arXiv:2207.06652v2 [cs.IR] 21 Jul 2022

Every Preference Changes Differently: Neural Multi-Interest Preference Model with Temporal Dynamics for Recommendation

Hui Shi[1], Yupeng Gu[2], Yitong Zhou[2], Bo Zhao[2], Sicun Gao[1], Jishen Zhao[1]
[1]University of California, San Diego, [2]Pinterest Inc.
[1]San Diego, CA [2]San Francisco, CA, USA
{hshi,sicung,jzhao}@ucsd.edu,{yupeng,yzhou,bozhao}@pinterest.com

ABSTRACT

User embeddings (vectorized representations of a user) are essential in recommendation systems. Numerous approaches have been proposed to construct a representation for the user in order to find similar items for retrieval tasks, and they have been proven effective in industrial recommendation systems as well. Recently people have discovered the power of using multiple embeddings to represent a user, with the hope that each embedding represents the user's interest in a certain topic. With multi-interest representation, it's important to model the user's preference over the different topics and how the preference change with time. However, existing approaches either fail to estimate the user's affinity to each interest or unreasonably assume every interest of every user fades with an equal rate with time, thus hurting the recall of candidate retrieval. In this paper, we propose the Multi-Interest Preference (MIP) model, an approach that not only produces multi-interest for users by using the user's sequential engagement more effectively but also automatically learns a set of weights to represent the preference over each embedding so that the candidates can be retrieved from each interest proportionally. Extensive experiments have been done on various industrial-scale datasets to demonstrate the effectiveness of our approach. [1]

1 INTRODUCTION

Today, the recommendation system is commonly used in technology companies to help users discover relevant items and delivers a positive user experience. In industrial recommendation systems, there are usually billions of items in the data store, which makes it impossible to calculate the similarity between a user and every one of them. Therefore, the first stage of industrial recommendation systems will be candidate generation, where only hundreds or thousands of items will be chosen as the candidates and will be sent to the later more nuanced scoring and ranking stages. These candidates will be selected based on the similarity to a user representation on an approximate level (e.g. inverted indexes, random walks on graphs) without consuming too much computational power. The ranking stage is usually more computationally costly, therefore it is imperative for candidate generators to return a limited number of items (e.g. several thousand). Given the limited budget, it is crucial to find effective user representations in order to retrieve as many relevant items as possible.

The user representations learned from the neural networks are proven to work well on large-scale online platforms, such as Google[4], YouTube [5], Alibaba [35], etc. In addition, these user embedding vectors can also be used as features in other tasks such as search retrieval [2, 15], and support collaborative filtering as well [23, 29]. User embeddings are learned mostly by aggregating the item embedding that the user has engaged with in the past. Note that in these works, item embeddings are usually assumed to be known beforehand. These embeddings can be constructed using similarity-preserving approaches or standard graph embedding methods such as node2vec [10], GraphSAGE [11] or graph convolutional neural networks [42]. The aggregation of item embeddings is commonly achieved by sequential models researchers sequence information in user engagements [13, 14, 17, 26, 43]. They usually introduce a recurrent neural network (RNN) model or an attention mechanism on the user engagement history sequence. Due to the sequential modeling strategies, they are able to capture the relative importance of recent vs. former actions.

Recently researchers [8, 21, 25, 38] have discovered the importance of having multiple embeddings for an individual, with the hope that they can capture a user's multiple interests. The intuition is quite clear: a user may have interests in multiple categories, and collapsed embedding of multiple interests may be mostly close to some other topic (as illustrated in Fig. 1). Consequently, even though the collapsed single embedding may still be similar to the user's true interests, the recommendation could recommend items that are most close to the single embedding vector but totally from a user disliked category. Furthermore, the advantage of having multiple embedding has been shown in classification and link prediction tasks [24, 38]. Pinterest has adopted this methodology in their production system, and each of the user embeddings is the embedding medoid of items that belong to that category [25]. However, conventional sequential models like RNN or the Transformer network do not naturally produce multiple sequence-level embeddings as desired in the multi-interest user representation. Existing solutions

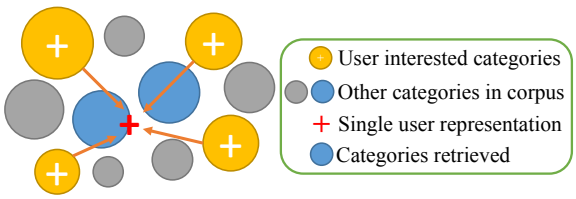

Figure 1: Mis-representation with single user embedding: let circles represent categories in the corpus, and + shows the mean of the user's multiple interests. The closest categories (blue circles) to the user representation may be irrelevant.

[1]The code is available at: https://github.com/shihui2010/MIP

include 1) clustering the items in the user engagement history and computing a representation embedding per cluster [25]; 2) taking the output of the multi-head attention on the latest user engagement and letting each attention head result be an interest embedding. The limitation of the two approaches is obvious: clustering then processing the sub-sequence can not be applied without the item feature (e.g. inapplicable to collaborative filtering dataset where only the item ID is known); and the multi-head attention models bias towards the popular categories owing to its' shared query vector among all the users. Also, both approaches are inflexible to adjust the number of interests.

Moreover, as far as we are concerned, all of the existing works are ignoring one important aspect: the weights for each embedding. It may not matter in the prediction task because the (user, item) affinity score is usually determined by the maximum similarity between item embedding and each of the user embeddings. However in the candidate generation stage, given the limited number of items returned, retrieving items from each embedding uniformly will cause a recall problem when the user clearly indicates a high affinity towards one or two categories. Some existing approaches, e.g. PinnerSage[25], use exponentially decayed weights to assign a higher score to interests that have more frequent and recent engagements. However, the methods still assume that given the same period, for each user and for any interest, the level of interest decays equally. Furthermore, these multi-embedding methods always assume the number of embeddings to be fixed across all users. Not only is this hyperparameter costly to find, but also the assumption that all users have the same number of interests is questionable. Some dormant users can be well represented using one or two vectors, while others have a far more diverse set of niche interests and it is necessary to have tens of embeddings to represent them.

In this paper, we propose Multi-Interest Preference (MIP) model that learns user embeddings on multiple interest dimensions with personalized and context-aware interest weights. Unlike existing multi-interest approaches, MIP is able to utilize the sequential information in a user's engagement history to extract personalized interest embeddings without biasing towards popular categories. In addition, MIP assigns a set of weights to each of the embedding representations, allowing the candidate generator to adjust the number of items retrieved from each interest accordingly. This fits the actual needs of the candidate generation stage in industrial recommendation systems. Furthermore, MIP does not assume each user should have the same number of interests, and the number of user interests can be adjusted after the training process of the offline model with marginal computation cost, which gives the system large flexibility to balance between storage cost and the recommendation performance. The main contribution of this paper can be summarized as follows:

- We propose a methodology that utilizes the sequential engagement history to find multiple embedding for a user, each of which represents the user's interest in a certain category. These representations will be used in the retrieval task in recommendation systems, which we find successful in various industry-scale datasets.
- In addition to the multi-facet vector representations of a user, we will assign weights to each embedding, which is automatically customized for each user interest, and improve the recall of candidate generation by retrieving more candidates from the most representative embedding.
- We do not require any prior knowledge of the number of categories per user, nor do we assume it to be the same among all users. The number of meaningful categories can be trivially modified post-training process.

2 RELATED WORK

This work relates to two important aspects of existing recommendation systems: sequential model and multi-interest framework.

