ITERATIVE MEMORY NETWORK FOR LONG SEQUENTIAL USER BEHAVIOR MODELING IN RECOMMENDER SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Sequential user behavior modeling is a key feature in modern recommender systems, seeking to capture users' interest based on their past activities. There are two usual approaches to sequential modeling : Recurrent Neural Networks (RNNs) and the attention mechanism. As the user behavior sequence gets longer, the usual approaches encounter problems. RNN-based methods incur the problem of fast forgetting, making it difficult to model the user's interests long time ago. The self-attention mechanism and its variations such as the transformer structure have the unfortunate property of a quadratic cost with respect to the input length, which makes it difficult to deal with long inputs. The target attention mechanism, despite having only $O(L)$ memory and time complexity, cannot model intra-sequence dependencies. In this paper, we propose Iterative Memory Network (IMN), an end-to-end differentiable framework for long sequential user behavior modeling. In our model, the target item acts as a memory trigger, continuously eliciting relevant information from the long sequence to represent the user's memory on the particular target item. In the Iterative Memory Update module, the model walks over the long sequence multiple iterations and keeps a memory vector to memorize the content walked over. Within each iteration, the sequence interacts with both the target item and the current memory for both target-sequence relation modeling and intra-sequence relation modeling. The memory is updated after each iteration. The framework incurs only $O(L)$ memory and time complexity while reduces the maximum length of network signal travelling paths to $O(1)$, which is achieved by the self-attention mechanism with $O(L^2)$ complexity. Various designs of efficient self-attention mechanisms are at best $O(LlogL)$. Extensive empirical studies show that our method outperforms various state-of-the-art sequential modeling methods on both public and industrial datasets for long sequential user behavior modeling.

1 INTRODUCTION

Click-through rate (CTR) prediction is critical for recommender systems. User sequential modeling is the key to mine users' interest for accurate predictions. As the user sequence gets longer, particularly with the sequence length larger than 1000, the prediction task requires extraordinary long-range dependency modeling. Furthermore, it requires efficient memory storage, acceptable training speed and real-time inference.

Hidasi et al. (2016) employ Recurrent Neural Networks (RNNs) for sequential recommenders, summarizing previous actions with a hidden state for the next action prediction. The long short-term memory (LSTM) is a special class in RNNs that models sequential behaviors (Hochreiter & Schmidhuber, 1997). Graves et al. (2014) prove that LSTM forgets quickly and fails to generalize to sequences longer than 20. Many empirical results also verify that RNN-based sequential recommenders do not perform as well as attention-based methods since the hidden state forgets long-term information quickly (Zhou et al., 2017; Kang & McAuley, 2018; Zhou et al., 2018; Pi et al., 2019).

Lately, the self-attention mechanism has proven to benefit a wide range of application domains, such as machine translation (Vaswani et al., 2017), speech recognition (Chan et al., 2015), computer vision (Xu et al., 2015; Parmar et al., 2019) and the recommender systems (Kang & McAuley, 2018).

The self-attention mechanism attends to different positions in the sequence, captures the most important features and allows the model to handle dependencies between elements across long distances. SASRec employs the self-attention mechanism for sequential recommenders and demonstrates it is superior to both convolution-based and recurrence-based approaches (Kang & McAuley, 2018).
Two problems arise applying the self-attention mechanism to long sequential recommender systems. Firstly, the memory complexity and the computational complexity are both quadratic with respect to the sequence length. Since the self-attention mechanism allows for parallelization, the quadratic computational complexity might not be the major bottleneck given enough computational resources. Yet, the $O(L^2)$ memory complexity makes it infeasible to handle long sequences. Secondly, the self-attention mechanism is usually performed in a fixed fully-connected structure, which could be non-optimal for downstream tasks.
To alleviate the quadratic complexity problem, efficient self-attention mechanisms are designed. Examples include Sparse Adaptive Connection (Li et al., 2020), ProbSparse self-attention (Zhou et al., 2020), the Sparse Transformer (Child et al., 2019), the Reformer (Kitaev et al., 2020) and so on. All these methods reduce the complexity to at best $O(LlogL)$. Furthermore, as these methods are experimented in Natural Language Processing or Computer Vision, they do not utilize the domain knowledge for sequential recommenders. For sequential recommenders, the relation between the sequence item and the target item is crucial.
Deep Interest Network (DIN) is an architecture designed for sequential recommender systems (Zhou et al., 2018). It proposes the target attention mechanism (called 'local activation unit' in the paper), adaptively learning the user interest representation from historical behaviors with respect to a particular target item. Though DIN only incurs $O(L)$ complexity, it does not model dependencies between elements in the sequence. Research has shown that the length of paths signals need to traverse is the key influence on the ability to learn long-range dependencies (Vaswani et al., 2017). The shorter the paths between any combination of positions, the easier it is to learn long-range dependencies. The self-attention mechanism can reduce the maximum length of network signals traveling paths into the theoretical shortest of $O(1)$ since it considers pairwise dependencies between elements in the sequence. On the contrary, the target attention mechanism allows no signal traveling between elements in the sequence and hence models no intra-sequence dependencies.
In this paper, we propose an Iterative Memory Network (IMN) for long sequential user behavior modeling. In our IMN, the target item is a memory trigger, continuously eliciting relevant information from the sequence. The IMN walks over the sequence for multiple iterations, keeping a memory vector to represent the content walked over. During each iteration, a multi-way attention is calculated between the target item, the memory and the sequence items. Afterwards, the memory vector is updated based on the information retrieved from this particular iteration. The contributions of this paper are summarized as follows:

- We propose the Iterative Memory Network (IMN), an end-to-end differentiable framework for sequential recommenders. To the best of our knowledge, it outperforms the state-of-the-art models for long sequential user behavior modeling. The framework is scalable to sequential recommenders with other objective functions.
- We propose the method to walk the sequence for multiple iterations, with a memory vector to memorize the content walked over. Our IMN models both long-range intra-sequence dependencies as in the self-attention mechanism and target-sequence dependencies as in the target-attention mechanism. While the target-attention mechanism only models the co-occurence between the target item and sequence items, the IMN models co-occurence beyond the $(target\ item, sequence\ item)$ pair to triplets, quadruplets and so on. We validate with extensive empirical results that this method outperforms the usual single pass target attention significantly.
- The IMN framework is efficient with both memory and time complexity at $O(L)$. We replace the fully-connected structure in the self-attention mechanism with our domain knowledge for sequential recommenders, and reduce the $O(L^2)$ complexity to $O(L)$. Various designs for efficient self-attention mechanisms unanimously explore positions that dominate attention weights calculation, and achieve $O(LlogL)$ memory and time complexity at best.
- We design the iterative memory update module with the multi-way attention for intra-sequence dependency modeling. As the memory vector encapsulates the sequence content walked over, the multi-way attention between the sequence, the target item and the memory reduces the maximum length of signals traveling path within the sequence to $O(1)$. In con-

trast to the target attention mechanism which allows for no intra-sequence network signals travelling, IMN achieves this with the same linear memory and time complexity.

2 RELATED WORK

Sequential Recommender Systems. A critical task of sequential recommender systems is to predict the user's next clicking behavior based on his past activities. Recurrent Neural Networks (RNNs) are introduced for sequential recommenders (Wu et al., 2016; Hidasi et al., 2016). Due to RNNs' sequential nature, RNN-based methods are difficult to parallelize. RNNs also suffer from the problem of fast forgetting (Graves et al., 2014). Attention is first introduced in the encoder-decoder framework, for better positional alignments in the machine translation task (Bahdanau et al., 2016). Researchers prove empirically that the self-attention mechanism with timestamp encodings can replace RNN structures to encode sequences, with significantly less training time (Vaswani et al., 2017). Attention-based sequential modeling methods proliferate in many other tasks, such as computer vision (Xu et al., 2015), reading comprehension (Cui et al., 2016; Lin et al., 2017) and speech recognition (Chan et al., 2015). Attention-based recommender systems include self-attention based methods (Kang & McAuley, 2018; Zhou et al., 2017), target-attention based methods (Zhou et al., 2018) and the integration between RNN-based methods and attention-based methods (Zhou et al., 2019). Target-attention based methods calculate the attention weights between the sequence items and the target item, with no intra-sequence comparisons.

Memory Networks. Memory Networks have wide applications in Question Answering (QA) on Natural Language Processing (NLP) tasks, finding facts related to a particular query from a knowledge database (Chaudhari et al., 2021). It can be viewed as a generalization to the attention mechanism with an external memory component. Neural Turing Machines (NTM) introduces the addressing-read-write mechanism for the memory searching and update process (Graves et al., 2014). Weston et al. (2014) proposes the general architecture for Memory Networks. Dynamic Memory Networks (DMN) updates the episodic memory based on a new input sentence with a modified GRU (Kumar et al., 2016). DMTN and DMN+ are improvements to DMN on parameterizations and architectural components (Xiong et al., 2016; Ramachandran & Sohmshetty, 2017). In recommender systems, MIMN uses fixed memory slots to represent user interests and uses a GRU as the controller to update the memory slots with each new clicked item (Pi et al., 2019).
Our design is different from the typical Memory Networks architecture. Though we keep an external memory vector, we do not update the memory vector with each new input as in the typical Memory Networks architecture. The number of sequential operations is $O(1)$ in our design but $O(L)$ in the typical Memory Networks architecture.

