Hyperbolic Hypergraphs for Sequential Recommendation

Yicong Li
University of Technology Sydney
Australia
Yicong.Li@student.uts.edu.au

Hongxu Chen†
University of Technology Sydney
Australia
Hongxu.Chen@uts.edu.au

Xiangguo Sun
Southeast University
China
sunxiangguo@seu.edu.cn

Zhenchao Sun
Shandong University
China
zhenchao.sun@mail.sdu.edu.cn

Lin Li
Wuhan University of Technology
China
cathylin@whut.edu.cn

Lizhen Cui
Shandong University
China
clz@sdu.edu.cn

Philip S. Yu
University of Illinois at Chicago
the United States
psyu@cs.uic.edu

Guandong Xu†
University of Technology Sydney
Australia
Guandong.Xu@uts.edu.au

ABSTRACT
Hypergraphs have been becoming a popular choice to model complex, non-pairwise, and higher-order interactions for recommender system. However, compared with traditional graph-based methods, the constructed hypergraphs are usually much sparser, which leads to a dilemma when balancing the benefits of hypergraphs and the modelling difficulty. Moreover, existing sequential hypergraph recommendation overlooks the temporal modelling among user relationships, which neglects rich social signal from the recommendation data. To tackle the above shortcomings of the existing hypergraph-based sequential recommendations, we propose a novel architecture named Hyperbolic Hypergraph representation learning method for Sequential Recommendation (H²SeqRec) with pre-training phase. Specifically, we design three self-supervised tasks to obtain the pre-training item embeddings to feed or fuse into the following recommendation architecture (with two ways to use the pre-trained embeddings). In the recommendation phase, we learn multi-scale item embeddings via a hierarchical structure to capture multiple time-span information. To alleviate the negative impact of sparse hypergraphs, we utilize a hyperbolic space-based hypergraph convolutional neural network to learn the dynamic item embeddings. Also, we design an item enhancement module to capture dynamic social information at each timestamp to improve effectiveness. Extensive experiments are conducted on two real-world datasets to prove the effectiveness and high performance of the model.

†Having equal contribution with the first author.
Corresponsing authors.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:
Yicong Li, Hongxu Chen, Xiangguo Sun, Zhenchao Sun, Lin Li, Lizhen Cui, Philip S. Yu, and Guandong Xu. 2021. Hyperbolic Hypergraphs for Sequential Recommendation. In Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM ’21), November 1–5, 2021, Virtual Event, QLD, Australia. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482351

1 INTRODUCTION
Graph-based approaches have been widely used and achieved great improvement for next-item recommender systems. However, most existing literature [2, 12, 15, 18, 23, 29, 32, 35, 45] treat the dynamic time-dependent user-item interactions as a temporal bipartite graph and learn their latent representations for action. Though the graph-based graph modelling can capture the beneficial first-order (i.e., user-item interactions) and second-order (i.e., co-purchasing) interactions for recommendation, the higher-order signals among users and items are usually neglected by existing works due to the limitations of traditional graph modelling. With noting the shortcoming, recent works [4, 24, 25, 29, 30, 36, 43] resorted hypergraphs to make up and developed hypergraph-based modelling approaches for sequential recommender systems. The basic idea is illustrated in Figure 1, when using traditional graphs to model user-item dynamic relationships evolution, the learned information from Figure 1.(a) to 1.(b) is monotonous due to the simple pair-wise data structure. In contrast, hypergraphs are able to capture high-order dynamic relationships (buy items at the same time, similar user groups, etc.) thanks to the non-pairwise data structure. That is, unlike traditional graphs, an edge in the hypergraph (a.k.a hyperedge) is allowed to connect more than two nodes, promising to capture multi-scale signals for the recommendation.

Nevertheless, all these existing works ignored a critical issue of such hypergraph-based approaches, which is the sparsity issue is becoming more severe in hypergraph based recommender systems. As analysed in [42], the recommendation benchmark...
Amazon dataset is sparse and exhibits long-tailed distribution, in which many users have limited interactions to the items, while only a small number of users interact with many items. In such a case, the constructed hypergraphs from the original dataset will be much sparser, resulting in insufficient training samples for action. To be specific, if we construct hypergraphs based on the original simple user-item bipartite graph, the number of items (nodes) does not change, but the number of hyperedges is dramatically shrunken compared to the number of original links between users and items, which is obvious in Figure 1. Therefore, when constructing hypergraphs, the challenging sparsity issue and long-tailed data distribution are becoming even more severe.

The second limitation of existing hypergraph-based sequential recommendation lies in lacking exploitation of hidden hyperedges among users, which we believe will be beneficial to understand the hidden but insightful behaviours among users (e.g., common interests groups, co-purchasing groups, users who have similar buying patterns, etc.). Taking the group recommendation task [31, 41] as an example, it focuses on a group of users’ preference, which means users in the same group may tend to have a similar preference, or at least they have more common interest compared to the rest of the world. Inspired from the idea, we are curious about the possibility of exploring and leveraging hypergraphs constructed from users side to improve the overall recommendation performance. For example, in Figure 1(c) and (d), from the timestamp $t_0$ to $t_1$, they are the evolution of sequential hypergraph constructions. It is obvious that in the timestamp $t_0$, the user $u_1$ is of co-purchasing relationship with $u_2$, and the hyperedges of $u_1$ and $u_2$ are connected. In this case, the items $i_1$, $i_3$, $i_4$ and $i_5$ are likely to be more similar to each other than to $i_2$. The same is also in the timestamp $t_1$. Therefore, the aim is to capture the dynamic items diversification through dynamic hypergraphs constructions according to users’ relationships to enhance the model. Intuitively, like traditional graph-based methods, this try may gain improvement on recommendation.

