Time Interval Aware Self-Attention for Sequential Recommendation

Jiacheng Li
University of California, San Diego
j9li@eng.ucsd.edu

Yujie Wang
Florida State University
yw18@my.fsu.edu

Julian McAuley
University of California, San Diego
jmcauley@eng.ucsd.edu

ABSTRACT

Sequential recommender systems seek to exploit the order of users' interactions, in order to predict their next action based on the context of what they have done recently. Traditionally, Markov Chains (MCs), and more recently Recurrent Neural Networks (RNNs) and Self Attention (SA) have proliferated due to their ability to capture the dynamics of sequential patterns. However a simplifying assumption made by most of these models is to regard interaction histories as ordered sequences, without regard for the time intervals between each interaction (i.e., they model the time-*order* but not the actual timestamp). In this paper, we seek to explicitly model the timestamps of interactions within a sequential modeling framework to explore the influence of different time intervals on next item prediction. We propose TiSASRec (Time Interval aware Self-attention based sequential recommendation), which models both the absolute positions of items as well as the time intervals between them in a sequence. Extensive empirical studies show the features of TiSASRec under different settings and compare the performance of self-attention with different positional encodings. Furthermore, experimental results show that our method outperforms various state-of-the-art sequential models on both sparse and dense datasets and different evaluation metrics.

CCS CONCEPTS

• **Information systems** → **Retrieval models and ranking**.

KEYWORDS

Recommender System, Sequential Prediction, Self-attention, Time-aware Model

ACM Reference Format:
Jiacheng Li, Yujie Wang, and Julian McAuley. 2020. Time Interval Aware Self-Attention for Sequential Recommendation. In *The Thirteenth ACM International Conference on Web Search and Data Mining (WSDM '20), February 3–7, 2020, Houston, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3336191.3371786

1 INTRODUCTION

Modeling sequential interactions is essential in applications like e-commerce, friend suggestion, news recommendation, etc. There are two important lines of work that seek to mine users' interaction histories: temporal recommendation and sequential recommendation. Temporal recommendation [17, 29, 31, 35] focuses on modeling *absolute timestamps* to capture the temporal dynamics of users and items. For example, the popularity of an item might change during different time slots, or users' average ratings might increase or decrease over time. These models are useful when exploring the temporal changes in datasets. Instead of sequential patterns, they consider temporal patterns which are dependent on the time.

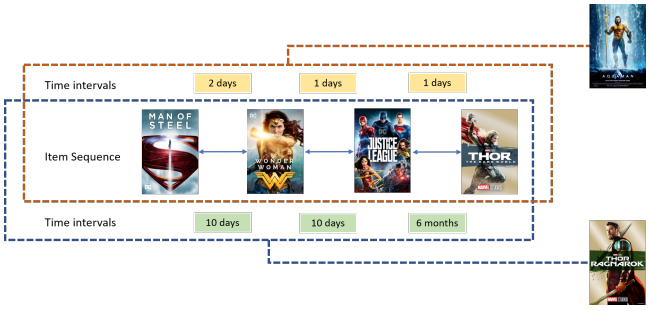


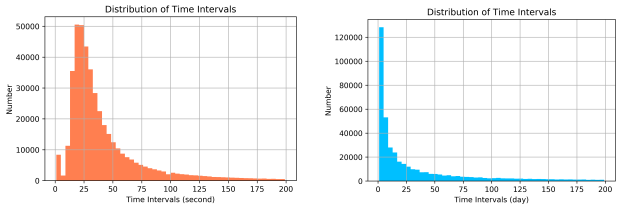
Figure 1: Given the same item sequence with different time intervals, our model will yield a different prediction for the next item.

Most previous sequential recommenders sort items by interaction timestamps, and focus on sequential pattern mining to predict the next item likely to be interacted with. One typical solution is Markov Chain based approaches [7, 8, 21], where an L-order Markov chain makes recommendations based on the L previous actions. Markov Chain based approaches have been successfully adopted to capture short-term item transitions for recommendation [14]. They perform well in high-sparsity settings by making strong simplifying assumptions but may fail to capture the intricate dynamics in more complex scenarios. Recurrent Neural Networks (RNNs) [11, 12] have also been applied in this setting. While RNN models have long 'memory' across users' preferences, they require large amounts of data (and especially *dense* data) before they can outperform simpler baselines. To tackle the shortcomings of Markov Chain models and RNN-based models, inspired by Transformer [26] for machine translation, [14] propose to apply self-attention mechanisms to sequential recommendation problems. Self-attention based models significantly outperform state-of-the-art MC/CNN/RNN-based sequential recommendation methods.

Generally, previous sequential recommenders discard timestamps and preserve only the order of items, that is, these methods (implictly) assume that all adjacent items in a sequence have the same time intervals. The factors that influence the next items are

WSDM '20, February 3–7, 2020, Houston, TX, USA

ACM ISBN 978-1-4503-6822-3/20/02…$15.00
https://doi.org/10.1145/3336191.3371786



(a) Ml-1m (b) Amazon cds vinyl

Figure 2: Time interval distribution of two datasets (excluding the interval of zero).

only the position and identity of the previous items. However, intuitively, items with more recent timestamps will have more influence on the next item. For instance, two users have the same interaction sequence but one of the users produced these interactions within one day, while another user completed these interactions in one month, therefore, the interactions of the two users should have different impact on the next items even if they have the same sequential position. However, previous sequential recommendation techniques view the two situations as the same because they only consider the sequential position.

In this paper, we argue that user interaction sequences should be modeled as a sequence with different time intervals. Figure 2 indicates that the interaction sequence has different time intervals, some of which might be large. Previous work omitted these intervals and their influence on the predicted item. To address these above limitations, inspired by Self-Attention with Relative Position Representations [22], we propose a time-aware self-attention mechanism. This model not only considers the absolute positions of items like SASRec [14], but the relative time intervals between any two items. From our experiments, we observe that our model outperforms state-of-the-art algorithms on both dense and sparse datasets. Finally, our contributions are summarized as follows:

- We propose to view user's interactions history as a sequence with different time intervals, and model different time intervals as relations between any two interactions.
- We combine the advantages of absolute position and relative time interval encodings for self-attention and design a novel time interval aware self-attention mechanism to learn the weight of different items, absolute positions and time intervals to predict future items.
- We conduct thorough experiments to study the impact of absolute positions and relative time intervals and different components on the performance of TiSASRec, and show that it outperforms state-of-the-art baselines in terms of two ranking metrics.

