Graph Convolutional Networks for Collaborative Filtering

ABSTRACT

Recommender Systems play an essential role in our daily lives, as they affect how people navigate the vast amounts of products available in online services. The task of recommendation is to provide users with a list of relevant items based on several so-called seed items. Collaborative filtering methods (*e.g.*, Factorization Machines) have achieved great success in many real-world applications, due to their simplicity, efficiency, and effectiveness. In general, these models rely on users' historical profiles and recommend new items that are similar to these ones each user has purchased in the past. This iterative procedure of accumulating user feedback subsequently improves their understanding of users' preferences, resulting in fine-tuning recommender systems.

Here we present a highly-scalablet Shallow Graph Neural Networks to capture long-range dependencies *without* increasing the depths of GNNs. We mainly marries graph convolutions and clustering methods to achieve effective local and non-local encoding. To be specific, at each layer, graph convolutions are used to compute node embeddings by aggregating information from their local neighbors. the proposed model then performs k-Means clustering in the embedding space to obtain graph-level representations (*e.g.*, user/item centroids), which highly reduces the complexity.

KEYWORDS

Graph Neural Networks, Recommender System

ACM Reference Format:
. 2018. Graph Convolutional Networks for Collaborative Filtering. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Recommender Systems play an essential role in our daily lives, as they affect how people navigate the vast amounts of products available in online services [35, 47]. The task of recommendation is to provide users with a list of relevant items based on several so-called seed items. Collaborative filtering methods (*e.g.*, Factorization Machines [10, 24]) have achieved great success in many real-world applications, due to their simplicity, efficiency, and effectiveness [2, 13, 25, 51]. In general, these models rely on users' historical profiles and recommend new items that are similar to these ones each user has purchased in the past. This iterative procedure of accumulating user feedback subsequently improves their understanding of users' preferences, resulting in fine-tuning recommender systems.

Graph Neural Networks (GNNs) are being increasingly used in the tasks of recommendation, owing to their theoretical elegance and good performance [6, 12, 15, 28, 42, 47]. GNNs learn the representation of a user/item through an iterative process of transferring, transforming, and aggregating the information from its neighbors [19]. By considering user-item interactions as a bipartite graph, GNNs are thus able to obtain expressive representations of users and items, and have achieved state-of-the-art performance. For example, PinSage [47] combines random walks and graph convolutions to generate item embeddings. NGCF [42] iteratively propagates user and item embeddings to obtain the high-order collaborative signals. LightGCN [12] simplifies the design of GNNs to make it more concise for recommendation.

Despite the encouraging performance, many of GNNs use a fairly shallow architecture to obtain node embeddings, and thus have no ability to capture *long-range dependencies* in graphs [34]. The main reason is that graph convolutions are naturally *local operators*, *i.e.*, a single graph convolution only aggregates messages from a node's one-hop neighborhoods [19]. Clearly, a k-layer GNN model can collect relational information up to k-hops away, but cannot discover the dependency with a range longer than k-hops away from any given node. Therefore, the ability of GNNs to capture long-range dependencies heavily depends on their depths.

While in principle deep GNNs should have more expressive ability to model complex graph structures, training deep GNNs poses unique challenges in practice. First, it is computationally inefficient since the complexity of GNNs is exponential to the number of layers, causing high demand on training time and GPU memory [4]. Second, it makes optimization difficult. Deep GNNs suffer from the issues of over-fitting, over-smoothing, and possible vanishing gradient [23]. These "bottleneck effects" may inhibit the potential benefits of deep GNNs. Third, repeating GNN operators implicitly assumes *homophily*[1] in graphs [53]. However, real-word graphs, especially for implicit user-item graphs, are often complex and exhibit a mixture of topological *homophily* or *heterophily* [18, 26]. Therefore, simply training deep GNNs is likely to have a drop in performance if GNNs are not well regularized [22]. *i.e.*, performing more than 5-10 message passing steps has not shown consistent improvements in many tasks [23].

In recent years, non-local neural networks have shed light on capturing long-range dependencies in the field of computer vision [27, 33, 41, 43, 49, 54]. In particular, non-local neural networks [27] first measure the pairwise relations between query

Woodstock '18, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

[1] homophily means connected nodes have similar features, while heterophily indicates connected nodes can have different features in graphs.

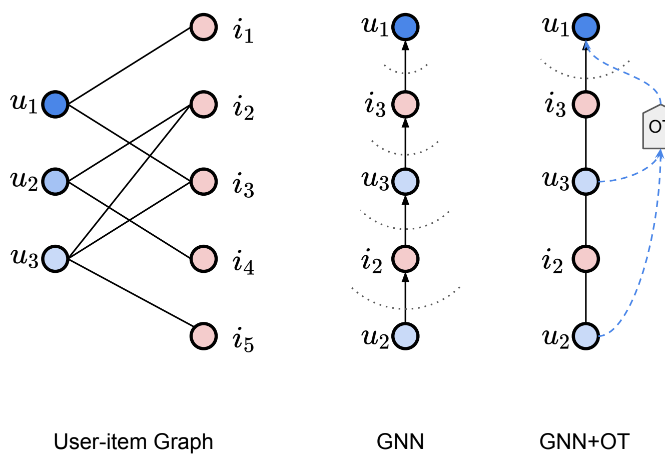

Figure 1: An illustration of local and non-local message passing schemes. Considering the message between u_1 and u_2, the local operators need four layers to collect the message: $u_1 \leftarrow i_3 \leftarrow u_3 \leftarrow i_2 \leftarrow u_2$, while our non-local operators only require one layer: $u_1 \underleftarrow{OT} u_2$.

position and all positions to form an attention map, and then aggregate the features by using self-attention mechanism [39]. This choice of design allows messages to be efficiently communicated across the whole images. Having this analogy, Geom-GCN [31] proposes geometric aggregators to compute the Euclidean distance between every pair of nodes. However, Geom-GCN is computationally prohibitive for large graphs as it requires to measure node-level pairwise relations, resulting in quadratic complexity. To remedy this issue, the recently proposed NL-GCN [29] leverages attention-guided sorting with a single calibration vector to redefine non-local neighborhoods. Nevertheless, NL-GCN solely calibrates the output embeddings of GNNs, which lacks adaptability. The purpose of this work is to reap the benefits of GNNs while avoiding such limitations.

