World Wide Web (2023) 26:561–583
https://doi.org/10.1007/s11280-022-01056-9

Long short-term enhanced memory for sequential recommendation

Jiasheng Duan[1] · Peng-Fei Zhang[1] · Ruihong Qiu[1] · Zi Huang[1]

Received: 4 October 2021 / Revised: 15 February 2022 / Accepted: 12 April 2022 /
Published online: 13 May 2022

Abstract

Sequential recommendation is a stream of studies on recommender systems, which focuses on predicting the next item a user interacts with by modeling the dynamic sequence of user-item interactions. Since being born to explore the dynamic tendency of variable-length temporal sequence, Recurrent Neural Networks (RNNs) have been paid much attention in this area. However, the inherent defects caused by the network structure of RNNs have limited their applications in sequential recommendation, which are mainly shown on two factors: RNNs tend to make point-wise predictions and ignore the collective dependencies because the temporal relationships between items change monotonically; RNNs are likely to forget the essential information during processing long sequences. To solve these problems, researchers have done much work to enhance the memory mechanism of RNNs. However, although previous RNN-based methods have achieved promising performance by taking advantage of external knowledge with other advanced techniques, the improvement of the intrinsic property of existing RNNs has not been explored, which is still challenging. Therefore, in this work, we propose a novel architecture based on Long Short-Term Memories (LSTMs), a broadly-used variant of RNNs, specific for sequential recommendation, called **Long Short-Term enhanced Memory (LSTeM)**, which boosts the memory mechanism of original LSTMs in two ways. Firstly, we design a new structure of gates in LSTMs by introducing a "Q-K-V" triplet, a mechanism to accurately and properly model the correlation between the current item and the user's historical behaviors at each time step. Secondly, we propose a "recover gate" to remedy the inadequacy of memory caused by the forgetting mechanism, which works with a dynamic global memory embedding. Extensive experiments have demonstrated that LSTeM achieves comparable performance to the state-of-the-art methods on the challenging datasets for sequential recommendation.

Keywords Sequential recommendation · LSTM · Memory enhanced

✉ Jiasheng Duan
j.duan@uqconnect.edu.au

Extended author information available on the last page of the article

1 Introduction

Information explosion has fuelled the rapid development of recommendation techniques, since online products and services, such as e-business, social networking sites, and news media, are in urgent need of effective and practical recommender systems [1–3]. Recommender systems are a class of information filtering engines that aim to predict a user's preference on items, for the purpose of solving the problems of information overload and information management difficulty caused by the sharp growth of the amount of available data [4, 5].

Hence, in recent years, plenty of works on recommender systems have emerged, which are proposed to accurately identify and extract the specific information of users' needs and intentions that are generally implicitly expressed in users' behaviors. One stream of these studies focuses on sequential recommendation, which models a recommendation task as a successive retrieval problem based on a dynamic sequence of user-item interactions. Specifically, given a chain of items that a user consecutively interacts with, such as clicking, purchasing and watching, a sequential recommender system seeks to capture this user's interest and to properly recommend the next item the user prefers at each time step.

The early works on sequential recommendation mostly merely succeed in short-term recommendation, but fail in long-term prediction. To fill this gap and apprehend users' long-term interest, professionals have paid attention to deep neural networks, especially Recurrent Neural Networks (RNNs). RNNs are born to explore the dynamic tendency of variable-length temporal sequences. They model sequences along their temporal directions and leverage the internal hidden states to compile and memorize the ordered data entities. Hence, RNNs, including the popular variants, LSTMs (Long Short-Term Memories) [6, 7] and GRUs (Gated recurrent units) [8], have been broadly adopted in sequential recommendation. The first RNN-based work in this task [9] utilizes the multi-layer GRU-based model, with session-parallel mini-batches, to predict the next item within sessions. Over the next years, an abundance of RNN-based studies sprang up and almost dominated the deep learning-based sequential recommendation area. For example, [10] employs GRUs in the session-based recommendation, and devises a new set of list-wise loss functions and a sampling strategy that provides top-k gains to improve the prediction.

However, RNN-based architectures are not flawless for sequential recommendation. Although the variants, LSTMs and GRUs, have made great strides in improving the memory and regulating the storage by the gate mechanisms, the long-term memory of RNNs needs to be further enhanced for sequential recommendation, because they still struggle to capture the complex semantic dependencies among elements in users' action histories. Among various others, there are two obvious factors that contribute to this situation are the two most obvious ones.

Firstly, RNNs assume the temporal relationships between items change monotonically [11], *i.e.*, any items in a sequence have a much heavier dependency with their neighbors than with other points. Due to this, RNNs tend to make point-wise predictions [1] that heavily depend on a single neighbor, and ignore the collective dependencies among all items. Nonetheless, in the real world of recommendation, the item, interacted with by a user at the current time, may be more relevant to multiple other items in a very early time separated from itself than the adjacent one. For example, in the history of a user's behaviors, a mouse could be related to several previous purchases, inclusive of his computer, keyboard and even earphones, but irrelevant with the shoes he/she just bought before the mouse.

Another contributing factor is that RNNs are likely to forget the essential information during processing long sequences. The vanilla RNNs suffer from gradient vanishing and exploding when dealing with long-term information, so LSTMs and GRUs are designed to be equipped with the forget gate mechanism that allows the model to remember the key points and forget the insignificance along with the prediction procedure. However, this mechanism is more applicable for the sequences with only one or few key points, *e.g.*, sentences, but does not work well for the sequences with plentiful key points, *e.g.*, users' behavior sequences, in which, the dynamics of users' interests produce the complex dependencies among the items, and further generate many key points in a sequence. This is because the forget gates make the model abandon the information irrelevant to the current and very recent states, but may be useful later. Hence, this operation results in information loss in processing the items, and concludes the fake relationships and dependencies.

To remedy these defects, researchers have attempted to incorporate other advanced techniques into RNN-based models [12–14] to boost the memory of RNNs. For instance, [12] combines the global and local RNN-based encoder-decoder architectures with the attention mechanism, so that previous sessions can be involved into the next item prediction to retain both of the long- and short-term important information. In [15], a Recurrent Convolutional Neural Network (RCNN) is created which employs the recurrent architecture of RNNs to remember long-term dependencies and adopts the convolutional operation of Convolutional Neural Networks (CNNs) to retrieve short-term relationships among recurrent hidden states. [13] builds a parallel RNN-based architecture to exploit rich features of items, such as images, texts and videos, in multiple branches, to enhance the knowledge of the recommender system. [14] integrates an external memory based on Key-Value Memory Networks (KV-MN) [16] into a GRU network, which frames the item attributes into pairs of key-value vectors, and employs the outputs of RNN cells as the query to extract the relevant representations into the final prediction.

Nevertheless, these works focus on attaching one or several external components (such as attention, CNNs and KV-MV) to the traditional RNN architectures, in order to take advantage of the external knowledge. The improvement of inner structures of RNNs, including LSTMs and GRUs, is still challenging. Hence, fuelled by the desire for effectively and fundamentally overcoming the challenge, we propose to address these issues via a novel network based on an LSTM, called **Long Short-Term enhanced Memory (LSTeM)**, specific for Sequential recommendation, which boosts the memory mechanism of original LSTMs in two ways.

Firstly, we design a new structure for the gates in LSTMs. The gates are internal components in original LSTMs that regulate the flow of information by deciding which part of data information in a sequence to input, keep or discard. Specifically, they act as filters to take in and pass down the important information, *i.e.*, the information relevant to the key points, and to delete other information considered unnecessary to the decision-making process, based on the situation at each time step. The traditional gates are represented as the activated sum of the weighted current input and the hidden states, which input and discard information without considering the specific context at each time step. Inspired by [17], we replace the original addition operations with the product of query-key pairs of the "Q-K-V" triplet, which maps each embedding into a query, a key and a value, and express the dependencies of two items by the compatibility of one's key and the other's query. From the other perspective, as discussed above, a sequence is modeled into a directed graph along the temporal dimension in RNNs, so the query-key pair can be seen as a graph kernel function that transforms the data entities to a common latent space and then calculates their correlation by inner products. Thereby, the new gates are more able to dynamically capture the relationship between the current input and the historical memory at each time step, instead of simply taking their summation. To this end, the personalized item information is introduced at different steps, and the correlation between the item at each step and the previous items are extracted and represented more flexibly and more comprehensively. In other words, the model processes the information in both the item and the memory depending on the relationships instead of mechanically excessively emphasizing the adjacent items. Hence, this strategy alleviates the first problem of RNNs that tends to make point-wise predictions and ignore the collective dependencies.