2 RELATED WORK

2.1 Sequential Recommendation

Sequential recommender systems mine patterns in user interaction sequences. Some capture item-item transition matrices to predict the next item. For instance, FPMC [21] models long-term preferences and dynamic transitions of users via matrix factorization and a transition matrix. The transition matrix is a first-order Markov Chain (MC) which only considers the relation between the current and the previous item. Fossil [8] uses similarity-based methods and high-order Markov Chains which assumes that the next item is related to several previous items and [7] demonstrated that the high-order MC based models have strong performance on sparse datasets. CNN (Convolutional Neural Network) based methods [24, 25] have also been proposed to consider several previous items as an 'image' and mine transitions between items via a CNN at a union-level. MARank [34] unifies both individual- and union-level previous item transitions by combining the residual network and multi-order attention. SHAN [33] models the previous several items and a long history of a user via a two-layer attention mechanism to obtain the short- and long-term preferences of a user.

RNNs (Recurrent Neural Network) based models like [3, 12, 13, 18] use RNNs to model entire user sequences. These methods perform well on dense datasets, but generally exhibit poor performance on sparse datasets.

2.2 Attention Mechanisms

Attention mechanisms have been shown to be effective in various tasks such as image captioning [32] and machine translation [2]. Essentially the idea behind attention mechanisms is that outputs depend on specific parts of an input that are relevant. Such mechanisms can calculate weights of inputs and make models more interpretable. Recently, attention mechanisms have been incorporated into recommender systems [5, 28, 30]. A purely attention-based sequence-to-sequence method, Transformer [26], achieved state-of-the-art performance on machine translation tasks. Transformer uses the scaled dot-product attention which is defined as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}}\right)\mathbf{V} \tag{1}$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ represent queries, keys and values respectively. In self-attention, the three inputs usually use the same object.

The self-attention modules of Transformer have also been used in recommender systems [14] and achieved state-of-the-art results on sequential recommendation. Since the self-attention model doesn't include any recurrent or convolutional module, it is not aware of the positions of previous items. One solution is to add positional encodings to the inputs, which can be a deterministic function or learnable positional embedding [23, 26]. Another solution uses relative position representations [4, 22]. They model the relative positions between two input elements as pairwise relationships. Inspired by self-attention with relative positions, we combine the absolute positions and relative positions to design time-aware self-attention which models items' positions and time intervals.

3 PROPOSED METHOD

In this section, we first formulate our next item recommendation problem and then describe our approaches to obtain time intervals and components of TiSASRec, which includes personalized time interval processing, an embedding layer, time-aware self-attention blocks, and a prediction layer. We embed items, their absolute positions, and relative time intervals. The attention weights are calculated by these embeddings. As shown in Figure 1, our model

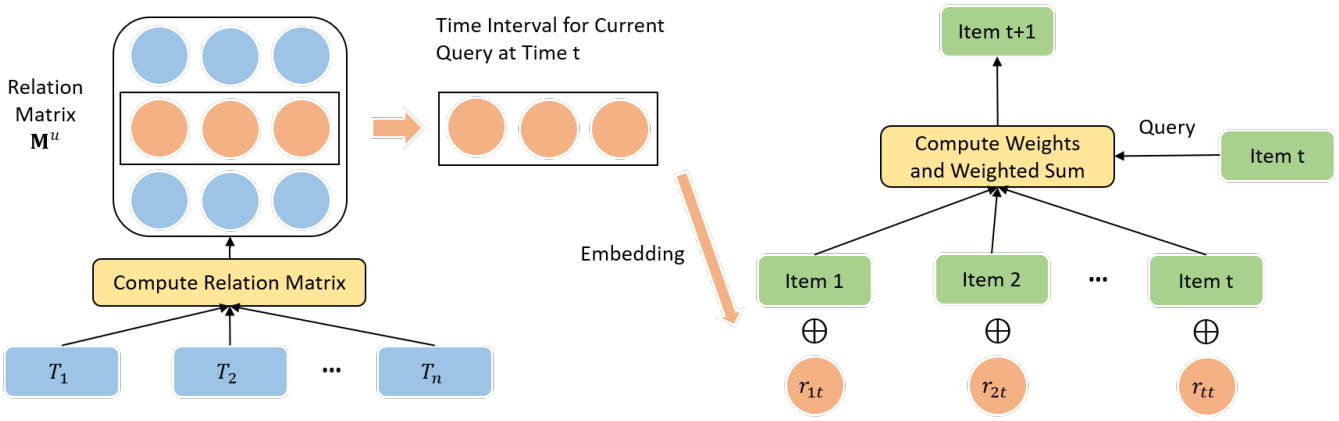


Figure 3: The overall framework of TiSASRec. We only demonstrate the components corresponding to time intervals and self-attention.

Table 1: Notation.

Notation	Description
U, I	User and Item set
S^u	historical interaction sequence for user u
T^u	timestamp sequence of user u corresponding to S^u
R^u	time interval matrix between any two items.
n	maximum sequence length
d	latent vector dimension
$\mathbf{M}^u$	time interval matrix after personalized processing
$\mathbf{M}^I$	item embedding matrix
$\mathbf{M}_K^P, \mathbf{M}_V^P$	embedding matrix of position for key and value
$\mathbf{M}_K^R, \mathbf{M}_V^R$	embedding matrix of intervals for key and value
$\mathbf{Z}_t$	output of model as time step t

will have different predicted items given different time intervals even if several previous items are the same. To learn the parameters, we employ a binary cross entropy loss as our objective function. The goal of TiSASRec is to capture sequence patterns and explore influence of time intervals on the sequential recommendation which is an unexplored area.