Present Work: Here we present a highly-scalablet Shallow Graph Neural Networks to capture long-range dependencies *without* increasing the depths of GNNs. We mainly marries graph convolutions and clustering methods to achieve effective local and non-local encoding. To be specific, at each layer, graph convolutions are used to compute node embeddings by aggregating information from their local neighbors. the proposed model then performs k-Means clustering in the embedding space to obtain graph-level representations (*e.g.*, user/item centroids). A non-local attention operator is further proposed to measure the similarity between every pair of (node, centroid), which enables long-range messages to be communicated among distant but similar nodes.

Figure 1 illustrates how the long-range messages are propagated via local and non-local graph convolutional operators, respectively. Given target user u_1, the local operators in vanilla GNNs require two layers (*e.g.*, $u_1 \leftarrow i_3 \leftarrow u_3$) to aggregate information from its 2-hops user u_3, and four layers (*e.g.*, $u_1 \leftarrow i_3 \leftarrow u_3 \leftarrow i_2 \leftarrow u_2$) for its 4-hops user u_2. In contrast, our proposed non-local operators perform fast clustering on all users via Optimal Transport (OT) [5], and compute the attention once per cluster. As such, one single layer is sufficient to collect cluster-level messages from all users (*e.g.*, $u_1 \underleftarrow{OT} u_3$ and $u_1 \underleftarrow{OT} u_2$). Our non-local attention operators can seamlessly work with local operators in original GNNs. To this end, our model is able to capture both local and non-local message passing in graphs by only using shallow GNNs, which avoids the bottleneck effects of deep GNNs.

The introduced k-Means clustering in GNNs has a variety of merits: 1) By clustering items (users) into groups, similarity of the same item (or user) pair does not change significantly inside a group. This enables users to do item-centric exploration of inventories. 2) Unlike measuring pairwise attentions for every query in Geom-GCN [31], we group user/item queries into clusters and compute node-centroid attentions, which yields linear complexity as the number of centroids in graphs is often very small. 3) As clustering is performed on each layer of GNNs, and the major network architecture keeps unchanged, our our model is *model-agnostic* and can be applied to any GNNs. In addition, vanilla k-Means clustering contains non-differentiable operators, which are not suitable for end-to-end training. By revisiting the k-Means clustering as an Optimal Transport task [5], we further derive a fully differential clustering that can be joint training with GNNs at scale. Experimental results on real-word dataset demonstrate that our model achieves better performance compared with the state-of-the-art GNNs. We summarize our key contributions below:

- We propose our model, a model that can capture long-range dependencies without increasing the depths of GNNs. This largely reduce the bottleneck effects of over-fitting, over-smoothing, and vanishing gradient in deep GNNs.
- We perform k-Means clustering on embedding space, aiming to obtain compact centroids. Non-local operators are proposed to measure node-centroid attentions, which achieve linear complexity to capture long-range dependencies.
- We propose a fully differential k-Means clustering by casting it to an equivalent Optimal Transport task, which can be solved efficiently by greedy Sinkhorn algorithm. As such, the parameters of GNNs and k-Means can be jointly optimized at scale.
- Our our model is model-agnostic and can be applied to any GNNs. We conduct extensive experiments on four public datasets. The results demonstrate the benefits of SGCN on the effectiveness of pruning noisy edges and the usage of low-rank constraints, resulting in 14.92% ~ 26.23% performance gains.

2 RELATED WORK

In this section, we discuss the relevant work on collaborative filtering, GNN-based Recommendations, and non-local neural networks. We also highlight the differences between prior efforts and our proposed model.

2.1 Collaborative Filtering

Collaborative Filtering is a prevalent technique in recommender systems, which aims to guide users to find the items of interest. Factorization Machines (FMs) [20, 35] are early attempts to learn the embeddings of users and items from historical profiles and use inner product to model pairwise interactions. Inspired by the success of deep neural networks, several variants of FMs have been proposed to exploit more complex and nonlinear feature interactions. Some representative models include NCF [13], DeepFM [10], xDeepFM [24], xLightFM [16], FwFMs [32], FmFM [38] etc. These frameworks usually combines the design of factorization machines and deep neural networks, which can learn jointly learn low-order and high-order feature interactions in one unified architecture.

Nevertheless, FM-based methods largely rely on user-item historical interactions, which lack of ability to handle data sparsity issues *e.g.,* inactive users and items. Moreover, they may incur high memory consumption when the users/items contains an enormous number of categorical features [16]. Recently, graphs have been used to explore the implicit high-order proximity among nodes, which is useful for discovering topological information between users and items, leading to better recommendations [8, 47].

2.2 GNN-based Recommendations

Graph Neural Networks (GNNs) have received a lot of attention for their great performance in graph learning tasks [11, 19, 44]. GNNs strive to learn how to aggregate node messages from their local neighbors using neural networks. Recently, researchers have successfully applied GNNs to user-item bipartite graphs [1, 6, 12, 42, 47]. For instance, PinSage [47] utilizes random walks and graph convolutions to generate item embeddings from item-item graphs in Pinterest. GraphRec [6] integrates user social networks to refine the representations of nodes with attention mechanisms. NGCF [42] exploits the multi-hop community information to harvest the high-order collaborative signals between users and items. LightGCN [12] further simplifies the design of NGCF to make it more concise for recommendation purpose. Recently, IMP-GCN [28] performs sub-graph convolution to avoid negative message passing, and MixGCF [15] further empowers GNNs by providing hard negative samples through mixup techniques.

