Learning User Representations with Hypercuboids for Recommender Systems

Shuai Zhang  
Department of Computer Science,  
ETH Zurich  
shuai.zhang@inf.ethz.ch

Huoyu Liu  
Alibaba Group  
huoyu.lhy@alibaba-inc.com

Aston Zhang  
University of Illinois at Urbana-Champaign

Yue Hu  
Alibaba Group

Ce Zhang  
Department of Computer Science,  
ETH Zurich

Yumeng Li, Tanchao Zhu,  
Shaojian He, Wenwu Ou  
Alibaba Group

ABSTRACT
Modeling user interests is crucial in real-world recommender systems. In this paper, we present a new user interest representation model for personalized recommendation. Specifically, the key novelty behind our model is that it explicitly models user interests as a hypercuboid instead of a point in the space. In our approach, the recommendation score is learned by calculating a compositional distance between the user hypercuboid and the item. This helps to alleviate the potential geometric inflexibility of existing collaborative filtering approaches, enabling a greater extent of modeling capability. Furthermore, we present two variants of hypercuboids to enhance the capability in capturing the diversities of user interests. A neural architecture is also proposed to facilitate user hypercuboid learning by capturing the activity sequences (e.g., buy and rate) of users. We demonstrate the effectiveness of our proposed model via extensive experiments on both public and commercial datasets. Empirical results show that our approach achieves very promising results, outperforming existing state-of-the-art.

CCS CONCEPTS
• Information systems → Recommender systems; • Computing methodologies → Neural networks.

KEYWORDS
Recommender Systems, Hypercuboids, User Representation

1 INTRODUCTION
We propose a novel recommendation approach with hypercuboids. The hypercuboid (also known as hyperrectangle) is a generalization of a three-dimensional cuboid to four or more dimensions. More specifically, we use the expressive hypercuboids to represent users in order to capture their complex and diverse interests. Recommendations are made based on the relationships between user hypercuboids and items.

This work is motivated by the recent popularity and growing importance of recommender systems in real-world applications (e.g., e-commerce and online entertainment platforms). As an effective tool to ameliorate information overload and generate income, its importance cannot be underestimated. A major step in recommender systems is finding the right items that suit users’ preferences and interests. However, obtaining informative user representations is a challenging task as users can have a wide range of interests and different tastes concerning different categories of items. The massive amount and diversity of item sets in real world applications make the case even worse.

Common user modeling approaches aim to represent users as points in vector space. For example, the standard factorization model [17] factorizes the large interaction matrix into user and item latent embeddings in low-dimensional space. As a result, recommendation is reduced to identifying items which are near or similar to the user in that space. Despite its popularity and wide applications, this paradigm faces weaknesses. On one hand, embedding the disparate and diverse interests into single points is problematic [42]. Intuitively, users are likely to have different tastes
and considerations for items from different categories or domains. On the other hand, for most factors, users have preferred ranges spread. For example, consumers usually have their acceptable price range. It is inexpensive to model such range with a single value. Clearly, such systems are geometrically restrictive.

Thus, we need a much richer latent representation system to overcome these limitations. To this end, we propose a new paradigm that represents users as hypercuboids. We go beyond point embeddings, adopting the more expressive hypercuboids to effectively capture the intricate and complex interests of users. A hypercuboid is a subspace that encapsulates innumerable points, enabling more representation capacities than a single point. Moreover, the preference range of each factor (e.g., accepted price) can be naturally modelled with the edge of hypercuboids.

Overall, the basic idea is that: for each user, we learn single or multiple hypercuboids to represent the user’s interests. The relationship between the user hypercuboids and item points is captured by a re-designed distance measure, where both distances outside and inside the hypercuboid are considered. Ideally, the user hypercuboid will allow for a greater degree of flexibility and expressiveness.

The main contributions of this work are as follows:

- We present a novel user representation method for personalized recommendation. For the first time, we adopt hypercuboids to represent users’ complicated user preferences or interests. Two variants are proposed to further enhance the representation capacity in capturing the diversity of user interests.
- We design a distance function to measure the distance between items and user hypercuboids. This measure provides improvements in terms of flexibility and model expressiveness of the algorithms.
- We conduct experiments on six datasets. Our model achieves the best performances on all the benchmark datasets. We investigate the inner workings of our model to gain a better intuition pertaining to the model’s high performance. We also provide case study on real-world datasets to show the effectiveness of the proposed approach.

2 THE PROPOSED METHOD

In this section, we formulate the hypercuboid representation for user interest modeling, introduce two variants of hypercuboid, and present the overall architecture for hypercuboid learning.

