Deep Self-Attention for Sequential Recommendation

Beichuan Zhang, Zhijiao Xiao* and Shenghua Zhong
College of Computer Science and Software Engineering, Shenzhen University, China
zhangbeichuan2019@email.szu.edu.cn, {cindyxzj, csshzhong}@szu.edu.cn

***Abstract*—Sequential recommendation aims to recommend the next item that a user will likely interact with by capturing the useful sequential patterns from users' historical behaviors. Recently, it has become an important and popular component in various e-commerce platforms. As a successful network, Transformer has been widely used to adaptively capture the dynamics of users' historical behaviors for sequential recommendation. In recommender systems, the size of embedding is usually set to be small. Under small embedding, the dot-product in Transformer may have the limitation on calculating the complex relevance between keys and queries. To address the common but neglected issue, in this paper, we present a new model, Deep Self-Attention for Sequential Recommendation (DSASrec), which proposes a chunking deep attention to compute attention weights. The chunking deep attention has two modules: a deep module and a chunking module. The deep module is used to improve the nonlinearity of the attention function. The chunking module is used to calculate attention weights several times like the multi-head attention in Transformer. Extensive experiments on three benchmark datasets show that our model can achieve state-of-the-art results. Our implementation is available in PyTorch [1].**

***Keywords*—Recommender System, Transformer, Dot-product, Chunking Representation, Deep Learning**

I. Introduction

Recommender system has become an important prevalent component in real-world applications. Learning the embeddings of users and items is an essential topic in recommender systems [1]–[4]. Beyond using the embeddings of users, sequential recommendation considers the sequential patterns in users' historical behaviors as the pre-existing features of users. To exploit the sequential patterns, Transformer [5] has been widely deployed to sequential recommendation. For example, SASrec [6] tried to capture the dynamics of users' historical behaviors via Transformer instead of using Recurrent Neural Networks (RNNs). BERT4rec [7] introduced a deep bidirectional sequential self-attention model and a Cloze objective to the field of recommender systems.

Although the previous methods have been proven effective, the previous methods fail to consider the dot-product in Transformer may have the limitation on calculating the complex relevance between keys and queries. In most cases, Transformer is used in the field which has high dimensional embeddings. For example, Transformer paper and GPT [8] paper used 512-dimensional vectors and 768-dimensional vectors to represent words, respectively. In recommender systems, the size of embedding is usually set to be small. For example, SASrec used 50-dimensional vectors to represent items. NCF [9] used 64-dimensional vectors to represent items and users. The size of embedding vector has a great influence on the limitation of dot-product [9]. Suppose we have a user-item interaction graph $\mathcal{G}$ as Figure 1(left). There are the similarity relations between u_1, u_2 and u_3 as $s_{23} > s_{12} > s_{13}$, where s_{ab} indicates the similarity of user a and user b. When we project the users into 2D space, the geometric relations of u_1, u_2 and u_3 can be expressed by dot-product as in Figure 1(right). There are other similarity relations about u_4 as $s_{41} > s_{43} > s_{42}$. However, the relations $s_{41} > s_{43} > s_{42}$ can't be expressed accurately in the 2D space. If we place v_4 closest to v_1 as the red vectors in Figure 1, v_4 is closer to v_2 than v_3. It would contradict $s_{41} > s_{43} > s_{42}$. Thus, under small embeddings, using dot-product may lead to that the complex relevance between keys and queries is ignored.

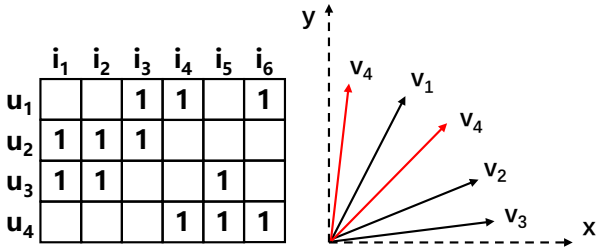

Fig. 1. An example of dot-product's limitation. Since u_4 is most similar to u_1, u_4 only can be piloted as the red vectors. However, no matter how u_4 is piloted, it can not satisfy this relationship: $s_{41} > s_{43} > s_{42}$.

