

Required concepts

Hypergraph

Hypergraphs differ from traditional graphs in that they allow for edges to connect more than two vertices. A hyperedge, denoted by $G = (V, E, W_e)$, consists of a set of vertices (V), a set of hyperedges (E), and a set of weights assigned to each hyperedge (W_e) [^]. An $|V| \times |E|$ incidence matrix, denoted by L , can be used to represent the hypergraph G , as illustrated in Figure (1).

Entries of the incidence matrix are defined as formula (1).

$$L = \begin{cases} 1, & v \in e \\ 0, & v \notin e \end{cases} \quad (1)$$

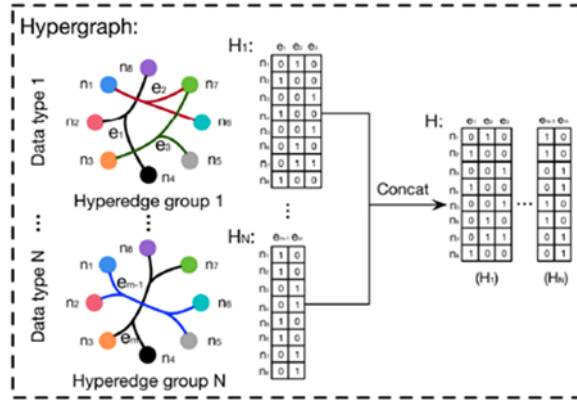


Figure (1): hypergraph and its incidence matrix [^]

Static and dynamic embedding

Embeddings are low-dimensional vectors to represent features of vertices and edges. Two embeddings can be considered for each item. First, the static item embedding, which is unchanged over different times and symbolizes features such as inherent features (such as color, etc.) and does not change over time and between users. Second, the dynamic item embedding, which is their dynamic meaning, and is made by the hypergraph convolution network from the neighbors of item in each time period.

Hypergraph convolution network

The graph convolutional network is developed by generalizing the convolutional neural network for graph data and combining it with graph representation learning. In short, general function of the graph convolution network is merging the features of each vertex (which can be a vector, number or...) under an operation called "message passing", with features of its adjacent vertices. If network consists of more than one layer, the subsequent layers repeat the message passing operation and aggregate information from the neighbors of neighbors of the target vertex and merge them with the values obtained from the previous layer. Finally, the output layer generates the embedding vector. Interestingly, this process is very similar to how convolutional neural networks work from pixel data. But in this layer, we have used the hypergraph convolution network model, which is a generalization of the graph convolution network for hypergraphs.

The formula of hypergraph convolution network was obtained for the first time in [14] from the generalization of graph convolution formula using new hypergraph concepts. (2) shows recursive formula to obtain the dynamic embedding of the l th convolution layer from the $l-1$ th layer of each item in different time periods interval.

$$X^{t_n, l} = \tau(D^{t_n-1/\gamma} H^{t_n} W^{t_n} B^{t_n-1} H^{t_n T} D^{t_n-1/\gamma} X^{t_n, l-1} P^l) \quad (2)$$

Here t_n is time. H^{t_n} is hypergraph incidence matrix at time t_n . W^{t_n} is diagonal weight matrix for each hyperedge. D^{t_n} and B^{t_n} are the diagonal degree matrices for vertex and hyperedge correspondingly. $X^{t_n, l}$ is matrix of dynamic embeddings of items at time t_n and the l th convolution layer. P^l is trainable weight matrix and $\tau(\cdot)$ represents the activation function (ReLU in our experiment).

Gate function

This method was created by authors of [13] to create intermediate connection in the neural network and they have observed significant results in improving the feature learning performance in the neural network by proposing a solution to create a suitable weight for the gate outputs. (3) shows the gate function formula and its coefficient, respectively.

$$x_i^{t_{n'}} = g x_i^{t_{<n,L}} + (1 - g) e_i \quad (3)$$

$$g = \frac{e^{Z_R^T \sigma(W_R x_i^{t_{<n,L}})}}{e^{Z_R^T \sigma(W_R x_i^{t_{<n,L}})} + e^{Z_R^T \sigma(W_R e_i)}}$$

In which W_R and Z_R is the transformation matrix and vector for the gate. $\sigma(\cdot)$ is the tanh. $x_i^{t_{<n,L}}$ are dynamic item embeddings before time t_n and e_i is static item embedding. In this way, g is used to control the percentage of residual information that will be retained.