3 PROBLEM FORMULATION

The recommender system models the user-item interaction as a matrix $C = \{c_{mn}\}_{M \times N}$, where M and N are the total number of users and items respectively. The interaction is either explicit ratings (Koren, 2009) or implicit feedback (Agarwal et al., 2009). The CTR prediction task is usually based on implicit feedback. We denote $u \in U$ as user and $i \in I$ as item, and the user u_m clicking on the item i_n makes c_{mn} 1 in the matrix C and others 0. Each interaction is associated with a timestamp t that records the time of interaction. Thus, the data in the recommender system is denoted by $D = \{u, i, t, c\}$, each of which refers to the user u interacting with the item i at the time t.
The sequential user behavior modeling refers to the task that predicts the probability of a user $u \in U$ clicking on the target item $i_{target} \in I$ based on the past behaviors of the user, $i_1, i_2, ..., i_L$, where L is the length of the user click sequence. The click sequence is usually in chronological order.
Our paper focuses on long sequential user behavior modeling, where the length of the user behaviors L is on the scale of thousands. We use the term 'target-sequence dependencies' to refer to dependencies between the target item and the sequence items. We use the phrase 'intra-sequence dependencies' to refer to dependencies between the items within the sequence. We use the term 'sequence items' to refer to items in the user behavior sequence.

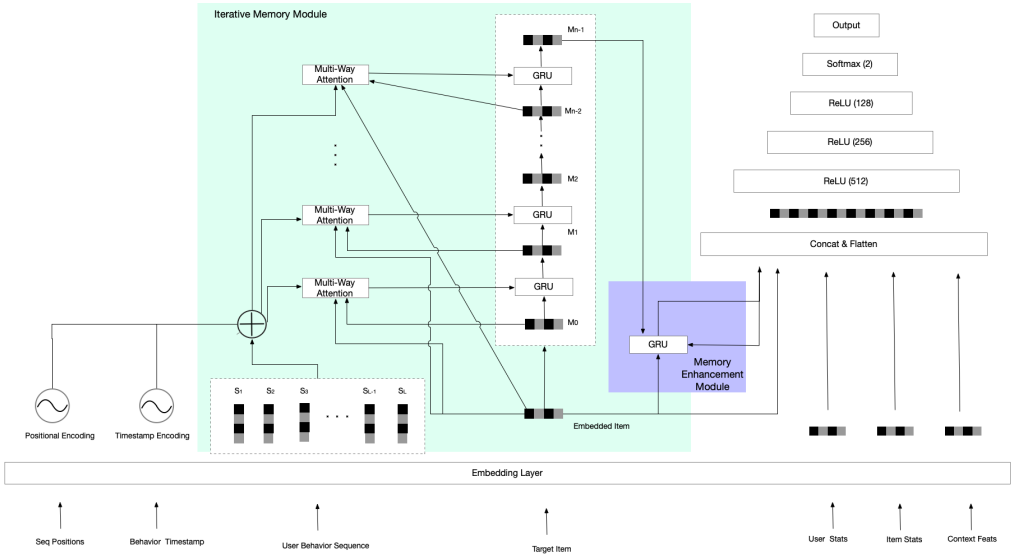

Figure 1: Overview of the Iterative Memory Network architecture. The Iterative Memory module (in green) illustrates the repeated memory update process.

4 Iterative Memory Network

We illustrate the overall Iterative Memory Network architecture in Fig.(1). The encoder layer converts user behavior features, the temporal features, and the target item features to embedding vectors. The Iterative Memory Update module is the major component in our architecture. It models both target-sequence dependencies and intra-sequence dependencies. The Memory Enhancement module repeatedly elicits clearer memory with the target item and outputs the final user interest vector. Next, we discuss the framework in detail.