To address the aforementioned problems, in this paper, we present a new model, Deep Self-Attention for Sequential Recommendation (DSASrec). DSASrec takes a user's historical behaviors and a candidate item as input and outputs the user's preference for the candidate item. Specifically, we first project users' historical behaviors into vector representations and then apply a self-attention mechanism to predict users' preferences. Distinct from existing works [6], [10], we propose a chunking deep attention (CDA) to compute attention weights. The chunking deep attention has two modules: a deep module and a chunking module. The deep module in CDA is used to improve the nonlinearity of attention function. The chunking module in

[1]https://github.com/Book1996/DSASrec
DOI reference number: 10.18293/SEKE2021-035

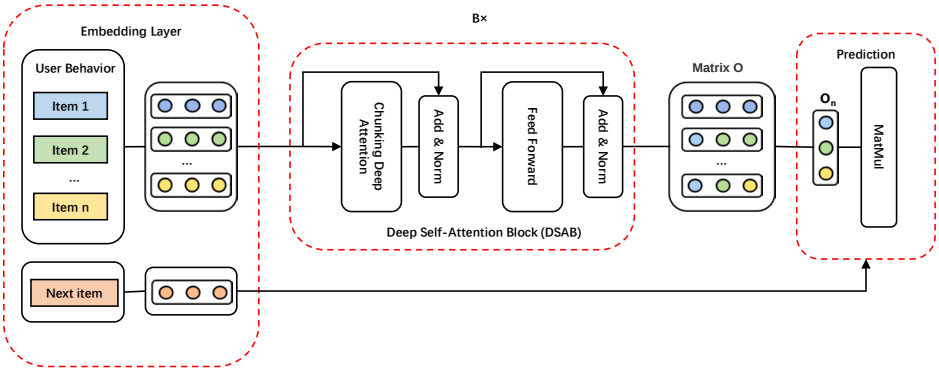

Fig. 2. The network architecture of DSASrec. Matrix O indicates the output of DSAB. O_n indicates the $n-th$ row of O, which combines information of n items. O_n is used to compute the user's preference for the $(n+1)-th$ (next) item.

CDA is used to calculate attention weights several times like the multi-head attention in Transformer. We perform extensive experiments on four standard large real-world datasets, and the results show our model can achieve state-of-the-art results. To justify the designs in our model, we further conduct ablation studies on DSASrec. The results show each component of DSASrec has contributions to performance. To summarize, this work makes the following main contributions:

- We point out the dot-product in Transformer may have the limitation on calculating the relevance of keys and queries when the size of embedding is set to be small.
- We propose a new model DSASrec, which applies chunking deep attention instead of the multi-head attention in Transformer to model attention weights.
- We demonstrate our proposed can achieve state-of-the-art results by extensive experiments on four standard large real-world datasets.

II. Relate Work

A. Attention Mechanism

Attention mechanism can be described as a weighted sum of values, where the weights assigned to each value are computed by a compatibility function [5]. Attention mechanism has become more and more popular in various tasks such as recommender system [6],machine translation [11] and Multimedia [12]–[14]. Recently, Transformer [5] was proposed and achieved promising empirical results in machine translation. Due to the efficiency of Transformer, substantial research focuses on improving the performance of Transformer. For example, Transformer-XL [15] introduced a segment-level recurrence mechanism and a novel positional encoding scheme to learn sequential dependency beyond a fixed-length without disrupting temporal coherence. Transformer-XL learned dependency that is 80% longer than RNNs and 450% longer than vanilla Transformers. Reformer [16] replaced dot-product attention by using locality-sensitive hashing and reduced the complexity of dot-product. Synthesizer [17] proved that using dot-product to learn attention weights from token-token (query-key) interactions was useful but not that important.