Our work is concerned with the recommendation task with implicit feedback. In a standard setting, we suppose that there are a set of users, user \( u \in \{1,\ldots,|U|\} \) and a set of items, item \( i \in \{1,\ldots,|I|\} \). The goal is to generate a set of recommendable items for a user given existing feedback, typically, implicit feedback such as clicks, buys, likes. We set the interaction between user \( u \) and item \( i \), \( y_{ui} \), to 1 if the implicit feedback exists, otherwise, \( y_{ui} = 0 \), where \( y_{ui} \) is an entry of the interaction matrix \( Y \in \mathbb{R}^{|U|\times|I|} \). In addition, time steps will also be considered in the following subsections.

2.1 Representing Users as Hypercuboids

Hypercuboid is an analogue of a cuboid in four or more dimensions. For the convenience of computation, we define a hypercuboid with a center and an offset, both with the same dimensions. Formally, we operate on \( \mathbb{R}^d \). Let \( c \in \mathbb{R}^d \) represent the center of a hypercuboid and \( f \in \mathbb{R}^d_+ \) denote the non-negative offset. The offset is used to determine the lengths of the edges of the hypercuboid. A hypercuboid is defined as:

\[
\text{Hypercuboid} \equiv \{ p \in \mathbb{R}^d : c - f \leq p \leq c + f \},
\]

where \( p \) denotes the point which is inside the hypercuboid.

In our framework, every user is modeled with such a hypercuboid. We use bold uppercase letters \( C \in \mathbb{R}^{|U|\times d} \) and \( F \in \mathbb{R}^{|U|\times d}_+ \) to denote the centers and offsets respectively for all the users. As such, a user is represented by a hypercuboid user \( (C_u, F_u) \). Items are points and are embedded by \( V \in \mathbb{R}^{|I|\times d} \). Obviously, an item point can be outside, inside, or on the surface of the user hypercuboid. A proper method is needed to measure the relationships between users and items. Meanwhile, the characteristics of hypercuboid should be retained.

To achieve this, a composition points-to-hypercuboid distance is adopted. It consists of an outside distance and an inside distance. Firstly, we find the nearest point on the surface of the hypercuboid to the item, which is performed with the following expression:

\[
p_{ui} = \min(p_{u}^{\max}, \max(p_{u}^{\min}, V_{i})),
\]

where \( p_{u}^{\max} \) and \( p_{u}^{\min} \) denote the upper-right corner and lower-left corner of the user hypercuboid (shown in Figure 2). Their formal definitions are:

\[
p_{u}^{\max} = C_u + F_u
\]
\[
p_{u}^{\min} = C_u - F_u.
\]

It is clear that \( p_{ui} \) is relevant to both the user hypercuboid and the item embedding. The movement of either will result in transformation of this point.

Then, the outside distance is the distance between \( p_{ui} \) and \( V_i \), and the inside distance indicates the distance between \( p_{ui} \) and the
center of the hypercuboid. So we have:

\[ \ell_{\text{out}}(u, i) = \| p_{ui} - V_i \|^2_2 \]
\[ \ell_{\text{in}}(u, i) = \| p_{ui} - C_u \|^2_2, \]

where euclidean distance is used. Then, we compose these two distances as follows to measure the user-item relationships.

\[ \ell(u, i) = \ell_{\text{out}}(u, i) + \gamma \cdot \ell_{\text{in}}(u, i), \]  

where the coefficient \( \gamma \) is employed to control the contribution of the inside distance. Theoretically, if we set \( \gamma = 0 \), we presume that the user is interested in an item regardless of its exact coordinate as long as this item is located in the user hypercuboid. This might alleviate the congestive problem of common approaches which map users and her interacted items onto the same points [35].

This paradigm may suggest two benefits. On one hand, a user hypercuboid can be regarded as a set that contains countless item points, which means that it could encompass as many items as the user likes without much loss of information, enabling higher capacities. On the other hand, the composition distance in (3) could overcome an obvious impediment of the direct distance adopted in the existing body of literature. To explain the latter, suppose that we have two items located at the circumference of the circle with center \( C_u \). With commonly used euclidean distance, the system will conclude that this user likes both items equally. However, consider the case in Figure 2. As we can see, this might not be true when the edges of the hypercuboid is taken into account: it is clear that the distances to the hypercuboid of item1 and item2 are not equal. This is provable with the cosine theorem, or by directly setting the hyper-parameter \( \gamma \) to 0. In this example, we can clearly conclude that \( \ell(u, \text{item2}) > \ell(u, \text{item1}) \) so this user is more interested in the item1. One might think of the offset as a range for user preference in each dimension. If the y-axis indicates prices and the z-axis represents styles, it is evident that this user has less tolerance in style than that in the price factor. As such, even though item2 has a more acceptable price than item1 for this user, she still likes item1 more because the item2 does not suit her taste in style at all.

