Alleviating Over-Smoothing in LightGCN for Improved Graph Representation Learning in Recommender Systems

Project Group 14

Yeo Ming Jie, Jonathan Chen Pu Shi Qin

***Abstract*—Graph Convolution Networks (GCNs) have emerged as a promising approach for representation learning in graph data, particularly in recommender systems. The LightGCN model, a simplified version of GCNs, has demonstrated improved performance in collaborative filtering tasks compared to its predecessor, NGCF. However, LightGCN still suffers from the issue of over-smoothing, which can result in reduced accuracy and performance degradation at higher graph convolution layers. Additionally, LightGCN faces challenges related to high computational costs and poor scalability, especially in large-scale recommendation scenarios. In this project, we propose novel approaches to alleviate over-smoothing in LightGCN to enhance its performance in graph representation learning for recommender systems. By exploring solutions such as cluster-based sampling and dimensionality reduction, our aim is to enhance the performance of LightGCN in graph representation learning for recommender systems, while addressing the limitations and challenges posed by over-smoothing.**

I. Introduction

Recommender systems leverage data and machine learning algorithms to provide personalized content recommendations. Collaborative Filtering (CF) [1] is a widely-used method in recommender systems, with classical approaches such as matrix factorization (e.g., ALS [2] and SVD-based [3]) and more recent deep learning methods like NeuMF [4]. However, these methods face limitations, such as insufficiently capturing complex relationships between users and items, especially when incorporating external structured information like social relationships or item knowledge graphs. They also depend heavily on the quality and quantity of user-item interactions and often perform poorly in the presence of sparse data.

Graph Convolution Networks (GCNs) have emerged as a promising approach in recommender systems due to their effectiveness in representation learning on graph data and the natural modeling of interaction data in recommendation systems as graphs [5]. User-item recommendations can be represented as a bipartite graph, with nodes representing users and items and links representing their interactions. Unlike traditional methods, GCNs can explicitly encode collaborative signals, such as topological structures, to improve user and item representations. GCNs have been successfully applied to Web-scale applications, as demonstrated by models like PinSage [6], M2GRL [7], and NGCF [8].

Among these models, LightGCN [9] has garnered significant attention for its efforts to simplify GCNs for collaborative filtering tasks while delivering performance improvements compared to its predecessor, NGCF. However, LightGCN suffers from several issues, including over-smoothing, which causes a decline in recommendation accuracy at higher graph convolution layers, and poor scalability, which can lead to slow convergence, especially with increasing graph size. In this paper, we focus on addressing the over-smoothing issue in LightGCN by exploring two approaches: 1) Cluster-Based Sampling, and 2) a novel method of performing dimensionality reduction on the user-item interaction graph by collapsing the user-item representation. Lastly, we further investigate the use of MixGCF [10], a complementary training approach and state-of-the-art plug-in for graph-based recommendation systems.

II. Graph Neural Networks (GNNs)

In this section, we first review the fundamental definitions and concepts of Graph Neural Networks (GNNs) [11] and then provide a focused introduction to Graph Convolution Networks (GCNs), which are widely adopted in the field of recommendation.

A. *Mathematical Preliminaries of Graph Neural Networks*

A graph is represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. Let $v_i \in \mathcal{V}$ be a node and $e_{ij} = (v_i, v_j) \in \mathcal{E}$ be an edge pointing from v_j to v_i. The neighborhood of a node v is denoted as $\mathcal{N}(v) = \{u \in \mathcal{V} \mid (v, u) \in \mathcal{E}\}$.

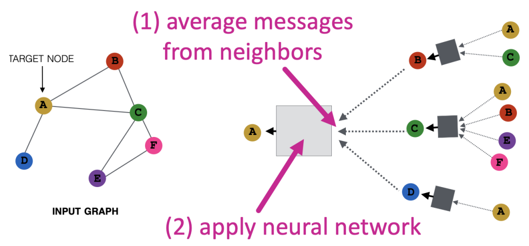

Fig. 1. Main idea of GNNs: Message Aggregation

GNNs iteratively aggregate information from a target node's neighboring nodes to obtain the representation of each node in the graph. The process consists of two operations in a propagation layer: aggregation and update. The aggregation operation calculates an intermediate representation $\mathbf{n}_v^{(l)}$ of node v as a function of its neighbors' representations $\mathbf{h}_u^{(l)}$, where u is a neighbor of v:

$$\textbf{Aggregation: } \mathbf{n}_v^{(l)} = \textit{Aggregator}_l\left(\left\{\mathbf{h}_u^{(l)}, \forall u \in \mathcal{N}_v\right\}\right), \quad (1)$$

The update operation uses the previous representation $\mathbf{h}_v^{(l)}$ of node v and the intermediate representation $\mathbf{n}_v^{(l)}$ to obtain the updated representation $\mathbf{h}_v^{(l+1)}$:

$$\textbf{Update: } \mathbf{h}_v^{(l+1)} = \textit{Updater}_l\left(\mathbf{h}_v^{(l)}, \mathbf{n}_v^{(l)}\right). \quad (2)$$

Various GNN frameworks, such as GraphSAGE [12] and Graph Attention Networks (GATs) [13], have been proposed, each with unique aggregation and update rules. In the context of recommender systems, predicting new user-item interactions can be framed as a link prediction problem. This paper focuses on Graph Convolution Networks (GCNs), a framework widely adopted for recommendation tasks.

B. Graph Convolution Networks (GCNs)

GCNs generalize the operation of convolution from uniform grid data to graphs. The primary goal is to generate a node representation, $\mathbf{h}_v$, by aggregating its features with its neighbors. Mathematically, the neighborhood aggregation and update rule in GCNs can be expressed as follows:

$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \sum_{u \in \mathcal{N}(v)} \frac{\mathbf{h}_u^{(l-1)}}{|\mathcal{N}(v)|}\right), \quad (3)$$

where $\sigma(\cdot)$ is the non-linear activation function (e.g., using a Rectified Linear Unit, ReLU), and $\mathbf{h}_v^{(l)}$ and $\mathbf{W}^{(l)}$ denote the representation of node v and the learnable weight matrix at the l-layer, respectively. The following section introduces LightGCN, a GCN model designed explicitly for our recommendation problem. Introduced by He et al. [9], LightGCN has gained recognition for its simplicity and powerful performance in recommendation tasks.

