ABSTRACT

With the tremendous success of Graph Convolutional Networks (GCNs), they have been widely applied to recommender systems and have shown promising performance. However, most GCN-based methods rigorously stick to a common GCN learning paradigm and suffer from two limitations: (1) the limited scalability due to the high computational cost and slow training convergence; (2) the notorious over-smoothing issue which reduces performance as stacking graph convolution layers. We argue that the above limitations are due to the lack of a deep understanding of GCN-based methods. To this end, we first investigate what design makes GCNs effective for recommendation. By simplifying LightGCN, we show the close connection between GCN-based and low-rank methods such as Singular Value Decomposition (SVD) and Matrix Factorization (MF), where stacking graph convolution layers is to learn a low-rank representation by emphasizing (suppressing) components with larger (smaller) singular values. Based on this observation, we replace the core design of GCN-based methods with a flexible truncated SVD and propose a simplified GCN learning paradigm dubbed SVD-GCN, which only exploits $K$-largest singular vectors for recommendation. To alleviate the over-smoothing issue, we propose a renormalization trick to adjust the singular value gap, resulting in significant improvement. Extensive experiments on three real-world datasets show that our proposed SVD-GCN not only significantly outperforms state-of-the-arts but also achieves over 100x and 10x speedups over LightGCN and MF, respectively.

CCS CONCEPTS
- Information systems → Recommender systems.

KEYWORDS
Collaborative Filtering, Graph Convolutional Network

1 INTRODUCTION

With rapid development of the Internet and web services, recommender systems have been playing an important role in people’s daily life. As a fundamental task for recommendation, Collaborative Filtering (CF) focuses on digging out the user preference from past user-item interactions, and has received much attention for decades. One of the most widely used CF methods, low-rank matrix factorization (MF) [21], characterizes user/item as latent vectors in an embedding space and estimates ratings as the cosine similarity between user and item latent vectors. To overcome the drawback of MF that a linear function is inefficient to capture complex user behaviour, subsequent works incorporate side information (e.g., user reviews, image data, temporal information, etc.) [3, 6, 16] and exploit advanced algorithms [11, 30, 31] to infer user preference.

However, traditional CF methods heavily rely upon the quality of interactions as they can only learn the direct user-item relations. Therefore, they always show poor performance due to the common data sparsity issue in practice. Recently, Graph Convolutional Networks (GCNs) [19] have shown great potential in various fields including social network analysis [5, 36] and recommender systems [2, 38]. Much research effort has been devoted to adapt GCNs for recommendation, such as augmenting GCNs with other advanced algorithms [15, 31, 35], simplifying GCNs to improve training efficiency and model effectiveness [4, 10, 24], and so on. By representing user-item interactions as a bipartite graph, the core idea of GCNs is to repeatedly propagate user and item embeddings on the graph to aggregate higher-order collaborative signals, thereby learning high quality embeddings even with limited interactions. Despite its effectiveness, most existing GCN-based methods suffer from the following limitations:

- The core step of GCNs is implemented by repeatedly multiplying by an adjacency matrix, resulting in high computational cost and poor scalability.
- As shown in many works [22, 40], stacking graph convolution layers tends to cause the oversmoothing issue, resulting in similar user/item representations and reducing the recommendation accuracy. As a result, most existing GCN-based CF methods remain shallow (two, three layers at most).
- Unlike traditional CF methods, user/item representations are contributed from tremendous higher-order neighborhood, making the model difficult to train. Some GCN-based CF methods such as LightGCN requires about 800 epochs to reach the best accuracy, which further increases the training cost.

We argue that the above limitations are due to the lack of a deep understanding of GCNs. Thus, in this work, we aim to figure out: what is the core design making GCNs effective for recommendation?
Based on our answer to this question, we propose a scalable and simple GCN learning paradigm without above limitations.

To this end, we first dissect LightGCN, a linear GCN-based CF method which only exploits neighborhood aggregation and removes other designs. By simplifying LightGCN, we show that it is closely related to low-rank CF methods such as Singular Value Decomposition (SVD) and low-rank Matrix Factorization (MF), where stacking graph convolution layers is to learn a low-rank representation by emphasizing (suppressing) the components with larger (smaller) singular values. With empirical analysis, we further show that only a very few components corresponding to $K$-largest singular values contribute to recommendation performance, whereas most information (over 95% on the tested data) are noisy and can be removed. Based on the above analysis, we replace the core component of GCNs (i.e., neighborhood aggregation) with a flexible truncated SVD and propose a simplified GCN learning paradigm dubbed SVD-GCN. Specifically, SVD-GCN only requires a very few ($K$-largest) singular values (vectors) and model parameters (less than 1% of MF’s on the tested data) for prediction. To alleviate the over-smoothing issue, we propose a renormalization trick to adjust the singular value gap, making important features of interactions well preserved, thereby resulting in significant improvement. Furthermore, to make the best of interactions, we augment SVD-GCN with user-user and item-item relations, leading to further improvement. Since the superiority of GCNs over traditional CF methods lies in the ability to augment interactions with higher-order collaborative signals, we only use 20% of the interactions for training.

