Graph Neural Transport Networks with Non-local Attentions for Recommender Systems

Huiyuan Chen
hchen@visa.com
Visa Research

Chin-Chia Michael Yeh
miyeh@visa.com
Visa Research

Fei Wang
feiwang@visa.com
Visa Research

Hao Yang
haoyang@visa.com
Visa Research

ABSTRACT

Graph Neural Networks (GNNs) have emerged as powerful tools for collaborative filtering. A key challenge of recommendations is to distill long-range collaborative signals from user-item graphs. Typically, GNNs generate embeddings of users/items by propagating and aggregating the messages between local neighbors. Thus, the ability of GNNs to capture *long-range dependencies* heavily depends on their depths. However, simply training deep GNNs has several bottleneck effects, *e.g.,* over-fitting & over-smoothing, which may lead to unexpected results if GNNs are not well regularized.

Here we present Graph Optimal Transport Networks (GOTNet) to capture long-range dependencies without increasing the depths of GNNs. Specifically, we perform k-Means clustering on nodes' GNN embeddings to obtain graph-level representations (e.g., centroids). We then compute node-centroid attentions, which enable long-range messages to be communicated among distant but similar nodes. Our non-local attention operators work seamlessly with local operators in original GNNs. As such, GOTNet is able to capture both local and non-local messages in graphs by only using shallow GNNs, which avoids the bottleneck effects of deep GNNs. Experimental results demonstrate that GOTNet achieves better performance compared with state-of-the-art GNNs.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computer systems organization** → **Neural networks**.

KEYWORDS

Graph neural Networks, Optimal Transport, Non-local Attentions

ACM Reference Format:
Huiyuan Chen, Chin-Chia Michael Yeh, Fei Wang, and Hao Yang. 2022. Graph Neural Transport Networks with Non-local Attentions for Recommender Systems. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3485447.3512162

WWW '22, April 25–29, 2022, Virtual Event, Lyon, France

ACM ISBN 978-1-4503-9096-5/22/04…$15.00
https://doi.org/10.1145/3485447.3512162

1 INTRODUCTION

Graph Neural Networks (GNNs) are widely deployed in recommender systems, owing to their theoretical elegance and good performance [5, 14, 17, 26, 35, 40, 46]. By considering user-item interactions as bipartite graphs, GNNs learn the representation of a user/item through an iterative process of transferring, transforming, and aggregating the information from its neighbors [19]. This allows to obtain expressive representations of users and items, and achieve state-of-the-art performance. For example, PinSage [46] combines random walks and graph convolutions to generate item embeddings. NGCF [40] exploits the multi-hop community information to harvest high-order collaborative signals between users and items. LightGCN [14] further simplifies the design of NGCF to make it more concise for recommendation.

Despite their encouraging performance, many GNNs use a fairly shallow architecture to obtain node embeddings, and thus have no ability to capture *long-range dependencies* in graphs [3, 30]. 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 greater expressive ability to model complex graph structures, training deep GNNs poses several challenges in practice. First, it is computationally inefficient since the complexity of GNNs is exponential to the number of layers, causing high demand on the training time and the GPU memory [7]. 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 [51]. However, real-word graphs, especially for implicit user-item graphs, are often complex and exhibit a mixture of topological *homophily* or *heterophily* [24]. Therefore, simply training deep GNNs may cause unexpected results if GNNs are not well regularized [22].

In recent years, non-local neural networks have shed light on capturing long-range dependencies in computer vision domains [25, 29, 39, 42, 49, 52]. In particular, non-local neural networks [25] first measure the pairwise relations between the query position and all

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

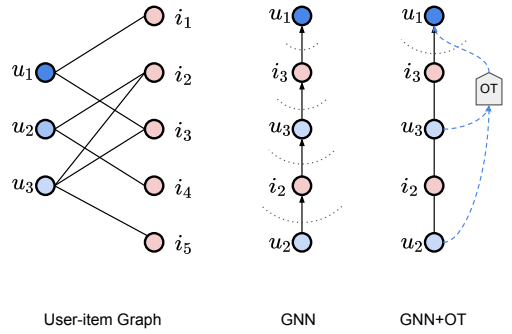

Figure 1: An illustration of local and non-local operators for collecting long-range messages.

positions to form an attention map, and then aggregate the features via a self-attention mechanism [36]. This design choice allows messages to be efficiently communicated across the whole images. In graph domains, Geom-GCN [30] uses 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 [27] 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 scalable Graph Optimal Transport Networks (GOTNet) to capture long-range dependencies *without* increasing the depths of GNNs. GOTNet marries graph convolutions and clustering methods to achieve effective local and non-local encoding, respectively. To be specific, at each layer, graph convolutions are used to compute node embeddings by aggregating information from local neighbors. GOTNet then performs k-Means clustering on the users' and items' embeddings to obtain their 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) [8], and compute the attention once per cluster. As such, one single layer is sufficient for collecting cluster-level messages from all users (*e.g.*, $u_1 \xleftarrow{OT} u_3$ and $u_1 \xleftarrow{OT} u_2$). Our non-local attention operators work seamlessly with local operators of the original GNNs. To this end, GOTNet 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, items (users) within same group share common properties. This enables users to do item-centric exploration of inventories. 2) Instead of measuring pairwise attentions for every query in Geom-GCN [30], 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 GOTNet 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 framing the k-Means clustering as an Optimal Transport task [8], we further derive a fully differential clustering that can be jointly trained with GNNs at scale. Experimental results on real-word datasets demonstrate that GOTNet achieves better performance compared with the state-of-the-art GNNs. We summarize our contributions as:

- We propose GOTNet to capture long-range dependencies without increasing the depths of GNNs. This largely reduce the bottleneck effects (*e.g.*, over-fitting, over-smoothing, and gradient vanishing) in training deep GNNs.
- We perform k-Means clustering on the embedding space to obtain compact centroids. Non-local operators are proposed to measure node-centroid attentions, which achieves linear complexity to collect long-range messages.
- We propose a fully differential k-Means clustering by casting it to an equivalent Optimal Transport task, which can be solved efficiently by the greedy Sinkhorn algorithm. As such, the parameters of GNNs and k-Means can be jointly optimized at scale.
- Our GOTNet is model-agnostic and can be applied to any GNNs. We conduct extensive experiments on four public datasets. The results demonstrate the benefits of GOTNet on the effectiveness of capturing long-range messages, resulting in 5.21% $\sim$ 19.45% performance gains.

2 RELATED WORK

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) [4, 20, 32] are early attempts to learn the embeddings of users and items from historical profiles and use inner products 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, including NCF [15] and DeepFM [13]. Nevertheless, FM-based methods largely rely on user-item historical interactions, which cannot reveal topological information between users and items. Recent work uses graphs to explore the implicit high-order proximity among nodes, leading to better recommendations [46].

