Adaptive Popularity Debiasing Aggregator for Graph Collaborative Filtering

Huachi Zhou
huachi.zhou@connect.polyu.hk
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong

Hao Chen*
sundaychenhao@gmail.com
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong

Junnan Dong
hanson.dong@connect.polyu.hk
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong

Daochen Zha
daochen.zha@rice.edu
Rice University
Houston, Texas, America

Chuang Zhou
chuang-qqzj.zhou@connect.polyu.hk
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong

Xiao Huang*
xiaohuang@comp.polyu.edu.hk
The Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong

ABSTRACT

The graph neural network-based collaborative filtering (CF) models user-item interactions as a bipartite graph and performs iterative aggregation to enhance performance. Unfortunately, the aggregation process may amplify the popularity bias, which impedes user engagement with niche (unpopular) items. While some efforts have studied the popularity bias in CF, they often focus on modifying loss functions, which can not fully address the popularity bias in GNN-based CF models. This is because the debiasing loss can be falsely backpropagated to non-target nodes during the backward pass of the aggregation.

In this work, we study whether we can fundamentally neutralize the popularity bias in the aggregation process of GNN-based CF models. This is challenging because 1) estimating the effect of popularity is difficult due to the varied popularity caused by the aggregation from high-order neighbors, and 2) it is hard to train learnable popularity debiasing aggregation functions because of data sparsity. To this end, we theoretically analyze the cause of popularity bias and propose a quantitative metric, named *inverse popularity score*, to measure the effect of popularity in the representation space. Based on it, a novel graph aggregator named APDA is proposed to learn per-edge weight to neutralize popularity bias in aggregation. We further strengthen the debiasing effect with a weight scaling mechanism and residual connections. We apply APDA to two backbones and conduct extensive experiments on three real-world datasets. The results show that APDA significantly outperforms the state-of-the-art baselines in terms of recommendation performance and popularity debiasing.

SIGIR '23, July 23–27, 2023, Taipei, Taiwan

ACM ISBN 978-1-4503-9408-6/23/07…$15.00
https://doi.org/10.1145/3539618.3591635

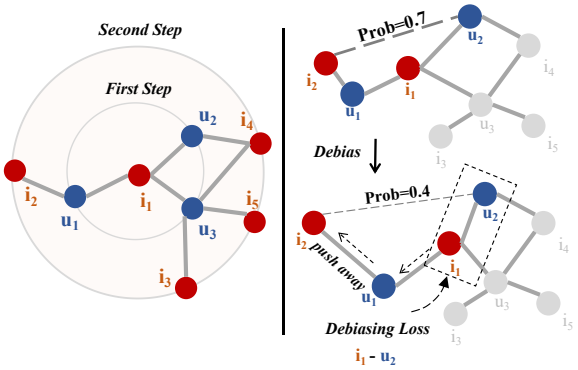

Figure 1: Left: An illustrative example of popularity bias. The popular item i_1 aggregates much more nodes in two propagation steps than the niche item i_2, which results in popularity bias. Right: Cascading effect in popularity debiasing. Naively applying a debiasing loss to i_1 and u_2 could also undesirably reduce the interaction probability between i_2 and u_2 since the loss will be backpropagated to i_2 in the backward pass of aggregation to push i_2 away from u_2 as well.

CCS CONCEPTS

• **Information systems** → Recommender systems.

KEYWORDS

collaborative filtering; graph neural networks; popularity bias

ACM Reference Format:
Huachi Zhou, Hao Chen*, Junnan Dong, Daochen Zha, Chuang Zhou, and Xiao Huang*. 2023. Adaptive Popularity Debiasing Aggregator for Graph Collaborative Filtering. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '23), July 23–27, 2023, Taipei, Taiwan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3539618.3591635

1 INTRODUCTION

Recommender systems play an indispensable role in alleviating information overload by suggesting a small set of items to users.

This is typically achieved by modeling users' preferences based on their historical user-item interactions, known as collaborative filtering (CF) [16, 21, 29]. Among many CF techniques, graph neural networks (GNNs) have recently received increasing attention since many recommendation tasks can be naturally modeled as graphs. These methods often align user-item interactions with a bipartite graph and exploit high-order connectivity information by aggregating user and item embeddings [41, 52]. The aggregation operation enables GNN-based CF to learn high-quality user and item representations and achieve state-of-the-art performances [13, 38].

Despite the encouraging results of GNN-based CF, the aggregation of GNNs may lead to significant popularity bias. In a typical recommendation dataset, a small portion of items collects the majority of the user feedback (popular items), while the rest receive little feedback (niche items). This long-tailed distribution leads to a large difference in the number of edges connected to items. For example, in the left-hand side of Figure 1, popular item nodes (such as node i_1) aggregate much more users/items than niche items (such as i_2). It is well known that the aggregation makes the node representations similar to each other [2]. As a result, those popular items will be similar to more users in the representation space, and thus they become the "hub" and tend to have higher rating scores for the majority of users. The skewed distribution of rating scores could, in turn, amplify the popularity bias since it may impede future user engagement with niche items.

To tackle the popularity bias in CF, the existing techniques often modify loss functions directly or indirectly and can be mainly divided into three categories [6]. *(1) Regularization:* these methods either penalize the correlation between rating scores and item popularity [35, 54] or apply orthogonal regularization to encourage user/item embedding irrelevant with each other [8, 39]. *(2) Causal graph:* they build a causal graph to analyze the popularity bias and then eliminate the bias in user-item relevance score calculation (indirectly) [32, 36]. *(3) Inverse propensity score:* they upweight the importance of the ranking loss for the user-niche item pair based on inverse propensity score [18, 40].

We argue, however, that the loss-based debiasing methods can not fully address the popularity bias during the GNNs' aggregation process. This is because the debiasing loss can be falsely back-propagated to non-target nodes during the backward pass of the aggregation, causing a cascading effect, which, on the contrary, may even reduce the recommendation opportunities for niche items. For example, on the right of Figure 1, if we apply a debiasing loss to the i_1-u_2 pair to decrease their similarity, the representation of i_2 will be undesirably pushed away from u_2 as well due to the aggregation. In this case, the popularity bias of i_1 could be beneficial since it helps the niche item i_2 win more recommendation opportunities. The above analysis motivates us to mitigate the popularity bias from the aggregation perspective rather than loss functions. Specifically, we ask: **can we fundamentally neutralize the popularity bias in the aggregation process of GNN-based CF models?**

It is challenging to achieve this goal due to the following reasons. **First**, before we can mitigate the popularity bias, we need to accurately estimate the effect of popularity in the representation space; however, estimating its effect is difficult due to the varied popularity of the aggregation of high-order neighbors. In a single aggregation step, even if two items have the same extent of popularity before aggregation, their aggregated representations could have quite different extents of popularity since they are influenced by their neighbors. The estimation becomes even more complicated if we consider multiple aggregation layers (the popularity effect for an item may differ in each layer) and the fact that the representations keep changing in the optimization process. **Second**, even if we can accurately estimate the effect of popularity, it remains a challenge to design a learnable aggregator to mitigate the bias. Due to data sparsity in recommendation data [37], it is shown that an over-complicated aggregator will lead to overfitting and degrade the performance [15]. As a result, LightGCN [15] and its variants often exclude feature transformation and non-linear activation functions. It is hard to introduce a debiasing mechanism into the aggregator without increasing the risk of overfitting.

To this end, this work aims to systematically analyze the cause of popularity bias in GNN-based CF and investigate the mitigation methods. In particular, we aim to answer the following questions: (1) How to estimate the effect of popularity given the complex aggregation of higher-order neighbors? (2) How to effectively mitigate the popularity bias in the aggregation process? By answering these questions, we make the following contributions.

