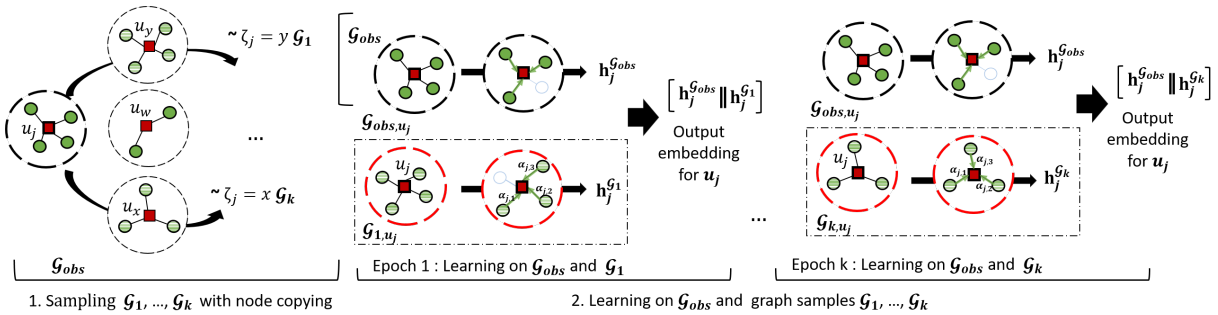


Figure 2: Overall architecture for our proposed BGCF training process.

where n_j denotes the number of neighbors of node j in a sampled graph $\mathcal{G}$. $\boldsymbol{W}^1_{\mathcal{G}}$ and $\boldsymbol{W}^2_{\mathcal{G}}$ are shared weight matrices. $\|$ denotes the concatenation operation.

4.3.2 Learning on Observed Graph $\mathcal{G}_{obs}$. For a target node j, we learn its representation vector $\tilde{\boldsymbol{h}}_j^{\mathcal{G}_{obs}}$ on the observed graph by applying a mean aggregator over its neighbors. We select the mean aggregator because edges in the observed graph are all ground-truth and the mean aggregator is the simplest permutation-invariant operator. The process is as follows:

$$\tilde{\boldsymbol{h}}_j^{\mathcal{G}_{obs}} = \sigma\Big(\boldsymbol{W}_{\mathcal{G}_{obs}} \cdot \frac{1}{n_j} \sum_{k \in \mathcal{N}_{\mathcal{G}_{obs}}(j)} \boldsymbol{e}_k\Big), \tag{18}$$

where n_j denotes the number of neighbors of node j in the observed graph $\mathcal{G}_{obs}$, and $\boldsymbol{W}_{\mathcal{G}_{obs}}$ is a shared weight matrix.

4.3.3 Joint Embedding Vector as Node Representation. By learning on the observed graph $\mathcal{G}_{obs}$ and the sampled graphs $\mathcal{G}$, for a target node j, we obtain two representation vectors $\tilde{\boldsymbol{h}}_j^{\mathcal{G}}$ and $\tilde{\boldsymbol{h}}_j^{\mathcal{G}_{obs}}$. These represent the potential preferences and observed preferences, respectively, of a user or an item. The final representation of a node is hence developed as the combination of $\tilde{\boldsymbol{h}}_j^{\mathcal{G}}$ and $\tilde{\boldsymbol{h}}_j^{\mathcal{G}_{obs}}$:

$$\hat{\boldsymbol{h}}_j = \sigma\big[\, \tilde{\boldsymbol{h}}_j^{\mathcal{G}} \,\|\, \tilde{\boldsymbol{h}}_j^{\mathcal{G}_{obs}} \,\big], \tag{19}$$

where $\sigma(\cdot)$ denotes the *tanh* non-linear transformation, and $\|$ denotes the concatenation operation.

5 EXPERIMENTS

Datasets. To evaluate the effectiveness of our method, we conduct extensive experiments on four popular benchmarks: *Amazon-Movies*, *Amazon-Beauty*, *Amazon-CDs*. These benchmark datasets are publicly accessible, real-world data with various domains, sizes, and sparsity levels. We filter out long-tailed users and items with fewer than 10 interactions for all 3 datasets.

For each user, we randomly select 20% of the rated items as ground truth for testing, The remaining 70% and 10% data constitutes the training and validation set. Table 1 summarizes the statistics of all the datasets.

Table 1: Statistics of public datasets.