2.2 GNN-based Recommendations

Graph Neural Networks have received a lot of attention in graph domains [19, 37]. 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 [14, 40, 46]. For example, PinSage [46] utilizes random walks

and graph convolutions to generate item embeddings from graphs in Pinterest. NGCF [40] exploits the multi-hop community information to harvest the high-order collaborative signals between users and items. LightGCN [14] further simplifies the design of NGCF to make it more concise for recommendation purpose. Recently, SGCNs [5] performs stochastic masks to avoid negative message passing, and MixGCF [17] further empowers GNNs by providing hard negative samples through mixup techniques.

However, GNNs are not very friendly to distill long-range collaborative signals [22], which are important in understanding user behaviors. The main reason is that regular graph convolutions are essentially local operators with limited receptive fields [19, 37]. 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 to avoid those bottleneck effects.

2.3 Non-local Neural Networks

Capturing long-range dependencies is proven to benefit numerous visual tasks [16]. However, regular CNNs have limited ability to deliver messages between distant positions as the CNN layer only aggregates pixel relations in a local region. Non-local neural networks have been proposed to address this issue [25, 29, 39, 42, 49, 52]. The key idea of non-local neural networks is to characterize 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 [39, 52]. Beyond CNNs, non-local mechanism is widely adopted in various kind of network architectures such as RNNs [25], ResNets [49], and U-nets [42].

Inspired by non-local neural networks, Geom-GCN [30] 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, NL-GCN [27] leverages a single calibration vector to redefine non-local neighborhoods. Nevertheless, NL-GCN solely calibrates the output embeddings of GNNs, which lacks adaptability. Our model builds upon this line of work by further clustering node embeddings with non-local attentions, which calibrates node embeddings in each layer.

3 PROBLEM AND BACKGROUND

3.1 Problem Setup

Here we focus on implicit feedback recommendations, where the behavior data 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 purchased 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

GNNs provide promising results for recommendations such as PinSage [46], NGCF [40], and LightGCN [14]. The key idea of GNNs is to learn node embeddings by smoothing features over graphs. We briefly introduce the design of GNNs, including embedding lookup, aggregation, pooling, and optimization.

3.2.1 ***Embedding Layer.*** The initial representations of a user u and an item i can be obtained via embedding lookup tables [14, 40]:

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

where u and i denote the IDs of a user and an 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.

3.2.2 ***Aggregation.*** GNNs perform graph convolution to aggregate features of neighbors [43]:

$$\mathbf{e}_u^{(l)} = \text{Agg}(\mathbf{e}_i^{(l-1)}, i \in N_u), \quad \mathbf{e}_i^{(l)} = \text{Agg}(\mathbf{e}_u^{(l-1)}, u \in N_i), \tag{2}$$

where $\mathbf{e}_u^{(l)}$ and $\mathbf{e}_i^{(l)}$ denote the refined embeddings in the l-layer; $\mathcal{N}_u$ denotes the set of items that are directly interacted by user u, and $\mathcal{N}_i$ is defined in the same way. Both $\mathcal{N}_u$ and $\mathcal{N}_i$ can be retrieved from the user-item graph $\mathbf{A}$.

3.2.3 ***Pooling.*** GNNs usually adopt the pooling techniques to readout $L + 1$ embeddings of a node. That is:

$$\mathbf{e}_u^* = \text{Pool}(\mathbf{e}_u^{(0)}, \ldots, \mathbf{e}_u^{(L)}), \quad \mathbf{e}_i^* = \text{Pool}(\mathbf{e}_i^{(0)}, \ldots, \mathbf{e}_i^{(L)}), \tag{3}$$

where the final representations of users/items $\mathbf{e}_u^*$ and $\mathbf{e}_i^*$ can be used for downstream tasks.

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

$$\hat{y}_{ui} = \mathbf{e}_u^{*T} \mathbf{e}_i^*. \tag{4}$$

The Bayesian Personalized Ranking loss can be adopted [32] to optimize model parameters, which can be achieved by minimizing:

$$\mathcal{L}_{BPR}(\mathbf{\Theta}) = \sum_{(u,i,j) \in O} -\ln \sigma\left(\hat{y}_{ui} - \hat{y}_{uj}\right) + \alpha \|\mathbf{\Theta}\|_2^2, \tag{5}$$

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, $\mathbf{\Theta}$ denotes model parameters, and α controls the L_2 norm of $\mathbf{\Theta}$ to prevent over-fitting.

3.3 The Local Aggregator Problem

The aggregator in Eq. (2) plays a critical role in collecting messages from neighbors. However, the graph convolutions are essentially 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 [14, 40].

Several regularization & normalization techniques have been suggested to overcome the bottleneck of deep GNNs, like PairNorm [50],

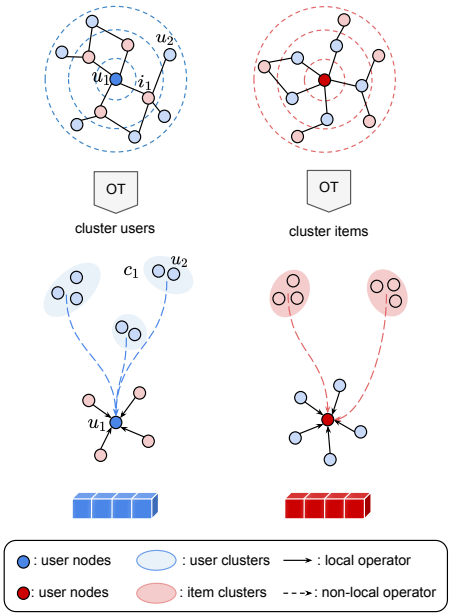

Figure 2: An overview of GOTNet, where a target user/item node will aggregate both local messages (*e.g.*, neighbors) and non-local messages (*e.g.*, user/item centroids).

DropEdge [33], and skip connection [22]. However, the performance gains of deeper architectures are not always significant nor consistent in 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 [7]. This barrier makes the pursuit of deeper GNNs unpersuasive for billion-scale graphs. In this work, we ask the question that whether we can capture long-range dependencies by using a shallow architecture of GNNs.

4 THE GOTNET METHOD

In this work, we aim to integrate graph convolutions and k-Means clustering to collect both local and non-local messages without increasing the depths of GNNs. Our GOTNet 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, though user u_1 and u_2 are 2-hops away from each other, the information of u_2 can be compactly represented by the centroid c_1, which can be passed to u_1 with only one layer of GNN. We next introduce our GOTNet in details.