Attention model

The attention model is inspired by the human way of thinking, for which the importance of issues is different. The general working of this model is that by assigning different weights to the input to produce the output, it makes the system understand the importance of different parts of the input sequence and improve its performance. Recommender systems also learn how to assign weight to each interaction in user interactions sequence, build a suitable representation for them, and by finding similarities between these representations, predict the next interaction and recommend it to user.

Transformer

Transformer model is a deep learning model that uses attention mechanism. This model was first introduced in 2017 in [14]. The transformer model is commonly used in natural language processing research. The transformer is a sequence-to-sequence (encoder-decoder) model that tries to use only the attention mechanism without using recursive mechanisms. Transformers are one of the most important tools for working with sequential data, with the possibility of parallelization and thus reducing time to a significant amount. An overview of the transformer architecture is shown in Figure (2).

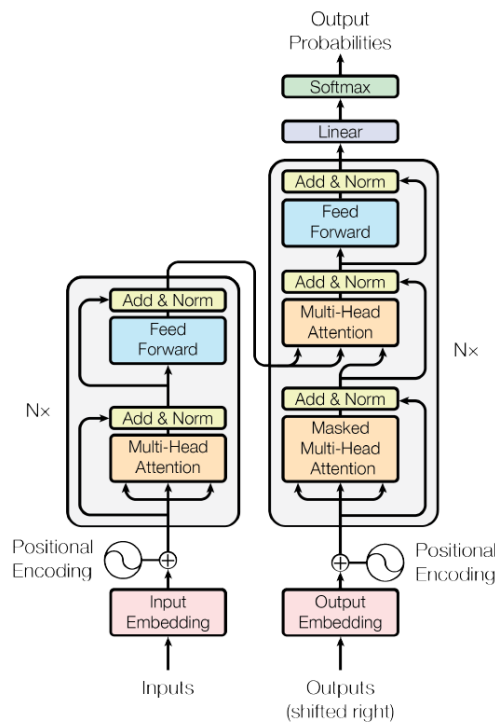


Figure (2): Transformer architecture [14]

Proposed approach

Figure (3) shows a general outline of the framework of the proposed approach in this work. our model consists of 4 layers of item embedding generation, user embedding generation, user-item interaction generation, and predicting the next user interaction, which will be explained in detail below.

Item embedding generation

The purpose of this layer is to produce a strong representation for the items, and it includes two parts of the hypergraph convolution network and the residual gate, which we will first explain the part of the hypergraph convolution network.

- hypergraph convolution network

The meaning of an item can change over time and change across users, for example, a user at a certain time prepared a bouquet of flowers for his wedding because the other products purchased by him are related to the wedding. That is, here the meaning of the bouquet is "a device for the wedding ceremony". But at another time, we see this user has prepared a bouquet of flowers as a gift for another person because he has also bought other accessories such as sweets and vases, and the meaning of the bouquet changes to "gift". For this reason, in this model, we use the short-term correlation of neighboring items in the hypergraph to mean them and to display these short-term correlations.

For each time period considered here as a year, there is a hypergraph. In other words, hypergraphs separate interactions between user and items in different time periods.

As we said before, there are two embeddings for each item. We used word2vec algorithm to make static embedding. Word2vec algorithm is a text processing model used to generate word embeddings. We also use the convolution network to make dynamic embedding. To be more precise, in each time period, the latest version of the information of neighboring vertices (the most up-to-date version of their dynamic embedding) is sent to the target item and is merged with dynamic embedding of the item itself in the previous time. This is done in layers. In this way, in the first layer, information is gathered from the first-order neighbors of the item, i.e., its direct neighbors, and in the next layer, information is gathered from the neighbors of its neighbors, and so on.