3.1 Problem Formulation

Let U and I represent the user and item set respectively. In the setting of time-aware sequential recommendation, for each user $u \in U$, we are given the user's action sequence, which is denoted as $S^u = (S_1^u, S_2^u, \ldots, S_{|S^u|}^u)$ where $S_t^u \in I$ and time sequence $T^u = (T_1^u, T_2^u, \ldots, T_{|T^u|}^u)$ corresponding to the action sequence. During the training process, at time step t, the model predicts the next item depending on the previous t items and the time intervals r_{ij}^u between item i and j. Our model's input is $(S_1^u, S_2^u, \ldots, S_{|S^u|-1}^u)$ and time intervals R^u between any two items in the sequence, where $R^u \in \mathbb{N}^{(|S^u|-1)\times(|S^u|-1)}$. Our desired output is the next item at every time: $(S_2^u, S_3^u, \ldots, S_{|S^u|}^u)$.

3.2 Personalized Time Intervals

We transform the training sequence $(S_1^u, S_2^u, \ldots, S_{|S^u|-1}^u)$ into a fixed-length sequence $s = (s_1, s_2, \ldots, s_n)$, where n represents the maximum length that we consider. If the sequence length is greater than n, we only consider the most recent n actions. If the sequence length is less than n, we add padding items to the left until the length is n. Similarly for the time sequence $T^u = (T_1^u, T_2^u, \ldots, T_{|T^u|-1}^u)$, we also consider a fixed-length time sequence $t = (t_1, t_2, \ldots, t_n)$. If the time sequence length is less than n, we pad it with the first time stamp $T_{s_1}^u$ where s_1 is not a padded item. Otherwise, we only consider the first n time stamps.

We model time intervals in an interaction sequence as the relation between two items. Some users have more frequent interactions while others do not and we are only concerned about the relative length of time intervals in one user sequence. Hence, for all time intervals, we divide by the shortest time interval (other than 0) in a user sequence to get the personalized intervals. Specifically, giving the fixed-length time sequence $t = (t_1, t_2, \ldots, t_n)$ of user u, the time intervals of two items i and j is $|t_i - t_j|$ and the time interval set (except for 0) of user u is $\mathbf{R}^u$. We denote the minimum time interval of a user as $r_{\min}^u = \min(\mathbf{R}^u)$. The scaled time intervals $r_{ij}^u = \left\lfloor \frac{|t_i - t_j|}{r_{\min}^u} \right\rfloor$, where $r_{ij}^u \in \mathbb{N}$. Hence, the relation matrix $\mathbf{M}^u \in \mathbb{N}^{n\times n}$ of user u is:

$$\mathbf{M}^u = \begin{bmatrix} r_{11}^u & r_{12}^u & \cdots & r_{1n}^u \\ r_{21}^u & r_{22}^u & \cdots & r_{2n}^u \\ \cdots & \cdots & \cdots & \cdots \\ r_{n1}^u & r_{n2}^u & \cdots & r_{nn}^u \end{bmatrix} \tag{2}$$

The maximum relative time interval between two items we consider is clipped to k. We hypothesize that precise relative time intervals are not useful beyond a certain threshold. Clipping the maximum intervals also avoids sparse relation encodings and enables the model to generalize to time intervals not seen during training. Hence, the clipped matrix is $\mathbf{M}_{clipped}^u = clip(\mathbf{M}^u)$, where the clip operation of the matrix applies to every element $r_{ij}^u = \min(k, r_{ij}^u)$.

3.3 Embedding Layer

We create an embedding matrix $\mathbf{M}^I \in \mathbb{R}^{|I| \times d}$ for items, where d is the latent dimension. A constant zero vector $\mathbf{0}$ is used as the embedding for the padding items. The embedding look-up operation retrieves the previous n items' embeddings, and stacks them together resulting in a matrix $\mathbf{E}^I \in \mathbb{R}^{n \times d}$:

$$\mathbf{E}^I = \begin{bmatrix} m_{s_1} \\ m_{s_2} \\ \dots \\ m_{s_n} \end{bmatrix} \tag{3}$$

Following [22], we use two distinct learnable positional embedding matrices $\mathbf{M}_K^P \in \mathbb{R}^{n \times d}$, $\mathbf{M}_V^P \in \mathbb{R}^{n \times d}$ for keys and values in the self-attention mechanism respectively. This method is more suitable for use in the self-attention mechanism without requiring additional linear transformations [22]. After retrieval, we get the embedding $\mathbf{E}_K^P \in \mathbb{R}^{n \times d}$ and $\mathbf{E}_V^P \in \mathbb{R}^{n \times d}$:

$$\mathbf{E}_K^P = \begin{bmatrix} p_1^k \\ p_2^k \\ \dots \\ p_n^k \end{bmatrix} \qquad \mathbf{E}_V^P = \begin{bmatrix} p_1^v \\ p_2^v \\ \dots \\ p_n^v \end{bmatrix} \tag{4}$$

Similar to the positional embedding, relative time interval embedding matrices are $\mathbf{M}_K^R \in \mathbb{R}^{k \times d}$, $\mathbf{M}_V^R \in \mathbb{R}^{k \times d}$ for keys and values in self-attention. After retrieving the clipped relation matrix $\mathbf{M}^u_{clipped}$, we get the embedding matrix $\mathbf{E}_K^R \in \mathbb{R}^{n \times n \times d}$ and $\mathbf{E}_V^R \in \mathbb{R}^{n \times n \times d}$:

$$\mathbf{E}_K^R = \begin{bmatrix} r_{11}^k & r_{12}^k & \dots & r_{1n}^k \\ r_{21}^k & r_{22}^k & \dots & r_{2n}^k \\ \dots & \dots & \dots & \dots \\ r_{n1}^k & r_{n2}^k & \dots & r_{nn}^k \end{bmatrix} \mathbf{E}_V^R = \begin{bmatrix} r_{11}^v & r_{12}^v & \dots & r_{1n}^v \\ r_{21}^v & r_{22}^v & \dots & r_{2n}^v \\ \dots & \dots & \dots & \dots \\ r_{n1}^v & r_{n2}^v & \dots & r_{nn}^v \end{bmatrix} \tag{5}$$

The two relative time interval embeddings are symmetric matrices, and elements on the main diagonal are all zeros.