Although GNNs have been proven to be effective in generating node embeddings, they are not very friendly to distill long-range collaborative signals [22], which are important in understanding of user behaviors. The main reason is that regular graph convolutions are essentially local operators with limited receptive fields [29, 31]. While stacking multiple layers can theoretically enable GNNs to communicate information over long ranges, training deeper GNNs causes several bottleneck effects (*e.g.*, over-fitting and over-smoothing). In this study, we aim to learn both local and long-range messages over an entire graph without increasing the depths of GNNs, avoiding those bottleneck effects.

2.3 Over-fitting and Over-smoothing

Two main obstacles encountered when developing deeper GCNs are over-fitting and over-smoothing [21, 23, 30]. Over-fitting comes from the case when an over-parameterized GCN is used to fit a distribution given limited training data. Over-smoothing, on the contrary, leads to features of graph nodes gradually converging to the same value when increasing the number of convolutional layers [23]. Both of the above two issues can be alleviated by using *dropout* tricks in the GCNs. For example, vanilla Dropout [37] randomly masks out the elements in the weight matrix to reduce the effect of over-fitting. However, Dropout does not prevent over-smoothing since it does not make any change of the graph adjacency matrix. DropNode [11] is a node-oriented method that randomly selects the nodes for the mini-batch training. DropEdge [36] is an edge-oriented method that randomly removes a certain number of edges from the input graphs, acting like a data augmenter. Message dropout [42] randomly drops the outgoing messages in each propagation layer to refine representations. DropoutNet [40] applies input dropout during training to address cold start issue in recommender systems. Nevertheless, these dropout techniques typically remove a certain fraction of nodes, edges, or features by *random*, which may lead to sub-optimal performance.

2.4 Non-local Neural Networks

Capturing long-range dependencies (*a.k.a.* global semantic information) is proven to benefit numerous visual tasks [14]. However, regular CNNs have limited ability to deliver messages between distant positions as the CNN layer only aggregates pixel relations in a local region. To address this issue, non-local neural networks have been widely used for numerous visual tasks [27, 33, 41, 43, 49, 54]. The key idea of non-local neural networks is to affect an individual element (*e.g.*, a region in an image) by aggregating information from a set of elements (*e.g.*, all the regions in the image), where the attention weights are determined on the feature similarities among elements [41, 54]. Beyond CNNs, non-local mechanism is widely adopted in various kind of network architectures, such RNNs [27], ResNets [49], and U-nets [43] etc.

Inspired by non-local neural networks, Geom-GCN [31] learns the long-range dependencies by computing node-level pairwise relations. However, Geom-GCN is infeasible for large-scale graphs due to its quadratic complexity. To overcome this issue, the recently proposed NL-GCN [29] leverages a single calibration vector to redefine non-local neighborhoods. Nevertheless, NL-GCN solely calibrates the output embeddings of GNNs, which lacks adaptability. Our work here builds on this line of work but extends it by reaping the benefits of GNNs while avoiding such limitations. We introduce differential clustering method to calibrate the node embeddings of each layer by using non-local attentions. This allows that a node at any position perceives contextual messages from all nodes, which can benefit GNNs to learn long-range dependencies.

3 PROBLEM AND PRELIMINARY

3.1 Problem Definition

In this study, we focus on implicit feedback in recommendations. Formally, the behavior data, *e.g.*, click, comment, purchase, etc., involves a set of users $\mathcal{U} = \{u\}$ and items $\mathcal{I} = \{i\}$, such that the set $\mathcal{I}_u^+$ represents the items that user u has interacted with before, while $\mathcal{I}_u^- = \mathcal{I} - \mathcal{I}_u^+$ represents unobserved items. The goal is to estimate the user preference towards unobserved items.

By viewing user-item interactions as a bipartite graph $\mathcal{G}$, one can construct a user-item interaction matrix $\mathbf{A} \in \mathbb{R}^{N \times M}$, where N and M denote the number of users and items, respectively. Each entry $\mathbf{A}_{ui} = 1$ if user u has interacted with item i , and $\mathbf{A}_{ui} = 0$ otherwise. We aim to recommend a ranked list of items from $\mathcal{I}_u^-$ that are of interests to the user $u \in \mathcal{U}$, in the same sense that inferring the unobserved links in the bipartite graph $\mathcal{G}$.

3.2 GNNs for Recommendations

Recently, GNNs provide promising results for recommendations, like PinSage [47], NGCF [42], and LightGCN [12]. The key idea of GNNs is to learn node embeddings by smoothing features over graphs. We briefly introduce the design of LightGCN [12], including embedding lookup, aggregation, pooling, and optimization.

3.2.1 ***Embedding Layer.*** Following the mainstream GNN-based recommendations [12, 13, 42], the initial representations of a user u and an item i can be obtained via embedding lookup tables:

$$\mathbf{e}_u^{(0)} = \text{lookup}(u), \qquad \mathbf{e}_i^{(0)} = \text{lookup}(i), \tag{1}$$

where u and i denote the IDs of user and item; $\mathbf{e}_u^{(0)} \in \mathbb{R}^d$ and $\mathbf{e}_i^{(0)} \in \mathbb{R}^d$ are the embeddings of user u and item i, respectively, and d is the embedding size. These embeddings are expected to memorize the initial characteristics of items and users, and can be directly fed into a GNN model.

3.2.2 ***Aggregation.*** In order to exploit high-order collaborative signals from neighbors. GNN perform graph convolution iteratively, *i.e.*, aggregating features of neighbors to refine the embedding of a target node. Take LightGCN as an example, its aggregator is

$$\begin{aligned} \mathbf{e}_u^{(l+1)} &= \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|\,|\mathcal{N}_i|}} \mathbf{e}_i^{(l)}, \\ \mathbf{e}_i^{(l+1)} &= \sum_{u \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i|\,|\mathcal{N}_u|}} \mathbf{e}_u^{(l)}. \end{aligned} \tag{2}$$

where $\mathbf{e}_u^{(l)}$ and $\mathbf{e}_i^{(l)}$, with initialization $\mathbf{e}_u^{(0)}$ and $\mathbf{e}_i^{(0)}$ in Eq. (1), denote the refined embeddings of user u and item i after l-layer propagation, respectively; $\mathcal{N}_u$ denotes the set of items that are directly interacted by user u, and $\mathcal{N}_i$ denotes the set of users that are connected to item i. Both $\mathcal{N}_u$ and $\mathcal{N}_i$ can be retrieved from the user-item graph $\mathbf{A}$. As stacking multiple aggregation layers, each node can gather messages from its high-order neighbors.