- Through theoretically analyzing the gradient magnitudes, we show insights into why popular items suffer from popularity bias. Based on our theoretical findings, we propose a quantitative metric, named inverse popularity score, to measure the effect of popularity in the representation space.
- We present a simple yet effective graph aggregator, dubbed APDA, which adaptively learns per-edge weights to neutralize popularity bias in aggregation. We further propose a weight scaling strategy and use residual connections to strengthen the debiasing effect.
- We apply APDA to two representative backbones, LightGCN [15] and LR-GCCF [7]. Extensive experiments on three real-world datasets show that, compared with the state-of-the-art debiasing methods, APDA significantly reduces the popularity bias while retaining its ranking performance.

2 PRELIMINARY

We first define the notations used in this paper. Then we introduce LightGCN [15], one of the most popular GNN-based CF models.

Notations. We represent the historical interactions in recommender systems as a user-item bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the union of m user nodes $\mathcal{U} = \{v_1, v_2, \ldots, v_m\}$ and n item nodes $\mathcal{I} = \{v_{m+1}, v_{m+2}, \ldots, v_{m+n}\}$, and the edge set $\mathcal{E}$ denotes the observed node connections. Let $\mathbf{A} \in \mathbb{R}^{(n+m)\times(n+m)}$ denote the binary adjacency matrix of graph $\mathcal{G}$, where $\mathbf{A}_{u,i} = 1$ when there is an interaction between user node u and item node i, and $\mathbf{A}_{u,i} = 0$ otherwise. We denote d_u as the degree of the user node u, which is the summation of all the elements in row $\mathbf{A}_u$. A trainable embedding lookup table $\mathbf{E}^{(0)} \in \mathbb{R}^{(n+m)\times t}$, where t is embedding dimension, maps user u and item i from one-hot encoding to dense vectors $\mathbf{e}_u^{(0)}$ and $\mathbf{e}_i^{(0)}$, respectively. Here, the superscript $^{(0)}$ denotes the initial embedding. Given a target user, the goal is to recommend top-N unconnected items that are likely to be clicked by the user.

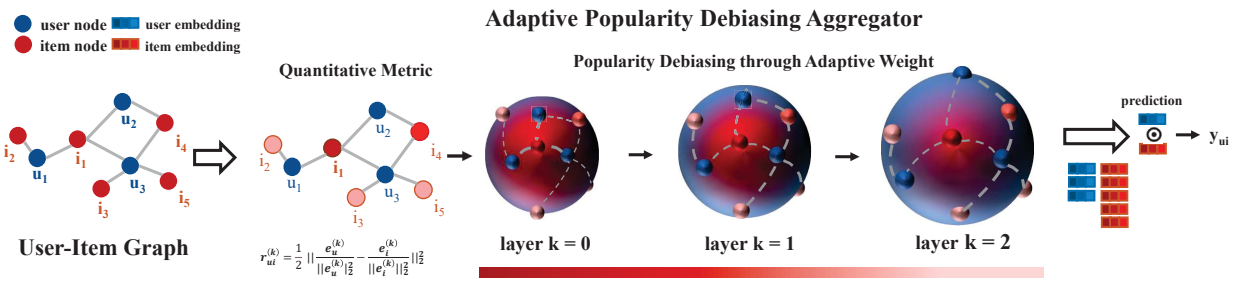

Figure 2: A sketch illustration of APDA. Since we normalize the representations, we represent the user/item vector in a two-dimensional space unit sphere. As more neighbors are aggregated, the sphere volume becomes larger. The shade of the color represents the popularity effect on item representations. The popularity effect on the central item node declines since the connected edges are assigned lower weights.

LightGCN. As one of the first GNN-based CF methods, LightGCN learns user/item representations following the general message passing of GNNs. Specifically, the representations at the k-th propagation layer are obtained by

$$\mathbf{E}^{(k)} = H(\mathbf{E}^{(k-1)}, \mathcal{G}), \tag{1}$$

where H is the aggregation function. When $k = 1$, the input is the initial embedding table $\mathbf{E}^{(0)}$. It is shown that feature transformation and non-linear activation from the original designs of GCNs may increase the risk of overfitting and do not contribute to the performance in recommendation tasks [15]. Thus, LightGCN and its variants use linear embedding propagation, i.e., a simple weighted sum as the aggregation function:

$$\mathbf{E}^{(k)} = \widetilde{\mathbf{A}}\mathbf{E}^{(k-1)}, \tag{2}$$

where $\widetilde{\mathbf{A}} = (\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})$ and $\mathbf{D}$ is the diagonal node degree matrix. To learn user and item embeddings, LightGCN adopts the pairwise Bayesian personalized ranking (BPR) loss [27] to maximize the rating score difference between positive and negative samples with an L_2 regularization term to avoid overfitting:

$$\mathcal{L}_{BPR} = \sum_{u,i\in\mathcal{N}(u),j\notin\mathcal{N}(u)} -\ln\sigma(y_{ui} - y_{uj}) + \gamma\|\Theta\|_2^2, \tag{3}$$

where γ is a hyperparameter controlling the effect of L_2 penalty; $\mathcal{N}(u)$ is the item set that user u has interacted with in the history; y_{ui} denotes the inner product of the final representations of user u and item i, indicating their relevance score; σ denotes the sigmoid activation function; Θ are the embeddings of the user and items in the current batch.

To facilitate understanding, we describe the proposed APDA mainly with LightGCN as the backbone. Note that APDA is general and can be applied to other backbones as well. In the experiments, we have considered another backbone, LR-GCCF [7].

3 METHODOLOGY

We start this section by theoretically analyzing the effect of popularity in GNN-based CF methods and introducing a quantitative metric to measure the effect of popularity (Section 3.1). Then we present a simple yet effective aggregator, APDA, to mitigate popularity bias by adaptively adjusting the per-edge weight in aggregation (Section 3.2). Finally, we show how APDA can be applied to GNN-based CF models using LightGCN [15] as an example.

3.1 Estimating the Effect of Popularity

In what follows, we theoretically analyze how item popularity (i.e., the normalized number of item clicks) influences representation learning. Following the analysis, we propose *inverse popularity score*, a quantitative metric to measure the effect of popularity based on user and item representations. Further, we empirically study the relationship between inverse popularity score and item popularity.

3.1.1 Effect of Popularity. Intuitively, the inner product between the user and item representations reflects the effect of popularity since we often use the inner product to rank items. The BPR training objective increases the inner products for the positive user-item pairs while decreasing those of negative pairs. To understand the effect of popularity, we draw a connection between the extent to which a user-item pair is being pushed in training and node degrees with the theorem below. We provide its proof in Appendix A.1.

Theorem 3.1 (Effect of Popularity). *Given a self-looped connected graph $\mathcal{G}$ with sufficiently large embedding dimension t, assume that we use infinite linear embedding propagation layers with $\widetilde{\mathbf{A}}$ to train user and item embedding in a full-batch manner. For layer k, the gradient magnitude of BPR loss w.r.t. $\mathbf{e}_u^{(k)\top}\mathbf{e}_i^{(k)}$ satisfies the following:* $|\frac{\partial L_{BPR}}{\partial(\mathbf{e}_u^{(k)\top}\mathbf{e}_i^{(k)})}| \propto \sum_{m\in O^{(k)}(u),n\in O^{(k)}(i)} \widetilde{\mathbf{A}}^{(k)}_{mn}\sqrt{(d_m+1)(d_n+1)}$, *where $O^{(k)}(u)$ represents the neighbor nodes within k hops of node u; $\widetilde{\mathbf{A}}^{(k)}$ represents repeated multiplication of $\widetilde{\mathbf{A}}$ for k times, and $\widetilde{\mathbf{A}}^{(k)}_{mn}$ is the element in the row m and column n of $\widetilde{\mathbf{A}}^{(k)}$.*

Theorem 3.1 implies that the speed of item representations moving towards the user representations is positively correlated to the degrees of the aggregated nodes in the surrounding subgraphs.