3.4 Time Interval-Aware Self-Attention

Inspired by relative position self-attention mechanisms [4, 22], we propose an extension to self-attention to consider the different time intervals between two items in a sequence. However, only considering the time intervals is not enough, because the user interaction sequence might have many instances with the same timestamp. Under this condition, the model would become self-attention without any position or relation information. So, we also consider the position of items in a sequence.

Time Interval-Aware Self-attention Layer: For each input sequence $\mathbf{E}^I = (m_{s_1}, m_{s_2}, \dots, m_{s_n})$ of n items where $m_{s_i} \in \mathbb{R}^d$, compute a new sequence $\mathbf{Z} = (z_1, z_2, \dots, z_n)$, where $z_i \in \mathbb{R}^d$. Each output element, z_i, is computed as a weighted sum of linearly transformed input elements and the relation/position embeddings:

$$z_i = \sum_{j=1}^{n} \alpha_{ij} \left(m_{s_j} W^V + r_{ij}^v + p_j^v \right) \tag{6}$$

where $W^V \in \mathbb{R}^{d \times d}$ is input projection for value.

Each weight coefficient α_{ij} is computed using a softmax function:

$$\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{n} \exp e_{ik}} \tag{7}$$

And e_{ij} is computed using a compatibility function that considers inputs, relations and positions:

$$e_{ij} = \frac{m_{s_i} W^Q \left(m_{s_j} W^K + r_{ij}^k + p_j^k \right)^T}{\sqrt{d}}, \tag{8}$$

where $W^Q \in \mathbb{R}^{d \times d}$, $W^K \in \mathbb{R}^{d \times d}$ are input projections for a query and key respectively. The scale factor $\sqrt{d}$ is used to avoid large values of the inner product, especially when the dimension is high.

Causality: Due to the nature of sequences, the model should only consider the first t items when predicting the $(t+1)^{\text{st}}$ item. However, the t-th output of the time-aware self-attention layer contains all input information. Hence, as in [14], we modify the attention by forbidding all links between $\mathbf{Q}_i$ and $\mathbf{K}_j (j > i)$, where $\mathbf{Q}_i = m_{s_i} W^Q$, $\mathbf{K}_j = m_{s_j} W^K + r_{ij}^k + p_j^k$.

Point-Wise Feed-Forward Network: Though our time interval aware attention layer is able to incorporate all previous items, absolute position, and relative time information with adaptive weights, it does so via a linear combination. After each time-aware attention layer, we apply two linear transformations with a ReLU activation in between, which could endow non-linearity to the model:

$$\text{FFN}(z_i) = \max(0, z_i W_1 + b_1) W_2 + b_2 \tag{9}$$

where $W_1, W_2 \in \mathbb{R}^{d \times d}$ and $b_1, b_2 \in \mathbb{R}^d$. While the linear transformations are the same across different items, they use different parameters from layer to layer.

As discussed in [14], after stacking the self-attention layers and feed-forward layers, more problems will emerge including overfitting, unstable training processes (e.g. vanishing gradients), and requiring more training time. Like [26] and [14], we also adopt layer normalization, residual connections and dropout regularization techniques to solve these problems:

$$\mathbf{Z}_i = z_i + \text{Dropout}(\text{FFN}(\text{LayerNorm}(z_i))) \tag{10}$$

Layer normalization is used to normalize the inputs across features (i.e., zero-mean and unit-variance). According to [1], this can stabilize and accelerate neural network training. Assume $\mathbf{x}$ is a vector containing all features of the sample, layer normalization is defined as:

$$\text{LayerNorm}(\mathbf{x}) = \alpha \odot \frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{11}$$

where $\odot$ is an element-wise product, μ and σ are the mean and variance of $\mathbf{x}$, α and β are learned scaling factors and bias terms.

3.5 Prediction layer

After stacked self-attention blocks, we get the combined representation of items, positions and time intervals. To predict the next item, we employ a latent factor model to compute users' preference score of item i as follows:

$$R_{i,t} = \mathbf{Z}_t \mathbf{M}_i^I \tag{12}$$

where $\mathbf{M}_i^I \in \mathbb{R}^d$ is the embedding of item i and $\mathbf{Z}_t$ is the representation given the first t items (i.e., $s_1, s_2, \dots, s_t$) and their time intervals (i.e., $r^u_{1(t+1)}, r^u_{2(t+1)}, \dots, r^u_{t(t+1)}$) between the $(t+1)$th item.

3.6 Model Inference

Recall that we convert item sequences $(S_1^u, S_2^u, \ldots, S_{|S^u|-1}^u)$ and time sequence $T^u = (T_1^u, T_2^u, \ldots, T_{|T^u|-1}^u)$ to a fixed length sequence $s = (s_1, s_2, \ldots, s_n)$ and $t = (t_1, t_2, \ldots, t_n)$ respectively. We define $o = (o_1, o_2, \ldots, o_n)$ as the expected output given a time and item sequence. Element o_i of o is:

$$o_i = \begin{cases} < \text{pad} > & \text{if } s_i \text{ is a padding item} \\ s_{t+1} & 1 \le t < n \\ S_{|S^u|}^u & t = n \end{cases} \tag{13}$$

where < pad > denotes a padding item. Because user interactions are implicit data, we cannot directly optimize the preference score $R_{i,t}$. The goal of our model is to provide a ranked item list. Hence, we adopt negative sampling to optimize the ranking of items. For each expected positive output o_i, we sample one negative item $o_i' \notin S^u$ to generate a set of pairwise preference orders $D = \{(S^u, T^u, o, o')\}$. We transform model output scores into the range $(0, 1)$ by a *sigmoid* function $\sigma(x) = 1/(1 + e^{-x})$ and adopt binary cross entropy as the loss function:

$$-\sum_{S^u \in S} \sum_{t \in [1,2,\ldots,n]} \left[\log(\sigma(r_{o_t,t})) + \log(1 - \sigma(r_{o_t',t})) \right] + \lambda ||\Theta||_F^2 \tag{14}$$

where $\Theta = \{\mathbf{M}^I, \mathbf{M}_K^P, \mathbf{M}_V^P, \mathbf{M}_K^R, \mathbf{M}_V^R\}$ is the set of embedding matrices, $|| \cdot ||_F$ denotes the *Frobenius* norm, λ is the regularization parameter. Note that we mask the loss of the padding item.