- residual gate

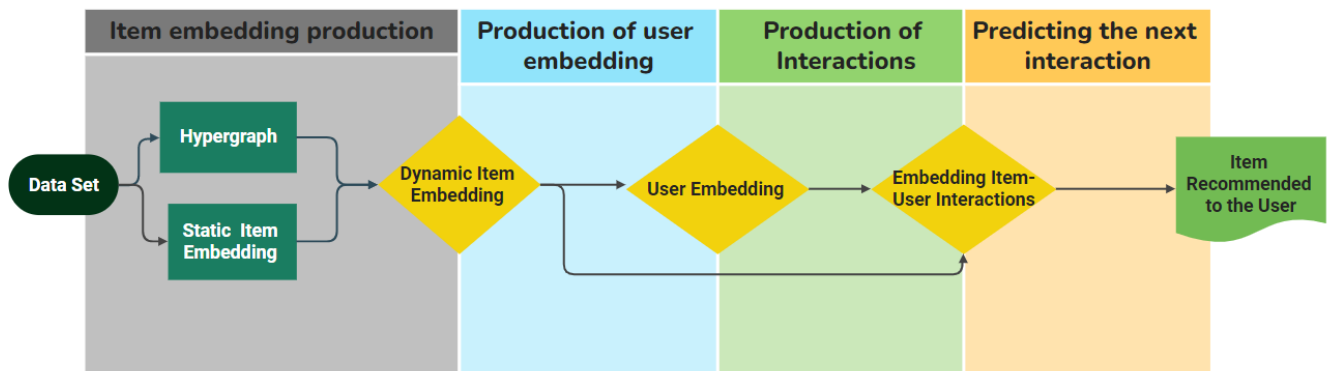


Figure (3): The framework of the proposed approach

Meaning of items is changing over time, but there is a connection between their meanings at different time periods. More precisely, it can be said that the meanings of an item in the past can help to discover its meaning in the present. Therefore, in the first layer, a component called "residual gate" is designed to establish a connection between time periods. That is, we will combine the static embedding, which is the unchanged features of item, with dynamic embeddings of the previous times in a weighted manner and send it to the next time period as the initial dynamic embedding.

user embedding generation

After modeling the items as vectors with real numbers, it's time for the users. our goal in the second layer of the recommendation system is to model the users who are the hyperedges in the hypergraph. In fact, the user's representation in any time period can be considered as his/her short-term intent in that period of time. The user's short-term intent can also be understood from the items that he interacted with during that time. We can merge those items by using the formula (ξ) and get an embedding for each user's short-term intent.

$$U^{t_n} = \tau(B^{t_n-1/r} H^{t_n T} D^{t_n-1/r} X^{t_n L} P^L) \quad (\xi)$$

The resulting matrix $U^{t_n} = [u_1^{t_n}, u_r^{t_n}, \dots, u_{|\mathcal{E}^{t_n}|}^{t_n}]$ can be regarded as an assembly of short-term user intents at t_n .

user-item interactions generation

In the third layer, we are looking for a way to represent user-item interactions because we need them in the next layers of the recommendation system. For this purpose, it combines the dynamic item embedding and the embedding of the short-term user intent to achieve the embedding of these interactions. Separating the concept of user-item interaction from the item is the effect of users on items.

The formula and implementation of the code of this layer is like the residual gate layer, shown in (°) and it trains the w_f and Z parameters to properly combine the three values of static and dynamic item embedding and user embedding in a weighted manner.

$$e_{i,u}^{t_n} = \alpha_u u_u^{t_n} + \alpha_d x_i^{t_n L} + (1 - \alpha_d - \alpha_u) e_i \quad (°)$$

$$\alpha_u = \frac{e^{z^T \sigma(w_F u_u^{t_n})}}{e^{z^T \sigma(w_F u_u^{t_n})} + e^{z^T \sigma(w_F x_i^{t_n L})} + e^{z^T \sigma(w_F e_i)}}$$

$$\alpha_d = \frac{e^{z^T \sigma(w_F x_i^{t_n L})}}{e^{z^T \sigma(w_F u_u^{t_n})} + e^{z^T \sigma(w_F x_i^{t_n L})} + e^{z^T \sigma(w_F e_i)}}$$

in which e_i and $x_i^{t_n L}$ is the static and dynamic item embedding correspondingly, and $u_u^{t_n}$ is the vector in the matrix generated by Equation? to indicate the short-term user intent at t_n . w_f and Z is the transformation matrix and vector correspondingly.

Prediction of the next interaction

Now that we have built user interactions with items, we can use them to predict the user's next interaction and make a recommendation for him/her. This is done using the transformer model. To find different patterns of user preferences, various mechanisms are used in recommender systems. It is enough to look at his preferences during different times in terms of the components of a sequence, and the recommender system will find the continuation of this sequence and suggest it to the user. In fact, Transformer looks at the sequence of user interactions like sentence components and tries to find similarities between them. Then, using similarity, it guesses the user's next interaction.

Recommendation the next item to user

Next user interaction embedding as an output is not acceptable to us. The recommendation system should consider items to suggest to user. So, it is necessary to extract the item embedding required by the user from the existing user-item interaction embedding.

For this, a simple mathematical operation is performed. More precisely, with the internal multiplication of two vectors, we can convert the vector into a number and obtain a number to show the score of each item to be recommended to user. Therefore, we add the dynamic item embedding to its static embedding and then multiply it with the latest version of the user's dynamic preference to predict a recommendation for user.