4.1 k-Means Clustering with GNNs

Motivated by the fact that users with similar interests would purchase similar items, we cluster the user/item embeddings into different groups for each layer of GNNs. This allows that a node at any position perceives contextual messages from all nodes, which can benefit GNNs to learn long-range dependencies. Clustering users/items is widely used in traditional collaborative filtering models [12, 18], but far less studied in graph neural models.

Let $\{\mathbf{e}_{u_1}, \mathbf{e}_{u_2}, \ldots, \mathbf{e}_{u_N}\} \subset \mathbb{R}^d$ denote all N user embeddings in l-th layer of GNNs from Eq. (2)[2], we perform k-Means clustering on user embeddings. In general, these user embeddings from GNNs already live in the low-dimensional space that is friendly to k-Means. To make the model more flexible [36], we further project the 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. Here we simply define $f_\theta(\mathbf{e}_{u_i}) = \mathbf{W} \cdot \mathbf{e}_{u_i}$, for $1 \le i \le N$, 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:

$$\min_{\boldsymbol{\pi} \in \{0,1\}} \sum_{i=1}^{N} \sum_{k=1}^{K} \pi_{ki} \cdot \|f_\theta(\mathbf{e}_{u_i}) - \mathbf{c}_k\|_2^2, \tag{6}$$

where π_{ki} indicates the cluster membership of the 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 where high-dimensional data would have pseudo-labels, alleviating data annotation bottlenecks [9, 45, 47]. 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 attentions in Sec. 4.3.

However, solving Eq. (6) 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 the discrete cluster assignment in k-Means. Two recent work [9, 45] attempt to address this issue by proposing surrogate losses and alternately optimizing neural networks and cluster assignments. However, it is not guaranteed that final representations are good for the clustering task if it has been optimized for another task [47]. We next rewrite the k-Means as an Optimal Transport task and derive a fully differentiable loss function for joint training.

4.2 Differentiable k-Means Clustering via OT

Optimal Transport (OT) theory [8] is introduced to study the problem of resource allocation with lowest transport cost. We briefly introduce the concepts of OT in Appendix A.

4.2.1 ***Parameterizing k-Means as Optimal Transport.*** The connection between k-Means and OT was first noticed by Pollard [31], and has been revisited in literature [2, 21]. Roughly speaking, OT is equivalent to constrained k-Means clustering [10]. To parameterize k-Means using the optimal transport plan in Eq. (13), we put forward a new k-Means objective for clustering users as:

$$\min_{\boldsymbol{\pi}} \sum_{i=1}^{N} \sum_{k=1}^{K} \pi_{ki} \cdot \|f_\theta(\mathbf{e}_{u_i}) - \mathbf{c}_k\|_2^2 \quad \text{s.t. } \boldsymbol{\pi}^\top \mathbf{1}_K = \frac{1}{N}\mathbf{1}_N,\ \boldsymbol{\pi}\mathbf{1}_N = \frac{1}{N}\boldsymbol{w}, \tag{7}$$

where $\boldsymbol{w} = [n_1, \ldots, n_K]$ denotes cluster proportions, *i.e.*, n_k is the number of points in the partition k, and $\sum_{k=1}^{K} n_k = N$. The constraint $\boldsymbol{\pi}^\top \mathbf{1}_K = \mathbf{1}_N$ implies the soft assignment π_{ki} of each

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

data point i to each cluster k, while $\boldsymbol{\pi}\mathbf{1}_N = \boldsymbol{w}$ further encourages that each cluster k contains exactly n_k data points. As suggested by [28], we simply set $n_1 = \cdots = n_K = N/K$ for balanced partitions. Moreover, we explicitly introduce the normalization factor $1/N$ on both constraints, which follows the condition of probability simplex in OT and does not affect the loss.

By doing so, our proposed objective Eq. (7) becomes a standard OT problem, and we can regard the cluster assignment $\boldsymbol{\pi}$ as a transport plan and the Euclidean distance $\|f_\theta(\mathbf{e}_{u_i}) - \mathbf{c}_k\|_2^2$ as transport cost. Also, if we remove the second constraint on the size of each cluster, Eq. (7) boils down to the original objective as in Eq. (6).

4.2.2 ***Differentiable k-Means Loss.*** The Eq. (7) can be solved by Linear Programming (LP), yet with computational burden. This cost can be largely mitigated by introducing a strictly convex entropy regularized OT via fast Sinkhorn's algorithms [8]. Here we use Sinkhorn loss of Eq. (7) for clustering user $\mathcal{L}_{uOT}$:

$$\mathcal{L}_{uOT} = \min_{\boldsymbol{\pi}} \sum_{i=1}^{N}\sum_{k=1}^{K} \Big(\pi_{ki} \cdot \|f_\theta(\mathbf{e}_{u_i}) - \mathbf{c}_k\|_2^2 + \varepsilon \cdot \pi_{ki}(\log \pi_{ki} - 1)\Big)$$
$$\text{s.t.} \quad \boldsymbol{\pi}^\top \mathbf{1}_K = \frac{1}{N}\mathbf{1}_N,\ \boldsymbol{\pi}\mathbf{1}_N = \frac{1}{N}\boldsymbol{w}, \tag{8}$$

where $\varepsilon > 0$ is a regularization parameter. It is known that LP algorithm reaches its optimums on vertices, while the entropy term moves the optimums away from the vertices, yielding smoother feasible regions. Note that all Sinkhorn operations are differentiable, which makes k-Means fully differentiable and allows joint training with GNNs and f_θ in a stochastic fashion [11, 34]. Analogously, we can define the item clustering loss $\mathcal{L}_{iOT}$ in a similar way.

To obtain cluster assignment $\boldsymbol{\pi}$, we choose a greedy Sinkhorn algorithm, namely Greenkhorn [1], to solve Eq. (8), which accelerates training process significantly in practice. As the optimal transport plan $\boldsymbol{\pi}^*$ is computed, we can refine the centroids: $\mathbf{c}_k = \sum_{i=1}^N \pi^*_{ki} f_\theta(\mathbf{e}_{u_i}) / \sum_{i=1}^N \pi^*_{ki}$, for $k = 1, \cdots, K$. We are aware that other OT solvers [44] might further improve the numerical stability of training. We leave this extension in the future since designing new OT solvers is not our focus here.

4.3 Attention Mixing

4.3.1 ***Single Head Non-local Attention.*** In GNNs, node embeddings are updated via iterative message passing. For each layer, we obtain a set of user clusters $C^{(l)} = \{\mathbf{c}_1^{(l)}, \mathbf{c}_2^{(l)}, \ldots, \mathbf{c}_K^{(l)}\} \subset \mathbb{R}^d$, for $0 \le l \le L$, via differential k-Means clustering. Similar, a set of item clusters $D^{(l)} = \{\mathbf{d}_1^{(l)}, \mathbf{d}_2^{(l)}, \ldots, \mathbf{d}_P^{(l)}\} \subset \mathbb{R}^d$ (for $0 \le l \le L$) can be computed by clustering item embeddings, where K and P are the number of user and item clusters, respectively. Also, we assume the values of K and P are invariant within GNNs.