Secondly, we design a "recover gate" to assist the model to find back the lost information discarded in previous steps, but found useful currently. This recover gate is served with a dynamic global memory embedding generated by a memory function, which is endowed with a Self-Attention mechanism to provide an aggregated expression of all the previous items to the current item. Self-Attention is a promising mechanism designed to summarize the information of a sequence based on the dependencies among its elements [17], through mimicking human's cognitive attention that strengthens the important parts of the data features and the inner correlations within a sequence and fades out the rest. Hence, the key information of a user's sequence can be condensed into a single vector, which is then used by the recover gate.

The main contributions of this paper are listed as follows:

- To the best of our knowledge, LSTeM is the very first work to enhance the memory mechanism of LSTMs by proposing a new internal structure of LSTMs specific for sequential recommender systems. The "Q-K-V" mechanism is introduced to design the new gates of LSTMs to boost the ability to capture the information and relationships between items and memories. A recover gate with dynamic global memory embeddings is integrated into the novel structure.
- Extensive experiments on four popular sequential recommendation datasets have been conducted to demonstrate the promising performance of LSTeM by comparing it with 9 state-of-the-art methods.

The rest of this paper is organized as follows: In Section 2, we will discuss the related work; in Section 3, we will explicitly introduce the proposed approach LSTeM; in Section 4, the extensive experiments will be discussed, followed by the conclusion of the whole paper in Section 5.

2 Related work

2.1 Sequential recommendation

The existing sequential recommendation works can be generally separated into three categories, traditional methods, latent representation approaches and deep neural network-based models [1]. The traditional methods proposed in the early time mainly focus on frequent pattern mining and Markov Chains (MC), a stochastic model to learn sequential patterns and predict the next item, such as [18, 19]. The Markov Chains seem innately applicable to tackle the next item prediction, but this stochastic model only considers the relationships between adjacent items in an interaction history, so MC-based works merely succeed in short-term recommendation [1], but fail in long-term prediction.

After that, latent representation approaches sprung up, and hence, researchers shifted their attention to factorization and embedding learning, among which Matrix Factorization (MF) is a popular algorithm. MF decomposes the user-item interaction matrix into the product of two lower dimensionality rectangular matrices to identify the latent correspondence between a user factor and an item factor [20]. Hence, in deep learning-based recommender systems [21–24], it is a natural thought to build a two-branch network to predict the interaction rate between users' interest representations and items features.

In recent years, driven by the development of deep learning, a large number of neural network-based methods have been proposed in this study domain. The commonly adopted techniques include Recurrent neural networks (RNNs), Convolutional neural networks (CNNs), and Graph neural networks (GNNs).

CNN-based methods, such as [25, 26], first assemble the item embeddings into a matrix and implement convolution operations on it. There is an obvious drawback in this class that CNNs are not able to adequately capture the orders in sequences, so the intrinsic sequential relationships are more likely to be ignored. By contrast, GNNs are more strong to model temporal data. The typical GNN-based methods on sequential recommendation tasks build a directed graph, in which the nodes represent items, and the edges are deemed as the dependencies, so that graph theories can be used to analyze the relationships of these components. In that way, a GNN-based model is able to explore complex and implicit relationships among users and items. Hence, this class has a potential on explainable sequential recommendation. RNN-based methods have dominated in the area of sequential recommendation for a long time, since RNNs are born to handle sequential data, by sequentially modeling items in their cells, in order to dig latent patterns. The popular variants of RNNs, LSTMs and GRUs are frequently used in this task, such as [9, 10, 12, 24, 27–29]. Nevertheless, as discussed in section 1, RNN-based models still have some drawbacks to overcome for sequential recommendation.

To overcome the limitations, a number of advanced hybrid methods that combine the three basic classes and other techniques have been devised. Attention mechanism is one of the most popular techniques to be adopted in sequential recommendation. In [30], the authors propose a two-layer hierarchical attention network to properly capture the dynamic users' preference, in which the first layer learns the long-term features, while the second layer couples long-term and short-term patterns and outputs users' representation. [31] applies the Self-Attention mechanism created for transformers to allow the model to consider long-range dependencies on dense data, and to pay more attention to recent activities on sparse distribution. Memory networks are another class of techniques to enhance the model's memory capacity, which ordinarily merge an external memory matrix into the original model. For example, [14] adds a Key-Value Memory Network to RNN-based networks, which splits a memory slot into a key vector and a value vector and integrates the key-value pairs with GRU outputs to enhance the semantic representation and capture the sequential user preference at the same time.

2.2 Recurrent neural networks (RNNs)

Recurrent Neural Networks (RNNs) are a class of unit-based models for tackling sequences of data entities. The main idea of RNNs is to model a sequence into a directed graph along the temporal dimension, and retrieve the information step by step using the same unit (cell), so that the dependencies among the elements of sequences can be accurately captured. However, vanilla RNNs are not applicable for long-term sequences, and hence, to better deal with the long-term dependencies, two variants of RNNs, the Long-Short Term Memories (LSTMs) [6, 7] and the Gated Recurrent Units (GRUs) [8], have been proposed and widely applied, which are featured by the internal memory mechanism. In this paper, we focus on LSTMs, which have an edge over vanilla RNNs due to the property of selectively remembering and forgetting information for both long and short periods of time. An LSTM unit is a neural network containing a group of repeating cell modules, whose main components are three gates, including an input gate, a forget gate and an output gate. The gates serve as filters to determine what information should be allowed to get into the system and be retained or dropped. In this way, only the valid information is passed down the sequence. Nevertheless, there are still a part of useful elements inevitably discarded ahead of the time when they are needed, or even have not gotten into the memory. This is a natural defect of LSTMs, and hence, to allow this class of models to be used in sequential recommendation, many methods have been devised to fill this gap in applications. For example, in [12–15, 32], authors incorporate other advanced techniques, such as attention mechanism, Key-Value Memory Networks and CNNs into RNNs to achieve better performance in sequential recommendation.

2.3 Self-attention

First introduced for language processing, the promising mechanism, Self-Attention, has seen widespread use in deep learning, and the applications are not only for NLP, image processing, but also for recommender systems. The main idea of the general attention is to capture intrinsic relationships of the relevant data entities while ignoring others, which works in a similar way to human brain actions. In the early articles, people usually use attention mechanism as a component to build models with other technique frameworks, such as attention with RNNs [12, 16], attention with Matrix Factorization, etc. In these works, the important information is properly extracted, summarized and emphasized. In comparison, in recent studies, people apply attention mechanism in transformers, an independent method to deal with the components within an individual sequence. This Self-Attention-based sequence-to-sequence model relates elements in different positions to each other in a sequence, depending on their dependencies, in order to gain the aggregated representation of this sequence or every element.

For instance, [31] constructs a Self-Attention-based model that adds adaptive weights to the current item according to its relationships with previous items at each time step. [33] employs the deep bidirectional Self-Attention paradigm, BERT [34], to model users' behavior sequences and predict the random masked items by jointly depending on their left and right context. On the foundation of it, [35] merge side information of items into the queries and keys of BERT framework to generate better attention distribution so that information overwhelming can be avoided.

3 Long short-term enhanced memory

In this section, we will show the details of our proposed Long Short-Term enhanced Memory (LSTeM) for sequential recommendation. We will first briefly introduce the key ideas of this method and the problem formulation, and then describe the structure of LSTeM in detail.

