A Hierarchical Self-Attentive Model for Recommending User-Generated Item Lists

Yun He, Jianling Wang, Wei Niu and James Caverlee
Department of Computer Science and Engineering, Texas A&M University
{yunhe,jlwang,caverlee}@tamu.edu;weiniu.2010@gmail.com

ABSTRACT

User-generated item lists are a popular feature of many different platforms. Examples include lists of books on Goodreads, playlists on Spotify and YouTube, collections of images on Pinterest, and lists of answers on question-answer sites like Zhihu. Recommending item lists is critical for increasing user engagement and connecting users to new items, but many approaches are designed for the item-based recommendation, without careful consideration of the complex relationships between items and lists. Hence, in this paper, we propose a novel user-generated list recommendation model called AttList. Two unique features of AttList are careful modeling of (i) hierarchical user preference, which aggregates items to characterize the list that they belong to, and then aggregates these lists to estimate the user preference, naturally fitting into the hierarchical structure of item lists; and (ii) item and list consistency, through a novel self-attentive aggregation layer designed for capturing the consistency of neighboring items and lists to better model user preference. Through experiments over three real-world datasets reflecting different kinds of user-generated item lists, we find that AttList results in significant improvements in NDCG, Precision@k, and Recall@k versus a suite of state-of-the-art baselines. Furthermore, all code and data are available at https://github.com/heyunh2015/AttList.

CCS CONCEPTS

• **Information systems** → **Recommender systems**;

KEYWORDS

Recommender System, User-Generated Item Lists, Self-Attention

ACM Reference Format:
Yun He, Jianling Wang, Wei Niu and James Caverlee. 2019. A Hierarchical Self-Attentive Model for Recommending User-Generated Item Lists. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3357384.3358030

1 INTRODUCTION

User-generated item lists are a widespread feature of many platforms. Examples include music playlists on Spotify, collections of videos on YouTube, lists of books on Goodreads, wishlists of products on Amazon, collections (boards) of images (pins) on Pinterest, and lists of interesting answers on question-answer sites like Zhihu. These item lists directly power user engagement – for example, more than 50% of Spotify users listen to playlists, accounting for more than 1 billion plays per week [33]; and Pinterest users have curated more than 3 billion pins to about 4 billion boards [7].

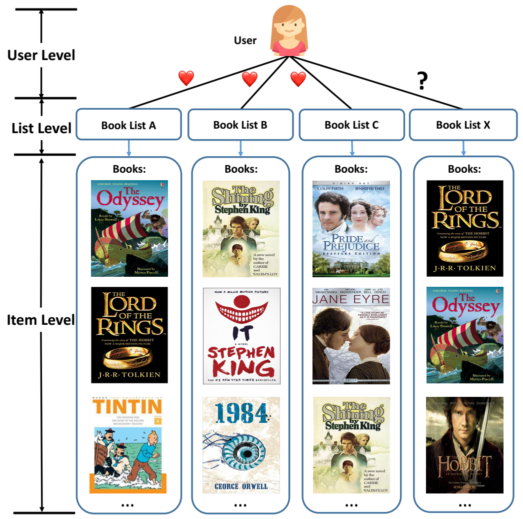


Figure 1: Example: A Goodreads user likes three different lists (A, B, and C), each composed of a collection of books. Our proposed model exploits this user-list-item hierarchical structure to recommend additional lists (e.g., X).

Across platforms, user-generated item lists are manually created, curated, and managed by users, providing a unique perspective into how items (e.g., songs, videos, books, and answers) can be grouped together. Lists may be organized around theme, genre, mood, or other non-obvious pattern, so users can easily explore and consume correlated items together. To illustrate, Figure 1 presents an example of several lists on Goodreads, a book sharing platform. This user likes book lists organized around different themes (e.g., adventure and horror).

But how can we effectively connect users to the right lists? Traditional *item-based recommendation* has shown good success in modeling user preferences for specific items [12, 17]. In contrast, *list-based recommendation* is a nascent area with key challenges in modeling the complex interactions between users and lists, between lists and items, and between users and items. For example, a user's preference for a list may be impacted by factors such as the overall theme of the list, all of the items or just a few of the items on the list, the position of items on the list, and so on. The user in Figure 1 may prefer list A for its classic mix of young adult adventure stories,

CIKM '19, November 3–7, 2019, Beijing, China
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3358030

but only like list B for its first two horror novels (by the same author Stephen King), regardless of the rest of the items on the list. Furthermore, the variety across book lists liked by this user raises tough challenges in terms of modeling preference for new lists (as in the unknown list X). A few previous efforts [3, 23] have proposed to treat user-list and user-item interactions through a collective matrix factorization approach [30], where a user-list interaction matrix and a user-item matrix are jointly factorized [3, 23]. However, it is non-trivial to weight user-list and user-item interactions in the loss function for optimizing the list recommendation task. The number of user-item interactions are often much more than user-list interactions and thus dominate the loss. This imbalance in the number of user-list and user-item interactions stems from the hierarchical structure of item lists, i.e., a list often contains tens or even hundreds of items, as shown in Section 5.1.

Hence, with these challenges in mind, we propose a novel hierarchical self-attentive model for recommending user-generated item lists that is motivated by two key observations:

Hierarchical user preferences. First, users and user-generated item lists naturally form a hierarchical structure that could be helpful for modeling user preferences. To illustrate, Figure 1 shows how a user (from the top) can like a collection of different lists, which in turn are composed of different items (at the bottom). Hence, it is natural to represent lists by their constituent items, and represent users by the lists that they prefer. Our first motivating observation then is *user preferences can be propagated bottom-up from the item level to the list level and finally to the user level.*

Item and list consistency. Second, the similarity between items and their neighboring items (whether by genre, theme, or other pattern) is a key clue for modeling the importance of those items towards reflecting user preferences. Likewise, the similarity between a list and its neighboring lists can reveal the importance of those lists to the user's overall preferences. We refer to this similarity as *item consistency* and *list consistency*. For example, if a scary book (e.g., *The Shining*) is curated in a list (like list C in Figure 1) mainly composed of romance books (e.g., *Pride and Prejudice*, *Jane Eyre*), then it is less informative in terms of the overall list characteristics. Likewise, a romance list may be less informative about the user preference if it is not consistent with the rest of the user's profile. Our second motivating observation is *the lower the consistency between an item and the rest of the items on a list, the less likely it can reveal the list's characteristics. And conversely, the higher the consistency, the more likely it can reveal the list's characteristics.* Similarly, the higher the consistency, the more likely a list can reveal the user's preference. That is, these inconsistent items and lists should be assigned lower weights to represent the list and the user.

These observations motivate us to attack the user-generated item list recommendation problem with a *hierarchical* user-list-item model that naturally incorporates a *self-attentive* aggregation layer to capture item and list consistency by correlating them to their neighboring items and lists. In summary:

- We study the important yet challenging problem of recommending user-generated item lists – a complex recommendation scenario that poses challenges beyond existing item-based recommenders.
- We propose a hierarchical self-attentive recommendation model (AttList), naturally fitting the user-list-item hierarchical structure, and enhanced by a novel self-attentive aggregation layer to capture user and list consistency.
- Experiments on three real-world datasets (Goodreads, Spotify, and Zhihu) demonstrate the effectiveness of AttList versus state-of-the-art alternatives. Further, all code and data are released to the research community for further exploration.