To ensure the items in a hypercuboid to be higher ranked than items outside the hypercuboid, we can add an additional distance to \( \ell(u, i) \).

\[ \ell_{\text{add}}(u, i) = 2(\sigma(\| p_{ui} - V_i \|^2_2) - \frac{1}{2}) \| F_u \|^2_2. \]

where \( \alpha \) is set to a value greater than 100 and \( \sigma \) is the sigmoid function. This distance equals to zero when item \( i \) is in the hypercuboid. Otherwise, it is approximately greater than the half of the maximum diagonal length of the hypercuboid.

2.2 Variants of Hypercuboids

To further enhance the capability, we devise the following two variants of hypercuboids. The rational is to generate multiple hypercuboids to represent each user. In this case, we add several offset embeddings and keep the center unchanged. With concentric hypercuboid we can not only extend the regular hypercuboid into irregular space, but also mitigate the above problem. Suppose we have \( M \) offsets, there are two distances for each hypercuboid. We integrate them into the following final distance:

\[ \ell(u, i) = \sum_{j=1}^{M} \ell^{j}_{\text{out}}(u, i) + \gamma \cdot \min(\ell^{1}_{\text{in}}(u, i), ..., \ell^{M}_{\text{in}}(u, i)). \]

All the outside distances are summed up while only the minimum inside distance is kept. The motivation is to weaken the effects of the inside distances.

2.2.2 Multiple Independent Hypercuboids. To further enhance the capability in modeling diverse interests, we can also use multiple hypercuboids with different centers and offsets. The final distance function is defined as:

\[ \ell(u, i) = \min(\ell^1(u, i), ..., \ell^M(u, i)). \]

As for the addition distance, we can take the maximum additional distance from all hypercuboids.

2.3 Learning the Centers and Offsets

An important step is to learn the center and offset of the user hypercuboid. Generally, user interests can be derived from users’ historical behaviors, e.g., buy, watch, and click. So it is natural to learn the centers and offsets from the past interaction log. In practical applications, user actions are often ordered and timestamped, which also records users’ interests evolution. Formally, we make use of the last \( L \) actions of users, and aim to predict the next recommendable item(s). At the embedding layer, we get an embedding matrix \( S_u^{(t)} \in \mathbb{R}^{L \times d} \), to represent these \( L \) items at time-step \( t \), by looking up the item embedding matrix \( V \) with the index of the \( L \) items.

To capture the shifts of user interests, we employ a stacked neural network consisting of a bidirectional LSTM [28] and a self-attention module over the \( L \) sequential actions.

\[ S' = \text{Bi-LSTM}(S_u^{(t)}). \]

The bidirectional LSTM is used to enhance the sequence representation capability. Then, a self-attention network is applied.

\[ a_u^{(t)} = \text{softmax}\left( f(S') \cdot f(S')^T \right) \cdot S_u^{(t)}, \]

where \( a \in \mathbb{R}^{L \times d} \) is transformed representation for the current sequence; \( f \) is nonlinear layer. It is then passed into a pooling layer
For multiple hypercuboids, we can add dense layers over the 2-dimensional vector.

to get a \( d \) dimensional vector.

\[
s_u^{(t)} = \text{pooling}(a_u^{(t)}),
\]

where the pooling function can be average-pooling, max-pooling, min-pooling, or sum-pooling. We can use \( s_u^{(t)} \) to represent the center, the offset, or both.

In addition, we notice that the offset of hypercuboid usually needs to memorize mass amount of information. For example, to get the price interval, we need to store the prices of all items the user bought to accurately infer her acceptable price range. To this end, we adopt a key-value memory network to perform the memorization process. Let \( M \in \mathbb{R}^{d \times N} \) denote the memory and \( K \in \mathbb{R}^{d \times N} \) denote the key matrix, where \( N \) controls the capacity of the memory.

The input of this memory module is \( s_u^{(t)} \), which is used to retrieve the key from the key matrix.

\[
k = \text{softmax}(s_u^{(t)}K).
\]

With this attentive key vector \( k \), we select relevant pieces of information from the memory.

\[
m = k \cdot M^T.
\]

We wrap all the offsets with the ReLU function [5] to ensure all the values are non-negative. Figure 4 illustrates the overall architecture of the proposed model.

For multiple hypercuboids, we can add dense layers over the center/offset and get multiple centers and offsets.

### 2.4 Optimization

In this section, we introduce the objective function and training scheme. We adopt the pairwise ranking loss for the optimizing objective, which is defined as.

\[
\mathcal{L} = \sum_{(u,i) \in T^+} \sum_{(u,j) \in T^-} \max(0, t(u,i) + \lambda - t(u,j)),
\]

where \( T^+ \) and \( T^- \) are the positive user-item pairs and negative user-item pairs respectively, \( \lambda \) is the margin that separates the two items. For regularization, standard \( L_2 \) normalization and dropout [31] are viable.