3.1 Motivation

Previous works [12, 13, 15, 36, 37] have already shown that RNN-based models, including LSTMs, can carry out sequential recommendation tasks, and they have made attempts to introduce other advanced techniques to solve the inherent problems of RNNs. These paradigms work as external components, so that the performance of the models can be boosted by taking advantage of the external knowledge. In this work, we consider to overcome these challenges by improving the intrinsic property of the RNNs and propose a Long Short-Term enhanced Memory (LSTeM) to enhance the performance of LSTMs in sequential recommendation. Specifically, we merge the "Q-K-V" mechanism, which is conventionally applied in Self-Attention, into LSTM cells to create a novel structure of gates and improve the model's memory. In addition, we add a component, the recover gate with a global memory embedding, which is a Self-Attentional representation of previous items in a sequence, to find back the lost but useful information. The whole architecture of LSTeM is illustrated in Figure 1.

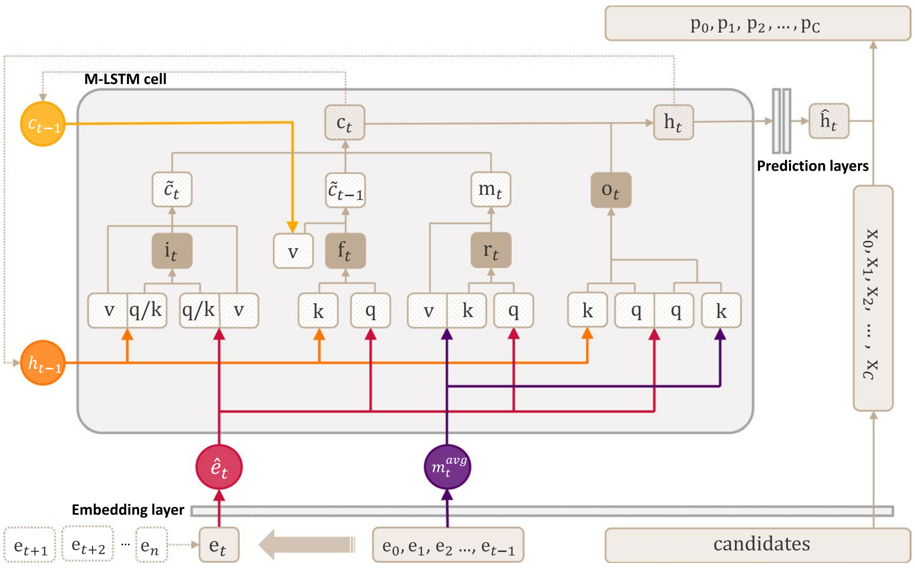

Figure 1 The architecture of LSTeM

3.2 Problem formulation

A dataset of users' historical actions consists of M users' temporally ordered user-item interaction sequences, which is denoted as $\left\{S^{(u)}\right\}_{u=1}^{M}$. A sequence comprises k items and is denoted as $S^{(u)} = [e_1, e_2^{(u)}, \cdots, e_k^{(u)}]$, where k is a variable number, which means that the length of a sequence is dynamic. Each element of $S^{(u)}$ is denoted as $e_t^{(u)} \in \left\{e_j\right\}_{j=1}^{N}$, where $\left\{e_j\right\}_{j=1}^{N}$ is the set of all N items in the dataset, and t is the time step index of the item being interacted with in a user's behavior sequence. In sequential recommendation, given a user's interaction history with t time steps, the model aims to predict $e_{t+1}^{(u)}$, *i.e.*, the item this user interacts with at the next step $t + 1$.

3.3 Framework

As described in Sects. 1 and 2, an original LSTM model is a neural network composed of a chain of repeating cell modules that allows a sequence of items to pass into the model one by one. Each cell has a three-gate structure to decide what information to take in, remove or retain. Specifically, given an input x_t at time t, and the hidden state h_{t-1} and memory c_{t-1} from last time step, the model works as follows:

$$i_t = \sigma_s(U^{(i)}x_t + W^{(i)}h_{t-1}), \tag{1}$$

$$f_t = \sigma_s(U^{(f)}x_t + W^{(f)}h_{t-1}), \tag{2}$$

$$o_t = \sigma_s(U^{(o)}x_t + W^{(o)}h_{t-1}), \tag{3}$$

$$\tilde{c}_t = \sigma_t(U^{(c)}x_t + W^{(c)}h_{t-1}), \tag{4}$$

$$c_t = f_t\, c_{t-1} + i_t\, \tilde{c}_t, \tag{5}$$

$$h_t = o_t\, (\sigma_t(c_t)), \tag{6}$$

where i_t, f_t and o_t are the input (*i.e.*, update) gate, the forget gate and the output gate, respectively; c_t and c_{t-1} are the long-term memory cell state at step t and step $t-1$; σ_s and σ_t are sigmoid function and hyperbolic tangent function. Here, the three gates (Eq. 1, 2 and 3) and the current memory cell $\tilde{c}_t$ (Eq. 4) are represented as the activated sum of the weighted current input x_t, and last hidden state h_{t-1}. In Eq. 5, the forget gate f_t multiplies c_{t-1} to determine what to discard from the history memory, and the input gate i_t times $\tilde{c}_t$ to compute what to input into the memory, while the sum of these two terms is the current memory cell state c_t. The current hidden state h_t is the product of the output gate o_t and activated c_t (Eq. 6). Here, $U^{(*)}$ and $W^{(*)}$ are the parameters of the model.

In LSTeM, we employ a novel LSTM architecture as the basis. Each novel cell is composed of four gates, including an input gate, a forget gate, a recover gate and an output gate. Among them, the input gate, the forget gate and the output gate are variants of LSTMs' original components, while the recover gate is a new part fused into the original structure.

Concretely, for a user's action history $S^{(u)}$, at time step t, the item $e_t^{(u)}$ in the sequence are firstly converted into an embedding and then fed into the cell, together with the hidden state h_{t-1} and long-term memory c_{t-1} from the previous time step, as well as a global historical memory embedding m_t generated by a Self-Attention model. The cell computes the four gates and the corresponding values using the "Q-K-V" mechanism, and then outputs the updated hidden state h_t and the updated memory c_t. The hidden state h_t can be regarded as the representation of the user's interest at the current time t to predict the next item. The details are described in the following sections.

3.3.1 Embedding layer

To allow data instances to pass into the model in mini-batches, we first transform each user's behavior history to a fixed-length embedding sequence $E^{(u)} = [\hat{e}_1^{(u)}, \hat{e}_2^{(u)}, \cdots, \hat{e}_n^{(u)}]$, where n is the fixed length of the sequence. If the number of historical points is less than n, we insert padding items in the front of the sequence, while if the number is larger than n, we only keep the last n items. We create an embedding matrix $E \in \mathbb{R}^{N\times d}$, where d is the latent dimension of the embeddings. Here, we use a constant vector *zero* to stand for a padding entity. We put the uniform-length sequences into an embedding lookup table that stores embeddings of all items in a dataset, to get the representation matrix. In summary, the embedding matrix of an individual sequence $S^{(u)}$ is denoted as $E^{(u)} \in \mathbb{R}^{n\times d}$.

3.3.2 The new structures of gates

As described above, the vanilla LSTMs have three gates, including an input gate, a forget gate and an output gate, and each gate is represented as $(U^{(*)}\hat{e}_t^{(u)} + W^{(*)}h_{t-1})$, where $U^{(*)}$ and $W^{(*)}$ are the parameters of the projection functions. Then the gates function as the filters to multiply the corresponding components, in order to accept or discard information. We argue that this operation is not adequate to perform the personalized information extraction and to convey the dynamic interests of users, in terms of a time step. This is because the addition of the two vectors (*i.e.*, the current interacted item and the historical memory) tends to present the summation of their semantic content, rather than the filtered information relevant to each other. In other words, a gate treats all information contained in an embedding equivalently, no matter how related it is to the situation at the current step, which results in that the adjacent item receiving excessive consideration (due to the short-term memory h_{t-1} is highly related to the adjacent item) while other information is ignored. In contrast, the "Q-K-V" mechanism selectively extracts the information from $\hat{e}_t^{(u)}$ and h_{t-1}), which emphasizes the correlation between the item at this step and the memory, and neglects the irrelevant content, in respect of the objective of the gate. Therefore, we use the "Q-K-V" mechanism to build the gates' structures.