III. LightGCN

A. Embedding Propagation Layers

In LightGCN, He et al. propose the removal of feature transformations and non-linear activations in favor of a more simplified model retaining only the core component of neighborhood aggregation. The graph convolution operation (or propagation rule) can thus be expressed as follows:

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

where $e_u^{(k)}$ and $e_u^{(k)}$ denotes the embeddings of user u and item i at the k-th layer, while $\mathcal{N}(u)$ and $\mathcal{N}(i)$ represent the corresponding neighbor node sets. The equivalent matrix representation of the above propagation rule is given by:

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

where $\mathbf{A}$, $\mathbf{D}$ and $\mathbf{E}^{(k)}$ are the adjacency matrix, the diagonal node degree matrix, and the k-th layer embedding matrix, respectively. Note that the adjacency matrix $\mathbf{A}$ has the following form by nature of the bipartite graph structure

$$\mathbf{A} = \begin{pmatrix} \mathbf{0} & \mathbf{R} \\ \mathbf{R}^T & \mathbf{0} \end{pmatrix}, \quad (6)$$

where $\mathbf{R} \in \mathbb{R}^{M \times N}$ represents the user-item interaction matrix, with M and N indicating the number of users and items, respectively. The matrix entry $R_{ui} = 1$ if user u has engaged with item i, and it is zero otherwise. The final embedding matrix used for model prediction is thus given by

$$\begin{aligned} \mathbf{E} &= \alpha_0\mathbf{E}^{(0)} + \alpha_1\mathbf{E}^{(1)} + \alpha_2\mathbf{E}^{(2)} + \ldots + \alpha_K\mathbf{E}^{(K)} \\ &= \alpha_0\mathbf{E}^{(0)} + \alpha_1\tilde{\mathbf{A}}\mathbf{E}^{(0)} + \alpha_2\tilde{\mathbf{A}}^2\mathbf{E}^{(0)} + \ldots + \alpha_K\tilde{\mathbf{A}}^K\mathbf{E}^{(0)}, \end{aligned} \quad (7)$$

where

$$\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} \quad (8)$$

is the symmetrically normalized matrix. Note that graph convolution is a special form of Laplacian smoothing [5], and as such, $\tilde{\mathbf{A}}$ represents the Laplacian matrix for the user-item graph.

B. Prediction Layer

Following K layers of graph convolution operations, the final representations of users and items are given by a linear combination of the embeddings obtained at each layer:

$$\mathbf{e}_u = \sum_{k=0}^{K} \alpha_k \mathbf{e}_u^{(k)}; \quad \mathbf{e}_i = \sum_{k=0}^{K} \alpha_k \mathbf{e}_i^{(k)}, \quad (9)$$

where $\alpha_k \geq 0$ are weight hyper-parameters denoting the importance of the k-th layer embeddings. Uniform weights $\alpha_k = 1/(K+1)$ are chosen given the stable performance yielded and the maintained simplicity of the model. Lastly, the model prediction output is given by the inner product of the final user-item representations:

$$\hat{y}_{ui} = \mathbf{e}_u^T \mathbf{e}_i, \quad (10)$$

where $\hat{y}_{ui}$ denotes the predicted rating of user u for item i.

C. Training of Recommendation Model

The sole parameters to be trained in the model are the initial layer embeddings, specifically, $\mathbf{e}_u^{(0)}$ for every user and $\mathbf{e}_i^{(0)}$ for each item. These parameters are trained by employing the Bayesian Personalized Ranking (BPR) loss [14], a pairwise loss that encourages predicting an observed interaction to be higher than unobserved ones and penalizes otherwise.

$$L_{BPR} = -\sum_{u=1}^{M} \sum_{i \in \mathcal{N}_u} \sum_{j \notin \mathcal{N}_u} \ln \sigma \left(\hat{y}_{ui} - \hat{y}_{uj} \right) + \lambda \left\| \mathbf{E}^{(0)} \right\|^2, \tag{11}$$

where λ denotes the regularization strength parameter. In summary, the overall propagation-prediction model architecture of the LightGCN model is shown in Figure 2.

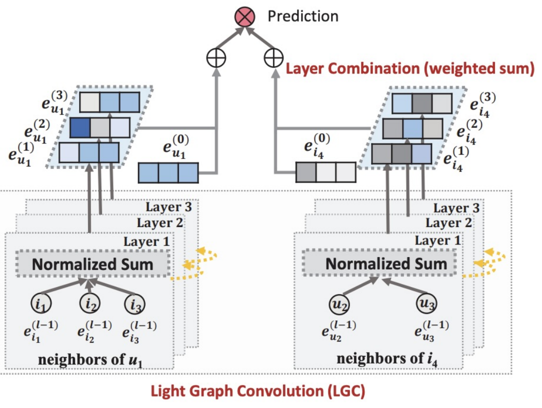

Fig. 2. An illustration of LightGCN model architecture.

D. Successes

Empirically, the LightGCN model has improved performance over traditional GCNs on evaluation metrics commonly used in recommender systems, achieving a relative improvement of about 16% on average compared to the NGCF model. The improved performance of LightGCN, combined with its ease of training, has led to its recognition as a state-of-the-art recommendation model and its widespread use as a benchmark in the research community of GNNs and Recommender Systems. Additionally, LightGCN's versatility has been demonstrated through its adaptation for various other applications, including social recommendation systems [15] and time series tasks such as stock prediction [16].

E. Limitations of LightGCN

Despite these successes, LightGCN has its shortcomings. Its message-passing operation may not effectively capture the relationships between items and users. The weight assignment in its item-item and user-user modeling may not reflect the varying importance of relationships. This can result in uninformative, noisy, and ambiguous relationships, reducing the model's accuracy.

Furthermore, the LightGCN model faces challenges such as high computational costs, poor scalability, over-smoothing, declining performance with increasing layers, and a need for further optimization in training. The core component of LightGCN, neighborhood aggregation, requires repeated multiplication of a large adjacency matrix, leading to high computational costs and poor scalability, especially with increasing graph size. For example, Kelong et al. [17] found that a three-layer LightGCN required more than 700 epochs for convergence on the Amazon-Books dataset, which may be deemed unacceptably long for industry standards. Over-smoothing, a common issue in GCNs, occurs when repeated graph convolutions make node embeddings indistinguishable, causing a decline in recommendation accuracy and limiting the ability of the model to capture higher-order collaborative signals. Empirical evidence suggests that the performance of LightGCN starts to decline after 2 or 3 layers. Finally, He et al. [9] have also noted that there was room for further optimizations in training through the introduction of methods such as negative sampling and adversarial sampling.

IV. Over-Smoothing Problem

The problem of over-smoothing was first formally proposed by Li et al. [18], who observed that with the repeated application of Laplacian smoothing during graph convolution, features of nodes in the graph would converge to similar values, eventually rendering node embeddings to become indistinguishable and adversely affecting the performance of GCN models at deeper layers. Intuitively, this phenomenon can be explained using the concept of receptive fields and receptive field overlap [19].