B. Sequential Recommendation

Most early researches in sequential recommendation use Markov Chains (MCs) to estimate users' preference for items. FPMC [18] combined Matrix Factorization (MF) and MC for each user owning a personalized transition matrix. Extensive experiments showed FPMC could outperform MF and MCs. Fossil [19] fused similarity-based methods with MC to tackle sparsity issues and the long-tailed distribution of datasets. With progress in deep learning, Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) based methods have proliferated. Since the MC-based methods were difficult to consider all users' historical behaviors, GRU4Rec [20] introduced a ranking loss function and used Gated Recurrent Unit (GRU) [21] to exploit users' historical behaviors. Caser [22] argued that not all adjacent actions had dependency relationships. Hence, they proposed a CNN-based model that regarded users' historical behaviors in the latent space as an "image". Inspired by Transformer [5], several models were proposed to adaptively capture the heterogeneous, polysemous relationship between items in dynamic sequence for sequential recommendation. For example, SASRec [6] balanced long-term pattern and predictions based on relatively several previous actions via Transformer. SASrec is over ten times faster than RNN and CNN-based methods with GPUs and achieved state-of-the-art results. TiSASRec [10] combined the advantages of absolute position and relative time intervals to learn the weights of different items. BERT4rec [7] employed the deep bidirectional self-attention to model user behavior sequences. SSE-PT [23] introduced additional personalized embeddings to improve the performance of Transformer for sequential recommendation. Although the previous methods have been proven effective, they ignore the limitation of dot-product when the size of embedding vector is small.

III. Proposed Method

In this section, we present the architecture of DSASrec, which concludes an embedding layer, stacked deep self-attention blocks, and a prediction layer as Figure 2.

A. Problem Formulation

In sequential recommendation, let $U = \{u_1, u_2, ..., u_{|U|}\}$ be a set of users, $I = \{i_1, i_2, ..., i_{|I|}\}$ be a set of items, and $S^u = \{s_1^u, s_2^u, ..., s_{|S^u|}^u\}$ be a historical interaction sequence for a user $u \in U$, where $s_t^u \in I$ is the item that u has interacted with at time step t. Given the interaction history S^u, the sequential recommendation seeks to predict the next item that user u will interact with.

B. Embedding Layer

Following the prior works [6], [10], we firstly transform the sequence $S^u = \{s_1^u, s_2^u, ..., s_{|S^u|}^u\}$ into a fixed-length sequence $\{s_1^u, s_2^u, ..., s_N^u\}$, where N is a hyper-parameter meaning maximum sequence length. If the length of S^u is less than N, we add zero-paddings to the left side. If the length of S^u is greater than N, we only consider the most recent N interactions. Then, we create an item embedding matrix $M \in \mathbb{R}^{|I| \times d}$ and apply a lookup layer on M to transform S^u into vector representations, where d is latent dimensionality. We inject a learnable position embedding $P \in R^{N \times d}$. Thus, the input embedding E^u corresponding with S^u is defined as:

$$E^u = \begin{bmatrix} M_{s_1^u} + P_1 \\ M_{s_2^u} + P_2 \\ ... \\ M_{s_N^u} + P_N \end{bmatrix}. \tag{1}$$

C. Deep Self-Attention Block

1) Deep Module: An attention function can be described as mapping a query and a set of key-value pairs to a weighted sum of the values. In Transformer, the attention function is defined as:

$$Attention(Q, K, V) = \mathrm{S}oftmax(\frac{QK^T}{\sqrt{d}})V, \tag{2}$$

where $Q \in \mathbb{R}^{N \times d}, K \in \mathbb{R}^{N \times d}, V \in \mathbb{R}^{N \times d}$. N is the length of sequence length and d is latent dimensionality as Section III-B shows. To address the limitation of dot-product, our method replaces the matrix multiplication term QK^T with a multi-layer perceptron (MLP). The deep attention can be defined as:

$$Deep\ Attention(Q, K, V) = \mathrm{S}oftmax(MLP(QK))V, \tag{3}$$

where $QK \in \mathbb{R}^{N \times N \times d}$ is the matrix that includes all key-query pairs and MLP: $\mathbb{R}^{N \times N \times d} \to \mathbb{R}^{N \times N}$. The element of QK is defined as:

$$QK_{mn} = Q_m || K_n, \tag{4}$$

where Q_m is $m-th$ row of Q, K_n is $n-th$ row of K and $||$ is the concatenation operation. Recall that we add zero-paddings to the left side, if the length of S^u is less than N. And our model should consider only the first h items when predicting the $(h+1)-th$ item. Thus, some key-query pairs are useless and should not be used to compute corresponding weights. Formally, there are three cases that would let QK_{mn} be an useless key-query pair: Case 1: $m < n$; Case 2: Q_m represents pad item; Case 3: K_n represents pad item. In order to alleviate the computing cost of MLP, we highly optimize our code to no longer use these useless key-query pairs.