3.2.3 ***NGCF.*** Following the standard GCN [19], NGCF [42] leverages the user-item bipartite graph to perform embedding propagation and feature transformation as:

$$\begin{aligned} \mathbf{e}_u^{(k+1)} &= \sigma\left(\mathbf{W}_1\mathbf{e}_u^{(k)} + \sum_{i\in\mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} \left(\mathbf{W}_1\mathbf{e}_i^{(k)} + \mathbf{W}_2(\mathbf{e}_i^{(k)} \odot \mathbf{e}_u^{(k)})\right)\right), \\ \mathbf{e}_i^{(k+1)} &= \sigma\left(\mathbf{W}_1\mathbf{e}_i^{(k)} + \sum_{u\in\mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_u||\mathcal{N}_i|}} \left(\mathbf{W}_1\mathbf{e}_u^{(k)} + \mathbf{W}_2(\mathbf{e}_u^{(k)} \odot \mathbf{e}_i^{(k)})\right)\right), \end{aligned} \tag{3}$$

where $\mathbf{e}_u^{(k)}$ and $\mathbf{e}_i^{(k)}$, with initialization $\mathbf{e}_u^{(0)} = \mathbf{e}_u$ and $\mathbf{e}_i^{(0)} = \mathbf{e}_i$ as in Eq. (2), denote the refined representations of user u and item i in the k-th layer of GCN, respectively; $\sigma(\cdot)$ is the nonlinear activation function and $\odot$ denotes the element-wise product; $\mathbf{W}_1$ and $\mathbf{W}_2$ are trainable weight matrices; $\mathcal{N}_u$ denotes the set of items that are directly interacted by user u, and $\mathcal{N}_i$ denotes the set of users that are connected to item i. As stacking more convolutional layers, the model is able to explore high-order collaborative signals between users and items.

3.2.4 ***Pooling.*** GNNs usually adopt the pooling techniques to readout the final representations of users/items [12, 42]. After propagating L layer, LightGCN obtains $L+1$ embeddings to represent a user $(\mathbf{e}_u^{(0)}, \ldots, \mathbf{e}_u^{(L)})$ and an item $(\mathbf{e}_i^{(0)}, \ldots, \mathbf{e}_i^{(L)})$. A weighted sum-based pooling is used to obtain the final representations:

$$\mathbf{e}_u^* = \sum_{l=0}^{L} \lambda_l \mathbf{e}_u^{(l)}, \quad \mathbf{e}_i^* = \sum_{l=0}^{L} \lambda_l \mathbf{e}_i^{(l)}, \tag{4}$$

where λ_l denotes the importance of the l-th layer embedding in constructing the final embeddings, which can be treated as hyper-parameter to be tuned manually.

3.2.5 ***Optimization.*** Finally, one can conduct inner product to estimate the user's preference towards the target item as

$$\hat{y}_{ui} = \mathbf{e}_u^{*T} \mathbf{e}_i^*. \tag{5}$$

The Bayesian Personalized Ranking (BPR) loss can be employed [35] to optimize the model parameters. The idea behind pairwise BPR loss is that an observed item should be predicted with a higher score than an unobserved one, which can be achieved by minimizing:

$$\mathcal{L}_{BPR}(\Theta) = \sum_{(u,i,j)\in O} - \ln \sigma\left(\hat{y}_{ui} - \hat{y}_{uj}\right) + \alpha\|\Theta\|_2^2, \tag{6}$$

where $O = \left\{(u,i,j) \mid u \in \mathcal{U} \wedge i \in I_u^+ \wedge j \in I_u^-\right\}$ denotes the pairwise training data; $\sigma(\cdot)$ is the sigmoid function; Θ denotes model parameters, and α controls the L_2 norm of Θ to prevent over-fitting.

3.3 The Local Aggregator Problem

The aggregator in Eq. (9) plays a critical role in collecting messages from neighbors. However, the graph convolutions are essential local operators, *i.e.*, $\mathbf{e}_u$ only collects information from its first-order neighbors $\mathcal{N}_u$ in each iteration. It is clear that the ability of LightGCN to capture long-range dependencies heavily depends on its depth: at least k GNN layers are needed to capture information that is k-hops away from a target node. Indeed, simply training deeper GNNs increases the receptive fields, but will cause several bottleneck effects, such as over-fitting and over-smoothing. In fact, most of GNN-based recommendation models achieve their best performance with at most 3 or 4 layers [12, 42].

Several regularization & normalization techniques have been suggested to overcome the bottleneck of deep GNNs, such as pairwise distance normalization between node features (PairNorm) [50], differentiable group normalization [52], randomly dropping edge [36], and skip connection [22]. However, the performance gains of deeper architectures are not always significant nor consistent for many tasks [22, 23]. Also, the complexity of GNNs is exponential to the number of layers, causing high demand on training time and GPU memory [4]. This barrier makes pursuit of deeper GNNs unpersuasive for billion-scale graphs. In this work, we ask the question of whether we can capture long-range dependencies by using shallow architecture of GNNs.