Inspired by attention mechanisms [36], we attempt to collect relevant long-range messages via non-local *node-centroid* attentions. An attention module takes three groups of vectors as inputs, named the query, key and value, where key and value can be the same.

Given the original GNN embeddings $e_u^{(l-1)}$ and $e_i^{(l-1)}$ in Eq. (2), and user/item centroids from the $(l-1)$-th layer, the attention mechanism gives the output vectors:

$$\mathbf{e}'^{(l)}_u = \sum_{k=1}^{K} a_k^{(l-1)} \mathbf{c}_k^{(l-1)}, \ \text{ and } \ a_k^{(l-1)} = \text{ATTEND}(e_u^{(l-1)}, c_k^{(l-1)}),$$
$$\mathbf{e}'^{(l)}_i = \sum_{k=1}^{K} b_k^{(l-1)} \mathbf{d}_k^{(l-1)}, \ \text{ and } \ b_k^{(l-1)} = \text{ATTEND}(e_i^{(l-1)}, d_k^{(l-1)}), \tag{9}$$

where $\mathbf{e}'^{(l)}_u$ and $\mathbf{e}'^{(l)}_u$ are cluster-aware representations of user u and item i in the l-th layer, respectively. These cluster-aware representations compactly summarize the topological information of all users and items, which contains long-range messages in graphs. ATTEND$(\cdot)$ could be any function that outputs a scalar attention score $a_k^{(l)}$ or $b_k^{(l)}$. We simply choose the scaled dot-product [36] for ATTEND$(\cdot)$. Notably, existing GNNs usually use the attention mechanism for local aggregation [37]. Here we extend for non-local aggregation from user/item centroids.

4.3.2 ***Multi-head Non-local Attentions.*** In Eq. (9), we perform a single attention to compute cluster-aware user and item embeddings. In NLP tasks, multi-head attention allows the model to simultaneously attend to information from different subspaces, making token embeddings more flexible [36]. Having this analogy, we further improve our GOTNet by using multiple OT heads.

Following Sec 4.1, we project users' GNN embeddings into H separate subspaces $\{f_\theta^h(\mathbf{e}_{u_1}), f_\theta^h(\mathbf{e}_{u_2}), \ldots, f_\theta^h(\mathbf{e}_{u_N})\}_{h=1}^H \subset \mathbb{R}^d$ via H separate functions, *i.e.,* $f_\theta^h(\mathbf{e}_{u_i}) = \mathbf{W}^h \cdot \mathbf{e}_{u_i}$, where $\mathbf{W}^h \in \mathbb{R}^{d\times d}$ is the weight for head h. Similar, we can compute H-head OT spaces for items. To this end, we can obtain H cluster-aware user representations $\{\mathbf{e}'^{(1,l)}_u, \ldots, \mathbf{e}'^{(H,l)}_u\} \subset \mathbb{R}^d$ and items' $\{\mathbf{e}'^{(1,l)}_i, \ldots, \mathbf{e}'^{(H,l)}_i\} \subset \mathbb{R}^d$, where each element is calculated as Eq. (9). Then, we concatenate them to produce:

$$\mathbf{e}'^{(l)}_u = \text{CONCAT}(\mathbf{e}'^{(1,l)}_u, \ldots, \mathbf{e}'^{(H,l)}_u) \cdot \mathbf{W}_u^{(l)},$$
$$\mathbf{e}'^{(l)}_i = \text{CONCAT}(\mathbf{e}'^{(1,l)}_i, \ldots, \mathbf{e}'^{(H,l)}_i) \cdot \mathbf{W}_i^{(l)}, \tag{10}$$

where $\{\mathbf{e}'^{(l)}_u, \mathbf{e}'^{(l)}_u\} \in \mathbb{R}^d$ are multi-head cluster-aware representations for user u and item i, respectively; $\{\mathbf{W}_u^{(l)}, \mathbf{W}_i^{(l)}\}$ are learned weights that project multi-head attentions of users and items to d-dimensional space.

4.3.3 ***Mixing Local and Non-local Attentions.*** Recall that $\mathbf{e}_u^{(l)}$ and $\mathbf{e}_i^{(l)}$ in Eq. (2) collect local messages from neighbors, while $\mathbf{e}'^{(l)}_u$ and $\mathbf{e}'^{(l)}_i$ in Eq. (10) capture non-local messages from centroids, we further introduce the idea of mixing both local and non-local attention by using *mixup* techniques [48]:

$$\mathbf{e}''^{(l)}_u = \lambda^{(l)} \mathbf{e}_u^{(l)} + (1-\lambda^{(l)}) \mathbf{e}'^{(l)}_u,$$
$$\mathbf{e}''^{(l)}_i = \lambda^{(l)} \mathbf{e}_i^{(l)} + (1-\lambda^{(l)}) \mathbf{e}'^{(l)}_i, \tag{11}$$

where $\mathbf{e}''^{(l)}_u$ and $\mathbf{e}''^{(l)}_i$ are the final representations for downstream tasks; $\lambda^{(l)} \in [0, 1]$ is the l-hop mixing coefficient that is simply sampled from a uniform distribution[3]: $\lambda^{(l)} \sim \text{Uniform}(0, 1)$, for each hop l. By doing so, mixup extends the training distribution

[3]The original mixup method uses Beta distribution, where Beta(a, a) is Uniform distribution when $a = 1$, Bell-shaped when $a > 1$, and Bimodal when $a < 1$.

by linear interpolations of embedding manifolds, which has been shown to effectively increase the generalization ability [17, 41, 48].

4.4 Optimization with GOTNet

4.4.1 ***Joint Training.*** Now, we can plug the user/item embeddings Eq. (11) into Eq. (3)-(4) to predict preference scores. To this end, we jointly learn GNNs and k-Means in one unified model. Combining the losses in Eq. (5) and Eq. (8), the overall objective of GOTNet is given as:

$$\mathcal{L}_{GOTNet} = \mathcal{L}_{BPR} + \beta \cdot \sum_{l=0}^{L-1} \sum_{h=1}^{H} \mathcal{L}_{uOT}^{(h,l)} + \gamma \cdot \sum_{l=0}^{L-1} \sum_{h=1}^{H} \mathcal{L}_{iOT}^{(h,l)}, \quad (12)$$

where $\mathcal{L}_{uOT}^{(h,l)}$ and $\mathcal{L}_{iOT}^{(h,l)}$ are the Sinkhorn losses for users and items for h-head in the l-th hop, respectively; β and γ are their corresponding regularized coefficients. This loss is differentiable w.r.t. all variables. The algorithm of GOTNet is summarized in Algorithm 1.