2.2 Low-Rank Methods

Low rank representation plays a fundamental role in modern recommender systems [17]. The core idea of low-rank methods is inspired by Singular Value Decomposition (SVD):

$$\mathbf{R} = \mathbf{U} \mathbf{V}^T = \sum_{k=1}^{\min(|U|,|V|)} s_k \mathbf{u}_k \mathbf{v}_k^T.$$

The interaction matrix can be decomposed to three matrices, where the column of $[\mathbf{U}$ and $\mathbf{V}$ (i.e., $\mathbf{u}_k$ and $\mathbf{v}_k$)] and $s_k$ are $[\mathbf{U} \times \mathbf{V}]$ and singular value, respectively; $s_1 > s_2 > \cdots > 0$; $\text{diag}(\cdot)$ is the diagonalization operation. Since the components with larger (smaller) singular values contribute more (less) to interactions, we can approximate $\mathbf{R}$ with only $K$-largest singular values. Alternatively, we can learn low-rank representations in a dynamical way through matrix factorization (MF) [21]:

$$\min_{(r_{ui}) \in \mathbb{R}^{n \times m}} \sum_{(u,i) \in \mathcal{R}} \left\| r_{ui} - \mathbf{e}_u^T \mathbf{e}_i \right\|_2^2 + \lambda \left( \| \mathbf{e}_u \|_2^2 + \| \mathbf{e}_i \|_2^2 \right),$$

where $\lambda$ is the strength for regularization. Each user and item is represented as a trainable vector with dimension $d \leq \min(|\mathcal{U}|,|\mathcal{V}|)$. By optimizing the following objective function, the model is expected to learn important features from interactions (e.g., components corresponding to $d$-largest singular values).

3 METHODOLOGY

3.1 Connections Between GCNs and SVD

As activation functions and feature transformations have been shown ineffective for CF [10], we focus on LightGCN whose final embeddings are generated as follows:

$$\mathbf{O} = \sum_{l=0}^{L} \frac{\mathbf{H}^{(l)}}{L+1} = \sum_{l=0}^{L} \left( \frac{\mathbf{A}}{L+1} \right)^l \mathbf{E},$$
where the pooling function is $\frac{1}{T}$. If we take a closer look at the power of adjacency matrix $\tilde{A}^l$, we have the following observation:

$$
\tilde{A}^l = \begin{cases}
(\tilde{R}\tilde{R}^T)^{\frac{l}{2}} & l \in \{0, 2, 4, \ldots\} \\
0 & l \in \{1, 3, 5, \ldots\}.
\end{cases}
$$

(9)

Following the definition of $\tilde{A}, \tilde{R} = D_U^{-\frac{1}{2}} RD_I^{-\frac{1}{2}}$, where $D_U$ and $D_I$ are the node degree matrices for users and items, respectively. Then, we can split Equation (8) as follows:

$$
O_U = \frac{\sum_{l=0,2,4,\ldots} \left(\tilde{R}\tilde{R}^T\right)^{\frac{l}{2}} E_U + \sum_{l=1,3,5,\ldots} (\tilde{R}^T\tilde{R})^{\frac{l}{2}} E_I}{L + 1},
$$

$$
O_I = \frac{\sum_{l=0,2,4,\ldots} (\tilde{R}^T\tilde{R})^{\frac{l}{2}} E_U + \sum_{l=1,3,5,\ldots} \left(\tilde{R}\tilde{R}^T\right)^{\frac{l}{2}} E_I}{L + 1}.
$$

(10)

The first and second terms represent the messages from heterogeneous (even-hops) and heterogeneous (odd-hops) neighborhood, $O_U$ and $O_I$ are final embeddings for user and items, $E_U$ and $E_I$ are embedding matrices for users and items, respectively. Similar to the definition in Section 2.2, let $P, Q,$ and $E_k$ denote the stacked left, right singular vectors, and singular value for $\tilde{R}$, respectively, and we formulate the following theorem.

**Theorem 1.** The adjacency relations in Equation (10) can be rewritten as the following forms:

$$
(\tilde{R}\tilde{R}^T)^{\frac{l}{2}} = \text{Pdiag}\left(\sigma_k^l\right)P^T,
$$

$$
(\tilde{R}^T\tilde{R})^{\frac{l}{2}} = \text{Qdiag}\left(\sigma_k^l\right)Q^T,
$$

$$
\tilde{R}^T (\tilde{R}\tilde{R}^T)^{\frac{l}{2}} = \text{Pdiag}\left(\sigma_k^l\right)Q^T,
$$

$$
\tilde{R} (\tilde{R}^T\tilde{R})^{\frac{l}{2}} = \text{Qdiag}\left(\sigma_k^l\right)P^T.
$$

(11)

(12)

Following Theorem 1, we can rewrite Equation (10) as:

$$
O_U = \text{Pdiag}\left(\frac{\sum_{l=0,2,\ldots} \sigma_k^l}{L + 1}\right)P^T E_U + \text{Pdiag}\left(\frac{\sum_{l=1,3,\ldots} \sigma_k^l}{L + 1}\right)Q^T E_I,
$$

$$
O_I = \text{Qdiag}\left(\frac{\sum_{l=0,2,\ldots} \sigma_k^l}{L + 1}\right)Q^T E_U + \text{Qdiag}\left(\frac{\sum_{l=1,3,\ldots} \sigma_k^l}{L + 1}\right)P^T E_I.
$$

(13)

Now the final embeddings are contributed from $\tilde{R}$’s singular vectors and values instead of neighborhood. Note that:

$$
P^T E_U = \frac{\sum_{k=0} \sigma_k^l}{L + 1} \sum_{k=0} \sigma_k^l P_k^T P_k.
$$

(14)

$$
\sum_{l=0,2,\ldots} \sigma_k^l \text{ and } \sum_{l=1,3,\ldots} \sigma_k^l
$$

can be considered as weights of singular vectors when considering even and odd hop neighbors, respectively. We illustrate the normalized weights in Figure 2 (a) and (b), and make the following observation:

**Observation 1.** As stacking more graph convolution layers, the goal of GCNs is to learn a low-rank representation by stressing (suppressing) more components with larger (smaller) singular values.