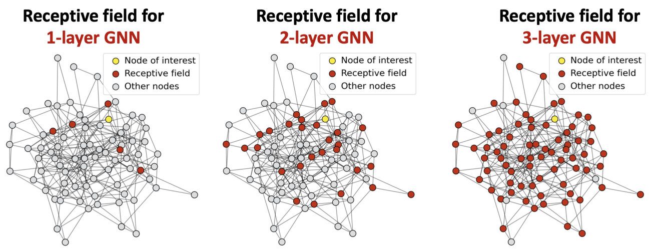
Receptive field for 1-layer GNN
Receptive field for 2-layer GNN
Receptive field for 3-layer GNN
Node of interest
Receptive field
Other nodes

Fig. 3. Receptive fields for multi-layer GNN

In the context of GNNs, the receptive field of a node v is defined as the set of all its L-hop neighbors, i.e., the nodes that are reachable from v within L hops. These neighbors can be viewed as the set of nodes that affect the representation of v after L layers of graph convolution. As the number of layers increases, the receptive fields of nodes expand, allowing nodes further apart in the graph to influence each other's embeddings. However, as more layers are added, the receptive fields of different nodes begin to overlap significantly. Each node aggregates information over the same set of nodes at message aggregation, eventually leading to indistinguishable node representations.

A. Traditional Approaches to Over-Smoothing Problem

In summary, adding more layers to GNNs does not yield the same performance improvements observed in traditional deep neural networks. This limitation restricts GNNs' ability to capture higher-order collaborative signals. Traditional approaches to addressing over-smoothing focus on enhancing the expressiveness of GNNs [20]. Common methods include:

- increasing the expressive power within each GNN layer,
- incorporating non-message passing layers such as pre- or post-processing layers [21], and
- employing skip connections to bypass layers [22].

These techniques mitigate the over-smoothing issue while maintaining the network's ability to learn complex patterns and representations. It is important to note that this is an active research field, with numerous researchers investigating the problem and proposing various improvement methods. However, there has yet to be a unified framework that conclusively proves the effectiveness of these methods.

V. Problem Statement

In this project, our focus is on addressing the limitations related to high computational costs and over-smoothing to improve the performance of LightGCN in graph-based recommendation tasks.

VI. Related Works

Since the introduction of the LightGCN model, several works have been conducted to address its limitations, particularly with respect to over-smoothing. Kelong et al. [17] proposed UltraGCN, an improvement on LightGCN that skips explicit message passing and directly trains embeddings with a constraint loss approximating infinite graph convolution layers. Consequently, UltraGCN demonstrates better training efficiency and performance than LightGCN and other state-of-the-art models. In another work, Shaowen et al. [23] presented SVD-GCN, which combines the singular value decomposition (SVD) method with LightGCN to enhance recommendation performance. This approach reduces the number of trained parameters and requires significantly fewer epochs to converge (8 epochs compared to 700 epochs for LightGCN). Lastly, Xiyao et al. [24] proposed using contrastive learning and collaborative training to improve the accuracy and diversity of the LightGCN model, further contributing to the ongoing efforts to address over-smoothing and other limitations in this area.

In light of UltraGCN and the other literature's approach to over-smoothing, our efforts aim to complement and extend the existing body of work. We seek to explore alternative strategies for alleviating oversmoothing specific to the LightGCN model architecture, which may offer additional benefits or synergies when combined with existing techniques. By investigating these methods and their potential impact on the LightGCN model, we aim to achieve a deeper understanding of the over-smoothing issue and the LightGCN's model architecture.

VII. Proposed Approach

In the following section, we present two approaches that we have explored to address the over-smoothing problem specific to the LightGCN model.

A. *Cluster-Based Sampling*

The first approach we explored was inspired by the ClusterGCN model proposed by Chiang et al. [25], which adopts cluster-based sampling to partition a large input graph into clusters. This method leverages the observation that many real-world graphs exhibit community structures, which can be used to improve the performance of GNNs.

1) ClusterGCN: In traditional full-batch GNNs, all node embeddings are updated together using embeddings of the previous layer, which leads to a computational bottleneck for large input graphs. To overcome this limitation, we employ graph clustering algorithms to partition the graph into clusters that better capture its community structure. Graph clustering methods, such as Metis [26] and Graclus [27], construct partitions over the vertices in the graph such that the number of links within clusters is significantly greater than the number of between-cluster links. This approach helps to preserve the clustering and community structure of the graph. These clusters are sampled and aggregated to form an induced sub-graph. Batch training of the GCN is then executed over the sub-graph at each iteration.

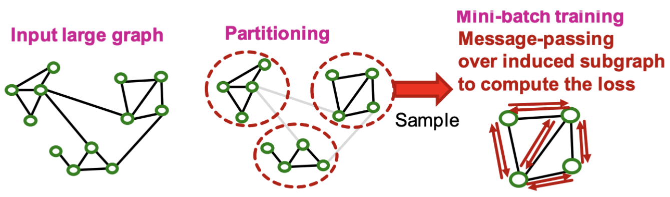

Fig. 4. Cluster-Based Sampling approach

ClusterGCN was primarily designed to address scalability issues in graph convolutional networks by enabling efficient mini-batch training for large graphs. With their proposed framework and model, Chiang et al. demonstrated improvements in scalability and increased memory efficiency compared to traditional full-batch GNNs.

2) Relationship to Over-Smoothing: Partitioning the graph minimizes the receptive field overlap between nodes, as each cluster retains a distinct portion of the graph's structure. This strategy helps preserve the unique features of individual nodes and reduce the over-smoothing issue by limiting information propagation to a local neighborhood within each cluster. Consequently, the resulting embeddings are more expressive and capture finer-grained information about the graph's topology, improving performance on downstream tasks such as recommendations. Empirically, deeper models were successfully trained using the cluster-based sampling approach, achieving state-of-the-art results. This not only highlights the effectiveness of the ClusterGCN model in handling large-scale graphs but also suggests that cluster-based sampling can help mitigate the over-smoothing issue commonly encountered in GNNs with increasing depth.

3) Our approach: Therefore, our approach aims to integrate the cluster-based sampling technique into our graph pre-processing and training. Instead of using the ClusterGCN model, we will employ the LightGCN model for propagation, leveraging the benefits of cluster-based sampling while maintaining the advantages of the LightGCN framework.

B. Novel Proposed Method: Dimensionality Reduction

In many recommender systems, summing a user's historical item embeddings effectively represents the user [4]. Notably, LightGCN simplifies GCN's message-passing, leading to improved performance. Inspired by these observations, we propose a novel dimensionality reduction approach for the user-item interaction graph.

