Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 5457044, 12 pages
https://doi.org/10.1155/2022/5457044

Research Article

A Hierarchical Network with User Memory Matrix for Long Sequence Recommendation

Jiawei Dong, Fuzhen Sun, Tianhui Wu, Xiangshuai Wu, Wenlong Zhang, and Shaoqing Wang

School of Computer Science and Technology, Shandong University of Technology, Shandong, Zibo 255000, China

Correspondence should be addressed to Fuzhen Sun; sunfuzhen@sdut.edu.cn

Received 26 September 2021; Accepted 30 December 2021; Published 31 January 2022

Academic Editor: Yingjie Wang

In many recommendation scenarios, the interactions between users and items are divided into a series of sessions according to the time interval. The traditional Recurrent Neural Network has some shortcomings, such as limited memory ability, inflexible access to memory data, and obvious deficiency in feature capture for long sequences. To deal with the mentioned issues, we propose a hierarchical network with user memory matrix, named HNUM^2, which utilizes the memory network to store users' long-term and short-term interests. The memory network is more flexible to access memory data, which can solve the problem of insufficient capture of long sequence features. The proposed model is a hierarchical recommendation algorithm, which consists of two layers. The first layer is the session-level GRU model, which obtains the sequence characteristics of the current session to predict the next item. The second layer is the user-level memory network model which exploits the attention mechanism and incorporates the write module and read module. The experimental results on two public available datasets show that HNUM^2 has achieved significant performance improvement comparing to the state-of-the-art methods.

1. Introduction

With the development of the big data era, recommender systems are still an effective means to solve information overload [1]. Sequential Recommender Systems (SRSs) have received more and more attention in recent years. Through the interaction between users and items, SRSs understand and generate user behavior sequences while capturing changes in users' interests [2]. Session-based recommender systems (SBRs) are a branch of sequential recommender systems, which received considerable attention from industry and academia [3]. Deep learning technology has set off an upsurge in academia and industry. More and more scholars have applied deep learning technology to recommender systems [4]. Deep learning models have powerful learning ability and can avoid the problem of traditional recommendation models, such as the manual design model features [5]. Yap et al. [6] introduced a recommendation framework based on personalized sequential pattern mining, which used a new score metric to effectively learn user-specific sequences important for accurate personalized recommendations. In recent years, similarity-based methods have been applied to session-based recommendations, with good results on sparse datasets. Hidasi et al. [7] first applied Recurrent Neural Network to recommender systems, which designed a parallel session recommendation model GRU4REC. Experimental results showed that Recurrent Neural Network has a good performance in session-based recommendation algorithms. Quadrana et al. [8] proposed a hierarchical recommendation model. The model designed two levels of RNN: the user-level RNN model and the session-level RNN model.

RNN has a relatively good performance in the sequential tasks. It can store limited information and more content as the memory units. However, it loses more information [9]. In 2014, Weston et al. [10] introduced a new learning model, memory network. In the same year, the DeepMind team of Google proposed neural turing machines [11]. Both of them use external memory for memorization. The neural turing machine was designed with attention-based read and write

operations that allow for more flexible reading of memories. In 2015, Sukhbaatar et al. proposed end-to-end memory network [12], where external storage space of the network is a memory matrix, which is introduced to better capture long sequence features.

Memory network was initially used in Q&A systems. Recently, memory network has been widely used in the recommender systems, which has attracted people's attention. Chen et al. [13] stored and updated user's history by using external storage matrix in memory network and enhanced the expressiveness of the model. Huang et al. [14] obtained two benefits from the hybrid module by using a mixture of RNN and key-value memory networks (KV-MNS). A combination of sequential preference representation and attribute-level preference representation is used as the final representation of user preferences. Due to the addition of knowledge-based information, the model is highly interpretable [15]. To take full advantage of textual information and visual information, Ma et al. [16] proposed new cross-attention memory network to perform multimodal tweet reference recommendation, which combined users' interests with external memory and uses cross-attention mechanism to extract textual information and visual information [17].

Based on the above problems, the contributions of this paper are essentially threefold:

(1) According to previous work, sessions are assumed to be independent of each other, and historical session information is ignored. To solve the above problem, we propose a hierarchical network with user memory matrix (HNUM^2), which considers the interaction between sessions and historical session information to read the user's historical sessions and provides initial input to the GRU unit within the session.