4 THE OUR MODEL METHOD

Instead of increasing the depth of GNNs, our proposed our model aims to integrate graph convolutions and k-Means clustering to collect both local and non-local messages in graphs. The flow of our

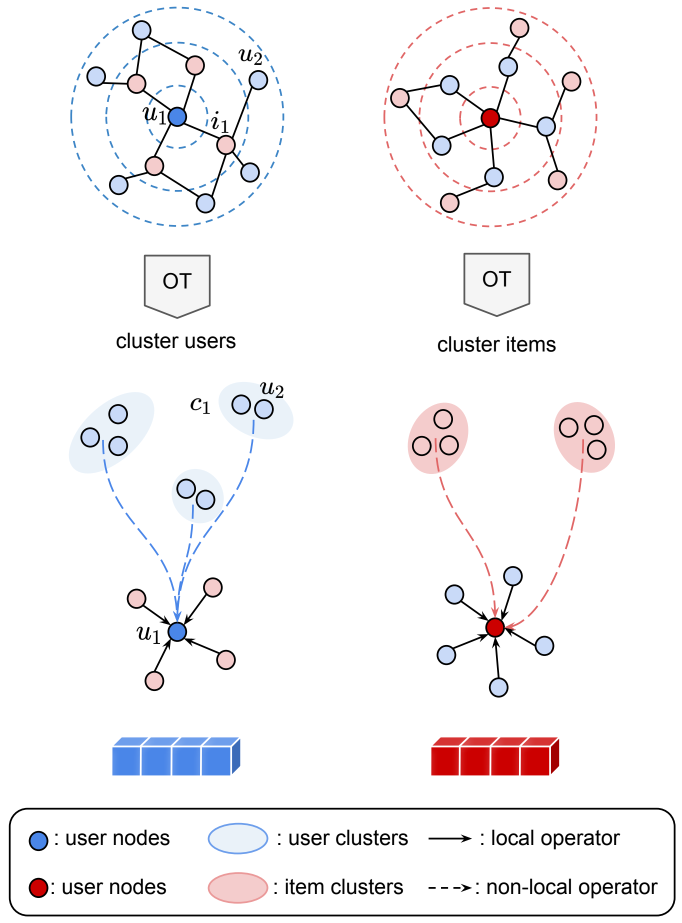

Figure 2: An overview of our model, where a target user/item node will aggregate both local messages (*e.g.*, neighbors) and non-local messages (*e.g.*, user/item centroids).

model is shown in Figure 2. Our proposed our model mainly consists of two stages: 1) obtain cluster-aware representations of users and items via k-Means clustering; 2) capture long-range dependencies via pairwise node-centroid attentions, rather than node-node attentions. As shown in Figure 2, even user u_1 and u_2 are 2-hops away from each other, the information of u_2 will be compactly represented by the centroid c_1, which can be passed to u_1 with one layer of GNN. We next introduce our our model in details.

4.1 k-Means Clustering with GNNs

Inspired by the fact that recommender systems suggest similar item to users that have similar interests, we perform clustering techniques on the user/item embeddings for each layer of GNNs. Clustering users/items are widely used in traditional collaborative filtering models [9, 17], but far less studied in graph neural models.

From Eq. (9), let $\{\mathbf{e}_{u_1}, \mathbf{e}_{u_2}, \ldots, \mathbf{e}_{u_N}\} \subset \mathbb{R}^d$ denote all the N user embeddings in l-th layer of GNNs[2], we attempt to separate users to different groups based on user embeddings from GNNs. Nevertheless, clustering may be ineffective when data points live in high-dimensional spaces. Therefore, we first construct a low-dimensional user space $\{f_\theta(\mathbf{e}_{u_1}), f_\theta(\mathbf{e}_{u_2}), \ldots, f_\theta(\mathbf{e}_{u_N})\} \subset \mathbb{R}^{d'}$ via a function $f_\theta(\cdot) : \mathbb{R}^d \to \mathbb{R}^{d'}$, where $f_\theta(\cdot)$ can be a neural network or even a linear projection. In this work, we simply implement $f_\theta(\mathbf{e}_{u_i}) = \mathbf{W} \cdot \mathbf{e}_{u_i}$, for $1 \leq i \leq N$, where $\mathbf{W} \in \mathbb{R}^{d' \times d}$ is a trainable weight. Then, we perform k-Means clustering in that low-dimensional space to obtain K user centroids $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K\} \subset \mathbb{R}^{d'}$ by minimizing:

$$\min_{\pi \in \{0,1\}} \sum_{i=1}^{N} \sum_{k=1}^{K} \pi_{ki} \cdot \| f_\theta(\mathbf{e}_{u_i}) - \mathbf{c}_k \|_2^2 \tag{7}$$

where π_{ki} indicates the cluster membership of user representation $f_\theta(\mathbf{e}_{u_i})$ *w.r.t.* centroid $\mathbf{c}_k$, *i.e.*, $\pi_{ki} = 1$ if data point $f_\theta(\mathbf{e}_{u_i})$ is assigned to cluster $\mathbf{c}_k$, and zero otherwise. The idea of combining clustering and neural networks is not new. Its original goal is to obtain "k-Means friendly" space in which high-dimensional data would have pseudo-labels, alleviating data annotation bottleneck in unsupervised learning tasks [7, 45, 48]. Here we use k-Means to summarize the graph-level information of users within clusters $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K\}$, which can be used to deliver long-range messages via non-local attention in Sec. 4.3.

However, solving Eq. (7) is not trivial since the term $f_\theta(\mathbf{e}_{u_i})$ involves another optimization task. EM-style methods cannot be jointly optimized with the GNNs or f_θ using standard gradient-based end-to-end learning, due to the non-differentiability of discrete cluster assignment in k-Means objective. Two recent work [7, 45] attempt to address this issue by proposing surrogate losses and alternately optimizing neural networks and cluster assignments with stochastic gradient descent. However, it is not guaranteed that final representations are good for the clustering task if it has been optimized for another task [48]. We next rewrite the k-Means as an Optimal Transport task, and derive a fully differentiable loss function for the purpose of joint training.

4.2 Differentiable k-Means Clustering via OT

Optimal Transport (OT) theory [5] is introduced to study the problem of resource allocation with lowest transport cost. By regarding the "cost" as distance, Wasserstein Distance is a commonly used metric for matching two distributions in OT [3].