Sequential models. A basic consensus in the recommendation system is that user embeddings should be inferred from the users' historical behavior, and thus the sequential models have been at the heart of recommendation models. A typical and classical sequential model is the Markov Chain [12, 27]. While Markov Chain captures short-term patterns of engagement sequence well, it fails to make the recommendation that requires memorizing long sequences. With stronger representation power on long sequences, Recurrent neural networks (RNNs), have been adopted for learning user embedding from arbitrarily long sequences [6, 7, 14, 40]. Besides the standard RNN models, specialized recurrent units are proposed to meet the special need of incorporating certain information, e.g. user demographic information [7] and global context [39], interest drifts with time [3], and interaction session[14]. Recently, the success of the Transformer network [32] has brought evolution to sequential modeling tasks, and has been soon adapted to the recommendation models [1, 17, 19, 20].

MIP is also a variation of the Transformer network, but different from existing adaptions for two aspects. In contrast to TiSAS [20], MIP is a multi-interest user embedding model; on the contrary to ComiRec [1] and PinText2 [48] which follows the global-query models proposed by [22, 41], MIP uses the items the user engaged with as query and learns even personalized interests.

Multi-interest user representation. Representing users by multiple embeddings greatly improved the recommendation quality, but not every existing recommendation model can easily extend to a multi-interest framework. Classical collaborative filtering and matrix factorization methods do not naturally produce multiple user embeddings, and so do sequential models like RNNs and attention-based models. To discover multiple interests from user engagement history, heuristic methods [16, 45] and unsupervised learning methods like clustering [25, 33] and community mining [34, 44] has been adopted. Besides, researchers have made efforts to modify the existing neural networks to produce multiple results, for instance, the capsule network [1, 19, 28] and multi-head attention models [1, 20, 47]. However, they require an estimation of the number of interests of users as a hyperparameter and do not learn the weight of interests. Therefore, unlike MIP, they produce an equal number of clusters for every user and treat each interest with uniform importance.

3 BACKGROUND

A typical scalable recommendation system involves two stages: candidate generation (retrieval) and ranking. Embedding-based retrieval systems can efficiently retrieve a large volume of candidates via approximate nearest neighbor (ANN) search. And user embeddings can also be used as features by ranking models at the next stage. The key to the success of the recommendation system lies in two folds: 1) we need to deeply understand the potential topics a user is interested in, which can be captured and represented by the user's multi-facet embedding vectors; 2) we need to be able to distinguish multiple interests of a user and understand the relative importance. Importance scores of users' multi-facet embedding vectors can help decide query budget at the candidate retrieval stage and can serve as an estimator of users' likelihood of engaging with a given interest topic at any point.

User interests typically evolve over time. While several topics can draw long-term persistent attention from a user, others can be one-off enthusiasm that trend up during recent sessions and fade away swiftly. Estimating accurate importance scores of users' multi-facet embeddings at any given time becomes highly challenging. A good embedding importance estimator needs to establish a delicate balance between the long-term and the short-term interests, as well as a balance between casual users with 1 or 2 focused interests and core users with 100+ diverse interests.

4 METHODOLOGY

In this section, we formulate the recommendation problem and the neural architecture to model the multiple user interests with preference weights in detail.

4.1 Problem Statement

Let $\mathcal{I}$ denote the collection of items (with the ID v or the feature vector $\boldsymbol{p}$) in the data store, and the $\mathcal{U}$ represents the set of users. For each of user $u \in \mathcal{U}$, we observe the historical sequence of items that the user has engaged with, $\mathcal{S}^u = (\boldsymbol{p}^u_{t_1}, \boldsymbol{p}^u_{t_2}, ..., \boldsymbol{p}^u_{t_{|l_u|}})$ (or $(v_{t_1}, v_{t_2}, ..., v_{t^u_{|l_u|}})$), and the timestamps of the engagement $\mathcal{T}^u = (t^u_1, t^u_2, ..., t^u_{|l_u|})$. The objective is to learn a set of user embedding $\boldsymbol{z}^u_i \in \mathbb{R}^d$ $(i = 1, ..., k)$ and their weights w_i. Since the user representation is learned only from the history of that user, hereinafter, we omit the superscript for u in both input and output side for simplicity. Notations as summarized in Table 1.

We distinguish two cases of the $\mathcal{I}$:

- **Dense feature $\boldsymbol{p}$.** In the industrial recommendation systems, the learning of item features is often decoupled from the recommendation model. For instance, the item feature may contain semantic information extracted from NLP models and visionary features from CV models. Given those feature vectors, the recommendation object mainly focuses on learning the user feature, or user embeddings, from the history of user behaviors.
- **Sparse feature v.** On the contrary, in many public datasets, the items have no attainable feature except item IDs (encoded as one-hot vectors), and the recommendation model must learn the dense representation of features through only the user-item interactions.

The proposed model is capable to handle both cases.

4.2 Process Overview

Overall, the proposed model consists of two parts to learn the multi-interest user representation and cluster weights. Figure 2 shows the model architecture. Respectively, the rest of this section will introduce the sequential module for learning a set of user embedding (Section 4.3), the module for cluster weight prediction (Section 4.4), and the final prediction modules that combines them together (Section 4.5). An unsupervised clustering step (Section 4.7 is required in both parts, which is utilized differently in dense and sparse feature scenarios.

4.3 Multi-interest User Representation

Item Encoding We distinguish two conditions of an item. In most practical systems, every item has a pre-computed feature vector (denote as $\boldsymbol{p}_j$) defined in the space where the distance can capture the similarity between the items. In other cases where the recommendation system knows nothing but unique ids (denoted by a one-hot vector v_j) about the item, the system needs to learn the dense item feature through an embedding layer:

$$\boldsymbol{p}_i = W_{emb}\boldsymbol{v}_i \tag{1}$$

Additionally, how the engaged items reflect the user's interest may be implied by the sequential order and the time when they interact. Thus, we concatenate the item features with both its positional and temporal information to produce the action encoding:

$$\mathbf{e}_j = [\boldsymbol{p}_j; \boldsymbol{\tau}(t_j); \boldsymbol{\rho}(j)] \tag{2}$$

Where the [;] denotes a concatenation of vectors, and the positional ($\boldsymbol{\rho}$) and temporal ($\boldsymbol{\tau}$) encodings are given by [32]:

$$\begin{aligned}
\boldsymbol{\tau}_{2j}(t_i) &= sin\big(t_i/(\tau_{max})^{2j/m_t}\big) \\
\boldsymbol{\tau}_{2j+1}(t_i) &= cos\big(t_i/(\tau_{max})^{2j/m_t}\big) \\
\boldsymbol{\rho}_{2j}(i) &= sin\big(i/(\rho_{max})^{2j/m_p}\big) \\
\boldsymbol{\rho}_{2j+1}(i) &= cos\big(i/(\rho_{max})^{2j/m_p}\big)
\end{aligned} \tag{3}$$

Notations	Description
$\mathcal{I}, \mathcal{U}$	Item set and user set
$\mathcal{S}$	Abbreviation of $\mathcal{S}^u$, engagement history of user u
l	Abbreviation of l_u, length of $\mathcal{S}$
d	The item embedding dimension
$\boldsymbol{p}$	An item embedding, $\boldsymbol{p} \in \mathbb{R}^d$
$\boldsymbol{p}_i, t_i$	Short form of the $\boldsymbol{p}^u_{t_i}$ and t^u_i
k	Maximum number of embedding vectors per user
C	Cluster assignment, $C \in \mathbb{R}^l$.
$\boldsymbol{z}_i, Z$	User embedding vector(s), $\boldsymbol{z}_i \in \mathbb{R}^d$, $Z \in \mathbb{R}^{d \times k}$
h	Attention head superscript, $h = 1, ..., H$
$W^h_q, \boldsymbol{b}^h_q$	Query projection weights and bias
$W^h_r, \boldsymbol{b}^h_r$	Key projection weights and bias
$\mathbb{1}_{[cond]}$	Indicator function, output 1 if *cond* is true, 0 otherwise
$[\cdot]$	Vector concatenation operator