To holistically solve the above issues, we propose a novel architecture named Hyperbolic Hypergraph representation learning for Sequential Recommendation (H$^2$SeqRec). Specifically, we propose to firstly pre-train the model with self-supervised learning on three well-sophisticated tasks. It is worth noting that compared with the prior works paying more attention to chronologically model each user’s buying history, one of our contributions is to introduce a hyperedge prediction task, which explicitly models the users’ historical records as hyperedges at each timestamp and explores the potential relationships between linked hyperedges (users who have similar buying history) [37, 48]. Besides, we also investigate hyperbolic embedding spaces [46] and manage to map the sparse data points to the hyperboloid manifold directly. The rationale is that hyperbolic space has a stronger ability than Euclidean space to accommodate networks with long-tailed distributions and sparse structures [1, 3, 13, 14, 21], which is also verified in our experiments.

In addition, to exploit hidden but useful user-side information in sequential recommendation settings, we propose to construct an induced hypergraph for each user to model his behaviour pattern in group-level. Intuitively, a user’s behaviour pattern is not only reflected in his historical records but can also be excavated from other users who have similar behaviour pattern. This is particularly helpful to remit the “cold start” problem. Specifically, for the target user $u$, we first use a hyperedge to collect historical items taken by $u$ at the timestamp $t$. Then we find the other hyperedges from other users who share the overlapping items with $u$ at the timestamp $t$. All these hyperedges construct an induced hypergraph to describe the user $u$’s behaviour pattern from the group view at the timestamp $t$.

Although constructing hypergraphs from original simple user-item bipartite graphs seems a better choice, it is difficult to decide the time granularity to which the length of item sequence for each hyperedge is split. After analysing the items in the dataset, we find the purchase history is not only in a chronological sequence way, but with a periodic regularity, like seasonal evolution. As shown in Figure 2, the user bought a Christmas tree and some decorations in both 2018 and 2019, and she also bought two dresses in the summer of 2018 and 2019. Therefore, rather than model each user’s items in a chronological way, we design three views for hypergraph construction, namely yearly, quarterly and monthly views, and manage to hierarchically learn latent item representations. Moreover, we also find that when constructing quarterly and yearly hypergraphs, the hypergraphs are no longer as sparse as original single-time grained hypergraphs. In this way, the hierarchical time-span method for hypergraph construction also alleviates the above mentioned sparse problem.

In summary, the contributions of this work are as follows.

- We propose three self-supervised learning tasks as the pre-training phase. To our best knowledge, we are the first to
The section presents the definition and the problem formulation.

**Problem Formulation**

**Definition 1. Hypergraph.** Assuming $G^H = (V, E^H)$ denotes a hypergraph with $n$ nodes set $V$ and a hyperedges set $E^H$, a hyperedge $e^H \in E^H$ connects multiple $n$ nodes ($n \geq 2$). The nodes set connected by a hyperedge $e^H$ is a hypernode $v^H \subset V$.

**Problem 1. Sequential Recommendation.** In the sequential recommendation task, the user’s related items are in chronological order. Given the user set $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ and the item set $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$, each user has sequential items $S_{u_1} = \{s_{u_1}^1, s_{u_1}^2, \ldots, s_{u_1}^{t_u}\}$, where $s_{i}^{t_u} \in I$ denotes the $t^{th}$ historical item of user $u$. The sequential recommendation problem is to predict the next $q$ items $\{s_{u_1}^{t_u+1}, \ldots, s_{u_1}^{t_u+q}\}$ associated to each user $u$, according to sequential historical items $S_u = \{S_{u_1}, S_{u_2}, \ldots, S_{u_n}\}$. In this paper, $q = 1$. For each timestamp $t$, a hypergraph $G^H_t$ is constructed according to the current user set $\mathcal{U}^t$ and the item set $I^t$. Each user’s historical items within the timestamp $t$ are connected by a hyperedge.

3 MODEL

Figure 3 is an overview of our proposed sequential recommendation system. Technically, we first generate yearly, quarterly, and monthly hypergraph snapshots as views from users’ historical records. Here each hyperedge in the hypergraph refers to one user’s historical items. With these hypergraphs, we design a novel hypergraph convolutional network in the hyperbolic space to learn multi-scale item embeddings. To enhance the item embedding, we further explore user’s behaviour pattern via similar users’ behaviours. In the end, we leverage the Transformer to learn users’ preference and predict the next item via multi-layer perception (MLP). Moreover, before the $H^2$SeqRec, we design three self-supervised tasks, as shown in Figure 3, to pre-train item representations. The pre-trained item representations could be as initial embeddings for $H^2$SeqRec, named $H^2$SeqRec-init; and they could also be fused into $H^2$SeqRec, named $H^2$SeqRec-fuse.