2 RELATED WORK

Recommendation with Implicit Feedback. Early recommender systems mainly focused on rating prediction based on explicit feedback, such as ratings from reviewers [17]. However, implicit feedback is usually much easier to collect, as is the case in our user-list interactions dataset described in Section 5.1. Thus, more attention has been paid to ranking items based on user preference reflected from implicit feedback, such as purchase history, clicks, or likes. Among the various ranking algorithms [11, 12, 27, 28], Bayesian Personalized Ranking (BPR) [28] is a well-known pair-wise ranking framework. Recently, there have been some efforts to improve item recommendation by introducing non-linear transformations with neural networks [6, 10, 19–21, 35], where Neural Collaborative Filtering (NCF) [10] is a typical example, where a multi-layer perceptron model and a generalized matrix factorization model are combined to learn the user preference. Their experiments show that neural based recommender systems outperform traditional methods like matrix factorization and BPR.

User-Generated Item Lists. Recently, user-generated item lists receive more and more interest. Lo et al. [24] analyze the growth of image collections on Pinterest. Lu et al. [22] and Eksombatchai et al. [7] distill user preference from user-generated item lists to enhance individual item recommendation. Besides, Greene et al. [8] support users to continue their user-lists on Twitter. In this paper, we focus on recommending user-generated item lists.

Although many existing algorithms successfully recommend individual and independent items to users, approaches for recommending user-generated item lists are not fully explored. There are two studies close to ours: LIRE [23] and EFM [3]. The List Recommending Model (LIRE) [23] is a Bayesian-based pairwise ranking approach, which takes user preference over items within the lists into consideration when inferring the user-lists preference, whereby each list is modeled as a linear combination of the items. The Embedding Factorization Model (EFM) [3] uses the co-occurrence information between items and lists to improve list recommendation. These models are based on a *collective matrix factorization framework* [30] which jointly factorizes several related matrices, e.g., a user-list interaction matrix and a user-item interaction matrix. In contrast, the proposed AttList is designed to capture the user-list-item hierarchical structure, which is better customized for user-generated item list recommendation. Furthermore, both LIRE and EFM incorporate both user-item and user-list interactions; in contrast, AttList requires only user-list interactions to uncover user preferences though it can be easily extended to incorporate additional user-item interactions as well.

Attention Networks. AttList incorporates an attention layer inspired by recent work like the attention mechanism proposed in

[1] and the *transformer* in [34] for machine translation, where it can be used to relieve the long-range dependency problem in RNNs. Attention networks have been used in recommendation and can be grouped into two classes: vanilla attention and self-attention [39].

Vanilla Attention. In this case, a two-layer network is normally used to calculate the attention score by matching a sequence of representations of the target's components against a learnable global vector. Xiao et al. [36] propose an attentional factorization machine model where the importance of feature interactions is learned by attention networks. Zhou et al. [41] employ attention networks to calculate the weights for different user behaviors (e.g., reading the comments, carting and ordering) for modeling the user preference in E-commerce sites. Chen et al. [5] propose an attentive collaborative filtering framework, where each item is segmented into component-level elements, and attention scores are learned for these components for obtaining a better representation of items. Attention networks are also applied in group recommendation [2], sequential recommendation [37], review-based recommendation [4, 29, 32] and context-aware recommendation [26].

Self-Attention. In this case, attention scores are learned by matching the representations against themselves and update each representation by incorporating the information of other representations. Zhou et al. [40] apply self-attention networks to capture the inner-relations among different user behaviors for modeling the user preference. Other approaches [13, 15, 38] use self-attention networks for sequential recommendation where the item-item relationship is inferred from user's historical interactions. Self-attention is also applied for point-of-interest recommendation [25]. In this paper, self-attention is used to distill the consistency of neighboring items and lists for revealing user preferences.

3 PROBLEM FORMULATION

We denote the set of users, lists and items as $\mathcal{U}$ and $\mathcal{L}$ and $\mathcal{T}$ where the size of these sets is $|\mathcal{U}|$, $|\mathcal{L}|$ and $|\mathcal{T}|$ respectively. Every list in $\mathcal{L}$ is composed of items from $\mathcal{T}$, where the containment relationship between $\mathcal{L}$ and $\mathcal{T}$ is denoted as C. We reserve u to denote a user and l to denote a list. We define the user-list interaction matrix as $\mathbf{R} \in \{0, 1\}^{|\mathcal{U}|\times|\mathcal{L}|}$, where r_{ul} indicates the feedback from u to l, typically in the form of a "like", vote, following, or other feedback signal. Since the feedback is naturally binary, we further let $r_{ul} = 1$ indicate l has feedback from u and $r_{ul} = 0$ otherwise.

The *user-generated item list recommendation* problem takes as input the users $\mathcal{U}$, lists $\mathcal{L}$, items $\mathcal{T}$, user-list interactions $\mathbf{R}$ and C, the containment relationship between lists $\mathcal{L}$ and items $\mathcal{T}$. It outputs K item lists for each user, where each list l is ranked by the estimated preference of u to l, denoted as $\hat{r}_{ul}$.

4 THE PROPOSED MODEL: ATTLIST

As discussed in Section 1, for user-generated item list recommendation, user preference naturally propagates hierarchically from items to lists and finally to users. Furthermore, the user preference on an item is affected by the rest of the items in the same list. Likewise, the user preference for a list is affected by the rest of lists interacted with by the user. Guided by these observations, we propose our hierarchical self-attentive recommendation model (AttList), as shown in Figure 2, which is built around three research questions:

- **RQ1:** How can we hierarchically model the user preference from items to lists and finally to users?
- **RQ2:** How can we incorporate the consistency of neighboring items and lists to better model user preferences?
- **RQ3:** Finally, how can we recommend user-generated item lists given the hierarchical preference model?

4.1 RQ1: Hierarchical User Preference Model

The goal of the hierarchical user preference model is to learn the latent representation for a user – $\mathbf{x}_u$ – and the latent representation for a list – $\mathbf{y}_l$. Concretely, we propose an item-level aggregation layer to collect user preference on items to represent the list and a list-level aggregation layer to collect user preference on lists to model the user.

Input Layer. The number of items contained in a list varies and we let M denote the maximum number of items that our model can handle. If a list length is greater than M, we only keep the earliest M items curated in the list. If the number of items is less than M, a "padding" item is repeatedly added to the list until the length is M. Likewise, the number of lists interacted with a user varies and we let N denote the maximum number of lists and apply the same padding strategy as items. Thus, the input for the user in a user-list pair is an item ID matrix $\mathbf{U} \in \mathbb{N}^{N\times M}$, where each element in $\mathbf{U}$ is an item ID. And the input for the list in the user-list pair is an item ID vector $\mathbf{L} \in \mathbb{N}^{M}$. Note that M and N are tuned on the validation set and will be discussed in Section 6.3.

Positional Item Representation Layer. We first create a learnable item representation matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{T}|\times d}$ where d is the latent dimensionality. Based on $\mathbf{E}$, the i-th item in l can retrieve its representation $\mathbf{e}_{li} \in \mathbb{R}^{d}$. For the padding item, a constant zero vector $\mathbf{0}$ is used. Following [23], we also take the position of items in a list into consideration because users usually see the top items in a list first and may stop exploring the item list after enough items have been consumed. Thus, we also create a learnable position representation matrix $\mathbf{O} \in \mathbb{R}^{M\times d}$ (M is the length of the list) and $\mathbf{o}_{li}$ denotes the position representation for $\mathbf{e}_{li}$. We further add them together as the positional item representation:

$$\mathbf{z}_{li} = \mathbf{e}_{li} + \mathbf{o}_{li} \tag{1}$$

The effect of the position representation will be empirically studied in our experiments.