We further observe that:

$$
O_u = \left(p^T \Omega^U \otimes \frac{\sum_{l=0,2,\ldots} \sigma_k^l}{L + 1} P_k^T \bigoplus P_k^T \otimes \frac{\sum_{l=1,3,\ldots} \sigma_k^l}{L + 1} Q_k^T \right)Q^T E_I,
$$

(15)

where $\sigma$ is a vector containing all singular values, $p^T_{u}$ is the $u$-th row vector, $\otimes$ represents the element-wise multiplication. We can see $P^T E_U$ and $Q^T E_I$ are common terms for distinct users/items, what makes representations unique lies in the term in parentheses.

**Assumption 1.** $P^T E_U$ and $Q^T E_I$ are redundant.

On the other hand, the above two terms play an important role constituting the core design of GCNs (i.e., neighborhood aggregation), replacing or removing them leads to a new learning paradigm without explicitly aggregating neighborhood. To verify this assumption, we evaluate three models: (1) the original model Equation (13); (2) we simply replace $P^T E_U$ and $P^T E_I$ with two different weight matrices; (3) we use a shared weight matrix based on (2).

The results in Figure 1 (a) and (b) show that the performance of the three models are fairly close, and thus: (1) neighborhood
aggregation is not necessary for GCNs; (2) The power of GCNs for CF does not heavily rely on model parameters, since reducing parameters (by half) does not reduce the accuracy and even results in faster convergence. Based on the model (3), we can merge the two terms in Equation (13) and simplify it as:

\[
O_U = P_{\text{diag}} \left( \frac{\sum_{l=0}^{L} \sigma_l^U}{L + 1} \right) W,
\]

\[
O_I = Q_{\text{diag}} \left( \frac{\sum_{l=0}^{L} \sigma_l^I}{L + 1} \right) W,
\]

and name it SVD-LightGCN. We can interpret it as a two-step procedure. We first obtain a weighted singular matrices by assigning the weight \( \frac{\sum_{l=0}^{L} \sigma_l^U}{L + 1} \) to singular vectors (i.e., \( p_k \) and \( q_k \)); then, we learn a condensed embeddings of the singular vectors through a feature transformation \( W \). Figure 2 (c) shows the goal of SVD-LightGCN is also to learn a low-rank representation, where the weights of singular vectors are adjustable through \( L \). We also observe that:

**Observation 2.** SVD is a special case of SVD-LightGCN where \( W = I \) and \( L = \frac{1}{2} \) (fixed to a square root).

### 3.2 Analysis on SVD-LightGCN

**Training Efficiency.** Observation 1 provides an alternative way to build GCNs, that we can directly focus on the weights over singular vectors instead of stacking layers. However, retrieving all singular vectors is computationally expensive and not applicable on large datasets as well. On the other hand, Observation 1 implies that most small singular values are not so helpful for recommendation. To further verify this observation, we compare SVD-LightGCN and SVD-LightGCN-T which only exploits \( K \)-largest singular values and vectors, and report the accuracy of them in Figure 1 (c) and (d), where x-axis represents the singular vector ratio: \( \frac{K}{\min(|U|,|I|)} \).

We can see SVD-LightGCN-T with only the top 1% largest singular values and vectors outperforms SVD-LightGCN which exploits all singular vectors, and the best accuracy is achieved at 4% on CitEULike, 6% on ML-100K. This finding not only shows that most small singular values and vectors are noisy that even reduces the performance, but also helps largely reduce the training cost and improve the training efficiency. For instance, retrieving 4% of the singular vectors and values only takes 1.8s on CitEULike, the learning parameters (\( |Kd| \)) are merely 1% of that of MF and LightGCN (\(|U|d + |I|d|\)).

**Over-Smoothing.** Users and items tend to have the same representations when the model layer \( L \) is large enough [22].

**Theorem 2.** The maximum singular value of \( R \) is 1.

As shown from Figure 2 (b), the larger singular values are further emphasized as increasing the model layers. Following Theorem 2, if we further increase the layer \( L \):

\[
\lim_{L \to \infty} = \frac{\sum_{l=0}^{L} \sigma_l^I}{L + 1} \to 0,
\]

where the weights of any singular vectors are reduced to 0 compared with the largest one \( \sigma_{\text{max}} \), where user/item representations are only contributed by the largest singular vector. Thus, increasing model layers does not necessarily lead to better representations and might instead cause information loss. The over-smoothing issue lies in the gap between singular values, where it is enlarged as stacking layers, which suppresses some important information that matters for recommendation. To alleviate this issue, we define a renormalized interaction matrix as: \( \bar{R} = (D_U + \alpha I)^{\frac{1}{2}} R (D_I + \alpha I)^{\frac{1}{2}} \) where \( \alpha \geq 0 \).

**Theorem 3.** Given the singular value \( \sigma_k \) of \( \bar{R} \), \( \sigma_{\text{max}} \leq \frac{d_{\text{max}}}{d_{\text{max} + 1}} \) where \( d_{\text{max}} \) is the maximum node degree.

The maximum singular value becomes smaller as increasing \( \alpha \), indicating a smaller gap. On the other hand, a too small gap fails to emphasize the difference of importance of different components (i.e., the component with a larger singular value is more important). Thus, we can adjust \( \alpha \) to regulate the gap to assure that important information is well preserved and to adapt to different datasets.