4.2.1 ***Optimal Transport.*** Let $\boldsymbol{\mu} \in \mathbf{P}(\mathbb{X}), \boldsymbol{\nu} \in \mathbf{P}(\mathbb{Y})$ denote two discrete distributions, where $\boldsymbol{\mu} = \sum_{i=1}^{n} \mathrm{u}_i \delta_{\boldsymbol{x}_i}$ and $\boldsymbol{\nu} = \sum_{j=1}^{m} \mathrm{v}_j \delta_{\boldsymbol{y}_j}$, with $\delta_{\boldsymbol{x}}$ as the Dirac function centered on $\boldsymbol{x}$; $\Pi(\boldsymbol{\mu}, \boldsymbol{\nu})$ denotes all joint distributions $\gamma(\boldsymbol{x}, \boldsymbol{y})$, with marginals $\boldsymbol{\mu}(\boldsymbol{x})$ and $\boldsymbol{\nu}(\boldsymbol{y})$. The weight vectors $\mathbf{u} = \{\mathrm{u}_i\}_{i=1}^{n} \in \Delta_n$ and $\mathbf{v} = \{\mathrm{v}_i\}_{i=1}^{m} \in \Delta_m$ are the probability simplices[3] of size n and m, respectively. The Wasserstein Distance between $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ is defined as:

[2]To ease the explanation, we simply discard the subscript of embeddings $\{\mathbf{e}_{u_1}^{(l)}, \mathbf{e}_{u_2}^{(l)}, \ldots, \mathbf{e}_{u_N}^{(l)}\} \to \{\mathbf{e}_{u_1}, \mathbf{e}_{u_2}, \ldots, \mathbf{e}_{u_N}\}$ in the l-th layer of GNNs

[3]The probability simplex is denoted by $\Delta_K = \{z \in \mathbb{R}_+^K : \sum_{k=1}^{K} z_k = 1\}$.

$$\mathcal{D}_w(\boldsymbol{\mu},\boldsymbol{\nu}) = \inf_{\boldsymbol{\gamma}\in\Pi(\boldsymbol{\mu},\boldsymbol{\nu})} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\boldsymbol{\gamma}}[c(\boldsymbol{x},\boldsymbol{y})] \\ = \min_{\mathrm{T}\in\Pi(\mathbf{u},\mathbf{v})} \sum_{i=1}^{n}\sum_{j=1}^{m} \mathrm{T}_{ij}\cdot c(\boldsymbol{x}_i,\boldsymbol{y}_j) \tag{8}$$

where $\Pi(\mathbf{u},\mathbf{v}) = \{\mathbf{T}\in\mathbb{R}_{+}^{n\times m} \mid \mathbf{T}\mathbf{1}_m = \mathbf{u}, \mathbf{T}^\top\mathbf{1}_n = \mathbf{v}\}$; $\mathbf{1}_n$ denotes the n-dimensional vector of ones, and $c(\boldsymbol{x}_i,\boldsymbol{y}_j)$is the cost that evaluates the distance between $\boldsymbol{x}_i$ and $\boldsymbol{y}_j$ (samples of two distributions). The matrix $\mathbf{T}$ is denoted as the transport plan, where $\mathbf{T}_{ij}$ represents the amount of mass shifted from u_i to v_j. The optimal solution $\mathbf{T}^*$ is referred as*optimal transport plan.*

In this work, we use Optimal Transport Algorithm to solve Eq. (7).

4.3 Fast Non-local Attention

To this end, we can capture the attention via

$$\mathbf{e}_u^{(l+1)} = \sum_{i\in N_u} \frac{1}{\sqrt{|\mathcal{N}_u|\,|\mathcal{N}_i|}}\mathbf{e}_i^{(l)} + \text{Attention}(\mathbf{e}_u^{(l)},\mathbf{c}_k), \\ \mathbf{e}_i^{(l+1)} = \sum_{u\in\mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i|\,|\mathcal{N}_u|}}\mathbf{e}_u^{(l)} + \text{Attention}(\mathbf{e}_u^{(l)},\mathbf{d}_k). \tag{9}$$

The above embedding can be directly plugin into the Optimization (6)

5 EXPERIMENTS

Here we conduct experiments to answer the following questions:

- **RQ1:** How effective is the proposed SGCN compared to state-of-the-art baselines?
- **RQ2:** How can SGCN alleviate the problem of noisy edges?
- **RQ3:** How do different components (e.g., stochastic binary masks and low-rank constraints) affect the performance of SGCN?

5.1 Experimental Settings

5.1.1 Datasets. We use four public benchmark datasets for evaluating recommendation performance:

- **Movielens-1M**[4] is a widely used benchmark for recommendations. The dataset contains one million user-movie ratings.
- **Gowalla**[5] is a check-in dataset obtained from the location-based social website *Gowalla.*
- **Yelp**[6] is released by the *Yelp* challenge. The *Yelp2018* version is used in the experiments.
- **Amazon**[7] contains a large corpus of user reviews, ratings, and product metadata, collected from *Amazon.com.* We use the largest category *Books.*

For MovieLens, we treat all ratings as implicit feedback, e.g., each rating score is transformed to either 1 or 0 indicating whether a user rates a movie. For sparse datasets: Gowalla, Yelp, and Amazon, we use the 10-core setting of the graphs to ensure that all users and items have at least 10 interactions [12, 42]. We summarize the statistics of the datasets in Table 1.

[4]https://grouplens.org/datasets/movielens/20m/
[5]https://github.com/kuandeng/LightGCN/tree/master/Data
[6]https://www.yelp.com/dataset
[7]https://jmcauley.ucsd.edu/data/amazon/

Table 1: Dataset statistics.

Dataset	#User	#Items	#Interactions	Sparsity
MovieLens	6,040	3,900	1,000,209	4.190%
Gowalla	29,858	40,981	1, 027, 370	0.084%
Yelp	31, 668	38, 048	1, 561, 406	0.130%
Amazon	52, 643	91, 599	2, 984, 108	0.062%

For each dataset, we randomly select 80% of historical interactions of each user to construct the training set, and treat the remaining as the test set. From the training set, we randomly select 10% of interactions as validation set to tune hyper-parameters. For each observed user-item interaction, we treat it as a positive instance, and then conduct the ranking triplets by sampling from negative items that the user did not consume before. We repeat five random splits independently and report the averaged results.

