UltraGCN: Ultra Simplification of Graph Convolutional Networks for Recommendation

Kelong Mao*
Gaoling School of AI
Renmin University of China
mkl@ruc.edu.cn

Jieming Zhu
Huawei Noah's Ark Lab
Shenzhen, China
jmzhu@ieee.org

Xi Xiao†
Tsinghua University
Peng Cheng Laboratory
xiaox@sz.tsinghua.edu.cn

Biao Lu
Huawei Noah's Ark Lab, China
lubiao4@huawei.com

Zhaowei Wang
Huawei Noah's Ark Lab
wangzhaowei3@huawei.com

Xiuqiang He
Huawei Noah's Ark Lab
hexiuqiang1@huawei.com

ABSTRACT

With the recent success of graph convolutional networks (GCNs), they have been widely applied for recommendation, and achieved impressive performance gains. The core of GCNs lies in its message passing mechanism to aggregate neighborhood information. However, we observed that message passing largely slows down the convergence of GCNs during training, especially for large-scale recommender systems, which hinders their wide adoption. LightGCN makes an early attempt to simplify GCNs for collaborative filtering by omitting feature transformations and nonlinear activations. In this paper, we take one step further to propose an ultra-simplified formulation of GCNs (dubbed UltraGCN), which skips infinite layers of message passing for efficient recommendation. Instead of explicit message passing, UltraGCN resorts to directly approximate the limit of infinite-layer graph convolutions via a constraint loss. Meanwhile, UltraGCN allows for more appropriate edge weight assignments and flexible adjustment of the relative importances among different types of relationships. This finally yields a simple yet effective UltraGCN model, which is easy to implement and efficient to train. Experimental results on four benchmark datasets show that UltraGCN not only outperforms the state-of-the-art GCN models but also achieves more than 10x speedup over LightGCN.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; **Collaborative filtering**.

KEYWORDS

Recommender systems; collaborative filtering; graph convolutional networks

* Part of the work was done during the internship at Huawei Noah's Ark Lab when the author studied at Tsinghua University.
† Corresponding Author

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482291

ACM Reference Format:
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 and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482291

1 INTRODUCTION

Nowadays, personalized recommendation has become a prevalent way to help users find information of their interests in various applications, such as e-commerce, online news, and social media. The core of recommendation is to precisely match a user's preference with candidate items. Collaborative filtering (CF) [11], as a fundamental recommendation task, has been widely studied in both academia and industry. A common paradigm of CF is to learn vector representations (i.e., embeddings) of users and items from historical interaction data and then perform top-k recommendation based on the pairwise similarity between user and item embeddings.

As the interaction data can be naturally modelled as graphs, such as user-item bipartite graph and item-item co-occurrence graph, recent studies [10, 13, 24, 27] opt for powerful graph convolutional/neural networks (GCNs, or GNNs in general) to learn user and item node representations. These GCN-based models are capable of exploiting higher-order connectivity between users and items, and therefore have achieved impressive performance gains for recommendation. PinSage [31] and M2GRL [26] are two successful use cases in industrial applications.

Despite the promising results obtained, we argue that current model designs are heavy and burdensome. In order to capture higher-order collaborative signals and better model the interaction process between users and items, current GNN-based CF models [1, 24, 27, 32] tend to seek for more and more sophisticated network encoders. However, we observed that these GCN-based models are hard to train with large graphs, which hinders their wide adoption in industry. Industrial recommender systems usually involve massive graphs due to the large numbers of users and items. This brings efficiency and scalability challenges for model designs. Towards this end, some research efforts [4, 10, 17] have been made to simplify the design of GCN-based CF models, mainly by removing feature transformations and non-linear activations that are not necessary for CF. These simplified models not only obtain much

better performance than those complex ones, but also brings some benefits on training efficiency.

Inspired by these pioneer studies, we performed further empirical analysis on the training process of GCN-based models and found that message passing (i.e., neighborhood aggregation) on a large graph is usually time-consuming for CF. In particular, stacking multiple layers of message passing could lead to the slow convergence of GCN-based models on CF tasks. Although the aforementioned models such as LightGCN [10] have already been simplified for training, the message passing operations still dominate their training. For example, in our experiments, three-layer LightGCN takes more than 700 epochs to converge to its best result on the Amazon-Books dataset [9], which would be unacceptable in an industrial setting. How to improve the efficiency of GCN models yet retain their effectiveness on recommendation is still an open problem.

To tackle this challenge, in this work, we question the necessity of explicit message passing layers in CF, and finally propose an ultra-simplified form of GCNs (dubbed UltraGCN) without message passing for efficient recommendation. More specifically, we analyzed the message passing formula of LightGCN and identified three critical limitations: 1) The weights assigned on edges during message passing are counter-intuitive, which may not be appropriate for CF. 2) The propagation process recursively combines different types of relationship pairs (including user-item pairs, item-item pairs, and user-user pairs) into the model, but fails to capture their varying importance. This may also introduce noisy and uninformative relationships that confuse the model training. 3) The over-smoothing issue limits the use of too many layers of message passing in LightGCN. Therefore, instead of performing explicit message passing, we seek to directly approximate the limit of infinite-layer graph convolutions via a constraint loss, which leads to the ultra-simplified GCN model, UltraGCN. The loss-based design of UltraGCN is very flexible, allowing us to manually adjust the relative importances of different types of relationships and also avoid the over-smoothing problem by negative sampling. This finally yields a simple yet effective UltraGCN model, which is easy to implement and efficient to train. Furthermore, we show that UltraGCN achieves significant improvements over the state-of-the-art CF models. For instance, UltraGCN attains up to 76.6% improvement in NDCG@20 and more than 10x speedup in training over LightGCN on the Amazon-Books dataset.

In summary, this work makes the following main contributions:

- We empirically analyze the training inefficiency of LightGCN and further attribute its cause to the critical limitations of the message passing mechanism.
- We propose an ultra simplified formulation of GCN, namely UltraGCN, which skips infinite layers of explicit message passing for efficient recommendation.
- Extensive experiments have been conducted on four benchmark datasets to show the effectiveness and efficiency of UltraGCN.

2 MOTIVATION

In this section, we revisit the GCN and LightGCN models, and further identify the limitations resulted from the inherent message passing mechanism, which also justify the motivation of our work.

2.1 Revisiting GCN and LightGCN

GCN [14] is a representative model of graph neural networks that applies message passing to aggregate neighborhood information. The message passing layer with self-loops is defined as follows:

$$E^{(l+1)} = \sigma\Big(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}E^{(l)}W^{(l)}\Big) \tag{1}$$

with $\hat{A} = A + I$ and $\hat{D} = D + I$. A, D, I are the adjacency matrix, the diagonal node degree matrix, and the identity matrix, respectively. I is used to integrate self-loop connections on nodes. $E^{(l)}$ and $W^{(l)}$ denote the representation matrix and the weight matrix for the l-th layer. $\sigma(\cdot)$ is a non-linear activation function (e.g., ReLU).