Dataset	# User	# Item	# Interaction	Density
Movies	44,439	25,047	1,070,860	0.096%
Beauty	7,068	3,750	79,506	0.299%
CDs	43,169	35,648	777,426	0.051%

Amazon-Movies, **Amazon-Beauty** and **Amazon-CDs**: Amazon-review* is a popular dataset for product recommendations [12]. We select three subsets, Amazon-Movies, Beauty and CDs.

Evaluation Metrics. For all experiments, we evaluate the recommendation accuracy of our model and baselines in terms of *Recall@k* and *NDCG@k*. Because accuracy alone does not guarantee satisfactory recommendations, we also assess *serendipity@k* [22], which factors in how surprising and relevant a recommendation is. Surprise is measured as a weighted average of the differences between the probability that an item i is recommended for a specific user and the probability that item i is recommended for any user. It is computed [27] as:

$$\text{SRDP@k} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \Big(\frac{1}{|I_k(u)|} \sum_{i \in I_k(u)} \max\big(P_i(u) - P_i(\mathcal{U}), 0\big) * rel_i(u) \Big). \tag{20}$$

Here $P_i(u) = \frac{|I_k(u)| - rank_i}{|\mathrm{R}_{u,k}| - 1}$ represents the probability of recommending item i to a specific user u, and $P_i(\mathcal{U}) = D(i)/\sum_{u \in \mathcal{U}} D(u)$ represents the approximate probability of recommending that item for any user. We use $D(i)$ and $D(u)$ to denote the degrees of item i and user u in the observed graph.

Baselines. To demonstrate the effectiveness, we compare our proposed model with the following methods:
1) Classical collaborative filtering methods: BPRMF [26], NeuMF [13]
2) Graph neural network-based CF methods: GC-MC [31], PinSAGE [38], PinSAGE-LSTM, and NGCF [34].

Please refer to the supplementary material for the detailed descriptions of the above baselines.
3) Our proposed methods:

- **Base**: To prove that our proposed BGCF framework can be applied to any graph learning based recommendation models, we modified a simplified GNN-based base model from [29]. Instead

*http://jmcauley.ucsd.edu/data/amazon/

of applying GCN on self graphs, we utilize self connections as an additional regularization term in the loss function. A graph convolutionn layer with mean aggregator is applied on each side of the user-item bipartite graph, to learn user & item embeddings by aggregating from one-hop neighbors.

- **BGCF**: This is the main model that we propose; it incorporates a Bayesian graph neural network to account for uncertainty in the observed user-item interactions and employs node-copying as a graph-generative model.

6 DISCUSSION AND ANALYSIS

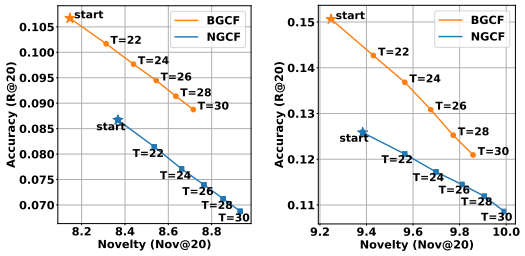
In Table 2, we can see that the proposed model outperforms all the baselines on *recall@20* and *.serendipity@20* metrics. This trend can be observed across all considered datasets. Also, the base model is consistently being outperformed by **BGCF** on all metrics, which suggests that the Bayesian formulation is advantageous. In the following section, we further investigate the advantages coming from this model by considering alternative metrics focused on diversity rather than recall. We also provide some insight into which kind of user sees the most improvement and conduct a more thorough ablation study.

6.1 Accuracy-Diversity Trade-off under Ranking-based Techniques

Diversity (i.e. novelty) and accuracy metrics have different objective that are at odds with each other, in that increasing one can result in sacrificing the other. This trade-off has been studied in both the natural language generation setting [2], where Caccia et al. evaluate diversity vs. quality and show the danger of focusing on one metric, and in recommendation [1], where the trade-off between diversity-in-top-N and precision-in-top-N has been examined. In practice, having a recommender system with a better accuracy/diversity trade-off allows the platform to have more personalized recommendations while maintaining comparable levels of accuracy. In this section we evaluate the quality of accuracy-diversity trade-off between our proposed algorithm and the second best model which has the highest accuracy results than other baselines. To evaluate on the quality of trade-off, we need a method that can modify one metric. In [1], Adomavicius et al. proposed the application of another ranking function $rank(i, T_R)$ upon the top-k recommendation list. The parameter T_R is a ranking threshold; only items initially ranked above T_R are re-ranked.