Furthermore, the weighting function is a crucial design as it controls the weights of singular vectors, while LightGCN adopts a polynomial in a heuristic way. Let \( \psi(\cdot) \) denotes the weighting function. Basically, we can parameterize \( \psi(\cdot) \) with advanced algorithms to dynamically learn the weights of singular vectors. Alternatively, if we consider \( \psi(\cdot) \) as a static continuous function of singular values \( \sigma_k \), it is expected to weight the singular vectors through a function with easy-to-adjust hyperparameters instead of by repeatedly increasing the model layer \( L \). In addition, by replacing the polynomial in LightGCN with \( \psi(\cdot) \), following the Taylor series \( \psi(\sigma_k) = \sum_{l=0}^{L} \alpha_l \sigma_k^l \), we can rewrite Equation (8) as:

\[
O = \left( \sum_{l=0}^{L} \alpha_l A^l \right) E,
\]

where \( \alpha_l \) is \( \psi(\sigma_k) \)'s \( l \)-th order derivative at 0, \( L \) is \( \psi(\cdot) \)'s highest order. From a spatial perspective, \( \alpha_l \) is also the contribution of \( l \)-th order neighborhood, and \( L \) corresponds to the farthest neighborhood being incorporated. Intuitively, it is expected that user/item representations are constructed from as many positive neighborhood signals as possible (i.e., \( \alpha_l > 0 \) and \( L \to \infty \)), implying that \( \psi(\cdot) \) is infinitely differentiable with any-order derivatives positive.
3.3 SVD-GCN

Based on the analysis in Section 3.2, we formulate the user and item representations as follows:

\[
O_u = \tilde{P}^{(K)} \text{diag} \left( \psi \left( \sigma \beta \right) \right) W, \\
O_i = \tilde{Q}^{(K)} \text{diag} \left( \psi \left( \sigma \beta \right) \right) W,
\]

where \(\tilde{P}^{(K)}\) and \(\tilde{Q}^{(K)}\) are composed of \(K\)-largest left and right singular vectors of \(R\), respectively. Our initial attempt is to dynamically model the importance of singular vectors through a neural network given singular values as the input. However, we found that such a design underperforms static designs in most cases, and speculate that the reason is due to the data sparsity on CF. Unlike other recommendation tasks with rich side information, the only available data is the user/item ID besides interactions, which increases the difficulty to learn the intrinsic data characteristics. Based on previous analysis in Section 3.2, extensive experiments show that an exponential kernel \([20]\) achieves superior accuracy on the tested data, thus we set \(\psi(\sigma \beta) = e^{\beta \sigma^2}\) where \(\beta\) is a hyperparameter to adjust the extent of emphasis over larger singular values (i.e., a larger (smaller) \(\beta\) emphasizes the importance of larger (smaller) singular values more). We will also compare different \(\psi(\cdot)\) designs in Section 4.3. Unlike conventional GCNs updating all user/item embeddings simultaneously in a matrix form resulting in a large spatial complexity, we can train SVD-GCN in a node form with more flexibility as:

\[
o_u = p_u^T \odot \left( e^{\beta \sigma_u} \right) W, \\
o_i = q_i^T \odot \left( e^{\beta \sigma_i} \right) W,
\]

where \(p_u^T\) and \(q_i^T\) are the rows of \(\tilde{P}^{(K)}\) and \(\tilde{Q}^{(K)}\), respectively; \(\sigma_u\) is a vector containing all singular values. Note that the element-wise multiplication does not involve parameters thus can be pre-computed. Then, inspired by BPR loss \([26]\), we formulate the loss function as follows:

\[
L_{\text{main}} = \sum_{u \in \mathcal{U}} \sum_{(u,i) \in \mathcal{R}: \{u,i\} \in \mathcal{R}^u} \ln \sigma \left( o_u^T o_i - o_u^T o_{i}^- \right).
\]

As shown in Equation (10), in GCN-based CF methods, user/item representations are contributed from three kinds of information flows: user-item, user-user, and item-item relations. Thus, besides the user-item relations, homogeneous (i.e., user-user and item-item) relations also help increase model effectiveness. We define a user-user graph \(G_U = (\mathcal{V}_U, \mathcal{E}_U)\), and an item-item graph \(G_I = (\mathcal{V}_I, \mathcal{E}_I)\), where \(\mathcal{V}_U = \mathcal{U}\) and \(\mathcal{V}_I = \mathcal{I}\); \(\mathcal{E}_U = \{(u,q) | q \in \mathcal{N}_u, i \in \mathcal{N}_u\}\) and \(\mathcal{E}_I = \{(i,b) | b \in \mathcal{N}_u, i \in \mathcal{N}_u\}\), where \(\mathcal{N}_u\) and \(\mathcal{N}_i\) are the sets of directly connected neighbors for \(u\) and \(i\), respectively. Naturally, we can define the normalized adjacency matrix of \(G_U\) and \(G_I\) as \(R_U = R_U^T\) and \(R_I = R_I^T\), respectively. According to Equation (26) in Section 7, the eigenvectors of \(R_U\) and \(R_I\) are actually \(R\)’s left and right singular vectors, respectively; and the eigenvalues are both the square of \(R\)’s singular values. Thus, \(G, G_U,\) and \(G_I\) are closely connected. We formulate the following loss to learn the relations on \(G_U\):

\[
L_{\text{user}} = \sum_{u \in \mathcal{U}} \sum_{(u, u') \in \mathcal{E}_U: \{u, u'\} \in \mathcal{R}_U} \ln \sigma \left( o_u^T o_u - o_u^T o_{u'} \right).
\]