The "Q-K-V" mechanism is a paradigm firstly introduced to Self-Attention (discussed in Section 2.3) by [17]. The main idea is to encode each element x_i within a sequence to a triplet made up of a query q_i, a key k_i and a value v_i, and then represent the correlation α_{ij} between two elements, x_i and x_j by the normalized product of one's query q_i and the other's key k_j, and output the weighted value $(\alpha_{ij}v_j)$. The formula for the whole sequence is

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{7}$$

where $Q = \{q_i\}_i$, $K = \{k_i\}_i$, $V = \{v_i\}_i$ and d_k is the dimensionality of the key K. In practice, *softmax* function is a commonly used compatibility function to normalize the attention weights, so that the sum of the elements' weights equals one. These normalized weights represent the relevant proportions of other elements.

In LSTeM, we leverage the "Q-K-V" mechanism to enhance the ability of capturing the correlation between the item embedding and the memory of the model. Specifically, at time t, given the current item $\hat{e}_t^{(u)}$, the hidden state h_{t-1} and the memory c_{t-1} from the previous step, and the global memory m_{t-1}^{avg} (explained in Section 3.3.3), we map these embeddings to different groups of query, key and value for the corresponding gates. It is worth emphasizing that we only apply the form of "Q-K-V" mechanism to capture the correlations and get the corresponding values in LSTeM instead of using the Self-Attention mechanism, because we only need to get the correlations of these embeddings and do not need to compute the relevant proportion of each pair of them. Hence, we substitute *sigmoid* function for the *softmax* function in Self-Attention to normalize the gate weights to [0, 1]. For the purpose of simplification and clearness, in the following part, we omit the annotation (u), and apart from the training parameters, all the components illustrated are for an individual user.

The input gate is in control of the information accepted into the memory at the current step derived from two sources, the input item $\hat{e}_t$ and the hidden state h_{t-1} (*i.e.*, the most recent historical memory). Therefore, the gate is defined as the sum of two products: one is the product of the query of $\hat{e}_t$ and the key of h_{t-1}, which indicates what information in the short-term memory is related to this item; on the contrary, the other is the product of the query of h_{t-1} and the key of $\hat{e}_t$, which presents what characteristics of this item interest the user. The "Q-K-V" triplet of $\hat{e}_t$ for the input gate can be denoted as $(q_{t,i,e}, k_{t,i,e}, v_{t,i,e})$, and the triplet of h_{t-1} is $(q_{t,i,h}, k_{t,i,h}, v_{t,i,h})$. Here, i represents the input gate, t is the time step, and e and h denote the original embedding of the triplets, which are the item embedding and the hidden state, respectively. However, for the sake of simplicity, we use a single vector to serve as both of the query and the key of $\hat{e}_t$ or h_{t-1} for computing the reciprocal dependency with them each other, so the simplified formula is

$$i_t = Sigmoid\left(\frac{qk_{t,i,e}\, qk_{t,i,h}}{\sqrt{d}}\right), \tag{8}$$

$$\tilde{c}_t = i_t\,(v_{t,i,e} + v_{t,i,h}), \tag{9}$$

where $qk_{t,i,e} = U^{(i)}\,\hat{e}_t$ is the query and key of $\hat{e}_t$; $qk_{t,i,h} = W^{(i)}\,h_{t-1}$ is the query and key of h_{t-1}, and d is the dimension of the embeddings; $v_{t,i,e} = P^{(i,e)}\,\hat{e}_t$ and $v_{t,i,h} = P^{(i,h)}\,h_{t-1}$ are the values. ($W^{(*)}$, $U^{(*)}$ and $P^{(*)}$ are trainable parameters.) In this way, the information accepted into the long-term memory $\tilde{c}_t$ can be computed as the sum of the two triplets of "Q-K-V" mechanism.

For the forget gate that is in charge of the information which needs to be removed from the long-term memory c_{t-1}, we also use a "Q-K-V" triplet to build it. To be specific, this operation is represented as

$$f_t = Sigmoid\left(\frac{q_{t,f,e}\, k_{t,f,h}}{\sqrt{d}}\right), \tag{10}$$

$$\tilde{c}_{t-1} = f_t\, c_{t-1}, \tag{11}$$

where $q_{t,f,e} = U^{(f)}\hat{e}_t$, $k_{t,f,h} = W^{(f)}h_{t-1}$, and "$f$" represents the forget gate. It should be noted that, here, we use the key of h_{t-1} to replace the key of c_{t-1}, in view of h_{t-1} is a selective projection of c_{t-1}, which emphasizes most recent information (explained in section 3.3.4). Besides, c_{t-1} is employed to be the value of itself.

In this manner, $\tilde{c}_t$ and $\tilde{c}_{t-1}$ get the semantic information imported from $\hat{e}_t$ and kept in h_{t-1}, respectively, which are specific for the current time step. In other words, they imply that "according to the user's current interest, which information should be noticed, obtained and passed down from this step and the most recent steps (*i.e.*, the short-term memory); and which part should be removed from the long-term memory."

Here, we rethink on the problems of RNN-based architectures for sequential recommendation discussed in Section 1. By the "Q-K-V" mechanism, the gates in LSTeM filter information depending on the dependencies between the current item and the historical memory, so that the personalized information of the item is introduced at different steps, and the connections between this item and the previous items are captured more flexibly and more comprehensively. That is to say, the model does not accept the same information from the item in different steps, and processes the information in the memory dynamically depending on the relationships instead of excessively emphasizing the adjacent item. Hence, this strategy alleviates the problem that RNNs tend to make point-wise predictions and ignore the collective dependencies among the whole sequence. However, it is still difficult for the model to tackle the long-term sequence, since the selectively updating and forgetting mechanism is naturally likely to abandon the information not needed in the recent and current time steps. To fill this gap, we design a recover gate, working with a global memory.

3.3.3 The recover gate

To allow the model to find back the memory lost along the way, we introduce a global memory that contains the integrated information extracted from the history to serve as the source for memory recovering. In order to fuse the global memory into the model, it is represented as a vector generated from all of the previous item embeddings in the sequence by using a Self-Attention method.

As described in Section 2.3, Self-Attention works within an individual sequence to relate elements in different positions, depending on their dependencies, to extract the aggregated representation of this sequence or every element. Given a sequence $X = \{x_i\}_{i=1}^{n}$ with n elements, for an element x_q, the aggregated expression is

$$\hat{x_q} = \sum_{i=1}^{n} \alpha(x_q, x_i)x_i, \tag{12}$$

where $\alpha(x_q, x_i)$ is an attention function to calculate the relevant weights of each x_i for x_q, based on the dependency between x_i and x_q.

In LSTeM, we use a simple Self-Attention manner to generate an integrated memory vector in LSTeM. At time t, given the embedding matrix of the previous items, $E_{t-1} \in \mathbb{R}^{(t-1)\times d}$, we first create the affinity that represents the dependencies among the individual items, and compute the weighted sequence embedding. Then we implement the Global Average Pooling (GAP) along the sequence to construct a compact global memory embedding m_{t-1}^{avg}. The formulas are

$$\alpha = softmax(E_{t-1}\ {E_{t-1}}^{T}), \tag{13}$$

$$\tilde{E}_{t-1} = \alpha\ E_{t-1}, \tag{14}$$

$$m_{t-1}^{avg} = GAP(\tilde{E}_{t-1}). \tag{15}$$

m_{t-1}^{avg} is an aggregated embedding of all the previous items, and aids the model to find back the lost history that has been discarded by the forget gate during the previous steps, but needed at the current step. Here, we continue to use the "Q-K-V" mechanism on $\hat{e}_t$ and m_{t-1}^{avg} to construct the recover gate:

$$r_t = Sigmoid\left(\frac{q_{t,r,e}\ k_{t,r,m}}{\sqrt{d}}\right), \tag{16}$$

$$m_t = r_t\ v_{t,r,m}, \tag{17}$$

where $q_{t,r,e} = U^{(r)}\ \hat{e}_t,\quad k_{t,r,m} = W^{(r)}\ m_{t-1}^{avg},\ v_{t,r,m} = P^{(r)}\ m_{t-1}^{avg}$ and "r" represents the recover gate.

