Advanced Self-Attentive Sequential Recommendation

Trang Dang Duc Tin*, Pham Quoc Buu†, Duong Thanh Trieu‡, and Ngo Quang Minh§
* 22125106, 22APCS1, HCMUS
Email: trangdangductin@gmail.com
†22125012, 22APCS1, HCMUS
Email: pphambuu@gmail.com
‡22125112, 22APCS1, HCMUS
Email:duongthanhtrieu6@gmail.com
§22125057, 22APCS1, HCMUS
Email:nqominh219@gmail.com

***Abstract*- Nowadays, Recommender Systems are widely used in a variety of fields. Many of them are constructed based on the Sequential Recommendation (SR) concept. The main idea of SR is to guess the next items that the users have interest in or most likely to buy by mainly modelling the sequential dependencies over their previous interactions with items in a sequence. Currently, there are mainly 2 approaches of capturing the user's actions pattern: Markov Chains (MCs) and Recurrent Neural Networks(RNNs). Mcs assumes that the next action or behavior of the user can be predicted based on their prior or last few actions, while the RNNs can be used for longer-term semantics. Both of them have their ways to perform better, while RNNs is better in working with a dense dataset when it is feasible to improve the complexity of the model, on the other hand MCs performs better while handling the datasets that are extremely sparse where the parsimony is the most important. SASRec, a model that combine these 2 approaches, aims to balance these 2 approaches by capturing long-term semantics but using an attention mechanism, guessing based on few last actions. With the SASRec is our base model, we introduces an advanced SASRec, which is called Advanced SASRec(ASASRec). In this paper, we try to improve the SASRec model by focusing on re-modelling the function, adding the the timestamp input in order to increase the accuracy, speed and efficiency of the model. We have tested it with some big datasets and the result show the effectiveness and flexibility of the model suggesting its applicability in the real-world recommendation scenarios.**

I. Introduction

Recommendation Systems play an important role in various fields, from e-commerce to media platforms, increasing user's satisfaction and experience. One of the most popular methodologies of recommendation is Sequential Recommendation(SR), which is used to predict user's next behaviors based on their historical interactions sequence. Since the space of SR input grows expotentially with the number of user's past actions use das context, it's challenging to make use of the patterns from sequential dynamics. Therefore, there are many researches to find an approach to capturing these sequential dynamics.
Currently, there are 2 popular approaches to capturing such patterns: Markov Chains(MCs) and Recurrent Neural Networks(RNNs). MCs assumes that the next behavior of the user only depends on last or last few action, which works best with sparse datasets and the parsimony of the model is a crucial element. On the other hand, RNNs uses all the user's last actions and summarize them for longer-term semantics and performs better with dense datasets, when it's feasible to improve the complexity of the model. Each of these approaches may performs good in some specific cases, has their own limitations. While MCs somewhat underperform if it's used in more complex scenarios, the RNNs require a large amount of data.
SASRec is a model that has just been introduced recently, aims to combine these 2 methods together. This model applies the self-attention , an efficient mechanism to uncover syntactic and semantic patterns between words in a sentenceto SR. This approach can address both of the problems of the two mentioned methods: while being to handle and capture all the user's behaviors context from the past, it can also make predictions based on few last previous actions. SASRec show a better performance than many MC/CNN/RNN-based methods. SASRec, however, is still struggling while dealing with some specific cases, given that it only uses the user's ID and item's ID as inputs for the model, which leads to some mis predictions. Inspired by SASRec, and with the aim of upgrading the model and increasing its accuracy, we propose an improvement of SASRec, called ASASREC. By using timestamp as an additional timestamp input for the model, and re-modelling the mathematics of the model with an additional change of regularization method, we successfully improve the accuracy of the old version SASRec. The result shows that our work in improving the SASRec into ASASRec will contribute to solving the real world problems in building a efficient and accurate recommender system.