4.4.2 ***Model-agnostic.*** As our GOTNets are built on top of GNNs and the major network architectures keep unchanged, our GOTNets are model-agnostic and can be seamlessly applied to any GNNs, like PinSage [46], NGCF [40], and LightGCN [14]. In fact, if we simply set $\beta = \gamma = 0$ in Eq. (12) and mixup coefficients $\lambda^{(l)} = 1$ in Eq. (11), our GOTNets are essentially equivalent to the original GNNs. More importantly, our GOTNet is fully compatible with existing regularization & normalization techniques (*e.g.*, PairNorm [50], DropEdge [33]), which are commonly used for deep GNNs [22]. We leave this extension in the future.

4.4.3 ***Model Complexity.*** Compared to original GNNs, GOTNets involve an extra cost: k-Means clustering via Optimal Transport as shown in Eq. (8). For clustering users, the cost of Greenkhorn is $O(NKd)$ [1], where N denotes the number of users, K is the number of centroids, and d is the GNN embedding size. Similar complexity can be obtained for clustering items. In practice, the values of K and d are often very small. Therefore, the complexity of GOTNet remains the same order as GNNs.

4.4.4 ***Discussion.*** It is worth mentioning that our GNOTNet is slightly different from Cluster-GCN [7], even though they both use clustering techniques. Cluster-GCN uses graph clustering to obtain a set of subgraphs and performs graph convolutions within these subgraphs. In contrast, our GOTNet performs clustering on GNN embeddings to exploit long-range dependencies. Our work is more relevant to recent fast Transformers with clustered attentions [38], in which the queries are grouped into clusters, and attention is only computed once per cluster. This significantly reduces the complexity of attention mechanisms for large-scale data.

5 EXPERIMENTS

5.1 Experimental Settings

5.1.1 ***Datasets.*** In this work, we conduct experiments on four benchmark datasets: Movielens-1M, Gowalla, Yelp-2018, and Amazon-Book. The datasets are briefly introduced in Appendix B. We follow the same settings as [14, 40] to split the datasets into training, validation, and testing sets.

5.1.2 ***Evaluation Metrics.*** We choose the widely used Recall@k and NDCG@k as the evaluation metrics [14, 40]. For fair comparison, we compute default Recall@20 and NDCG@20 by the all-ranking protocol as in [14]. We report the average results of ten independent experiments for both metrics.

5.1.3 ***Compared Methods.*** In this work, we compare with three groups of baselines: 1) Factorization-based models: BPR-MF [32] and NeuMF [15]; 2) GNN-based models: PinSage [46], NGCF [40], and LightGCN [14]; 3) Non-local models: Geom-GCN [30], NL-GCN [27]. For Geom-GCN, NL-GCN and GOTNet, we can choose any GNNs as their backbones, such as PinSage, NGCF, and LightGCN. The baselines are provided in Appendix B.

5.1.4 ***Parameter Settings.*** For all models, the embedding size d of users and items (Eq. (1)) is searched within $\{16, 32, 64, 128\}$. For all baselines, their hyper-parameters are initialized as in their original settings and are then carefully tuned to attain optimal performance for all datasets.

For the GNN modules inside Geom-GCN, NL-GCN, and GOTNet, we use the same hyper-parameters as their backbones, such as batch size, , learning rate in Adam optimizer, etc. For GOTNets, we simply set the clustering loss regularizers as $\beta = \gamma = 0.01$, the entropy regularizer as $\varepsilon = 0.01$, the number of user/item clusters: $K \approx 0.01N$ and $P \approx 0.01M$, and the number of head $H = 1$. The parameter sensitivity of GOTNet will be investigated later.

5.2 Performance Comparison

The performance comparisons of all methods are summarized in Table 1. The best performance is highlighted in boldface and the second is underlined. Our proposed models consistently yield the best performance across all datasets. From Table 1, we have the following key observations and conclusions:

- Compared with factorization-based methods BPR-MF and NeuMF, GNN-based methods (*e.g.,* PinSage, NGCF, and LightGCN) generally achieve better performance for all cases. This greatly suggests the benefits of exploiting high-order proximity between users and items in the bipartite graph. As much more informative messages being aggregated from neighbors, GNN-based methods are capable of inferring explicit or implicit correlations among neighbors via graph convolutions.
- Among GNN-based methods, non-local GNN methods (*e.g.,* Geom-GCNs, NL-GCNs, and GOTNets) perform better than vanilla GNNs. This is because non-local operators allow to capture long-range dependencies from distant nodes, while original graph convolutional operators only aggregate information from local neighbors. In addition, NL-GCNs are slightly better than Geom-GCNs. Geom-GCNs require pre-trained node embedding to construct latent spaces, which may be not task-specific. In contrast, NL-GCNs employ calibration vectors to refine non-local neighbors, which can be jointly trained with GNNs, leading to better results than Geom-GCNs. Nevertheless, NL-GCNs only calibrate the output embeddings of GNNs, which lacks adaptability.
- GOTNets consistently outperforms NL-GCNs by a large margin on all datasets. Comparing to NL-GCNs, GOTNets have on average 13.74% improvement in terms of Recall@20, and over

Table 1: The performance of different models. %Improv denotes the relative improvement of GOTNets over NL-GCNs.

	MovieLens-1M		Gowalla		Yelp-2018		Amazon-Book	
Methods	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20	Recall@20	NDCG@20
BPR-MF [32]	0.2503	0.1987	0.1298	0.1112	0.0431	0.0352	0.0253	0.0197
NeuMF [15]	0.2612	0.2047	0.1403	0.1211	0.0448	0.0364	0.0260	0.0211
PinSage [46]	0.2731	0.2314	0.1431	0.1202	0.0463	0.0374	0.0286	0.0234
Geom+PinSage [30]	0.2717	0.2219	0.1489	0.1216	0.0467	0.0366	0.0303	0.0235
NL+PinSage [27]	0.2810	0.2378	0.1522	0.1371	0.0487	0.0391	0.0314	0.0246
GOTNet+PinSage	0.2971	0.2502	0.1663	0.1493	0.0562	0.0460	0.0340	0.0264
%Improv.	5.73%	5.21%	9.26%	8.90%	15.40%	17.65%	8.28%	7.32%
NGCF [40]	0.2702	0.2243	0.1563	0.1325	0.0581	0.0478	0.0338	0.0263
Geom+NGCF [30]	0.2711	0.2304	0.1591	0.1346	0.0583	0.0470	0.0341	0.0265
NL+NGCF [27]	0.2750	0.2343	0.1614	0.1459	0.0601	0.0488	0.0356	0.0271
GOTNet+NGCF	0.2896	0.2557	0.1786	0.1591	0.0652	0.0526	0.0411	0.0307
%Improv.	5.31%	9.09%	10.66%	6.85%	8.49%	7.79%	15.45%	13.28%
LightGCN [14]	0.2783	0.2399	0.1826	0.1549	0.0649	0.0541	0.0418	0.0320
Geom+LightGCN [30]	0.2774	0.2380	0.1818	0.1547	0.0651	0.0538	0.0405	0.0316
NL+LightGCN [27]	0.2801	0.2412	0.1829	0.1560	0.0659	0.0545	0.0443	0.0324
GOTNet+LightGCN	**0.3063**	**0.2762**	**0.2122**	**0.1772**	**0.0784**	**0.0651**	**0.0490**	**0.0361**
%Improv.	9.35%	14.51%	16.02%	13.59%	18.97%	19.45%	10.61%	11.42%