2) Chunking Module: Transformer has found it beneficial to linearly project the queries, keys and values several times with different, learned linear projections. We keep the multi-head mechanism to disentangle different information from representation by splitting the queries, keys and values evenly into P chunks as follows:

$$\begin{aligned} K &= (K^1; K^2; ...; K^P), \\ Q &= (Q^1; Q^2; ...; Q^P), \\ V &= (V^1; V^2; ...; V^P), \end{aligned} \tag{5}$$

where K^j, Q^j, and $V^j \in \mathbb{R}^{N \times \frac{d}{P}}$. The chunking deep attention (CDA) are defined as:

$$\begin{aligned} &CDA(Q, K, V) = Concat(head_1, ..., head_h), \\ &where\ head_j = Deep\ Attention(Q^j, K^j, V^j). \end{aligned} \tag{6}$$

3) Self-Attention: Recently, a self-attention method was proposed which used the same objects as queries, keys, and values [5]. Following previous methods [6], [10], we apply the self-attention mechanism to capture the sequential patterns in users' historical behaviors. The deep self-attention (DSA) in our case is defined as follows:

$$DSA(E^u) = CDA(E^u, E^u, E^u). \tag{7}$$

Following Transformer, we employ two position-wise fully connected feed-forward networks (FFN) with a ReLU activation in between to strengthen the performance of DSA as follows:

$$\begin{aligned} A &= LayerNorm(E^u + Dropout(DSA(E^u))), \\ A' &= ReLU(AW_1 + b_1)W_2 + b_2, \\ Z_1^u &= LayerNorm(A + Dropout(A')), \end{aligned} \tag{8}$$

where $W_1 \in \mathbb{R}^{d' \times d}, b_1 \in \mathbb{R}^{d'}, W_2 \in \mathbb{R}^{d \times d'}$, $b_2 \in \mathbb{R}^d$ are model parameters, d' is a hyper-parameter and $Z_1^u \in \mathbb{R}^{N \times d}$ indicates the output of the first deep self-attention block. For the sake of simplicity, we define the entire deep self-attention block (DSAB) as follows:

$$Z_1^u = DSAB(E^u). \tag{9}$$

We stack DSABs to capture more complex feature transition. The $b-th$ DSAB is defined as:

$$Z_b^u = \begin{cases} DSAB(E^u) & \text{b = 1}, \\ DSAB(Z_{b-1}^u) & \text{b > 1}. \end{cases} \tag{10}$$

D. Prediction Layer

After an embedding layer and B deep self-attention blocks, the behaviors sequence S^u is transformed into $Z_B^u \in \mathbb{R}^{N \times d}$. Following the prior works [6], [10], we calculate user u's preference for item $o \in I$ through a dot-product operation as follows:

$$y_{o,j}^u = z_j M_o^T. \tag{11}$$

where z_j denotes the $j-th$ row of Z_B^u, M_o denotes the embedding of item o and $y_{o,j}^u$ denotes the possibility of item o being the $(j+1)-th$ item for user u given the previous j items.

E. Model Training

Following previous methods [6], [23], we adopt a binary cross entropy loss to optimize DSASrec, which is defined as:

$$-\sum_{u \in U} \sum_{1 \le j < N} \left[log(\sigma(\hat{y}_{o_j,j}^u) + \sum_{o' \notin S^u} log(\sigma(1 - \hat{y}_{o',j})) \right], \tag{12}$$

where σ is Sigmoid function, o' indicates a negative item and o_j indicates an expected item. It is worth noting that when our model inputs a sequence $\{s_1^u, s_2^u, ..., s_{|S^u|-1}^u\}$ and its expected output is

Table I. The statistics of the datasets

Dataset	Users	Items	avg. actions /user	avg. actions /item	Samples
Beauty	52,240	57,289	6.9	7.6	0.4M
Games	31,013	23,715	12.1	9.3	0.3M
Steam	334,730	13,047	11.0	282.5	3.7M
ML-1M	6,040	3,416	163.5	289.1	1.0M

a 'shifted' version of the same sequence: $\{s_2^u, s_2^u, ..., s_{|S^u|}^u\}$ during training process. Thus, o_j can be defined as:

$$o_j = \begin{cases} 0 & \text{if } S_j^u \text{ is a padding item} \\ S_{j+1}^u & \text{if } 1 \leq j < N \end{cases} \tag{13}$$

For alleviating calculation, as in [6], [23], we randomly generate one negative item o' for each of expected items.