3.1 Item Features Extraction via Self-supervised Learning

Before we start, we need the initial item features to support the downstream model. Inspired by the self-supervised learning framework [48], which can learn item representations without annotations, we propose three novel pre-training tasks (as shown in Figure 4) to learn the pre-trained item representations:

- The first task is to predict whether a masked item sequence is derived from a user’s historical records. The rationale is that in the real world, one item usually cannot change a user’s long-term preference. Therefore, we randomly mask different items for a target user’s historical sequence and then let these sequences’ representations as close as possible.
- The second task is to predict whether a sub-sequence is consistent with a user’s short-term interest. The motivation is that the consecutive items taken in a short time usually contain a user’s short-term interest, and these items might share similar functions. If we mask part of them, the user’s short-term interest should be still observable from the rest items. Therefore, we first mask a sub-sequence with length 2 and then wish the masked sequence be close to the complete sequence.
- The third task is to predict whether two users have the same behaviour patterns. Intuitively, users’ behaviour patterns can be reflected in their historical records. If two users share many historical items, they are more likely to have the same preference, hobbies or even tastes. To this end, we use a hyperedge to connect one user’s historical items and then let a pair of hyperedges be as close as possible if they share the overlapping items.

Let $h_{u}$ and $h_{v}$ being embeddings of two sequences, we adopt contrastive loss [37] for each task:

$$
\mathcal{L} \{u, v, S_u\} = - \log \frac{\exp \left(\frac{\text{sim}(h_u, h_v)}{\tau}\right)}{\exp \left(\frac{\text{sim}(h_u, h_{S_u})}{\tau}\right) + \Sigma_{S_{v}} \exp \left(\frac{\text{sim}(h_v, h_{S_v})}{\tau}\right)}
$$

(1)

here $S_{u}$ is the sampled negative sequences of the user $u$. $h_{S_u}$ is the embedding of the negative sequence, $\text{sim}(h_{u}, h_{S_u})$ is defined
With the initial features, we then design a dynamic hypergraph where each hyperedge is constructed monthly or yearly to learn the semantics of items. To tackle the sparsity problem, we use multi-scale time-spans views (monthly, quarterly, and yearly) to learn the semantics of items. Inspired by [29], we utilize hypergraphs instead of traditional graphs to model different sub-sequences of each user's behavior patterns. The motivation is that a user's behavior patterns are not just suggested in anteroposterior periodical variance. For example, users prefer to buy Christmas presents but also reflect via the seasonal and another hyperedge to connect the flower and the wedding dress, and use another hyperedge to connect the flower and the vase. In this way, the flower's semantic for user u1 is about the wedding but for user u2 is about decoration. However, if we use a traditional graph, the graph is hard to reveal different item semantics directly.

In addition, the user-item record usually follows the long-tail distribution and is sparse. The traditional Euclidean space usually cannot capture this structure leading to representation distortion, while hyperbolic space is beneficial to deal with the issue. [1]

Based on the above motivations, we propose a hypergraph neural network to model the items evolution in hyperbolic space. Specifically, we first transform the initial item features from Euclidean space to hyperbolic space, as shown in Figure 4.

Figure 5: Multi-scale Embeddings via Dynamic Hierarchical Hyperbolic Graphs.

as: $\frac{h_{su, \tau} - h_{su}}{||h_{su, \tau}|| \cdot ||h_{su}||}$ where $\tau$ is the hyper-parameter. The sequence embeddings $h_{su, \tau}$ and $h_{su}$ are both calculated by Transformer network with two multi-head attention and feed forward blocks. We combine the above three tasks together with optimized weights $[0.1, 1, 1]$.

3.2 Multi-scale embeddings via dynamic hierarchical hyperbolic graphs

With the above initial features, we then design a dynamic hypergraph neural network for learning the multi-scale item embeddings in hyperbolic space, as shown in Figure 5.

3.2.1 Hierarchical Time-span Hypergraph Construction. To model the complex dependencies among the sequential items, we propose a hierarchical architecture to learn the monthly, quarterly and yearly relationship among sequential items. The motivation is that a user's behavior patterns are not just suggested in anteroposterior periodical variance. For example, users prefer to buy Christmas-related products before Christmas, and purchase T-shirts in summer and coat in winter. Inspired by [29], we use hypergraphs instead of traditional graphs to model different sub-sequences of each user and we also use multi-scale time-spans views (monthly, quarterly, and yearly) to learn the semantics of items. To tackle the sparsity and the long-tail distribution of these hypergraphs, we learn the graph representations in hyperbolic space because it is perfectly suitable for long-tail structures.

Specifically, when we deal with the monthly item representations, we first construct a monthly hypergraph where each hyperedge connects user’s items within one month. Then we use hyperbolic space-based hypergraph graph neural network (introduced in 3.2.2) to learn the dynamic item embeddings in each month. Following this approach, we can also learn quarterly and yearly item representations.

3.2.2 Hyperbolic Space-based Hypergraph Convolutional Network. Compared with traditional graphs that mainly rely on pairwise user-item interactions, hypergraphs can model much higher relations in user-item interactions and promise to fuse item context to remit the sparsity problem. For example, if user u1 bought flower and wedding dress and u2 bought flower and a vase, we can use a hyperedge to connect the flower and the wedding dress, and use another hyperedge to connect the flower and the vase. In this way, the flower’s semantic for user u1 is about the wedding but for user u2 is about decoration. However, if we use a traditional graph, the graph is hard to reveal different item semantics directly.