After the input gate, the forget gate and the recover gate are generated and used, we add the information accepted into the memory at this step together:

$$c_t = \tilde{c}_t + \tilde{c}_{t-1} + m_t. \tag{18}$$

3.3.4 The output gate

The last gate is the output gate that assesses the informativeness of the cell output, *i.e.*, which part of the c_t is exported. Since this gate is closely related to both the current state and the global memory, it is determined by the product of three query-key pairs, including the reciprocal query-key pair of $\hat{e}_t$ and h_{t-1}, namely, $qk_{t,o,e} = U^{(o1)}\ \hat{e}_t$ and $qk_{t,o,h} = W^{(o1)}\ h_{t-1}$; the query-key pair of $\hat{e}_t$ and m_{t-1}^{avg}, namely, $q_{t,o,e} = U^{(o2)}\ \hat{e}_t$ and $k_{t,o,m} = W^{(o2)}\ m_{t-1}^{avg}$. Here, o represents the output gate; e, h and m represent the original embeddings of the pairs. Hence,

$$o_t = Sigmoid\left(\frac{(qk_{t,o,e}\ qk_{t,o,h})}{\sqrt{d}}\ \frac{(q_{t,o,e}\ k_{t,o,m})}{\sqrt{d}}\right), \tag{19}$$

$$h_t = o_t\ c_t, \tag{20}$$

where h_t is the output of an LSTeM cell, *i.e.*, the hidden state of time t.

3.3.5 Prediction layers

After the hidden state of the current step h_t is obtained, we pass it into a dual-layer feed-forward network to get the final user's interest representation at time t:

$$\hat{h}_t = W^{(p2)} \left(\sigma \left(W^{(p1)} h_t\right)\right), \tag{21}$$

where $W^{(p*)} \in \mathbb{R}^{d \times d}$ are the trainable parameters, σ is the Rectified Linear Unit (ReLU) activation function between two feed-forward layers, and $\hat{h}_t$ is the vector that indicates the user's current preference.

3.3.6 Layer normalization and dropout

To accelerate and stabilize the training process and help to avoid gradient errors [38], layer normalization statistic is used in the LSTeM cell to normalize the item embeddings and the hidden states into a zero-mean and unit-variance space. Specifically, given an input vector x, the operation is formulated as:

$$LayerNorm(x) = f\left(g \frac{x - \mu}{\sqrt{\sigma^2 + \varepsilon}} + b \right), \tag{22}$$

where μ and σ are the mean and variance of x, while g and b are scale and bias parameters to learn.

We also use dropout [39] in each feed-forward layer in LSTeM. Dropout regularization allows the model to simulate a number of different architectures by randomly turning off network nodes during the training process. It is used to reduce overfitting and to improve the generalization of deep learning models. In LSTeM, we apply dropout layers after the embedding layer and between the two prediction layers.

3.4 Recommendation and objective function

Recall that LSTeM consists of an embedding layer, a new LSTM structure, and a dual-layer prediction network. For each step time t, the current implicit user's interest embedding $p_t \in \mathbb{R}^d$ is extracted from the current interacted item and the historical memory of the sequence by these model components. To train the model, we adopt Matrix Factorization (MF) to rate the probability of a set of candidates to be the next item in the user's sequence. The candidate set comprises C items, including a positive one (*i.e.*, the real next item), and $(C - 1)$ randomly selected negative ones.

As introduced in Section 2, MF is a commonly used method for recommendation, in which the user-item interaction rates are predicted as their inner products. Therefore, the candidates are fed into the embedding layer and converted to the item embeddings $\{x_i\}_{i=1}^{C}$, where $x_i \in \mathbb{R}^d$, and the probability that the user interacts with x_i at next time step is expressed as

$$p_{t,i} = \hat{h}_t \, x_i. \tag{23}$$

We then use Binary Cross Entropy as the objective function to calculate the loss. Here, the real next item is deemed as the positive answer, while the random candidates are the negative ones. The loss is computed as:

$$L = -\sum_{u \in [1:M]} \sum_{t \in [1:n]} \left[log(p_{t,i}) + \sum_{j \notin S} log(1 - p_{t,j}) \right]. \tag{24}$$

Here, we get rid of the padding items using a mask to avoid the influence on the prediction accuracy.

After the training process, we test the model with the same strategy for candidates as in training. For a single sequence, We continuously pass all the items except the last one through the model, and the user's interest embedding at the second last time step p_{n-1} is retained to test with the candidates for the last step.

3.5 Discussion

In LSTeM, we design the new gates for LSTMs with the "Q-K-V" mechanism, which filters information based on the dependencies between the item and the memory. Therefore, at different time steps, the input information of an item is personalized, and the model can capture the relationships between the current item and the previous items more flexibly and more comprehensively than the original LSTMs. This strategy alleviates the first problem discussed in Section 1 that RNNs tend to make point-wise predictions and ignore the collective dependencies among the whole sequence.

Moreover, we propose a recover gate, which integrates the information of previous items into a global memory embedding by the simple Self-Attention method, to overcome the second problem that the forgetting mechanism of original LSTMs tends to discard the information not needed in the recent and current time step but important for afterward time steps.

4 Experiment

In this section, we show the experiments on LSTeM, including the experiment setting, the baselines, the datasets and the results. Also, we analyze the effectiveness of our method and have a discussion on the ablation studies. The experiments are designed to answer the following questions:

- How effective is LSTeM compared to other state-of-the-art baselines? (**Q1**)
- Do the gates with the new structures work better in learning user interest patterns than the original components of LSTMs? (**Q2**)
- Can the recover gate help the model recover the information lost during previous time steps? (**Q3**)
- How sensitive is the dimension of item embeddings? (**Q4**)

4.1 Datasets

We evaluate LSTeM on four datasets, which remarkably vary in domain, platform, and data sparsity:

- MovieLens: A popular benchmark dataset in the domain of recommendation, which collects users' actions on the movie website. In LSTeM, We employ two stable ver-

Table 1 Statistics of the datasets used to evaluate the proposed LSTeM model

Datasets	#users	#items	avg. actions/user	avg. actions/item	avg. length	#actions
Beauty	52,024	57,289	7.6	6.9	6.02	0.39M
Steam	334,730	13,047	11.0	282.5	9.3	3.7M
ML-1m	6,040	3,416	165.5	292.6	163.5	1.0M
ML-20m	138,493	26,744	144.4	747.8	142.4	20M

sions, MovieLens 1m (*ML*-1*m*)[1] and MovieLens 20m (*ML*-20*m*)[2]. *ML*-1*m* is the densest dataset in our evaluation.

– Amazon[3]: A series of product purchase and review datasets collected from *Amazon.com*. It is characterized by high sparsity and variability. Amazon consists of several subsets depending on the product categories, and in our experiments, we adopt “Beauty”.
– Steam[4]: A rich user-item interaction dataset gathered from Steam, which is a large cross-platform video game distribution system, created in 2016.

We process the datasets by following the common practice employed in [18, 25, 31, 33]. For all datasets, the user-item interactions, including numeric ratings and the presence of reviews, are considered as implicit feedback. If a user has a piece of implicit feedback for an item, that means this user interacts with it. We build the users' behavior sequences for all users in the order of feedback timestamps. Besides, the sequences whose length is less than five are discarded to ensure the quality of the datasets. We separate each sequence into three parts: the last item of each sequence belongs to the test sets; the second last one is for validation; the rest are used in training. The data statistics are shown in Table 1. It can be seen that *Beauty* is the most sparse dataset, whose average length of user behavior sequences is only 6.02, while *ML*-1*M* is the densest one that has 163.5 items in a sequence in average.