II. RELATED WORKS

A. Stochastic Shared Embeddings

Regularization techniques are used to manage model complexity and prevent over-fitting. ℓ_2 regularization arises as the

most popular approach and has been widely used in various matrix factorization models within recommender systems. On the other hand, ℓ_1 regularization is used when a sparse model is preferred. For deep neural networks, it has been shown that ℓ_p regularizations often lack potency, while dropout is more effective in practical scenarios.Additionally, there are numerous other regularization such as techniques, including parameter sharing, max-norm regularization, gradient clipping, etc.
In this paper, we utilize a regularization method called SSE, which is a tested data-driven technique for deep neural network without the need of dropout. In brief, the technique is data-driven in that the loss influences the regularization step (the method stochastically replace embedding with another embedding with some pre-defined probability during SGD). Additionally, the original paper [40] has experimented it with various architectures, including recommendation systems with encouraging results.

B. Self-Attentive Sequential Recommendation

Sequential dynamics is used to capture the context of users' next behaviors based on their recent actions. There are 2 methods that have arisen: Markov Chains (MCs), which predict a user's next action based solely on their recent history, and Recurrent Neural Networks (RNNs), which consider the entire action history to uncover long-term semantics. While MCs performs best in extremely sparse datasets where model simplicity is important, RNNs works better in denser datasets which requires higher model complexity.
SASRec is introduced to make use of the advantages of these 2 approaches by combining the ability to capture long-term semantics, akin to RNN, with the efficiency of making predictions based on a limited number of actions, similar to MC, through the use of an attention mechanism. At each time step, SASRec identifies 'relevant' items from a user's action history and uses them to predict the next item. Studies show that this method outperforms various state-of-the-art sequential models (including MC/CNN/RNN-based approaches) on both sparse and dense datasets. Moreover, the model is more efficient than comparable CNN/RNN-based models.
However, our model ASASRec outperforms SASRec by almost 0.05 in terms of NDCG@10 (Normalized Discounted Cumulative Gain at rank 10) on real-world datasets. The improved performance of demonstrates its superiority in personalized recommendation accuracy.

III. Methodology

To begin with, normally, the Sequential Recommendation systems will get input as a user's action sequence

$$S^u = (S^u_1, S^u_2, ..., S^u_{|S^u|}) \tag{1}$$

where S^u is the number of actions, the systems will predict for the next item. During the predict process, at step t, the model will predicts the next item based on the previous t items. As shown in Figure 1, conceptually, the input of the model can be though as $S^u = (S^u_1, S^u_2, ..., S^u_{|S^u|-1})$ and its output is $S^u = (S^u_2, S^u_3, ..., S^u_{|S^u|})$ which is a 'right shifted'

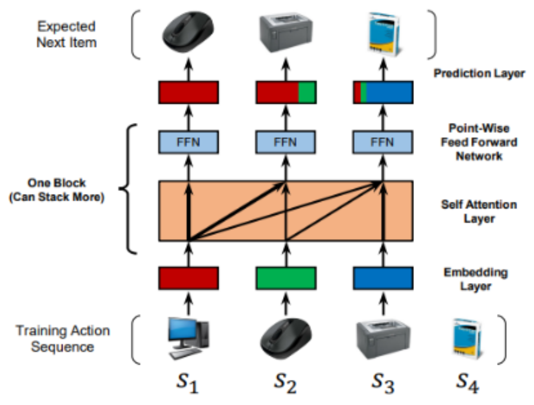
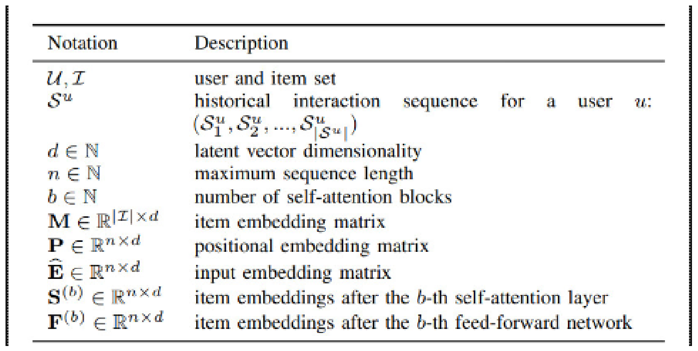


Figure 1: A simplified diagram showing the training process of SASRes. At each time step, the model considers all previous items, and uses attention to 'focus' on items relevant to the next action