4.1 Encoder Layer

We use $\boldsymbol{e}_i$ to denote the embedding for the sequence item i. Each user behavior consists of not only the clicked item $\boldsymbol{e}_i$, but also the action time. Different users have different action patterns. Some users click on a daily basis regularly and others click between intervals of months. The action time is a very important continuous feature for temporal information. Since it is difficult to learn a good embedding directly with continuous time features, we bucketize the time feature into multiple granularities and perform categorical feature look-ups. We slice the elapsed time with respect to the ranking time into intervals whose gap length increases exponentially. In other words, we map the time in range[0, 1), [1,2), [2,4), ..., [2^k, 2^{k+1}) to categorical features 0,1,2,...$k+1$. While the bucketized timestamp encodes the absolute temporal information, positional encodings are added to represent the relative positions of sequence items. The timestamp encoding and the positional encoding have the same dimension as the dimension of the item embedding $\boldsymbol{e}_i$ to be directly summed. The sum-up operation is similar to the positional encoding in Transformer structure (Vaswani et al., 2017). We encode the j_{th} user behavior $\boldsymbol{e}_b^j$ as

$$\boldsymbol{e}_b^j = \boldsymbol{e}_i \oplus \boldsymbol{e}_t \oplus \boldsymbol{e}_p \tag{1}$$

where $\oplus$ denotes element-wise sum-up. We denote the embedding for the target item as $\boldsymbol{v}_T$. It shares the item id embedding look-up table with the item id embedding $\boldsymbol{e}_i$ for the sequence.

4.2 Multi-way Attention

We calculate the similarities between the sequence item, the target item and the memory. Then we concatenate them into a single feature vector. Similarities are in both Euclidean distance and the Hadamard distance (denoted by $\circ$) to capture multi-faceted similarity measures.

$$\alpha(\boldsymbol{e}, \boldsymbol{v}, \boldsymbol{m}) = [\boldsymbol{e} - \boldsymbol{m}, \boldsymbol{e} - \boldsymbol{v}, \boldsymbol{e} \circ \boldsymbol{m}, \boldsymbol{e} \circ \boldsymbol{v}] \tag{2}$$

where $\boldsymbol{e}$ is the embedding vector for sequence items, $\boldsymbol{m}$ is the memory embedding vector and $\boldsymbol{v}$ is the embedding vector for the target item.
We input the feature vector into a two-layer point-wise feed-forward network, with *sigmoid* as the activation function. Here point-wise feed-forward means the fully-connected feed-forward network is applied to each sequence item separately and identically.

$$a(\boldsymbol{e}, \boldsymbol{v}, \boldsymbol{m}) = \sigma(W^{(2)}\sigma(W^{(1)}\alpha(\boldsymbol{e}, \boldsymbol{v}, \boldsymbol{m}) + b^{(1)}) + b^{(2)}) \tag{3}$$

We have also experimented with the *softmax* activation function for the second layer. There is very slight change in model performance. The multi-way attention mechanism encodes similarities between the user behavior sequence and the target item to extract the user's interest specific to a particular target item. It also encodes similarities between sequence items and the memory. As the memory vector encapsulates the sequence information, the multi-way attention mechanism can model intra-sequence dependencies between items in the sequence and reduce the maximum length of signal traversal paths to $O(1)$.

4.3 Iterative Memory Update Module

In our Iterative Memory Update module, the target item acts as the memory trigger, continuously waking up relevant memory in the sequence. The memory vector is updated after each iteration with the relevant information retrieved from this iteration. We use a GRU to model the memory update process. The initial memory $\boldsymbol{m}_0$ is initialized from the target item vector, to represent that the user's initial memory on the target item is the target item itself.

$$\boldsymbol{m}_0 = \boldsymbol{v}_T \tag{4}$$

For iteration t, we calculate the user interest representation $\boldsymbol{v}_u^t$ as a weighted sum pooling of the sequence item vectors, with weights derived from Eq.(3).

$$\boldsymbol{v}_u^t = f(\boldsymbol{v}_T, \boldsymbol{e}_b^1, \boldsymbol{e}_b^2, ..., \boldsymbol{e}_b^L) = \sum_{j=1}^{L} a(\boldsymbol{e}_b^j, \boldsymbol{v}_T, \boldsymbol{m}^{t-1})\boldsymbol{e}_b^j = \sum_{j=1}^{L} w_j \boldsymbol{e}_b^j \tag{5}$$

where $\boldsymbol{e}_b^1, \boldsymbol{e}_b^2, ..., \boldsymbol{e}_b^L$ is the list of user behavior embedding vectors, $\boldsymbol{v}_T$ is the embedding vector of target item T and $\boldsymbol{m}$ is the memory embedding vector.
We use the user interest representation $\boldsymbol{v}_u^t$ after iteration t as the input to update the memory $\boldsymbol{m}^{t-1}$.