Similarly, we learn the relations on \(G_I\) via the following loss:

\[
L_{\text{item}} = \sum_{i \in \mathcal{I}} \sum_{(i,i') \in \mathcal{E}_I: \{i, i'\} \in \mathcal{R}_I} \ln \sigma \left( o_i^T o_i - o_i^T o_{i'} \right).\]

Finally, we propose the following four SVD-GCN variants:

\[
\begin{align*}
\text{SVD-GCN-B: } & \quad L = L_{\text{main}} + \lambda \| \Theta \|_2^2, \\
\text{SVD-GCN-U: } & \quad L = L_{\text{main}} + \gamma L_{\text{user}} + \lambda \| \Theta \|_2^2, \\
\text{SVD-GCN-I: } & \quad L = L_{\text{main}} + \gamma L_{\text{user}} + \zeta L_{\text{item}} + \lambda \| \Theta \|_2^2, \\
\text{SVD-GCN-M: } & \quad L = L_{\text{main}} + \gamma L_{\text{user}} + \zeta L_{\text{item}} + \lambda \| \Theta \|_2^2,
\end{align*}
\]

where \(\Theta\) denotes the model parameters. Besides the above variants, to evaluate the effect of the feature transformation, we propose a non-parametric method SVD-GCN-S by removing \(W\).

3.4 Discussion

3.4.1 Model Complexity. The complexity of SVD-GCN mainly comes from two parts. We first retrieve \(K\) singular vectors through SVD for the low-rank matrix \([8]\), with a complexity as: \(O(K |R|^d + K^2 |\mathcal{U}| + K^2 |\mathcal{I}|)\). We run the algorithm on GPU and only require a few very singular vectors, which only costs several seconds. Except for SVD-GCN-S, other variants require training with time complexity as \(O(c |R|^d)\), \(c\) is comparable to MF: \(c \in [K^d, d]\), where \(c\) denotes the number of epochs. On the other hand, the model parameters of MF is \(\|R\|_2^2\) time that of SVD-GCN. Overall, SVD-GCN is lighter than MF, and we will show more quantitative results in terms of efficiency in Section 4.2.

3.4.2 Comparison with GCN-based CF Methods. Compared with conventional GCN-based methods, GCN-SVD replaces neighborhood aggregation with a truncated SVD and significantly reduces the model parameters. Overall, SVD-GCN is equipped with a lighter structure and more scalable. Recent proposed work UltraGCN \([24]\) simplifies LightGCN by replacing neighborhood aggregation with a weighted MF and shows lower complexity:

\[
\max \sum_{u \in \mathcal{U}, i \in \mathcal{N}_u} \beta_{u,i} e_u^T e_i,
\]

where \(\beta_{u,i}\) is obtained from single-layer LightGCN. However, UltraGCN improves based on single-layer LightGCN, which can only exploit the first order neighborhood and losses the ability of incorporating high-order neighborhood to augment training interactions. On the other hand, SVD-GCN is derived from any-layer LightGCN and we further generalize it to the situation of infinite layers, hence maximizes the power of GCNs.
4 EXPERIMENTS
In this section, we comprehensively evaluate our proposed SVD-GCN. The rest of this section is organized as follows: we introduce experimental settings in Section 4.1, compare baselines with SVD-GCN in terms of recommendation accuracy and training efficiency in Section 4.2; in Section 4.3, we dissect SVD-GCN to show the effectiveness of our proposed designs and how different hyperparameter settings (i.e., 𝑘, 𝛼, 𝛽, 𝜀, 𝜆, and 𝜇) affect performance.

4.1 Experimental Settings

4.1.1 Datasets and Evaluation Metrics. We use five public datasets in this work, where the results of Figure 1 are based on CiteULike1 and ML-100K [9]. To demonstrate the effectiveness of our proposed methods on more datasets and to justify the previous analysis, we evaluate SVD-GCN on three other datasets: Gowalla [34], Yelp [12], and ML-1M [9]. Since we focus on implicit feedback, we only keep user/item ID and transform feedbacks to binary ratings. Table 1 lists statistics of datasets.

We adopt two widely-used metrics: Recall and nDCG [14] to evaluate our methods. Recall measures the ratio of the relevant items in the recommended list to all relevant items in test sets, while nDCG takes the ranking into consideration by assigning higher scores to items ranking higher. The recommendation list is generated by ranking unobserved items and truncating at position 𝑘. Since the advantage of GCN-based methods over traditional CF methods is the ability of leveraging high-order neighborhood to augment training data, thereby alleviating the data sparsity, we only use 20% of interactions for training and leave the remaining for test to evaluate the model robustness and stability; we randomly select 5% from the training set as validation set for hyper-parameter tuning and report the average accuracy on test sets.

4.1.2 Baselines. We compare our methods with the following competing baselines, where the hyperparameter settings are based on the results of the original papers:

- BPR [26]: This is a stable and classic MF-based method, exploiting a Bayesian personalized ranking loss for personalized rankings.
- EASE [29]: This is a neighborhood-based method with a closed form solution and show superior performance to many traditional CF methods.
- LightGCN [10]: This method uses a light GCN architecture for CF by removing activations functions and feature transformation. We use a three-layer architecture as the baseline.
- LCFN [39]: This model replaces the original graph convolution with a low pass graph convolution to remove the noise from interactions for recommendation. We set 𝐹 = 0.005 and use a single-layer architecture.
- SGL-ED [35]: This model generates different node views by randomly removing the edge connections and maximizes their agreements, and the proposed self-supervised loss is implemented on LightGCN [10]. We set 𝜏 = 0.2, 𝜆1 = 0.1, 𝑝 = 0.1, and use a three-layer architecture.
- UltraGCN [24]: This model simplifies LightGCN by replacing neighborhood aggregation with a weighted MF, which shows faster convergence and less complexity.