(2) We proposed a hierarchical recommendation model. The first layer is a session-level GRU model for predicting the next item. The second layer is the user-level memory network model, which refers to the changes in users' long-term interests. The model consists of two modules: the read module and the write module.

(3) The experimental results show that the proposed model has better performance improvement than the current algorithm when the number of user sessions is large.

The rest of the paper is organized as follows: In Section 2, we briefly review the existing research on session-based recommender systems and Recurrent Neural Network. In Section 3, we first present the whole structure of the model, then introduce the two levels of the model and the loss function, and finally give the algorithm flow of the model. Section 4 describes and analyzes these assessments, and a large number of experiments on two real datasets of different volumes demonstrate the recommender performance of the proposed model compared to other models. In Section 5, we summarize our work and propose several future research directions.

2. The Related Work

We first review current models of session-based recommender systems and then introduce Recurrent Neural Network (RNN) and GRU. Finally, we review the latest research on memory network.

2.1. Session-Based Recommendation Algorithm. Session sequences refer to a set of item sequences used by a user in an interactive transaction or collected over a period of time [18]. Traditional recommendation algorithms only model user's long-term preferences and static preferences and ignore short-term and dynamic transaction patterns of users, which can lead to missing the transfer of user preferences over time. In this case, a user's intention at a past time can easily be replaced by a new user's historical behavior, resulting in poor and unreliable recommendations. In order to solve the above problems, it is necessary to consider the affair structure to capture richer information in the recommendation [19]. Therefore, session-based recommender systems are proposed.

Session-based recommendation problem can be expressed as sequence prediction problems; we define a session $\{x_1, x_2, \ldots, x_{s-1}, x_s\}\{x_1, x_2, \ldots, x_{s-1}, x_s\}$, where $x_i\,(1 \le i \le S)$ denotes the index of the user's interactive items in the total number of N items. Define the output as the sort list $y = [y_1, y_2, \ldots, y_n] \in R^N$ of all possible items in the session, where y_i corresponds to the score of item i. The usual practice is to sort according to the size of y_i, taking the top-K items.

Aghdam et al. [20] introduced a hierarchical hidden Markov model to capture changes in user preferences, used the feedback sequence of users to items, modeled users as a hierarchical hidden Markov process, and used users' current content attributes as hidden variables in this model. Gu et al. [21] proposed using Markov chains to track user purchase behavior chains, using purchase intervals to improve the temporal diversity of e-commerce recommendations. The algorithm has a significant improvement in accuracy, conversion rate, and time diversity. He et al. [22] proposed Mixture Variable Memory Markov (MVMM) model, which is a new method of sequential query prediction. This method attempts to capture the user's search intent based on the user's past query sequences. Markov model only considers relatively short historical information, and its representation ability is minimal [23].

2.2. Recurrent Neural Network. Recurrent Neural Network (RNN) is a kind of neural network specifically designed for sequential data. By receiving its own information, RNN achieves a certain "memory function" and retains a certain amount of memory for the processed information [24]. Given an input sequence $\{x_1, x_2, \ldots, x_t, \ldots, x_T\}$ of length T, x_t represents the input vector of the sequence data at the moment t. The index t is not necessarily the elapsed time in the real world, and sometimes it only represents the position in the sequence data. The active value h_t of the hidden layer with feedback edge is updated by the following formula:

$$h_1 = f(h_{t-1}, x_t), \tag{1}$$

where $h_0 = 0$. $f(\cdot)$ is a nonlinear function. Figure 1 shows an example of Recurrent Neural Network.