4.2 Evaluation metrics

We evaluate our model with three metrics, Hit Rate (HR), Normalized Discounted Cumulative Gain (NDCG), and Mean Reciprocal Rank (MRR). For the TOP-K metrics, Hit Rate@K and NDCG@K, we take K = 1, 5, 10. Since there is only one ground truth in each sequence, here, HR@K can be regarded as Recall@k and in proportion to Precision@k. We omit NDCG@1, which equals to HR@1. To be efficient, we follow [14, 22, 31, 40] to adopt the strategy that randomly samples 100 negative items that the user has

[1] https://grouplens.org/datasets/movielens/1m/
[2] https://grouplens.org/datasets/movielens/20m/
[3] http://jmcauley.ucsd.edu/data/amazon
[4] https://cseweb.ucsd.edu/ jmcauley/datasets.html#steam_data

not interacted with, together with the positive item, to be the test candidates. In addition, to ensure the test results are reliable and objective, we refer to [14, 33], and sample half of the negative items based on their popularity, *i.e.*, the probability of appearing in all actions.

4.3 Baselines

To evaluate the performance of LSTeM, we conduct the experiments on the baselines that can be separated into two categories. The first category contains the traditional methods on recommender systems, which only make the prediction based on users' feedback without considering their temporal order, or only depending on the adjacent items:

- POPRec: Global Popularity is a naive method that simply sorts the item based on their popularity in the dataset in descending order.
- BPR-MF [41]: Bayesian Personalized Ranking is a classic approach in recommendation. It is designed to learn to rank the items based on a pairwise personalized ranking loss derived from the maximum posterior estimator.
- FPMC [18]: Factorized Personalized Markov Chains is a hybrid model that merges matrix factorization and factorized Markov chains to build individual transition matrix for each user, in order to predict the next items in users' action sequences.
- NCF [22]: Neural network-based Collaborative Filtering leverages a multi-layer perception model to learn the prediction of users' interest in items. It is a classic method to use MLP in the traditional approach, matrix factorization.

By contrast, the methods in the second category are based on deep neural networks. At least several of the items previous to the current item and the temporal orders in a sequence are considered in prediction.

- GRU4Rec [9]: This is a typical work that applies recurrent neural networks (RNNs) on sequential commendation. It designs the session-parallel mini-batches to model the whole sessions into GRUs to predict the next items.
- GRU4Rec$^+$ [10]: This is an advanced variant of GRU4Rec, which proposes a new set of ranking loss functions and the sampling strategy that provides top-k gains to improve the performance.
- Caser [25]: Convolutional Sequence Embeddings is a model based on a CNN architecture. It adopts convolutional operations on the item embedding matrix in both horizontal and vertical directions to model high-order Markov chains for sequential recommendation.
- SASRec [31]: Self-Attentive Sequential Recommendation applies the left-to-right Self-Attention mechanism on recommendation, which works well in handling the relations among items with both long and short intervals.
- RCNN [15]: Recurrent Convolutional Neural Network leverages the recurrent architecture of RNNs to retrieve long-term dependencies of items and conducts the convolutional operation of CNNs to extract short-term relationships among recurrent hidden states. It is noteworthy that, for the sake of fairness, we change the output dimension of RCNN's last fully-connected layer to the same as the item embeddings, and modify the objective function to the same as LSTeM, so that we can use our evaluation metrics to test it.

4.4 Implementation details

We implement LSTeM with PyTorch[5] and initialize the parameters with normal distribution in the range $[-\sqrt{d}, \sqrt{d}]$, where d is the dimensionality of item embeddings. The model is optimized by the Adam optimizer [42], which is a combination of AdaGrad and RMSProp algorithms. Adam takes an adaptive moment estimation strategy to compute individual learning rates for model parameters. Hence, it performs better in dealing with the sparse gradients on noisy problems. The learning rate of Adam optimizer is set to $1e-3$,and $\beta_1 = 0.9$, $\beta_2 = 0.999$. We set the batch size to 128. Depending on the density of datasets, the dropout rates for ML-$1m$ and ML-$20m$ are set to 0.2, while for other datasets, the rates are set to 0.5. The fixed sequence lengths n are different based on the average length of users' action histories, so we set 200 for ML-$1m$ and ML-$20m$, 50 for others. The sensitivity of the dimension of item embeddings is examined in the experiments and we discuss this in 4.6.1. For the sake of fairness, we reproduce the codes of the baselines in PyTorch, referring to their papers and original codes (if published).

4.5 Performance analysis

To answer **Q1**, we compare LSTeM with 9 existing works, and the evaluation results are shown in Table 2. It can be observed that LSTeM relatively performs the best in all methods. On the whole, it is obvious that all the approaches do better on the denser datasets (*i.e.*, ML-$1m$ and ML-$20m$) than on the sparser ones (*i.e.*, *Beauty* and *Steam*). The methods based on deep neural networks (*i.e.*, GRU4Rec, GRU4Rec$^+$, Caser, RCNN, SASRec and LSTeM) perform better than the traditional approaches (*i.e.*, BPR-MF, FPMC and NCF) and the naive method (*i.e.*, POPRec). This is not only due to the inherent advantages of deep learning, but also because the deep-neural-network group considers the temporal orders of items and involves at least several previous items to predict the next item, while the traditional approaches do not take into account the positions of the items in the sequence, or even only considers the neighboring items in the sequences. This observation verifies that any previous item may have an effect on the current time step. Besides, POPRec gets the worst results because of its non-individuality, which demonstrates that users' preference is dynamic, complex and widely different.

As to LSTeM, compared to the state-of-the-art baselines, SASRec, RCNN, Caser and GRU4Rec$^+$, it gains relative improvements in both sparse datasets (*i.e.*, Beauty and Steam) and the dense datasets (*i.e.*, ML-$1m$ and ML-$20m$). Among them, the most significant improvement can be found in ML-$20m$, which demonstrates that LSTeM has the stronger ability to take the advantage of large data volume. In addition, as an RNN-based model, LSTeM shows a distinct enhancement in comparison to RCNN, GRU4Rec and GRU4Rec$^+$. This indicates that, to some extent, the improvement of the internal structure in RNNs achieves a satisfactory result and we show and analyze more testing results in this respect in Section 4.6. It is noteworthy that, The performance of LSTeM and SASRec are close, and especially they are level pegging on *Beauty*. However, as a model based on Self-Attention, SASRec is naturally disadvantaged in terms of perceiving and making use of the item position and order information in users' interaction sequences, which is quite important for capturing users' interests. By contrast, LSTeM, an RNN-based method, has the absolute