Notation	Description
$\mathcal{U}, \mathcal{I}$	user and item set
$\mathcal{S}^u$	historical interaction sequence for a user u: $(\mathcal{S}^u_1, \mathcal{S}^u_2, ..., \mathcal{S}^u_{\lvert\mathcal{S}^u\rvert})$
$d \in \mathbb{N}$	latent vector dimensionality
$n \in \mathbb{N}$	maximum sequence length
$b \in \mathbb{N}$	number of self-attention blocks
$\mathbf{M} \in \mathbb{R}^{\lvert\mathcal{I}\rvert \times d}$	item embedding matrix
$\mathbf{P} \in \mathbb{R}^{n \times d}$	positional embedding matrix
$\widehat{\mathbf{E}} \in \mathbb{R}^{n \times d}$	input embedding matrix
$\mathbf{S}^{(b)} \in \mathbb{R}^{n \times d}$	item embeddings after the b-th self-attention layer
$\mathbf{F}^{(b)} \in \mathbb{R}^{n \times d}$	item embeddings after the b-th feed-forward network

Figure 2

version of the input. In this section, we will describe about the construction of sequential recommendation model via an *embedding layer*, *several self attention* blocks and some improvement with the respective architecture. You can find notation summarized in Figure 2.

A. Embedding layer

We will transform the training sequence $S^u = (S^u_1, S^u_2, ..., S^u_{|S^u|-1})$ into a fixed length sequence $s = (s_1, s_2, s_3 ..., s_n)$ where n is the maximum length that the model can handle. Now, we will talk about cases where sequence length is not n. Firstly its length is bigger than n, we will choose the most recent n actions (e.g $(s_1, s_2, .., s_n)$ of $(s_1, s_2, .., s_{n+1}, ...)$) Secondly, if sequence length is smaller than n, we will add a *padding* item to the left until its length is n. Then, we create an item embedding matrix $M \in R^{|I| \times d}$, where d is the latent dimensionality, and retrieve the input embedding matrix $E \in R^{n \times d}$, where $E_i = M_{S_i}$. A constant zero vector 0 is used as the embedding for the padding term.

- **Positional Embedding:** In fact, the self-attention model doesn't include any recurrent or convolutional module, it is not aware of the positions of the previous items. Hence we inject an earnable position embedding $P \in R^{n \times d}$ into the input embedding

$$\widehat{E} = \begin{bmatrix} M_{S_1} + P_1 \\ M_{S_2} + P_2 \\ \dots \\ M_{S_n} + P_n \end{bmatrix} \tag{2}$$

B. Self-Attention Block

First, let get start with the scaled dot-product attention:

$$Attention(Q, V, K) = softmax(\frac{QK^T}{\sqrt{d}})V \tag{3}$$

Where Q represents the queries, K the keys and V the values (each row represents an item). Intuitively, the attention layer calculates a weighted sum of all values, the weight of between query i and value j relates to the interaction between query i and key j. In addition, the scale $\sqrt{d}$ is to avoid large value of the inner product in the case that the dimensionality is high.

- **Self-Attention Layer:** his is an element of the self-attention block, the self-attention operation takes the embedding matrix $\widehat{E}$ as input, converts it to three matrices through linear projections and then put these matrices to the attention layer:

$$S = SA(\widehat{E}) = Attention(\widehat{E}W^Q, \widehat{E}^K, \widehat{E}^L) \tag{4}$$

  where the projection matrices $\widehat{E}W^Q, \widehat{E}W^K, \widehat{E}W^L \in R^{d \times d}$. In fact, the projections make the model more flexible. That is, for example, the model can also learn asymmetric interactions (i.e (query i, key j) and (query j,key) can have different interactions).
- **Point-Wise Feed-Forward Network:** Self-attention is able to aggregate all previous items' embeddings with adaptive weights. However, it is still a linear model. In order to endow the model with nonlinearity and consider interactions between different latent dimensions, we apply point-wise two-layer feed forward network to all S_i identical:

$$F_i = FFN(S_i) = ReLU(S_iW^{(1)} + b^{(1)})W^{(2)} + b^{(2)} \tag{5}$$

  where $W^{(1)}, W^{(2)}$ are $d \times d$ matrix and $b^{(1)}, b^{(2)}$ are dimensional vectors. Note that, there is no interaction between two S_i and $S_j (j \neq i)$, this mean the information is not leaked from back to front.