5.1.2 Baselines. We compare with the following baselines:

- **BPR-MF** [35]: A classic model that seeks to optimize the Bayesian personalized ranking loss. We employ matrix factorization as its preference predictor.
- **NeuMF** [13]: NeuMF learns nonlinear interactions between user and item embeddings via a multi-layer perceptron as well as a generalized matrix factorization.
- **GC-MC** [1]: GC-MC employs a graph auto-encoder approach to learn the embeddings of users and items. A bilinear decoder is then used to predict the preference scores.
- **HOP-Rec** [46]: HOP-Rec discovers high-order indirect information of neighborhood items for each user from the bipartite graph by conducting random surfing on the graph.
- **BiNE** [8]: BiNE learns both explicit and implicit user-item relationships by performing biased random walks on the bipartite graph.
- **NGCF** [42] and **LightGCN** [12]: Two state-of-the-art GCN-based collaborative filtering models. They are briefly introduced in the Section 3.2.

5.1.3 Implementation Details. We implement our SGCNs in the TensorFlow. For all models, the embedding dimension d of users and items (e.g., in Eq. (2)) is searched among $\{16, 32, 64, 128\}$. For baselines BPR-MF, NeuMF, GC-MC, HOP-Rec, and BiNE, their hyper-parameters are initialized as in their original papers and are then carefully tuned to achieve the optimal performance. For the GCN components inside the proposed SGNNs, we use the same hyper-parameters as the original NGCF and LightGCN, such as batch size, stopping criteria, learning rate in Adam optimizer, etc. In addition, the SGCNs have two hyper-parameters β and γ to control the degree of sparsity and low-rank structure, respectively. We tune both β and γ within $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$ to investigate the parameter sensitivity of our models.

To evaluate the performance of top-n recommendation, we adopt two widely used evaluation metrics [12, 42]: *Recall* and *Normalized Discounted Cumulative Gain (NDCG)* over varying numbers of top ranking items.

Table 2: Recommendation performance comparison for different models. Note that R and N are short for Recall and NDCG, respectively.

	MovieLens				Gowalla				Yelp				Amazon			
Metric	R@50	N@50	R@100	N@100	R@50	N@50	R@100	N@100	R@50	N@50	R@100	N@100	R@50	N@50	R@100	N@100
BPR-MF	0.282	0.243	0.371	0.354	0.129	0.118	0.346	0.156	0.093	0.038	0.140	0.047	0.069	0.041	0.122	0.059
NeuMF	0.297	0.251	0.378	0.368	0.143	0.124	0.350	0.169	0.103	0.040	0.151	0.050	0.074	0.047	0.135	0.061
GC-MC	0.291	0.247	0.375	0.360	0.137	0.122	0.347	0.163	0.098	0.036	0.146	0.044	0.070	0.044	0.128	0.064
HOP-Rec	0.314	0.260	0.373	0.367	0.135	0.125	0.352	0.182	0.111	0.048	0.163	0.053	0.080	0.059	0.143	0.074
BiNE	0.312	0.253	0.381	0.371	0.141	0.126	0.354	0.188	0.110	0.042	0.155	0.049	0.076	0.052	0.134	0.069
NGCF	0.325	0.289	0.393	0.382	0.160	0.132	0.356	0.197	0.114	0.054	0.172	0.061	0.092	0.065	0.157	0.076
LightGCN	0.328	0.294	0.399	0.384	0.163	0.134	0.360	0.205	0.117	0.059	0.181	0.067	0.098	0.071	0.162	0.083
SGNNs	**0.351**	**0.326**	**0.456**	**0.477**	**0.199**	**0.163**	**0.401**	**0.256**	**0.176**	**0.101**	**0.208**	**0.092**	**0.134**	**0.097**	**0.181**	**0.112**

5.2 Performance Comparison (RQ1)

In this section, we compare the proposed SGCNs with baselines in terms of *Recall@n* and *NDCG@n* on all four datasets, where *n* is set to 50 and 100. The performance for different top-*n* values is similar in the experiments, we omit them for space limitation. The results for top-*n* recommendation are summarized in Table 2. Our proposed models consistently yield the best performance across all cases. From Table 2, our proposed model highly outperform the existing models.

6 CONCLUSION AND FUTURE WORK

In this paper, we propose Structured Graph Convolutional Networks to reduce the negative effects of noise in user-item bipartite graphs. In particular, we enforce sparsity and low-rank structure of the input graph while simultaneously training the parameters of GCNs. Our proposed SGCNs are compatible with various GCNs, such as NGCF and LightGCN, which can improve their robustness and generalization. The extensive experiments with real-world datasets show that SGCNs outperform the existing baselines.

For future work, we plan to improve our SGCNs by incorporating semantic information of graphs, such as users' social information or items' knowledge graph. Moreover, we are interested in applying pre-training strategies to enhance the embeddings of users and items. Lastly, we will also explore the explainability of GCNs, e.g., discovering a compact subgraph structure for a target user.

REFERENCES

[1] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. In *KDD Workshop on Deep Learning Day*.

[2] Huiyuan Chen and Jing Li. 2020. Neural Tensor Model for Learning Multi-Aspect Factors in Recommender Systems. In *IJCAI*.

[3] Liqun Chen, Zhe Gan, Yu Cheng, Linjie Li, Lawrence Carin, and Jingjing Liu. 2020. Graph optimal transport for cross-domain alignment. In *ICML*.

[4] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*. 257–266.

[5] Marco Cuturi. 2013. Sinkhorn distances: Lightspeed computation of optimal transport. *NeurIPS* (2013), 2292–2300.

[6] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *WWW*. 417–426.

[7] Maziar Moradi Fard, Thibaut Thonet, and Eric Gaussier. 2020. Deep k-means: Jointly clustering with k-means and learning representations. *Pattern Recognition Letters* (2020).