We remove some popular GCN-based methods such as PinSage [38], NGCF [34], and SpectralCF [41] as aforementioned baselines have already shown superiority over them.

4.1.3 Implementation Details. We implemented the proposed model based on PyTorch2 and released the code on Github3. For all models, We use SGD as the optimizer, the embedding size 𝑑 is set to 64, the regularization rate 𝜆 is set to 0.01 on all datasets, the learning rate is tuned around {0.001, 0.005, 0.01, · · · , 1}; without specification, the model parameters are initialized with Xavier Initialization [7]; the batch size is set to 256. We report other hyperparameter settings in the next subsection.

4.2 Comparison

4.2.1 Performance. We report the accuracy of baselines and our proposed GCN-SVD variants in Table 2, and have the following observations:

- Overall, GCN-based methods outperforms traditional CF methods, indicating the effectiveness of GCNs for CF and demonstrating the importance of augmenting training interactions by incorporating high-order neighborhood information, thereby alleviating data sparsity.
- Among all baselines, SGL-ED achieves the best across all datasets, while our proposed SVD-GCNs show consistent improvements over SGL-ED, indicating the effectiveness and superiority over conventional GCN designs. UltraGCN shows relatively poor performance among GCN-based methods. As shown in our previous analysis in Section 3.4.2, UltraGCN improves based on single-layer GCN which fails to leverage the higher-order neighborhood, thus cannot perform stably with limited interactions.
- Since our key contribution is to replace neighborhood aggregation, the improvement is more clear if we compare with pure GCN-based methods such as LightGCN. SVD-GCN outperforms LightGCN on Yelp, ML-1M, and Gowalla by 53.6%, 11.7%, and 29.0%, respectively, in terms of nDCG@10. The improvements over sparse data tend to be more significant, indicating the stability of SVD-GCN under extreme data sparsity.
- Among SVD-GCN variants, the basic model SVD-GCN-B and SVD-GCN-S already outperform all baselines by a large margin. In addition, introducing user-user and item-item relations results in further improvement. We also notice that mixing user-user and item-item relations does not necessarily leads to better accuracy, and we speculate that the reason might be related to the data density. On the dense data such as ML-1M where the user-item interactions are relatively sufficient, the improvement by introducing user-user and item-item relations is not as significant as that of sparser datasets, and incorporating both relations even performs worse; while on the sparsest data Gowalla, introducing auxiliary relations shows consistent improvements.

4.2.2 Training Efficiency. The results shown in this subsection are obtained on a machine equipped with AMD Ryzen 9 5950X and GeForce RTX 3090. Figure 3 shows how the preprocessing time and accuracy change with 𝐾, where SOTA is the best baseline. The best accuracy is achieved at 𝐾 = 90, 𝑅 = 60, and 𝐾 = 60, where

1https://github.com/j05212/citeulike-a
2https://pytorch.org/
3https://github.com/tanatosuu/svd_gcn
the preprocessing time is 3.07s, 0.82s, and 1.74s, on Gowalla, ML-
1M, and Yelp, respectively. Overall, only 1% singular vectors are
required on ML-1M, and less than 0.5% singular vectors are required
on Gowalla and Yelp, when the model reaches the best accuracy.
Table 3 shows the training time and running epochs of several
methods, where the running time includes both preprocessing and
training time. Overall, LightGCN is the most time consuming model
(3.858s) as it is a conventional GCN model; SVD-GCN-S is the
most time efficient model (3.07s) since it does not require model
optimization and shows over 1000x speed-up over LightGCN. BPR
is the fastest model (1.04s) in terms of training time per epoch, while
it still requires hundreds epochs to reach the best accuracy due to
the large amount of parameters need to be optimized. Although SVD-
GCN variants (excluding SVD-GCN-S) are slightly slower than BPR
on training time per epoch, they show fast training convergence as
the model parameters are only 0.08% of that of BPR.

### 4.3 Model Analysis

#### 4.3.1 How Homogeneous Relations Affect Performance?
The direct comparison between SVD-GCN-B and SVD-GCN-U, SVD-GCN-I,
and SVD-GCN-M demonstrates the positive effect of homogeneous
relations. Furthermore, Figure 4 shows how different \( \gamma \) and \( \zeta \)
affect the accuracy, where the accuracy increases first then drops as
constant increasing the value of \( \gamma \) and \( \zeta \). The best accuracy is
achieved at \( \gamma = 0.5 \), while the optimal \( \zeta \) (0.9 on Gowalla and 0.7
on Yelp) is larger than \( \gamma \). One reasonable explanation is that item-item
relations are usually sparser (0.21% on Gowalla and 0.33% on Yelp)
than user-user relations (0.41% on Gowalla and 0.48% on Yelp).

#### 4.3.2 Do We Need Feature Transformation? By comparing SVD-
GCN-S and SVD-GCN-B, we can see \( W \) results in worse accuracy
on Gowalla and Yelp and only a slight improvement on ML-1M,
which shows that feature transformation does not help much learn
user-item interactions. On the other hand, we can identify the
positive effect of \( W \) when incorporating user-user and item-item
relations, which leads to improvement compared with SVD-GCN-B.
We speculate that the ineffectiveness of feature transformation is
related to the data density, where the intrinsic characteristic of
sparse data such as user-item interactions is difficult to learn, while
user-user and item-item relations are much denser thus is easier to
learn. Overall, SVD-GCN can achieve superior accuracy without any
model training, implying that the key design making GCN effective
for recommendation lies in a good low-rank representation.