Assuming that the input to the RNN at moment t is x_t, the hidden layer state h_t is not only related to the input x_t at the current moment, but also related to the hidden layer state h_{t-1} at the previous time.

$$z_t = \mathbf{U}h_{t-1} + \mathbf{W}x_t + b, \tag{2}$$

$$h_t = f(z_t). \tag{3}$$

where Z_t is the net input of the hidden layer, $f(\cdot)$ is the nonlinear activation function, usually logistic function or Tanh function, $\mathbf{U}$ is the state-state weight matrix, $\mathbf{W}$ is the state-input weight matrix, and b is the bias term. Figure 2 shows the Recurrent Neural Network expanded by time.

h_{t-1} is a memory feature, which extracts the input features of the previous $t-1$ moments. Sometimes h_{t-1} is called the old state, and h_t is the new state. Therefore, the RNN model is particularly suitable for sequence problems. Structurally, the RNN can be regarded as a neural network model with loops, and it can be expanded into a standard neural network model, but this neural network is not separated. In this way, RNN performs the same calculation process each time, but the inputs are different each time, which seriously restricts the ability of RNN to capture features of long sequences.

2.3. Gated Recurrent Unit. Gated Recurrent Unit (GRU) is a kind of RNN with gated control units. Because the structure of the GRU unit is simpler and easier to train, the efficiency of training can be improved by using GRU. The GRU unit not only saves computing costs but also does not cause performance degradations. At present, there is no relevant research to point out that the performance of the GRU unit is worse than other recurrent networks. The input and output structure of GRU are the same as those of RNN.

The GRU combines the forget gate and the input gate into one: the update unit. In addition, GRU does not require additional memory units and introduces a linear dependency directly between the current state h_t and the historical state h_{t-1}. In the GRU network, the current candidate state is $\widetilde{h}_t$.

$$\widetilde{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h), \tag{4}$$

where $r_t \in [0,1]$ is a reset gate, which is used to control whether the computation of the candidate state $\widetilde{h}_t$ depends on the state h_{t-1} of the previous time.

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r). \tag{5}$$

When $r_t = 0$, the candidate state $\widetilde{h}_t = \tanh(W_c x_t + b)$ is related to the current input x_t, but not related to the history state. When $r_t = 1$, the candidate state $\widetilde{h}_t = \tanh(W_h x_t + U_h h_{t-1} + b_h)$ is related to the current input x_t and the historical state h_{t-1}, which is consistent with the simple recurrent network. The hidden state h_t of the GRU network is updated in the following way:

$$h_t = z_t \cdot h_{t-1} + (1 - z_t) \cdot \widetilde{h}_t. \tag{6}$$

$z \in [0,1]$ is the update gate, which controls how much information is retained by the current state from the historical state and how much new information it receives from the candidate state.

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z). \tag{7}$$

When $z_t = 0$, there is a nonlinear function between the current state h_t and the historical state h_{t-1}. If both $z_t = 0$ and $r = 1$ exist, the GRU network degenerates to the simple recurrent network. If both $z_t = 0$ and $r = 0$ exist, the current state h_t is only related to the current input x_t and not to the history state h_{t-1}. When $z_t = 1$, the current state h_t is equal to the previous state h_{t-1} and is independent of the current input x_t.

2.4. Memory Network. Generally, memory network can be regarded as composed of five components. The first component is a memory module $m = \{m_1, m_2, m_3, \ldots, m_n\}$ used to store memories, and this module is usually implemented with m_i as the matrix of indexes. The other four-module components are Input module, Generalization module, Output module, and Response module. These four modules are usually referred to simply as I, G, O, and R.

Memory network is a general machine learning framework so that memory network can target different problems. Due to the use of long-term memory components for learning performs better than RNN in long-term memory, so it is called memory network.

The Input module, Generalization module, Output module, and Response module can use any existing algorithm in the field of machine learning, such as SVM and random forest [25]. The working process of each of the four modules is introduced, respectively.

Module I: The function of module I is to do a simple preprocessing of the external input. Usually, the external input is transformed into a vector that is easier to handle in machine learning. For example, word2vec technology converts words into dense vectors.

Module G: The implementation of module G is very flexible. For example, the easiest way is to add the output of module I directly into the memory space. Literature [26] uses a first-in-first-out method to add new memories into the memory space when memory network is applied in the recommender systems.

$$m_{H(x)} = I(x), \tag{8}$$

where $H(\cdot)$ is the function of the selected slot, and $I(x)$ is the output of module I.

Module O: The most important task of module O is responsible for reading memory and generating outputs. Both module O and module G can be implemented in the simplest way, such as reading the memory in order.