In our case, we apply a popularity based ranking function, $rank(i, T_R) = D(i)$, where $D(i)$ denotes the degree of an item. Given a ranked recommendation list with $k = 20$, we re-rank the items above T_R based on their popularity. Thus, items with high prediction score but high popularity are demoted. Choosing different T_R values allows the user to set the desired balance between accuracy and novelty, e.g., when $T_R = 22$, we re-rank the top 22 items based on popularity and produce the final top-20 results based on the re-ranked list.

We conduct the re-ranking process on two datasets, comparing the effect on our model and on NGCF. The novelty of recommendations is computed [32] as $\text{Nov@k} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \sum_{i \in I_k(u)} \frac{-\log_2 D(i)/|\mathcal{U}|}{|I_k(u)|}$.



(a) Amazon-Movies (b) Amazon-CDs

Figure 3: Re-ranking results on our proposed algorithm and second base model on Amazon-CDs and Amazon-Movies under thresholds $T_R \in \{22, 24, 26, 28, 30\}$.

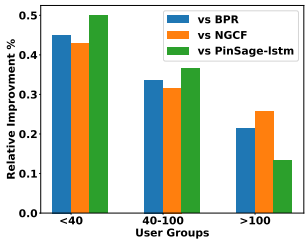


Figure 4: Distribution of the relative improvement of our proposed model BGCF over different baselines on Amazon-CDs dataset with respect to user category (partitioned by degree, i.e., the number of historical interactions).

From Figure 3 we can see that for any desired diversity-level (measured by *Nov@20*), our model has a higher accuracy than NGCF.

6.2 User density analysis

We analyse our results to highlight which type of users benefits from this recommender model. In Figure 4, we can view the relative improvement of our proposed model over the best preforming baselines (BPR, NGCF, PinSage-lstm) for different groups of users based on sparsity. The overall trend is that the algorithm offers the most relative improvement over the alternative methods for users with less items, hence for sparser users (user that have a history of less than 40 clicks).

Statistics of Sampled Graphs. In Table 3, we analyse the quality of the generated graphs from the copying model by computing the edge overlap between the observed graph and the realizations. As a reference point, we also report the same statistics obtained from a copying model with an uninformative copying distribution ζ (uniform over all nodes) $\mathcal{G} \sim p(\mathcal{G}|\mathcal{G}_{obs}, \zeta_{Uni})$ and from a random degree-corrected model $p_{degree}(\mathcal{G})$ which randomly assign edges to each user while preserving each user's degree. From this table we can see that the copying model is able to maintain some of the initial edges and can even discover some directly present in the test set, while the other models generate graphs that contain less than 2% of the initial links (training and test clicks combined). This

Table 2: The overall performance comparison. Bold values denote scenarios where a Wilcoxon signed rank test indicates a statistically significant difference between the best and second-best (underline) algorithms

Amazon-CDs	R@20	N@20	SRDP@20	Amazon-Movies	R@20	N@20	SRDP@20	Amazon-Beauty	R@20	N@20	SRDP@20
BPRMF	0.0794	0.0501	0.0097	**BPRMF**	0.0667	0.0436	0.0103	**BPRMF**	0.1312	0.0778	0.0100
NMF	0.1008	0.0604	0.0109	**NMF**	0.0820	0.0511	0.0110	**NMF**	0.1152	0.0692	0.0081
GC-MC	0.0791	0.0473	0.0093	**GC-MC**	0.0638	0.0401	0.0084	**GC-MC**	0.1082	0.0659	0.0077
PinSage	0.1265	0.0799	0.0134	**PinSage**	0.0872	0.0559	0.0114	**PinSage**	0.1378	0.0821	0.0138
PinSage-lstm	0.1269	0.0805	0.0135	**PinSage-lstm**	0.0886	0.0567	0.0115	**PinSage-lstm**	0.1354	0.0786	0.0135
NGCF	0.1258	0.0792	0.0129	**NGCF**	0.0866	0.0555	0.0112	**NGCF**	0.1513	**0.0917**	0.0151
Base	0.1455	0.0907	0.0161	**Base**	0.1023	0.0663	0.0132	**Base**	0.1432	0.0869	0.0145
BGCF	**0.1506**	**0.0948**	**0.0168**	**BGCF**	**0.1066**	**0.0693**	**0.0139**	**BGCF**	**0.1534**	0.0912	**0.0153**

Table 3: Overlap ratio of the generated samples with the observed graph (%) averaged over 10 graphs spurious edges / train edges/ test edges ratio (%).