C. Stacking Self-Attention Blocks:

With the first self-attention block, F_i is aggregates all previous items' embeddings (i.e $\widehat{E_j with j} \leq i$). However, it may be useful to find out more item transitions via another self-attention block based on F his mean, we stack self-attention block together, the output of one block will be input of an other (i.e a self-attention layer and a feed-forward network). The b-th ($b > 1$) block is defined as:

$$S^{(b)} = SA(F^{(b-1)})F_i^{(b)} = FFN(S_i^{(b)}, \forall i \in 1, 2, 3.., n \tag{6}$$

the 1-st block is defined as $S^{(i)} = S$ and $F^{(i)} = F$ However, there are some issue occur when the network goes deeper:

- The increased model capacity leads to overfitting.
- The traing process becomes unstable (due to vanishing gradients etc.).
- Model with more parameters often require more training time.

We willl talk about some idea to handle these issues.

- **Residual Connections:** The core idea behind residual networks is to propagate low-layer features to higher layers by residual connection. Hence, if low-layer features are useful, the model can easily propagate them to the final layer. For eample, in out cases, the last visited item plays a key role on predicting the next item. However, after several self-attention blocks, the embedding of the last visited item is entangled with all previous items, adding residual connections to propagate the last visited item's embedding to the final layer would make it much easier for the model to leverage low-layer information.
- **Layer Normalization**: Layer normalization is used to normalize the inputs across features (i.e., zero-mean and unit-variance) which is beneficial for stabilizing and accelerating neural network training. Assuming the input is a vector x which contains all features of a sample, the operation is defined as:

$$LayerNorm(x) = \alpha \odot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{7}$$

  where $\odot$ is an element-wise product (i.e the Hadamard product), μ and σ are the mean and the variance of x, α and β are learned scaling factors and bias term.
- **Dropout:** 'Dropout' is regularization techniques to alleviate overfitting problems in deep neural networks. The idea of dropout is simple: randomly 'turn off' neurons with probability p during training, and use all neurons when testing. From these idea, the following operation can reduce these above problem:

$$g(x) = x + Dropout(g(LayerNorm(x))) \tag{8}$$

where $g(x)$ represents the self attention layer or the feed-forward network. That is to say, for layer g in each block, we apply layer normalization on the input x before feeding into g, apply dropout on $g's$ output, and add the input x to the final output. We introduce these operations below.

D. Prediction Layer:

After b self-attention blocks that adaptively and hierarchically extract information of previously consumed items, we predict the next item (given the first t items) based on F_t^b.Particularly, we adopt an Matrix Factorization layer to predict the relevance of item i:

$$r_{i,t} = F_t^{(b)} N_i^{(T)} \tag{9}$$

where $r_{i,t}$ is the relevance of item i being the next item given the first t items (i.e $s_1, s_2, ..., s_t$) and $N \in R^{|I| \times d}$

is an item embedding matrix. Hence, a high interaction score $r_{i,t}$ means a high relevance, and we can generate recommendations by ranking the scores.

- **Shared Item Embedding:**To reduce the model size and alleviate overfitting, we consider another scheme which only uses a single item embedding M:

$$r_{i,t} = F_t^{(b)} M_i^{(T)} \tag{10}$$

Note that $F_t^{(b)}$ can be represented as a function depending on the item embedding M: $F_t^{(b)} = f(M_{s_1}, M_{s_2}, ..., M_{s_t})$

- **Explicit User Modeling:** In order to provide personalized recommendations, existing methods often take one of two approaches: 1) learn an explicit user embedding representing user preference. 2) consider the user's previous actions, and induce an implicit user embedding from embeddings of visited items. Our method falls into the latter category, as we generate an embedding $F_n^{(b)}$ by considering all actions of a user. However, we can also insert an explicit user embedding at the last layer, for example via addition:

$$r_{u,i,t} = (U_u + F_t^{(b)}) M_i^{(T)} \tag{11}$$

Where U is user embedding matrix.