At the end, according to the calculated score, a rank is assigned to item, which is used to calculate the loss function and evaluate the system performance. finally, the first items are suggested to the user.

EXPERIMENTS

In this part, we introduce the dataset used in this research. Also, we describe the details of various calculations and operations for training, including sampling, etc. At the end, we show the results of the obtained indicators.

Data

In this work, we used the Amazon Review¹ dataset, which is the purchase history of customers from the Amazon website from 1996 to 2014. Data include user-product interactions, user descriptions of items, other products purchased by the buyer of an item, and general descriptions of Product, including price, brand, product category, etc. The data in category 19, including books, health and beauty, home appliances, etc. contain more than 233 million interactions. The data files consisted of two parts, metadata and reviews, of json file type. The metadata file is the description of each item. In such a way that next to the ID of each item, features such as price, category, description of appearance features and how to use it, the address of the corresponding page on the Amazon site, etc. are mentioned. In the reviews file, along with the item ID and the user ID, values such as the user's opinion about the product, the date and the summary of the description of that product, etc. are given. We selected 6 categories of sports equipment, health care products, clothing and jewelry, home appliances, and kitchen appliances for our work. The statistics of the dataset is given in Table (1).

Table 1: Statistics of the dataset

Dataset	Number of interactions	Number of users	Number of items	Density
Amazon review	1007	023	417	0.0305%

For data preprocessing, we cleaned and removed problematic data. For example, some attributes for items or interactions were missing and the value was null. This issue caused system performance problems. Also, there was a lot of duplicate data in it, which increased the data volume unnecessarily.

To improve system performance by removing items with less than 10 interactions, which accounted for about 23% of the total data, and users with less than 10 interactions, which accounted for about 32% of the remaining 77%, a dataset with 1007 interactions, 023 users, and 417 The item remained for us.

Experimental setting

- Baselines

Next, in this section, we will introduce two new models in the field of sequential recommender systems based on graph neural networks that we have compared their performance with our own model. These models have used different basic models (which we have explained in section 1) in the construction stages of their neural network.

KE-GNN [9] is a recommendation system based on a graph neural network that converts sequential information into directed edges in the graph with a simple method. In addition, it uses graph neural network to learn knowledge graph embeddings. A knowledge graph shows a network of real-world entities, i.e. objects, events, situations or concepts and the relationship between them. It then combines these embeddings and recommends a model to be trained based on them. The outline of this model consists of two parts of current interest, i.e., sequential preference, and general interest, i.e., meaning-based preference, which finally combine these two parts.

ISSR [10] is a recommendation system based on graph neural network that uses several types of graphs to model items and then recommends items to user using recurrent neural network and attention mechanism. The design of this model is based on two concepts: 1) Correlation between sequences of items. it is the similarity between two items in the sequences of different users and 2) Correlation within the sequence of items: similarity between two items that belong to the same user. In this work, both inter-sequence correlation and intra-sequence correlation are used to create a suitable representation for the items. These two concepts are modeled in different modules, and after obtaining the user's preference, the probability of each candidate item to be recommended is checked.

- Training and validating the model

Since our training data for each user is small, the leave one out system has been used for training. That is, in the sequence of user interactions, each time we consider one of the items as test data and the rest of the items as training data.


¹https://cseweb.ucsd.edu/~jmcauley/datasets.html#amazon_reviews

On the other hand, number of positive samples, i.e., the items that user has interacted with, is much less than the items in the data set which he has not interacted, we use negative sample technique in our model. That is, training data is in the form of pair training (u, i, j) that user u has interacted with item i and not with item j. Finally gives it to a pair loss function. In the implementation of our code, negative sampling is done at a rate of 1, that is, in the training data, for every positive sample i, there is also a negative sample j. In the model testing stage for each positive sample, we randomly select a negative item among the items that user has not interacted with.

We train the model with positive and negative samples. According to the user's preference that model has learned, we assign a rank to the positive item that is in test data next to the negative test samples. Finally, we give the ranks to the Bayesian pairwise loss function to measure the performance of the model. Figure (4) shows how we train and validate the model in this work.

Figure (4): Negative sampling and leave one out technique.