Remark 1 (Hub Effect). *The effect of popularity may further lead to a hub effect. In the optimization, the positive user-item pair may drag the embeddings of the nodes in the surrounding subgraph closer to each other. Since popular items appear more frequently in the subgraph, they will be pushed closer to other nodes. As a consequence, they serve as the hubs in the subgraph.*

Remark 2 (Symmetric Normalization). *Although the symmetric normalization in the standard graph aggregation can mitigate the impact of node degree during propagation to some extent, it can not prevent the popular items from obtaining large inner products. This is because popular items have dense edges connected to a larger number of neighbors. The weighted sum of aggregated node degrees is still large, so it will also enlarge the inner product.*

3.1.2 Inverse Popularity Score. Based on Theorem 3.1, we estimate the effect of popularity based on the inner products of the item and user representations. Specifically, we propose the following metric.

Definition 3.2 (Inverse Popularity Score). Given user and item embeddings learned at k-th layer $e_u^{(k)}$ and $e_i^{(k)}$, the inverse popularity score $r_{ui}^{(k)}$ is defined as:

$$r_{ui}^{(k)} = 1 - \frac{\mathbf{e}_u^{(k)\top}\mathbf{e}_i^{(k)}}{\|\mathbf{e}_u^{(k)}\|_2^2\|\mathbf{e}_i^{(k)}\|_2^2}. \tag{4}$$

Remark 3 (Normalized User-item Distance). *The inverse popularity score can also be interpreted as the normalized distance between the user and item representations. Specifically, we can rewrite* $r_{ui}^{(k)}$ *as follows:* $r_{ui}^{(k)} = 1 - \frac{e_u^{(k)\top}e_i^{(k)}}{\|e_u^{(k)}\|_2^2\|e_i^{(k)}\|_2^2} = \frac{1}{2}\left(\frac{e_u^{(k)\top}e_u^{(k)}}{\|e_u^{(k)}\|_2^2} + \frac{e_i^{(k)\top}e_i^{(k)}}{\|e_i^{(k)}\|_2^2} - 2 * \frac{e_u^{(k)\top}e_i^{(k)}}{\|e_u^{(k)}\|_2^2\|e_i^{(k)}\|_2^2}\right) = \frac{1}{2}\left\|\frac{e_u^{(k)}}{\|e_u^{(k)}\|_2^2} - \frac{e_i^{(k)}}{\|e_i^{(k)}\|_2^2}\right\|_2^2$.

Note that rather than directly using the inner product as the score, we use cosine similarity, which introduces an L_2 normalization to eliminate the impact of representation magnitude, i.e., the denominator $\|\mathbf{e}_u^{(k)}\|_2^2\|\mathbf{e}_i^{(k)}\|_2^2$. This is because real-world user-item graphs may not satisfy the connected graph assumption in Theorem 3.1. Geometrically, the inner product is jointly determined by the euclidean magnitude and cosine similarity of the two representations. However, when using the BPR loss for training, the user or item representations may have a growing euclidean magnitude. In this case, the inner product may not well represent the effect of popularity but rather be dominated by the large representation magnitude. In an extreme case, the following proposition suggests that the unbounded representation magnitude could go to infinity.

Proposition 3.3. *Assume that graph $\mathcal{G}$ contains two components, where the small component has one (niche) user and one item node while other nodes reside in the other component. The optimal representation magnitude of the item in the small component is infinite.*

The L_2 normalization restricts user/item node representations in a unit sphere and bounds the score in the range of $[-1, 1]$. Finally, we use one subtracted by the cosine similarity as the final score. Intuitively, the niche items tend to have higher inverse popularity scores. The score implies how much we should promote an item.

The benefits of estimating the popularity effect based on representations are twofold. **First**, the inverse popularity scores are adaptively adjusted based on the user and item representations. Thus, the scores can dynamically fit the representations in each aggregation layer throughout the training process. **Second**, this metric can be generally applied to all the GNN-based CF models.

3.1.3 Relationship between Inverse Popularity Score and Item Popularity. We analyze whether the proposed inverse popularity score aligns with the actual popularity of the items. Specifically, we retrieve the trained item embeddings from LightGCN and rank them by the ascending order of the normalized number of clicks (which is defined as the total number of clicks of the item divided by the maximum total number of clicks across all items in the training data). Then we split the items evenly, where each group contains 5% of all the items. The inverse popularity score for each item is computed as the average of all user embeddings. We average the inverse popularity scores and the normalized number of clicks for all items in each group. The average inverse popularity scores of each group on three datasets are scaled using min-max scaling to fit within the interval $[0, 1]$. Their relationship is plotted in Figure 3. We observe that the inverse popularity aligns with item popularity (i.e., the normalized number of clicks), which verifies that the proposed metric can accurately estimate the effect of popularity.

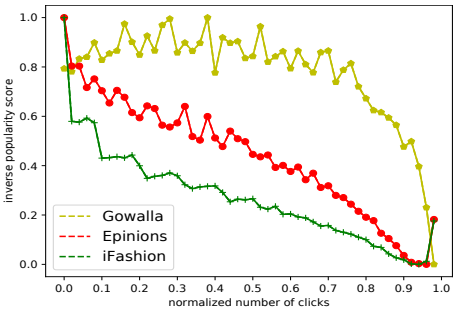

Figure 3: Inverse popularity score versus item popularity (normalized number of clicks). We rank items by the ascending order of the normalized number of clicks and then split them into 20 equal-size groups, where each split has 5% of the items. We plot the average of the inverse popularity score and the normalized number of clicks for each group.

Note that, while the normalized number of clicks can also imply the popularity effect [20, 50], it is a fixed value, so it may only be applicable to trained embeddings. Whereas the proposed metric is adaptively adjusted based on the current embeddings, and thus it can estimate the popularity effect throughout the training process.

3.2 Adaptive Popularity Debiasing Aggregator

Figure 2 shows an overview of APDA. We will first introduce how to enable adaptive per-edge weight based on the proposed inverse popularity score. Then we discuss how weight scaling and residual connections can strengthen the debiasing effect.

3.2.1 Adaptive Weight. The key idea is to reweigh the neighboring nodes during the aggregation process to neutralize the popularity bias. Specifically, at the k-th propagation step, we calculate the inverse popularity scores defined in Eq. (4) using the embeddings in the current layer. Then we assign the inverse popularity scores to all the user-item pairs as the weights.

We discuss the effect of this weighting mechanism on popular and niche items below. **On the one hand**, popular items generally get assigned smaller inverse popularity scores, which reduce the aggregation weights with their k hop neighbors. As a result, the popular items aggregate less information from the high-order

neighbors, which prevents them from obtaining excessive similarity with other nodes. **On the other hand**, niche items are assigned larger inverse popularity scores, which prioritize their message passing to other nodes and help them win more recommendation opportunities. Putting the above two effects together, the adaptive weights can penalize the popular items and promote the niche items, which neutralizes the popularity bias in aggregation.

3.2.2 Weight Scaling. When the item popularity difference of neighbors is not large, the inverse popularity score may not be differentiated enough to mitigate the popularity bias. To tackle this problem, we introduce a weight scaling mechanism to amplify weight differences. Specifically, we add exponential function and symmetric normalization term to further hinder popular item message propagation. The scaled weight for a user-item pair (u, i) in the k-th propagation step is given as

$$w_{iu}^{(k)} = w_{ui}^{(k)} = \frac{\exp(r_{ui}^{(k)})}{\sqrt{d_u d_i}}, \tag{5}$$

where $\exp(\cdot)$ denotes exponential function. Eq. (5) applies to not only item nodes but also user nodes since user representations may also suffer from popularity bias due to aggregation. With the scaled weights, the aggregator aggregates neighbor messages with

$$\mathbf{e}_u^{(k+1)} = \sum_{i \in N(u)} w_{ui}^{(k)} \hat{\mathbf{e}}_i^{(k)}, \tag{6}$$

$$\mathbf{e}_i^{(k+1)} = \sum_{u \in N(i)} w_{iu}^{(k)} \hat{\mathbf{e}}_u^{(k)}. \tag{7}$$

3.2.3 Residual Connection. The over-smoothing issue of GNNs [2] can make all representations similar to each other, which will reduce the sensitivity of the proposed metric and make our debiasing strategy less effective. This is because stacking multiple GNN layers will smooth the locality feature, i.e., users/items residing in the same subgraph. This long-range dependency modeling may lead to peripheral nodes being similar. The indistinguishable representations will undermine the metric sensitivity since the similar representation of proximal nodes narrows the range of values for the inverse popularity score. In an extreme case, if the representations of all nodes become exactly the same, the inverse popularity score will collapse to uniform distribution and degrade to the original aggregation function. In this case, the proposed aggregator will not be able to debias based on inverse popularity scores.