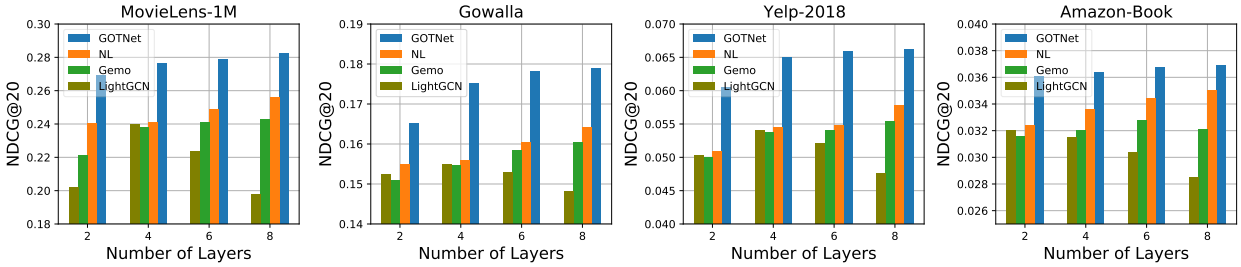

Figure 3: The results of different GNNs with varying number of layers.

14.75% improvements with respect to NDCG@20. These improvements are attributed to our multi-heads node-centroid attentions and attention mixups. Through clustering users and items in multiple OT spaces, GOTNets augment both user-to-user and item-to-item correlations, which provides a better way to help users explore their inventory (*e.g.*, people who like an items also like items in same group). Also, our attention mixups extend the distribution of training data by linear interpolations of local and non-local embeddings, which largely improves the generalization and robustness of GNNs.

It is noticed that GOTNet can generalize both Geom-GCNs and NL-GCNs by relaxing certain constraints: 1) If we detach cluster-aware centroid vectors from Sinkhorn loss, these vectors have the potential to serve as calibration vectors in NL-GCNs; 2) If we simply set the number of clusters to be equal to the number of nodes, GOTNet is then able to measure node-node attentions as Geom-GCNs do.

In summary, the experimental results demonstrate the superiority of our proposed GOTNet. Specifically, it can empower the existing state-of-the-art GNNs to capture long-range messages by using additional non-local attention mechanisms.

5.3 Non-local Aggregation Analysis

Here we further study the benefits of non-local aggregation from two aspects: (1) over-smoothing and (2) data sparsity. Due to page limitations, we only report the results of LightGCN and its variants, while we have similar trends for both PinSage and NGCF.

5.3.1 ***Over-smoothing.*** The over-smoothing phenomenon exists when training deep GNNs [23]. To illustrate this influence, we conduct experiments with varying numbers of layers L within $\{2, 4, 6, 8\}$. Figure 3 presents the results in terms of NDCG@20. We observe that the peak performance of LightGCN is achieved when stacking 2 or 4 layers, but increasing depth will cause performance degradation. In contrast, non-local GNNs (*e.g.*, Geom-GCN, NL-GCN, and GOTNet) continue to benefit from deeper architectures. Specifically, our GOTNet clearly outperforms Geom-GCN and NL-GCN for all datasets. The reason is that GOTNet explicitly adopts a mixture of local and non-local aggregations to prevent all node representations converging to the same values even with deeper structures. Additionally, our multi-head projections allow users and items to have distinguishable representations in multiple different spaces, which makes node representations less similar and alleviates the over-smoothing problem.

Table 2: The performance for sparse recommendation.

	Yelp-2018		Amazon-Book	
Methods	R@20	N@20	R@20	N@20
LightGCN	0.0314	0.0286	0.0211	0.0190
Geom+LightGCN	0.0321 (+2.22%)	0.0293 (+2.45%)	0.0220 (+4.27%)	0.0198 (+4.21%)
NL+LightGCN	0.0324 (+3.18%)	0.0302 (+5.59%)	0.0234 (+10.9%)	0.0206 (+8.42%)
GOTNet+LightGCN	0.0338 (+7.64%)	0.0327 (+14.34%)	0.0239 (+13.27%)	0.0221 (+16.32%)

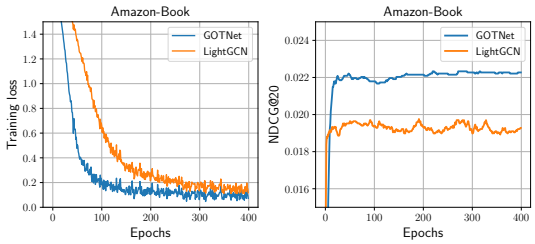

Figure 4: Training curves and test NDCG@20 of LightGCN and GONet for sparse Amazon dataset.

5.3.2 ***Sparse Recommendation.*** As discussed before, graph convolutions are essentially local operators, which gives an advantage to high-degree nodes to collect enough information from their neighbors. However, many real-world graphs are often long-tailed and sparse where a significant fraction of nodes has low degrees. For these tail nodes, GNNs only aggregate messages from a small number of neighbors, which can be biased or underrepresented. Towards this end, we further investigate the effectiveness of non-local GNNs to sparsify recommendations. For this purpose, we focus on users and items that have at least 5 interactions, but less than 10 interactions. Table 2 shows the results for Yelp and Amazon. We see similar trends for other datasets, and omit them here.

As can be seen, non-local methods perform better than LightGCN, verifying that non-local mechanisms facilitate the representation learning for inactive users/items. In particular, GOTNet reaches a relative improvement over LightGCN by average 10.46% and 15.33% in terms of Recall and NDCG, respectively. These comparisons demonstrate that non-local node-centroid attentions used in GOTNet are able to capture long range dependencies. For inactive users/items, their clusters can identify groups of users/items that appear to have similar representations. As such, more meaningful messages can be passed via node-centroid attentions, rather than their sparse neighbors. Figure 4 also shows the training curves of training loss and testing NDCG. LightGCN attains stable training losses but produce fluctuations in the stage of test. Conversely, GOTNet is more robust against the issue of over-fitting.