1) SimpleLGN: Here, we introduce our proposed framework, which shall be termed as Simple LGN, where we represent user embeddings by item embeddings before performing propagation using LightGCN. Recall the original LightGCN propagation rule in matrix form given by (7). Consider the scenario of a 4-layer LightGCN model. For convenience, let scalar coefficients $\alpha_k = 1$ for all k. The expansion is thus given by:

$$\mathbf{E}^{(4)} = \mathbf{E}^{(0)} + \tilde{\mathbf{A}}\mathbf{E}^{(0)} + \tilde{\mathbf{A}}^2\mathbf{E}^{(0)} + \tilde{\mathbf{A}}^3\mathbf{E}^{(0)} + \tilde{\mathbf{A}}^4\mathbf{E}^{(0)} \quad (12)$$

Let $\tilde{\mathbf{R}}$ denote the symmetrically normalized user-item interaction matrix. Note that the 0-th layer embedding matrix, $\mathbf{E}^{(0)} \in \mathbb{R}^{(M+N)\times T}$ for some embedding size T, can be represented as a concatenation of both the user embedding matrix $U \in \mathbb{R}^{M\times T}$ and item embedding matrix $I \in \mathbb{R}^{N\times T}$:

$$\mathbf{E}^{(0)} = \begin{pmatrix} \mathbf{U}^{(0)} \\ \mathbf{I}^{(0)} \end{pmatrix}. \quad (13)$$

Then, by block matrix multiplication, the corresponding final user-item embedding matrices are given by:

$$\begin{aligned}\mathbf{U} = \mathbf{U}^{(0)} &+ \tilde{\mathbf{R}}\mathbf{I}^{(0)} + \tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\mathbf{U}^{(0)} \\ &+ \tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\mathbf{I}^{(0)} + \tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\mathbf{U}^{(0)}\end{aligned} \quad (14)$$

$$\begin{aligned}\mathbf{I} = \mathbf{I}^{(0)} &+ \tilde{\mathbf{R}}^T\mathbf{U}^{(0)} + \tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\mathbf{I}^{(0)} \\ &+ \tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\mathbf{U}^{(0)} + \tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\mathbf{I}^{(0)}\end{aligned} \quad (15)$$

Under the original LightGCN framework, both the user and item embeddings are initialized separately using a random Gaussian distribution. However, under our proposed framework, we let

$$\mathbf{U}^{(0)} = \tilde{\mathbf{R}}\mathbf{I}^{(0)}, \quad (16)$$

where only the item embedding matrix $\mathbf{I}^{(0)}$ is initialized and trained. With the proposed representation, the final user embedding matrix can thus be fully expressed in terms of the item embedding matrix in the propagation, given by:

$$\mathbf{U} = 2\tilde{\mathbf{R}}\mathbf{I}^{(0)} + 2\tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\mathbf{I}^{(0)} + \tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\mathbf{I}^{(0)} \quad (17)$$

$$\mathbf{I} = \mathbf{I}^{(0)} + 2\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\mathbf{I}^{(0)} + 2\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\mathbf{I}^{(0)} \quad (18)$$

2) Intuition for Proposed Formulation: The proposed operation has strong interpretability and suggests that a user's historical item interactions play the most influential role in predicting their subsequent item preferences. Additionally, maintaining two separate trainable embedding matrices for users and items may lead to information overlap, which should be minimized to prevent over-smoothing. By representing user embeddings through item embeddings, our approach reduces overlap and mitigates the over-smoothing issue.

3) Weight Sharing Extension: Next, we consider a further generalization in the coefficients of our proposed formulation, given by:

$$\mathbf{U} = \alpha_0\tilde{\mathbf{R}}\mathbf{I}^{(0)} + \alpha_1\tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\mathbf{I}^{(0)} + \alpha_2\tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\mathbf{I}^{(0)} \quad (19)$$

$$\mathbf{I} = \alpha_0\mathbf{I}^{(0)} + \alpha_1\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\mathbf{I}^{(0)} + \alpha_2\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}\mathbf{I}^{(0)} \quad (20)$$

As mentioned earlier, we can obtain a simplified message-passing function as given in formulas (16) and (17) under the assumption that $\mathbf{U}^{(0)} = \tilde{\mathbf{R}}\mathbf{I}^{(0)}$. However, it is evident that the difference between every coefficient of terms in user embedding and item embedding—which implies that the weights of user's representation in every layer differ from those of the items—is unreasonable in real life. Therefore, we attempt to unify the coefficients of terms in user embedding and item embedding. Ultimately, the final user embedding can be represented by the interaction matrix multiplied by the final item embedding:

$$\mathbf{U} = \tilde{\mathbf{R}}\mathbf{I} \quad (21)$$

Furthermore, the various weight mechanisms are unproductive for message passing in LightGCN. After simplification, we can explore assigning different weights to the terms of item embedding and user embedding.

VIII. Experimental Set-Up

A. Dataset Description

In this project, we perform experiments on three classic benchmark datasets for GNN-based recommendation systems: the MovieLens (ML)-100k [28], Gowalla [29], and the Yelp2018 dataset [30].

- **ML-100k:** This small-scale dataset consists of user-item rating pairs, movie attributes, tags, and user demographic features and is widely used as a benchmark dataset for collaborative filtering and knowledge graph-based recommendation tasks.
- **Gowalla:** This medium-scale dataset is obtained from Gowalla, where users share their locations by checking in. It is commonly used for Point-of-interest (POI) recommendation, user-item collaborative filtering, and sequential recommendation. The dataset helps understand user behavior patterns in location-based services.
- **Yelp2018:** This large-scale dataset contains user check-ins, with local businesses like restaurants and bars as items. It is widely used in user-item collaborative filtering and is ideal for evaluating recommendation algorithms in real-world settings. The dataset's size allows for testing our method's scalability and robustness.

The statistics of the datasets are summarized in Table I.

More specific details on data pre-processing are given in the next section for each experiment respectively.

TABLE I
STATISTICS OF THE DATASETS

Dataset	# User	# Item	# Interaction	Density
ML-100k	610	9724	100,836	0.01699
Gowalla	29,858	40,981	1,027,370	0.00084
Yelp2018	31,668	38,048	1,561,406	0.00130

B. Evaluation Metrics

The following evaluation metrics will be adopted to assess the performance of compared methods in recommendation: Recall@K, Precision@K and Normalized Discounted Cumulative Gain, NDCG@K [11]. By default, we set $K = 20$.

- **Recall@K** measures the proportion of relevant items found in the top-k recommendations, where

$$Recall@K = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|\mathcal{I}_{u,K}|}{|\mathcal{I}_u|}, \tag{22}$$

with $\mathcal{U}$, $\mathcal{I}_u$ and $\mathcal{I}_{u,K}$ denoting the test set of users, items and number of relevant items occurring in the top K terms of recommendation scores.
- **Precision@K** measures the fraction of the items the user will click among the recommended K items, where

$$Precision@K = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|\mathcal{I}_{u,N}|}{K}, \tag{23}$$

- **NDCG@K** differentiates the contributions of the accurately recommended items based on their ranking positions:

$$NDCG@K = \frac{1}{|\mathcal{U}|} \Sigma_{u \in \mathcal{U}} \frac{\sum_{k=1}^{K} \frac{I\left(R_k^K(u) \in T(u)\right)}{\log(k+1)}}{\sum_{k=1}^{K} \frac{1}{\log(k+1)}} \tag{24}$$

where $R_k^K(u)$ denotes the k^{th} item in the recommended list $R^K(u)$, and $T(u)$ denotes the ground truth item set.