After the item representations are obtained, the next question is how can we aggregate item representations to model the latent representation of the list containing the items:

Item-Level Self-Attentive Aggregation Layer. This layer is composed of two attention networks: a self-attention network [34] to take consistency of neighboring items to improve the item representations and a vanilla attention network to aggregate the item representations into a list representation space.

We first use a self-attention network to refine the item representations by incorporating the consistency of their neighboring items within a list:

$$\mathbf{z}'_{li} = SelfAttention(\mathbf{z}_{li})$$

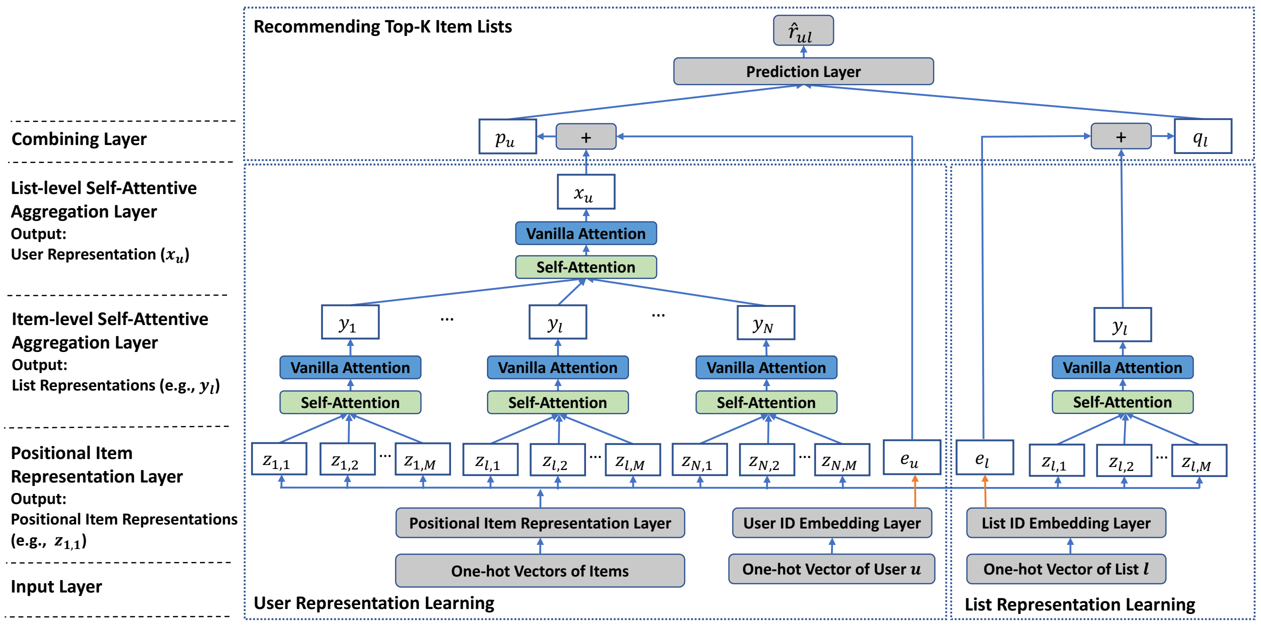

Recommending Top-K Item Lists
$\hat{r}_{ul}$
Prediction Layer
Combining Layer
p_u
+
q_l
List-level Self-Attentive Aggregation Layer
Output:
User Representation (x_u)
x_u
Vanilla Attention
Self-Attention
Item-level Self-Attentive Aggregation Layer
Output:
List Representations (e.g., y_l)
y_1
y_l
y_N
Positional Item Representation Layer
Output:
Positional Item Representations (e.g., $z_{1,1}$)
$z_{1,1}$ $z_{1,2}$ $z_{1,M}$ $z_{l,1}$ $z_{l,2}$ $z_{l,M}$ $z_{N,1}$ $z_{N,2}$ $z_{N,M}$
e_u
e_l
Input Layer
Positional Item Representation Layer
User ID Embedding Layer
List ID Embedding Layer
One-hot Vectors of Items
One-hot Vector of User u
One-hot Vector of List l
User Representation Learning
List Representation Learning

Figure 2: Framework of AttList for recommending user-generated item lists. The model is composed of three components: prediction, user representation learning, and list representation learning. Specifically, the user representation learning component is a stack of four layers: input layer, item positional representation layer, item-level self-attentive aggregation layer, and list-level self-attentive aggregation layer. The list representation learning component also contains the same layers as the user part except for a list-level self-attentive aggregation layer.

where $\mathbf{z}'_{li} \in \mathbb{R}^d$ denotes the representation of the i-th item in list l, refined by our self-attention network, which will be introduced in detail in Section 4.2.

Then, we apply the weighted sum to aggregate the refined item representations for constructing the list latent representation. The higher the weight is, the more informative the item is to reveal user preference:

$$\mathbf{y}_l = \sum_{i \in \mathcal{M}} \alpha_i \cdot \mathbf{z}'_{li} \tag{2}$$

where $\mathbf{y}_l \in \mathbb{R}^d$ is the aggregated list representation. And α_i is the weight learned by the vanilla attention network:

$$\begin{aligned} \alpha_i &= \mathbf{u}_I^T tanh(\mathbf{W}_I \mathbf{z}'_{li} + \mathbf{b}_I), \\ \alpha_i &= \frac{exp(\alpha_i)}{\sum_{n \in \mathcal{M}} exp(\alpha_n)}, \end{aligned} \tag{3}$$

where α_i is a scalar as the weight, $\mathcal{M} = \{n \in \mathbb{N} | 1 \leq n \leq M\}$ and M is the length of the list, $\mathbf{W}_I \in \mathbb{R}^{d \times d}$ is the weight matrix, $\mathbf{b}_I \in \mathbb{R}^d$ is the bias vector, $\mathbf{u}_I \in \mathbb{R}^d$ is the global vector and tanh is used as activation function empirically. The softmax function is used to normalize the weight. Note that the consistency information of the item has been incorporated in $\mathbf{z}_{li}$ by the self-attention network, thus α_i is learned by considering not only the item's intrinsic properties but also its consistency with neighboring items.

Based on the aggregated list representations, we further aggregate them to model the latent representation of the user who has interacted with the lists:

List-Level Self-Attentive Aggregation Layer. Which is similar to the structure of the item-level aggregation layer, this list-level layer is composed of two attention networks. The first is a self-attention network to improve the list representations by considering the consistency of a user's interaction profile. The second is a vanilla attention network to aggregate the list representations into a user representation space.

We first refine the list representations by injecting consistency information of their neighboring lists:

$$\mathbf{y}'_l = SelfAttention(\mathbf{y}_l)$$

where $\mathbf{y}'_l \in \mathbb{R}^d$ denotes the representation of list l, refined by our self-attention network, detailed in Section 4.2.