[5] https://pytorch.org/

Table 2 Performance of LSTeM and baselines on four public datasets

Dataset	Metric	POPRec	BPR-MF	FPMC	NCF	GRU4Rec	GRU4Rec$^+$	Caser	RCNN	SASRec	LSTeM
Beauty	MRR	0.0415	0.1009	0.1393	0.1043	0.1198	0.1293	0.1293	0.1470	**0.1476**	**0.1476**
	HR@1	0.0082	0.0415	0.0483	0.0495	0.0517	0.0590	0.0547	0.0706	0.0758	**0.0773**
	HR@5	0.0353	0.1196	0.1537	0.1278	0.1388	0.1631	0.1758	0.1889	**0.2064**	0.2052
	HR@10	0.0812	0.2007	0.2600	0.2232	0.2491	0.2554	0.2611	0.2693	0.2827	**0.2836**
	NDCG@5	0.0233	0.0870	0.1016	0.0956	0.1002	0.1024	0.1090	0.1260	**0.1391**	0.1384
	NDCG@10	0.0446	0.0902	0.1474	0.1089	0.1223	0.1377	0.1476	0.1579	0.1634	**0.1637**
Steam	MRR	0.0617	0.0903	0.1086	0.1109	0.1620	0.1834	0.1869	0.1899	0.1943	**0.1955**
	HR@1	0.0191	0.0303	0.0573	0.0346	0.0598	0.0716	0.0710	0.0823	0.1019	**0.1021**
	HR@5	0.0814	0.1245	0.1402	0.1098	0.2234	0.2447	0.2321	0.2614	0.2713	**0.2734**
	HR@10	0.1125	0.2039	0.2336	0.2217	0.3476	0.3753	0.3493	0.4197	0.4305	**0.4314**
	NDCG@5	0.0484	0.0787	0.1079	0.0923	0.1567	0.1740	0.1721	0.1966	0.2049	**0.2171**
	NDCG@10	0.0697	0.0989	0.1218	0.1226	0.1772	0.2157	0.1785	0.2438	0.2513	**0.2578**
ML-1m	MRR	0.0628	0.1813	0.2512	0.1602	0.3198	0.3481	0.3519	0.3530	0.3654	**0.3665**
	HR@1	0.0131	0.0879	0.1094	0.0459	0.1673	0.2097	0.2036	0.2086	0.2110	**0.2149**
	HR@5	0.0723	0.2724	0.3969	0.2133	0.4540	0.5223	0.5267	0.5327	0.5453	**0.5502**
	HR@10	0.1333	0.4122	0.5677	0.3617	0.6104	0.6412	0.6801	0.6852	0.6978	**0.7001**
	NDCG@5	0.0393	0.1795	0.2588	0.1405	0.2929	0.3517	0.3644	0.3718	0.3831	**0.3865**
	NDCG@10	0.0608	0.2210	0.3606	0.1712	0.3518	0.3850	0.4145	0.4214	0.4293	**0.4326**
ML-20m	MRR	0.0699	0.1323	0.1974	0.1136	0.2897	0.3163	0.3211	0.3223	0.3375	**0.3388**
	HR@1	0.0217	0.0544	0.0946	0.0782	0.1667	0.1818	0.1753	0.1877	0.1976	**0.2026**
	HR@5	0.0809	0.1907	0.3435	0.1594	0.4701	0.5083	0.4991	0.5165	0.5304	**0.5315**
	HR@10	0.1401	0.3196	0.5369	0.3098	0.5940	0.6558	0.6625	0.6992	0.7095	**0.7145**
	NDCG@5	0.0518	0.1297	0.2003	0.1206	0.3066	0.3241	0.3379	0.3364	0.3511	**0.3559**
	NDCG@10	0.0707	0.1582	0.2597	0.1125	0.3092	0.3677	0.3702	0.4078	0.4234	**0.4284**

Note: The best results are bolded and the second best results are underline

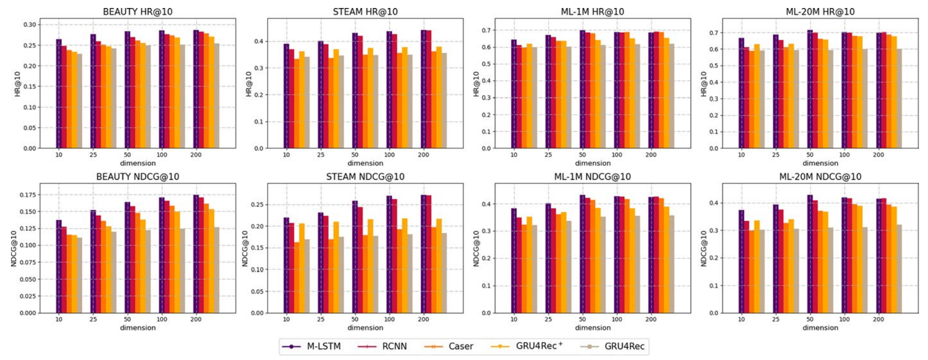

Figure 2 Sensitivity of item embedding dimensionality on HR@10 and NDCG@10 for the deep neural network-based models

advantage of processing item positions and orders, and there is plenty of room for development on the gates and memory mechanism. Hence, it can be expected that future versions of LSTeM have the potential to achieve even better performance.

4.6 Ablation study

To answer Question 2, 3, 4 listed at the beginning of this section, we conduct a series of experiments to comprehensively examine the effectiveness of LSTeM in sequential recommendation, including three ablation studies: the sensitivity of the item embedding dimensionality, the efficacy of the recover gate, and the validity of the new gates with the "Q-K-V" mechanism.

4.6.1 Sensitivity of embedding dimensionality

The sensitivity of item embedding dimensionality (**Q4**) in five deep neural network-based methods is shown in Figure 2. We evaluate HR@10 and NDCG@10 of these methods with the item embedding dimension 10, 25, 50, 100 and 200, respectively. Overall, it can be seen that, for the sparse datasets, the larger the embedding dimension is, the better performance can be achieved, and this variate has a relatively heavier effect on *Beauty* than *Steam*. For the dense datasets, the models based on CNNs, such as RCNN and Caser, are very sensitive to the embedding dimension, because the larger embeddings allow the convolutional filters to receive more information. Moreover, the performance of LSTeM steadily increases with the embedding dimension getting larger from 10 to 50, but drops down between 50 and 100, and then gradually levels off. This phenomenon demonstrates

Table 3 Ablation studies run on the four datasets, evaluated using NDCG@10

	Beauty	Steam	ML-1m	ML-20m
LSTeM	**0.1637**	**0.2578**	**0.4326**	**0.4284**
no r-gate	0.1467	0.2371	0.4154	0.4133
v-gates + sa	0.1483	0.2336	0.4139	0.4165
v-gates + mean	0.1328	0.2178	0.3892	0.3981
LSTM	0.1062	0.2025	0.3787	0.3826

Note: Bold font indicates the best performances

Table 4 Time cost for one epoch for models run against the four datasets (seconds)

	Beauty	Steam	ML-1m	ML-20m
LSTeM	51.65	273.05	18.07	414.72
no r-gate	35.98	224.41	12.83	278.29
v-gates + sa	46.44	257.13	16.39	353.21
v-gates + mean	37.51	231.99	13.60	279.40
LSTM	35.10	193.72	11.41	237.39

that LSTeM tends to converge as the dimensionality rises to 50. Therefore, in practice, for learning efficiency, there is no need to take a very large dimensionality.

4.6.2 Effectiveness of the recover gate and the new gate structures

To explore the effect of the recover gate (**Q3**) and the new gate structures (**Q2**) in LSTeM, we introduce three variants of LSTeM, and evaluate them and the traditional LSTM with NDCG@10 in the four datasets. The results are shown in Table 3. The first variant (*No r-gate*) removes the recover gate along with the global memory vector, namely, the variant model only retains three gates, input, forget and output. Here, for the output gate, we use

$$o_t = Sigmoid\left(\frac{qk_{toe}\, qk_{toh}}{\sqrt{d}}\right). \tag{25}$$

The second variant (*v-gates* + *sa*) employs the vanilla addition operations within all gates as the same as traditional LSTMs, but also has the recover gate with the global memory, and the recover gate is formulated as

$$r_t = Sigmoid(U_{(r)}\hat{e}_t^{(u)} + W_{(r)}m_{t-1}^{avg}) \tag{26}$$

$$m_t = r_t\, m_{t-1}^{avg} \tag{27}$$

The last variant (*v-gates* + *mean*) is the same as the second one apart from the global memory. Here, we generate the global memory vector simply using the element-wise mean of all $(t-1)$ item embeddings, instead of Self-Attention.

From Table 3, it can be seen that the complete LSTeM obviously performs the best in all models, and all variants are better than the traditional LSTM. The variant without r-gate has an obvious decrease in performance, namely, the recover gate has a significant effect on the model's memory, which verifies our motivation to enhance the memory using the recover gate. Besides, both of the variants with vanilla gates perform worse than the complete model, which validates that the new gate structures with the "Q-K-V" mechanism contribute a distinct improvement in interpreting the dependencies between items and memories. The substantial difference between $v-gates+sa$ and $v-gates+mean$ shows that the utilization of Self-Attention in the global memory generation makes a big contribution toward capturing the latent relationships among items.

In addition, the time costs of LSTeM, LSTM and the three variants are also evaluated. All experiments are conducted with the same GeForce RTX 2080 Graphics Card. Table 4 shows the average training time costs for one epoch of the five models in the four datasets.

It can be seen that the time cost of LSTeM is acceptable, but there is still room for improvement. It is worth noting that the difference between LSTeM and the variant with vanilla gates (v-gates + sa) and the difference between the variants with Self-Attentional global memory (v-gates + sa) and mean-global memory (v-gates + mean) are both small, which means that the time cost of the "Q-K-V" mechanism and Self-Attention are relatively low.