Despite the wide success of GCN in graph learning, several recent studies [4, 10, 17, 29] found that simplifying GCN appropriately can further boost the performance on CF tasks. LightGCN [10] is one such simplified GCN model that removes feature transformations (i.e., $W^{(l)}$) and non-linear activations (i.e., σ). Its message passing layer can thus be expressed as follows:

$$E^{(l+1)} = (\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}})E^{(l)} \tag{2}$$

It is worth noting that although LightGCN also removes self-loop connections on nodes, its layer combination operation has a similar effect to self-loops used in Equation 2, becauase both of them output a weighted sum of the embeddings propagated at each layer as the final output representation. Given self-loop connections, we can rewrite the message passing operations for user u and item i as follows:

$$e_u^{(l+1)} = \frac{1}{d_u+1}e_u^{(l)} + \sum_{k\in\mathcal{N}(u)}\frac{1}{\sqrt{d_u+1}\sqrt{d_k+1}}e_k^{(l)}, \tag{3}$$

$$e_i^{(l+1)} = \frac{1}{d_i+1}e_i^{(l)} + \sum_{v\in\mathcal{N}(i)}\frac{1}{\sqrt{d_v+1}\sqrt{d_i+1}}e_v^{(l)} \tag{4}$$

where $e_u^{(l)}$ and $e_u^{(l)}$ denote the embeddings of user u and item i at layer l. $\mathcal{N}(u)$ and $\mathcal{N}(i)$ represent their neighbor node sets, respectively. d_u denotes the original degree of the node u.

As shown in the left part of Figure 1, LightGCN performs a stack of message passing layers to obtain the embeddings and finally uses their dot product for training.

2.2 Limitations of Message Passing

We argue that such message passing layers have potential limitations that hinder the effective and efficient training of GCN-based models in recommendation tasks. To illustrate it, we take the l-th layer message passing of LightGCN in Equation 3 and 4 for example. Note that u and v denote users while i and k denote items. LightGCN takes the dot product of the two embedding as the final logit to capture the preference of user u on item i. Thus we obtain:

$$\begin{aligned} e_u^{(l+1)}\cdot e_i^{(l+1)} = \alpha_{ui}(e_u^{(l)}\cdot e_i^{(l)}) + \sum_{k\in\mathcal{N}(u)}\alpha_{ik}(e_i^{(l)}\cdot e_k^{(l)}) + \\ \sum_{v\in\mathcal{N}(i)}\alpha_{uv}(e_u^{(l)}\cdot e_v^{(l)}) + \sum_{k\in\mathcal{N}(u)}\sum_{v\in\mathcal{N}(i)}\alpha_{kv}(e_k^{(l)}\cdot e_v^{(l)}), \end{aligned} \tag{5}$$

where α_{ui}, α_{ik}, α_{uv}, and α_{kv} can be derived as follows:

$$\alpha_{ui} = \frac{1}{(d_u+1)(d_i+1)},$$
$$\alpha_{ik} = \frac{1}{\sqrt{d_u+1}\sqrt{d_k+1}(d_i+1)},$$
$$\alpha_{uv} = \frac{1}{\sqrt{d_v+1}\sqrt{d_i+1}(d_u+1)},$$
$$\alpha_{kv} = \frac{1}{\sqrt{d_u+1}\sqrt{d_k+1}\sqrt{d_v+1}\sqrt{d_i+1}}$$

Therefore, we can observe that multiple different types of collaborative signals, including user-item relationships (u-i and k-v), item-item relationships (k-i), and user-user relationships (u-v), are captured when training GCN-based models with message passing layers. This also reveals why GCN-based models are effective for CF.

However, we found that the edge weights assigned on various types of relationships are not justified to be appropriate for CF tasks. Based on our empirical analysis, we identify three critical limitations of the message passing layers in GCN-based models:

- **Limitation I**: The weight α_{ik} is used to model the item-item relationships. However, given the user u, the factors of item i and item k are asymmetric ($\frac{1}{\sqrt{d_k+1}}$ for item k while $\frac{1}{d_i+1}$ for item i). This is not reasonable since it is counter-intuitive to treat the item k and item i unequally. Similarly, α_{uv} that models the user-user relationships also suffer this issue. Such unreasonable weight assignments may mislead the model training and finally result in sub-optimal performance.
- **Limitation II**: The message passing recursively combine different types of relationships into the modeling. While such collaborative signals should be beneficial, the above message passing formula fails to capture their varying importance. Meanwhile, stacking multiple layers of message passing as in Equation 5 likely introduce uninformative, noisy, or ambiguous relationships, which could largely affect the training efficiency and effectiveness. It is desirable to flexibly adjust the relative importances of various relationships. We validate this empirically in Section 4.4.
- **Limitation III**: Stacking more layers of message passing should capture higher-order collaborative signals, but in fact the performance of LightGCN begins to degrade at layer 2 or 3 [10]. We partially attribute it to the over-smoothing problem of message passing. As graph convolution is a special form of Laplacian smoothing [16], performing too many layers of message passing will make the nodes with the same degrees tend to have exactly the same embeddings. According to Theorem 1 in [5], we can derive the infinite powers of message passing which take the following limit:

$$\lim_{l\to\infty}(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}})^l_{i,j} = \frac{\sqrt{(d_i+1)(d_j+1)}}{2m+n} \tag{6}$$

where n and m are the total numbers of nodes and edges in the graph, respectively.

The above limitations of message passing motivate our work. We question the necessity of explicit message passing layers in CF and further propose an ultra-simplified formulation of GCN, dubbed UltraGCN.

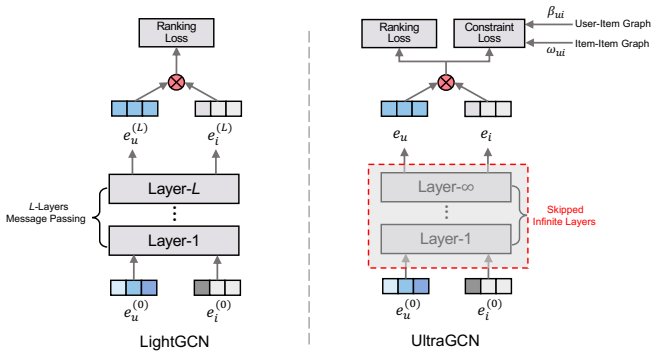


Figure 1: Illustrations of training of LightGCN (left) and UltraGCN (right). LightGCN needs to recurrently perform L-layers message passing to get the final embeddings for training, while UltraGCN can "skip" such message passing to make the embeddings be directly trained, largely improving training efficiency and helping real deployment.