E. Network Training:

Recall that, we convert user sequence (excluding the last action) $(S_1^u, .., S_{|S^u|-1}^u)$ to $s = (s_1, s_2, .., s_n)$ via truncation or padding items. We define t as the expected output at time step t:

$$o_t = \begin{cases} < pad > & \text{if } s_t \text{ is a padding item} \\ s_{t+1} & 1 \le t < n \\ S_{S^u|}^u \end{cases} \tag{12}$$

where $< pad >$ indicates a padding item. Our model takes a sequence t as input, the corresponding sequence o as expected output, and we adopt the binary cross entropy loss as the objective function:

$$\sum_{S^u \in S} \sum_{t \in [1,2,3..,n]} = [\log(\sigma(r_{o_t,t})) + \sum_{j \notin S^u} (1 - \sigma(r_{o_t,t}))] \tag{13}$$

Note that we ignore the terms where $o_t =< pad >$

IV. Experiments

In this section, we present our experimental setup and empirical results using various means and metrics.

A. Datasets

We assess our methodologies using two real-world datasets from different applications. These datasets differ significantly in their domains, platforms, and levels of sparsity:

- **Amazon:** This dataset, presented in the VisualSIGIR paper, comprises product reviews collected by crawling Amazon.com. Each main product category on Amazon is treated as a distinct dataset. We focus on two categories, namely 'Beauty' and 'Games.' This dataset is particularly notable for its high sparsity and variability. [47]
- **Steam:** This dataset contains information from October 2010 to January 2018 and includes 2,567,538 users, 15,474 games, and 7,793,069 English reviews. It also provides additional information that could be valuable for future researches, such as users' play hours, pricing details, media scores, categories, developers, and more.

For all datasets, we consider the presence of a review or rating as implicit feedback, indicating that the user has interacted with the item. We utilize timestamps to determine the chronological order of these actions. Users and items with fewer than 5 related actions are excluded from the analysis. To partition the data, we divide the historical sequence $\mathcal{S}^u$ for each user u into three parts: (1) the most recent action $\mathcal{S}_{|\mathcal{S}^u|}^u$ for testing, (2) the second most recent action $\mathcal{S}_{|\mathcal{S}^u|-1}^u$ for validation, and (3) all remaining actions for training. Note that during testing, the input sequences contain training actions and the validation action.

Data statistics are shown in Table I. From the table,it is clear that the *Amazon* datasets have the fewest actions per user and per item (on average) and *Steam* has a high average number of actions per item.

Table I: Dataset statistics (after preprocessing)

Dataset	#users	#items	avg. actions /user	avg. actions /item	#actions
Amazon Beauty	52,024	57,289	7.6	6.9	0.4M
Amazon Games	31,013	23,715	9.3	12.1	0.3M
Steam	334,730	13,047	11.0	282.5	3.7M

B. Evaluation Metrics

After reviewing a variety of metrics from past papers, we have decided to choose the two most common evaluation standard ranking system, namely NDCG and Hit for top recommendations:

- NDCG@K: defined as:

$$\text{NDCG@}K = \frac{1}{n} \sum_{i=1}^{n} \frac{\text{DCG@}K(i, \Pi_i)}{\text{DCG@}K(i, \Pi_i^*)}, \tag{14}$$

where i represents i-th user and

$$\text{DCG@}K(i, \Pi_i) = \sum_{l=1}^{K} \frac{2^{R_{i\Pi_{il}}} - 1}{\log_2(l+1)}. \tag{15}$$

In the DCG definition, Π_{il} represents the index of the l-th ranked item for user i in test data based on the learned score matrix X. R is the rating matrix and R_{ij} is the rating given to item j by user i. Π_i^* is the ordering provided by the ground truth rating.

- Hit@K: defined as a fraction of positive items retrieved by the top K recommendations the model makes:

$$\text{Hit@}K = \frac{\sum_{i=1}^{n} 1\{\exists 1 \le l \le K : R_{i\Pi_{il}} = 1\}}{n}, \tag{16}$$

here we already assume there is only a single positive item that user will engage next and the indicator function

Table II: Recommendation performance. The best-performing method in each row is boldfaced, and the second-best method in each row is underlined. Improvements over non-neural and neural approaches are shown in the last two columns respectively.