Table 1: Notations

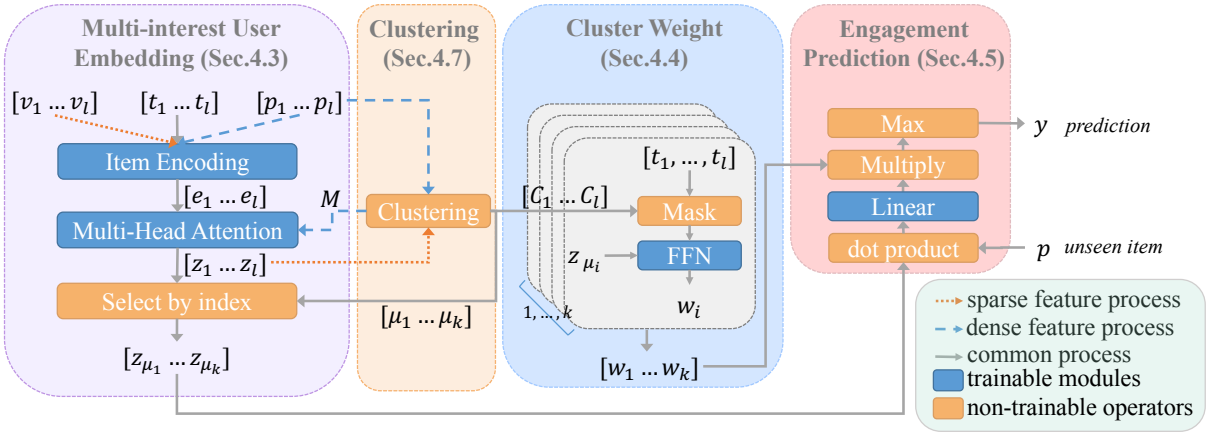

Figure 2: An overview of MIP architecture. The input to the model is the user engagement history containing item embeddings ($[v_1 ... v_l]$ or $[p_1 ... p_l]$) and their timestamps ($[t_1 ... t_l]$). The multi-interest user embedding module produces k embeddings, where k is decided by the clustering method or as hyper-parameter. With clustering and multi-interest representation, the cluster weight module will then estimate the cluster weights for each cluster. Finally, the multi-interest embeddings with corresponding weights are combined to predict the user's interest in an unseen item p. The different process on dense and sparse feature are shown (detailed in Section 4.3,4.7).

The hyper-parameters are set as $\tau_{max} = 10^4$, $m_t = 1$, and $m_p = 1$. The unit of timestamps is *day*. In the Section 7.2, other forms of encodings are compared, and the Equation 3 has a weak advantage.

User Representation Candidates To find only a few vectors to represent the entire user engaged items, the model has to consider the similarity within the engaged items. The process is achieved through the self-attention layer:

$$e^h_{i,j} = \frac{(W^h_q \boldsymbol{e}_j + \boldsymbol{b}^h_q)^\top \cdot (W^h_r \boldsymbol{e}_i + \boldsymbol{b}^h_r)}{\sqrt{d_{model}}} \tag{4}$$
$$a^h_{i,j} = softmax_i(e^h_{i,j})$$

We constraint the attention scores a with a mask matrix M. In the sparse feature case, let $M_{i,j} = 1$ for any i, j; while in the dense feature case, M indicate the intra-cluster relationship, *i.e.* $M_{i,j} = \mathbb{1}_{[C_i = C_j]}$, where the C denotes the cluster ID. Thus, M requires the attention model to only aggregate items within the same cluster. The generation of M in the dense feature case is detailed in Sec. 4.7. With M and attention scores a, each attention head aggregates the sequences as:

$$\boldsymbol{z}^h_j = \sum_i a^h_{i,j} M_{i,j} \boldsymbol{p}_i \tag{5}$$

To process the aggregated vector from all attention heads, the dropout layer and a feed-forward network (FFN) are applied, and compute the output vectors as

$$\boldsymbol{z}_j = FFN(Dropout([\boldsymbol{z}^1_j; ...; \boldsymbol{z}^H_j])) \tag{6}$$

where $j = 1, ..., l$. The $FFN()$ has two fully-connected layers with a hyperbolic tangent activation function after the first layer.

Now, each $\boldsymbol{z}_j$ contains not only the item feature p_j, but also its adjacency with other items and neighbor item features. The remaining question is which vectors from the $\{z_j\}$ are the most representative.

Cluster Representations So far, the multi-head attention module has produced l output vectors $\boldsymbol{z}_1, ..., \boldsymbol{z}_l$, and each $\boldsymbol{z}_i$ uses $\boldsymbol{p}_i$ as the (unprojected) query. To find representative vectors, the model utilizes the clustering results. Denote the last item in each cluster as $\boldsymbol{p}_{\mu_1}, ..., \boldsymbol{p}_{\mu_k}$, we take the $\boldsymbol{z}_*$ that uses the last item as query, to represent the corresponding cluster. Let the user representation be $Z \in \mathbb{R}^{k \times d}$, then

$$Z = [z^\top_{\mu_1}; ...; z^\top_{\mu_k}] \tag{7}$$

4.4 Cluster Weight Module

Besides the multi-interest assumption, it's also likely that the user favors each interest unequally. As mentioned in the earlier section of this paper, ranking these interests can greatly benefit the candidate generation task given its limited budget. In general, a higher weight should be assigned to an interest cluster if the user engages more with items that belong to it. In order to utilize both the contextual cluster information and user's engagement sequence in that cluster, we build a two-layer feed-forward network on top of the cluster representation $\boldsymbol{z}_j$ and the temporal encoding of items $\boldsymbol{\tau}$ that belong to the cluster j. We will mask those items that belong to other clusters out as zero to make the input dimension consistent. The cluster weight can be written as

$$w_j = FFN([\boldsymbol{z}_j; \mathbb{1}_{[C_1 \in L_j]} \cdot \boldsymbol{\tau}_1; ...; \mathbb{1}_{[C_l \in L_j]} \cdot \boldsymbol{\tau}_l]) \tag{8}$$

Concretely, the $FFN()$ consist of two fully-connected layers with sigmoid function as activation function in between. Physically, the clusters learned from Sec. 4.3 are topics that the user likes, so the preference weights of the clusters should be positive. Therefore, the output of the second layer are normalized to the range of $[0, +\infty]$ by softplus function[2].

[2] $y = \frac{1}{\beta} log(1 + exp(\beta x))$. We set $\beta = 1$

4.5 User-Item Engagement Prediction

Intuitively, a user will engage with an item as long as the item match *one* of his/her interests (not *all*). In other words, it's important that the item embedding is very close to one of the user embeddings, rather than being as close as possible to all of them. Therefore, the user-item affinity should depend on the maximum of item embedding and user embedding on each interest dimension. Furthermore, the user's affinity to each interest cluster should be embodied in the likelihood as well: a higher weight should be assigned to an interest cluster when the user engages more with items that belong to it. Therefore, we propose the likelihood that a user will engage with an item as follow:

$$y = \max\{w_j Linear(\boldsymbol{z}_j \cdot \boldsymbol{p})\}_{j=1}^{k} \tag{9}$$

where $Z = [\boldsymbol{z}_{\mu_1}^\top; \cdots; \boldsymbol{z}_{\mu_k}^\top]$ is the aforementioned user multi-interest embedding (on k interest dimensions), and $\boldsymbol{p}$ is the item embedding.