The proposed model is optimized by the *Adam* optimizer [15]. Since each training sample $D = \{(S^u, T^u, o, o')\}$ could be constructed independently, we apply mini-batch SGD to speed up the training efficiency.

4 EXPERIMENTS

In this section, we introduce our experimental setup and present our empirical results. Our experiments are designed to answer the following research questions:

RQ1: Can our proposed method outperform state-of-the-art baselines for sequential recommendation tasks?

RQ2: Which of absolute positions or relative time intervals is more important for sequential recommendation?

RQ3: How do the parameters affect model performance, such as the number of dimensions, the maximum length of sequences and the maximum time intervals we consider?

RQ4: Are personalized time intervals useful in this model?

4.1 Datasets

We evaluate our methods on six datasets from three real world platforms. These datasets have different domains, size and sparsity, and all are publicly available:

- **MovieLens:** A widely used benchmark dataset for evaluating collaborative filtering algorithms. We use the version (MovieLens-1M) that includes 1 million user ratings.
- **Amazon:** A series of datasets introduced in [9, 19], comprising large corpora of product reviews crawled from *Amazon.com*. We consider four categories, 'CDs and Vinyl', 'Movies and TV', 'Beauty' and 'Video Games'. This dataset is highly sparse.

Table 2: Basic dataset statistics.

Dataset	#users	#items	avg. actions /user	#actions
MovieLens-1m	6,040	3,416	163.5	0.987M
Amazon CDs&Vinyl	26,875	42,779	24.35	0.65M
Amazon Movies&TV	40,928	37,564	25.55	1.05M
Amazon Beauty	51,369	19,369	4.39	0.225M
Amazon Game	30,935	12,111	6.46	0.2M
Steam	114,796	8,648	7.58	0.87M

Table 3: Hyperparameter settings.

Dataset	max sequence length	max time intervals	regularization
MovieLens-1m	50	2048	0.00005
Amazon CDs&Vinyl	50	512	0.00005
Amazon Movies&TV	50	512	0
Amazon Beauty	25	512	0.00005
Amazon Game	25	256	0.00005
Steam	25	256	0

- **Steam:** This dataset was crawled from *Steam*, a large online video game distribution platform. It was introduced in [14] and includes information like users' play hours, media score, and developer.

All of these datasets contain the timestamps or specific date of interactions. To preprocess we follow the procedure from [7, 14, 21]. For all datasets, we treat the presence of a review or rating as implicit feedback (i.e., the user interacted with the item) and order the items by timestamps. For all users we subtract the smallest timestamp in their own sequence to let timestamps start from zero. We filter out cold-start users and items with fewer than 5 actions. Following [14], we use the most recent item for testing, the second recent item for validation and the remaining items for training. Dataset statistics are shown in Table 2. MovieLens-1m is the most dense dataset which has the longest average actions and the least users and items. Amazon Beauty and Games have the fewest actions per user.

4.2 Evaluation Metrics

We adopt two common Top-N metrics, Hit Rate@10 and NDCG@10, to evaluate recommendation performance [10]. Hit@10 counts the rates of the ground-truth items among the top 10 items. NDCG@10 considers the position and assigns higher weights to higher positions. Following [16], for each user u, we randomly sample 100 negative items, and rank these items with the ground-truth item. We calculate Hit@10 and NDCG@10 based on the rankings of these 101 items.

Table 4: Recommendation Performance. The best performing method in each row is boldfaced, and the second best method in each row is underlined. Improvements are shown in the last column

Dataset	Metric	Pop	BPR	FPMC	TransRec	GRU4Rec+	Caser	MARank	TiSASRec	Improvement
MovieLens-1m	NDCG@10	0.2389	0.3421	0.3917	0.2488	0.4334	0.5011	0.5199	**0.5706**	9.75%
	Hit@10	0.4386	0.5952	0.6182	0.4478	0.6522	0.7517	0.7652	**0.8038**	5.04%
Amazon CDs&Vinyl	NDCG@10	0.1862	0.3626	0.3355	0.1893	0.2005	0.2285	0.4355	**0.5047**	15.89%
	Hit@10	0.3335	0.5627	0.5122	0.3356	0.3536	0.3865	0.6464	**0.7212**	11.57%
Amazon Movies&TV	NDCG@10	0.2723	0.3482	0.3376	0.2891	0.2987	0.3212	0.4568	**0.4974**	8.89%
	Hit@10	0.4576	0.5537	0.5104	0.4793	0.4848	0.5015	0.6584	**0.7125**	9.21%
Amazon Beauty	NDCG@10	0.1758	0.1523	0.179	0.1704	0.1743	0.1446	0.2127	**0.2818**	32.49%
	Hit@10	0.3215	0.2554	0.2806	0.3016	0.3184	0.2639	0.3472	**0.4345**	25.14%
Amazon Game	NDCG@10	0.23	0.2731	0.341	0.2292	0.2321	0.2661	0.4437	**0.4797**	8.11%
	Hit@10	0.4006	0.4417	0.5222	0.3943	0.3971	0.4474	0.6498	**0.7087**	9.06%
Steam	NDCG@10	0.4136	0.3598	0.4083	0.412	0.4138	0.4787	0.5135	**0.5897**	14.84%
	Hit@10	0.6611	0.5941	0.6017	0.6594	0.6605	0.7135	0.751	**0.8103**	7.9%

4.3 Compared Methods

We compare TiSASRec with the following methods. These methods include classic general recommendation (e.g. POP, BPR) without considering sequential patterns, first-order Markov Chain-based methods (e.g. FPMC, TransRec) and Neural Network (NN) based methods (e.g. GRU4Rec+, Caser, MARank).