Dataset	Metric	(a) PopRec	(b) FPMC	(c) TransRec	(d) GRU4Rec	(e) GRU4Rec$^+$	(f) ASASREC
Beauty	Hit@10	0.4053	0.4360	<u>0.4657</u>	0.2175	0.3999	**0.4904**
	NDCG@10	0.2327	0.2941	<u>0.3070</u>	0.1253	0.2606	**0.3269**
Games	Hit@10	0.4774	0.6852	<u>0.6888</u>	0.2988	0.6749	**0.7460**
	NDCG@10	0.2829	0.4730	0.4607	0.1887	<u>0.4809</u>	**0.5410**
Steam	Hit@10	0.7222	0.7760	0.7674	0.4240	<u>0.8168</u>	**0.8879**
	NDCG@10	0.4585	0.5061	0.4902	0.2746	<u>0.5745</u>	**0.6456**

$1\{\exists 1 \leq l \leq k : R_{i\Pi_{il}} = 1\}$ is defined to indicate whether the positive item falls into the top K position in our obtained ranked list .

In the ranking setting, at time point t, the rating matrix R can be constructed in two ways. One approach is to include all ratings after t, while the other approach is to only include ratings at time point $t+1$. For the purpose of maintaining a similar setting that allows for convenient performance measurement, we adopt the latter approach.

When dealing with a large dataset that consists of numerous users and items, the evaluation process can become computationally intensive. Specifically, computing the ranking of all items based on their predicted scores for every individual user, as described in (14), would be quite time-consuming. To speed up the evaluation process, we employ a sampling strategy. Specifically, we sample a fixed number C of negative candidates while also including the positive item that we know the user will engage with next. By doing so, both the set of item candidates R_{ij} and the set of item predictions Π_i are reduced to a smaller subset. Consequently, prediction scores only need to be computed for this reduced set of items through a single forward pass of the neural network.

Our ideal scenario is to achieve NDCG and Hit values of exactly 1. A NDCG@$K = 1$ indicates that the positive item is consistently ranked at the top position of the top-K recommendation list. Similarly, a Hit@$K = 1$ signifies that the positive item is always included in the top-K recommendations generated by the model.

During our evaluation procedures, increasing the value of C or decreasing the value of K makes the recommendation problem harder. This is because a larger candidate pool and a higher ranking quality are desired, therefore requiring the model to make more accurate and precise recommendations.

To show the effectiveness of our method, we include three groups of recommendation methods.

The first group includes general recommendation methods which only consider user feedback without considering the sequence order of actions:

- **PopRec**.

The second group contains methods based on first order Markov chains, which consider the last visited item:

- **Factorized Personalized Markov Chains (FPMC) [1]**
- **Translation-based Recommendation (TransRec) [19]**

The last group contains deep-learning based sequential recommender systems, which consider several (or all) previously visited items:

- **GRU4Rec [26]**
- **GRU4Rec$^+$ [26]**

C. *Implementation Details*

For the architecture in the default version of ASASREC,we use the same setting as its predecessor SASREC which has two self-attention blocks ($b = 2$), and use the learned positional embedding. Item embeddings in the embedding layer and prediction layer are shared. We implement ASASREC with *TensorFlow*. The optimizer is the *Adam* optimizer [42], the learning rate is set to 0.001, and the batch size is 128. The dropout rate of turning off neurons is 0.5 for the three datasets due to their sparsity. The maximum sequence length n is set to 50 for the datasets, i.e., roughly proportional to the mean number of actions per user. Performance of variants and different hyper-parameters is examined below.

D. *Recommendation Performance*

Table II presents the recommendation performance of all methods on the three datasets. By considering the second best methods across all datasets, a general pattern emerges with non-neural methods performing better on sparse datasets and neural approaches performing better on denser datasets. Presumably this owes to neural approaches having more parameters to capture high order transitions (i.e., they are expressive but easily overfit), whereas carefully designed but simpler models are more effective in high-sparsity settings. Our method ASASREC outperforms all baselines on both sparse and dense datasets, and gains 6.9% Hit Rate and 9.6% NDCG improvements (on average) against the strongest baseline. One likely reason is that our model can adaptively attend items within different ranges on different datasets (e.g. only the previous item on sparse datasets and more on dense datasets).