3 UltraGCN

In this section, we present our ultra-simplified UltraGCN model and demonstrate how to incorporate different types of relationships in a flexible manner. We also elaborate on how it overcomes the above limitations and analyze its connections to other related models.

3.1 Learning on User-Item Graph

Due to the limitations of message passing, in this work, we take one step forward to question the necessity of explicit message passing in CF. Considering that the limit of infinite powers of message passing exists as shown in Equation 6, we wonder whether it is possible to skip the infinite-layer message passing yet approximate the convergence state reached.

After repeating infinite layers of message passing, we express the final convergence condition as follows:

$$e_u = \lim_{l\to\infty} e_u^{(l+1)} = \lim_{l\to\infty} e_u^{(l)} \tag{7}$$

That is, the representations of the last two layers keep unchanged, since the vector generated from neighborhood aggregation equals to the node representation itself. We use e_u (or e_i) to denote the final converged representation of user u (or item i). Then, Equation 3 can be rewritten as:

$$e_u = \frac{1}{d_u+1}e_u + \sum_{i\in\mathcal{N}(u)} \frac{1}{\sqrt{d_u+1}\sqrt{d_i+1}}e_i \tag{8}$$

After some simplifications, we derive the following convergence state:

$$e_u = \sum_{i\in\mathcal{N}(u)} \beta_{u,i}e_i, \quad \beta_{u,i} = \frac{1}{d_u}\sqrt{\frac{d_u+1}{d_i+1}} \tag{9}$$

In other words, if Equation 9 is satisfied for each node, it reaches the convergence state of message passing.

Instead of performing explicit message passing, we aim to directly approximate such convergence state. To this end, a straightforward way is to minimize the difference of both sides of Equation 9. In this work, we normalize the embeddings to unit vectors and then maximize the dot product of both terms:

$$\max \sum_{i \in \mathcal{N}(u)} \beta_{u,i} e_u^\top e_i \,, \quad \forall u \in U \,, \tag{10}$$

which is equivalent to maximize the cosine similarity between e_u and e_i. For ease of optimization, we further incorporate sigmoid activation and negative log likelihood [2], and derive the following loss:

$$\mathcal{L}_C = -\sum_{u \in U} \sum_{i \in \mathcal{N}(u)} \beta_{u,i} \log \left(\sigma(e_u^\top e_i)\right) , \tag{11}$$

where σ is the sigmoid function. The loss is optimized to fulfill the structure constraint imposed by Equation 9. As such, we denote $\mathcal{L}_C$ as the *constraint loss* and denote $\beta_{u,i}$ as the *constraint coefficient*.

However, optimizing $\mathcal{L}_C$ could also suffer from the over-smoothing problem as $\mathcal{L}_C$ requires all connected pairs ($\beta_{u,i} > 0$) to be similar. In this way, users and items could easily converge to the same embeddings. To alleviate the over-smoothing problem, conventional GCN-based CF models usually fix a small number of message passing layers, e.g., 2~4 layers in LightGCN. Instead, as UltraGCN approximates the limit of infinite-layer message passing via a constraint loss, we choose to perform negative sampling during training. This is inspired from the negative sampling strategy used in Word2Vec [19], which provides a more simple and effective way to counteract the over-smoothing problem. After performing negative sampling, we finally derive the following constraint loss:

$$\mathcal{L}_C = -\sum_{(u,i) \in N^+} \beta_{u,i} \log \left(\sigma(e_u^\top e_i)\right) - \sum_{(u,j) \in N^-} \beta_{u,j} \log \left(\sigma(-e_u^\top e_j)\right) \tag{12}$$

where N^+ and N^- represent the sets of positive pairs and randomly sampled negative pairs. Note that we omit the summation over U for ease of presentation. The constraint loss $\mathcal{L}_C$ enables UltraGCN to directly approximate the limit of infinite-layer message passing to capture arbitrary high-order collaborative signals in the user-item bipartite graph, while effectively avoiding the troublesome over-smoothing issue via negative sampling. Furthermore, we note that $\beta_{u,i}$ acts as the loss weight in $\mathcal{L}_C$, which is inversely proportional to d_u and d_i with similar magnitudes. This is interpretable for CF. If a user interacts with many items or an item is interacted by many users, the influence of their interaction would be small, and thus the loss weight of this (u, i) pair should be small.

3.1.1 Optimization. Typically, CF models perform item recommendation by applying either pairwise BPR (Bayesian personalized ranking) loss [22] or pointwise BCE (binary cross-entropy) loss [11] for optimization. We formulate CF as a link prediction problem in graph learning. Therefore, we choose the following BCE loss as the main optimization objective. It is also consistent with the loss format of $\mathcal{L}_C$.

$$\mathcal{L}_O = -\sum_{(u,i) \in N^+} \log \left(\sigma(e_u^\top e_i)\right) - \sum_{(u,j) \in N^-} \log \left(\sigma(-e_u^\top e_j)\right) \tag{13}$$

where N^+ and N^- represent positive and randomly sampled negative links (i.e., u-j pairs). Note that for simplicity, we use the same sets of sample pairs with $\mathcal{L}_C$, but they could also be made different conveniently.

As $\mathcal{L}_O$ and $\mathcal{L}_C$ depends only on the user-item relationships, we define it as the base version of UltraGCN, denoted as UltraGCN$_{Base}$, which has the following optimization objective.

$$\mathcal{L} = \mathcal{L}_O + \lambda \mathcal{L}_C \,, \tag{14}$$

where λ is the hyper-parameter to control the importance weights of two losse terms.

3.2 Learning on Item-Item Graph

As Equation 5 shows, except for user-item relationships, some other relationships (e.g., item-item and user-user relationships) also greatly contribute to the effectiveness of GCN-based models on CF. However, in conventional GCN-based models, these relationships are implicitly learned through the same message passing layers with user-item relationships. This not only leads to the unreasonable edge weight assignments as discussed in Section 2.2, but also fails to capture the relative importances of different types of relationships. In contrast, UltraGCN does not rely on explicit message passing so that we can separately learn other relationships in a more flexible way. This also enables us to manually adjust the relative importances of different relationships.