[8] Ming Gao, Leihui Chen, Xiangnan He, and Aoying Zhou. 2018. Bine: Bipartite network embedding. In *SIGIR*. 715–724.

[9] Songjie Gong. 2010. A collaborative filtering recommendation algorithm based on user clustering and item clustering. *J. Softw.* (2010), 745–752.

[10] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In *IJCAI*.

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*. 1024–1034.

[12] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, YongDong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR*. 639–648.

[13] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*. 173–182.

[14] Han Hu, Jiayuan Gu, Zheng Zhang, Jifeng Dai, and Yichen Wei. 2018. Relation networks for object detection. In *CVPR*.

[15] Tinglin Huang, Yuxiao Dong, Ming Ding, Zhen Yang, Wenzheng Feng, Xinyu Wang, and Jie Tang. 2021. MixGCF: An Improved Training Method for Graph Neural Network-Based Recommender Systems. In *KDD*. 665–674.

[16] Gangwei Jiang, Hao Wang, Jin Chen, Haoyu Wang, Defu Lian, and Enhong Chen. 2021. xLightFM: Extremely Memory-Efficient Factorization Machine. In *SIGIR*.

[17] Jyun-Yu Jiang, Patrick H Chen, Cho-Jui Hsieh, and Wei Wang. 2020. Clustering and constructing user coresets to accelerate large-scale top-k recommender systems. In *WWW*. 2177–2187.

[18] Wei Jin, Tyler Derr, Yiqi Wang, Yao Ma, Zitao Liu, and Jiliang Tang. 2021. Node similarity preserving graph convolutional networks. In *WSDM*. 148–156.

[19] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[20] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* (2009), 30–37.

[21] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns?. In *CVPR*. 9267–9276.

[22] Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. 2021. Training Graph Neural Networks with 1000 layers. In *ICML*.

[23] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*.

[24] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *KDD*. 1754–1763.

[25] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. 2018. Variational autoencoders for collaborative filtering. In *WWW*. 689–698.

[26] Derek Lim, Xiuyu Li, Felix Hohne, and Ser-Nam Lim. 2021. New Benchmarks for Learning on Non-Homophilous Graphs. *WWW Workshop on Graph Learning Benchmarks* (2021).

[27] Ding Liu, Bihan Wen, Yuchen Fan, Chen Change Loy, and Thomas S Huang. 2018. Non-Local Recurrent Network for Image Restoration. *NeurIPS* 31 (2018).

[28] Fan Liu, Zhiyong Cheng, Lei Zhu, Zan Gao, and Liqiang Nie. 2021. Interest-aware Message-Passing GCN for Recommendation. In *WWW*. 1296–1305.

[29] Meng Liu, Zhengyang Wang, and Shuiwang Ji. 2020. Non-local graph neural networks. *arXiv preprint arXiv:2005.14612* (2020).

[30] Dongsheng Luo, Wei Cheng, Wenchao Yu, Bo Zong, Jingchao Ni, Haifeng Chen, and Xiang Zhang. 2021. Learning to Drop: Robust Graph Neural Network via Topological Denoising. In *WSDM*.

[31] Federico Monti, Michael Bronstein, and Xavier Bresson. 2017. Geometric matrix completion with recurrent multi-graph neural networks. In *NeurIPS*. 3697–3707.

[32] Junwei Pan, Jian Xu, Alfonso Lobos Ruiz, Wenliang Zhao, Shengjun Pan, Yu Sun, and Quan Lu. 2018. Field-weighted factorization machines for click-through rate prediction in display advertising. In *WWW*.

[33] Jinsun Park, Kyungdon Joo, Zhe Hu, Chi-Kuei Liu, and In So Kweon. 2020. Non-local spatial propagation network for depth completion. In *ECCV*. 120–136.

[34] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. [n.d.]. Geom-GCN: Geometric Graph Convolutional Networks. In *ICLR, year=2020*.

[35] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*. 452–461.

[36] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2019. Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*.

[37] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* (2014), 1929–1958.

[38] Yang Sun, Junwei Pan, Alex Zhang, and Aaron Flores. 2021. FM2: Field-matrixed Factorization Machines for Recommender Systems. In *WWW*.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*. 5998–6008.

[40] Maksims Volkovs, Guangwei Yu, and Tomi Poutanen. 2017. Dropoutnet: Addressing cold start in recommender systems. In *NeurIPS*. 4957–4966.

[41] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. 2018. Non-local neural networks. In *CVPR*. 7794–7803.

[42] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *SIGIR*. 165–174.

[43] Zhengyang Wang, Na Zou, Dinggang Shen, and Shuiwang Ji. 2020. Non-local u-nets for biomedical image segmentation. In *AAAI*. 6315–6322.

[44] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *ICML*. 6861–6871.

[45] Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. 2017. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *ICML*.

[46] Jheng-Hong Yang, Chih-Ming Chen, Chuan-Ju Wang, and Ming-Feng Tsai. 2018. HOP-rec: high-order proximity for implicit recommendation. In *RecSys*. 140–144.

[47] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*. 974–983.

[48] Asano YM., Rupprecht C., and Vedaldi A. 2020. Self-labelling via simultaneous clustering and representation learning. In *ICLR*.

[49] Yulun Zhang, Kunpeng Li, Kai Li, Bineng Zhong, and Yun Fu. 2019. Residual Non-local Attention Networks for Image Restoration. In *ICLR*.

[50] Lingxiao Zhao and Leman Akoglu. 2020. PairNorm: Tackling Oversmoothing in GNNs. In *ICLR*.

[51] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *KDD*. 1059–1068.

[52] Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. 2020. Towards Deeper Graph Neural Networks with Differentiable Group Normalization. *NeurIPS* (2020).

[53] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs. *NeurIPS* 33 (2020).

[54] Zhen Zhu, Mengde Xu, Song Bai, Tengteng Huang, and Xiang Bai. 2019. Asymmetric non-local neural networks for semantic segmentation. In *CVPR*. 593–602.