To tackle this problem, we add residual connections to enforce the embeddings not to be closely similar as follows:

$$\hat{\mathbf{e}}_u^{(k)} = \mathbf{e}_u^{(k)} + \lambda \mathbf{e}_u^{(0)}, \tag{8}$$

$$\hat{\mathbf{e}}_i^{(k)} = \mathbf{e}_i^{(k)} + \lambda \mathbf{e}_i^{(0)}. \tag{9}$$

We have added an initial embedding to the current step with a hyperparameter λ to balance the importance of self-information and neighbor representations. This design helps maintain the sensitivity of the metric.

3.3 Application to LightGCN

This subsection shows how to apply APDA to the existing GNN-based CF models. We use LightGCN [15] as an example.

For ease of implementation, we can rewrite the aggregator as a matrix form:

$$\mathbf{E}^{(k+1)} = \mathbf{C}\widetilde{\mathbf{A}}(\mathbf{E}^{(k)} + \lambda \mathbf{E}^{(0)}), \tag{10}$$

where $\mathbf{C} = \exp(1 - L_2(\cdot))$ is the reweighting function; $L_2(\cdot)$ represents row-wise L_2 normalization function. After k layers propagation, each user/item obtains k embedding outputs. Based on LightGCN, we encode the structural information and adopt a mean pooling function to combine the information into the final representations: $\mathbf{e}_u = \frac{1}{k+1}\sum_{q=0}^{k} \mathbf{e}_u^{(q)}$, $\mathbf{e}_i = \frac{1}{k+1}\sum_{q=0}^{k} \mathbf{e}_i^{(q)}$. The final representations contain different orders of connectivity. We compute their inner products of them to obtain the final interaction probability for each user-item pair $y_{ui} = \mathbf{e}_u^\top \mathbf{e}_i$.

4 EXPERIMENTS

In this section, we conduct extensive experiments to verify the effectiveness of APDA and understand how it behaves. We aim to answer the following research questions:

- **RQ1:** How effective is APDA compared with the existing popularity debiasing methods on different backbones in terms of accuracy and popularity debiasing?
- **RQ2:** What are the contributions of different components of APDA to the overall model performance?
- **RQ3:** How much performance improvement do popular and niche items gain compared with popularity debiasing baselines over the backbone, and does performance improvement on either one come at the expense of sacrificing the other?
- **RQ4:** What are the impacts of different hyper-parameters on model performance?

Table 1: Detailed datasets statistics.

Datasets	User	Item	Interaction	Density
Gowalla	29,858	40,981	1,027,370	0.00084
iFashion	23,405	24,803	378,713	0.00065
Epinions	11,496	11,656	327,942	0.00245

4.1 Datasets

Three public real-world datasets are processed to compare the performances of all the methods: Gowalla, iFashion, and Epinions. **Gowalla**[1] is collected from a social networking website where users share their daily check-in locations. **iFashion**[2] is an industrial dataset released by Alibaba, a famous e-commercial website. **Epinions**[3] comes from a product review website.

We follow previous experimental setting [33, 50] and only reserve users and items who have at least ten interactions. For each user, 70% interacted items are randomly sampled as the training set, and another 10% is used as the validation set to tune hyperparameters. The remaining 20 % of interactions are used as the test set. The detailed data statistics are summarized in Table 1.

[1] https://snap.stanford.edu/data/loc-gowalla.html
[2] https://github.com/wenyuer/POG
[3] http://www.trustlet.org/downloaded_epinions.html

Table 2: Performances of APDA variants and baselines using the LightGCN backbone, where ↑ means the higher the better, while ↓ means the lower the better. The best performance is marked in bold, and the second-best performance is underlined. N.A. suggests that the model does not learn well and only achieves a marginal improvement over an untrained model.

Methods	Gowalla			iFashion			Epinions		
	Recall@20↑	NDCG@20↑	PRU@20↓	Recall@20↑	NDCG@20↑	PRU@20↓	Recall@20↑	NDCG@20↑	PRU@20↓
LightGCN	0.1810	0.1529	0.4960	0.0779	0.0478	0.3096	0.1085	0.0790	0.3968
PC_LightGCN	0.1830	0.1559	0.4804	0.0762	0.0469	0.3045	0.1082	0.0790	0.3902
IPS_LightGCN	0.1795	0.1513	0.4895	0.0851	0.0528	0.2975	0.0990	0.0725	0.3020
Reg_LightGCN	0.1809	0.1538	0.4892	0.0752	0.0459	0.2835	0.1087	0.0798	0.3510
DICE_LightGCN	0.1743	0.1478	0.4780	0.0821	0.0510	0.2723	0.1138	0.0832	0.3156
MACR_LightGCN	0.1749	0.1486	0.4882	0.0615	0.0376	0.2835	0.0873	0.0638	0.3657
AdjNorm_LightGCN	0.1795	0.1527	0.4841	0.0781	0.0479	0.2850	0.1094	0.0803	0.3205
SimGCL	0.1790	0.1483	0.4891	0.0929	0.0582	0.3430	0.1189	0.0867	0.3639
GTN	0.1875	0.1573	0.4981	0.0941	0.0590	0.2931	0.1239	0.0914	0.3286
Improvement(%)	+1.68%	+2.72%	+6.78%	+6.06%	+5.59%	+15.53%	+1.78%	+1.42%	+14.11%
APDA_LightGCN	**0.1907**	**0.1617**	**0.4456**	**0.0998**	**0.0623**	**0.2300**	**0.1261**	**0.0927**	**0.2594**
APDA w/o-AW	0.1886	0.1596	0.4507	0.0612	0.0377	0.2786	0.1192	0.0874	0.3643
APDA w/o-WS	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	0.0793	0.0540	0.4086
APDA w/o-RC	0.1904	0.1613	0.4479	0.0979	0.0612	0.2408	0.1214	0.0886	0.2902

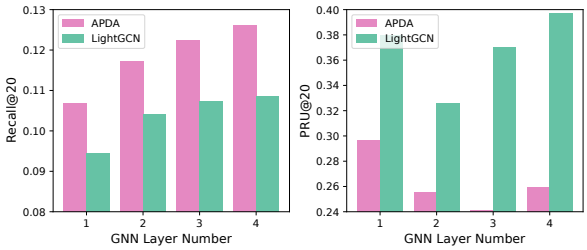

Figure 4: The effect of the number of layers on APDA and LightGCN.

4.1.1 Evaluation Metrics. To estimate model performance w.r.t. top-N recommendation, we adopt two widely used metrics: Recall@N and NDCG@N. To investigate the extent of popularity bias suffered by models, we adopt PRU@N to calculate the correlation between item rank and popularity following [39, 50, 54], defined by:

$$PRU@N = -\frac{1}{|\mathcal{U}|}\sum_{u\in\mathcal{U}} SRC(pop(i), rank_{ui}), \tag{11}$$

where $SRC(\cdot,\cdot)$ calculates Spearman's rank correlation between two lists; $pop(\cdot)$ returns the number of item clicks in the training set and $rank_{ui}$ represents item order in the recommendation list.