### 4.3.3 Effect of Renormalization Trick
We have two observations from Figure 5 (a): as increasing \( \alpha \) (i.e., shrinking the singular value gap), (1) the accuracy increases first then drops, reaches the best at
\( \alpha = 3 \); (2) the model tends to require fewer singular vectors. In Figure
5 (b), as increasing \( \alpha \), (1) the maximum singular value becomes
smaller, which is consistent with Theorem 3; (2) singular values
drops more quickly, which explains why fewer singular vectors

![Figure 3: How the preprocessing time and accuracy (nDCG@10) vary on SVD-GCN-B.](image-url)
are required. For instance, the model with \( \alpha = 0 \) has more large singular values which contribute significantly to the interactions compared with the model with \( \alpha > 0 \), thus more singular vectors are required; while the important large singular values are fewer as increasing \( \alpha \). In other words, the important information is concentrated in fewer top singular values when we constantly increase \( \alpha \). Surprisingly, we have the same observation on other datasets. Theoretical analysis on this interesting phenomenon is beyond the scope of this work, we leave it for future work.

4.3.4 Effect of \( \beta \). Figure 6 shows the accuracy with varying \( \beta \). The accuracy first increases as increasing \( \beta \), then starts dropping after reaching the best performance at \( \beta = 2.5 \) on ML-1M, \( \beta = 6.0 \) on Gowalla; there is a similar trend on Yelp that the best accuracy is achieved at \( \beta = 4.0 \). We observe that \( \beta \) tends to be larger on sparser data, implying that the large singular values are more important on the sparser data. We speculate that there is less useful information on sparser datasets, thus the small singular values contain more noise and should be depressed more than denser datasets.

4.3.5 The Choice of Weighting Function. We show the accuracy of SVD-GCN-S with different weighting functions in Table 4. For dynamic designs, we use a neural network to attempt to model the importance of singular vectors with singular values as the input, while it underperforms most static designs, showing that the dynamic design is not suitable for the weighting function. For static designs, following the previous analysis in Section 3.2, we list some properties that matter to accuracy: (from left to right) if the function (1) is increasing, (2) has positive taylor coefficients, (3) is infinitely differentiable, and evaluate some functions, where the setting of \( \beta \) is based on the best accuracy of each function. We can see the importance of the three properties is (1)>>(2)>>(3), (1) implies that the larger singular values are assigned higher weights, which is important according to the previous analysis; (2) and (3) suggest if the model can capture neighborhood from any-hops with positive contributions. Overall, the importance of the three properties is (1)>>(2)>>(3), and the functions satisfying all three properties perform the best.

5 RELATED WORK

Collaborative Filtering (CF) is an extensively used technique in modern recommender systems. Early memory-based CF methods [27] predict user preference by computing the similarity between users or items. Later on, model-based methods become prevalent [21] which characterizes users and items as latent vectors and calculate their dot products to predict the unobserved ratings. Subsequent works focus on modeling complex user-item interactions with advanced algorithms, such as neural network [11, 37], attention mechanism [18], transformer [30], and so on. Behind the learning in Euclidean space, some methods [32] explore the potential of learning in non-Euclidean space. On another line, auxiliary information such as social relations [23], review data [1], temporal information [18] etc. is also well incorporated to obtain a better understanding of user preference.

The data sparsity issue on recommendation datasets limits the aforementioned traditional CF methods. The development of GCNs helps alleviate this issue by incorporating higher-order neighborhood to facilitate user/item representations, and thus much effort has been made to adapt GCNs to recommendation. Early work such as GC-MC [2] accumulates messages from different neighbors based on the rating for explicit feedbacks; SpectralCF [41] adapts the original graph convolution to CF with implicit feedbacks; NGCF
improves based on vanilla GCN [19] by additionally encoding the interactions via an element-wise multiplication. To improve the scalability on large-scale datasets, Ying et al. [38] defines a flexible graph convolution on spatial domain without passing messages with adjacency matrix. By showing the redundency of feature transformation and non-linear activation function, LightGCN [10] only keeps neighborhood aggregation for recommendation. Recent works fuse other research topics into GCNs, such as contrastive learning [35, 42], learning in hyperbolic space [31], negative sampling [13], graph signal processing [25], etc. and achieves further success.

Despite the superior performance that the aforementioned GCN-based methods have achieved, the computational cost of GCNs is much larger than traditional CF methods, making them unscaleable on large-scale datasets. Although some works [4,10] reduces the cost to some extent by removing feature transformation and non-linear activation functions, while the complexity mainly comes from the neighborhood aggregation, which is implemented by multiplying by an adjacency matrix. One recent work UltraGCN [24] further simplifies GCNs by replacing the neighborhood aggregation with a weighted MF, where the weight is obtained from a single-layer LightGCN, which significantly reduces the complexity. However, such a simplification degrades the power of GCNs as it can only capture the first-order neighborhood, and the experimental results also show its ineffectiveness under extreme sparsity. On the other hand, our proposed SVD-GCN is based on comprehensive theoretical and empirical analysis on LightGCN with any layers, whose superiority and effectiveness have been demonstrated through extensive experimental results.