- **POP**. All items are ranked by their popularity in all users' training sets, and the popularity is calculated by counting the number of actions.
- **BPR[20]**. Bayesian personalized ranking is a classic method for general item recommendation. Matrix factorization is used as the recommender.
- **FPMC[21]**. This method combines matrix factorization and first-order Markov Chains, which capture long-term preferences and dynamic transitions respectively.
- **TransRec[7]**. This method models each user as a tanslation vector from item to item. It is a first-order method to capture transitions.
- **GRU4Rec+[11]**. Models user action sequences for session-based recommendation. Compared with GRU4Rec, GRU4Rec+ adopts a different loss function and sampling strategy that shows significant improvement on GRU4Rec.
- **Caser [24]**. Embeds a sequence of recent items into an 'image' in the time and latent spaces. This method can capture high-order Markov chains considering the L most recent items.
- **MARank [34]**. A state-of-the-art model that was proposed recently. This method considers the most recent items and applies multi-order attention to capture individual- and union-level item dependency.

SASRec [14] can be viewed as a method which only considers the absolute position and will be discussed and compared with our model in Section 4.6. Other sequential recommendation methods (e.g. PRME [6], HRM [27], Fossil [8], GRU4Rec[12]) have been outperformed by the baselines above, so we omit comparison against them.

For fair comparison, we implement BPR, FPMC, TransRec using *Tensorflow* with an Adam optimizer. For GRU4Rec+, Caser, MARank, we use code provided by the authors. We search latent dimensions in {10, 20, 30, 40, 50}, regularization hyperparameters in {0.0001, 0.001, 0.01, 0.1, 1} and learning rate in $\{0.1, 10^{-2}, \ldots, 10^{-4}\}$. For all other parameters, we use the default settings according to the respective papers. We tune hyper-parameters using the validation set, and terminate training if validation performance doesn't improve for 20 epochs.

4.4 Implementation Details

We implement TiSASRec with *tensorflow* and fine-tune hyperparameters on our validation set. We use two time interval aware self-attention layers, and learned positional and interval embeddings and shared item embeddings in the embedding layer and prediction layer. The learning rate is 0.001, batch size is 128, dropout rate is 0.2 for all datasets. The rest of our parameter settings are shown in Table 3. All experiments are conducted with a single GTX-1080 Ti GPU.

4.5 Recommendation Performance

Table 4 shows the recommendation performance of all methods on the six datasets (**RQ1**). Among the baseline methods, MARank [34] has state-of-the-art performance compared with other baselines. MARank can capture individual-level, union-level user interactions and tackle sparse datasets well. For dense datasets, neural network based methods (i.e., GRU4Rec+, Caser) have obviously better performance than Markov chain-based methods because these methods have stronger ability to capture long-term sequential patterns which is important for dense datasets. Since Markov chain-based methods (i.e., FPMC, TransRec) focus on dynamic transition of items, they perform better on sparse datasets.

TiSASRec improves over the best baseline methods on all dense and sparse datasets with respect to the two metrics. On the one hand, our model takes advantage of the attention mechanism which can adapt weights according to different items, absolute positions and time intervals, while the former models only consider the first two. On the other hand, TiSASRec takes advantage of the attention mechanism, as discussed in [14], it can adaptively attend on items

Table 5: Comparison between time intervals and absolute positions. We highlight the best results by underline except for TiSASRec.

Dataset	Metric	SASRec	TiSASRec-R	TiSASRec
MovieLens	NDCG@10	0.5524	0.5648	0.5706
	Hit@10	0.7929	0.8031	0.8038
CDs&Vinyl	NDCG@10	0.4880	0.4978	0.5047
	Hit@10	0.7001	0.7176	0.7212
Movies&TV	NDCG@10	0.4882	0.4853	0.4974
	Hit@10	0.7035	0.701	0.7125
Beauty	NDCG@10	0.2722	0.2748	0.2818
	Hit@10	0.4185	0.4223	0.4345
Game	NDCG@10	0.4714	0.4711	0.4797
	Hit@10	0.6952	0.6939	0.7087
Steam	NDCG@10	0.5801	0.5731	0.5897
	Hit@10	0.8058	0.7937	0.8103

within different ranges on different datasets. We will discuss the effect of relative time intervals in Section 4.6.

4.6 Relative Time Intervals

Given our motivation for comparing the performance of models which only have absolute positions or relative time intervals, we modify Eqs. 15 and 16 to let TiSASRec only consider the relative time intervals and not consider absolute positions. We modify Eq. 15 as:

$$z_i = \sum_{j=1}^{n} \alpha_{ij} \left(m_{s_j} W^V + r_{ij}^v \right) \tag{15}$$

and modify Eq. 16 as:

$$e_{ij} = \frac{m_{s_i} W^Q \left(m_{s_j} W^K + r_{ij}^k \right)^T}{\sqrt{d}} \tag{16}$$

We remove the absolute position term in the two equations to obtain a new model. We denote this model as TiSASRec-R. We view SASRec [14] as a self-attention method which only considers the absolute position (**RQ2**). We compare SASRec, TiSASRec-R and TiSASRec by applying them on the six datasets discussed above. For fair comparison, we use the same maximum sequence length for SASRec as shown in Table 3. Results are shown in Table 5.

TiSASRec-R has better performance than SASRec on the MovieLens, CDs&Vinyl and Beauty datasets, but not for the other three. Overall, using only relative time intervals will have similar performance. The main reasons that restrict the performance of TiSASRec-R is that there are many identical timestamps in a user sequence (some have only a single timestamp). Under this situation, TiSASRec-R will degrade to self-attention without any positional information. To alleviate this problem, we combine relative time intervals with absolute positions. This way, TiSASRec incorporates richer item relations to compute attention weights.

4.7 Influence of Hyper-parameters

Influence of Latent Dimensionality d. Figure 4 shows NDCG for dimensionality d from 10 to 50 while keeping the other optimal hyperparameters unchanged. In most cases, a larger d leads to better model performance. The value of d has limited influence on TransRec. For the dataset 'Amazon Beauty', Caser, MARank, TransRec have worse performance when d is larger.