5.4 Study of GOTNet

We further investigate the parameter sensitivity of GOTNet w.r.t. the following hyper-parameters: two regularizer parameters $\{\beta, \gamma\}$ in Eq. (12), entropy regularizer ε in Eq. (8), the number of users/items clusters K and P, and the number of heads H. We mainly use the Amazon dataset for hyper-parameter studies.

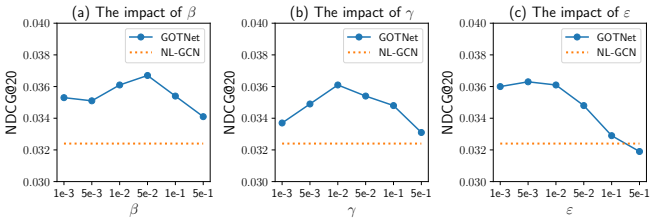

Figure 5: Parameter sensitivity of GOTNet on Amazon.

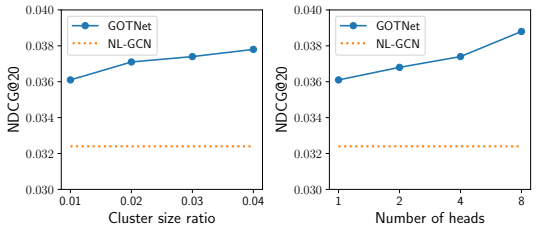

Figure 6: The impacts of clustering sizes and attention heads.

5.4.1 ***Regularizers.*** Here we analyze the performance sensitivity of GOTNet with different regularizers $\{\beta, \gamma, \varepsilon\}$. Figure 5(a) and 5(b) show the performance by changing one parameter while fixing the rest as 0.01. As can be seen, GOTNet is relatively insensitive to β and γ, and achieves the best performance with $\beta = 5e^{-2}$ and $\gamma = 1e^{-2}$. Also, we find that the non-zero choices of β and γ generally outperform NL-GCNs, indicating the importance of the clustering losses in our models.

Figure 5(c) also shows the impact of ε in Sinkhorn loss. The choices of ε is crucial in OT solvers. Theoretically, we can get closer to true Optimal Transport plan when ε get smalls (*e.g.*, $\varepsilon \leftarrow 0$). Sinkhorn's algorithm, however, requires more iterations to converge [8], leading to more running time. We observe that setting ε within $[5e^{-3}, 5e^{-2}]$ is a good trade-off in our experiments.

5.4.2 ***Clustering Size and Multi-head.*** We also conduct experiments to see whether using large cluster sizes and more heads is beneficial to final results. To this end, we vary the clustering size of users and item $r = \frac{K}{N} = \frac{P}{M}$ from 0.01 to 0.04, and the number of heads H within $[1, 8]$. From Figure 6, we find that the performance increases with larger cluster sizes as well as more heads. Such results are expected since higher cluster sizes and more heads lead to a more discriminative representations of users/items.

6 CONCLUSION AND FUTURE WORK

In this work, we propose a simple yet effective method GOTNet to improve the ability of capturing long-range dependencies. Instead of training deep architectures, GOTNet combines k-Means clustering and GNNs to obtain compact centroids, which can be used to deliver long-range messages via non-local attentions. Extensive experiments suggest that GOTNet can empower many of existing GNNs. For future work, we plan to improve our GOTNet by incorporating several existing regularization & normalization techniques that generally contribute to effective training of GNNs.

REFERENCES

[1] Jason Altschuler, Jonathan Weed, and Philippe Rigollet. 2017. Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration. In *NeurIPS*. 1961–1971.

[2] Guillermo D Canas and Lorenzo A Rosasco. 2012. Learning probability measures with respect to optimal transport metrics. In *NeurIPS*. 2492–2500.

[3] Jiangxia Cao, Xixun Lin, Shu Guo, Luchen Liu, Tingwen Liu, and Bin Wang. 2021. Bipartite graph embedding via mutual information maximization. In *WSDM*. 635–643.

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

[5] Huiyuan Chen, Lan Wang, Yusan Lin, Chin-Chia Michael Yeh, Fei Wang, and Hao Yang. 2021. Structured graph convolutional networks with stochastic masks for recommender systems. In *SIGIR*. 614–623.

[6] Liqun Chen, Zhe Gan, Yu Cheng, Linjie Li, Lawrence Carin, and Jingjing Liu. 2020. Graph optimal transport for cross-domain alignment. In *ICML*. 1542–1553.

[7] 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.

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

[9] Maziar Moradi Fard, Thibaut Thonet, and Eric Gaussier. 2020. Deep k-means: Jointly clustering with k-means and learning representations. *Pattern Recognition Letters* (2020), 185–192.

[10] Aude Genevay, Gabriel Dulac-Arnold, and Jean-Philippe Vert. 2019. Differentiable deep clustering with cluster size constraints. *arXiv preprint arXiv:1910.09036* (2019).

[11] Aude Genevay, Gabriel Peyré, and Marco Cuturi. 2018. Learning generative models with sinkhorn divergences. In *AISTATS*. 1608–1617.

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

[13] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In *IJCAI*. 1725–1731.

[14] 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.

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

[16] Han Hu, Jiayuan Gu, Zheng Zhang, Jifeng Dai, and Yichen Wei. 2018. Relation networks for object detection. In *CVPR*. 3588–3597.

[17] 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.

[18] 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.

[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] Charlotte Laclau, Ievgen Redko, Basarab Matei, Younes Bennani, and Vincent Brault. 2017. Co-clustering through optimal transport. In *ICML*. 2492–2500.

[22] Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. 2021. Training Graph Neural Networks with 1000 layers. In *ICML*. 6437–6449.

[23] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*. 3538–3545.

[24] 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).

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

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

[27] Meng Liu, Zhengyang Wang, and Shuiwang Ji. 2021. Non-Local Graph Neural Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[28] Mikko I Malinen and Pasi Fränti. 2014. Balanced K-Means for Clustering. In *Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*. 32–41.

[29] 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.

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

[31] David Pollard. 1982. Quantization and the method of k-means. *IEEE Transactions on Information theory* (1982), 199–205.

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

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

[34] Tim Salimans, Han Zhang, Alec Radford, and Dimitris Metaxas. 2018. Improving GANs Using Optimal Transport. In *ICLR*.