IX. EXPERIMENTS

A. Experiment 1: Cluster-Based Sampling Approach

To investigate the effectiveness of the cluster-based sampling approach in addressing the over-smoothing issue, we conducted experiments using our custom implementation of LightGCN on the MovieLens100k dataset. Our custom implementation allowed us to modify the data pre-processing and mini-batch pipelines to test the cluster-based sampling approach with LightGCN.

We split the dataset into an 80-10-10 training-validation-test split and trained the original LightGCN model with $K = 3$ layers for $10,000$ iterations. Figure 5 shows the training and validation loss curves and the performance in recall, precision, and NDCG values on the validation set. The training time was approximately 12 minutes. As seen in the figure, the training and validation curves diverge, indicating overfitting.

Next, we implemented the cluster-based sampling approach in conjunction with LightGCN. We partitioned the original dataset into 100 clusters and randomly selected 10 to form

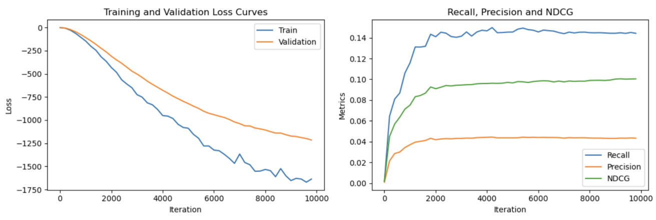

Fig. 5. Experiment 1: Training and validation loss curves, recall, precision, and NDCG values on the validation set of LightGCN with $K = 3$ layers.

an induced sub-graph at each iteration for mini-batch training. Note that the choice of partitions and clusters sampled are hyper-parameters for tuning. However, the chosen parameters yielded strong empirical results. This method resulted in a 50% reduction in training time.

Figure 6 shows the training and validation loss curves and the performance in recall, precision, and NDCG values on the validation set after applying the cluster-based sampling approach. As seen in the figure, the validation curve lies below the training curve, suggesting effective generalization to unseen data.

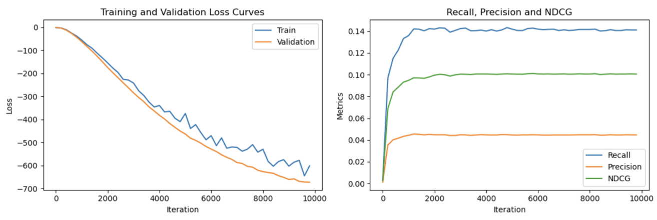

Fig. 6. Experiment 1: Training and validation loss curves, recall, precision, and NDCG values on the validation set of LightGCN with the cluster-based sampling approach.

The following Tables II and III consolidates both the model training time and corresponding model performance results over the test set conducted for $K = 2$ to $K = 5$ layers.

TABLE II
TRAINING TIME FOR BASELINE LIGHTGCN AND CLUSTER-LIGHTGCN

Model	Layers	Train Time (s)	Reduction (%)
Baseline LightGCN	2	569.90	-
	3	707.76	-
	4	774.28	-
	5	905.09	-
Cluster-LightGCN	2	338.05	40.70
	3	360.55	49.06
	4	380.65	50.85
	5	405.88	55.14

The results from Table III are plotted in Figure 7 for the test set, from 2 to 5 layers.

To conclude, our experimentation with Cluster-LightGCN yielded the following results:

TABLE III
PERFORMANCE METRICS FOR BASELINE LIGHTGCN AND CLUSTER-LIGHTGCN

Model	Layers	Recall	Precision	NDCG
Baseline LightGCN	2	0.12678	0.04629	0.09615
	3	0.12919	0.04420	0.08841
	4	0.11712	0.03832	0.08576
	5	0.11854	0.03995	0.08385
Cluster-LightGCN	2	0.12222	0.04565	0.09705
	3	0.11758	0.04502	0.09455
	4	0.11566	0.04366	0.09231
	5	0.12154	0.04511	0.09482

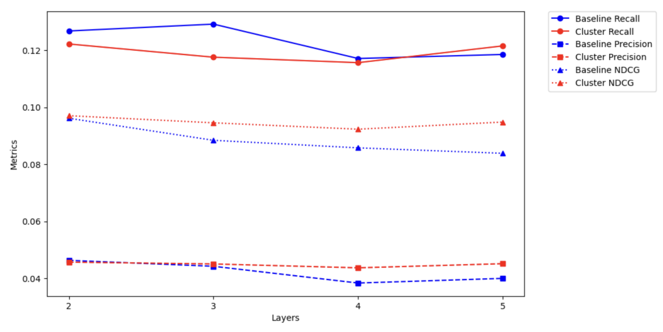

Fig. 7. Experiment 1: Precision and NDCG values on the test set of Baseline LightGCN and Cluster-LightGCN.

- **Training Time:** Cluster-LightGCN consistently exhibited faster training times than the Baseline LightGCN, demonstrating efficiency improvements.
- **Recall@20:** While the Baseline LightGCN performed better at lower layers, Cluster-LightGCN exhibits a more consistent trend across layers.
- **Precision@20:** Across all layer configurations, Cluster-LightGCN demonstrates slightly higher precision values than Baseline LightGCN models, suggesting improved recommendation accuracy.
- **NDCG@20:** Similarly, Cluster-LightGCN showcased slightly higher NDCG values than the Baseline LightGCN across all layer configurations, indicating that the recommended items are better ranked within the top 20 recommendations.

Based on the above findings, we conclude that Cluster-LightGCN offers significantly faster training times and consistent, better recommendation quality compared to the Baseline LightGCN, indicating the effectiveness of the proposed solution for alleviating over-smoothing.

B. Experiment 2: Dimensionality Reduction Approach

As highlighted in Section VII-B, we proposed reducing user dimension and implementing weight sharing based on LightGCN theoretically. In this section, we will demonstrate the rationality of the two operations and the superiority of our model in practice. To this end, we conduct experiments to:

- Compare the performance of LightGCN, SimpleLGN, and sharing-weight SimpleLGN (SWSLGN) .
- Analyze the results of the sharing-weight SimpleLGN and draw conclusions from them.

Experiments for this approach were conducted using a modified version of the original LightGCN source code. This modification allowed us to leverage the extensive data pipelines available for testing on both the Gowalla and Yelp2018 datasets with ease. The performance of LightGCN, SimpleLGN, and sharing-weight SimpleLGN are shown in Table IV. As the table indicates, the trainable sharing-weight SimpleLGN has superior performance compared to the original LGN.