Table III: Comparing SASREC and ASASREC on Amazon Games Dataset while varying dimension of embeddings.

METHODS	NDCG@10	HIT@10	USER DIM	ITEM DIM
SASREC	0.5936	0.8233	N/A	50
SASREC	0.5919	0.8202	N/A	100
ASASREC	**0.6221**	**0.8129**	50	50
ASASREC	**0.6292**	**0.8389**	50	100

Table IV: Comparing SASREC and ASASREC on Amazon Games Dataset while varying the maximum length allowed.

METHODS	NDCG@10	RECALL@10	MAX LEN	USER DIM	ITEM DIM
SASREC	0.5919	0.8202	200	N/A	100
ASASREC	**0.6281**	**0.8341**	200	50	100
SASREC	0.5769	0.8045	100	N/A	100
ASASREC	**0.6186**	**0.8318**	100	50	100

E. Scalability

As with standard MF methods, ASASREC scales linearly with the total number of users, items and actions, which is roughly the same as the original SASREC. A potential scalability concern is the maximum length n, however the computation can be effectively parallelized with GPUs. Here we measure the training time and performance of ASASREC with different n, empirically study its scalability, and analyze whether it can handle sequential recommendation in most cases. Table V shows the performance and efficiency of ASASREC with various sequence lengths. Performance is better with larger n, up to around $n = 500$ at which point performance saturates (possibly because 99.8% of actions have been covered). However, even with $n = 600$, the model can be trained in 2,000 seconds, which is still faster than Caser and GRU4Rec$^+$. Hence, our model can easily scale to user sequences whose lengths is a few hundred, and we think it's sufficient for most cases, especially for feedback type like reviews and purchases. We plan to investigate approaches for handling very long sequences in the future.

Table V: Scalability: performance and training time with different maximum length n on *Steam dataset*.

n	10	50	100	200	300	400	500	600
Time(s)	75	101	157	341	613	965	1406	1895
NDCG@10	0.480	0.557	0.571	0.587	0.593	0.594	0.596	0.595

V. CONCLUSION

In this work, we proposed an extension for the self-attention based sequential model *SASREC* for next item recommendation. ASASREC models the entire user sequence (without any recurrent or convolutional operations), and adaptively considers consumed items for prediction. Our model provides additional complexities regarding to the aspect of user/item interactions history in comparison with the original SASREC model. In the future, we plan to extend the model by incorporating context information (e.g. dwell time, action types, rating, devices, etc.), and possibly CTR (e.g. clicks through rate).

REFERENCES

[1] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized markov chains for next-basket recommendation," in *WWW*, 2010.

[2] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," in *ICLR*, 2016.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017.

[4] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *ICDM*, 2008.

[5] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "BPR: bayesian personalized ranking from implicit feedback," in *UAI*, 2009.

[6] F. Ricci, L. Rokach, B. Shapira, and P. Kantor, *Recommender systems handbook*. Springer US, 2011.

[7] Y. Koren and R. Bell, "Advances in collaborative filtering," in *Recommender Systems Handbook*. Springer, 2011.

[8] S. Kabbur, X. Ning, and G. Karypis, "Fism: factored item similarity models for top-n recommender systems," in *SIGKDD*, 2013.

[9] S. Zhang, L. Yao, and A. Sun, "Deep learning based recommender system: A survey and new perspectives," *arXiv*, vol. abs/1707.07435, 2017.

[10] S. Wang, Y. Wang, J. Tang, K. Shu, S. Ranganath, and H. Liu, "What your images reveal: Exploiting visual contents for point-of-interest recommendation," in *WWW*, 2017.

[11] W. Kang, C. Fang, Z. Wang, and J. McAuley, "Visually-aware fashion recommendation and design with generative image models," in *ICDM*, 2017.

[12] H. Wang, N. Wang, and D. Yeung, "Collaborative deep learning for recommender systems," in *SIGKDD*, 2015.

[13] D. H. Kim, C. Park, J. Oh, S. Lee, and H. Yu, "Convolutional matrix factorization for document context-aware recommendation," in *RecSys*, 2016.

[14] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua, "Neural collaborative filtering," in *WWW*, 2017.