4.6 Loss Function

Given the set of items with a positive label ($\mathcal{I}_+$) and a negative label ($\mathcal{I}_-$), the negative log-likelihood (NLL) loss of our model can be written as:

$$\mathcal{L} = -\frac{\sum_{u\in\mathcal{U}} \left(\sum_{\boldsymbol{p}_i \in \mathcal{I}_+^u} \log(y_i^u) + \sum_{\boldsymbol{p}_i \in \mathcal{I}_-^u} \log(1 - y_i^u) \right)}{\sum_{u\in\mathcal{U}} \left(|\mathcal{I}_+^u| + |\mathcal{I}_-^u| \right)} \tag{10}$$

4.7 Clustering

We have referred to a clustering step in producing attention mask (M in Eq. 5), finding cluster representation through last item in each cluster (μ_k in Eq. 7), and learning the cluster weight (C in Eq. 8). Generically but formally, the clustering algorithm takes the set of vectors and produces the clusters $L_1, L_2, ..., L_k$ which is a partition of $\mathcal{S}$. From the partition, the following items are computed:

Cluster Assignment C. C is a list of the same length as $\mathcal{S}$, and the i-th entry of which is denoted as C_i, representing the cluster ID of items. Let $C_i = r$ if $\boldsymbol{p}_i \in L_r$, for $i = 1, ..., l$ and $r = 1, ..., k$.

Attention Mask M. M is a symmetric binary matrix where the element at i, j indicates if item at position i and item at position j are from the same cluster, *e.g.* $M_{i,j} = \mathbb{1}_{[C_i = C_j]}$.

Last Item Index μ. From each cluster r, denote the index of the latest item as μ_r.

The clustering is applied differently with the dense and sparse features. With dense features, the exogenous item embeddings ($\boldsymbol{p}_j$) are clustered, and the cluster results lead to a non-constant M mask, while with a sparse feature, the clustering is applied to the user representation candidates (Eq. 6).

In this work, if unspecified, the model uses the Ward[37] clustering method[3]. In Section 7.3, other clustering methods are compared

5 EXPERIMENT ON PINTEREST DATA

In this section, we conduct an exhaustive analysis to demonstrate the effectiveness of MIP by examining data from Pinterest, one of the largest online content discovery platforms. In Sec. 6, we show the MIP also works notably well on the public datasets.

[3]We adopted the implementation in scikit-learn https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html#sklearn.cluster.AgglomerativeClustering, with n_cluster=5, and other default arguments.

5.1 Dataset.

The dataset contains the user engagement history collected from Pinterest, an image sharing and social media service that allows users to share and discover visual content (images and videos). The interactions between a user and an item (also referred to as a *pin*) are categorized into *impression* (pin is shown to the user), *clickthrough* (user clicks the pin), *repin* (user saves the pin into their board collection), and *hide* (user manually hides the pin). In total, there are 38 million interactions from 510 thousand users during three weeks period of time. Each pin is represented as a 256-dimension feature extracted by the PinSage model [42].

Active users could have very long engagement sequences in a day. Let the engagement sequence of a user be $\{j_1, j_2, \cdots, j_n\}$, then for each training sample, we use the first l engagements to predict the l future engagements. We also enforce a one-day gap between these two segments, because nearby user engagements are usually very consistent with each other (falling into the same category) and thus make the prediction task easy. l is set to 50, which is the same as the setting in [1]. We treat clickthrough and repin as the positive label, and hide (which is less often), and impression (without click or repin) as the negative label. Since these negative data have also been recommended to the user at some point, they are likely to be still relevant to the user, and thus correlate with the positive data. In order to alleviate the bias, we introduce the *random* negative data where pins are sampled from the whole set of pins. The entire negative dataset will consist of 50% *observed* negative data (impression and hide), and 50% *random* negative data.

5.2 Baselines and Model Configuration.

These recent state-of-the-art models are compared:

- PinnerSage [25]: a clustering-based multi-interest model. The user engagement history are clustered and each cluster is represented by the pin embedding of the cluster medoid. Cluster weight is estimated in the exponential decay form: $w_i = \sum_{\boldsymbol{p}_j \in L_i} e^{-\lambda(t-t_j)}$, with heuristic parameters ($\lambda = 0.01$).
- TiSAS [20]: a self-attention based single-interest framework. The user engagement history with timestamps are fed into a multi-head self-attention network. User embedding is the query vector of the last item in the sequence.
- ComiRec [1]: an attention-based multi-interest framework. The multi-head attention is applied to the engagement history. Each attention head has a global query vector as a trainable parameter so that each attention head can aggregate the input sequence into one vector. H attention heads produce H user embeddings. The number of attention is set to 8, therefore the ComiRec model outputs 8 interests per user engagement history.

	precision@20	recall@20	AUC	NLL
PinnerSage	74.02%	29.61%	0.8145	1.033
Tisas	–	–	0.8496	0.478
ComiRec	86.35%	34.54%	0.8750	0.407
MIP	**88.20%**	**35.28%**	**0.8926**	**0.377**

Table 2: Performance of MIP and baseline models on the Pinterest dataset.

For fair comparison, all models have a single attention layer with $d_{model} = 64$. Our model is trained with Adam optimizer [18] with learning rate 10^{-3}. In MIP, the hidden size of FFN in Equation (6) is 64, and the number of attention head is also 8. The hidden size of FFN in Equation (8) is 32.

5.3 Metrics.

The models are evaluated in the retrieval scenario, where the recommendation system needs to recommend a batch of items to the user, instead of predicting the next item that the user will engage with. To this end, the following metrics are used:

- Area Under the ROC Curve (AUC).
- Negative Log Likelihood (NLL): as of Equation10
- Precision@K: Let $\hat{\mathcal{I}}_K^u$ denotes the top-K recommended items for the user u.

$$Precision@K = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|\hat{\mathcal{I}}_K^u \cap \mathcal{I}_+^u|}{K} \tag{11}$$

- Recall@K:

$$Recall@K = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|\hat{\mathcal{I}}_K^u \cap \mathcal{I}_+^u|}{|\mathcal{I}_+^u|} \tag{12}$$

5.4 Results and Analysis

As shown in Table 2, MIP outperforms all the state-of-the-art models that are published very recently. The detailed comparison of the model and explanation of the improvement are as follows:

- PinnerSage shares the clustering algorithm with MIP, but differs in 1) the cluster is represented by the medoid, 2) and the cluster weight uses the heuristic model. Instead, MIP learns the cluster representations and weights from data end-to-end from data, which aligns with our intuition.
- TiSAS has a similar attention model structure as MIP, except only using the last output of the attention model as the user embedding. The comparison confirms the necessity of multi-interest representation, as in ComiRec and MIP.
- Compared to ComiRec, MIP interestingly shows that self-attention has stronger representation power than attention with global-query. In Appendix A, we use synthetic data to illustrate the internal difference between the two types of models.

6 EXPERIMENTS ON PUBLIC DATASETS

In this section, we evaluate MIP on learning from other collaborative filtering datasets, where the item features are absent and will be learned from the user-item interactions.

Dataset: Three public datasets are used: Amazon-book [4] (hereinafter, Amazon), Taobao[5], and MovieLens[6]. We adopted a 10-core setting as previous works [20, 36] and filtered out rare items that appear less than 10 times in the whole dataset, and the inactive users who interact with less than 100 items. We split each user's engagement history into non-overlapping sequences of length 100, and use the first 50 items to learn the user embedding(s) and the last 50 items as positive samples to rank. For each sequence, another 50 negative samples are uniformly randomly selected from the items that the user does not interact with. The numbers of sequences in the datasets are listed in Table 3.