Dataset	$p_{degree}(\mathcal{G})$	$p(\mathcal{G}\|\mathcal{G}_{obs}, \zeta_{uni})$	$p(\mathcal{G}\|\mathcal{G}_{obs}, \zeta)$
CDs	98.9 / 1.0 / 0.1	98.8 / 1.1 / 0.1	60.5 / 38.3 / 1.2
Movie	98.9 / 1.0 / 0.1	98.7 / 1.1 / 0.2	66.5 / 32.4 / 1.1
Beauty	98.5 / 1.4 / 0.1	98.4 / 1.5 / 0.1	80.8 / 18.3 / 0.9

highlights the sparsity of the datasets and further motivates the use of the copying model.

6.3 Ablation Studies

As our proposed model involves multiple novel components, we conduct an ablation study to assess their individual contributions to the performance. In particular, we want to evaluate the benefits coming from 1) using the sampled graphs from the node copying model ($\sim \mathcal{G}$); 2) training jointly with the observed graph and the sampled graphs ($\sim \mathcal{G} + \mathcal{G}_{obs}$); and 3) using attention aggregation on the sampled graphs together with mean aggregation. We report results on partial architectures for one trial on the Amazon-CDs dataset in Table 4. The first observation is that jointly training with both graphs (mean + $\mathcal{G}_{obs}$ + $\sim \mathcal{G}$) is better than only training with either the observed graphs or the sampled graphs. Secondly, using an attention-based aggregation also yields a performance improvement of the same order as the joint training.

Table 4: Ablation study for the proposed algorithm.

Architecture	R@10	R@20	N@10	N@20
Base (mean + $\mathcal{G}_{obs}$)	0.0971	0.1455	0.0761	0.0917
mean + $\sim \mathcal{G}$	0.0989	0.1459	0.0769	0.0921
mean + $\mathcal{G}_{obs}$ + $\sim \mathcal{G}$	0.1004	0.1486	0.0784	0.0938
BGCF*	**0.1016**	**0.1506**	**0.0791**	**0.0948**

7 APPLICATION ON INDUSTRIAL DATASET

In addition to the benchmark comparison, we also validate the superiority of our proposed model on an industrial dataset.

7.1 Experimental Settings

7.1.1 Datasets. We collect and sample 33 consecutive days of user-app download records from the game center of a mainstream App Store for training, and the next 7 days for testing. Compared with the four publicly accessible datasets used above, this industrial dataset has two unique characteristics:

1) Rich side information, including app features (e.g., app size, category and etc), user features (e.g., user's various behaviors in the App Store), and context features (e.g., device type, operation time and etc) are kept in the dataset. As side information is usually available in live recommender systems and is very valuable to alleviate the cold-start problem, it is necessary to verify the performance of our model under such a setting.

2) Benchmark datasets are usually split into training and test sets in such a way that the interaction records of each user are randomly distributed between the two sets. Every user in the test set is therefore also present in the training set. However, in live recommender systems, it is required to serve every user, including those that have had no interactions in the system before, who are called cold-start users. In order to better match this scenario, we split our industrial dataset according to the timestamp, i.e., the first 33 consecutive days of records are the training set, while the next 7 days of records are the test set.

7.1.2 Evaluation Protocol. In the industrial dataset, there are 943,177 users, 9,768 apps and 4,000,000 download records. Each download record in the dataset is considered as a positive instance, while the interactions of a user and her unobserved apps in the universal app set are all considered as negative. We evaluate the performance of difference models on such binary labeled instances by LogLoss (also known as cross entropy), a widely used metric in binary classification. A smaller LogLoss value indicates better performance. We use the same baseline models described in Section 5.

7.2 Performance Comparison

We compare the performance of different models for three different settings: all the test users, warm-start test users (who appear in the training set) and cold-start test users (who do not appear in the training set). The performance comparison on the industrial dataset is presented in Table 5. As can be observed, our model has superior performancecompared to the best baseline (NGCF) by 2.17% in terms of LogLoss on all the test users. This result demonstrates the effectiveness of our model in recommendation scenarios with

Table 5: Performance Comparison (LogLoss) in Industrial Dataset. The lower the better.