TABLE IV
PRELIMINARY COMPARISON OF MODELS ON GOWALLA

Model	Recall	Precision	NDCG
LightGCN	0.179	0.055	0.153
SimpleLGN	0.174	0.052	0.146
Uniform-SWSLGN	0.169	0.049	0.140
Trainable-SWSLGN+	0.182	0.054	0.148

”+” indicates that a dropout layer was used in the respective model.

The Uniform SWSLGN is defined as the model (16) and ((17) with $\alpha_k = 1$ for all k. The Trainable SWSLGN is defined as the model (16) and (17) with α_k as trainable parameters.

As shown in Table IV, the SimpleLGN has similar performance to the LightGCN, indicating that the assumption $\mathbf{U}^{(0)} = \tilde{\mathbf{R}}\mathbf{I}^{(0)}$ is applicable for the task. Furthermore, the trainable sharing-weight SimpleLGN exhibits the highest performance among these four models. Its superior performance over LGN (with 1.6% higher recall) and Uniform SWSLGN (with 7.7% higher recall) suggests that the average weight for the terms in each layer is unreasonable and that there are appropriate weights for predicting interactions between users and items.

TABLE V
PRELIMINARY COMPARISON OF MODELS ON YELP2018

Model	Recall	Precision	NDCG
LightGCN	0.0623	0.0281	0.0514
SimpleLGN	0.0623	0.0272	0.0502
Uniform-SWSLGN	0.0602	0.0260	0.0480
Trainable-SWSLGN+	0.0645	0.0275	0.0512

We also conducted the same experiments on the Yelp2018 dataset and obtained similar conclusions as with the Gowalla dataset.

Focusing on the optimal model, the Trainable Sharing-weight SimpleLGN, we can derive several interesting insights from its experiments. As illustrated in Figure 8, we observe that the Trainable SWSLGN model experiences overfitting issues during training. However, employing dropout to prevent overfitting leads to an improvement in the Recall metric for the

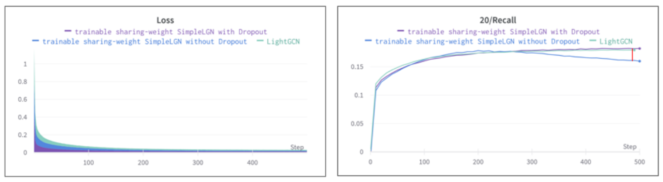

Fig. 8. Over-fitting in Trainable Sharing-weight SimpleLGN model. Inclusion of Dropout(p=0.6) layer results in recall improvement from 0.1598 to 0.1822.

trainable weight-sharing method in SimpleLGN. Additionally, this approach results in faster convergence of the model.

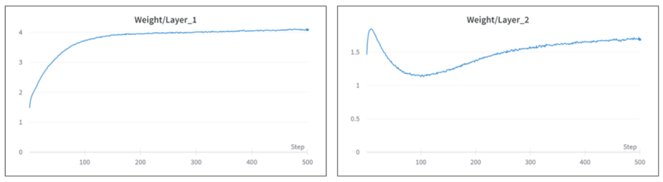

Fig. 9. Trainable Sharing-Weights Curve in Trainable SWSLGN+

Figure 9 demonstrates the convergence of the trainable weights towards constant values. The weights/coefficients converge to $\alpha_1 = 4$ and $\alpha_2 = 1.7$ (with fixed $\alpha_0 = 1$). This empirical observation supports our proposal to generalize the coefficients of embedding terms as a valid exploratory step in the development of our model.

X. Improving Training with Tuned-MixGCF

In this section, we introduce Tuned-MixGCF to improve the accuracy of our model and further boost the training efficiency. Tuned-MixGCF is a refinement of the state-of-the-art MixGCF model [10], which focuses on negative sampling in the context of GNN-based recommender systems. MixGCF employs a hop mixing technique to synthesize hard negative samples by selecting neighborhoods of raw negatives and aggregating their embeddings to form a synthetic negative one.

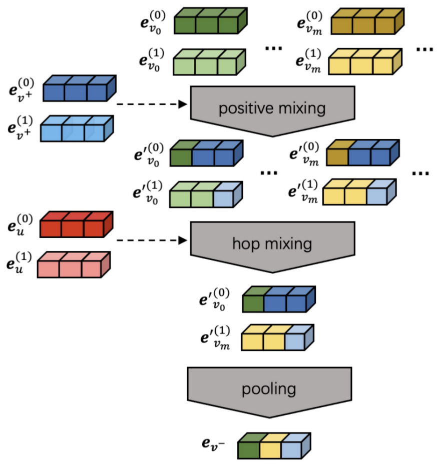

Fig. 10. Main structure of MixGCF

MixGCF consists of two main sub-models: positive mixing and hop mixing. In positive mixing, an interpolation mixing method is applied to inject information from positive samples into negative ones, making the negative samples harder to identify. In hop mixing, the model first uses a hard negative selection strategy to extract unique information from each of the previously generated hard negatives. It then employs a pooling operation (mean, sum, or concatenation) to combine the diverse information extracted from the synthesized negative items.

A. *Tuned-MixGCF*

In Tuned-MixGCF, we modify the positive mixing component and the BPR loss function to refine the original model. In the Positive Mixing part of the original model, we inject positive information e_{v+} into the negative embeddings with the following function:

$$e_{v_m}^{'(l)} = \alpha^{(l)} e_{v+}^{(l)} + (1-\alpha^{(l)}) e_{v_m}^{(l)}, \alpha \in (0,1) \tag{25}$$

where α is a random variable with the same dimensions as e. The authors also suggest that $\alpha^{(l)}$ could have come from some other family distributions like β-family distributions. However, we propose that if we view $\alpha(l)$ as a parameter that can be trained through the training process, rather than as a random variable, the following format can be used:

$$e_{v_m}^{'(l)} = w^{(l)} e_{v+}^{(l)} + (1-w^{(l)}) e_{v_m}^{(l)} \tag{26}$$

With this transformation, the model takes fewer epochs to converge, and its final accuracy performance remains at a similar level to the original model. However, since we introduced new parameters into the model, the loss function will be modified by adding a new regularization loss item related to the length of w, which can be simply formatted as $\lambda_w, ||w||^2$. These adjustments further enhance the training process and the model's performance, resulting in a more accurate and efficient GNN-based recommender system.

B. *Experimental Results of Tuned-MixGCF*

In this section, we first conducted two comparative experiments on the Tuned-MixGCF model to test its effectiveness. Then, we connected the Tuned-MixGCF model to the SimpleLGN model and compared it with the original LightGCN model.