[35] Jianing Sun, Yingxue Zhang, Wei Guo, Huifeng Guo, Ruiming Tang, Xiuqiang He, Chen Ma, and Mark Coates. 2020. Neighbor interaction aware graph convolution networks for recommendation. In *SIGIR*. 1289–1298.

[36] 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.

[37] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[38] Apoorv Vyas, Angelos Katharopoulos, and François Fleuret. 2020. Fast transformers with clustered attention. *NeurIPS* (2020), 21665–21674.

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

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

[41] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. 2021. Mixup for Node and Graph Classification. In *WWW*. 3663–3674.

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

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

[44] Yujia Xie, Xiangfeng Wang, Ruijia Wang, and Hongyuan Zha. 2020. A fast proximal point method for computing exact wasserstein distance. In *UAI*. 433–453.

[45] Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. 2017. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *ICML*. 3861–3870.

[46] 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.

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

[48] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. 2018. mixup: Beyond Empirical Risk Minimization. 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] 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*, 7793–7804.

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

APPENDIX

A. Optimal Transport

Optimal Transport is a mathematical model that defines distances or similarities between objects such as probability distributions, either continuous or discrete, as the cost of an optimal transport plan from one to the other. By regarding the "cost" as distance, Wasserstein Distance is a commonly used metric for matching two distributions in OT [6].

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 $\boldsymbol{\nu} = \sum_{j=1}^{m} \mathbf{v}_j \delta_{\boldsymbol{y}_j}$, 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[4] of size n and m, respectively. The Wasserstein Distance between $\boldsymbol{\mu}$ and $\boldsymbol{\nu}$ is then

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

defined as:

$$\begin{aligned}\mathcal{D}_w(\boldsymbol{\mu}, \boldsymbol{\nu}) &= \inf_{\boldsymbol{\gamma} \in \Pi(\boldsymbol{\mu}, \boldsymbol{\nu})} \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{y}) \sim \boldsymbol{\gamma}}[c(\boldsymbol{x}, \boldsymbol{y})] \\ &= \min_{\mathbf{T} \in \Pi(\mathbf{u}, \mathbf{v})} \sum_{i=1}^{n} \sum_{j=1}^{m} \mathbf{T}_{ij} \cdot c(\boldsymbol{x}_i, \boldsymbol{y}_j),\end{aligned} \tag{13}$$

where $\Pi(\mathbf{u}, \mathbf{v}) = \left\{\mathbf{T} \in \mathbb{R}_+^{n \times m} \mid \mathbf{T}\mathbf{1}_m = \mathbf{u}, \mathbf{T}^\top \mathbf{1}_n = \mathbf{v}\right\}$; $\mathbf{1}_n$ 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*.

B. Algorithm of GOTNet

The training process of GOTNet is summarized in Algorithm 1.

Algorithm 1: GOTNet

Input: Training set $\{(u, i)\}$, number of GNN layers L, number of user and item clusters K and P, number of heads H for OT spaces, and regularization coefficients α, β, γ, and ε (Sinkhorn loss in Eq. (8)).

Initialize user and item centroids $C^{(h,l)}$ and $D^{(h,l)}$ for all $0 \le l \le L-1$, and $1 \le h \le H$.
for *each mini-batch* **do**
  for $l \leftarrow 1$ ***to*** L **do**
    Compute GNN embeddings $\mathbf{e}_u^{(l)}$ and $\mathbf{e}_i^{(l)}$ by Eq. (2).
    Compute node cluster assignments and centroids by solving Eq. (8) via Greenkgorn.
    Compute non-local embeddings $\mathbf{e'}_u^{(l)}$ and $\mathbf{e'}_i^{(l)}$ by Eq. (10).
    Compute mixed embeddings $\mathbf{e''}_u^{(l)}$ and $\mathbf{e''}_i^{(l)}$ by Eq. (11).
    Perform pooling and inference by Eq. (3) and Eq. (4).
  end
  Compute the loss $\mathcal{L}_{GOTNet}$ in Eq. (12);
  Update the parameters of GNN $\boldsymbol{\Theta}$, projection function f_θ, and the weights in multi-head attentions.
end

Output: A well-trained GOTNet to predict $\hat{y}_{ui}$.

C. Dataset and Baselines

Datasets. We use four publicly available datasets as follows:

- **Movielens-1M**[5] is a widely used movie rating dataset that contains one million user-movie ratings. We transform the rating scores into binary values, indicating whether a user rated a movie.
- **Gowalla**[6] is obtained from the location-based social website *Gowalla*, where users share their locations by checking-in.
- **Yelp-2018**[7] is released by the *Yelp* 2018 challenge, where local business like restaurants are viewed as items.
- **Amazon-Book**[8] contains a large corpus of user reviews, ratings, and product metadata, collected from *Amazon.com*. We choose the largest category Books.

Table 3: The statistics of four benchmark datasets.

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-2018	31, 668	38, 048	1, 561, 406	0.130%
Amazon-Book	52, 643	91, 599	2, 984, 108	0.062%

The statistics of the datasets are briefly summarized in Table 3. We follow the same strategy as NGCF [40] and LightGCN [14] to split the train/valid/test data sets[9]. Due to the sparisity of datasets, we use the 10-core setting of the user-item graphs to ensure that all users and items have at least 10 interactions.

For each dataset, we randomly select 80% of historical interactions of each user to generate 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.

Baselines. We compare our model to following baselines:

- **BPR-MF** [32]: A classic model that seeks to optimize the Bayesian personalized ranking loss. We employ matrix factorization as its preference predictor.
- **NeuMF** [15]: NeuMF learns nonlinear interactions between user and item embeddings via a multi-layer perceptron as well as a generalized matrix factorization.
- **PinSage** [46]: PinSage designs a random-walk based sampling method to sample neighbors for each node and then combines graph convolutions to compute node embeddings.
- **NGCF** [40]: This method explicitly learns the collaborative signal in the form of high-order connectivities by performing embedding propagation from user-item graphs.
- **LightGCN** [14]: LightGCN is a simplified version of NGCF by removing the feature transformation and nonlinear activation, which achieves the state-of-the-art performance
- **Geom-GCN** [30]: It explores to capture long-range dependencies by using bi-level geometric aggregations from nodes. There are three variants of Geom-GCN, we simply report the best results. By choosing PinSage, NGCF, and LightGCN as its backbones, we have their corresponding Geom-GCNs.
- **NL-GCN** [27]: A recently proposed non-local GNN framework that uses efficient attention-guided sorting mechanism, which enables non-local aggregation through convolutions. Similar, we can select PinSage, NGCF, and LightGCN as its backbones.

For Geom-GCN, NL-GCN and GOTNet, we can choose any GNNs as their backbones, such as PinSage, NGCF, and Light-GCN.

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