	All users	Warm-start users	Cold-start users
BPR	0.37618	0.33590	0.41103
NMF	0.36028	0.31783	0.39588
GC-MC	0.36050	0.31189	0.40083
PinSage	0.36075	0.31032	0.40288
PinSage-lstm	0.35974	0.31090	0.40056
NGCF	0.35806	0.31158	0.39713
BGCF	**0.35030**	**0.30624**	**0.38729**

rich side information. More specifically, our model outperforms the best baseline (NGCF) by 1.71% and 2.48% on warm-start users and cold-start users, respectively. This improvement verifies the superiority of our model in alleviating the cold-start issue.

8 CONCLUSION

In this work, we propose a novel recommendation model based on Bayesian Graph Neural Networks and the node copying graph generative model to naturally incorporate the uncertainty in the underlying user-item interaction graph. From our extensive experiments, our proposed model shows consistent recommendation accuracy improvement over state-of-the-art methods for four public benchmark datasets and one large-scale industrial dataset. Via thorough analysis of case studies, we highlight that our proposed Bayesian Graph Collaborative Filtering framework can bring more recommendation diversity and alleviate the data sparsity problem. Besides, using Mindspore, the all-scenario deep learning framework developed by Huawei, the computation in the proposed approach can be easily and automatically parallelized to execute on multiple GPUs, rendering training extremely efficient.

REFERENCES

[1] Gediminas Adomavicius and YoungOk Kwon. 2012. Improving Aggregate Recommendation Diversity Using Ranking-Based Techniques. *IEEE Transactions on Knowledge and Data Engineering* (2012).

[2] Massimo Caccia, Lucas Caccia, William Fedus, Hugo Larochelle, Joelle Pineau, and Laurent Charlin. 2020. Language GANs Falling Short. In *Proc. Int. Conf. Learning Representations (ICLR).*

[3] Yukuo Cen, Xu Zou, Jianwei Zhang, Hongxia Yang, Jingren Zhou, and Jie Tang. 2019. Representation Learning for Attributed Multiplex Heterogeneous Network. In *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining.*

[4] Michelle Keim Condli, David D Lewis, David Madigan, and Christian Posse. 1999. Bayesian mixed-E ects models for recommender systems. In *Int. ACM SIGIR Conf. on Research and Development in Information Retrieval.*

[5] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Proc. Adv. Neural Information Processing Systems.*

[6] Shaohua Fan, Junxiong Zhu, Xiaotian Han, Chuan Shi, Linmei Hu, Biyu Ma, and Yongliang Li. 2019. Metapath-guided Heterogeneous Graph Neural Network for Intent Recommendation. In *Proc. Int. Joint Conf. Artificial Intelligence.*

[7] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph Neural Networks for Social Recommendation. In *Proc. Int. Conf. World Wide Web.*

[8] Wenqi Fan, Yao Ma, Dawei Yin, Jianping Wang, Jiliang Tang, and Qing Li. 2019. Deep social collaborative filtering. In *Proc. ACM Conf. Recommender Systems.*

[9] Marco Gori and Augusto Pucci. 2007. ItemRank: A Random-Walk Based Scoring Algorithm for Recommender Engines. In *Proc. Int. Joint Conf. Artificial Intell.*

[10] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *Proc. Int. Joint Conf. Artificial Intelligence.*

[11] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proc. Adv. Neural Inf. Proc. Systems.*

[12] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proc. Int. Conf. World Wide Web.*

[13] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proc. Int. Conf. World Wide Web.*

[14] Antonio Hernando, Jesús Bobadilla, and Fernando Ortega. 2016. A non negative matrix factorization for collaborative filtering recommender systems based on a Bayesian probabilistic model. *Knowledge-Based Systems* (2016).

[15] Thomas Hofmann. 2004. Latent semantic models for collaborative filtering. *ACM Trans. Information System* (2004).

[16] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proc. Int. Conf. Learning Representations.*

[17] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining.*

[18] Yehuda Koren, Robert M. Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *IEEE Computer* (2009).