Then, we select informative lists and aggregate list representations into the user representation space:

$$\mathbf{x}_u = \sum_{l \in \mathcal{R}_u^+} \beta_l \cdot \mathbf{y}_l \tag{4}$$

where $\mathbf{x}_u \in \mathbb{R}^d$ is the aggregated user representation. And β_l is the weight learned by the vanilla attention network, which can be interpreted as the contribution of list l to reveal the user preference:

$$\begin{aligned} \beta_l &= \mathbf{u}_L^T tanh(\mathbf{W}_L \mathbf{y}'_l + \mathbf{b}_L), \\ \beta_l &= \frac{exp(\beta_l)}{\sum_{l \in \mathcal{R}_u^+} exp(\beta_l)}, \end{aligned} \tag{5}$$

where β_l is a scalar as the weight. $\mathbf{u}_L \in \mathbb{R}^d$, $\mathbf{W}_L \in \mathbb{R}^{d \times d}$, $\mathbf{b}_L \in \mathbb{R}^d$ are the global vector, weight matrix and bias vector respectively.

Note that the consistency information of the list has been incorporated in $\mathbf{y}_l$ by the self-attention network, thus β_l is learned by considering not only the list's intrinsic properties but also its consistency with neighboring lists.

In this section, we have proposed a hierarchical user preference model based on the first motivating observation, and next we need to carefully take care of the list and item consistency to better model the user preference based on the second motivating observation.

4.2 RQ2: Capturing Item and List Consistency by Self-Attentive Networks

In this section, a self-attention network is carefully designed to consider the consistency of neighboring items and lists to better model the user preference. Note this self-attention network works in the same way for the item-level aggregation layer and the list-level aggregation layer. For simplicity, we introduce the details for the item-level layer only (the list-level is similarly defined).

Item consistency is first measured by the similarities between each item and its neighboring items (in Equation 6) and then preserved in the refined item representations (in Equation 7) and finally returned to the vanilla attention network for assigning weights to the items (in Section 4.1). Therefore, not only the item's intrinsic properties but also its "contextual" items can be considered in the vanilla attention network (in Section 4.1) to assign weight to the item. Particularly, an item representation will be less injected into the list representation if it is inconsistent with the rest of items in the list.

Specifically, we first pack all item representations from the same list, denoted as $\{\mathbf{z}_{li} | i \in \mathbb{N}, 1 \le i \le M\}$, together into as a matrix $\mathbf{Z}_l \in \mathbb{R}^{M \times d}$. Following [34], scaled dot-product attention is also used to calculate the self-attention score. We first have:

$$\mathbf{F}_l = softmax(\frac{\mathbf{Z}_l \mathbf{Z}_l^T}{\sqrt{d}}) \tag{6}$$

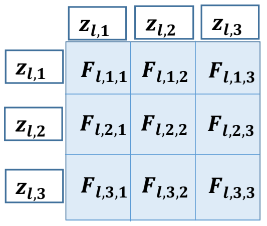
where $\mathbf{F}_l \in \mathbb{R}^{M \times M}$ is a self-attention score matrix which indicates the similarities among M items in a list. Since d is usually a large number (e.g. 128), $\sqrt{d}$ is used to scale the similarity (attention score) for preventing gradient vanishing and the softmax function is to normalize the self-attention scores. An example of Equation 6 is illustrated in Figure 3(a). Then, the output of this network is obtained by multiplying the self-attention score matrix $\mathbf{F}_l$ with the original item representation matrix $\mathbf{Z}_l$:

$$\mathbf{Z}'_l = \mathbf{F}_l \mathbf{Z}_l \tag{7}$$

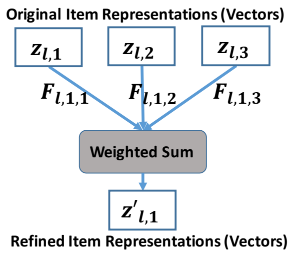
where $\mathbf{Z}'_l \in \mathbb{R}^{M \times d}$ is a set of updated item representations where each representation is a weighted sum of other ones and the weights are the self-attention scores. In this way, the item representations can be refined by incorporating information of its neighboring items, which is illustrated in Figure 3(b). Moreover, to stabilize the training as [15, 34], we use residual connections [9] to obtain the final output:

$$SelfAttention(Z_l) = \mathbf{Z}_l + \mathbf{Z}'_l \tag{8}$$

where the original item representation is element-wisely added to the refined item representation. The residual connections benefit the model training by propagating useful low-level features



(a) Self-attention scores F_l reflects the similarities among items in list l. (Equation 6)



(b) Refine item representation by using weighted-sum of original representations, where weights are the self-attention scores. (Equation 7)

Figure 3: Refining item representations ($\mathbf{z}_{l1}$) by considering the consistency with neighboring items (i.e., $\mathbf{z}_{l2}$ and $\mathbf{z}_{l3}$).

to higher layers. The effect of the residual connections will be discussed in our experiments. Besides, previous work [15, 34] has shown how to first convert the input representation matrix into three matrices through three linear projections and then feed them to the scaled dot-product attention. Though this way leads to a more expressive model, it increases the training difficulty and harms the performance in our experiments. So far, we have improved the user preference model by considering the item and list consistency, and next the model (i.e., x_u, y_l) is utilized to recommend item lists.

4.3 RQ3: Recommending Top-K Item Lists

Finally, we are ready to recommend the top-k item lists. We first combine the aggregated representations with the ID embedding of items and lists to capture general user preferences and then design a prediction layer to calculate the preference score for recommending item lists.

Combining Layer. In this layer, we element-wisely add the aggregated user representation ($\mathbf{x}_u$) with a dedicated ID embedding of the user ($\mathbf{e}_u \in \mathbb{R}^d$):

$$\mathbf{p}_u = \mathbf{x}_u + \mathbf{e}_u \tag{9}$$

where $\mathbf{p}_u \in \mathbb{R}^d$ is the final representation of user u. The intention is to consider the general user preference (modeled by $\mathbf{e}_u$) that in some cases cannot be fully characterized by the aggregation of the list representations, which has been proven beneficial for recommendation in [2, 4, 5]. Similarly, we have:

$$\mathbf{q}_l = \mathbf{y}_l + \mathbf{e}_l \tag{10}$$

where $\mathbf{y}_l \in \mathbb{R}^d$ is the aggregated representation of l of list l, $\mathbf{q}_l \in \mathbb{R}^d$ is the final representation and a dedicated ID embedding of the list $\mathbf{e}_l \in \mathbb{R}^d$ is also learned to model the general characteristic of the list, which cannot be fully expressed by the aggregation of the item representations.

Prediction Layer. In this layer, the estimated preference score $\hat{r}_{ul}$ is calculated given the user latent representation $\mathbf{p}_u$ and the list latent representation $\mathbf{q}_l$. Following [2, 10], we also first apply the element-wise product on $\mathbf{p}_u$ and $\mathbf{q}_l$, and then concatenate it with the original representations:

$$\mathbf{h}_0 = \begin{bmatrix} \mathbf{p}_u \odot \mathbf{q}_l \\ \mathbf{q}_l \\ \mathbf{p}_u \end{bmatrix} \tag{11}$$

where the element-wise product $\mathbf{p}_u \odot \mathbf{q}_l$ captures the interactive relationship between the user and the list which follows traditional latent factor models. The concatenation of the original representations $\mathbf{p}_u$ and $\mathbf{q}_l$ is to prevent information loss due to the element-wise product which has been proven effective for recommendation in [2, 10].