We use the same negative sampling strategy as that of HGN [20]. Our model is end-to-end differentiable and an adaptive gradient descent optimizer is used. It is noteworthy that other losses such as Bayesian personalized ranking loss or cross entropy loss are also applicable.

## 3 EXPERIMENTS

In this section, we present our experimental setup and empirical evaluation to verify the effectiveness of our method.

### 3.1 Datasets

We conduct our empirical evaluation based on four public well-studied benchmark datasets and two commercial datasets. The statistics of the datasets are reported in Table 1.

| Datasets          | Interactions | # Users | # Item | % Density |
|-------------------|--------------|---------|--------|-----------|
| Books             | 1,856,747    | 52,406  | 41,264 | 0.086     |
| CDs               | 475,265      | 17,052  | 35,118 | 0.079     |
| Movie & Tvs       | 694,970      | 23,337  | 33,582 | 0.089     |
| E-commerce        | 5,560,954    | 104,929 | 69,563 | 0.076     |
| Alibaba Dataset V1| 28,926,924   | 145,057 | 92,364 | 0.216     |
| Alibaba Dataset V2| 8,558,837    | 131,692 | 109,670| 0.059     |

### 3.1.1 Public Datasets

Four public datasets from different domains are used, including Amazon-Books, Amazon-Movies&TVs, Amazon-CDs [8, 20, 23] and an E-commerce dataset [19]. The feedback in the Amazon datasets is ratings from 1 to 5, we treated those with no less than 4 as positive feedback. For the E-commerce dataset, we use the version provided by Lv et al. [19] which consists of user-item interaction logs of four weeks. To filter noisy data, we discard users with fewer than ten actions and items with fewer than 5 associated actions for each dataset [20].

### 3.1.2 Commercial Datasets

This dataset is from an online retailer platform, Alibaba. It is collected from the interaction logs generated in seven days. We sampled two versions for two groups of users. The first version consists of users with 100 to 500 interactions and the second version consists of users with 50 to 100 interactions.

We target at the next item prediction task where the aim is to predict a user’s next interaction. As such, each dataset is split into three subsets based on the chronological order of interactions, with 70% of the user sequence as the training set, the next 10% as the validation set and the latest 20% as the test set. Same as [20], the validation set is used during the inference stage.
3.2 Evaluation Measures
Three commonplace evaluation measures \([29]\) including Recall@k, Normalized Discounted Cumulative Gain (NDCG@k), and Mean Average Precision (MAP@k) are adopted. For each user, we rank all the items (exclude items the user has rated) and select the highest ranked k items for recommendation. Recall at k indicates the proportion of relevant items found in the top-k recommendations. NDCG at k is a measure of ranking quality and the position of correctly recommended item is taken into account. MAP at k is the mean of the average precision scores of all users and is a precision measure that gives larger credit to correctly recommended items. Definitions are omitted for brevity.

3.3 Compared Methods
We compare our method with two groups of well-established and competitive baselines. The first group contains two non-neural recommendation methods.

- **Matrix Factorization with Bayesian Personalized Ranking (BPRMF)** \([26]\) is a classical method with pairwise Bayesian ranking loss. It models user-item interactions with matrix factorization.

- **Translational Recommender Systems (TransRec)** \([6, 7]\) is a sequential recommendation model where users and items are embedded into a transition space following the prev. item + user = next item translation operation.

Five neural networks based recommendation models are included in the second group.

- **YouTubeDNN** \([2]\) is a deep neural networks based recommendation model proposed by YouTube \(^1\). User vectors, vectors of the latest interacted items and the target item are concatenated and are fed into multiple dense layers.

- **Gated Recurrent Unit for Recommendation (GRU4Rec)** \([10]\) is a recurrent neural network based model for session-based recommendation (a user is a session).

- **Convolutional Sequence Embedding Recommendation Model (Caser)** \([34]\) is a convolutional neural networks based sequence-aware recommendation model. The proposed framework captures the sequential patterns at both point-level and union-level.

- **Self-Attentive Sequential Recommendation (SASRec)** \([15]\) is a self-attention \([36]\) based recommendation model, where user sequences are modelled with a transformer like network.

- **Hierarchical Gating Networks (HGN)** \([20]\) is a state of the art sequential recommendation model which is based on a novel gating mechanism and includes item-item relations.

3.4 Implementation Details
We implemented our model and baselines with tensorflow \(^2\). For all models, the embedding size \(d\) is set to 100 for fair comparison, the sequence length \(L\) and the target length \(T\) are set to 5 and 3, respectively. Adagrad optimizer \([3]\) is used for model parameters updates and the learning rate is set to 0.05. The \(\ell_2\) regularization rate is tuned amongst \(\{0.1, 0.01, 0.001, 1e-4, 1e-6\}\). The margin of hinge loss \(\lambda\) is set to 0.5. The coefficient \(\gamma\) is selected from \(\{0.1, 0.3, 0.5, 0.7, 0.9\}\). Learnable parameters are initialized with a normal distribution and trained from scratch. The batch size is set to 4,096. The memory capacity \(N\) is determined from \(\{10, 20, 30, 40, 50\}\). For all pooling operations in the model, average pooling is used by default. All the hyper-parameters are tuned with the validation set. Experiments are executed for five times and the average performances are reported.