Models: We evaluate MIP in recommending with sparse feature dataset and include another baseline: GRU4Rec [14], an RNN model for learning single user embedding. The GRU hidden state size and the output dimension in its output layer are 32. Other baseline models adopt the same configuration as in Section 7.3, except that there's another item embedding layer as in Equation (1), which produces vectors of dimension 32. For MIP, the attention takes both temporal and positional encoding. An ablation study of how those encoding influences the performance is included in the Section 7.2.

Training: All the models are trained for 100 epochs on Tesla T4 GPU with an early stop strategy that ceases the training when validation AUC does not improve for 20 epochs. The MIP is trained with two-phase. In the first phase, the parameters in the cluster weight module are frozen, and the weights for any cluster are set as 1. After the model converges in the first phase, the parameter freeze is removed, and then all parameters are trained until converge. Appendix A.1 compared and analyzed this training strategy of MIP.

Results: The performance is summarized in Figure 3. MIP has stronger performance on Amazon and Taobao dataset and is trivially worse than GRU4Rec and ComiRec in AUC on MovieLens. Intuitively, the results might be because the shopping scenario fits more to our multi-interest assumption, users purchase only items from their interested categories, like a type of sports or habits, while the majority of people would watch all types of movies, and if they like the movie largely depends on the movie quality, instead of movie category. In addition, since all models have very close performance, MIP is still a competitive approach in applications that do not support the strong multi-interest assumption.

	Amazon	MovieLens	Taobao
# Items	425,582	15,243	823,971
# Training sequences	57,165	127,212	343,171
# Test sequences	5,000	5,000	10,000
# Validation sequences	5,000	5,000	10,000

Table 3: Dataset statistics.

7 EFFECTIVENESS OF INDIVIDUAL MODULES

In this section, we examine the effectiveness of different modules and design choices in MIP by ablation study.

	MIP (Equal Weight)	MIP
precision@20	87.26%	**88.2%**
recall@20	34.91%	**35.28%**
AUC	0.8851	**0.8926**
NLL	0.388	**0.377**

Table 4: MIP vs. MIP (Equal Weight)

[4] https://jmcauley.ucsd.edu/data/amazon/
[5] https://tianchi.aliyun.com/dataset/dataDetail?dataId=649
[6] https://www.kaggle.com/grouplens/movielens-20m-dataset

Figure 3: Model performance on public datasets

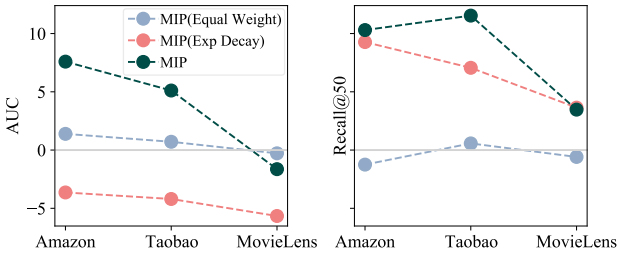

Figure 4: Cluster Weight Variants Performance (difference to the best baseline models)

7.1 Personalized Cluster Weights

Configurations. To validate the assumption that the preference trends (weights of multiple interests) changes from user to user, we compare the MIP with two variants which disable the cluster weight module. The first one (referred as *MIP (Equal Weight)*) constantly assign 1 as cluster weight, thus lead to an equally weighted multi-interest user representation. Another one (referred as *MIP (Exp Decay)*) uses the heuristic formula, an exponential decay weights given by $w_j = \sum_{C_i \in j} exp(-\lambda(t_{now} - t_i))$[25]. We let the t_{now} to be the last user engagement time t_l, since in practice it's unrealistic to update the weights in real time according to current timestamp. According to [25], we also set $\lambda = 0.01$, which balances well between emphasizing on recent engaged items and accentuating on frequently engaged categories. We let number of clusters $k = 5$.

Dataset and Results. We evaluate the variants on cluster weights on Pinterest (Tab. 4) and public dataset (Fig. 4).

Analysis. In most experiments, the AUC and recall can benefit from learned cluster weights (our proposal). From Fig. 4, the equal weights and exponential decay have an advantage on AUC and recall respectively, but learned weights are overall superior to other options. Note that the exponential decay rate is global to any user and any cluster, therefore, the results demonstrated that personalized and context-aware cluster weight estimation is a more reasonable solution to the recommendation.

Dataset	Configuration Choices	AUC	Recall
Amazon	item embedding only	76.34	73.75
	+positional	76.19	74.28
	+temporal	78.59	76.94
	+positional and temporal	**79.31**	**78.22**
Taobao	item embedding only	82.06	81.75
	+positional	86.54	**86.22**
	+temporal	**86.72**	86.56
	+positional and temporal	86.59	85.83
MovieLens	item embedding only	**95.26**	**94.45**
	+positional	95.01	94.31
	+temporal	94.96	94.19
	+positional and temporal	94.61	94.12

Table 5: Ablation study on the positional and temporal encoding on public datasets. (in 10^{-2})

7.2 Positional and Temporal Encoding

In Eq. 2, the sequential (positional) and temporal information are encoded and included in the self-attention module to produce the multi-interest representations. The motivation is that given items from the same category, the recent ones might better represent the user's current interest than the obsolete items. We verify 1) if the incorporation of positional and temporal encoding is critical to the performance; 2) how the encoding (Eq. 3) method affect the performance.

Configuration. The MIP are configured on the two set of choices: the Eq. 2 can be configured alternatively:

- item embedding only: $\boldsymbol{e}_j = \boldsymbol{p}_j$.
- + positional: $\boldsymbol{e}_j = [\boldsymbol{p}_j; \boldsymbol{\rho}(j)]$, where $\boldsymbol{\rho}(j)$ is given by Eq. 3.
- + temporal: $\boldsymbol{e}_j = [\boldsymbol{p}_j; \boldsymbol{\tau}(t_j)]$, where $\boldsymbol{\tau}(t_j)$ is given by Eq. 3.
- + positional and temporal: Eq. 2 and Eq. 3

and there are several other choices of temporal encodings other than the sinusoidal form in the Eq. 3:

- One-hot [47]: Create exponential buckets $[0, b)$, $[b, b^2)$, $\cdots$, $[b^{k-1}, \infty)$ with base b, and encode the timestamp as an one-hot vector, i.e. $\tau_i = lookup(buckets(t))$.
- Two-hot [30]: Similarly create exponential boundaries $\{0, b, b^2, \cdots, b^{k-1}, \infty\}$, and encode the timestamp as $\boldsymbol{\tau}_i = log_b(t) - i$ and $\boldsymbol{\tau}_{i+1} = i + 1 - log_b(t)$, where $b^i \leq t < b^{i+1}$.

Dataset and Results. The options to include positional and temporal information are evaluated on all the dataset (Tab. 5), and the encodings methods are compared exhaustively on Pinterest dataset (Tab. 6).

Analysis. Two conclusions can be made from Tab. 5 and Tab. 6: 1) including both temporal and positional information is a safe option, which has best performance on Amazon and Pinterest and marginally (< 0.01) worse performance on Taobao and MovieLens; and 2) the model is insensitive to encoding methods.