We emphasize that UltraGCN is flexible to extend to model many different relation graphs, such as user-user graphs, item-item graphs, and even knowlege graphs. In this work, we mainly demonstrate its use on the item-item co-occurrence graph, which has been shown to be useful for recommendation in [26]. We first build the item-item co-occurrence graph by linking items that have co-occurrences, which produces the following weighted adjacent matrix $G \in \mathcal{R}^{|I| \times |I|}$.

$$G = A^\top A \tag{15}$$

where each entry denotes the co-occurrences of two items. We follow Equation 9 to approximate infinite-layer graph convolution on G and derive the new coefficient $\omega_{i,j}$:

$$\omega_{i,j} = \frac{G_{i,j}}{g_i - G_{i,i}} \sqrt{\frac{g_i}{g_j}} \,, \quad g_i = \sum_k G_{i,k} \tag{16}$$

where g_i and g_j denote the degrees (sum by column) of item i and item j in G, respectively.

Similar to Equation 12, we can derive the constraint loss on the item-item graph to learn the item-item relationships in an explicit way. However, as the adjacency matrix G of the item-item graph is usually much denser compared to the sparse adjacency matrix A of the user-item graph, directly minimizing the constraint loss on G would introduce too many unreliable or noisy item-item pairs into optimization, which may make UltraGCN difficult to train. This is also similar to the **Limitation II** of conventional GCN-based models described in Section 2.2. But thanks to the flexible design of UltraGCN, we choose to select only informative pairs for training.

Specifically, to keep sparse item connections and retain training efficiency, we first select top-K most similar items $S(i)$ for item i according to $\omega_{i,j}$. Intuitively, $\omega_{i,j}$ measures the similarity of item i and item j, since it is proportional to the co-occurrence number of item i and item j, yet inversely proportional to the total degrees of both items. Instead of directly learning item-item pairs, we propose

to augment positive user-item pairs to capture item-item relationships. This keeps the training terms of UltraGCN being unified and decrease the possible difficulty in multi-task learning. We also empirically show that such a way can achieve better performance in Section 4.4. For each positive (u, i) pair, we first construct K weighted positive (u, j) pairs, for $j \in S(i)$. Then, we penalize the learning of these pairs with the more reasonable similarity score $\omega_{i,j}$ and derive the constraint loss $\mathcal{L}_I$ on the item-item graph as follow:

$$\mathcal{L}_I = -\sum_{(u,i)\in N^+} \sum_{j\in S(i)} \omega_{i,j} \log(\sigma(e_u^\top e_j)) \tag{17}$$

where $|S(i)| = K$. We omit the negative sampling here as the negative sampling in $\mathcal{L}_C$ and $\mathcal{L}_O$ has already enabled UltraGCN to counteract over-smoothing. With this constraint loss, we extend UltraGCN to better learn item-item relationships, and finally derive the following training objective of UltraGCN,

$$\mathcal{L} = \mathcal{L}_O + \lambda \mathcal{L}_C + \gamma \mathcal{L}_I \tag{18}$$

where λ and γ are hyper-parameters to adjust the relative importances of user-item and item-item relationships, respectively.

Figure 1 illustrates the simple architecture of UltraGCN in contrast to LightGCN. Similarly, in inference, we use the dot product $\hat{y}_{ui} = e_u^\top e_i$ between user u and item i as the ranking score for recommendation.

3.3 Discussion

3.3.1 Model Analysis. We first analyze the strengths of our UltraGCN model: 1) The weights assigned on edges in UltraGCN, i.e., $\beta_{i,j}$ and $\omega_{i,j}$, are more reasonable and interpretable for CF, which are helpful to better learn user-item and item-item relationships, respectively. 2) Without explicit message passing, UltraGCN is flexible to separately customize its learning with different types of relationships. It is also able to select valuable training pairs (as in Section 3.2), rather than learn from all neighbor pairs indistinguishably, which may be mislead by noise. 3) Although UltraGCN is trained with different types of relationships in a multi-task learning way, its training losses (i.e., $\mathcal{L}_C$, $\mathcal{L}_I$, and $\mathcal{L}_O$) are actually unified, following the form of binary cross entropy. Such unification facilitates the training of UltraGCN, which converges fast. 4) The design of UltraGCN is flexible, by setting γ to 0, it reduces to UltraGCN$_{Base}$, which only learns on the user-item graph. The performance comparison between UltraGCN and UltraGCN$_{Base}$ is provided in Table 2.

Note that in the current version, we do not incorporate the modeling of user-user relationships in UltraGCN. This is mainly because that users' interests are much broader than items' attributes. We found that it is harder to capture the user-user relationships from the user-user co-occurrence graph only. In Section 4.4, we empirically show that learning on the user-user co-occurrence graph does not bring noticeable improvements to UltraGCN. In contrast, conventional GCN-based CF models indistinguishably learn over all relationships from the user-item graph (i.e., Limitation II) likely suffer from performance degradation. The user-user relationships may be better modeled from a social network graph, and we leave it for future work.

3.3.2 Relations to Other Models. In this part, we discuss the relations between our UltraGCN and some other existing models.

Relation to MF. UltraGCN is formally to be a new weighted MF model with BCE loss tailored for CF. In contrast to previous MF models (e.g., NeuMF [11]), UltraGCN can more deeply mine the collaborative information using graphs, yet keep the same concise architecture and model efficiency as MF.

Relation to Network Embedding Methods. Qiu et al. [21] have proved that many popular network embedding methods with negative sampling (e.g., DeepWalk [20], LINE [25], and Node2Vec [7]) all can be unified into the MF framework. However, in contrast to these network embedding methods, the edge weights used in UltraGCN are more meaningful and reasonable for CF, and thus lead to much better performance. In addition, the random walk in many network embedding methods will also uncontrollably introduce uninformative relationships that affect the performance. We empirically show the superiority of UltraGCN over three typical network embedding methods on CF in Section 4.2.

Relation to One-Layer LightGCN. We emphasize that UltraGCN is also different from one-layer LightGCN with BCE loss, because LightGCN applies weight combination to embeddings aggregation while our constraint coefficients are imposed on the constraint loss function, which aims to learn the essence of infinite-layer graph convolution. On the contrary, UltraGCN can overcome the limitations of one-layer LightGCN as described in Section 3.2.

3.3.3 Model Complexity. Given the embedding size d, K similar items for each $S(i)$, R as the number of negative samples for each positive pair, and $|A^+|$ as the number of valid non-zero entries in the user-item interaction matrix, we can derive the training time complexity of UltraGCN: $O((K + R + 1) * |A^+| * (d^2 + 1))$. We note that the time complexities to calculate β and ω are $O(1)$, since we can pre-calculate them offline before training. As we usually limit K to be small (e.g., 10 in our experiments) in practice, the time complexity of UltraGCN lies in the same level with MF, which is $O((R + 1) * |A^+| * d^2)$. Besides, the only trainable parameters in UltraGCN are the embeddings of users and items, which is also the same with MF and LightGCN. As a result, our low-complexity UltraGCN brings great efficiency for model training and should be more practically applicable to large-scale recommender systems.