$$\boldsymbol{m}^t = GRU(\boldsymbol{v}_u^t, \boldsymbol{m}^{t-1}) \tag{6}$$

Since the memory is updated with a weighted sum pooling of sequence item embeddings, the memory contains information on the sequence items. As both the memory and the sequence contains information about the sequence items, the multi-way attention in Section 4.2 allows for intra-sequence dependency modeling.
The architecture allows us to model co-occurence beyond the $(target\ item, sequence\ item)$ pair. For example, the target item is rum and the user sequence contains $lime$ and $peppermint$. With the target attention mechanism, both the attention weight between the pair $(peppermint, rum)$ and that between the pair $(lime, rum)$ are not high. With our IMN architecture, after the first pass of sequence walk, the memory vector contains information about both the $peppermint$ and rum. When it meets the item $lime$ again, the memory of $peppermint$ and rum awakens the item $lime$ since the triplet $(rum, peppermint, lime)$ is the recipe for Mojito and likely to co-occur multiple times in different samples. Hence, with $lime$ and $peppermint$ in the sequence, the likelihood to click rum increases. While the target item mechanism is to find the items that co-occur frequently with the target item, our IMN finds what the composite group of the user's behavior items is like for the user to click the target item.
As the experimental results and analysis in Section 5 show, the number of sequence walk iterations does not exceed 4 for both the industrial and public datasets. Hence our IMN incurs only linear memory and time complexity, compared to $O(L^2)$ complexity for the self-attention mechanism.
Additionally, our IMN is different from the typical Memory Networks architecture. The typical architecture updates the memory with each incoming input, and uses the updated memory for computations on the next input. It is difficult to parallelize the sequential operations during the prediction phase, and thus infeasible to deploy on recommmender systems that need real-time responses.

4.4 MEMORY ENHANCEMENT MODULE

After N iterations of the memory update process, the user memory vector encapsulates both the information in the sequence relevant to the target item $\boldsymbol{v}_T$ and the intra-sequence information. The Memory Enhancement Module enhances the user memory with the target item repeatedly to elicit more clear memory specific to the target item and remove noises.
We use a GRU to model the repeated elicitation process. The GRU's initial state is initialized from the memory after the Iterative Memory Update module, $\boldsymbol{u}_0 = \boldsymbol{m}^N$. For each step, we apply a linear transformation W^u on the GRU's last hidden state, concatenate the transformed vector with the target item, and use the concatenated vector as the GRU's input.

$$\boldsymbol{u}^t = GRU([W^u \boldsymbol{u}^{t-1}, \boldsymbol{v}_T], \boldsymbol{u}^{t-1}) \tag{7}$$

where $\boldsymbol{u}^t$ is the user representation after t steps in the Memory Enhancement Module.

4.5 OTHER RELEVANT EXPERIMENTS

Dot Product Multi-way Attention. Instead of using the multi-way attention with Euclidean distance and Hadamard distance, we have also experimented with the dot product distance $\alpha(\boldsymbol{e}, \boldsymbol{v}, \boldsymbol{m}) = [\boldsymbol{e} \cdot \boldsymbol{m}, \boldsymbol{e} \cdot \boldsymbol{v}]$. It is as expected that the dot product attention does not perform as well as the additive attention.
Sequence Segmentation & Hierarchical Memory Network. We segment the long sequence of length 1000 into multiple sub-sequences of length 100. For each sub-sequence, we use the multi-way attention mechanism to compute the memory vector on the sub-sequence. The user memory vectors produced are treated as another sequence for another iterative memory update process. There is no improvement on the metrics.

5 EXPERIMENTS

In this section, we present our experiments in detail, with datasets, experimental setups, evaluation metrics, model comparisons, experimental results and corresponding analysis.

5.1 DATASETS AND EXPERIMENTAL SETUP

Amazon Dataset. We collect two subsets from the Amazon product data (McAuley et al., 2015), Books and Movies. The Amazon Books dataset contains 295982 users, 647589 goods and 6626872 samples. The Amazon Movies dataset contains 233282 users, 165851 goods and 4829693 samples. We split each dataset into 80% training and 20% test data according to the behavior timestamp. We use the Adam optimizer (Kingma & Ba, 2015), 0.001 learning rate and the mini-batch size is 512. We use 20G memory, 2 parameter servers and 4 workers for all experiments. The size of MLP layers is 128×64.

Industrial Dataset. We collect traffic logs from the an industrial recommender system. We use 30-day samples for training and the samples of the following day for testing. With 0.1 sampling on negative samples, the total number of training samples is 1.68 billion. The ratio of positive to negative samples is 1:2.24 in the training set. The test set contains 57 million data points. The dimensions of embedding vectors are 8 for categorical features. The size of MLP layers is $512 \times 256 \times 128$. The dimensions of the GRUs' hidden states for both the Iterative Memory Update module and the Memory Enhancement module are 32. The mini-batch size is 512. We use Adam as the optimizer (Kingma & Ba, 2015). The learning rate is 0.0001. We use 75G memory and 50 GPU for each worker. We use 5 parameter servers and 50 workers for all experiments.