Influence of maximum sequence length n. Figure 5 shows the NDCG for maximum length n from 10 to 50 while keeping other optimal hyperparameters unchanged. Performance improves when considering longer sequences and eventually begins to converge. SASRec converges earlier than TiSASRec.

Influence of maximum time intervals k **considered**. Figure 6 shows the NDCG of TiSASRec and TiSASRec-R. We choose time intervals $\{1, 64, 256, 1024, 2048\}$ for the 'MovieLens' dataset and $\{1, 16, 64, 256, 512\}$ for 'Amazon CDs & Vinyl'. We can see that TiSASRec has more stable performance under different maximum intervals. TiSASRec-R achieves its best performance when k is chosen properly and gets worse for larger k. Larger k means more parameters need to be trained.

Table 6: Comparison of three timestamp processing methods (NDCG@10).

Dataset	Method (1)	Method (2)	Method (3)
MovieLens-1m	0.5643	0.5658	**0.5706**
Amazon CDs&Vinyl	0.5001	0.4931	**0.5047**
Amazon Movies&TV	0.4838	0.4826	**0.4974**
Amazon Beauty	0.2692	**0.2825**	0.2818

4.8 Personalized time intervals

In this section, we explore different processing methods of timestamps. We will discuss three kinds of methods:

(1) Directly use timestamps as the features. We will subtract the minimum timestamps in the dataset to let all timestamps start from 0.
(2) Using unscaled time intervals. For every user, we subtract the minimum timestamps in the sequence to let users' timestamps start from 0.
(3) Using personalized time intervals. For every time interval of a user, these intervals are divided by the smallest intervals as stated in Section 3.4.

Table 6 presents the results of different timetamps processing methods. Please note that we don't clip timestamps in the first two methods. We highlight the highest NDCG with boldface. Method (3) (i.e. Personalized time intervals) achieves the best performance in the first three datasets.

4.9 Visualization

We use TiSASRec to predict the next item a user might interact with. We then set all time intervals to be the same to get another prediction. As shown in Figure 7, time intervals influence on prediction results.

Figure 8 shows four heatmaps of average attention weights on the first 256 time intervals. Note that when we calculate the average weight, the denominator is the number of valid weights, so as to

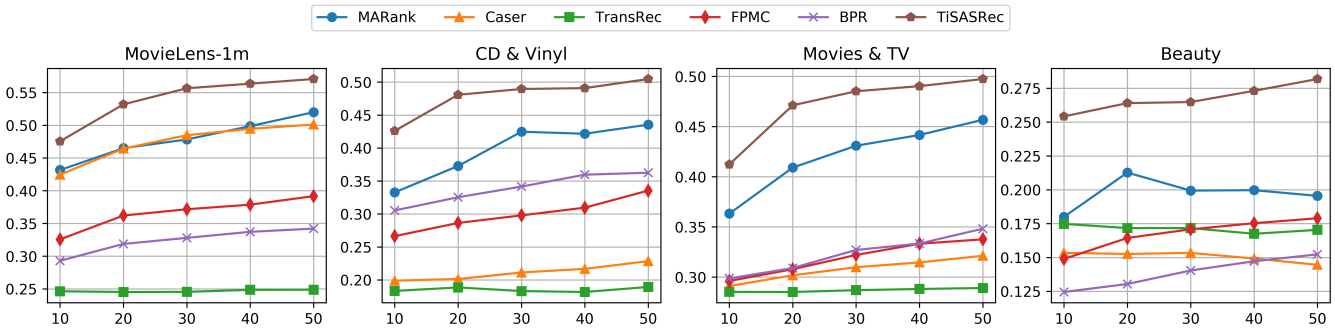


Figure 4: Effect of the latent dimensionality d on ranking performance (NDCG@10).

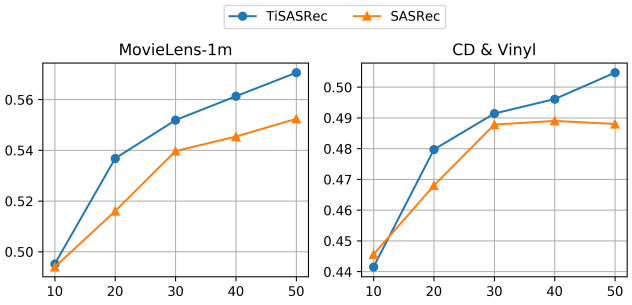


Figure 5: Effect of maximum length n on ranking performance (NDCG@10).

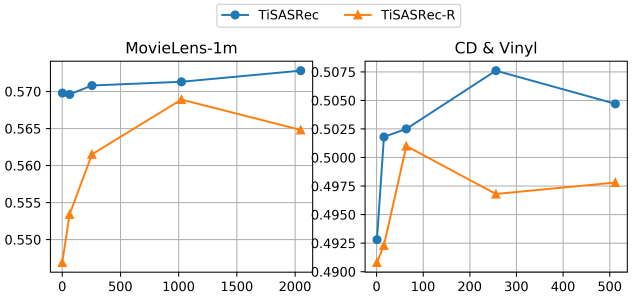


Figure 6: Effect of maximum time intervals k on ranking performance (NDCG@10).

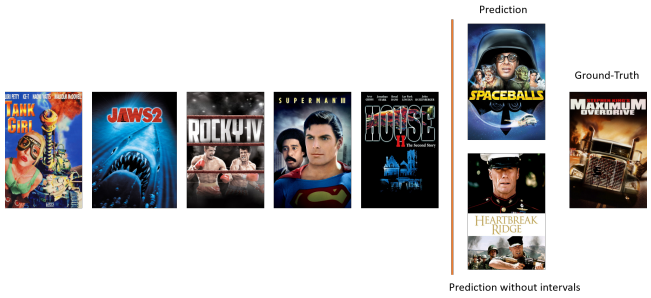
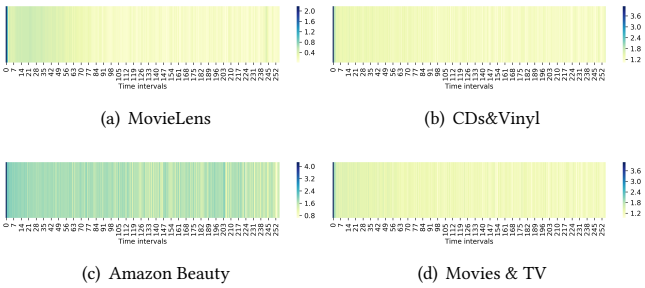


Figure 7: A prediction result of TiSASRec.