4.1.2 Competitors. We incorporate the following state-of-the-art baselines in our comparison. The baselines belong to three groups: i) **Backbone**: (1) LR-GCCF [7] excludes nonlinear activation unit and adds residual network to the original structure. (2) LightGCN [15] removes burdensome activation unit and enhances model generalization ability. ii) **Popularity debiasing**: (3) Adjnorm [50] increases the power of normalization term in graph aggregation. (4) PC [54] post-processes the prediction results to compensate for niche items being recommended. (5) Reg [54] regularizes the correlation between prediction results and item popularity in training. (6) MACR [36] performs counterfactual reasoning on the causal graph to mitigate the popularity bias. (7) DICE [53] disentangles the popularity and conformity from the user and item embeddings. (8) IPS [18, 40] lowers the weight of the loss for popular items. Notably, our model only uses user-item interactions without side information, e.g., timestamp. We leave out the comparison with other popularity debasing models such as PD [48], CD^{2}AN [8], DecRs [32]. iii) **Graph-based Collaborative Filtering**: SimCGL [42] is state-of-the-art graph contrastive learning method. It adds uniform noise to node representations when generating different views. GTN [11] adds L_1 normalization and L_2 normalization to the propagation and shrinks the coefficients of unimportant features to 0. Remarkably, we do not introduce more methods since we aim to mitigate the impact of the popularity of items instead of solely pursuing state-of-the-art performance.

Besides, to demonstrate the effectiveness of different components of APDA under the LightGCN backbone, we design three variants. (1) **APDA w/o-AW** removes adaptive weight to verify the effectiveness of constructing adaptive messages in GNN popularity debiasing. (2) **APDA w/o-WS** drops weight scaling to show the necessity of its success in controlling the impact of node degree on model performance. (3) **APDA w/o-RC** exclude the residual connection to demonstrate its success in improving the whole model performance and debiasing results.

4.1.3 ***Hyper-parameter Settings.*** We implement APDA with Pytorch and optimize it with Adam optimizer. We apply a grid search strategy to search optimal hyperparameters based on the performance on the validation set. The early stopping strategy is applied, and the threshold is fixed as 50, i.e., no more Recall@20 score improvement for 50 consecutive epochs [33]. This setting means we study the popularity bias when the model reaches the best recommendation performance. We tune the residual connection coefficient λ from $\{0, 0.1, \cdots, 1.0\}$. The learning rate is set as 0.001, and the weight decay equals $1e-4$. The batch size is set as 2048 for all baselines, and the baseline codes released by the original authors are utilized to tune hyperparameters, except for the default value

Table 3: Overall comparison with popularity debiasing baselines under the LR-GCCF backbone.

Methods	Gowalla			iFashion			Epinions		
	Recall@20↑	NDCG@20↑	PRU@20↓	Recall@20↑	NDCG@20↑	PRU@20↓	Recall@20↑	NDCG@20↑	PRU@20↓
LR-GCCF	0.1540	0.1301	0.6068	0.0915	0.0562	0.3482	0.1163	0.0837	0.4195
PC_LR-GCCF	0.1516	0.1289	0.5408	0.0910	0.0559	0.3398	0.1172	0.0844	0.4113
IPS_LR-GCCF	0.1480	0.1252	0.5976	0.0634	0.0376	0.3330	0.1074	0.0779	0.3796
Reg_LR-GCCF	0.1508	0.1282	0.6025	0.0920	0.0569	0.3458	0.1179	0.0850	0.4198
DICE_LR-GCCF	0.1552	0.1293	0.4210	0.0619	0.0377	0.2995	0.0940	0.0687	0.3502
MACR_LR-GCCF	0.1466	0.1246	0.4316	0.0818	0.0495	0.2851	0.1046	0.0756	0.3112
AdjNorm_LR-GCCF	0.1537	0.1307	0.5960	0.0923	0.0571	0.3306	0.1168	0.0843	0.4032
Improvement	+9.66%	+3.90%	+22.47%	+1.91%	+3.33%	+18.48%	+1.36%	+3.06%	+7.52%
APDA_LR-GCCF	**0.1702**	**0.1358**	**0.3264**	**0.0941**	**0.0590**	**0.2324**	**0.1195**	**0.0876**	**0.2878**

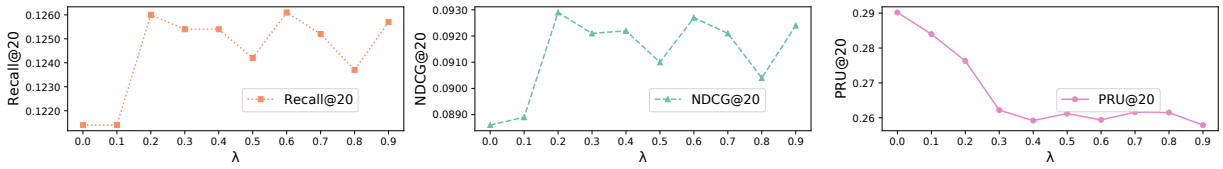

Figure 5: The effect of λ magnitude on APDA

and test performance. All models are run repeatedly five times, and we report the average results.

4.2 Overall Performance (RQ1)

To answer the first question, we compare APDA with all the baselines. Table 2 and 3 summarize the performances of all the models in terms of Recall@20, NDCG@20, and PRU@20 scores on three real-world datasets on LightGCN and LR-GCCF backbones, respectively. We have the following observations.

First, directly or indirectly modifying the debiasing loss mitigates the popularity bias to some extent but sacrifices the recommendation performance at the same time, including PC, IPS, Reg, DICE, and MACR. These methods achieve better performance in the PRU scores than the backbone, showing that the loss function can somewhat alleviate the bias. However, since they do not consider the key aggregation process, the improvement is marginal. Moreover, all these methods suffer from a drop in recommendation performance. The potential reason is that the debiasing loss prevents popular node representations from being too similar to other nodes while the effect is also propagated indiscriminately to non-target neighbor users/items in the optimization.

Second, debiasing the aggregation generally outperforms debiasing the loss in terms of recommendation performance and bias mitigation. AdjNorm ignores that representation is dynamically updated in the optimization and only uses the number of clicks to predefine edge weight. The results show that predefined weights are almost incapable of assigning appropriate static weights to further suppress the bias than APDA.

Third, improving recommendation performance does not necessarily bring gains in mitigating popularity bias. The SimGCL and GTN are state-of-the-art GNN-based CF models. The results demonstrate that they only slightly mitigate the bias and even amplify the bias in some cases.

Fourth, APDA boosts the model performance on two backbones. APDA differs from other popularity debiasing baselines in that it considers the unique aggregation mechanism of GNN and performs adaptive message propagation to mitigate the bias. The APDA relatively achieves +1.68%, +2.72%, +6.78%, +6.06%, +5.59%, +15.53%, +1.78%, +1.42%, +14.11% over the strongest baselines in three real-world datasets with respect to Recall@20, NDCG@20, PRU@20 scores under the LightGCN backbone. It demonstrates the effectiveness of APDA and the rationality of model design.

4.3 Ablation Study (RQ2)

To investigate whether different model components contribute to the whole model performance, we analyze three variants, i.e, APDA w/o-AW, APDA w/o-WS, and APDA w/o-RC and show the results in Table 3. APDA w/o-AW only uniformly aggregates neighbor nodes and does not learn attention in distinguishing niche neighbors. Therefore, it weakens the popularity bias to a lesser extent than APDA. It proves the effectiveness of adaptive weight in mitigating the popularity bias. APDA w/o-WS removes the weight scaling component and performs worse among all variants. It demonstrates the necessity of weight scaling towards further improving performance and popularity debiasing. APDA w/o-RC excludes residual connection, which hurts model performance. APDA takes advantage of the three components and consistently outperforms the three variants on three real-world datasets.

4.4 Popular and Niche Item Improvement (RQ3)

In this subsection, we examine the performance gain on the niche and popular items and investigate whether the improvement of one group could sacrifice the performance of the other. The item groups are divided based on the number of clicks in the training

Table 4: The performance of popularity debiasing methods evaluated on popular and niche item group.