7.3 Clustering Options

The Ward's algorithm is applied to MIP considering its success in PinnerSage[25], it's beneficial to explore the selection of clustering algorithm and number of clusters on the collaborative filtering

Configuration choices	AUC	NLL
item embedding only	0.8923	**0.377**
+ positional	0.8846	0.386
+ temporal (one-hot)	0.8850	0.388
+ temporal (two-hot)	0.8846	0.385
+ temporal (sinusoid)	0.8921	**0.377**
+ positional and temporal (one-hot)	0.8861	0.382
+ positional and temporal (two-hot)	0.8852	0.387
+ positional and temporal (sinusoid)	**0.8926**	**0.377**

Table 6: Influence of temporal and positional encoding in attention on the performance in MIP

dataset. To illustrate the impact, we evaluate MIP with a wide range of clustering algorithms.

Model Configuration and training: MIP models are configured with an attention module that takes both positional and temporal encoding. For unweighted MIP, no clustering method is applied to the encoded user engagement history $\{z_*\}$ (computed from Eq. 6) in the training stage. For weighted MIP, Ward's algorithm is applied to $\{z_*\}$ and the number of clusters is set to 5. To keep the MIP fully differentiable, the cluster embedding is the encoding of the last item in each cluster, instead of the medoid.

Inference: The choice of the clustering in the inference phase is independent of its configuration during the training. We explore the inference options on the pre-trained models. Different types of clustering methods are compared:

- Ward: hierarchical clustering method that minimizes the sum of squared distances within all clusters.
- K-Means: an iterative method also minimized the sum of in-cluster summed squared distances.
- Spectral[31]: performs clustering on the projection of the normalized Laplacian computed from the affinity matrix.
- BIRCH[46]: another hierarchical method that clusters the points by building the Clustering Feature Tree.
- DBSCAN[9]: a density-based clustering method that does not require specifying the number of clusters.

The number of clusters is set to 5, 8, and 10 when required. Note that during training, the number of clusters is fixed to 5, however, after training, MIP can produce other numbers of embeddings per user, which gives the system huge flexibility to trade-off between storage/computation cost and recommendation performance.

Result and analysis: There are two observations from Table 7. 1) the different clustering algorithm has a marginal impact on the performance. While PinnerSage reported that Ward's algorithm outperforms the K-Means, their result does not conflict with our observation here. Recall that for PinnerSage and our experiment on the Pinterest dataset, the clustering method is applied to the exogenous item embeddings, thus the clustering methods can be influenced by the non-flat geometry and outliers. However, with a collaborative filtering dataset, the clustering method is applied to the encodings produced by multi-head self-attention layers which average the embedding of the items and all other items (Eq. 4). The encodings after the multi-head self-attention should be smoothly distributed, and as a result, any clustering methods work almost equally well on that. 2) Selecting the number of clusters is a non-trivial trade-off. The motivation to decrease the number of clusters is the storage and computation cost which grow linearly as the number of clusters increases. For unweighted MIP, though the non-clustering (each item is a cluster) settings have the best AUC, decreasing the number of user embedding from 50 (non-clustering) to 10 is still acceptable. For weighted MIP, since it's impossible to learn the clustering weights without applying a clustering method, the trade-off can be more complicated: besides the storage concern, when the number of the cluster increases the average information to learn the weights of each cluster decreases and consequently may hurt the overall performance; on the other hand, 10-cluster settings are better than the 5-cluster settings for all the dataset.

8 CONCLUSIONS

In this paper, we study the problem of multi-interest user embedding for recommendation systems. We follow the recent findings on representing users with multiple embeddings, which has been proven helpful over the single user representation. However, we find that in industrial recommendation systems, it is important to have a set of weights for these multiple embeddings for a more efficient candidate generation process due to its budget on the number of items returned. More specifically, we define the likelihood of an engagement based on the *closest* user embedding to the item embedding and update the weight for the corresponding cluster. We also illustrate the different numbers of interests (embeddings) that users could have, which is fundamentally different from the assumption of similar works. In addition, an attention mechanism is applied in the model architecture, and we have done extensive studies on different model design choices. Finally, case studies on multiple real-world datasets have demonstrated our advantage over state-of-the-art approaches.

REFERENCES

[1] Yukuo Cen, Jianwei Zhang, Xu Zou, Chang Zhou, Hongxia Yang, and Jie Tang. 2020. Controllable Multi-Interest Framework for Recommendation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2942–2951.

[2] Wei-Cheng Chang, Felix X Yu, Yin-Wen Chang, Yiming Yang, and Sanjiv Kumar. 2020. Pre-training tasks for embedding-based large-scale retrieval. *arXiv preprint arXiv:2002.03932* (2020).

[3] Xu Chen, Yongfeng Zhang, and Zheng Qin. 2019. Dynamic explainable recommendation based on neural attentive models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 53–60.

[4] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[6] Robin Devooght and Hugues Bersini. 2017. Long and short-term recommendations with recurrent neural networks. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*. 13–21.

[7] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. 2017. Sequential user-based recurrent neural network recommendations. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. 152–160.

[8] Alessandro Epasto and Bryan Perozzi. 2019. Is a single embedding enough? learning node representations that capture multiple social contexts. In *The World Wide Web Conference*. 394–404.

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *kdd*, Vol. 96. 226–231.

[10] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on*

Knowledge discovery and data mining. 855–864.
[11] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216* (2017).
[12] Ruining He and Julian McAuley. 2016. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 191–200.
[13] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).
[14] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).
[15] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2553–2561.
[16] Hao Jiang, Wenjie Wang, Yinwei Wei, Zan Gao, Yinglong Wang, and Liqiang Nie. 2020. What Aspect Do You Like: Multi-scale Time-aware User Interest Modeling for Micro-video Recommendation. In *Proceedings of the 28th ACM International Conference on Multimedia*. 3487–3495.
[17] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 197–206.
[18] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[19] Chao Li, Zhiyuan Liu, Mengmeng Wu, Yuchi Xu, Huan Zhao, Pipei Huang, Guoliang Kang, Qiwei Chen, Wei Li, and Dik Lun Lee. 2019. Multi-interest network with dynamic routing for recommendation at Tmall. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2615–2623.
[20] Jiacheng Li, Yujie Wang, and Julian McAuley. 2020. Time interval aware self-attention for sequential recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining*. 322–330.
[21] Yu Li, Liu Lu, and Li Xuefeng. 2005. A hybrid collaborative filtering method for multiple-interests and multiple-content recommendation in E-Commerce. *Expert systems with applications* 28, 1 (2005), 67–77.
[22] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130* (2017).
[23] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7, 1 (2003), 76–80.
[24] Ninghao Liu, Qiaoyu Tan, Yuening Li, Hongxia Yang, Jingren Zhou, and Xia Hu. 2019. Is a single vector enough? exploring node polysemy for network embedding. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 932–940.
[25] Aditya Pal, Chantat Eksombatchai, Yitong Zhou, Bo Zhao, Charles Rosenberg, and Jure Leskovec. 2020. PinnerSage: Multi-Modal User Embedding Framework for Recommendations at Pinterest. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2311–2320.
[26] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. 2017. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *proceedings of the Eleventh ACM Conference on Recommender Systems*. 130–137.
[27] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*. 811–820.
[28] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. 2017. Dynamic routing between capsules. *arXiv preprint arXiv:1710.09829* (2017).
[29] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. 2007. Collaborative filtering recommender systems. In *The adaptive web*. Springer, 291–324.
[30] Hui Shi, Yang Zhang, Hao Wu, Shiyu Chang, Kaizhi Qian, Mark Hasegawa-Johnson, and Jishen Zhao. 2021. Continuous Cnn For Nonuniform Time Series. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 3550–3554.
[31] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence* 22, 8 (2000), 888–905.
[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762* (2017).
[33] Herman Wandabwa, M Asif Naeem, Farhaan Mirza, Russel Pears, and Andy Nguyen. 2020. Multi-interest User Profiling in Short Text Microblogs. In *International Conference on Design Science Research in Information Systems and Technology*. Springer, 154–168.
[34] Fang Wang. 2007. Multi-interest communities and community-based recommendation. (2007).
[35] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 839–848.
[36] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*. 165–174.
[37] Joe H Ward Jr. 1963. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association* 58, 301 (1963), 236–244.
[38] Jason Weston, Ron J Weiss, and Hector Yee. 2013. Nonlinear latent factorization by embedding multiple user interests. In *Proceedings of the 7th ACM conference on Recommender systems*. 65–68.
[39] Bin Xia, Yun Li, Qianmu Li, and Tao Li. 2017. Attention-based recurrent neural network for location recommendation. In *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*. IEEE, 1–6.
[40] Chengfeng Xu, Pengpeng Zhao, Yanchi Liu, Jiajie Xu, Victor S Sheng S. Sheng, Zhiming Cui, Xiaofang Zhou, and Hui Xiong. 2019. Recurrent convolutional neural network for sequential recommendation. In *The World Wide Web Conference*. 3398–3404.
[41] Baosong Yang, Jian Li, Derek F Wong, Lidia S Chao, Xing Wang, and Zhaopeng Tu. 2019. Context-aware self-attention networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 387–394.