4 EXPERIMENTS

We first compare UltraGCN with various state-of-the-art CF methods to demonstrate its effectiveness and high efficiency. We also perform detailed ablation studies to justify the rationality and effectiveness of the design choices of UltraGCN.

4.1 Experimental Setup

Datasets and Evaluation Protocol. We use four publicly available datasets, including Amazon-Book, Yelp2018, Gowalla, and MovieLens-1M to conduct our experiments, as many recent GCN-based CF models [10, 27, 28, 32] are evaluated on these four datasets. We closely follow these GCN-based CF studies and use the same data split as them. Table 1 shows the statistics of the used datasets.

For the evaluation protocol, Recall@20 and NDCG@20 are chosen as the evaluation metrics as they are popular in the evaluation of GCN-based CF models. We treat all items not interacted by a user as the candidates, and report the average results over all users.

Table 1: Statistics of the datasets.

Dataset	#Users	#Items	#Interactions	Density
Amazon-Book	52, 643	91, 599	2, 984, 108	0.062%
Yelp2018	31, 668	38, 048	1, 561, 406	0.130%
Gowalla	29, 858	40, 981	1, 027, 370	0.084%
Movielens-1M	6,022	3,043	995, 154	5.431%

Baselines. In total, we compare UltraGCN with various types of the state-of-the-art models, covering MF-based (MF-BPR [15], ENMF [3]), metric learing-based (CML [12]), network embedding methods (DeepWalk [20], LINE [25], and Node2Vec [7]), and GCN-based (NGCF [27], NIA-GCN [24], LR-GCCF [4], LightGCN [10], and DGCF [28]).

Parameter Settings. Generally, we adopt Gaussian distribution with 0 mean and 10^{-4} standard deviation to initialize embeddings. In many cases, we adopt L_2 regularization with 10^{-4} weight and we set the learning rate to 10^{-4}, the batch size to 1024, the negative sampling ratio R to 300, and the size of the neighbor set K to 10. In particular, we fix the embedding size to 64 which is identical to recent GCN-based work [10, 24, 27, 28] to keep the same level of the number of parameters for fair comparison. We tune λ in [0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4], and γ in [0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3, 3.5]. For some baselines, we report the results from their papers to keep consistency. They are also comparable since we use the exactly same datasets and experimental settings provided by them. For other baselines, we mainly use their official open-source code and carefully tune the parameters to achieve the best performance for fair comparisons. To allow for reproduciblility, we have released the source code and benchmark settings of UltraGCN at Github[1].

4.2 Performance Comparison

Table 2 reports the performance comparison results. We have the following observations:

- UltraGCN consistently yields the best performance across all four datasets. In particular, UltraGCN hugely improves over the strongest GCN-based baseline (i.e., DGCF) on Amazon-Book by 61.4% and 71.6% w.r.t. Recall@20 and NDCG@20 respectively. The results of significance testing indicates that our improvements over the current strongest GCN-based baseline are statistically significant (p-value < 0.05). With additional learning on the item-item graph, UltraGCN performs consistently better than its simpler variant UltraGCN$_{Base}$. We attribute such good performance of UltraGCN to the following reasons: 1) Compared with network embedding models and the other GCN-based models, UltraGCN can respectively filter uninformative user-item and item-item relationships in a soft way (i.e., optimize with β) and a hard way (i.e., only select K most similar item pairs). The edge weights for the learning of user-item and item-item relationships in UltraGCN are also more reasonable; 2) Compared with other baselines, UltraGCN can leverage powerful graph convolution to exploit useful and deeper collaborative information in graphs. These advantages together lead to the superiority of UltraGCN than compared state-of-the-art models.
- Overall, network embedding models perform worse than GCN-based models, especially on Gowalla. The reason might be that the powerful graph convolution is more effective than traditional random walk or heuristic mining strategies in many network embedding methods, to capture collaborative information for recommendation.
- Since UltraGCN is a special MF which only needs the dot product operation for embeddings, its architecture is orthogonal to some state-of-the-art models (e.g., DGCF). Therefore, similar to MF, UltraGCN can be deemed as an effective and efficient CF framework which is possible to be incorporated with other methods, such as enabling disentangled representation for users and items as DGCF, to achieve better performance. We leave such study in future work.

4.3 Efficiency Comparison

As highlighted in Section 3.3, UltraGCN is endowed with high training efficiency for CF thanks to its concise and unified designs. We have also theoretically demonstrated that the training time complexity of UltraGCN is on the same level as MF in Section 3.3.3. In this section, we further empirically demonstrate the superiority of UltraGCN on training efficiency compared with other CF models, especially GCN-based models. To be specific, we select MF-BPR, ENMF, LightGCN, and LR-GCCF as the competitors, which are relatively efficient models in their respective categories. To be more convincing, we compare their training efficiency from two views:

- The total training time and epochs for achieving their best performance.
- Training them with the same epochs to see what performance they can achieve.

Note that the validation time is not included in the training time. Considering the fact that the official implementations of the compared models can be optimized to be more efficient, we use a uniform code framework implemented by ourselves for all models for fair comparison. In particular, our implementations refer to their official versions and optimize them with uniform acceleration methods (e.g, parallel sampling) to ensure the fairness of comparison. We will release all of our code. Experiments are conducted on Amazon-Book with the same Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz machine with one GeForce RTX 2080 GPU for all compared models. Results of the two experiments are shown in Table 3 and Table 4 respectively. We have the following conclusions:

(1) Table 3 shows that the training speed (i.e., Time/Epoch) of UltraGCN is close to MF-BPR, which empirically justifies our analysis that the time complexities of UltraGCN and MF are on the same level. UltraGCN needs 75 epochs to converge which is much less than LR-GCCF and LightGCN, leading to only 45 minutes for total training. Finally, UltraGCN has around 14x, 4x, 4x speedup compared with LightGCN, LR-GCCF, and ENMF respectively, demonstrating the big efficiency superiority of UltraGCN.

(2) Table 4 shows that when UltraGCN converges (i.e., train the fixed 75 epochs), the performances of all the other compared models are much worse than UltraGCN. That is to say, UltraGCN can achieve much better performance with less time, which further

[1] https://github.com/xue-pai/UltraGCN

Table 2: Overall performance comparison. Improv. denotes the relative improvements over the best GNN-based baselines.