IV. EXPERIMENTS

A. Experimental Setup

1) Compared Methods: To show the effectiveness of our approach, we compare it with the following state-of-the-art methods:

- **GRU4Rec+ [24]**. GRU4Rec is an RNN based approach for recommendations, which introduced a novel ranking loss function and GRU in sequential recommendation.
- **Caser [22]**. This method viewed the sequence of recent items as an "image" and used convolutional filters to learn sequential patterns as obtained local features.
- **SASRec [6]**. This method applied Transformer to balance long-term pattern and predictions based on relatively several previous actions.
- **TiSASrec [10]**. This method combined the advantages of absolute position and relative time intervals to learn attention weight.
- **BERT4rec [7]**. This method employed the deep bidirectional self-attention to model user behavior sequences.
- **SSE-PT [23]**. This method introduced additional personalized embeddings to improve the performance of Transformer model for sequential recommendation.

2) Dataset: We evaluate our method on four datasets from three real-world platforms. The four datasets are exactly as same as SASrec used. We request the three datasets from the SASrec. These public datasets have different domains, sizes, and sparsity. In the prepossessing stage, we closely follow the common procedure from SASrec. For all datasets, we treat a review or rating for an item as implicit feedback, and we filter out cold-start users and items with fewer than 5 interactions. Each dataset is split into two parts: (1) the most recent action for testing, (2) all remaining actions for training. The statistics of the dataset are shown in Table I.

- **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 were introduced in [25]. Followed by the existing work [6], we consider two categories: 'Beauty' and 'Games.'
- **Steam**: A dataset is crawled from Steam by SASrec [6]. It includes rich information like users' play hours, pricing information, media scores, categories, and developers.

3) Evaluation Metrics: To evaluate recommendation, we use the same protocols as previous methods [6]: Hit@10 and NDCG@10. In the testing phase, for each user, we randomly sample 100 items and rank these items with the most recent action.

Table II. The recommendation results. Bold scores are the best and underlined scores are the second best.

Methods	Metrics	Beauty	Games	Steam	ML-1M
GRU4REC+	Hit@10	0.3949	0.6599	0.8018	0.7501
	NDCG@10	0.2556	0.5282	0.5595	0.5513
CASER	Hit@10	0.4264	0.5282	0.7874	0.7886
	NDCG@10	0.2547	0.3214	0.5381	0.5538
SASrec	Hit@10	0.4852	0.7412	0.8716	0.8132
	NDCG@10	0.3211	0.5633	0.6211	0.5842
TiSASec	Hit@10	0.4629	0.7323	0.8657	0.8125
	NDCG@10	0.3016	0.5437	0.6228	0.5711
BERT4rec	Hit@10	0.4952	0.7499	0.8755	0.8266
	NDCG@10	0.3311	0.5566	0.6315	0.6004
SSE-PT	Hit@10	<u>0.5028</u>	<u>0.7634</u>	<u>0.8764</u>	<u>0.8288</u>
	NDCG@10	<u>0.3370</u>	<u>0.5622</u>	<u>0.6378</u>	<u>0.6122</u>
DSASrec	Hit@10	**0.5341**	**0.7826**	**0.8803**	**0.8294**
	NDCG@10	**0.3645**	**0.5672**	**0.6416**	**0.6138**

Table III. The computing speed(s) for one epoch. Underlined scores indicates the computation cost of DSASrec is 30% higher than that of SASrec.

Dimension	Datasets	Beauty	Games	Steam	ML-1M
50	DSASrec	24	13	70	4
	SASrec	23	13	69	4
100	DSASrec	24	17	110	6
	SASrec	23	14	91	6
150	DSASrec	27	19	154	7
	SASrec	24	15	120	6
200	DSASrec	<u>26</u>	22	<u>200</u>	7
	SASrec	34	20	150	7

4) Parameter Settings: The proposed DSASRec is implemented on PyTorch, we use two deep self-attention blocks, and each deep self-attention block contains three network layers. The number of neurons of each layer in MLP is 80, 60, 1, respectively. Item embeddings in the embedding layer and prediction layer are shared. The optimizer is the Adam optimizer from [26]. The learning rate is 0.001; batch size is 64; the dropout rate is 0.2 for ML-1m and the Steam, 0.3 for the other datasets. By following the existing work [6], the maximum sequence length N is set to 200 for ML-1m and 50 for the other three datasets. For all methods, the embedding size d is searched in {30, 40, 50, 60}. The learning rate is searched in {0.005, 0.001, 0.0005, 0.0001}. The coefficient λ of L2 regularization term is tuned in $\{10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$.