After that, we apply a two-layer neural network to obtain the final preference score:

$$\begin{cases} \mathbf{h}_1 = ReLU(\mathbf{W}_1\mathbf{h}_0 + \mathbf{b}_1), \\ \hat{r}_{ul} = Sigmoid(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2) \end{cases} \tag{12}$$

where $\mathbf{h}_1 \in \mathbb{R}^D$ is the output of the hidden layer, $\mathbf{W}_1 \in \mathbb{R}^{D\times 3d}$ and $\mathbf{W}_2 \in \mathbb{R}^{1\times D}$ denote the weight matrix, $\mathbf{b}_1 \in \mathbb{R}^D$ and $\mathbf{b}_2 \in \mathbb{R}^D$ denote the bias vector, in which D as the predictive factors [10] controls the model capability. Empirically, ReLU function is used as the non-linear activation function for the hidden layer. For each user, all lists are ranked by their corresponding preference scores and the top-k lists are recommended to the user.

Objective Function. Following [10], we also treat top-K recommendation with implicit feedback as a binary classification problem and apply the binary cross-entropy loss as the loss function:

$$\ell = -\sum_{u\in\mathcal{U}} \sum_{\mathcal{R}_u^+\cup\mathcal{R}_u^-} r_{ul}\, log\, \hat{r}_{ul} + (1 - r_{ul})\, log\, (1 - \hat{r}_{ul}) \tag{13}$$

where $\mathcal{R}_u^+$ denotes the set of lists interacted with u and $\mathcal{R}_u^-$ is the set of negative samples which are uniformly sampled from unobserved interactions ($r_{ul} = 0$). Note that we generate a new batch of negative samples in each iteration and the amount of the negative samples can be controlled by a hyper-parameter ratio ρ, i.e., $|\mathcal{R}_u^-| = \rho|\mathcal{R}_u^+|$.

5 EXPERIMENTAL SETUP

In this section, we first introduce three list-based datasets. After that, baseline methods, details for reproducibility and evaluation metrics are also provided.

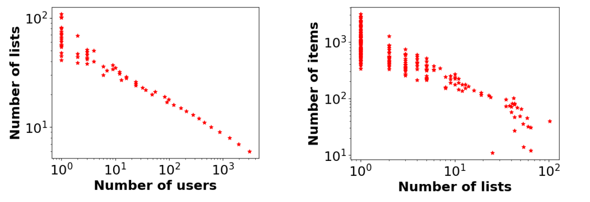
5.1 Datasets: Goodreads, Spotify, and Zhihu

In this section, we describe our three list-based datasets, which are are drawn from three popular platforms with different kinds of lists (book-based, song-based, and answer-based), reflecting a variety of scenarios to evaluate AttList and alternative approaches. Table 1 summarizes these three datasets.

Goodreads. Goodreads is a popular site for book reviews and recommendation. Users in Goodreads can browse all the books within a book list and vote for the list to express their preference. We randomly sample 18,435 users from all the users and then crawl all lists voted by them as well as the books within the lists, resulting in 24,217 lists containing 158,392 books (items).

Spotify. Spotify is a music streaming service where users can follow playlists created by other users. We first search for playlists by issuing 200 keyword queries representing popular topics on Spotify, like "pop" and "drive". Then, followers of the playlists are further crawled, arriving at 7,787 lists containing 49,434 songs (items).

Zhihu. Zhihu is a question and answer community where users can follow lists of answers that have been curated by other users.



(a) User-list Interaction Relationship (b) List-item Containing Relationship

Figure 4: Power law distribution between users & lists and lists & items in Spotify dataset.

For example, a list may contain two answers as the response to two questions correspondingly: "What is the best way to read a book?" and "How to dress for an interview?". We first randomly sample users from the answerers of three popular topics: life, learning and recreation. Then the lists followed by the users are also crawled, resulting in 12,715 lists containing 211,242 answers (items).

Following the preprocessing setup in [3, 23], we filter out items appearing in fewer than 5 lists. To study the impact of data density on recommendation performance, we also filter out users who have interacted with lists fewer than 5 times on Spotify and Zhihu, so that they are denser than the Goodreads dataset. Even with such filtering, note that the three datasets are all very sparse with density of 0.056%, 0.115% and 0.178% respectively, which reflects the data sparsity challenge in real-world list recommendation.

In Figure 4(a), we group users into buckets according to how many lists they interacted with and show the number of lists that belong to the buckets correspondingly. We find that most users only interact with a few lists while a few users interact with a large number of lists. A similar observation can be seen in Figure 4(b), where most lists contain a few items while a few lists contain a large number of items. Thus, we observe a power law distribution between the number of lists a user interacts and the number of items a list contains.

For our experiments, we randomly split each dataset into three parts: 80% for training, 10% for validation and 10% for testing. Since many users have very few interactions, they may not appear in the testing set at all. In this case, there will be no match between the top-K recommendations of any algorithm and the ground truth since there are no further interactions made by these users. The results reported here include these users, reflecting the challenges of list recommendation under extreme sparsity.

Table 1: Summary Statistics for the Evaluation Datasets

Dataset	#Users	#Lists	#Interactions	Density	#Unique Items
Goodreads	18,435	24,217	250,450	0.056%	158,392
Spotify	10,183	7,787	91,254	0.115%	49,434
Zhihu	6,101	12,715	138,458	0.178%	211,242

5.2 Baselines

In this section, we introduce a suite of baselines. The first set of methods are item-based top-k recommendation methods that we adapt to the problem of list recommendation:

ItemPop. This simple method recommends the most popular item lists. Lists are ranked by the number of times they have been interacted with (e.g., followed or liked). The same top-K most popular lists are recommended to all users.

MF. This is the traditional matrix factorization (MF) method proposed in [17] using mean squared error as the objective function with negative sampling from the non-observed lists ($r_{ul} = 0$).

BPR. Bayesian personalized ranking (BPR) [28] is a well-known pair-wise ranking framework for implicit recommendation. BPR assumes that, for each user, the preference of the observed list ($r_{ul} = 1$) is superior to the non-observed ones ($r_{ul} = 0$).

NCF. Neural collaborative filtering (NCF) [10] is a state-of-the-art neural network method for recommendation. NCF concatenates latent factors learned from a generalized matrix factorization model and a multi-layered perceptron model and then uses a regression layer to predict the preferences of a user to the lists. Following [10], negative samples are uniformly sampled from unobserved interactions ($r_{ul} = 0$).

The next set of methods is designed specifically for list recommendation. They use user-list interactions as well as the containing relationship between lists and items.

LIRE. List Recommendation Model (LIRE) [23] is a Bayesian-based pair-wise ranking approach. It models user preferences by the sum of two factors. The first factor is the inner product between user latent factors and list latent factors; the second factor is the sum of inner products between the user factor and item latent factors within the list.

EFM. Embedding Factorization Model (EFM) [3] is also a Bayesian-based pair-wise model designed for list recommendation. Inspired by paragraph2vec model in [18], it calculates the shifted positive pointwise mutual information (SPPMI) value between lists and items and uses this information to boost the ranking performance.

5.3 Reproducibility

All code and data are available at here[1]. We implement ItemPop, BPR, and MF. The implementation of NCF is from the authors.[2] For LIRE and EFM, we use code from the authors of EFM.[3] We implement AttList with Keras, and Adam [16] is applied as the optimizer. For AttList and baseline methods, all hyper-parameters are tuned on the validation dataset, where early stopping strategy is applied such that we terminate training if validation performance does not improve over 10 iterations. All neural network models were trained using Nvidia GeForce GTX Titan X GPU with 12 GB memory and 3,072 cores.

Eliminating Lists from the Testing Dataset for User Representation. As discussed in Section 4.1, AttList represents a user by the aggregation of list representations. In the training, it is important to eliminate the lists which are sampled into the testing dataset to represent the user, which makes the training and testing datasets mutually exclusive.