In addition, the user-item record usually follows the long-tail distribution and is sparse. The traditional Euclidean space usually cannot capture this structure leading to representation distortion, while hyperbolic space is beneficial to deal with the issue. [1]

Based on the above motivations, we propose a hypergraph neural network to model the items evolution in hyperbolic space. Specifically, we first transform the initial item features from Euclidean space to hyperbolic space $\mathbb{H}^c$, and then we feed the initial hypergraph item embeddings to learn item embeddings. For the hyperbolic space, we set $c = \{ \sqrt{c}, 0, 0, ... , 0 \} \in \mathbb{H}^c$ as the north pole in $\mathbb{H}^c$, where $-1/c$ is the negative curvature of hyperbolic model. As analyzed in [1], the initial item features in hyperbolic space can be deduced from Euclidean space as follows:

$$h_{i,0}^{E} = \exp_{c}^{E}\left(0, h_{i,0}^{E}\right)$$

$$= \left(\sqrt{c} \cosh\left(\frac{||h_{i,0}^{E}\|_{2}}{\sqrt{c}}\right), \sqrt{c} \sinh\left(\frac{||h_{i,0}^{E}\|_{2}}{\sqrt{c}}\right), h_{i,0}^{E}\right)$$

(2)

where $h_{i,0}^{E}$ and $h_{i,0}^{E}$ are the initial hyperbolic embedding and the initial Euclidean embedding, respectively.

With the initial features in hyperbolic space, we further transform the features via a linear function in hyperbolic space:

$$x_{i,L}^{H} = (W_L \odot^{c} h_{i,L-1}^{H}) \odot^{c} b_L$$

(3)

where $x_{i,L}^{H}$ is the hyperbolic hidden embedding of item $i$ in the $L$-th layer after transformation, $W_L$ and $b_L$ are the weight and bias, respectively. $h_{i,L-1}^{H}$ is item $i$'s hyperbolic embedding in the last layer. When $L = 1$, $h_{i,L-1}^{H} = h_{i}^{H}$. Then, the item embeddings
can be aggregated from the neighbouring nodes via the following convolutional operation in hyperbolic space:

\[ y^{L,H}_{i} = \exp_{x^{L,H}_j}^{c}(\sum_{j \in N(i); j \in N(t^*)}, \text{where } i \in e^H_i) \]

where \( y^{L,H}_{i} \) is the hyperbolic hidden embedding of item \( i \) in the \( L \)-th layer after aggregation, \( N(.) \) denotes the neighbors so \( N(i) \) is node \( i \)’s neighbors and \( N(t^*) \) is node \( t^* \)’s neighbors where \( t^* \) is on the same hyperedge as \( i \). The node \( j \)’s hyperbolic embedding is transformed to Euclidean embedding via \( \log_{c}(.) \), so the Euclidean-based sum and add operations are available. \( \exp^{c}(.) \) aims to transform the Euclidean-based embedding to hyperbolic embedding. According to [1], choosing \( x^{L,H}_i \) as the north pole is the best Euclidean approximation at this step. \( M_{ij} \) is the projecting weight defined as:

\[ M_{ij} = \operatorname{softmax}_{j \in N(i); j \in N(t^*)}, \text{where } i \in e^H_i \]

(5)

With the above node features’ aggregation, we then use an activation function to generate the hidden embedding on Layer \( L \):

\[ h^{L,H}_{i} = \text{ReLU}^e(y^{L,H}_{i}) \]

(6)

where \( h^{L,H}_{i} \) is the node \( i \)’s hyperbolic embedding at \( L \)-th layer, \( \text{ReLU}^e(.) \) is the \( \text{ReLU} \) operation in the hyperbolic space. Through \( \log_{c}(.) \), we transform the hyperbolic item embedding \( h^{L,H}_{i} \) to Euclidean-space based embedding \( h^{L}_{i} \) and feed into the next module.

3.2.3 Mix Layer. With item embeddings for each month, quarter and year, we then design a mixer layer to fuse them together. Specifically, to fuse monthly embeddings and yearly embeddings, we use a two-layer neural network to calculate their correlations as follows:

\[ h_{MY_{tm}} = \text{ReLU}(W_{MY}h_{Y_{tm}} + b_{MY}) \odot h_{M_{tm}} \]

(7)

where \( h_{MY_{tm}} \) represents the mixture of monthly and yearly item embeddings at timestamp \( t_m \). \( h_{M_{tm}} \) and \( h_{Y_{tm}} \) mean the monthly and yearly item embeddings respectively. \( t_m \) and \( t_y \) denote the monthly and yearly timestamp, respectively. \( W_{MY} \) and \( b_{MY} \) are weight matrix and bias. \( \odot \) means element-wise product. Similarly, the mixed embeddings with the month and the quarter can be defined as \( h_{MQ_{tm}} \). which is calculated by:

\[ h_{MQ_{tm}} = \text{ReLU}(W_{MQ}h_{Q_{tm}} + b_{MQ}) \odot h_{MY_{tm}} \]

(8)

where \( h_{MQ_{tm}} \) is the dynamic item embeddings at timestamp \( t_m \) learned by this hierarchical architecture.

3.3 Item Enhancement via Similar User Groups

Most existing hypergraph-based sequential recommender systems [29] ignore the hidden relationships among users, like social relationships, co-purchasing relationship, etc. However, these relationships are highly informative to capture the hidden users’ behaviours. For example, a user’s shopping behaviour may be affected by his friend circle, making the user tend to buy similar products with his friends. In light of this, we utilize user groups to enhance the former learned dynamic item embeddings.