B. Performance Comparison

Table II presents the recommendation performances of state-of-the-art methods on the four datasets regarding Hit@10 and NDCG@10. Table III presents the running time(s) of SASrec and DSASrec for one epoch. The main observations are as follows:

- DSASrec achieves the best results on all datasets. In particular, DSASrec significantly performs better on sparse datasets (e.g., Beauty, Game), where DSASrec's relative improvements over the strongest baselines w.r.t. Hit@10 are 6.22%, 2.51% in Beauty, Games. These results show the high effectiveness of DSASrec.
- Compared with SASrec, DSASrec significantly performs better on sparse datasets (e.g., Beauty, Game). This comparison shows that compared with the dot-product operation, deep neural networks may need fewer data to learn transition between items.

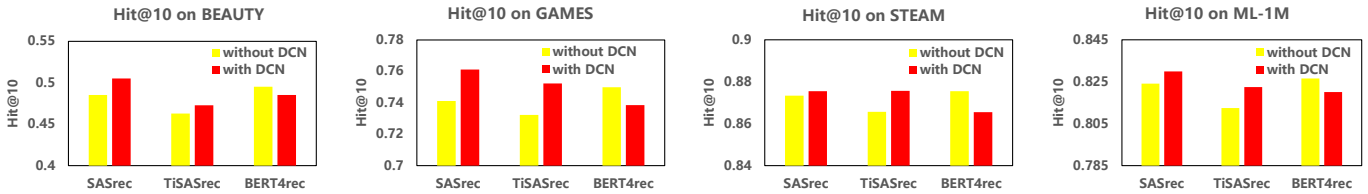

Fig. 3. Performance of SASrec, TiSASrec and BERT4rec w.r.t the scalability of CDA on HR@10.

Table IV. Performance comparison of DSASrec w.r.t. number of layers in MLP. Bold scores are the best and underlined scores are the second best.

Layer#	Metrics	Beauty	Games	Steam	ML-1M
1 layer	Hit@10	0.5141	0.7608	0.8634	0.8094
	NDCG@10	0.3536	0.5583	0.6204	0.5632
2 layers	Hit@10	0.5305	0.7649	0.8686	0.8165
	NDCG@10	0.3622	0.5604	0.6252	0.5957
3 layers	Hit@10	**0.5305**	**0.7734**	**0.8803**	**0.8294**
	NDCG@10	**0.5341**	**0.5672**	**0.6416**	**0.6138**

On the other hand, the dot-product operation is more likely to overfitting when a dataset is sparse. As discussed in [27], product features are more suitable for memorization, and deep neural networks can generalize better.

- As discussed in III-C1, we avoid computing useless weights to alleviate the computing cost of MLP, such as the weights for pad items. Thus, the computational complexity of the attention function in our model is $O(Cd^2/P)$, where d is the latent dimensionality, C is the number of valid key-query pairs and P is the number of chunks. The computational complexity of the attention function in SASrec is $O(N^2d)$, where N is the maximum sequence length. Table III shows the computational complexity of attention function in SASrec and DSASrec in practice. The results show that although CDA generates some additional computing costs compared with dot-product, in most cases, the computing cost of DSASrec is not 30% higher than that of SASrec.

C. Impact of Limitation of Dot-product

Table IV shows the results of DSASrec w.r.t different depths of MLP in CDA. Figure 4 shows the variances of attention weights learned by SASrec and DSASrec. The main observations are as follows:

- Compared with the dot-product, MLP can learn more sophisticated features. Thus, the limitation of the dot-product in Transformer may be related to nonlinear ability. To verify this, we conduct DSASrec with different depths of MLP, because MLP with different depths has different nonlinear ability. Table IV shows the results of DSASrec w.r.t different depths of MLP. The results show that stacking network layers from 1 to 3 can boost performance. Overfitting issues emerge in Beauty when we stack four network layers. This may demonstrate the representation ability of attention function plays a role in recommendation performance.
- Figure 4 shows the variances of attention weights learned by SASrec are lower than that of DSASrec. These results may indicate the attention weights learned by DSASrec are more decentralized and diversified and CDA can learn more the pluralistic relevance between keys and queries.