Methods	Gowalla				Epinions			
	Recall@20		NDCG@20		Recall@20		NDCG@20	
	niche	pop	niche	pop	niche	pop	niche	pop
LightGCN	0.0401(+0.00%)	0.3245(+0.00%)	0.0511	0.3892	0.0127(+0.00%)	0.1715(+0.00%)	0.0162	0.2228
PC_LightGCN	**0.0594**(+48.13%)	0.3048(-6.07%)	0.0750	0.3675	0.0213(+67.72%)	0.1496(-12.77%)	0.0272	0.1976
IPS_LightGCN	0.0503(+25.44%)	0.3119(-3.88%)	0.0643	0.3739	0.0153(+20.47%)	0.1529(-10.85%)	0.0194	0.2000
Reg_LightGCN	0.0396(-1.25%)	0.3234(-0.34%)	0.0503	0.3865	0.0137(+7.87%)	0.1700(-0.87%)	0.0177	0.2215
DICE_LightGCN	0.0469(+16.96%)	0.2992(-7.80%)	0.0594	0.3609	0.0164(-29.13%)	0.1781(+3.85%)	0.0209	0.2303
MACR_LightGCN	0.0593(+47.88%)	0.2898(-10.69%)	**0.0753**	0.3512	0.0123(-3.15%)	0.1346(-21.52%)	0.0165	0.1773
AdjNorm_LightGCN	0.041(+2.24%)	0.322(-0.77%)	0.052	0.3851	0.0149(+17.32%)	0.1707(-0.47%)	0.0191	0.2220
APDA_LightGCN	0.0508(+26.68%)	**0.3306**(+1.88%)	0.0647	**0.3956**	**0.0216**(+70.08%)	**0.1932**(+12.65%)	**0.0280**	**0.2479**

set. Specifically, we put the top 20% frequently clicked items into the popular item group and the rest into the niche item group.

We show the recommendation performance of two item groups on the Gowalla and Epinions datasets under LightGCN backbone in Table 4. We observe that **most baselines outperform the backbones on niche items on two datasets**. The improvement is mainly due to the fact that all baselines are trying to minimize the effect of item popularity on the recommendation results and recommend more unpopular items to users. **However, almost all baselines deteriorate popular item recommendation performance and neutralize the performance gain over the niche item group.** This observation suggests that, for graph-based recommender systems, it is challenging to mitigate the popularity bias without sacrificing model performance. **Interestingly, we find that our APDA does not sacrifice the performance of the popular item group.** This suggests that APDA can capture user niche preferences and put items in the appropriate order based on user preferences. The gap between APDA and the baselines demonstrates the effectiveness of the proposed aggregator.

4.5 Hyperparameter Sensitivity (RQ4)

To evaluate the impact of different hyperparameters on model performance, we divide vital hyperparameters into two groups.

The first group examines the effects of the number of GNN layers k on APDA and backbone LightGCN. We show Recall@20, and PRU@20 scores on the Epinions dataset since NDCG@20 scores exhibit the same trend, and we ignore them to save layout spaces in Figure 4. We observe that with embedding dimension and layer number increasing, two models get a recommendation performance improvement. For PRU scores, LightGCN performs worse other than the first layer as layers go deeper. Compared with the backbone, APDA shows a better capability in controlling the PRU@20 scores increasing when stacking GNN layers. A possible explanation is that the model automatically lowers the weight of popular neighbor weight which drives the model to mitigate the popularity bias. These results justify the effectiveness of the model design.

The second group targets to explore the impact of the magnitude of hyperparameter λ on model performance. We start from 0.1 and increment its value by 0.1 step size on Epinions datasets. The Recall@20, NDCG@20, PRU@20 scores are reported in Figure 5. We can observe the performance of APDA continuously increase before reaching the optimal λ. After that, the optimal λ achieves the best performance across three metrics. It validates the superiority of APDA. Then the performance keeps relatively stable. Therefore, it is vital to select an appropriate λ to achieve the best performance.

5 RELATED WORK

5.1 Popularity Debiasing

Popularity bias widely exists in the recommendation datasets and is characterized by the fact that popular items dominate the recommendation lists [4, 9, 12, 23, 28, 34]. Regularization is a potential solution whose basic idea is to divert attention from popular items and push the model to give a more balanced recommendation list. The LapDQ [35] regularize promotes diversity by adding the regularization terms on the item distances. INRS [19] applies the mean-match approach to correct the popularity bias to make less preferred and more diverse items recommended. Causal graphs can also be applied to understand and mitigate bias [32, 36, 48, 51, 53]. CausE [1] tries to adopt an unbiased method. A tricky finding claims that not all popularity bias will deteriorate performance. Accordingly, a new paradigm is proposed to remove the confounding popularity bias in model training [48]. Based on that, subsequent papers attempted to tackle this problem by considering the interaction timestamp and disentangling users' interests. Driven by casual effects, MPCI [14] is designed to eliminate the direct causal path from the popularity bias to the prediction. However, the existing debiasing methods do not target the aggregation process and may not fully address the popularity bias in GNN models.

5.2 GNN-based Recommender System

Deep learning based recommender systems reply on well-learned representations to predict interactions [45]. GNNs have shown this ability due to aggregating the information from neighboring nodes [3, 5, 10, 17, 46, 47, 49]. The propagation layers explicitly utilize the high-order connectivities in the user-item integration graph [33]. LightGCN further simplifies the architecture of the network by removing the transformation and non-activation parts [15]. NIA-GCN captures the relationships between pairs of neighbors at each layer [30]. After that, UltraGCN points out that multiple

stacks of layers might suffer from over-smoothing. They claim that skipping the explicit message passing and directly approximating the limit of infinite layers can lead to a more effective and efficient performance [25]. As self-supervised learning is gaining increasing popularity, it is also brought into the recommendation domain. To cope with the sparsity of the matrix and the noises in interactions, SGL [37] jointly employs the classic loss function and the additive contrastive objective through randomly constructing multiple sub-graphs. Subsequently, NCL [24] takes both the graph structure and the semantic space into account to capture potential node relatedness. Both methods improve the robustness of the model. DirectAU [31] analyzes the importance of the alignment and uniformity properties, i.e., representations of related positive pairs should be close to each other, and other instances should randomly scatter. Despite these efforts, they do not systematically tackle the popularity bias in the aggregation process. Our APDA can be combined with the existing GNN-based models to help neutralize the popularity bias.

6 CONCLUSION AND FUTURE WORK

In this work, we study the problem of how to neutralize the popularity bias in the aggregation process. We first theoretically analyze the cause of the popularity bias, i.e., the effect of item popularity, from the perspective of the gradient magnitude. Then we correspondingly propose a quantitative metric, named inverse popularity score, to measure the effect. We conduct empirical analysis and observe that the score aligns with the normalized number of item clicks. Based on the metric, we develop a novel graph aggregator which consists of three components: adaptive weight, weight scaling, and initial residual connection. We implement the APDA with two backbones. Extensive experiments verify the effectiveness of APDA. Our future work will explore popularity debiasing in view of enhancing data quality, given that the model effectiveness is contingent on utilizing appropriate training data [43, 44].

7 ACKNOWLEDGMENT

The work described in this paper was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. PolyU 25208322).