3.5 Experimental Results
Table 2 and Figure 5 present the recommendation performance of all baselines and our model with a single hypercuboid on the six datasets. The best performer on each row is highlighted in boldface. We observe that our model achieves the best performance across all datasets, outperforming a myriad of complex architectures. Our model consistently performs better than HGN, SASRec, Caser, and TransRec, which are all recent competitive sequence-aware recommendation methods. Notably, we obtain a clear performance gain over the second best baseline. Furthermore, our method performs much better than the inner product model (BPRMF) and the Euclidean distance based model (TransRec), which ascertains the effectiveness of the proposed distance function.

Among all the neural networks based baselines, HGN outperforms self-attention, RNN, and CNN based approaches. The second best neural networks based model often switches between YouTubeDNN and SASRec. We also note that non-neural models such as BPRMF and TransRec achieve very competitive performance on the three Amazon datasets and the E-commerce dataset.

Performance of top 5, 10, 20, 30, 50 on four public datasets are presented in Figure 5. Our model consistently outperforms all the other models, indicating that our model can generate not only more accurate topmost recommendations but also higher quality recommendation lists on the whole.

3.6 Adding More Hypercuboids
We conduct experiments to verify the effectiveness of the proposed hypercuboid variants. Table 3 shows the performance with varying number of hypercuboids. Notably, adding more hypercuboids can further enhance the model performance, hitting a new state-of-the-art, which reaffirms the importance of modeling the diversity of user interests. Generally, multiple independent hypercuboids usually marginally outperforms multiple concentric hypercuboids. Using four to six hypercuboids can usually lead to a satisfying improvement. We also tried eight and ten hypercuboids but performance degradation is observed. Note that these two variants will not incur large increases in the number of parameters as increasing the embedding size. We can choose the number of hypercuboids base on the degree of diversity of users’ interests in general.

3.7 Using Hypercuboids Only
We study the impacts of the neural architecture on the model performance. To this end, we report the performance by replacing all the neural architectures with a simple average pooling over the

---

\(^1\)https://www.youtube.com/

\(^2\)https://www.tensorflow.org/
Table 2: Performance comparison on two Alibaba datasets. Only k=10 and k=50 are reported due to space limitation. Only one hypercuboid is used in our model.

| Datasets          | Measure | k  | BPRMF | TransRec | YouTubeDNN | GRU4Rec | Caser | SASRec | HGN | Ours  |
|-------------------|---------|----|-------|----------|------------|---------|-------|--------|-----|-------|
| Alibaba Dataset V1| Recall  | 10 | 0.0156| 0.0192   | 0.0251     | 0.0196  | 0.0227| 0.0233 | 0.0271| 0.0299 |
|                   |         | 50 | 0.0459| 0.0608   | 0.0776     | 0.0631  | 0.0683| 0.0692 | 0.0774| 0.0885 |
|                   | MAP     | 10 | 0.0284| 0.03.0   | 0.0487     | 0.0367  | 0.0451| 0.0466 | 0.0558| 0.0610 |
|                   |         | 50 | 0.0108| 0.0151   | 0.0211     | 0.0163  | 0.0192| 0.0192 | 0.0232| 0.0231 |
|                   | NDCG    | 10 | 0.0665| 0.0789   | 0.1041     | 0.0808  | 0.0952| 0.0989 | 0.1141| 0.1255 |
|                   |         | 50 | 0.0427| 0.0543   | 0.0702     | 0.0562  | 0.0627| 0.0641 | 0.0724| 0.0814 |
| Alibaba Dataset V2| Recall  | 10 | 0.0370| 0.0465   | 0.0622     | 0.0344  | 0.0639| 0.0619 | 0.0649| 0.0667 |
|                   |         | 50 | 0.0997| 0.0128   | 0.1655     | 0.1069  | 0.1664| 0.1568 | 0.1626| 0.1764 |
|                   | MAP     | 10 | 0.0223| 0.0273   | 0.0396     | 0.0186  | 0.0415| 0.0413 | 0.0425| 0.0431 |
|                   |         | 50 | 0.0233| 0.0303   | 0.0441     | 0.0218  | 0.0451| 0.0441 | 0.0454| 0.0470 |
|                   | NDCG    | 10 | 0.0536| 0.0648   | 0.0881     | 0.0470  | 0.0913| 0.0897 | 0.0930| 0.0947 |
|                   |         | 50 | 0.0323| 0.0405   | 0.0537     | 0.0320  | 0.0543| 0.0521 | 0.0540| 0.0569 |

Figure 5: Performance comparison on four public datasets with different k. Only one hypercuboid is used in our model.

sequences in Table 4, Firstly, we observe that removing the neural architectures will decrease the performance, especially on the Amazon Books dataset and company dataset V1/V2, which confirms the importance of the neural architectures. Next, we note that, even without the neural architectures, the pure hypercuboid architecture still get higher performance than the second best baselines (on Books, E-commerce, and Company Dataset V1), which signifies that the hypercuboid is essential for the improvement.