Evaluation Metric. We use Area Under the Curve (AUC) as the metric for performance measurements.

5.2 MODEL COMPARISON

- **YouTube DNN**. YouTube DNN uses average pooling to integrate behavior embeddings to fixed-width vectors as the user's interest representation (Covington et al., 2016).

Table 1: Model performance (AUC) on public and industrial datasets

Model	Amazon Books		Amazon Movies		Industrial	
	AUC	Impr	AUC	Impr	AUC	Impr
Youtube DNN (Base)	0.8374	0.00	0.8343	0.00	0.7353	0.00
DIN	0.8516	1.42	0.8603	2.6	0.7375	0.22
DIEN	0.8550	1.76	0.8654	3.22	0.7381	0.28
SASRec	0.8214	-1.60	0.8221	-1.22	0.7346	-0.07
MIMN	0.8523	1.49	0.8714	3.71	0.7377	0.24
IMN 2P	0.8498	1.24	0.8821	4.78	0.7394	0.41
IMN 3P	0.8672	2.98	0.8835	4.92	0.7408	0.55
IMN 3P+	0.8692	**3.18**	0.8862	**5.19**	0.7423	**0.7**

- **DIN**. DIN proposes the target attention mechanism to soft-search user sequential behaviors with respect to the target item (Zhou et al., 2018).
- **DIEN**. DIEN integrates GRU with the target attention mechanism to model user interest evolutions (Zhou et al., 2019).
- **SASRec**.SASRec is a self-attention based sequential model. It outperforms the recurrence-based models, with significantly less training and inference time (Kang & McAuley, 2018).
- **MIMN**. MIMN uses fixed number of memory slots to represent user interests. When a new click takes place, it updates the user memory slots with a GRU as the controller (Pi et al., 2019).
- **IMN 2P/3P**. Our Iterative Memory Network models, without the Memory Enhancement module. 2P refers to 2 iterations of the memory update process, and 3P refers to 3 iterations.
- **IMN 3P+**. 3 memory update iterations, with the Memory Enhancement module. The number of steps t is 3 for the Memory Enhancement module.

5.3 Experimental Results

Table 1 presents the results on model performance. For all three datasets, the sequence length used is 1000 for all models except SASRec. Initially, we use sequences of length 1000 for SASRec, but incurs out of memory error during training. Since SASRec is a self-attention based method, it incurs $O(L^2)$ memory complexity. It is infeasible to store a matrix of the dimension $N \times L \times L \times d$ where N is the batch size, L is the sequence length and d the dimension of the model embedding. Thus we reduce the sequence length to 100 for SASRec, which explains for its below-the-baseline model performance.

DIN outperforms Youtube DNN since it assigns weights to user behaviors according to the target item. Since DIEN integrates GRU with the attention process to model user interest evolutions, it outperforms DIN. MIMN, the memory-network-based architecture, performs better on the Amazon Movies dataset compared to DIEN, but worse on the Amazon Books and industrial datasets. Since SASRec uses the self-attention mechanism, the memory bottleneck limits its scalability to longer sequences and explains for its worse performance compared to the baseline.

IMN 2P outperforms all other methods for the Amazon Movies and the industrial datasets. IMN 3P, having one more memory update iteration, outperforms all other methods for all three datasets. The ability to model both intra-sequence dependency and target-sequence dependency in linear complexity allows for its superior performance. With the Memory Enhancement module, IMN 3P+ outperforms IMN 3P. It also serves as an ablation test for the Memory Enhancement module.

5.4 Analysis

Since we propose that performing multiple iterations of sequence walks benefits the model performance, we analyze the changes in AUC as the number of iterations increases. We observe from Fig.(2) that for all three datasets, the AUC first increases with the number of sequence walks, showing that intra-sequence dependency modeling benefits the model performance. Another observation is that AUC improvements diminish as the number of iterations increases. The AUC curves plateau

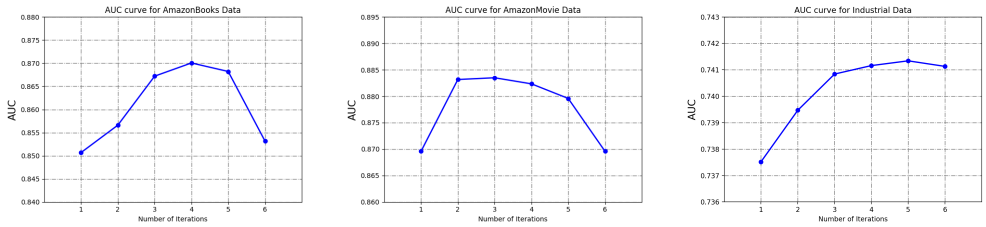

Figure 2: AUC on the test dataset against the number of memory update iterations. The three plots are for the Amazon Books dataset, the Amazon Movies dataset and the industrial dataset.