Clustering Method	Number of Clusters	Unweighted MIP			Weighted MIP		
		Amazon	Taobao	MovieLens	Amazon	Taobao	MovieLens
None	-	73.11	82.09	95.97	-	-	-
Ward	5	71.56	80.58	95.53	79.31	**86.49**	94.61
	8	71.99	80.99	95.72	80.47	87.85	**95.25**
	10	72.16	81.20	**95.78**	80.84	88.42	**95.25**
K-Means	5	71.58	80.62	95.53	79.26	86.18	**94.86**
	8	71.95	81.03	95.71	80.66	88.02	95.17
	10	72.14	81.22	95.77	80.62	**88.61**	95.10
Spectral	5	**72.28**	**80.72**	**95.54**	78.99	85.84	94.46
	8	0.7237	0.8108	95.73	**80.79**	87.61	94.81
	10	**72.64**	**81.26**	**95.78**	**81.19**	88.40	95.07
BIRCH	5	71.98	80.63	95.52	**79.39**	86.29	94.61
	8	0.7203	81.02	95.71	80.65	**88.03**	**95.25**
	10	72.44	81.21	**95.78**	80.91	88.53	**95.25**
DBSCAN[9]	-	71.98	80.63	95.52	70.05	75.58	89.63

Table 7: Comparison of clustering options in AUC (in 10^{-2}). The best performance with same number of clusters are in bold.

[42] 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 *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.

[43] Jiaxuan You, Yichen Wang, Aditya Pal, Pong Eksombatchai, Chuck Rosenburg, and Jure Leskovec. 2019. Hierarchical temporal convolutional networks for dynamic recommender systems. In *The world wide web conference*. 2236–2246.

[44] Li Yu. 2008. Using ontology to enhance collaborative recommendation based on community. In *2008 The Ninth International Conference on Web-Age Information Management*. IEEE, 45–49.

[45] Wu Yue and Chen Xiang. 2012. A multi-interests model of Recommendation System based on Customer Life Cycle. In *2012 Fifth International Conference on Intelligent Computation Technology and Automation*. IEEE, 22–25.

[46] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: an efficient data clustering method for very large databases. *ACM sigmod record* 25, 2 (1996), 103–114.

[47] Chang Zhou, Jinze Bai, Junshuai Song, Xiaofei Liu, Zhengchao Zhao, Xiusi Chen, and Jun Gao. 2018. Atrank: An attention-based user behavior modeling framework for recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[48] Jinfeng Zhuang, Jennifer Zhao, Anant Subramanian, Srinivas, Yun Lin, Balaji Krishnapuram, and Roelof van Zwol. 2020. PinText 2: Attentive Bag of Annotations Embedding. In *Proceedings of DLP-KDD 2020*.

Model	Global query		Self-attention
	Non-shared	Shared	
Key vector $\boldsymbol{k}_j$	$(W_r^h \boldsymbol{p}_j + \boldsymbol{b}^h)$		
Query vector	$\boldsymbol{q}^h$	$\boldsymbol{q}$	$\boldsymbol{q}_i^h = W_q^h \boldsymbol{p}_i + \boldsymbol{b}_q^h$
$\boldsymbol{q}$ shared between sequences	Yes		No
$\boldsymbol{q}$ shared between att. heads	No	Yes	No
Dot-product	$e_j^h = \boldsymbol{q}^{h\top} \cdot \boldsymbol{k}_j^h$	$e_j^h = \boldsymbol{q}^{\top} \cdot \boldsymbol{k}_j^h$	$e_{i,j}^h = \boldsymbol{q}_i^{h\top} \cdot \boldsymbol{k}_j^h$

Table 8: Variations of multi-head attention.

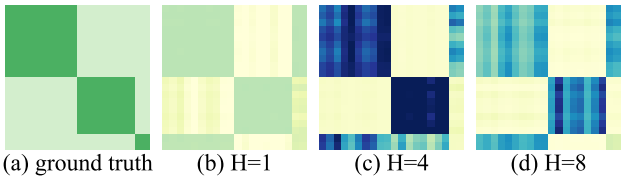

Figure 6: Learned attention scores in self attention model. Darker color represents higher values. (a) the indicator function $\mathbb{1}_{[C_i=C_j]}$, (b-d) $a_{i,j}$. The input sequence is re-ordered for better visualization.

A VARIANTS OF ATTENTION MECHANISM

To interpret the performance improvement of our models against other attention models that have been applied in the recommendation system. We further construct a synthetic dataset and visualize the internal attentions and the user representations aggregated from different attentions.

Synthetic dataset. Without loss of generality, we assume there are G global clusters in the corpus, representing different global categories, each of which is a d-dimensional Gaussian distribution. Each user is interested in up to k ($< G$) categories, referred to as user clusters. We generate the oracle user interest model by sampling no more than k clusters from G global clusters following a multinomial distribution. Then each of the items in the user engagement history is sampled in two steps: uniformly sample one cluster from user clusters, then sample from the d-dimensional Gaussian distribution. Note that in the synthetic model, the item-to-cluster affinity is measured in Euclidean distance, while in the recommendation model, the affinity is decided by cosine distance. To eliminate this discrepancy, we force the Gaussian distributions to center on the unit sphere, so that the rankings by cosine distance and Euclidean distance are consistent.

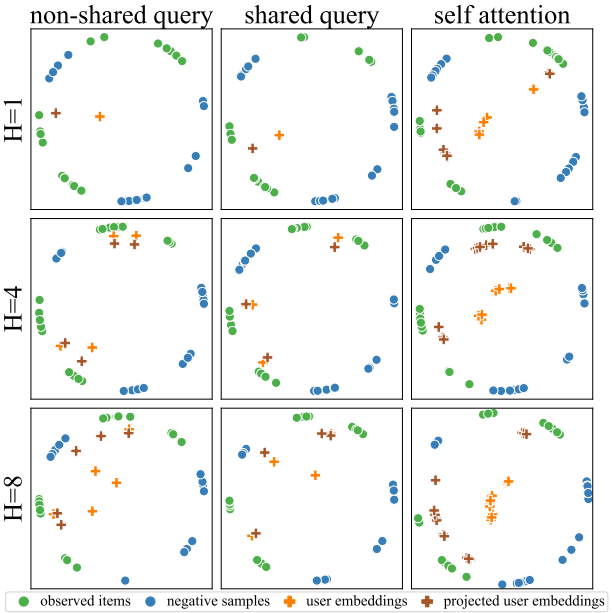

Figure 5: Learned user representations.