A PROOFS

A.1 Proof of Theorem 3.1

To prove Theorem 3.1, we incorporate the entry limit of $\widetilde{\mathbf{A}}_{ui}^{(k)}$ for self-looped connected graph $\mathcal{G}$ when stacking infinite GNN layers:

$$\lim_{k\to+\infty} \widetilde{\mathbf{A}}_{ui}^{(k)} = \frac{\sqrt{(d_u+1)(d_i+1)}}{2|\mathcal{E}|+|\mathcal{N}|}. \tag{12}$$

The relevance score of y_{ui} using last layer output is computed as:

$$y_{ui} = (\lim_{k\to+\infty} \sum_{m\in\mathcal{U}\cup\mathcal{I}} \widetilde{\mathbf{A}}_{um}^{(k)} e_m^{(0)})(\lim_{k\to+\infty} \sum_{n\in\mathcal{U}\cup\mathcal{I}} \widetilde{\mathbf{A}}_{in}^{(k)} e_n^{(0)}). \tag{13}$$

The relevance score of the randomly sampled negative items is unknown. And we take the expectation term of the relevance score with equal sampling probability distribution $P_I(j) = \frac{1}{|\mathcal{I}|}$ and for the sake of simplicity, interactions between the user and relevant items are not removed. Then the BPR loss is calculated as:

$$\begin{aligned}\mathcal{L}_{BPR} &= -\sum_{u,i\in\mathcal{N}(u)} \ln\sigma(y_{ui} - \mathbb{E}_{j\sim P_I}[y_{uj}]) \\ &\approx -\sum_{u,i\in\mathcal{N}(u)} \ln\sigma(y_{ui} - \sum_{j\in\mathcal{I}} \frac{y_{uj}}{|\mathcal{I}|}).\end{aligned} \tag{14}$$

Given that the embedding dimension is of sufficient large size, perfect node embedding pair training is achieved. That is to say, the optimization of any node embedding pair is independent [22, 26]. Thus, it suffices to consider only the inner product of the target node embedding pair, treating the remaining pairs as constants denoted by c. Then we combine Eq. (13)(14) and select loss terms related to the target node embedding pair. The BPR loss related to $(\mathbf{e}_u^{(0)}, \mathbf{e}_i^{(0)})$ is updated as:

$$\begin{aligned}\mathcal{L}_{BPR}(\mathbf{e}_u^{(0)}, \mathbf{e}_i^{(0)}) \approx -\sum_{(m,n)\in\mathcal{E}} \ln\sigma\Big(&\frac{\mathbf{e}_u^{(0)\top}\mathbf{e}_i^{(0)}\sqrt{(d_u+1)(d_i+1)}}{(2|\mathcal{E}|+|\mathcal{N}|)^2} \\ &* \frac{[(\sqrt{(d_m+1)(d_n+1)} - \frac{\sum_{j\in\mathcal{I}}\sqrt{(d_m+1)(d_j+1)}}{|\mathcal{I}|})]}{(2|\mathcal{E}|+|\mathcal{N}|)^2} + c\Big).\end{aligned} \tag{15}$$

Based on it, the gradient magnitude of BPR loss for inner product of node embedding pair $(\mathbf{e}_u^{(0)}, \mathbf{e}_i^{(0)})$ is calculated as follows:

$$\begin{aligned}|\frac{\partial L_{BPR}}{\partial(\mathbf{e}_u^{(0)\top}\mathbf{e}_i^{(0)})}| &= |\sum_{(m,n)\in\mathcal{E}} (y_{mn}-1)\frac{\sqrt{(d_u+1)(d_i+1)}}{(2|\mathcal{E}|+|\mathcal{N}|)^2} \\ &* \frac{(\sqrt{(d_m+1)(d_n+1)} - \frac{\sum_{j\in\mathcal{I}}\sqrt{(d_m+1)(d_j+1)}}{|\mathcal{I}|})}{(2|\mathcal{E}|+|\mathcal{N}|)^2}| \\ &\propto \sqrt{(d_u+1)(d_i+1)},\end{aligned} \tag{16}$$

where y_{mn} represents the interaction probability of user m and item n, which is shared for all node embedding pairs. Viewed this way, the speed at which any user-item embedding pair is pushed is proportional to their node degrees. And the node representation at layer k is the aggregated sum of node embeddings in k hops. Since each node embedding pair is independent, the gradient magnitude of BPR loss for $\mathbf{e}_u^{(k)\top}\mathbf{e}_i^{(k)}$ is defined as:

$$|\frac{\partial L_{BPR}}{\partial(\mathbf{e}_u^{(k)\top}\mathbf{e}_i^{(k)})}| \propto \sum_{m\in O^{(k)}(u), n\in O^{(k)}(i)} \widetilde{\mathbf{A}}_{mn}^{(k)}\sqrt{(d_m+1)(d_n+1)} \tag{17}$$

A.2 Proof of Proposition 3.3

We give the proof in the case of the niche item. We use $\mathbf{e}_m^{(0)}$ and $\mathbf{e}_n^{(0)}$ to represent the user and item embedding in the small component. Define the vector $\mathbf{v}$ as the negative direction of the large component. The vectors along the direction have a negative relevance score with nodes in the large component. The $\mathbf{e}_m^{(0)}$ and $\mathbf{e}_n^{(0)}$ could be pushed away from nodes in the large component along the direction $\mathbf{v}$ when optimizing the BPR loss since they are not connected to nodes in the large component. And the larger the magnitude, the smaller the relevance score and BPR loss. We have the optimal representation $\mathbf{e}_n^{(0)*} = \lim_{l\to+\infty} l\cdot\mathbf{v}$.

REFERENCES

[1] Stephen Bonner and Flavian Vasile. 2018. Causal embeddings for recommendation. In *Proceedings of the 12th ACM conference on recommender systems*. 104–112.

[2] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 3438–3445.

[3] Hao Chen, Zhong Huang, Yue Xu, Zengde Deng, Feiran Huang, Peng He, and Zhoujun Li. 2022. Neighbor enhanced graph convolutional networks for node classification and recommendation. *Knowledge-Based Systems* (2022), 108594.

[4] Hao Chen, Zefan Wang, Feiran Huang, Xiao Huang, Yue Xu, Yishi Lin, Peng He, and Zhoujun Li. 2022. Generative adversarial framework for cold-start item recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2565–2571.

[5] Hao Chen, Yue Xu, Feiran Huang, Zengde Deng, Wenbing Huang, Senzhang Wang, Peng He, and Zhoujun Li. 2020. Label-aware graph convolutional networks. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1977–1980.

[6] Jiawei Chen, Hande Dong, Xiang Wang, Fuli Feng, Meng Wang, and Xiangnan He. 2020. Bias and debias in recommender system: A survey and future directions. *arXiv preprint arXiv:2010.03240* (2020).

[7] Lei Chen, Le Wu, Richang Hong, Kun Zhang, and Meng Wang. 2020. Revisiting graph based collaborative filtering: A linear residual graph convolutional network approach. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 27–34.

[8] Zhihong Chen, Jiawei Wu, Chenliang Li, Jingxu Chen, Rong Xiao, and Binqiang Zhao. 2022. Co-training Disentangled Domain Adaptation Network for Leveraging Popularity Bias in Recommenders. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 60–69.

[9] Zhihong Chen, Rong Xiao, Chenliang Li, Gangfeng Ye, Haochuan Sun, and Hongbo Deng. 2020. Esam: Discriminative domain adaptation with non-displayed items to improve long-tail performance. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 579–588.

[10] Junnan Dong, Qinggang Zhang, Xiao Huang, Keyu Duan, Qiaoyu Tan, and Zhimeng Jiang. 2023. Hierarchy-Aware Multi-Hop Question Answering over Knowledge Graphs. (2023).

[11] Wenqi Fan, Xiaorui Liu, Wei Jin, Xiangyu Zhao, Jiliang Tang, and Qing Li. 2022. Graph Trend Filtering Networks for Recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 112–121.

[12] Wenjing Fu, Zhaohui Peng, Senzhang Wang, Yang Xu, and Jin Li. 2019. Deeply fusing reviews and contents for cold start users in cross-domain recommendation systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 94–101.

[13] Chen Gao, Xiang Wang, Xiangnan He, and Yong Li. 2022. Graph neural networks for recommender system. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. 1623–1625.

[14] Ming He, Changshu Li, Xinlei Hu, Xin Chen, and Jiwen Wang. 2022. Mitigating Popularity Bias in Recommendation via Counterfactual Inference. In *International Conference on Database Systems for Advanced Applications*. Springer, 377–388.

[15] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 639–648.

[16] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[17] Zhongyu Huang, Yingheng Wang, Chaozhuo Li, and Huiguang He. 2022. Going Deeper into Permutation-Sensitive Graph Neural Networks. In *International Conference on Machine Learning*. PMLR, 9377–9409.