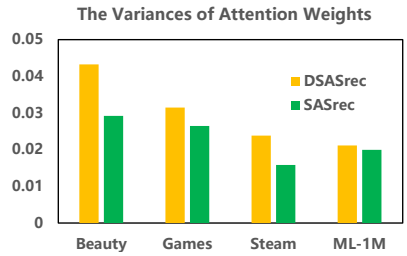

Fig. 4. The variances of attention weights learned by SASrec and DSASrec.

Table V. Performance of DSASrec and its variants. Bold scores are the best and underlined scores are the second best.

Architecture	Metric	Beauty	Games	Steam	ML-1M
Default	Hit@10	**0.5341**	**0.7826**	**0.8803**	**0.8294**
	NDCG@10	**0.3645**	**0.5779**	**0.6416**	**0.6138**
Remove PE	Hit@10	0.5205	0.7758	0.8456	0.8115
	NDCG@10	0.3558	0.5689	0.6047	0.5888
Remove FFN	Hit@10	0.5159	0.7677	0.8551	0.8159
	NDCG@10	0.3382	0.5586	0.5743	0.5850
Remove Dropout	Hit@10	0.5053	0.7526	0.8635	0.8163
	NDCG@10	0.3416	0.5521	0.6337	0.5875

D. Ablation Studies

We perform ablation studies on DSASrec to show how the components of DSASrec affect performance. Table V shows the results of DSASrec and its variants on three datasets.

- **Remove PE (Positional Embedding)**: DSASrec without PE achieves poor results on all datasets. This indicates the order information is important to learn the sequential patterns for sequential recommendation.
- **Remove Dropout**: On all datasets, this variant is significantly worse than the default model. This shows the dropout effectively alleviates overfitting problems in DSASrec.
- **Remove FFN**: We apply FFN to considers relationship between elements in vectors. DSASrec without FFN achieves poor results. Modeling the relationship between elements in vectors can lead to better performance.

E. Scalability of Chunking Deep Attention

Transformer has been widely used to adaptively captures the dynamics of the sequential patterns for sequential recommendation. To show the scalability of CDA, we apply CDA to SASrec, TiSASrec, and BERT4rec. Figure 3 shows SASrec+CDA and TiSASrec+CDA can achieve better performance than SASrec, TiSASrec. But BERT4rec+CDA achieves poor performance than BERT4rec. Compared with BERT4rec, SASrec and TiSASrec have relatively simple structures. The results show CDA may be more suitable for a relatively simple method that uses Transformer.

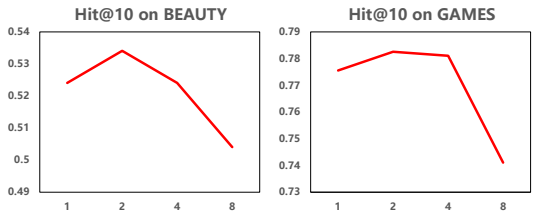

Fig. 5. Performance of DSASrec w.r.t different numbers of chunks on Beauty and Games.

F. Number of Chunks

To study the influence of chunks number, we vary the number of the chunks of queries, keys, and values in the range of $\{1, 2, 4, 8\}$ and show the performance on Beauty and Games datasets in Figure 5 (results on other dataset show similar trends which are omitted for space). Increasing the number of chunks from 1 to 2 leads to better performance. However, the recommendation performance drops when the chunk number increases from 2 to 8. This suggests that the DSASrec suffers from too fine-grained chunks.

V. Conclusions

In this paper, we propose a novel self-attention based method for Sequential Recommendation named DSASrec. It proposes a chunking deep attention (CDA) to computing the attention weights. CDA is used to alleviate the limitation of dot-product in Transformer. Experimental results on four real-world datasets show that DSASrec outperforms state-of-the-art techniques and each component of DSASrec has contributions to performance. In section IV-C, we demonstrate the performance of attention-based models can be improved by enhancing the nonlinearity of attention function. The variances of attention weights learned by dot-product attention are lower than the attention weights learned by CDA. In the future, we plan to extend the model by incorporating auxiliary information (e.g. action types, item knowledge, etc.).

Acknowledgment

This work was supported by the National Natural Science Foundation of China [No. 62002230], the Natural Science Foundation of Guangdong Province [No. 2019A1515011181], the Science and Technology Innovation Commission of Shenzhen under Grant [No. JCYJ20190808162613130], and the Shenzhen high-level talents program.