Specifically, we first find similar users for each user at each timestamp \( t_m \). The similar users are those who take the same products at the same timestamp \( t_m \). Using the current user’s shopping records at \( t_m \) and his similar users’ records at \( t_m \), we build an induced hypergraph to model his behaviour pattern in group-level. Each hyperedge connects each user’s items. Then we leverage our proposed hyperbolic space-based hypergraph convolutional network to learn item embeddings in group-level.

To enhance the item representations, we mix the above embeddings \( h_{i,m,\text{User}} \) with previous embeddings \( h_{i,m,\text{Hie}} \) in Section 3.2.1. The mix operation is defined as follows:

\[ h_{i,m,\text{User}} = \text{ReLU}(W_{i,m,\text{User}} + b) \odot h_{i,m,\text{Hie}} \]

(9)

where \( h_{i,m,\text{User}} \) is the item \( i \)’s embedding for user \( u \) at timestamp \( t_m \).

3.4 Learning User Preference for Sequential Recommendation

In this section, we leverage learned item embeddings to find users’ preference and then present the sequential recommendation model.

3.4.1 User Preference Learning. For each user \( u \), we use the Transformer [26] to model the dynamic item embeddings during the whole timestamps, mathematically,

\[ h_{S_{u}} = \text{Transformer}(h_{i,m,\text{User}}), \text{where } i \in S_{u} \]

(10)

where \( h_{S_{u}} \) is the embedding of user \( u \)’s sequence, which can describe \( u \)’s preference. The input of Transformer are the dynamic item embeddings learned in Equation 9. In this way, the user’s sequential items contain both the dynamic hierarchical information and the user’s potential group information.

3.4.2 The Complete Model. With user preference representation \( h_{S_{u}} \), we use a two-layer Multilayer Perceptron (MLP) to calculate the rating score of the next item, mathematically:

\[ r_{ui,t}^{m} = \text{MLP}(h_{S_{u}}; h_{i,m}^{H}) \]

(11)

where \( r_{ui,t}^{m} \) is the rating score of user \( u \) and item \( i \) at timestamp \( t_m \), \( h_{i,m}^{H} \) is the embedding of item \( i \) at timestamp \( t_m \). We train the recommendation by Bayesian Pairwise Loss [20], which aims to maximize the difference between the rating scores of the positive item \( i \) and negative sample \( j \):

\[ \text{loss} = \sum_{(u,i,t,j) \in D} -\ln \sigma(r_{ui,t}^{m}, r_{uj,t}^{m}) + \alpha ||\delta||^2 \]

(12)

where \( (u,i,t,j) \in D \) denotes the positive pair \( (u,i,t) \) from the training set \( D \). \( \sigma \) is the sigmoid function, \( \alpha \) is the weight of L2 regularization term \( ||\delta||^2 \).

4 EXPERIMENT

4.1 Experiment Settings

4.1.1 Dataset. We evaluate our method on two real-world datasets, named AMT and Goodreads. The statistics are shown in Table 1.

- AMT. It is the subset of the public Amazon dataset [17], which contains consumers’ buying record and reviews in 29 categories. In our experiment, we choose three categories, Automotive, Musical Instruments and Toys and Games, to form the dataset and remove users bought less than 5 items.

| Dataset | No. Users | No. Items | No. Interactions |
|---------|-----------|-----------|-----------------|
| AMT     | 5352      | 11672     | 59495           |
| Goodreads | 16701  | 20823     | 1084781         |

Table 1: Datasets Description.
The time of AMT dataset is range from 2014 to 2018. The first two items in 2018 are considered as valid data and test data, respectively. Generally, the distribution of the items among users is long-tail distribution [42].

- **Goodreads.** It [27, 28] is collected from goodreads website, which is an online book community website. We choose user-book (item) interactions between 2013 and 2015 as the dataset. Last two interactions are valid and test data, respectively. From Table 1, it is evident that Goodreads is much denser than AMT dataset.

4.1.2 Baselines.

- **GRU4Rec.** [5, 6] It is a well-known sequential-based recommendation model that utilizes GRU to sequentially model users’ interactions to achieve top-N recommendation.
- **SASRec.** [12] It is a self-attention based sequential approach for next item recommendation, which could capture each user’s both long-term sequential relations through Point-Wise Feed-Forward Network and short-term interactions with items through an attention mechanism.
- **BERT4Rec.** [23] This method employs deep bidirectional self-attention to sequentially model the user behaviours in two directions through Cloze task.
- **SRGNN.** [35] The method utilizes graph neural networks to model the session sequences and obtain the complex item transitions for each session.
- **HGN.** [15] It proposes a hierarchical gating network with the Bayesian Personalized Ranking in order to capture user’s both long- and short-term interest.
- **HyperRec.** [29] It uses sequential hypergraphs to model dynamic item embedding sequentially and fuses with static item embeddings as item representations. For each user, item representations are fed into Transformer network to obtain the next item recommendation.

4.1.3 Evaluation. Our proposed method focuses on recommending next item, and therefore we use Top K Hit Ratio (HR@K) and Top K Normalized Discounted Cumulative Gain (NDCG@K) as our evaluation metrics. We choose $K = \{1, 5, 10, 20\}$ in the baseline comparison experiment. Our baseline HyperRec [29] randomly selects 100 negative samples for each positive user-item pair. However, we think it is insufficient to reflect our model’s effect accurately. Moreover, the baseline BertRec and SRGNN’s evaluation speed is too slow to test all the data. Therefore, in our experiment, we randomly choose $\{100, 500\}$ negative samples and rank $\{101, 501\}$ items to calculate the HR@K and NDCG@K scores.