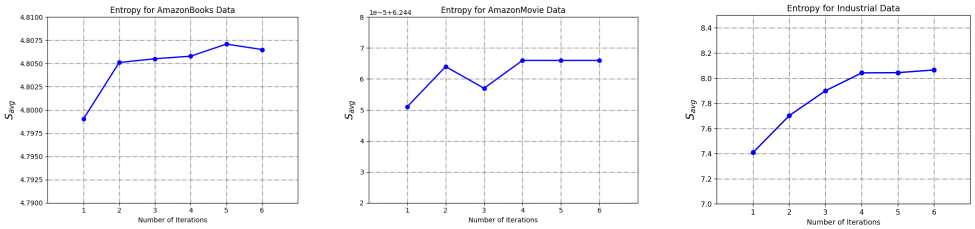

Figure 3: Average entropy per batch S_{avg} against number of memory walk iterations. The data is collected after the algorithm has converged, for the Amazon Books dataset, and the Amazon Movies dataset and the industrial dataset respectively.

after at most 4 iterations for all three datasets. For the Amazon Books and Movies datasets, AUCs drop significantly after 5 iterations, probably due to over-fitting with an increased number of model parameters.

Since we propose that the framework models intra-sequence dependencies, we expect the attention weights distributions to change with an increase in the number of iterations. We use entropy to measure information in the attention weights distribution and define *average entropy per batch* S_{avg} as the following to measure the entropy within a batch.

$$S_{avg} = \frac{\sum_j^N (-\sum_i (a_i / \sum a_i \times \log_2 (a_i / \sum a_i + \epsilon)))}{N} \tag{8}$$

where a_i denotes the attention weight for the i_{th} item in the user behavior sequence and N denotes the batch size. We add a small value $\epsilon = 10^{-12}$ on the nominator to prevent zero-values in our implementation.

Fig.(3) shows S_{avg} against the number of iterations. We observe that for all three datasets the entropy increases as the number of iterations increases. The entropy increase diminishes as the number of iterations further increase. After 4 to 5 iterations, there is negligible increase in entropy. This coincides with the trend in AUC improvement, showing that at most 4 passes of sequential walks are sufficient for the optimal model performance.

To visualize the attention weights for the multi-way attention as the number of iterations increases, we provide an example and plot the attention weights heatmap in Fig.(4). As seen in Fig.(4), the target item is a sunscreen with SPF 50. During the 1st iteration, the lipstick and the eye-shadow palette have the highest attention weights. This is expected because the sunscreen, the lipstick and the eye-shadow palette are in the Cosmetics category. During the 2nd iteration, the attention weights for travel backpacks, travel-size toiletries, and the bluetooth selfie stick increase. This coincides with the rationale for our design. During the 1st iteration, the Iterative Memory Network behaves like the target attention mechanism, modeling only the relations between the sequence items and the target item. From the 2nd iteration onwards, the Iterative Memory Network is able to model intra-sequence dependencies, with travel-related information encapsulated in the memory vector. Thus when it encounters travel-related products such as the bluetooth selfie stick again, the travel

Figure 4: The user behavior sequence contains 8 items. The target item is a sunscreen with SPF 50. The heatmap shows the attention weights for the 8 items during the 1st and 2nd iterations of the Iterative Memory Network. Attention weights for each iteration have been normalized to sum to 1.

Table 2: Complexity, minimum number of sequential operations, maximum path lengths for different methods. L is the sequence length, d is the representation dimension.