[1]https://github.com/heyunh2015/AttList

[2]https://github.com/hexiangnan/neural_collaborative_filtering

[3]https://listrec.wixsite.com/efms

Over-fitting Prevention. Dropout [31] is widely used in neural networks training to improve generalization performance. We also apply dropout in every layer of the model. Specifically, we randomly drop γ percentage of the output vectors: $\mathbf{e}_{li}$, $\mathbf{o}_{li}$, $\mathbf{y}_l$, $\mathbf{x}_u$ and $\mathbf{h}_1$. Following [34], we also drop γ percent of the output of the softmax function in the self-attention network. We also apply L_2 regularization to the weight matrix in the vanilla attention network (e.g., $\mathbf{W}_I$ and $\mathbf{W}_L$), where λ denotes the parameter controlling the regularization strength.

Parameter Settings. The batch size is tested from [16, 32, 64, 128, 256] and 32 is selected for all three datasets according to the results on the validation dataset. The learning rate is tested from [0.00005, 0.0001, 0.0005, 0.001, 0.005] and 0.0001 is selected for Spotify and Zhihu while 0.001 is better for Goodreads. The candidates for the latent dimensionality d is from [8, 16, 32, 64, 96] and we select 64 for Zhihu and 96 for the other two datasets. The maximum number of lists N and the maximum number of items M are selected from [5, 10, 15, 20] and [8, 16, 32, 64] respectively. The validation results show that $N = 15$ and $M = 32$ are better for performance. The ratio for negative sampling ρ is tested from [3, 5, 7, 9, 11] and 3, 5, 7 are selected for Goodreads, Spotify and Zhihu respectively. The predictive factor D is tested from [20, 50, 100] and set as 100 for all datasets. In our model, many components use dropout and L2 regularization. For simplicity, we introduce the range for their parameters tuning: the γ percentage for dropout is tested from [0, 0.3, 0.5, 0.8] and the L2 regularization strength λ is selected from [0.001, 0.01, 0.1].

5.4 Evaluation Metrics

Given a user, a top-K item list recommendation algorithm provides a list of ranked item lists according to the predicted preference of them. To assess the ranked lists with respect to the ground-truth lists set of what users actually interacted with, we adopt three evaluation metrics: Normalized Discounted Cumulative Gain (NDCG) [14] at 5 and 10 (N@5 and N@10), precision at 5 and 10 (P@5 and P@10), and recall at 5 and 10 (R@5 and R@10).

6 EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we present our experimental results and discussion toward answering the following experimental research questions (RQs):

- **RQ4:** How well does AttList perform compared to traditional top-k recommenders? And more importantly, how well does AttList perform compared to models that are specifically designed for list recommendation (LIRE and EFM)?
- **RQ5:** What is the impact of the design choices of AttList on the quality of list recommendation? Is attention important?
- **RQ6:** What is the impact of the hyper-parameters of AttList on the quality of list recommendation?
- **RQ7:** Finally, do the learned self-attention scores capture meaningful patterns (e.g., consistency) of items and lists?

6.1 RQ4: List Recommendation Quality

Tables 2, 3 and 4 report the experimental results on three datasets, where '**' indicates that the improvements over all baselines pass the t-test significance test with p-value < 0.01 and particularly $\triangle_{EFM}$ and $\triangle_{LIRE}$ denote the improvement upon EFM and LIRE. We observe that our model significantly outperforms all baselines on all metrics. For instance, AttList achieves the best NDCG@10 of 3.691%, 8.953% and 3.666% on the three datasets, with an improvement upon LIRE of 10.4%, 5.9% and 6.8%. Significant improvements in terms of precision and recall are also observed.

We also observe that the list recommending methods (AttList, LIRE, EFM) are consistently superior to the traditional top-k recommendation methods (ItemPop, BPR, MF, NCF). The possible reason is that the list recommending methods learn the user preference not only from user-list interactions but also from the information of items within the lists. This demonstrates that the top-k item list recommendation is quite different from the general top-k items recommendation and requires considering the containing relationship between lists and items. Besides, all three list recommending methods achieve a better performance in Spotify and Zhihu than in Goodreads, which shows that the denser the dataset is, the higher performance can be obtained by the list recommending methods.

Table 2: Experimental Results on Goodreads Dataset

Metric(%)	P@5	R@5	N@5	P@10	R@10	N@10
ItemPop	0.692	1.832	1.487	0.538	2.620	1.760
MF	0.883	2.128	1.708	0.765	3.417	2.165
BPR	1.004	2.270	2.043	0.852	3.801	2.597
NCF	1.086	2.668	2.192	0.900	4.167	2.694
EFM	1.311	3.347	2.587	1.045	5.027	3.214
LIRE	1.337	3.271	2.731	1.068	4.965	3.345
AttList	1.449**	3.580**	3.048**	1.148**	5.465**	3.691**
$\triangle_{EFM}$	10.6%	6.9%	17.8%	9.9%	8.7%	14.8%
$\triangle_{LIRE}$	8.4%	9.54%	11.6%	7.6%	10.1%	10.4%

Table 3: Experimental Results on Spotify Dataset

Metric(%)	P@5	R@5	N@5	P@10	R@10	N@10
ItemPop	0.621	2.113	1.454	0.532	3.533	1.961
MF	2.396	7.884	6.005	1.609	10.385	6.966
BPR	2.555	8.541	6.183	1.775	11.673	7.389
NCF	2.602	8.793	6.643	1.820	11.914	7.832
EFM	2.744	9.085	6.934	1.989	13.079	8.305
LIRE	2.850	9.468	7.065	2.020	13.277	8.453
AttList	2.944**	9.887**	7.520**	2.110**	13.935**	8.953**
$\triangle_{EFM}$	7.3%	8.8%	8.5%	6.1%	6.6%	7.8%
$\triangle_{LIRE}$	3.3%	4.4%	6.4%	4.5%	5.0%	5.9%

Table 4: Experimental Results on Zhihu Dataset

Metric(%)	P@5	R@5	N@5	P@10	R@10	N@10
ItemPop	1.138	2.275	1.860	0.959	3.951	2.419
MF	1.341	2.706	2.163	1.059	4.052	2.629
BPR	1.347	2.743	2.381	1.162	4.545	3.027
NCF	1.485	2.834	2.435	1.216	4.674	3.057
EFM	1.508	3.267	2.718	1.262	5.341	3.358
LIRE	1.596	3.319	2.853	1.303	5.350	3.433
AttList	1.754**	3.600**	2.986**	1.393**	5.646**	3.666**
$\triangle_{EFM}$	16.3%	10.2%	9.9%	10.4%	5.7%	9.2%
$\triangle_{LIRE}$	9.9%	8.5%	4.7%	6.9%	5.5%	6.8%

6.2 RQ5: Ablation Analysis

Since AttList is composed of several important design decisions – like the inclusion of self-attention – we next present an ablation analysis to study the impacts of these decisions on recommendation quality. In Table 5, we present the results of AttList versus several variants as described next:

– **Vanilla Attention:** In this first experiment, all the vanilla attention networks in the architecture are replaced with a simple average pooling method. We observe that performance becomes worse in all three datasets with a large drop, for example, 16.8% in P@10 for Spotify. This shows that different items and lists reveal fundamentally different evidence of user preferences. This demonstrates that AttList is able to select informative items and lists to characterize the list and the user respectively.