4.1.4 Parameter Details. We implement GRU4Rec, SASRec, BERT4Rec and SRGNN’s from the RecBole python package [47]. For other methods, we use the public code provided by each paper. For all the methods, the feature dimension size is 100. For AMT dataset, all baselines’ training batch size is set 512. We use the early stop function in RecBole whose condition is not updating NDCG@10 for 10 epochs. The hidden size is 100, and the dropout probability is 0.5, as the same as our proposed methods. Moreover, we choose BPR as the loss of GRU4Rec, SASRec, BERT4Rec and SRGNN to keep identical with our methods. For those baselines’ code authors provided, we employ the given default parameters, but the learning rate is 0.001 for all baselines. For Goodreads dataset, some baselines’ training time is so long with training batch size 512, so we change it to 0.005 if the baseline does not run out of CUDA memory. Other parameters are the same as training AMT dataset.

To evaluate our proposed model\(^1\), we design three strategies to initialize it: 1) $H^2$SeqRec, one-hot item embeddings as input without pre-training tasks, 2) $H^2$SeqRec-init, pre-trained item embeddings as input and 3) $H^2$SeqRec-fuse, a combination of the first two, one-hot item embeddings as input, and pre-training item embeddings fused with dynamic item embeddings and group-level item embeddings.

For our proposed methods $H^2$SeqRec, $H^2$SeqRec-init and $H^2$SeqRec-fuse, the parameter settings are the same. The layer of hyperbolic hypergraph convolutional network is 2, and we choose the hyperboloid model as hyperbolic geometry with the negative curvature -1. Moreover, since the Goodreads dataset is too dense, we sample 1/100 users to find similar user groups. To simplify our proposed methods, we fuse the item embeddings learned via similar users’ group with the dynamic item embeddings within each batch rather than for each user. In Transformer, we set the number of heads and blocks, 2 and 1, respectively. The epoch of hyperbolic hypergraph convolutional network is 300. The maximum sequence is set to 50 for all the above methods. Other parameters are the same as the baselines’.

4.2 Effectiveness Analysis on Baselines

4.2.1 Overall Performance Analysis Between Different Baselines. As shown in Table 2, we evaluate our proposed model with 6 state-of-the-art sequential recommendation baselines. Our proposed model $H^2$SeqRec, $H^2$SeqRec-init and $H^2$SeqRec-fuse could outperform all of them in both HR@K and NDCG@K in 100 and 500 negative sampling experiments. For AMT dataset, our best model achieves the best baseline 31.61% and 16.94% at NDCG@1 for negative sampling 100 and 500, respectively. In terms of Goodreads dataset, our best model outperforms the best baseline 10.43% and 13.71% at NDCG@1 and better SeqRec achieves the best performance and improves 5.41% and 5.33% than the NDCG@1 of $H^2$SeqRec with 100 and 500 negative samples. While for the Goodreads dataset, $H^2$SeqRec achieves the best performance on NDCG@1 and better than $H^2$SeqRec-fuse by 2.74% and 7.74% in negative sampling of 100 and 500, respectively. From the last column (Improvement), the advantages also could see in other evaluation metrics, especially in top 1 and 5 ranking. In recommendation task, it is significant to have better recommendation in top ranking, because proper recommendation in higher ranking means more effectiveness of the recommendation model.

4.2.2 Effectiveness on Pre-train Features. In Table 2, for AMT dataset, the proposed $H^2$SeqRec-fuse achieves the best performance and improves 5.41% and 5.33% than the NDCG@1 of $H^2$SeqRec with 100 and 500 negative samples. While for the Goodreads dataset, $H^2$SeqRec achieves the best performance on NDCG@1 and better than $H^2$SeqRec-fuse by 2.74% and 7.74% in negative sampling of 100 and 500, respectively. From the results, we could get the conclusion that when training sparser dataset, to do data augmentation as pre-training could help improve model. However, in terms of denser dataset, it is useless or even worse to do pre-training while our proposed model without pre-training could still achieve quite well performance. Therefore, unlike traditional recommendation models which blindly pursue pre-training to improve the model,\(^1\)https://github.com/Abigale001/h2seqrec
We perform ablation test on AMT dataset to study different modules’ effects. The evaluation metric is NDCG@[1,5,10,20], and the number of negative test sampling is 100. In Table 3, the \( \cdot \) represents removing the following module and keeping the others. The last column means the decrease rate of each row compared with \( H^2\text{SeqRec} \) on NDCG@[10].

![Table 2: Comparison with baselines. The last column is the improvement of the best proposed method than the best baseline.](image)

![Table 3: Impact of Different Modules (NDCG@[K]). \( \sim \) is removing the following module and keeping the others.](image)

![4.3 Improvement of Different Component for Overall Performance](image)
to fuse yearly information first and then quarterly, and the results also prove our rationality.

4.4 Analysis of Different Pre-training Tasks Influence on Performance

In Table 4, we evaluate different pre-training tasks’ influence on the performance of the AMT dataset with 100 negative samples. Minus after H²SeqRec-fuse denotes only considering the following pre-training tasks. $M$ means the masking random items task, $S$ represents the masking subsequence task, and $H$ is the hyperedge link prediction task.