Model	Amazon-Books		Yelp2018		Gowalla		Movielens-1M	
	Recall@20	NDCG@20	Recall@20	NDCG@20	F1@20	NDCG@20	Recall@20	NDCG@20
MF-BPR	0.0338	0.0261	0.0549	0.0445	0.1616	0.1366	0.2153	0.2175
CML	0.0522	0.0428	0.0622	0.0536	0.1670	0.1292	0.1730	0.1563
ENMF	0.0359	0.0281	0.0624	0.0515	0.1523	0.1315	0.2315	0.2069
DeepWalk	0.0346	0.0264	0.0476	0.0378	0.1034	0.0740	0.1348	0.1057
LINE	0.0410	0.0318	0.0549	0.0446	0.1335	0.1056	0.2336	0.2226
Node2Vec	0.0402	0.0309	0.0452	0.0360	0.1019	0.0709	0.1475	0.1186
NGCF	0.0344	0.0263	0.0579	0.0477	0.1570	0.1327	0.2513	0.2511
NIA-GCN	0.0369	0.0287	0.0599	0.0491	0.1359	0.1106	0.2359	0.2242
LR-GCCF	0.0335	0.0265	0.0561	0.0343	0.1519	0.1285	0.2231	0.2124
LightGCN	0.0411	0.0315	0.0649	0.0530	0.1830	0.1554	0.2576	0.2427
DGCF	0.0422	0.0324	0.0654	0.0534	0.1842	0.1561	0.2640	0.2504
UltraGCN$_{Base}$	0.0504	0.0393	0.0667	0.0552	0.1845	0.1566	0.2671	0.2539
UltraGCN	**0.0681**	**0.0556**	**0.0683**	**0.0561**	**0.1862**	**0.1580**	**0.2787**	**0.2642**
Improv.	61.37%	71.60%	4.43%	5.06%	1.09%	1.22%	5.57%	5.51%
p-value	4.03e-8	5.64e-8	1.61e-4	1.24e-4	7.21e-3	3.44e-4	4.19e-5	2.23e-5

Table 3: Efficiency comparison from the first view.

Model	Time/Epoch	#Epoch	Training Time
MF-BPR	**30s**	**23**	**12m**
ENMF	129s	81	2h54m
LR-GCCF	67s	165	3h5m
LightGCN	51s	780	11h3m
UltraGCN	**36s**	**75**	**45m**

Table 4: Efficiency comparison from the second view. All models are trained with the fixed 75 epochs except MF-BPR. Since MF-BPR needs less than 75 epochs to converge, we report its actual training time.

Model	Training Time	Recall@20	NDCG@20
MF-BPR	**12m**	0.0338	0.0261
ENMF	2h41m	0.0355	0.0277
LR-GCCF	1h13m	0.0304	0.0185
LightGCN	1h4m	0.0342	0.0262
UltraGCN	**45m**	**0.0681**	**0.0556**

demonstrates the higher efficiency of UltraGCN than the other GCN-based CF models.

4.4 Ablation Study

We perform ablation studies on Amazon-Book to justify the following opinions: (i) The designs of UltraGCN is effective, which can flexibly and separately learn the user-item relationships and item-item relationships to improve recommendation performance; (ii) Augmenting positive user-item pairs for training to learn item-item relationships can achieve better performance than optimizing between item-item pairs; (iii) User-user co-occurrence information is probably not very informative to help recommendation.

For opinion (i), we compare UltraGCN with the following variants to show the effectiveness of our designs in UltraGCN:

- UltraGCN($\lambda = 0, \gamma = 0$): when setting λ and γ to 0, UltraGCN is simply reduced to MF training with BCE loss function, which does not leverage graph information and cannot capture higher-order collaborative signals.
- UltraGCN($\gamma = 0$): this variant is identical to UltraGCN$_{Base}$, which only learns on the user-item graph and lacks more effective learning for item-item relationships.
- UltraGCN($\lambda = 0$): this variant lacks the graph convolution ability for learning on the user-item graph to more deeply mine the collaborative information.

Results are shown in Figure 2. We have the following observations:

(1) UltraGCN($\gamma = 0$) and UltraGCN($\lambda = 0$) all perform better than UltraGCN($\lambda = 0, \gamma = 0$), demonstrating that the designs of UltraGCN can effectively learn on both the user-item graph and item-item graph to improve recommendation; (2) Relatively, UltraGCN($\lambda = 0$) is inferior to UltraGCN($\gamma = 0$), indicating that user-item relationships may be better modeled than item-item relationships in UltraGCN; (3) UltraGCN performs much better than all the other three variants, demonstrating that our idea to disassemble various relationships, eliminate uninformative things which may disturb the model learning, and finally conduct multi-task learning in a clearer way, is effective to overcome the limitations (see Section 2.2) of previous GCN-based CF models.

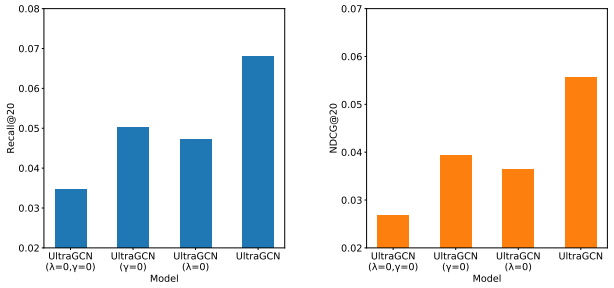
For opinion (ii), we change $\mathcal{L}_I$ to $\mathcal{L}'_I$:

$$\mathcal{L}'_I = -\sum_{(u,i)\in N^+} \sum_{j\in S(i)} \omega_{i,j} \log(\sigma(e_i^\top e_j)) \tag{19}$$

which is instead to optimize between the target positive item and its most K similar items. We compare the performance of UltraGCN using $\mathcal{L}_I$ and $\mathcal{L}'_I$ respectively with careful parameters tuning. Results are shown in Figure 3. It is clear that no matter incorporating $\mathcal{L}_C$ or

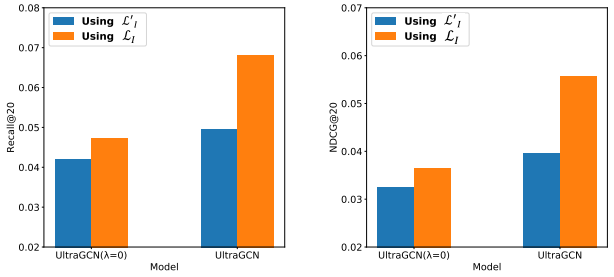
Table 5: Performance comparison of whether learning on the user-user co-occurrence graph.