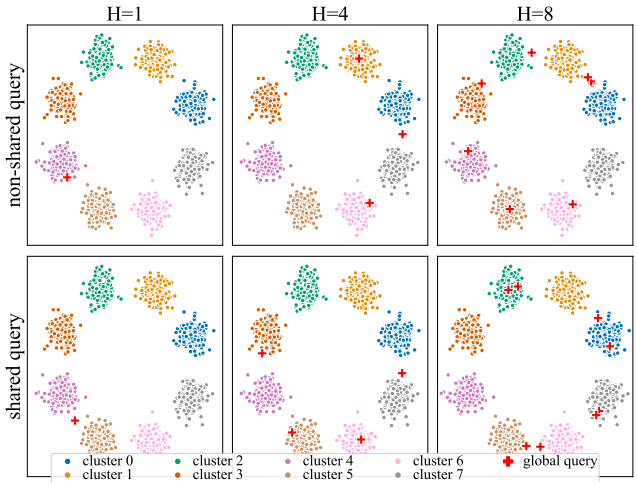

Figure 7: Learned global interests in global query models. The query vector is reversely projected and normalized.

We use a 2D dataset ($d = 2$) for visualization purpose and another high-dimensional dataset for quantitative evaluation. For 2D dataset, we set a relatively small $G = 8$ and $k = 4$ in order to have a clear boundary between clusters. Since there are only 162 distinct subsets[7] with $G = 8, k = 4$, we use 100 of them for training, 31 for validation and the remaining 31 for testing. For high-dimensional dataset, we set $G = 1024$ and $k = 8$, and let $d = 16, 32, 64, 128$. We generate 10000 users for training, 1000 for validation and another 1000 for testing.

Attention models. We focus on comparing our attention model (i.e. *self-attention*), the attention model utilized in ComiRec [1] (i.e. *Non-shared query*), and the one used in PinText2 [48] (i.e. *Shared query*). The comparison of the attentions are in Table 8. For simplicity, we remove the temporal and positional encoding from the computation of attentions, skip the Ward clustering step from MIP, and directly represent user as Equation 7. Also, the dropout layer is removed in order to eliminate randomness in visualization.

Metrics. We visualize the intermediate results and user representations learned from the 2D dataset for qualitative evaluation. For high-dimensional data, we evaluate the performance by AUC and normalized discounted cumulative gain (nDCG).

[7] Number of ways to select no more than 4 clusters from a pool of 8 clusters: $162 = \binom{8}{1} + \binom{8}{2} + \binom{8}{3} + \binom{8}{4}$

Model	Amazon			Taobao			MovieLens		
	Epoch	AUC	Recall	Epoch	AUC	Recall	Epoch	AUC	Recall
MIP (Equal Weight)	22	73.11	66.67	4	82.09	74.86	34	95.97	90.06
MIP (two-phase)	6	72.81	74.65	4	88.18	88.10	6	93.04	93.34
MIP (direct-train)	18	57.61	57.47	28	80.33	80.23	14	92.43	93.68

Table 9: Comparison of training strategy and the performance of weighted MIP. The columns Epoch shows the training epochs when best validation AUC is achieved.

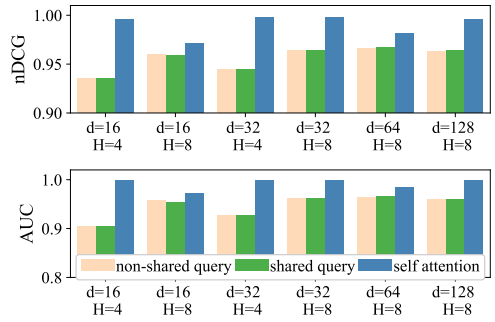

Figure 8: Performance comparison on high dimensional synthetic dataset. d denotes the feature dimension and H is the number of attention heads.

Qualitative results and interpretations. Figure 5 shows the learned user representations given the engagement history. There are three observations. 1) When $H = 1$, global query attention fails to capture all the user interests, while the self-attention model is free from the limitation. 2) Viewing from the third row, the self-attention model is more accurate in learning cluster representations than global query models. The latter is systematically biased due to the global query as shown in global query models Figure 7. 3) All the models learn super-clusters, depending on the bias in the dataset. For the example shown in Figure 5, the two adjacent clusters on the top side of the unit circle are often represented to be a super-cluster.

We also visualize the internal attention scores and self attention models (Figure 6). Some attention heads show highly similar attention patterns because their queries are close to each other, which can be verified from Figure 7. Figure 6 compares the ground truth attention model with the learned attention. The learned attention shows clear boundaries between clusters in the heatmap. Note that the ground truth ignores the adjacency of clusters but the self-attention model considers the similarity between clusters, so Figure 6(a) is block-diagonal while Figure 6(b-d) has dark blocks off the diagonal.

Quantitative results. Previous results show an intuitive comparison between global query models and the self-attention model, and the quantitative results further confirm the consistency of performance gain of self-attention. Experiments are repeated on dataset for feature dimension $d = 16, 32, 64, 128$ and number of attention heads $H = 4, 8$. Figure 8 shows that the MIP model constantly and significantly outperforms global query models. As illustrated in the 2D dataset, the performance gain benefits from the personalized user representation, rather than matching to the globally popular clusters. Another observation from the result is that for global query models, $H = 4$ under-performs $H = 8$ models, as the number of attention heads decides the number of global clusters the model can learn; however, for the self-attention model, $H = 4$ performs even better than $H = 8$. The explanation is that the self-attention model does not require a growing number of attention heads with respect to the number of global clusters, and $H = 4$ could be already enough for capturing user interest but easier than $H = 8$ to train.

A.1 Training Weighted MIP with Sparse Feature Dataset

As mentioned in Section 6, when training MIP on a collaborative filtering dataset, we first train the equally weighted MIP until converged, then lift the parameter freezing to train the cluster weight module together. We detailed the reason and explanation of the strategy here.

Configurations. The model is trained twice with the same configuration in Section 6. The training adopts a two-phase. The first phase model is denoted as *MIP (Equal Weight)*, and the final model is denoted as *MIP (two-phase)*. In the second round, the model is directly trained from random initialization and is denoted as *MIP (direct-train)*.

Results: Table 9 shows the performance and number of training epochs to achieve the best performance. The difference between the two-phase and direct train is significant. Two-phase training not only shortens the training process but also achieves remarkably better results.

Analysis: It's unsurprising that two-phase can accelerate the training, but we need to understand why random-initialized MIP under-performs equally weighted MIP, while the MIP is a generalization to equally weighted MIP. The clue is lying in the experiment on the Pinterest dataset. On the Pinterest dataset, the weighted model is randomly initialized and outperforms the equally weighted one. The difference between the two experiments is two-fold: 1) the Pinterest dataset contains dense item features thus no item embedding matrix needs to be learned; 2) the clustering algorithm is applied to the item embedding in the Pinterest dataset but to the output of multi-head self-attention (MHA) in the collaborative filter dataset. As a result, in the Pinterest experiments, the clustering assignment is reasonably reliable throughout the training, since the item embeddings are always meaningful. On the contrary, on the collaborative filtering dataset, with the random initialization, the item encodings after MHA are much noisier at the beginning of the training, and thus the cluster assignment can be fairly random. Then the input and output of the cluster weight module may be meaningless, which causes hardship in learning the weighted MIP.