H²SeqRec-fuse-SH and H²SeqRec-fuse-MH obtain better performance, compared with H²SeqRec-fuse-MS, which proves the effectiveness of our proposed hyperedge prediction task. We can also get this conclusion by the last row. H²SeqRec-fuse-H shows the highest performance compared with H²SeqRec-fuse-M and H²SeqRec-fuse-S. Therefore, this experiment shows our proposed hyperedge link prediction task’s effectiveness, which is more helpful than the existing pre-training methods merely model each user’s buying history in a chronological way.

|          | K = 1 | K = 5  | K = 10 | K = 20 |
|----------|-------|--------|--------|--------|
| H²SeqRec-fuse | 0.1170 | 0.2135 | 0.2589 | 0.3054 |
| H²SeqRec-fuse-SH | 0.1149 | 0.2103 | 0.2568 | 0.3014 |
| H²SeqRec-fuse-MH | 0.1158 | 0.2102 | 0.2544 | 0.3036 |
| H²SeqRec-fuse-MS | 0.1138 | 0.2130 | 0.2584 | 0.3043 |
| H²SeqRec-fuse-M | 0.1123 | 0.2111 | 0.2541 | 0.3015 |
| H²SeqRec-fuse-S | 0.1117 | 0.2080 | 0.2533 | 0.3006 |
| H²SeqRec-fuse-H | 0.1140 | 0.2129 | 0.2572 | 0.3020 |

Table 4: Impact of different pre-training tasks (NDCG@K). ~ denotes only considering the following pre-training tasks.

4.5 Parameters Sensitivity Study

In the Figure 6, we analysis different dimensions of items influence on the proposed model and the baselines, we randomly choose three baselines to compare with our proposed models. The negative test sampling is 100, and the evaluation metric is NDCG@1.

![NDCG@1 w.r.t. # Dimensions](image1)

(a) NDCG@1 w.r.t. # Dimensions

![NDCG@1 w.r.t. # Epochs](image2)

(b) NDCG@1 w.r.t. # Epochs

Figure 6: Parameters Sensitivity.

As shown in Figure 6a, our proposed models H²SeqRec and H²SeqRec-fuse could always outperform GRU4Rec, SASRec and HyperRec when the dimension is \{20, 40, 60, 100\}. Moreover, when the number of dimensions is larger in the figure, the pre-training phase is more useful to the recommendation performance. It is because the pre-training data augmentation can learn more information about the sparse dataset when the number of dimensions is larger.

In Figure 6b, we compare the NDCG@1 on models training. GRU4Rec is early stopped because of not updating NDCG@1 for 10 epochs, so the line is not complete. In general, our proposed two methods always achieve the best performance, especially the H²SeqRec-fuse, whose NDCG@1 is still updating after 25 epochs.

4.6 Complexity Analysis

Our model contains four parts, item features extraction via self-supervised learning tasks, learning multi-scale embeddings via dynamic hierarchical hyperbolic hypergraphs, item embeddings enhancement via similar users groups and learning user preference for sequential recommendation. Therefore, we will analyse the time complexity of four separate parts, respectively.

For the self-supervised learning phase, we have three tasks. The first task, for each user, we should randomly select two different items to mask, so the time complexity is $O(U \times I_U \times (I_U - 1))$, where $U$ is the number of users, and $I_U$ is each user’s related items. Since $I_U \ll U$, the time complexity could approximately be $O(U)$. The second task is that we randomly mask a subsequence for each user, and the length is 2, so the time complexity is about $O(U \times I_U) \approx O(U)$. For the last task, we should find each user’s neighbours and two-hop neighbours, and then build the hyperedge connecting each user’s items within a timestamp. Therefore, if there are $T_M$ timestamps, the time complexity is $O(U \times T_M \times N_U \times |N_{N_U}|)$, where $N_U$ is the average numbers of each user’s neighbours and $|N_{N_U}|$ is the average number of each user’s two-hop neighbours. Since $T_M \ll U$, $N_U \ll U$, $|N_{N_U}| \ll U$, the complexity is approximately $O(U)$. The pre-training tasks’ time complexity can be added together, and could approximately ignore the smaller magnitude terms, so the overall time complexity of pre-training is $O(U)$. In practice, it is unnecessary to do the pre-training phase every time. The learned pre-training features could be learned offline and stored. Even if new data comes, it can implement incremental training. Therefore, the time complexity is acceptable.

For the multi-scale dynamic hierarchical hyperbolic hypergraphs learning, we have three time-spans views including month, quarter, and year. For monthly hyperbolic hypergraph neural network learning, we consider each user’s items within a month connected by a hyperedge. We perform hyperbolic-based hypergraph convolutional neural network on it, so the time complexity is $O(T_M \times I \times N_I \times L \times D^2)$, where $N_I$ is the average number of neighbours for each hyperedge connected to the item, because of the aggregation function in the model. $L$ is the number of layers, and $D$ is the embedding’s dimension. The time complexity is the same as the quarterly and yearly model, but the number of quarter $T_Q$ and year $T_Y$ is less than $T_M$. To sum up, this module’s time complexity is $O((T_M + T_Q + T_Y) \times I \times N_I \times L \times D^2) \approx O(I)$, because $T_M \ll T_Q, T_Y, L$ are countable, the average number of item’s neighbours is $N_I \ll I$ and the dimension is set as 100 in our model. What’s more, this module can also learn offline due to our step-by-step learning style. In practice, this will significantly save the time of the recommendation. The learning strategy is the same as the pre-training phase. If there are new data, this module can also do incremental training. Therefore, the time complexity of this module seems good.