1) Comparison between Tuned and Original MixGCF: Figures 11 and 12 present the effectiveness of our Tuned model based on LGN. They show a comparison of the training results between Tuned-MixGCF and the original MixGCF, demonstrating that the Tuned model has better performance because the results converge much faster and achieve a higher precision. We deployed the Gowalla dataset, as it requires less time to train compared to others, and we only display 40 epochs because the test recall converges in about 30 epochs.

2) Comparison between Tuned-MixGCF and LightGCN: Table VI presents the comparison of training results between Tuned-MixGCF based on LGN and the original LGN. We can see that the training accuracy is improved on both datasets:

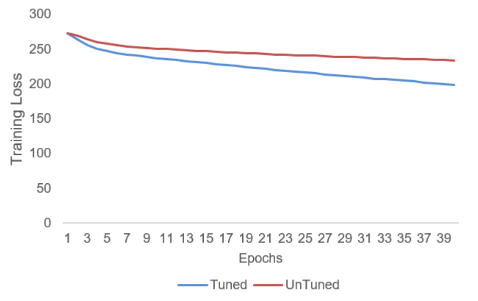

Fig. 11. The Effectiveness of Tuned model based on LGN - Training Loss: Tuned model v.s. Untuned model

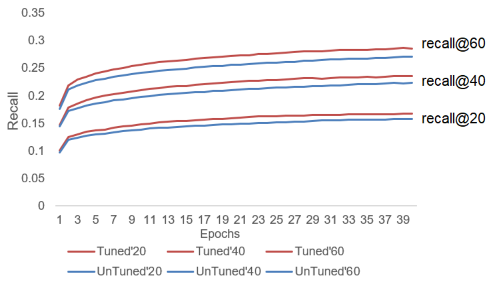

Fig. 12. The Effectiveness of Tuned model based on LGN - Test Recall: Tuned model v.s. Untuned model

Gowalla has a 1.6% improvement in recall@20, and Yelp has an 8.9% improvement in recall@20. Meanwhile, the training time is significantly reduced. For example, LightGCN requires 22 hours to train on the Gowalla dataset, while Tuned-MixGCF needs only 41 minutes.

TABLE VI
TRAINING RESULTS ON GOWALLA AND YELP2018

Model	Layers	Recall	NDCG
Gowalla	4-L LightGCN	0.1794	0.1526
	4-L Tuned MGCF	0.1823	0.1545
Yelp2018	4-L LightGCN	0.0649	0.0530
	4-L Tuned MGCF	0.0707	0.0581

3) The Effectiveness of the Integration Model: SimpleLGN and Tuned-MixGCF: In this part we do the integration of SimpleLGN and Tuned-MixGCF on Yelp dataset. Compared with the original LightGCN model, we can see an improvement of 14.2% in accuracy performance. The results are given in Table VII.

XI. SUMMARY

A. Achievements

In this project, we achieved several notable accomplishments that enhance the efficiency and performance of GNN-based recommender systems. Our primary achievements include:

- **Demonstrating the effectiveness of the Cluster-Based Sampling Approach:** We successfully showed that this approach reduces training time and alleviates the over-smoothing problem commonly encountered in GNN-based models.
- **Introducing a novel approach to collapsing the User-Item Representation in LightGCN expansion:** We proposed a more simplified form that reduces the complexity of the model while maintaining its effectiveness.
- **Boosting training speed and performance with the Tuned MixGCF Training Method:** We refined the original MixGCF model and demonstrated that our Tuned-MixGCF approach significantly improves both training efficiency and model accuracy.

TABLE VII
TRAINING RESULTS OF THE INTEGRATION MODEL

Model	Recall	Precision	NDCG
3-L LightGCN	0.0628	0.0281	0.0514
3-L LightGCN +Tuned-MGCF	0.0696	0.0313	0.0569
2-L SimpleLGN +Tuned-MGCF	0.0711	0.0320	0.0585
2-L Uniform-SWSLGN +Tuned-MGCF	0.0706	0.0318	0.0583
2-L Trainable-SWSLGN +Tuned-MGCF	0.0717	0.0323	0.0589

B. Discussion of Key Challenges and Future Work

Throughout this project, we encountered several challenges that limited the extent of our research and experimentation. However, these challenges also present opportunities for future work and continued development.

1) **Integration of Cluster-Based Sampling:** One significant challenge was integrating the Cluster-Based Sampling approach into the source code for the LightGCN model. Implementing this approach required extensive modification of the data pre-processing and mini-batch training pipelines, which proved time-consuming. Due to the limited time available for research, we could not complete the integration and perform experiments on the Gowalla and Yelp benchmark datasets. Future work could focus on successfully integrating this approach and evaluating its performance on these datasets.
2) **Joint Effects of Approaches and Tuned-MixGCF:** Another area for future work involves exploring the combined effects of Cluster-Based Sampling with SimpleLGN (Collapsing user-item representation approach) and Tuned-MixGCF. A comprehensive investigation of the potential synergies between these methods could further improve the LightGCN model's performance and efficiency. Additionally, the study could provide valuable insights into the most practical combination of tech-

niques for alleviating over-smoothing and enhancing the training process in GNN-based recommender systems.

By addressing these challenges and exploring future research directions, the LightGCN model can be further refined and optimized, ultimately delivering improved performance and efficiency in graph representation learning for recommender systems.

XII. Team Member Contributions

The authors acknowledge that each team member played an equal and vital role in the conducted research to address the limitations and challenges of LightGCN with respect to the over-smoothing problem, as well as the delivery of each of the project's milestone submissions, which include the Project Proposal, Progress Report, Presentation, and Final Report.

- **Jonathan:** Focused on exploring the Cluster-Based Sampling approach to mitigate the over-smoothing issue in the LightGCN model.
- **Shi Qin:** Concentrated on the Dimensionality Reduction approach to alleviating over-smoothing in the LightGCN model.
- **Chen Pu:** Main contribution was to develop the Tuned-MixGCF method to improve the training performance of the LightGCN model.

Together, our collaborative efforts have contributed to what we view as a successful project, where we have gained a deeper insight into the LightGCN model for recommendation and attempted to test simple statistical ideas for alleviating the over-smoothing problem in LightGCN, ultimately improving its performance and efficiency in graph representation learning for recommender systems.