[18] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased learning-to-rank with biased feedback. In *Proceedings of the tenth ACM international conference on web search and data mining*. 781–789.

[19] Toshihiro Kamishima, Shotaro Akaho, Hideki Asoh, and Jun Sakuma. 2014. Correcting Popularity Bias by Enhancing Recommendation Neutrality.. In *RecSys Posters*.

[20] Minseok Kim, Jinoh Oh, Jaeyoung Do, and Sungjin Lee. 2022. Debiasing Neighbor Aggregation for Graph Neural Network in Recommender Systems. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 4128–4132.

[21] Xiangnan He Le Wu, Xiang Wang, Kun Zhang, and Meng Wang. 2021. A survey on neural recommendation: From collaborative filtering to content and context enriched recommendation. *arXiv preprint arXiv:2104.13030* (2021).

[22] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. *Advances in neural information processing systems* 27 (2014).

[23] Langzhang Liang, Zenglin Xu, Zixing Song, Irwin King, and Jieping Ye. 2022. ResNorm: Tackling Long-tailed Degree Distribution Issue in Graph Neural Networks via Normalization. *arXiv preprint arXiv:2206.08181* (2022).

[24] Zihan Lin, Changxin Tian, Yupeng Hou, and Wayne Xin Zhao. 2022. Improving Graph Collaborative Filtering with Neighborhood-enriched Contrastive Learning. In *Proceedings of the ACM Web Conference 2022*. 2320–2329.

[25] Kelong Mao, Jieming Zhu, Xi Xiao, Biao Lu, Zhaowei Wang, and Xiuqiang He. 2021. UltraGCN: ultra simplification of graph convolutional networks for recommendation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 1253–1262.

[26] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the eleventh ACM international conference on web search and data mining*. 459–467.

[27] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).

[28] Wondo Rhee, Sung Min Cho, and Bongwon Suh. 2022. Countering Popularity Bias by Regularizing Score Differences. In *Proceedings of the 16th ACM Conference on Recommender Systems*. 145–155.

[29] Xiaoyuan Su and Taghi M Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Advances in artificial intelligence* 2009 (2009).

[30] Jianing Sun, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, Xiuqiang He, Chen Ma, and Mark Coates. 2020. Neighbor interaction aware graph convolution networks for recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1289–1298.

[31] Chenyang Wang, Yuanqing Yu, Weizhi Ma, Min Zhang, Chong Chen, Yiqun Liu, and Shaoping Ma. 2022. Towards Representation Alignment and Uniformity in Collaborative Filtering. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1816–1825.

[32] Wenjie Wang, Fuli Feng, Xiangnan He, Xiang Wang, and Tat-Seng Chua. 2021. Deconfounded recommendation for alleviating bias amplification. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1717–1725.

[33] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*. 165–174.

[34] Xinghua Wang, Zhaohui Peng, Senzhang Wang, Philip S Yu, Wenjing Fu, Xiaokang Xu, and Xiaoguang Hong. 2020. CDLFM: cross-domain recommendation for cold-start users via latent feature mapping. *Knowledge and Information Systems* (2020), 1723–1750.

[35] Jacek Wasilewski and Neil Hurley. 2016. Incorporating diversity in a learning to rank recommender system. In *The twenty-ninth international flairs conference*.

[36] Tianxin Wei, Fuli Feng, Jiawei Chen, Ziwei Wu, Jinfeng Yi, and Xiangnan He. 2021. Model-agnostic counterfactual reasoning for eliminating popularity bias in recommender system. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1791–1800.

[37] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. 2021. Self-supervised graph learning for recommendation. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*. 726–735.

[38] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. 2022. Graph neural networks in recommender systems: a survey. *Comput. Surveys* 55, 5 (2022), 1–37.

[39] Guipeng Xv, Chen Lin, Hui Li, Jinsong Su, Weiyao Ye, and Yewang Chen. 2022. Neutralizing Popularity Bias in Recommendation Models. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2623–2628.

[40] Longqi Yang, Yin Cui, Yuan Xuan, Chenyang Wang, Serge Belongie, and Deborah Estrin. 2018. Unbiased offline recommender evaluation for missing-not-at-random implicit feedback. In *Proceedings of the 12th ACM conference on recommender systems*. 279–287.

[41] Yonghui Yang, Le Wu, Richang Hong, Kun Zhang, and Meng Wang. 2021. Enhanced graph learning for collaborative filtering via mutual information maximization. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 71–80.

[42] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Lizhen Cui, and Quoc Viet Hung Nguyen. 2022. Are graph augmentations necessary? simple graph contrastive learning for recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1294–1303.

[43] Daochen Zha, Zaid Pervaiz Bhat, Kwei-Herng Lai, Fan Yang, and Xia Hu. 2023. Data-centric AI: Perspectives and Challenges. *arXiv preprint arXiv:2301.04819* (2023).

[44] Daochen Zha, Zaid Pervaiz Bhat, Kwei-Herng Lai, Fan Yang, Zhimeng Jiang, Shaochen Zhong, and Xia Hu. 2023. Data-centric artificial intelligence: A survey. *arXiv preprint arXiv:2303.10158* (2023).

[45] Daochen Zha, Louis Feng, Bhargav Bhushanam, Dhruv Choudhary, Jade Nie, Yuandong Tian, Jay Chae, Yinbin Ma, Arun Kejariwal, and Xia Hu. 2022. Autoshard: Automated embedding table sharding for recommender systems. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4461–4471.

[46] Peiyan Zhang, Jiayan Guo, Chaozhuo Li, Yueqi Xie, Jae Boum Kim, Yan Zhang, Xing Xie, Haohan Wang, and Sunghun Kim. 2023. Efficiently Leveraging Multi-level User Intent for Session-based Recommendation via Atten-Mixer Network. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*. 168–176.

[47] Qinggang Zhang, Junnan Dong, Keyu Duan, Xiao Huang, Yezi Liu, and Linchuan Xu. 2022. Contrastive Knowledge Graph Error Detection. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 2590–2599.

[48] Yang Zhang, Fuli Feng, Xiangnan He, Tianxin Wei, Chonggang Song, Guohui Ling, and Yongdong Zhang. 2021. Causal intervention for leveraging popularity bias in recommendation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 11–20.

[49] Yiding Zhang, Chaozhuo Li, Xing Xie, Xiao Wang, Chuan Shi, Yuming Liu, Hao Sun, Liangjie Zhang, Weiwei Deng, and Qi Zhang. 2022. Geometric Disentangled Collaborative Filtering. (2022).

[50] Minghao Zhao, Le Wu, Yile Liang, Lei Chen, Jian Zhang, Qilin Deng, Kai Wang, Xudong Shen, Tangjie Lv, and Runze Wu. 2022. Investigating Accuracy-Novelty Performance for Graph-based Collaborative Filtering. *arXiv preprint arXiv:2204.12326* (2022).

[51] Zihao Zhao, Jiawei Chen, Sheng Zhou, Xiangnan He, Xuezhi Cao, Fuzheng Zhang, and Wei Wu. 2022. Popularity bias is not always evil: Disentangling benign and harmful bias for recommendation. *IEEE Transactions on Knowledge and Data Engineering* (2022).

[52] Yu Zheng, Chen Gao, Liang Chen, Depeng Jin, and Yong Li. 2021. Dgcn: Diversified recommendation with graph convolutional networks. In *Proceedings of the Web Conference 2021*. 401–412.

[53] Yu Zheng, Chen Gao, Xiang Li, Xiangnan He, Yong Li, and Depeng Jin. 2021. Disentangling user interest and conformity for recommendation with causal embedding. In *Proceedings of the Web Conference 2021*. 2980–2991.

[54] Ziwei Zhu, Yun He, Xing Zhao, Yin Zhang, Jianling Wang, and James Caverlee. 2021. Popularity-opportunity bias in collaborative filtering. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 85–93.