Model	UltraGCN($\gamma = 0$)		UltraGCN($\lambda = 0$)		UltraGCN	
	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20
w/o $\mathcal{L}_U$	0.0504	0.0393	0.0472	0.0364	0.0681	0.0556
with $\mathcal{L}_U$	0.0513	0.0399	0.0470	0.0364	0.0683	0.0559



(a) Recall@20 (b) NDCG@20

Figure 2: Performance comparison of variants of UltraGCN.

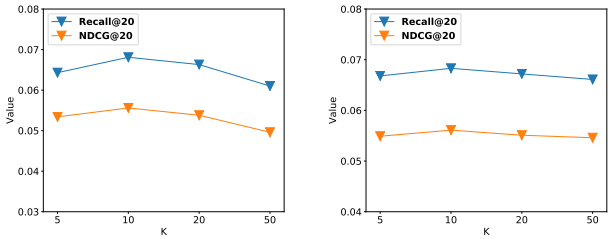


(a) Recall@20 (b) NDCG@20

Figure 3: Performance comparison of using $\mathcal{L}'_I$ and $\mathcal{L}_I$.

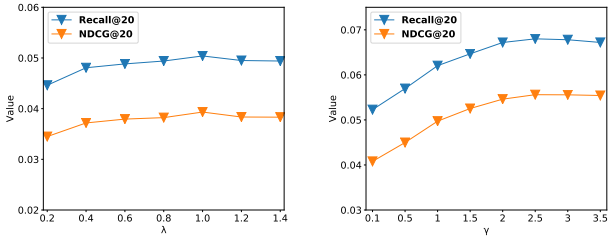
not, using $\mathcal{L}_I$ can achieves obvious better performance than using $\mathcal{L}'_I$, which proves that our designed strategy to learn on item-item graph is more effective. Furthermore, the performance gap between using $\mathcal{L}_I$ and using $\mathcal{L}'_I$ becomes large when incorporating $\mathcal{L}_C$, indicating that our strategy which makes the objective of UltraGCN unified can thus facilitate training and improve performance.

For opinion (iii), we derive the user-user constraint loss $\mathcal{L}_U$ with the similar method of Section 3.2 and combine it to the final objective. We carefully re-tune the parameters and show the comparison results of whether using $\mathcal{L}_U$ in Table 5. As can be seen, incorporating $\mathcal{L}_U$ to learn user-user relationships does not bring obvious benefits to UltraGCN. We attribute this phenomenon to the fact that the users' interests are broader than items' properties, and thus it is much harder to capture user-user relationships just from the user-user co-occurrence graph. Therefore, we do not introduce the modeling of user-user relationships into UltraGCN in this paper, and we will continue to study it in the future.



(a) Amazon-Book (b) Yelp2018

Figure 4: Performance comparison of setting different K.



(a) Impact of λ (b) Impact of γ

Figure 5: Performance comparison with different λ and γ.

4.5 Parameter Analysis

We investigate the influence of the number of selected neighbors K and the weights of the two constraint losses (i.e., λ and γ) on the performance for a better understanding of UltraGCN.

4.5.1 Impact of K. We test the performance of UltraGCN with different K in [5, 10, 20, 50] on Amazon-Book and Yelp2018. Figure 4 shows the experimental results. We can find that when K increases from 5 to 50, the performance shows a trend of increasing first and then decreasing. This is because that when K is 5, the item-item relationships are not sufficiently exploited. While when K becomes large, there may introduce some less similar or less confident item-item relationships into the learning process that affect model performance. Such phenomenon also confirms that conventional GCN-based CF models inevitably take into account too many low-confidence relationships, thus hurting performance.

4.5.2 Impact of λ and γ. We first set $\lambda = 0$ and show the performance of different λ from 0.2 to 1.4 (0.2 as the interval). Then we test with different γ in [0.1, 0.5, 1, 1.5, 2, 2.5, 3, 3.5] based on the best λ. Experiments are conducted on Amazon-Book, and we show results in Figure 5. For λ, we find that the small value limits the exertion of the user-item constraint loss, and a value of 1 or so would be suitable for λ. For the impact of γ, its trend is similar to λ but is more significant, and 2.5 is a suitable choice for γ. In general, our investigations for λ and γ show that these two parameters are important to UltraGCN, which can flexibly adjust the learning weights for different relationships and should be carefully set.

5 RELATED WORK

In this section, we briefly review some representative GNN-based methods and their efforts for model simplification toward recommendation tasks.

With the development and success of GNN in various machine learning areas, there appears a lot of excellent work in recommendation community since the interaction of users and items could be naturally formed to a user-item bipartite graph. Rianne van den Berg et al. [1] propose graph convolutional matrix completion (GC-MC), a graph-based auto-encoder framework for explicit matrix completion. The encoder of GC-MC aggregates the information from neighbors based on the types of ratings, and then combine it to the new embeddings of the next layer. It is the first work using graph convolutional neural networks for recommendation. Ying et al. [31] first applys GCN on web-scale recommender systems and propose an efficient GCN-based method named Pinsage, which combines efficient random walks and graph convolutions to generate embeddings of items that incorporate both graph structure as well as item feature information. Then, Wang et al. [27] design NGCF which is a new graph-based framework for collaborative filtering. NGCF has a crafted interaction encoder to capture the collaborative signals among users and items. Although NGCF achieves good performance compared with previous non-GNN based methods, its heavy designs limit its efficiency and full exertion of GCN. To model the diversity of user intents on items, Wang et al. [28] devise Disentangled Graph Collaborative Filtering (DGCF), which considers user-item relationships at the finer granularity of user intents and generates disentangled user and item representations to get better recommendation performance.

Although GNN-based recommendation models have achieved impressive performance, their efficiencies are still unsatisfactory when facing large-scale recommendation scenarios. How to improve the efficiency of GNNs and reserve their high performance for recommendation becomes a hot research problem. Recently, Dai et al. [6] and Gu et al. [8] extend fixed-point theory on GNN for better representation learning. Liu et al. [18] propose UCMF that simplifies GCN for the node classification task. Wu et al. [29] find the non-necessity of nonlinear activation and feature transformation in GCN, proposing a simplified GCN (SGCN) model by removing these two parts. Inspired by SGC, He et al. [10] devise LightGCN for recommendation by removing nonlinear activation and feature transformation too. However, its efficiency is still limited by the time-consuming message passing. Qiu et al. [21] demonstrate that many network embedding algorithms with negative sampling can be unified into the MF framework which may be efficient, however, their performances still have a gap between that of GCNs. We are inspired by these instructive studies, and propose UltraGCN for both efficient and effective recommendation.

6 CONCLUSION

In this work, we propose an ultra-simplified formulation of GCN, dubbed UltraGCN. UltraGCN skips explicit message passing and directly approximate the limit of infinite message passing layers. Extensive experimental results demonstrate that UltraGCN achieves impressive improvements over the state-of-the-art CF models in terms of both accuracy and efficiency.