References

[1] Koren, Y., Bell, R. & Volinsky, C. Matrix Factorization Techniques for Recommender Systems. *Computer*. **42**, 30-37 (2009)

[2] Hu, Y., Koren, Y. & Volinsky, C. Collaborative Filtering for Implicit Feedback Datasets. *2008 Eighth IEEE International Conference On Data Mining*. pp. 263-272 (2008)

[3] Koren, Y. Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model. *Proceedings Of The 14th ACM SIGKDD International Conference On Knowledge Discovery And Data Mining*. pp. 426-434 (2008), https://doi.org/10.1145/1401890.1401944

[4] He, X., Liao, L., Zhang, H., Nie, L., Hu, X. & Chua, T. Neural Collaborative Filtering. *Proceedings Of The 26th International Conference On World Wide Web*. pp. 173-182 (2017), https://doi.org/10.1145/3038912.3052569

[5] Kipf, T. & Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. (arXiv,2016), https://arxiv.org/abs/1609.02907

[6] Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. & Leskovec, J. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. *Proceedings Of The 24th ACM SIGKDD International Conference On Knowledge Discovery & Data Mining*. pp. 974-983 (2018), https://doi.org/10.1145/3219819.3219890

[7] Wang, M., Lin, Y., Lin, G., Yang, K. & Wu, X. M2GRL: A Multi-Task Multi-View Graph Representation Learning Framework for Web-Scale Recommender Systems. *Proceedings Of The 26th ACM SIGKDD International Conference On Knowledge Discovery & Data Mining*. pp. 2349-2358 (2020), https://doi.org/10.1145/3394486.3403284

[8] Wang, X., He, X., Wang, M., Feng, F. & Chua, T. Neural Graph Collaborative Filtering. *Proceedings Of The 42nd International ACM SIGIR Conference On Research And Development In Information Retrieval*. pp. 165-174 (2019), https://doi.org/10.1145/3331184.3331267

[9] He, X., Deng, K., Wang, X., Li, Y., Zhang, Y. & Wang, M. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. *Proceedings Of The 43rd International ACM SIGIR Conference On Research And Development In Information Retrieval*. pp. 639-648 (2020), https://doi.org/10.1145/3397271.3401063

[10] Huang, T., Dong, Y., Ding, M., Yang, Z., Feng, W., Wang, X. & Tang, J. MixGCF: An Improved Training Method for Graph Neural Network-Based Recommender Systems. *Proceedings Of The 27th ACM SIGKDD Conference On Knowledge Discovery; Data Mining*. pp. 665-674 (2021), https://doi.org/10.1145/3447548.3467408

[11] Wu, S., Sun, F., Zhang, W., Xie, X. & Cui, B. Graph Neural Networks in Recommender Systems: A Survey. *ACM Comput. Surv.*. **55** (2022), https://doi.org/10.1145/3535101

[12] Hamilton, W., Ying, R. & Leskovec, J. Inductive Representation Learning on Large Graphs. *Proceedings Of The 31st International Conference On Neural Information Processing Systems*. pp. 1025-1035 (2017)

[13] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P. & Bengio, Y. Graph Attention Networks. (arXiv,2017), https://arxiv.org/abs/1710.10903

[14] Rendle, S., Freudenthaler, C., Gantner, Z. & Schmidt-Thieme, L. BPR: Bayesian Personalized Ranking from Implicit Feedback. *Proceedings Of The Twenty-Fifth Conference On Uncertainty In Artificial Intelligence*. pp. 452-461 (2009)

[15] Liao, J., Zhou, W., Luo, F., Wen, J., Gao, M., Li, X. & Zeng, J. SocialLGN: Light Graph Convolution Network for Social Recommendation. *Inf. Sci.*. **589**, 595-607 (2022), https://doi.org/10.1016/j.ins.2022.01.001

[16] Yang, Y., Yang, L., Lv, H. & Di, X. A Stock Prediction Model based on LightGCN. *2022 IEEE 10th Joint International Information Technology And Artificial Intelligence Conference (ITAIC)*. **10** pp. 694-698 (2022)

[17] Mao, K., Zhu, J., Xiao, X., Lu, B., Wang, Z. & He, X. UltraGCN: Ultra Simplification of Graph Convolutional Networks for Recommendation. *Proceedings Of The 30th ACM International Conference On Information & Knowledge Management*. pp. 1253-1262 (2021), https://doi.org/10.1145/3459637.3482291

[18] Li, Q., Han, Z. & Wu, X. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. *Proceedings Of The Thirty-Second AAAI Conference On Artificial Intelligence And Thirtieth Innovative Applications Of Artificial Intelligence Conference And Eighth AAAI Symposium On Educational Advances In Artificial Intelligence*. (2018)

[19] Chen, J., Zhu, J. & Song, L. Stochastic Training of Graph Convolutional Networks with Variance Reduction. (2018), https://arxiv.org/abs/1710.10568

[20] Xu, K., Hu, W., Leskovec, J. & Jegelka, S. How Powerful are Graph Neural Networks?. *CoRR*. abs/1810.00826 (2018), http://arxiv.org/abs/1810.00826

[21] Wu, F., Zhang, T., Souza Jr, A., Fifty, C., Yu, T. & Weinberger, K. Simplifying graph convolutional networks. *International Conference On Machine Learning*. pp. 6861-6871 (2019)

[22] He, K., Zhang, X., Ren, S. & Sun, J. Deep Residual Learning for Image Recognition. *2016 IEEE Conference On Computer Vision And Pattern Recognition (CVPR)*. pp. 770-778 (2016)

[23] Peng, S., Sugiyama, K. & Mine, T. SVD-GCN: A Simplified Graph Convolution Paradigm for Recommendation. *Proceedings Of The 31st ACM International Conference On Information & Knowledge Management*. pp. 1625-1634 (2022), https://doi.org/10.1145/3511808.3557462

[24] Ma, X., Hu, Q., Gao, Z. & AbdelHady, M. Contrastive Co-training for Diversified Recommendation. *2022 International Joint Conference On Neural Networks (IJCNN)*. pp. 1-9 (2022)

[25] Chiang, W., Liu, X., Si, S., Li, Y., Bengio, S. & Hsieh, C. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. *Proceedings Of The 25th ACM SIGKDD International Conference On Knowledge Discovery; Data Mining*. pp. 257-266 (2019), https://doi.org/10.1145/3292500.3330925

[26] Karypis, G. & Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal On Scientific Computing*. **20**, 359-392 (1998)

[27] Dhillon, I., Guan, Y. & Kulis, B. Weighted Graph Cuts without Eigenvectors A Multilevel Approach. *IEEE Transactions On Pattern Analysis And Machine Intelligence*. **29**, 1944-1957 (2007)

[28] Harper, F. & Konstan, J. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.*. **5** (2015), https://doi.org/10.1145/2827872

[29] Cho, E., Myers, S. & Leskovec, J. Friendship and Mobility: User Movement in Location-Based Social Networks. *Proceedings Of The 17th ACM SIGKDD International Conference On Knowledge Discovery And Data Mining*. pp. 1082-1090 (2011), https://doi.org/10.1145/2020408.2020579

[30] He, X., Zhang, H., Kan, M. & Chua, T. Fast Matrix Factorization for Online Recommendation with Implicit Feedback. *Proceedings Of The 39th International ACM SIGIR Conference On Research And Development In Information Retrieval*. pp. 549-558 (2016), https://doi.org/10.1145/2911451.2911489