6 CONCLUSION

In this work, we proposed a simplified and scalable GCN learning paradigm for CF. We first investigated what design makes GCN effective. Particularly, by further simplifying LightGCN, we showed that stacking graph convolution layers is to learn a low-rank representation by emphasizing (suppressing) more components with larger (smaller) singular values. Based on the close connection between GCN-based and low-rank methods, we proposed a simplified GCN formulation by replacing neighborhood aggregation with a truncated SVD, which only exploits K-largest singular values and vectors for recommendation. To alleviate over-smoothing issue, we proposed a renormalization trick to adjust the singular value vectors for recommendation. To alleviate over-smoothing issue, information to be concentrated in fewer singular vectors. singular values to drop more quickly, thereby making important information to be concentrated in fewer singular vectors.

7 PROOFS

7.1 Proofs of Theorem 1

Proof. Following SVD, we know any two singular vectors are orthonormal (i.e., $P^TP = I$ and $QQ^T = I$), thus it is easy to derive the following equations:

$$\begin{align*}
\tilde{R}^TR &= P\text{diag}(\sigma_k^2)P^T,
\tilde{R}^T\tilde{R} &= Q\text{diag}(\sigma_k^2)Q^T.
\end{align*}$$

Equation (26), we can derive the following equations:

$$\begin{align*}
\tilde{R}^T\tilde{R} &= \left(\tilde{R}\tilde{R}^T\right)^{\frac{1}{2}},
\tilde{R}^T\tilde{R} &= \left(\tilde{R}\tilde{R}^T\right)^{\frac{1}{2}}.
\end{align*}$$

By repeating the above Equations $l$ times, we obtain Equation (11).

For simplicity, we let $R' = \tilde{R}\left(\tilde{R}\tilde{R}^T\right)^{\frac{1}{2}}$, and $R'' = \tilde{R}^T\left(\tilde{R}\tilde{R}^T\right)^{\frac{1}{2}}$. We let $P'$, $Q'$ and $\sigma_k'$ denote the stacked left singular vectors, right singular vectors and singular value for $R'$, respectively. Following Equation (26), we can derive the following equations:

$$\begin{align*}
P'^T(\tilde{R}\tilde{R})P' &= \text{diag}(\sigma_k^2) P'^T, \\
P''^T(\tilde{R}\tilde{R})Q'' &= \text{diag}(\sigma_k^2) Q''^T.
\end{align*}$$

It is easy to observe that $P' = P$, $Q' = Q$ and $\sigma_k' = \sigma_k$. Then, according to SVD, we derive Equation (12).□

7.2 Proofs of Theorem 2 and 3

Proof. We first introduce Rayleigh quotients [28]:

$$\lambda_{\min} \leq x^T\tilde{A}x \leq \lambda_{\max} \text{ s.t. } |x| = 1,$$

where $\lambda_{\min}$ and $\lambda_{\max}$ are the minimum and maximum eigenvalues of $\tilde{A}$, respectively. Then, we can show $\lambda_{\max} = 1$:

$$1 - x^T\tilde{A}x = x^T\tilde{A}x - x^T\tilde{A}x = \sum_{(u,i) \in E} \left(\frac{x_u}{\sqrt{d_u}} - \frac{x_i}{\sqrt{d_i}}\right)^2 \geq 0.\tag{29}$$

In the meanwhile, we have the following observation:

$$\tilde{A} = \begin{bmatrix} p_k & q_k \end{bmatrix} = \begin{bmatrix} R_k & q_k \end{bmatrix} \text{diag}(\sigma_k) \begin{bmatrix} p_k & q_k \end{bmatrix},$$

which implies that $\sigma_k \in [\lambda_{\min}, \lambda_{\max}] \leq 1$ with $[p_k, q_k]^T$ as the eigenvector. By observing the eigenvector of $\lambda_{\max}$, if $\lambda_{\max}$ is also a singular value, we have: $p_k = \sqrt{d_k}$ and $q_k = \sqrt{d_i}$ where $1 \leq i \leq 1$ is a vector with all 1 elements. It is easy to verify the solution satisfies SVD: $R_kq_k = p_k$, thus $\sigma_{\max} = 1$.

Given $\tilde{R}$, we can define the corresponding adjacency matrix $\tilde{A}$. Since the relation in Equation (30) still holds between $\tilde{R}$ and $\tilde{A}$, we only need to prove $\lambda_{\max} \leq \frac{d_{\max}}{d_{max} + \alpha}$. Note that $x^T\tilde{A}x = \sum_{(u,i) \in E} \frac{2x_u x_i}{\sqrt{d_u} + \alpha \sqrt{d_i} + \alpha} \leq \sum_{u \in V} \frac{d_u}{d_{\max} + \alpha} \frac{x_u^2}{\sqrt{d_u} + \alpha \sqrt{d_u} + \alpha}$

$$= 1 - \sum_{u \in V} \frac{\alpha}{d_{\max} + \alpha} \frac{x_u^2}{\sqrt{d_u} + \alpha \sqrt{d_u} + \alpha} \leq 1 - \alpha \frac{d_{\max}}{d_{\max} + \alpha} \frac{\sum_{u \in V} d_u x_u^2}{\sum_{u \in V} d_u} \leq 1 - \alpha \frac{\sum_{u \in V} d_u x_u^2}{d_{\max} + \alpha}.$$ $\tag{31}$

$= \text{ holds when } \alpha = 0.$ When $\alpha > 0$, $\lambda_{\max} < \frac{d_{\max}}{d_{max} + \alpha}$, since $x$ takes different values at $\sum_{(u,i) \in E} \frac{2x_u x_i}{\sqrt{d_u} + \alpha \sqrt{d_i} + \alpha} = \sum_{u \in V} \frac{d_u}{d_{\max} + \alpha} \frac{x_u^2}{\sqrt{d_u} + \alpha \sqrt{d_u} + \alpha}$.