The loss function used in this work is Bayesian Pairwise Loss [19] in the form of formula (6):



$$L = \sum_{(u,t,i,j) \in C} -\ln \delta(\bar{y}_{u,i}^t - \bar{y}_{u,j}^t) + \lambda \|\theta\|^2 \quad (6)$$

in which $\|\theta\|^2$ denotes the L^2 regularization and λ is used to control its weight. δ is the Sigmoid function. $\bar{y}_{u,i}^t$ is the rank obtained by the model for positive item i for user u at time t. $\bar{y}_{u,j}^t$ is the rank that the model obtains for negative item j for user u at time t.

After obtaining the ranks of items related to each user, we calculate the loss function for each user and use its average for all users as the overall loss function for that training epoch.

- Parameters

Implementation of the project is done with Python language using the methods available in the reliable Pytorch framework and adopt Adam as the optimizer. We use Google's pre-trained Word2Vec model to build static embedding. This model consists of an embedding vector of length 300 for 3 million words and phrases trained on 100 billion words from the Google News dataset.

We also use Grid search method to set the hyperparameters. In this method, the model is built with all different combinations for the values of hyperparameters and the best result obtained is considered as the final hyperparameters. we grid search for the dropout rates in $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, the regularization weight λ in $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$, the learning rate in $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ and the embedding size in $\{50, 100, 150, 200\}$. For the model-specific hyper-parameters, we fine-tune them based on results in the validation set. After a Grid search, we have given the best results in table (5):

Table (5): Final numbers of hyperparameters

General hyper parameters	Model-specific hyperparameters
Dropout rate = 0.5	Coevolution layer=3
$\lambda = 0.1$	Time period=12
Learning rate = 0.001	head attention=3
Embedding size = 100	blocks of transformer=3

- Evaluation Metrics

Recommender system evaluation metrics help us understand the quality of recommended items. The metrics we use are three below.

- MRR**. is an evaluation metric that evaluates the correctness of system outputs for queries that are assigned to it as tasks in form of formula (7).

¹ Google's pre-trained Word2Vec : E:/google news vector/GoogleNews-vectors-negative300.bin.gz

$$MRR = \frac{1}{Q} \sum_{q=1}^Q \frac{1}{rank_q} \quad (V)$$

Here Q is number of queries. $rank_q$ is the rank of the system's first correct answer for each query.

b) **NDCG@K** is an order-aware metric derived from several simpler metrics i.e., formula (Λ).

$$DCG@K = \sum_{k=1}^K \frac{rel_k}{\log_2(1+k)} \quad (\Lambda)$$

In which rel_k is degree of relevance of the k th system output with the query. K is number of system output items. Since the size of the value of this formula is greatly influenced by the range of rel_k , we make another change in the formula (Λ) as follows:

$$NDCG@K = \frac{DCG@K}{IDCG@K} \quad (A)$$

In which $IDCG@K$ is the ideal form of $DCG@K$ that most relevant answers are at the highest ranks.

c) **Hit@K**. as shown in (10) hit is an order-unaware metric and specifies the number of users for whom there is a correct answer in a list of k items recommended by the system.

$$Hit@K = |U_{hit}^K| \quad (10)$$

In our work, these three metrics are in the form of three formulas (11), (12) and (13).

$$MRR = \frac{1}{U} \sum_{u \in U} \frac{1}{r_u} \quad (11)$$

U : The number of users is actually the number of tasks that the system must perform.

r_u : The rating that the system gives to the positive font in the test set.

$$\begin{cases} NDCG@K = \frac{1}{\log_2(1+r_u)} & r_u \leq k \\ . & r_u > k \end{cases} \quad (12)$$

In the main paper, it is ignored for ease of calculation because $IDCG@K$ is constant for all users. We do the same so that our results can be compared with them.

The number of users whose test data contains a positive font in the Top- k list is measured as $Hit@K$:

$$\begin{cases} HIT@K = 1 & r_u \leq k \\ . & r_u > k \end{cases} \quad (13)$$

Result

In the end, to check the performance of our work and compare it with the previous methods that were introduced in section 1, we ran them on the Amazon review dataset. whose results are reported in table (3). The numbers you see in the table are the results of three evaluation metric MRR and NDCG for $k=1$ and $k=5$ and Hit $k=1$ and $k=5$. Of course, these two metrics are equal for $k=1$.

Table (3): Result. * indicates that the improvement of the best result is statistically significant compared with the next-best result with $p < 0.01$.

	Hit@1	Hit@5	NDCG@5	MRR
	NDCG@1			
KE-GNN	0.1090	0.4183	0.4632	0.4998
ISSR	0.1213	0.4000	0.4920	0.4803
HyperRec	0.1381	0.4901	0.5238	0.5112
Our work	0.1519*	0.5391*	0.5761*	0.5113*