References

[1] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009. I

[2] Y. Wang, K. Shi, and Z. Niu, "A session-based job recommendation system combining area knowledge and interest graph neural networks," in *The 32nd International Conference on Software Engineering and Knowledge Engineering, SEKE 2020, KSIR Virtual Conference Center, USA, July 9-19, 2020*, 2020, pp. 489–492. I

[3] R. Song, T. Li, X. Dong, and Z. Ding, "Identifying similar users based on metagraph of check-in trajectory data," in *The 32nd International Conference on Software Engineering and Knowledge Engineering, SEKE 2020, KSIR Virtual Conference Center, USA, July 9-19, 2020*, 2020, pp. 525–531. I

[4] N. Mu, D. Zha, L. Zhao, and R. Gong, "Collaborative denoising graph attention autoencoders for social recommendation," in *The 32nd International Conference on Software Engineering and Knowledge Engineering, SEKE 2020, KSIR Virtual Conference Center, USA, July 9-19, 2020*, 2020, pp. 519–524. I

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008. I, II-A, II-A, II-B, III-C3

[6] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2018, pp. 197–206. I, I, II-A, II-B, III-B, III-C3, III-D, III-E, III-E, IV-A1, IV-A2, IV-A2, IV-A3, IV-A4

[7] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang, "Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer," in *Proceedings of the 28th ACM international conference on information and knowledge management*, 2019, pp. 1441–1450. I, II-B, IV-A1

[8] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018. I

[9] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 173–182. I

[10] J. Li, Y. Wang, and J. McAuley, "Time interval aware self-attention for sequential recommendation," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020, pp. 322–330. I, II-B, III-B, III-C3, III-D, IV-A1

[11] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014. II-A

[12] J. Lin and S. Zhong, "Bi-directional self-attention with relative positional encoding for video summarization," in *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020*. IEEE, 2020, pp. 1161–1166. II-A

[13] S. Zhong, A. Fares, and J. Jiang, "An attentional-lstm for improved classification of brain activities evoked by images," in *Proceedings of the 27th ACM International Conference on Multimedia, MM 2019, Nice, France, October 21-25, 2019*. ACM, 2019, pp. 1295–1303. II-A

[14] S. hua Zhong, Y. Liu, T.-Y. Ng, and Y. Liu, "Perception-oriented video saliency detection via spatio-temporal attention analysis," *Neurocomputing*, vol. 207, pp. 178–188, 2016. II-A

[15] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019. II-A

[16] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," *arXiv preprint arXiv:2001.04451*, 2020. II-A

[17] Y. Tay, D. Bahri, D. Metzler, D.-C. Juan, Z. Zhao, and C. Zheng, "Synthesizer: Rethinking self-attention in transformer models," *arXiv preprint arXiv:2005.00743*, 2020. II-A

[18] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized markov chains for next-basket recommendation," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 811–820. II-B

[19] R. He and J. McAuley, "Fusing similarity models with markov chains for sparse sequential recommendation," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 191–200. II-B

[20] T. Donkers, B. Loepp, and J. Ziegler, "Sequential user-based recurrent neural network recommendations," in *Proceedings of the Eleventh ACM Conference on Recommender Systems*, 2017, pp. 152–160. II-B

[21] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014. II-B

[22] J. Tang and K. Wang, "Personalized top-n sequential recommendation via convolutional sequence embedding," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 2018, pp. 565–573. II-B, IV-A1

[23] L. Wu, S. Li, C.-J. Hsieh, and J. Sharpnack, "Sse-pt: Sequential recommendation via personalized transformer," in *Fourteenth ACM Conference on Recommender Systems*, 2020, pp. 328–337. II-B, III-E, III-E, IV-A1

[24] B. Hidasi and A. Karatzoglou, "Recurrent neural networks with top-k gains for session-based recommendations," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018, pp. 843–852. IV-A1

[25] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, "Image-based recommendations on styles and substitutes," in *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, 2015, pp. 43–52. IV-A2

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. IV-A4

[27] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir *et al.*, "Wide & deep learning for recommender systems," in *Proceedings of the 1st workshop on deep learning for recommender systems*, 2016, pp. 7–10. IV-B