3.8 Sensitivity of Hyper-parameters

We conduct hyper-parameters analysis on three public datasets.
Table 3: Performance (Measure@10) of multiple hypercuboids.

|        | Number of Hypercuboids | Concentric Hypercuboids | Independent Hypercuboid |
|--------|-------------------------|--------------------------|--------------------------|
|        | Recall@10 | NDCG@10 | MAP@10 | Recall@10 | NDCG@10 | MAP@10 |
| E-commerce | 2         | 0.1396 | 0.1300 | 0.0802 | 0.1476 | 0.1336 | 0.0830 |
|         | 3         | 0.1512 | 0.1402 | 0.0876 | 0.1501 | 0.1356 | 0.0846 |
|         | 4         | 0.1529 | 0.1415 | 0.0885 | 0.1525 | 0.1374 | 0.0859 |
|         | 5         | 0.1508 | 0.1392 | 0.0868 | 0.1529 | 0.1381 | 0.0866 |
|         | 6         | 0.1436 | 0.1334 | 0.0827 | 0.1530 | 0.1381 | 0.0866 |
| Alibaba Dataset V2 | 2         | 0.0722 | 0.1032 | 0.0480 | 0.0766 | 0.1119 | 0.0534 |
|         | 3         | 0.0746 | 0.1072 | 0.0504 | 0.0774 | 0.1130 | 0.0540 |
|         | 4         | 0.0760 | 0.1095 | 0.0519 | 0.0779 | 0.1136 | 0.0544 |
|         | 5         | 0.0676 | 0.0966 | 0.0453 | 0.0781 | 0.1138 | 0.0546 |
|         | 6         | 0.0763 | 0.1100 | 0.0522 | 0.0785 | 0.1148 | 0.0552 |

Table 4: Ablation analysis of the neural architectures on three datasets. \( \Delta_{\text{Default}} \) and \( \Delta_{\text{2nd best}} \) are the relative improvement(%) over the default version (in Table 2) and the second best baseline, respectively.

|        | Measure@10 | Remove NNs | \( \Delta_{\text{Default}} \) | \( \Delta_{\text{2nd best}} \) |
|--------|------------|------------|-------------------------------|-------------------------------|
| Books  | Recall@10  | 0.0558     | -6.84%                        | +16.25%                       |
|        | NDCG@10    | 0.0360     | -6.01%                        | +1.41%                        |
|        | MAP@10     | 0.0264     | -8.97%                        | +10.00%                       |
| E-commerce | Recall@10 | 0.1384     | -0.75%                        | +4.86%                        |
|        | NDCG@10    | 0.1277     | -1.00%                        | +10.10%                       |
|        | MAP@10     | 0.0782     | -0.84%                        | +11.48%                       |
| Alibaba Dataset V1 | Recall@10 | 0.0281     | -6.02%                        | +3.54%                        |
|        | NDCG@10    | 0.1174     | -6.45%                        | +2.94%                        |
|        | MAP@10     | 0.0563     | -7.70%                        | +0.90%                        |
| Alibaba Dataset V2 | Recall@10 | 0.0644     | -3.45%                        | -0.77%                        |
|        | NDCG@10    | 0.0909     | -4.01%                        | -2.26%                        |
|        | MAP@10     | 0.0414     | -3.94%                        | -2.59%                        |

3.8.1 Embedding Size. Here, we analyze the impact of the embedding size in Figure 6 (a), (b), and (c). It is clear that increasing the embedding size improves the performance. Unlike models such as Caser, SASRec, and HGN (checking the embedding size analysis in the corresponding papers.), our model is less likely to be over-fitting with larger dimension size. It suggests that setting a dimension size no less than 10 to be a reasonable choice for our model.

3.8.2 Sequence length. By varying the sequence length \( L \), we get Figure 6 (1), (2), and (3). We observe that the model gets better performance with shorter sequence length on CDs and Movies&TVs, while it is more stable on the Books datasets. Overall, it is more preferable to set \( L \) to a value around 5 as longer sequence might introduce too many irrelevant activities in the sequences.