[19] Feng Li, Zhenrui Chen, Pengjie Wang, Yi Ren, Di Zhang, and Xiaoyu Zhu. 2019. Graph Intention Network for Click-through Rate Prediction in Sponsored Search. In *Proc. ACM Int. Conf. Research and Development in Information Retrieval.*

[20] Wenzhe Li, Sungjin Ahn, and Max Welling. 2016. Scalable MCMC for mixed membership stochastic blockmodels. In *Proc. Artificial Intelligence and Statistics.*

[21] Ramon Lopes, Renato Assunção, and Rodrygo LT Santos. 2016. Efficient Bayesian methods for graph-based recommendation. In *Proc. ACM Conf. Recommender Systems.*

[22] Sean M McNee, John Riedl, and Joseph A Konstan. 2006. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI'06 extended abstracts on Human factors in computing systems.*

[23] Soumyasundar Pal, Saber Malekmohammadi, Florence Regol, Yingxue Zhang, Yishi Xu, and Mark Coates. 2020. Non-Parametric Graph Learning for Bayesian Graph Neural Networks. *Proc. Conf. Uncertainty in Artificial Intelligence* (2020).

[24] Soumyasundar Pal, Florence Regol, and Mark Coates. 2019. Bayesian Graph Convolutional Neural Networks using Node Copying. *Proc. Learning and Reasoning with Graph-Structured Representations Workshop (ICML)* (2019).

[25] Steffen Rendle. 2010. Factorization Machines. In *Proc. IEEE Int. Conf. Data Mining.*

[26] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proc. Conf. Uncertainty in Artificial Intelligence.*

[27] Thiago Silveira, Min Zhang, Xiao Lin, Yiqun Liu, and Shaoping Ma. 2019. How good your recommender system is? A survey on evaluations in recommendation. *Int. Journal of Machine Learning and Cybernetics* (2019).

[28] Weiping Song, Zhiping Xiao, Yifan Wang, Laurent Charlin, Ming Zhang, and Jian Tang. 2019. Session-based Social Recommendation via Dynamic Graph Attention Networks. In *Proc. ACM Int. Conf. Web Search and Data Mining.*

[29] Jianing Sun, Yingxue Zhang, Chen Ma, Mark Coates, Huifeng Guo, Ruiming Tang, and Xiuqiang He. 2019. Multi-Graph Convolution Collaborative Filtering. In *IEEE Int. Conf. on Data Mining.*

[30] Louis C. Tiao, Pantelis Elinas, Harrison Nguyen, and Edwin V. Bonilla. 2019. Variational Spectral Graph Convolutional Networks. (2019).

[31] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2018. Graph Convolutional Matrix Completion. In *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (Deep Learning Day).*

[32] Saúl Vargas and Pablo Castells. 2011. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proc. ACM Conf. Recommender Systems.*

[33] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *Proc. Int. Conf. Learning Representations.*

[34] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural Graph Collaborative Filtering. In *Proc. ACM Int. Conf. Research and Development in Information Retrieval.*

[35] Le Wu, Peijie Sun, Yanjie Fu, Richang Hong, Xiting Wang, and Meng Wang. 2019. A neural influence diffusion model for social recommendation. In *Int. ACM SIGIR Conf. on Research and Development in Information Retrieval.*

[36] Qitian Wu, Hengrui Zhang, Xiaofeng Gao, Peng He, Paul Weng, Han Gao, and Guihai Chen. 2019. Dual Graph Attention Networks for Deep Latent Representation of Multifaceted Social Effects in Recommender Systems. In *Proc. Int. Conf. World Wide Web.*

[37] Jheng-Hong Yang, Chih-Ming Chen, Chuan-Ju Wang, and Ming-Feng Tsai. 2018. HOP-rec: high-order proximity for implicit recommendation. In *Proc ACM Conf. Recommender Systems.*

[38] 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 *Proc. ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining.*

[39] Yingxue Zhang, Soumyasundar Pal, Mark Coates, and Deniz Üstebay. 2019. Bayesian graph convolutional neural networks for semi-supervised classification. In *Proc. AAAI Int. Conf. Artificial Intelligence.*

9 SUPPLEMENTARY

9.1 Baselines

To demonstrate its effectiveness, we compare our proposed model with the following methods:

Classical collaborative filtering methods:

- **BPRMF** [26]: A general learning framework for personalized ranking recommendation using implicit feedback.
- **NeuMF** [13]: NeuMF replaces the inner product with an MLP to learn the user-item interaction function.