(a) MovieLens (b) CDs&Vinyl

(c) Amazon Beauty (d) Movies & TV

Figure 8: Visualizations of average attention weights on time intervals.

avoid the influence of padding items in short sequences. From the four heatmaps, we make the following conclusion:

(1) Small time intervals usually have larger weights than big intervals which means more recent items have more impact on the next items predictions.
(2) Dense datasets (e.g. MovieLens) need larger scope of items than sparse datasets (e.g. CDs&Vinyl), as there's a larger green region on the left of MovieLens heatmap, and a yellow region on the right.
(3) The heatmap of Amazon Beauty dataset doesn't have an obvious green or yellow region, possibly because this dataset doesn't have obvious sequential patterns. This explains that why several sequential methods perform poorly on this dataset.

5 CONCLUSION

In this work, we proposed a time interval aware self-attention model for sequential recommendation (TiSASRec). TiSASRec models the relative time intervals and absolute positions among items to predict future interactions. Extensive experiments on both sparse and dense datasets show that our model outperforms state-of-the-art baselines. We also explore various features of this model. We demonstrated the influence of relative time intervals on next item prediction tasks.

REFERENCES

[1] Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *ArXiv* abs/1607.06450 (2016).

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR* abs/1409.0473 (2014).

[3] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed Huai hsin Chi. 2018. Latent Cross: Making Use of Context in Recurrent Recommender Systems. In *WSDM*.

[4] Ivan Bilan and Benjamin Roth. 2018. Position-aware Self-attention with Relative Positional Encodings for Slot Filling. *ArXiv* abs/1807.03052 (2018).

[5] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Weiwei Liu, and Tat-Seng Chua. 2017. Attentive Collaborative Filtering: Multimedia Recommendation with Item- and Component-Level Attention. In *SIGIR*.

[6] Shanshan Feng, Xutao Li, Yifeng Zeng, Gao Cong, Yeow Meng Chee, and Quan Yuan. 2015. Personalized Ranking Metric Embedding for Next New POI Recommendation. In *IJCAI*.

[7] Ruining He, Wang-Cheng Kang, and Julian J. McAuley. 2017. Translation-based Recommendation. *ArXiv* abs/1707.02410 (2017).

[8] Ruining He and Julian J. McAuley. 2016. Fusing Similarity Models with Markov Chains for Sparse Sequential Recommendation. *2016 IEEE 16th International Conference on Data Mining (ICDM)* (2016), 191–200.

[9] Ruining He and Julian J. McAuley. 2016. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. *ArXiv* abs/1602.01585 (2016).

[10] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. *ArXiv* abs/1708.05031 (2017).

[11] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent Neural Networks with Top-k Gains for Session-based Recommendations. In *CIKM*.

[12] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based Recommendations with Recurrent Neural Networks. *CoRR* abs/1511.06939 (2015).

[13] How Jing and Alexander J. Smola. 2017. Neural Survival Recommender. In *WSDM*.

[14] Wang-Cheng Kang and Julian J. McAuley. 2018. Self-Attentive Sequential Recommendation. *2018 IEEE International Conference on Data Mining (ICDM)* (2018), 197–206.

[15] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014).

[16] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*.

[17] Yehuda Koren. 2010. Collaborative filtering with temporal dynamics. *Commun. ACM* 53 (2010), 89–97.

[18] Qiang Liu, Shu Wu, Diyi Wang, Zhaokang Li, and Liang Wang. 2016. Context-Aware Sequential Recommendation. *2016 IEEE 16th International Conference on Data Mining (ICDM)* (2016), 1053–1058.

[19] Julian J. McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-Based Recommendations on Styles and Substitutes. In *SIGIR*.

[20] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *UAI*.

[21] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized Markov chains for next-basket recommendation. In *WWW*.

[22] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-Attention with Relative Position Representations. In *NAACL-HLT*.

[23] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-To-End Memory Networks. In *NIPS*.

[24] Jiaxi Tang and Ke Wang. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. *ArXiv* abs/1809.07426 (2018).

[25] Trinh Xuan Tuan and Tu Minh Phuong. 2017. 3D Convolutional Networks for Session-based Recommendation with Content Features. In *RecSys*.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *NIPS*.

[27] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2015. Learning Hierarchical Representation Model for NextBasket Recommendation. In *SIGIR*.

[28] Shoujin Wang, Liang Hu, Longbing Cao, Xiaoshui Huang, Defu Lian, and Wei Liu. 2018. Attention-Based Transactional Context Embedding for Next-Item Recommendation. In *AAAI*.

[29] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J. Smola, and How Jing. 2017. Recurrent Recommender Networks. In *WSDM*.

[30] Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. 2017. Attentional Factorization Machines: Learning the Weight of Feature Interactions via Attention Networks. In *IJCAI*.

[31] Liang Xiong, Xiao Dong Chen, Tzu-Kuo Huang, Jeff G. Schneider, and Jaime G. Carbonell. 2010. Temporal Collaborative Filtering with Bayesian Probabilistic Tensor Factorization. In *SDM*.

[32] Kelvin Xu, Jimmy Ba, Jamie Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *ICML*.

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

[34] Lu Yu, Chuxu Zhang, Shangsong Liang, and Xiangliang Zhang. 2019. Multi-Order Attentive Ranking Model for Sequential Recommendation. In *AAAI*.

[35] Chenyi Zhang, Ke Wang, Hongkun Yu, Jianling Sun, and Ee-Peng Lim. 2014. Latent Factor Transition for Dynamic Collaborative Filtering. In *SDM*.