– **Self-Attention:** In this experiment, all the self-attention networks are removed. Significant drops in all metrics are observed across all three datasets, especially in Zhihu with a drop of 7.3% in P@10. These results show that our self-attention networks can boost performance by considering consistency of neighboring items and lists to refine the item and list representations.

– **Attention Mechanism:** We next not only remove the self attention networks (*– Self-Attention*) but also replace the vanilla attention networks with the average pooling method (*– Vanilla Attention*). In other words, the arithmetic mean of item representations is used to model the list representation and list representations are just averaged to represent the user in this case. Unsurprisingly, by entirely removing the attention mechanism, we see a larger drop than in either of the previous two isolated experiments.

– **Residual Connections:** The impacts of removing residual connections in the self-attention networks is quite easily noticed, leading to a large performance drop on all datasets, proving that residual connections are very useful to stabilize the neural network training.

– **Position Information:** The experimental results show that the positional representation has a minor impact on our architecture where a drop of 1.4%, 0.1% and 2.0% is observed on the three datasets respectively. A possible reason is that the maximum number of items that our model can handle is set as 32 according to the hyper-parameters tuning in the validation set. Hence, the influence of the top 32 items does not vary too much in terms of their positions.

– **User and List ID embeddings:** We observe that incorporating the user and list ID embeddings into the final user and list representation is able to boost the performance, especially in Goodreads. This shows that our model captures some general characteristics of the user and list by the ID representation which benefits top-k list recommendation.

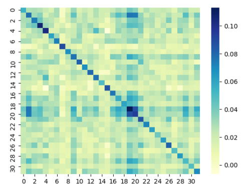
+ **Linear Projections:** Previous works [15, 34] apply three projection matrices to transform the input of self-attention layer into three matrices Q, K and V. Then, the scaled dot-product attention can be calculated as:

$$SelfAttention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d}})V$$

The projections make the model more flexible. However, we find that the projections make the network training more difficult and degrades the performance.

Table 5: Ablation analysis on the three datasets for precision and recall. Qualitatively similar results hold for NDCG as well.

	Goodreads				Spotify				Zhihu			
Architecture	P@10	Change	R@10	Change	P@10	Change	R@10	Change	P@10	Change	R@10	Change
AttList	**1.148**		**5.465**		**2.110**		**13.935**		**1.393**		5.646	
- Vanilla Attention	1.075	-6.4%	4.917	-10.0%	1.756	-16.8%	11.477	-17.6%	1.315	-5.6%	5.347	-5.3%
- Self-Attention	1.111	-3.2%	5.355	-2.0%	1.972	-6.6%	12.988	-6.8%	1.292	-7.3%	5.117	-9.4%
- Attention Mechanism	0.994	-13.4%	4.728	-13.5%	1.741	-17.5%	11.429	-18.0%	1.252	-10.1%	5.009	-11.3%
- Residual Connections	1.020	-11.2%	4.752	-13.1%	1.959	-7.2%	12.999	-6.7%	1.254	-10.0%	4.939	-12.5%
- Position Information	1.132	-1.4%	5.456	-0.2%	2.108	-0.1%	13.904	-0.2%	1.365	-2.0%	**5.668**	+0.4%
- ID Embedding	1.044	-9.1%	5.114	-6.4%	2.093	-0.8%	13.797	-1.0%	1.336	-4.1%	5.571	-1.3%
+ Linear Projections	1.115	-2.9%	5.360	-1.9%	2.027	-4.0%	13.325	-4.4%	1.306	-6.2%	5.213	-7.7%


(a) A Spotify playlist with high consistency among music tracks.

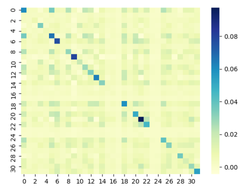

(b) A Spotify playlist with low consistency among music tracks.

Figure 5: Heatmap of self-attention scores of two lists.

In summary, the vanilla attention network, the self-attention network with residual connections and the user and list ID representation are quite important for the architecture. Inspired by previous work, we also try to use position information [23] and linear projections [34]. However, positional information of items has a limited impact while the length of a list is short (e.g., 32). Besides, self-attention networks should be carefully designed, where introducing linear projections increases a model's expressive capability but also makes the training more difficult.

6.3 RQ6: Impact of Hyper-parameters

Due to the limited space, we focus here on four representative hyper-parameters of AttList to discuss their impact on performance. In Figure 6(a), we observe that AttList benefits from larger numbers of the latent dimensionality d. The best results are obtained with d = 96 in Goodreads and Zhihu. In Figure 6(b), we surprisingly observe that a larger number ($M > 32$) of items harms the performance in Spotify and Zhihu. Similarly, we find that a larger number ($N > 15$) of lists harms the performance, which is omitted here for simplicity. Figure 6(c) shows that too many negative samples ($\rho > 5$) harms the performance in Goodreads and Spotify. Figure 6(d) shows that the learning rate should be tuned between 0.0001 and 0.0005 to obtain the best performance.

6.4 RQ7: Visualizing Self-Attention Scores

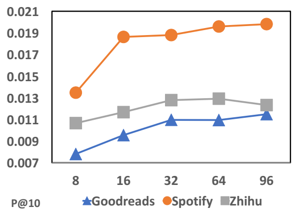
In this section, we visualize the self-attention scores and study if they can reflect consistency of neighboring items and lists as we claimed. As discussed in Section 4.2, the self-attention score matrix in item-level aggregation layer is $F_l \in \mathbb{R}^{M \times M}$, where each element is the similarity between two items in the list. We pick two typical lists from Spotify and plot the heatmap of self-attention scores of songs within the two lists as shown in Figure 5, where the darker the color is, the higher the self-attention score is. Two observations can be obtained from Figure 5. First, the diagonal elements in the two matrices have higher self-attention scores. This is understandable that each item is similar to itself, which reaffirms that the self-attention scores are actually the similarities between one item and its neighboring items. Second, as a whole, the heatmap in Figure 5(a) is darker than the heatmap in Figure 5(b). This illustrates that the list in 5(a) has a higher internal consistency than the list in 5(b). A possible reason for that is the first list curates songs more homogeneous (e.g., similar genre) than the second list. This case shows that our self-attention network is able to capture consistency among items within a list.

To further investigate the self-attention scores, we present here a case study for a well-known song *Safe and Sound* by *Taylor Swift*, which appears in the two lists in Figure 7. We show the self-attention scores of it in the high internal consistence list in Figure 7(a) and the scores in the low internal consistence list in Figure 7(b). We observe that *Safe and Sound* has higher self-attention scores with other songs in 7(a) than in 7(b), which is consistent with the observation from Figure 5. Taking a song *Shake It Off* in Figure 7(a) as an example, this song has a relatively high self-attention score (0.045, the maximum score is 0.072 in the list) to *Safe and Sound*. Interestingly, the two songs are both performed by *Taylor Swift*. This illustrates that the semantic relationship among songs can be captured by self-attention scores.