For the item embeddings enhancement via similar users groups module, the training method and the hypergraph construction strategy is the same as the last module. However, this module
foci on the relationship between users and will construct multiple hypergraphs for each user. Therefore, the time complexity is $O(TM \times U \times N_U \times N_{N_{U_I}} \times L \times D^2)$, where $I_U$ means each user’s related items, $N_{N_{U_I}}$ is the average number of neighbors of each user’s related items and $N_{N_{U_I}}$ is our defined similar user’s related items in section 3.3. Similarly, the approximate time complexity is $O(U)$ because of $N_{N_{U_I}} \ll U$. This module can also train offline in advance, thanks to our step-by-step training strategy. In practice, if there are new users fed into the model, it is convenient to train new users’ item embeddings. If some existing users are fed into the model again, it is convenient to train incrementally based on the previous version of item embeddings.

In the last module, learning user preference for sequential recommendation module, we use the Transformer to obtain each user’s preference embedding. For each user, we feed user’s sequence into the Transformer (self-attention layer), and therefore the time complexity is $O(U \times I_U^2 \times D) = O(U)$ because of $I_U \ll U$ as claimed before. To train the recommendation, we sample 1 negative training item to maximize the difference between the positive user-item pair score and the negative one. The recommendation part’s time complexity is $O((I_U \times (NEG + 1)))$. Since the user preference embedding learning part and the recommendation part are learned end to end, the whole time complexity of the last module is $O(U \times I_U^2 \times D \times I_U \times (NEG + 1)) \approx O(U)$ because of $I_U \ll U$ and $NEG \ll U$.

To sum up, the final time complexity of the proposed model is the maximum time complexity of the above modules, which is $max(O(U), O(U))$, so it is acceptable.

5 RELATED WORK

5.1 Non-graph sequential modelling for neural recommendation

Early neural recommendations building on typical deep neural networks mostly use recurrent neural networks (RNN) or convolutional neural networks (CNN) to capture the temporal patterns from users’ historical records. Specifically, Quaderna et al. [19] design a multi-level RNN structure for learning temporal patterns in a sequence. Although RNN-based methods have their advantages in sequential learning, user-item interactions usually contain noisy information. Only learning these relations is far from achieving more reliable recommendation system. Unlike RNN-based method, Yuan et al. [44] use CNN to learn from user-item sequences because CNN-based methods can not only capture long-term dependencies but also have the character of translation invariance, making the model more stable for various sequential orders. Later, attention is introduced in recommendation models [16, 22, 23, 34]. In particular, Kang et al. [12] treat the user-item sequence as a sentence and model the temporal relations via a transformer model [26]. The transformer takes a self-attention unit to translate sequences as entity embeddings and position embeddings, which can be used to the downstream recommendation system.

5.2 Graph-based sequential modelling for neural recommendation

In the real world, however, users and items contain more non-linear relations with different topological structures. To this end, graph-based sequential modelling approaches become more and more popular to model the recommendation data. By modelling graph structure data, graph neural networks (GNN) are introduced recently. As a conceptional extension of linear CNNs, GNN-based method [7, 35, 39] take the directed graph as input, and capture the interdependence directly on the graph. For example, Wu et al. [35] treat each user or item as a node in the graph, and transform the user-item sequence as a path. With GNN-based model, they can learn both users and items embeddings over the whole graph.

Later, more advanced technologies have been introduced such as, self-supervised learning [11, 33, 37, 38, 40], group awareness [8, 10, 31] and so on. In particular, Hwang et al. [9] propose self-supervised auxiliary learning tasks to predict meta-paths to capture rich-information of a heterogeneous graph, and thereby improve the primary task to do link prediction.

5.3 Hypergraph-based sequential modelling for neural recommendation

Recently, instead of traditional graph, constructing hypergraphs to learn the data structure to do recommendation become a popular approach. Yu et al. [43] design a multi-channel hypergraph convolutional network to enhance social recommendation by exploiting high-order user relations, which shows great improvement. However, it ignores the sequential information for users. Xia et al. [36] model session data as a hypergraph and propose a dual channel hypergraph convolutional network for session-based recommendation. Wang et al. [30] also construct hypergraphs for each session to model the item correlations, and they also introduce a hypergraph attention layer to flexibly aggregate correlated items in the session and infer next interesting item. Although the above two methods both consider some temporal information, it is within a specific session, not a whole sequence. Wang et al. [29] propose a novel next-item recommendation framework empowered by sequential hypergraphs to incorporate the short-term item correlations while modeling the dynamics over time and across users. The advantage is to consider sequential information, but all the above three hypergraph-based methods ignore the severe sparsity problem of hypergraphs.

6 CONCLUSION

In this work, we focus on the sparse problem of most existing hypergraph-based sequential recommendation and on the lacking exploitation of hidden hyperedges among users problem, and propose Hyperbolic Hypergraph representation learning method for Sequential Recommendation (H²Rec) with pre-training phase. Experiments show that our proposed model outperforms the state-of-the-art sequential recommendations and each of the components contributes to the whole architecture. However, on a sparse dataset, the data augmentation improves the recommendation, while on a dense dataset, it is useless and even worse, which proves dense datasets do not need data augmentation to achieve better performance. In future work, we will explore multiple sequential ways to model the sequential recommendation data on graphs.