[15] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, "Autorec: Autoencoders meet collaborative filtering," in *WWW*, 2015.

[16] Y. Koren, "Collaborative filtering with temporal dynamics," *Communications of the ACM*, 2010.

[17] C. Wu, A. Ahmed, A. Beutel, A. J. Smola, and H. Jing, "Recurrent recommender networks," in *WSDM*, 2017.

[18] L. Xiong, X. Chen, T.-K. Huang, J. Schneider, and J. G. Carbonell, "Temporal collaborative filtering with bayesian probabilistic tensor factorization," in *SDM*, 2010.

[19] R. He, W. Kang, and J. McAuley, "Translation-based recommendation," in *RecSys*, 2017.

[20] R. He, C. Fang, Z. Wang, and J. McAuley, "Vista: A visually, socially, and temporally-aware model for artistic recommendation," in *RecSys*, 2016.

[21] R. He and J. McAuley, "Fusing similarity models with markov chains for sparse sequential recommendation," in *ICDM*, 2016.

[22] J. Tang and K. Wang, "Personalized top-n sequential recommendation via convolutional sequence embedding," in *WSDM*, 2018.

[23] H. Jing and A. J. Smola, "Neural survival recommender," in *WSDM*, 2017.

[24] Q. Liu, S. Wu, D. Wang, Z. Li, and L. Wang, "Context-aware sequential recommendation," in *ICDM*, 2016.

[25] A. Beutel, P. Covington, S. Jain, C. Xu, J. Li, V. Gatto, and E. H. Chi, "Latent cross: Making use of context in recurrent recommender systems," in *WSDM*, 2018.

[26] B. Hidasi and A. Karatzoglou, "Recurrent neural networks with top-k gains for session-based recommendations," *CoRR*, vol. abs/1706.03847, 2017.

[27] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *ICML*, 2015.

[28] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *ICLR*, 2015.

[29] J. Chen, H. Zhang, X. He, L. Nie, W. Liu, and T. Chua, "Attentive collaborative filtering: Multimedia recommendation with item- and component-level attention," in *SIGIR*, 2017.

[30] J. Xiao, H. Ye, X. He, H. Zhang, F. Wu, and T. Chua, "Attentional factorization machines: Learning the weight of feature interactions via attention networks," in *IJCAI*, 2017.

[31] S. Wang, L. Hu, L. Cao, X. Huang, D. Lian, and W. Liu, "Attention-based transactional context embedding for next-item recommendation," in *AAAI*, 2018.

[32] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[33] J. Zhou, Y. Cao, X. Wang, P. Li, and W. Xu, "Deep recurrent models with fast-forward connections for neural machine translation," *TACL*, 2016.

[34] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *ECCV*, 2014.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[36] L. J. Ba, R. Kiros, and G. E. Hinton, "Layer normalization," *CoRR*, vol. abs/1607.06450, 2016.

[37] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015.

[38] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *JMLR*, 2014.

[39] D. Warde-Farley, I. J. Goodfellow, A. C. Courville, and Y. Bengio, "An empirical analysis of dropout in piecewise linear networks," *CoRR*, vol. abs/1312.6197, 2013.

[40] Liwei Wu and Shuqing Li and Cho-Jui Hsieh and James Sharpnack, "Stochastic Shared Embeddings: Data-driven Regularization of Embedding Layers ", 2020

[41] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, 2009.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[43] D. Povey, H. Hadian, P. Ghahremani, K. Li, and S. Khudanpur, "A time-restricted self-attention layer for asr," in *ICASSP*, 2018.

[44] P. Wang, J. Guo, Y. Lan, J. Xu, S. Wan, and X. Cheng, "Learning hierarchical representation model for next basket recommendation," in *SIGIR*, 2015.

[45] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *CoRR*, vol. abs/1803.01271, 2018.

[46] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[47] J. J. McAuley, C. Targett, Q. Shi, and A. van den Hengel, "Image-based recommendations on styles and substitutes," in *SIGIR*, 2015.

[48] S. Feng, X. Li, Y. Zeng, G. Cong, Y. M. Chee, and Q. Yuan, "Personalized ranking metric embedding for next new poi recommendation," in *IJCAI*, 2015.

[49] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *SIGKDD*, 2008.