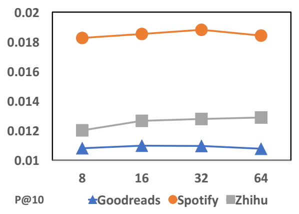
7 CONCLUSIONS AND FUTURE WORK

In this paper, we tackle the problem of recommending user generated item lists through a new hierarchical self-attentive recommendation model. Unlike traditional collaborative filtering based algorithms that are optimized for user-list interactions, our proposed model leverages the hierarchical structure of items, lists, and users to capture the containment relationship between lists and items, revealing additional insights into user preferences. With this hierarchical structure among items, lists, and users, our model first aggregates items to characterize the lists they belong to, and then aggregates these lists to estimate user preferences. A key aspect of the proposed approach is a novel self-attention network to refine the item representations and list representations by considering consistency of neighboring items and lists. Experiments over three real-world domains – Goodreads, Spotify, and Zhihu – demonstrate the effectiveness of AttList versus state-of-the-art item-based and list-based recommenders. Furthermore, these datasets and code will be released to the research community for further exploration.

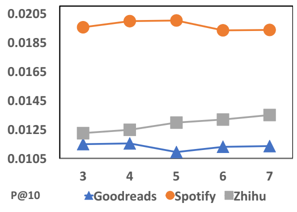
In our continuing work, we plan to improve AttList in the following two directions: 1) We plan to use list titles (e.g., "my favorite adventure books") for improving our model, which also contain rich signals of user preference. 2) The social community around a



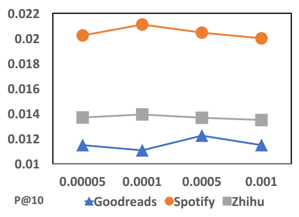
(a) Latent dimensionality d



(b) Number of items M

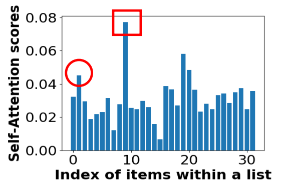


(c) Number of negatives ρ

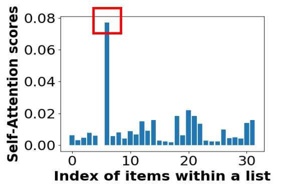


(d) Learning rate in Adam

Figure 6: Impact of the hyper-parameters of AttList.



(a) Self-attention scores for the high consistency Spotify playlist.



(b) Self-attention scores for the low consistency Spotify playlist.

Figure 7: The self-attention scores of Taylor Swift's *Safe and Sound* on the two lists also featured in Figure 5. The square marks *Safe and Sound* and the circle marks *Shake It Off*.

list also has a considerable impact on the behavior of interactions between users and lists. For example, users are more likely to interact with lists that their friends have interacted with. Thus, the social influence can be modeled to improve the performance.

ACKNOWLEDGMENTS

This work is supported in part by NSF (#IIS-1841138).

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[2] Da Cao, Xiangnan He, Lianhai Miao, Yahui An, Chao Yang, and Richang Hong. 2018. Attentive group recommendation. In *SIGIR*.

[3] Da Cao, Liqiang Nie, Xiangnan He, Xiaochi Wei, Shunzhi Zhu, and Tat-Seng Chua. 2017. Embedding factorization models for jointly recommending items and user generated lists. In *SIGIR*.

[4] Chong Chen, Min Zhang, Yiqun Liu, and Shaoping Ma. 2018. Neural attentional rating regression with review-level explanations. In *WWW*.

[5] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. 2017. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In *SIGIR*.

[6] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Recsys*.

[7] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. 2018. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *WWW*.

[8] Derek Greene, Fergal Reid, Gavin Sheridan, and Padraig Cunningham. 2011. Supporting the curation of twitter user lists. *arXiv preprint arXiv:1110.1349* (2011).

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*.

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

[11] Yun He, Haochen Chen, Ziwei Zhu, and James Caverlee. 2018. Pseudo-Implicit Feedback for Alleviating Data Sparsity in Top-K Recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1025–1030.

[12] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *ICDM*.

[13] Xiaowen Huang, Shengsheng Qian, Quan Fang, Jitao Sang, and Changsheng Xu. 2018. CSAN: Contextual Self-Attention Network for User Sequential Recommendation. In *MM*.

[14] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *TOIS* (2002).

[15] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. In *ICDM*.

[16] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[17] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. (2009).

[18] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *ICML*.

[19] Sheng Li, Jaya Kawale, and Yun Fu. 2015. Deep collaborative filtering via marginalized denoising auto-encoder. In *CIKM*.

[20] Xiaopeng Li and James She. 2017. Collaborative variational autoencoder for recommender systems. In *KDD*.

[21] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. 2018. Variational Autoencoders for Collaborative Filtering. (2018).

[22] Yuchen Liu, Dmitry Chechik, and Junghoo Cho. 2016. Power of Human Curation in Recommendation System.. In *WWW (Companion Volume)*.

[23] Yidan Liu, Min Xie, and Laks VS Lakshmanan. 2014. Recommending user generated item lists. In *Recsys*.

[24] Caroline Lo, Justin Cheng, and Jure Leskovec. 2017. Understanding online collection growth over time: A case study of Pinterest. In *WWW (Companion Volume)*.

[25] Chen Ma, Yingxue Zhang, Qinglong Wang, and Xue Liu. 2018. Point-of-Interest Recommendation: Exploiting Self-Attentive Autoencoders with Neighbor-Aware Influence. In *CIKM*.

[26] Lei Mei, Pengjie Ren, Zhumin Chen, Liqiang Nie, Jun Ma, and Jian-Yun Nie. 2018. An attentive interaction network for context-aware recommendations. In *CIKM*.

[27] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. 2008. One-class collaborative filtering. In *ICDM*.

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

[29] Sungyong Seo, Jing Huang, Hao Yang, and Yan Liu. 2017. Interpretable Convolutional Neural Networks with Dual Local and Global Attention for Review Rating Prediction. In *Recsys*.

[30] Ajit P Singh and Geoffrey J Gordon. 2008. Relational learning via collective matrix factorization. In *KDD*.

[31] 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).

[32] Yi Tay, Anh Tuan Luu, and Siu Cheung Hui. 2018. Multi-Pointer Co-Attention Networks for Recommendation. In *KDD*.

[33] Reggie Ugwu. 2016. https://www.buzzfeed.com/reggieugwu/the-unsung-heroes-of-the-music-streaming-boom. In *https://www.buzzfeed.com/reggieugwu/*.

[34] 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 *NIPS*.

[35] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *WSDM*.

[36] Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. 2017. Attentional factorization machines: Learning the weight of feature interactions via attention networks. *arXiv preprint arXiv:1708.04617* (2017).

[37] Haochao Ying, Fuzhen Zhuang, Fuzheng Zhang, Yanchi Liu, Guandong Xu, Xing Xie, Hui Xiong, and Jian Wu. 2018. Sequential Recommender System based on Hierarchical Attention Networks. In *IJCAI*.

[38] Shuai Zhang, Yi Tay, Lina Yao, and Aixin Sun. 2018. Dynamic Intention-Aware Recommendation with Self-Attention. *arXiv preprint arXiv:1808.06414* (2018).

[39] Shuai Zhang, Lina Yao, and Aixin Sun. 2017. Deep learning based recommender system: A survey and new perspectives. *arXiv preprint arXiv:1707.07435* (2017).

[40] Chang Zhou, Jinze Bai, Junshuai Song, Xiaofei Liu, Zhengchao Zhao, Xiusi Chen, and Jun Gao. 2017. ATRank: An Attention-Based User Behavior Modeling Framework for Recommendation. *arXiv preprint arXiv:1711.06632* (2017).

[41] Meizi Zhou, Zhuoye Ding, Jiliang Tang, and Dawei Yin. 2018. Micro behaviors: A new perspective in e-commerce recommender systems. In *WSDM*.