Graph neural network-based CF methods:

- **GC-MC** [31]: This is a graph-based auto-encoder approach that treats recommendation as a matrix completion problem.
- **PinSAGE** [38]: PinSAGE is a recent industry application of graph representation learning for recommendation. It deploys GraphSage [11] on an item-item graph with both image and text information as the input node features with mean aggregator. A hit rate improvement of more than 20% is reported in [38].
- **PinSAGE-LSTM**: Same overall architecture as PinSAGE but we replaced the mean aggregator with LSTM aggregation proposed in [11]. Although LSTM is an undesirable aggregator because it is not permutation invariant with respect to the ordering of neighborhood, it has better expressive capability and can model more complicated neighborhood relationships.
- **NGCF** [34]: NGCF is the state-of-the-art graph-based CF method. It explicitly integrates a bipartite graph structure into the embedding learning process to model the high-order connectivity in the user-item graph.

9.2 Industrial Application

9.2.1 System Panorama. In our implementation, the whole training procedure is composed of 3 components, namely Graph Construction, Graph Sampling and Network Training as shown in Algorithm 1. In Algorithm 1, $\mathcal{N}$ indicates the number of batches used to learn the representation of nodes in the constructed bipartite graph.

Algorithm 1: Overall training procedure

```
while not converge do
    Graph Construction with node copying;
    for i ∈ [0, N] do
        Graph Sampling for current batch i;
        Network Training for current batch i;
    end
end
```

Graph Construction: In the Graph Construction component, the bipartite graph is constructed based on the original user-item interaction records as well as the generated neighbors by the proposed node copying method. The nodes in the constructed bipartite graph contain user/item features.

Graph Sampling: In the Graph Sampling component, neighbor nodes are sampled (instead of considering all the neighbor nodes) to aggregate neighborhood information for a central node, in order to maintain training efficiency. A key difference between graph neural networks and traditional neural networks is that graph sampling is the efficiency bottleneck and also a critical factor affecting performance. Therefore we will elaborate on the details of the graph sampling methodology in Section 9.2.2.

Network Training: Each edge in the constructed bipartite graph is treated as a positive instance. For each such positive instance, several negative instances (each of which represents a user and an unobserved item) are sampled to conduct pair-wise training. Then the Network Training component performs the forward computing and backward updating.

9.2.2 Graph Sampling. As discussed in Section 2, random walk and its variants are widely used to retrieve the neighbor nodes of a given central node in GNN based algorithms. When we utilize Random Walk as the neighbor sampling method, two key factors should be considered. The first factor is the length of each random walk. If 2-hop neighbors of a central node are needed, then its 1-hop neighbors are sampled and afterwards 1-hop neighbors of such sampled neighbors of the target node are also sampled. That is to say, the length of a random walk determines the number of neighbours sampled in each random walk. The second factor is the number of random walks starting at a central node.

To improve the efficiency of graph sampling, we incorporate three practical techniques, namely *Layer-wise Uniqueness*, *Parallel Sampling* and *Asynchronous Sampling*.

9.2.3 Layer-wise Uniqueness. To reduce the number of random walks, we sample neighbors while enforcing layer-wise uniqueness. This means that when we sample the neighbors for a set of S nodes in the same layer, we first generate the set S_u of unique target nodes. The indices in the original array are recovered using the gather operation. The duplicate nodes share the same neighbors, therefore we only need to sample the neighbors of unique nodes.

By enforcing layer-wise uniqueness we can reduce the number of random walks dramatically and therefore improve the training efficiency. However, as it eliminates duplicate neighbors at each layer, it distorts the neighborhood distributions, which may degrade the accuracy of the algorithms. Therefore there is a trade-off between accuracy and efficiency.

9.2.4 Parallel sampling. When performing neighbor sampling for each layer, the random walks are independent. Therefore, we can perform parallel random walks with multi-processing and multi-threading, to accelerate the sampling procedure for one batch.

9.2.5 Asynchronous Sampling. As Graph Sampling and Network Training are disjoint across different batches, they can be scheduled sequentially or in parallel. In sequential scheduling, Graph Sampling and Network Training are processed sequentially, i.e., Network Training starts when the neighbor nodes of the current batch are sampled, and the Graph Sampling for the next batch is not started until the Network Training of the current batch finishes. Obviously, this is inefficient. We create multiple threads to perform Graph Sampling in parallel to Network Training, so that the network keeps training without being interrupted by the sampling procedure.