7 ACKNOWLEDGEMENT

This work was supported in part by the National Natural Science Foundation of China (61972219), the Research and Development Program of Shenzhen (JCYJ20190813174403598, SGDX20190918101201696), the National Key Research and Development Program of China (2018YFB1800601), and the Overseas Research Cooperation Fund of Tsinghua Shenzhen International Graduate School (HW2021013).

8 APPENDIX

To further demonstrate the effectiveness of UltraGCN, we additionally provide the results compared to some more recent state-of-the-art CF models, including NBPO [33], BGCF [23], SCF [34], LCFN [32], and SGL-ED [30]. For simplicity and fairness of comparison, we use the same dataset and evaluation protocol provided by each paper. We also duplicate the results reported in their papers to keep consistency. The results in Table 6 again validate the effectiveness of UltraGCN, which outperforms the most recent CF models by a large margin.

Table 6: Performance comparison with some more models, including SCF, LCFN, NBPO, BGCF, and SGL-ED.

Movielens-1M		
Model	F1@20	NDCG@20
NGCF	0.1582	0.2511
SCF	0.1600	0.2560
LCFN	<u>0.1625</u>	<u>0.2603</u>
UltraGCN	**0.2004**	**0.2642**
Improv.	23.3%	1.5%

Amazon-Electronics		
Model	F1@20	NDCG@20
MF-BPR	0.0275	0.0680
ENMF	<u>0.0314</u>	<u>0.0823</u>
NBPO	0.0313	0.0810
UltraGCN	**0.0330**	**0.0829**
Improv.	5.1%	0.7%

Amazon-CDs		
Model	Recall@20	NDCG@20
NGCF	0.1258	0.0792
NIA-GCN	0.1487	0.0932
BGCF	<u>0.1506</u>	<u>0.0948</u>
UltraGCN	**0.1622**	**0.1043**
Improv.	7.7%	10.0%

Amazon-Books		
Model	Recall@20	NDCG@20
NGCF	0.0344	0.0263
LightGCN	0.0411	0.0315
SGL-ED	<u>0.0478</u>	<u>0.0379</u>
UltraGCN	**0.0681**	**0.0556**
Improv.	42.5%	46.7%

REFERENCES

[1] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2018. Graph Convolutional Matrix Completion. In *KDD'18 Deep Learning Day*.

[2] Sebastian Bruch, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2019. An Analysis of the Softmax Cross Entropy Loss for Learning-to-Rank with Binary Relevance. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval (SIGIR)*. 75–78.

[3] Chong Chen, Min Zhang, Yongfeng Zhang, Yiqun Liu, and Shaoping Ma. 2020. Efficient Neural Matrix Factorization without Sampling for Recommendation. *ACM Transactions on Information Systems (TOIS)* 38, 2 (2020), 1–28.

[4] 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 (AAAI)*. 27–34.

[5] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and Deep Graph Convolutional Networks. In *International Conference on Machine Learning (ICML)*. 1725–1735.

[6] Hanjun Dai, Zornitsa Kozareva, Bo Dai, Alex Smola, and Le Song. 2018. Learning Steady-states of Iterative Algorithms over Graphs. In *International Conference on Machine Learning (ICML)*. 1106–1114.

[7] Aditya Grover and Jure Leskovec. 2016. Node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 855–864.

[8] Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. 2020. Implicit Graph Neural Networks. *Advances in Neural Information Processing Systems (NeurIPS)* (2020), 11984–11995.

[9] Ruining He and Julian McAuley. 2016. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In *Proceedings of the 25th International Conference on World Wide Web (WWW)*. 507–517.

[10] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yong-Dong 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 (SIGIR)*. 639–648.

[11] 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 (WWW)*. 173–182.

[12] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative Metric Learning. In *Proceedings of the 26th International Conference on World Wide Web (WWW)*. 193–201.

[13] Shuyi Ji, Yifan Feng, Rongrong Ji, Xibin Zhao, Wanwan Tang, and Yue Gao. 2020. Dual Channel Hypergraph Collaborative Filtering. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (SIGKDD)*. 2020–2029.

[14] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

[15] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37.

[16] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 32. 3538–3545.

[17] Kang Liu, Feng Xue, and Richang Hong. 2020. RGCF: Refined Graph Convolution Collaborative Filtering with Concise and Expressive embedding. *arXiv preprint arXiv:2007.03383* (2020).

[18] Qiang Liu, Haoli Zhang, and Zhaocheng Liu. 2020. Simplification of Graph Convolutional Networks: A Matrix Factorization-based Perspective. *arXiv preprint arXiv:2007.09036* (2020).

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems (NeurIPS)*. 3111–3119.

[20] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 701–710.

[21] 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 (WSDM)*. 459–467.

[22] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI)*. 452–461.

[23] Jianing Sun, Wei Guo, Dengcheng Zhang, Yingxue Zhang, Florence Regol, Yaochen Hu, Huifeng Guo, Ruiming Tang, Han Yuan, Xiuqiang He, et al. 2020. A Framework for Recommending Accurate and Diverse Items Using Bayesian Graph Convolutional Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (SIGKDD)*. 2030–2039.

[24] 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 (SIGIR)*. 1289–1298.

[25] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale Information Network Embedding. In *Proceedings of the 24th International Conference on World Wide Web (WWW)*. 1067–1077.

[26] Menghan Wang, Yujie Lin, Guli Lin, Keping Yang, and Xiao-Ming Wu. 2020. M2GRL: A Multi-task Multi-view Graph Representation Learning Framework for Web-scale Recommender Systems. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (SIGKDD)*. 2349–2358.

[27] 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 (SIGIR)*. 165–174.

[28] Xiang Wang, Hongye Jin, An Zhang, Xiangnan He, Tong Xu, and Tat-Seng Chua. 2020. Disentangled Graph Collaborative Filtering. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 1001–1010.

[29] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *International Conference on Machine Learning (ICML)*. 6861–6871.

[30] 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 (SIGIR)*. 726–735.

[31] 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 *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (SIGKDD)*. 974–983.

[32] Wenhui Yu and Zheng Qin. 2020. Graph Convolutional Network for Recommendation with Low-pass Collaborative Filters. In *International Conference on Machine Learning (ICML)*. 10936–10945.

[33] Wenhui Yu and Zheng Qin. 2020. Sampler Design for Implicit Feedback Data by Noisy-label Robust Learning. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 861–870.

[34] Lei Zheng, Chun-Ta Lu, Fei Jiang, Jiawei Zhang, and Philip S Yu. 2018. Spectral Collaborative Filtering. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys)*. 311–319.