3.8.3 Pooling Function. Table 5 shows the results with different pooling functions. We find that all the three choices (sum, min, and max pooling) lead to deprovements over the mean pooling on the three datasets. In addition, there is no clear winner between the three pooling functions.

3.9 Visualization of User Hypercuboids and Item Embeddings

In order to check what the proposed model is learning, we visualize the user hypercuboids on the Amazon books dataset. We project 3,000 items and 3,000 user centers into a 3-dimensional space with T-SNE [21] (Perplexity 9, learning rate 10). Then, we randomly choose eight users and, for each of the eight users, we sample 100 points in the hypercuboid uniformly and project them into the same space. Figure 7 shows the learned hypercuboids. Clearly, our model successfully learned a hypercuboid with different sizes and
Table 5: Effect of pooling functions on three datasets

|        | Measure@10 | Sum | Min | Max |
|--------|------------|-----|-----|-----|
| CDs    | Recall     | 0.0542 | 0.0529 | 0.0528 |
|        | NDCG       | 0.0282 | 0.0268 | 0.0267 |
|        | MAP        | 0.0251 | 0.0241 | 0.0235 |
| Books  | Recall     | 0.0546 | 0.0574 | 0.0570 |
|        | NDCG       | 0.0382 | 0.0370 | 0.0370 |
|        | MAP        | 0.0272 | 0.0275 | 0.0274 |
| MovisTVs| Recall   | 0.0369 | 0.0377 | 0.0370 |
|        | NDCG       | 0.0212 | 0.0209 | 0.0205 |
|        | MAP        | 0.0166 | 0.0165 | 0.0162 |

Figure 7: Visualization of the user hypercuboids on Books. We randomly show the hypercuboids of eight users. Blue points represent items and red points represent the user centers and the orange points are randomly sampled points in the user hypercuboids. We manually circle each hypercuboid with an irregular polygon.

positions for each user. Items may be located inside or outside the user hypercuboid. Items which are inside the user hypercuboid are more likely to be recommended to this user. Intersections between users’ hypercuboids can also be observed.

Moreover, our model can also learn effective item representations. To demonstrate this, we project the item embeddings we learned from the company dataset (V1) into 2-D space. As shown in Figure 8, we observe that items from different categories (e.g., snack, clothes, toy, furniture, etc.) are roughly separated. Related categories tend to be closer to each other (e.g., bedding and furniture).

3.10 Case Study with Multiple Hypercuboids

To demonstrate the efficacy of using multiple hypercuboids, we show the recommendations list for a randomly selected user in Figure 9. In this case, we learn three independent hypercuboids for this user. Several observations can be made: (1). The recalled items are strongly correlated with the corresponding user sequence; (2) Different hypercuboid can capture different aspect of the interests in the historical sequences, i.e, hypercuboid one recalls T-shirts, hypercuboid two recalls sport footwear; (3). A single hypercuboid can also recall a diverse set of items from closely related categories.

Figure 8: Visualization of Item Embeddings on the Alibaba dataset.

For example, hypercuboid three recalls both socks and basketballs. Clearly, our model is suitable for capturing user’s diverse interests.

4 RELATED WORK

Recommender systems have been widely studied in the literature [1, 16, 17, 27, 30, 32, 40]. Conventional recommendation methods usually represent users and items as points/vectors in a latent space. The interactions are modelled with vectors similarity [13, 17, 49] or points closeness [12, 39]. In the recent years, we can easily observe the increasing numbers of neural network models for recommendation [46, 48]. A number of works [2, 9, 43, 44] proposed to model the interactions with multilayered perceptrons. There is also an emerging line of works focusing on capturing the sequence patterns in user activities to enrich user representations [4, 24, 33]. Hidasi et al. [10, 11] proposed a recurrent neural network based model for session-based recommendation. Convolutional neural networks based sequence modeling approaches show promising results on sequential recommendation [34, 45]. Self-attention mechanism [14, 15, 50] can also utilized to enhance the performance by emphasizing important activities in user behaviors. The recent model, hierarchical gating networks [20], moving beyond RNN, CNN, and self-attention, is a gating network that consists of a feature gating module, an instance gating module and an item-item product module. Different from previous works, our
model takes the advantages of hypercuboid representations to overcome the limitations of traditional user representation methods. A customized sequence modeling block is also designed to boost the effectiveness of the hypercuboid by incorporating information from user sequential behaviors.

Hypercuboid is an extension of cuboid in higher dimensions. It has attracted attention in the field of machine learning recently [22, 25, 41, 47]. Maji et al. [22] proposed a rough hypercuboid method for relevant and significant features selection. Vilins et al. [38] proposed a probabilistic embedding method with hypercuboid lattice Measures for knowledge graphs. In the followed-up work, Li et al. [18] improved this model to enhance the robustness in the disjoint case while keeping the desirable properties. Ren et al. [25] proposed a hypercuboid representation approaches for reasoning over knowledge graphs. Our work is also related to Venn Diagrams [37] in the sense that we can treat users as a set of entities they preferred.