Method	Complexity	Sequential Operations	Maximum Path Length
DIN	$O(L \cdot d)$	$O(1)$	$O(\infty)$
DIEN	$O(L \cdot d)$	$O(L)$	$O(L)$
SASRec	$O(L^2 \cdot d)$	$O(1)$	$O(1)$
MIMN	$O(L \cdot d)$	$O(L)$	$O(L)$
IMN	$O(L \cdot d)$	$O(1)$	$O(1)$

information is reinforced and the attention weight for the bluetooth selfie stick increases.
We summarize the complexity, minimum number of sequential operations and maximum path length of various methods in Table 2. The minimum number of sequential operations measures the amount of computations that can be parallelized. Pure attention-based methods are at $O(1)$, while methods involving recurrence are at $O(L)$. Maximum path length refers to the length of the paths signals need to traverse. SASRec is at O(1) due to the pairwise comparisons in the self-attention mechanism. Our IMN is at O(1) as well because information related to the target item is encapsulated the memory vector. The multi-way attention calculates the attention weights between the memory vector and the sequence items hence allows for intra-sequence message passing. In fact, the IMN architecture is a target item triggered, intra-sequence dependency and target-sequence dependency modeling method.

6 Conclusion

The self-attention mechanism captures long-range intra-sequence dependencies, but incurs $O(L^2)$ memory complexity that limits the scalability to long sequences. The target attention mechanism, despite having only $O(L)$ memory complexity, can not model intra-sequence dependency. Current memory-network-based models in recommender systems utilize the Neural Turing Machine architecture directly, using RNN-based controllers to update the memory with each input. It is difficult to parallelize the computations, making it difficult for training and deployment.
Our Iterative Memory Network, by walking the sequence multiple iterations, models intra-sequence dependencies and target-sequence dependencies. It models co-occurrence beyond the $(target\ item, sequence\ item)$ pair to triplets, quadruplets and so on. It only incurs $O(L)$ memory and time complexity. The attention weights computations are parallel, making it efficient for training and prediction. We have demonstrated its superior performance in modeling long user sequential behaviors with respect to various state-of-the-art architectures in recommender systems. The IMN model has been deployed successfully in the industrial recommender system, bringing significant improvements in the click-through rate.

REFERENCES

Deepak Agarwal, Bee-Chung Chen, and Pradheep Elango. Spatio-temporal models for estimating click-through rate. In *WWW*, 2009.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *arXiv:1409.0473*, 2016.

William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. Listen, attend and spell. In *arXiv:1508.01211*, 2015.

Sneha Chaudhari, Varun Mithal, Gungor Polatkan, and Rohan Ramanath. An attentive survey of attention models. In *arXiv:1904.02874*, 2021.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. In *arXiv:1904.10509*, 2019.

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Recsys. 191–198*, 2016.

Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. Attention-over-attention neural networks for reading comprehension. In *arXiv:1607.04423*, 2016.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. In *arXiv:1410.5401*, 2014.

Balazs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *ICLR*, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. In *Neural computation, 9(8):1735–1780*, 1997.

Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *CIKM*, 2018.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *Proceedings of 3rd International Conference on Learning Representations*, San Diego, CA, USA, 2015.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *ICLR*, 2020.

Yehuda Koren. Collaborative filtering with temporal dynamics. In *KDD*, 2009.

Ankit Kumar, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *ICML*, 2016.

Xiaoya Li, Yuxian Meng, Mingxin Zhou, Qinghong Han, Fei Wu, and Jiwei Li. Sac: Accelerating and structuring self-attention via sparse adaptive connection. In *NeurIPS*, 2020.

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. In *arXiv:1703.03130*, 2017.

Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton 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.

Niki Parmar, Prajit Ramachandran, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. In *Advances in Neural Information Processing Systems 32, pages 68–80*, 2019.

Qi Pi, Weijie Bian, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. Practice on long sequential user behavior modeling for click-through rate prediction. In *Proceedings of the 25th ACM SIGKDD*, 2019.

Govardana Sachithanandam Ramachandran and Ajay Sohmshetty. Ask me even more: Dynamic memory tensor networks. In *arXiv:1703.03939*, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30, pages 5998–6008*, 2017.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In *arXiv:1410.3916*, 2014.

Sai Wu, Weichao Ren, Chengchao Yu, Gang Chen, Dongxiang Zhang, and Jingbo Zhu. Personal recommendation using deep recurrent neural networks in netease. In *IEEE International Conference on Data Engineering, 1218–1229*, 2016.

Caiming Xiong, Stephen Merity, and Richard Socher. Dynamic memory networks for visual and textual question answering. In *arXiv:1603.01417*, 2016.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

Chang Zhou, Jinze Bai, Junshuai Song, Xiaofei Liu, Zhengchao Zhao, Xiusi Chen, and Jun Gao. Atrank: An attention-based user behavior modeling framework for recommendation. In *AAAI*, 2017.

Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 1059–1068*, 2018.

Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. Deep interest evolution network for click-through rate prediction. In *Proceedings of the 33nd AAAI Conference on Artificial Intelligence*, 2019.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *AAAI*, 2020.