5 Conclusion

In this work, we propose a novel Long Short-Term enhanced Memory model for sequential recommendation. It is the very first work to boost LSTMs' internal memory by proposing the new structures for gates. We apply the "Q-K-V" mechanism in the gates to capture the latent dependency between the current item and the historical memory, and also develop a recover gate to remedy the inadequacy of memory caused by the forgetting mechanism. The recover gate works with a dynamic global memory embedding generated by using a Self-Attention model. Extensive experiments have demonstrated that LSTeM achieves comparable performance to the state-of-the-art methods on the challenging datasets. Results of the ablation studies further verify the effectiveness of the new gate structures and the recover gate.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Wang, S., Hu, L., Wang, Y., Cao, L., Sheng, Q.Z., Orgun, M.A.: Sequential recommender systems: Challenges, progress and prospects. In: IJCAI, pp. 6332–6338. ijcai.org (2019)
2. Lu, J., Wu, D., Mao, M., Wang, W., Zhang, G.: Recommender system application developments: A survey. Decis. Support Syst. **74**, 12–32 (2015)
3. Zhang, S., Yao, L., Sun, A., Tay, Y.: Deep learning based recommender system: A survey and new perspectives. ACM Comput. Surv. **52**(1), 5–1538 (2019)
4. Konstan, J.A., Riedl, J.: Recommender systems: from algorithms to user experience. User Model. User Adapt. Interact. **22**(1–2), 101–123 (2012)
5. Isinkaye, F.O., Folajimi, Y.O., Ojokoh, B.A.: Recommendation systems: Principles, methods and evaluation. Egyptian Informatics Journal **16**(3), 261–273 (2015)

6. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
7. Sundermeyer, M., Schlüter, R., Ney, H.: LSTM neural networks for language modeling. In: INTERSPEECH, pp. 194–197. ISCA (2012)
8. Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: EMNLP, pp. 1724–1734. ACL, (2014)
9. Hidasi, B., Karatzoglou, A., Baltrunas, L., Tikk, D.: Session-based recommendations with recurrent neural networks. In: ICLR (2016)
10. Hidasi, B., Karatzoglou, A.: Recurrent neural networks with top-k gains for session-based recommendations. In: CIKM, pp. 843–852. ACM (2018)
11. Liu, Q., Wu, S., Wang, L.: Multi-behavioral sequential prediction with recurrent log-bilinear model. IEEE Trans. Knowl. Data Eng. **29**(6), 1254–1267 (2017)
12. Li, J., Ren, P., Chen, Z., Ren, Z., Lian, T., Ma, J.: Neural attentive session-based recommendation. In: CIKM, pp. 1419–1428. ACM (2017)
13. Hidasi, B., Quadrana, M., Karatzoglou, A., Tikk, D.: Parallel recurrent neural network architectures for feature-rich session-based recommendations. In: RecSys, pp. 241–248. ACM (2016)
14. Huang, J., Zhao, W.X., Dou, H., Wen, J., Chang, E.Y.: Improving sequential recommendation with knowledge-enhanced memory networks. In: SIGIR, pp. 505–514. ACM (2018)
15. Xu, C., Zhao, P., Liu, Y., Xu, J., Sheng, V.S., Cui, Z., Zhou, X., Xiong, H.: Recurrent convolutional neural network for sequential recommendation. In: WWW, pp. 3398–3404. ACM (2019)
16. Miller, A.H., Fisch, A., Dodge, J., Karimi, A., Bordes, A., Weston, J.: Key-value memory networks for directly reading documents. In: EMNLP, pp. 1400–1409. The Association for Computational Linguistics (2016)
17. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: NIPS, pp. 5998–6008 (2017)
18. Rendle, S., Freudenthaler, C., Schmidt-Thieme, L.: Factorizing personalized markov chains for next-basket recommendation. In: WWW, pp. 811–820. ACM (2010)
19. He, R., McAuley, J.J.: Fusing similarity models with markov chains for sparse sequential recommendation. In: ICDM, pp. 191–200. IEEE Computer Society (2016)
20. Koren, Y., Bell, R.M., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)
21. Dziugaite, G.K., Roy, D.M.: Neural network matrix factorization. CoRR **abs/1511.06443** (2015)
22. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.: Neural collaborative filtering. In: WWW, pp. 173–182. ACM (2017)
23. Wang, H., Wang, N., Yeung, D.: Collaborative deep learning for recommender systems. In: KDD, pp. 1235–1244. ACM (2015)
24. Wu, C., Ahmed, A., Beutel, A., Smola, A.J., Jing, H.: Recurrent recommender networks. In: WSDM, pp. 495–503. ACM (2017)
25. Tang, J., Wang, K.: Personalized top-n sequential recommendation via convolutional sequence embedding. In: WSDM, pp. 565–573. ACM (2018)
26. Yuan, F., Karatzoglou, A., Arapakis, I., Jose, J.M., He, X.: A simple convolutional generative network for next item recommendation. In: WSDM, pp. 582–590. ACM (2019)
27. Quadrana, M., Karatzoglou, A., Hidasi, B., Cremonesi, P.: Personalizing session-based recommendations with hierarchical recurrent neural networks. In: RecSys, pp. 130–137. ACM (2017)
28. Donkers, T., Loepp, B., Ziegler, J.: Sequential user-based recurrent neural network recommendations. In: RecSys, pp. 152–160. ACM (2017)
29. Yu, F., Liu, Q., Wu, S., Wang, L., Tan, T.: A dynamic recurrent model for next basket recommendation. In: SIGIR, pp. 729–732. ACM (2016)
30. Ying, H., Zhuang, F., Zhang, F., Liu, Y., Xu, G., Xie, X., Xiong, H., Wu, J.: Sequential recommender system based on hierarchical attention networks. In: IJCAI, pp. 3926–3932. ijcai.org (2018)
31. Kang, W., McAuley, J.J.: Self-attentive sequential recommendation. In: ICDM, pp. 197–206. IEEE Computer Society (2018)
32. Cui, Q., Wu, S., Liu, Q., Zhong, W., Wang, L.: MV-RNN: A multi-view recurrent neural network for sequential recommendation. IEEE Trans. Knowl. Data Eng. **32**(2), 317–331 (2020)
33. Sun, F., Liu, J., Wu, J., Pei, C., Lin, X., Ou, W., Jiang, P.: Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In: CIKM, pp. 1441–1450. ACM (2019)
34. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: NAACL-HLT (1), pp. 4171–4186. Association for Computational Linguistics (2019)

35. Liu, C., Li, X., Cai, G., Dong, Z., Zhu, H., Shang, L.: Noninvasive self-attention for side information fusion in sequential recommendation. In: AAAI, pp. 4249–4256. AAAI Press (2021)
36. Wang, H., Liu, G., Liu, A., Li, Z., Zheng, K.: DMRAN: A hierarchical fine-grained attention-based network for recommendation. In: IJCAI, pp. 3698–3704. ijcai.org (2019)
37. Wu, J., Cai, R., Wang, H.: Déjà vu: A contextualized temporal attention mechanism for sequential recommendation. In: WWW, pp. 2199–2209. ACM / IW3C2 (2020)
38. Ba, L.J., Kiros, J.R., Hinton, G.E.: Layer normalization. CoRR **abs/1607.06450** (2016)
39. Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
40. Koren, Y.: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: KDD, pp. 426–434. ACM (2008)
41. Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L.: BPR: bayesian personalized ranking from implicit feedback. In: UAI, pp. 452–461. AUAI Press (2009)
42. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (Poster) (2015)

Authors and Affiliations

Jiasheng Duan[1] · Peng-Fei Zhang[1] · Ruihong Qiu[1] · Zi Huang[1]

Peng-Fei Zhang
mima.zpf@gmail.com

Ruihong Qiu
r.qiu@uqconnect.edu.au

Zi Huang
huang@itee.uq.edu.au

[1] School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, Australia