5 CONCLUSION

In this paper, we propose a novel recommendation method with hypercuboids. In our model, users are represented as hypercuboids and the relationship between users and items are modeled with the distance between points and hypercuboids. It demonstrates the state-of-the-art performance on four publicly available and two commercial datasets for personalized recommendation. To capture the diversity of user interests, we proposed two variants of hypercuboids which could further improve recalling performance. We also studied the impacts of effects of hyperparameters and provided case analysis on real-world datasets.

REFERENCES

[1] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE transactions on knowledge and data engineering 17, 6 (2005), 734–749.

[2] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for YouTube recommendations. In Proceedings of the 10th ACM conference on recommender systems. 191–194.

[3] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research 12, Jul (2011), 2121–2159.

[4] Shanshan Feng, Xutao Li, Yifeng Zeng, Gao Cong, Yewon Meng Chee, and Qian Yuan. 2015. Personalized ranking metric embedding for next new pop recommendation. In Twenty-Fourth International Joint Conference on Artificial Intelligence.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep learning. MIT press.

[6] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based recommendation. In Proceedings of the Eleventh ACM Conference on Recommender Systems. 161–169.

[7] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-Based Recommendation. In Proceedings of the Eleventh ACM Conference on Recommender Systems (Como, Italy) (RecSys ’17). Association for Computing Machinery, New York, NY, USA, 161–169. https://doi.org/10.1145/3109559.3109882

[8] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In Proceedings of the 25th international conference on world wide web. 507–517.

[9] Xiangnan He, Rui Liao, Hanwang Zhang, Linqing Nie, Xiao Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In Proceedings of the 26th international conference on world wide web. 173–182.

[10] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Dominos Tikk. 2015. Session-based recommendations with recurrent neural networks. arXiv preprint arXiv:1511.06999 (2015).

[11] Balázs Hidasi, Massimo Quadrana, Alexandros Karatzoglou, and Dominos Tikk. 2016. Parallel recurrent neural network architectures for feature-rich session-based recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems. 241–248.
[38] Luke Vilnis, Xiang Li, Shikhar Murty, and Andrew McCallum. 2018. Probabilistic Embedding of Knowledge Graphs with Box Lattice Measures. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, Melbourne, Australia, 263–272. https://doi.org/10.18653/v1/P18-1025

[39] Lucas Vinh Tran, Yi Tay, Shuai Zhang, Gao Cong, and Xiaoli Li. 2020. HyperML: A Boosting Metric Learning Approach in Hyperbolic Space for Recommender Systems. In Proceedings of the 13th International Conference on Web Search and Data Mining. 609–617.

[40] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative Deep Learning for Recommender Systems. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (Sydney, NSW, Australia) (KDD ’15). Association for Computing Machinery, New York, NY, USA, 1235–1244. https://doi.org/10.1145/2783258.2783273

[41] J. Wei, S. Wang, and X. Yuan. 2010. Ensemble Rough Hypercuboid Approach for Classifying Cancers. IEEE Transactions on Knowledge and Data Engineering 22, 3 (March 2010), 381–391. https://doi.org/10.1109/TKDE.2009.114

[42] 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. ACM, 65–68.

[43] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In Proceedings of the Ninth ACM International Conference on Web Search and Data Mining. 153–162.

[44] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. 2017. Deep Matrix Factorization Models for Recommender Systems. In IJCAI. 3293–3299.

[45] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiaoliang He. 2019. A simple convolutional generative network for next item recommendation. In Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining. 582–590.

[46] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. 2019. Dive into Deep Learning. Unpublished draft. Retrieved 3 (2019), 319.

[47] Hao Zhang and Lynne F Parker. 2011. 4-dimensional local spatio-temporal features for human activity recognition. In 2011 IEEE-RSJ international conference on intelligent robots and systems. IEEE, 2044–2049.

[48] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep Learning Based Recommender System: A Survey and New Perspectives. ACM Comput. Surv. 52, 1, Article 5 (Feb. 2019), 38 pages. https://doi.org/10.1145/3285029

[49] Shuai Zhang, Lina Yao, Lucas Vinh Tran, Aston Zhang, and Yi Tay. 2019. Quaternion collaborative filtering for recommendation. In Proceedings of the 28th International Joint Conference on Artificial Intelligence. AAAI Press, 4313–4319.

[50] Chang Zhou, Jinze Bai, Junshuai Song, Xiaofei Liu, Zhengchao Zhao, Xiuxi Chen, and Jun Gao. 2018. Atrank: An attention-based user behavior modeling framework for recommendation. In Thirty-Second AAAI Conference on Artificial Intelligence.