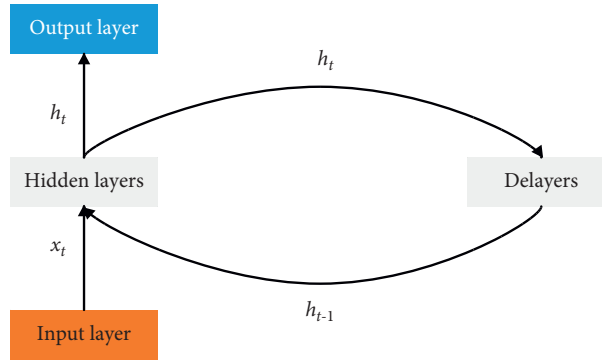

Figure 1: Simplified Recurrent Neural Network (RNN) structure diagram. The input of the Recurrent Neural Network subject consists of the x_t of the input layer and the hidden state of the previous moment h_{t-1}.

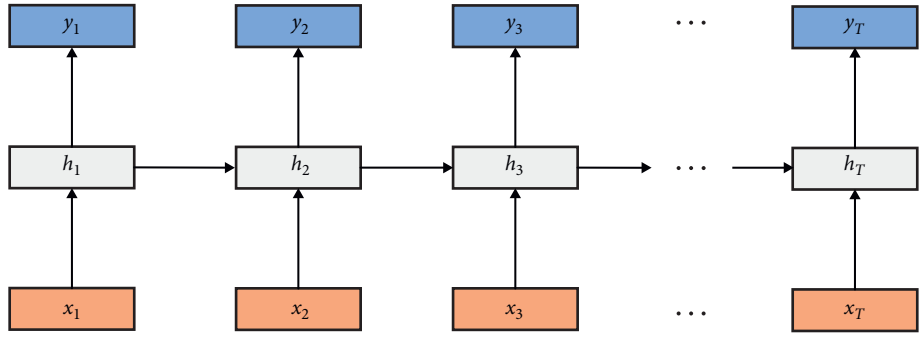

Figure 2: Recurrent Neural Network expanded by time. The RNN uses the results generated by the previous time step of the hidden layer, as part of the current time step, and influences the output of the current time step.

Module R: Module R converts the output of module O to the externally requested format.

2.5. Neural Turing Machine. Memory network is a branch of deep learning. The Facebook team's paper published in 2014 proposed memory network and introduced its application in Q&A systems [10]. In 2014, the Google DeepMind team used a similar idea to propose Neural Turing Machines (NTM) [11].

The NTM proposed by the DeepMind team refers to the idea of LSTM and generates an erase vector e_t and an add vector a_t for memory network to control the update of memory matrix.

The core of the model is module O and module R. Assuming that the input question in the Q&A systems is x, the task of module O is to select the TOP-N related memory from all the memories according to the input question vector. The specific selection method is first to select the most relevant memory.

$$o_1 = O_1(x, m) = \arg\max_{i=1,\ldots,N} s_O(x, m_i). \tag{9}$$

Next, select the memory o_2 that is most relevant to both of them based on the selected o_1 and input x together.

$$o_2 = O_2(x, m) = \arg\max_{i=1,\ldots,N} s_O\left(\left[x, m_{o_1}\right], m_i\right). \tag{10}$$

For equation (10) above, if linear vectors represent both x and o_1, they can be divided into the following way of addition:

$$s_O(x, m_i) + s_O\left(m_{o_1}, m_i\right). \tag{11}$$

Finally, module R needs to generate a text response r. The simplest response is to return m_{ok}, which is the output of the previously uttered sentences retrieved, and use the scoring function to calculate the relevance of all the candidate words to the input of module R, with the final word with the highest score being the correct answer.

$$r = \text{argmax}_{w \in W} s_R\left(\left[x, m_{o_1}, m_{o_2}\right], w\right), \tag{12}$$

where W is the set of all words in the dictionary and $s_R(\cdot)$ is the function that scores the matches.

3. The Proposed Model

3.1. Problem Formulation. Firstly, we introduce the overall structure of the model and then describe each module, respectively.

The hierarchical network with user memory matrix (HNUM^2) is a hierarchical network. The overall structure of the model is shown in Figure 3. The model consists of two layers. The first layer is a session-level GRU model, which is used to describe the sequence characteristics of the current session and store the user's short-term interests to predict

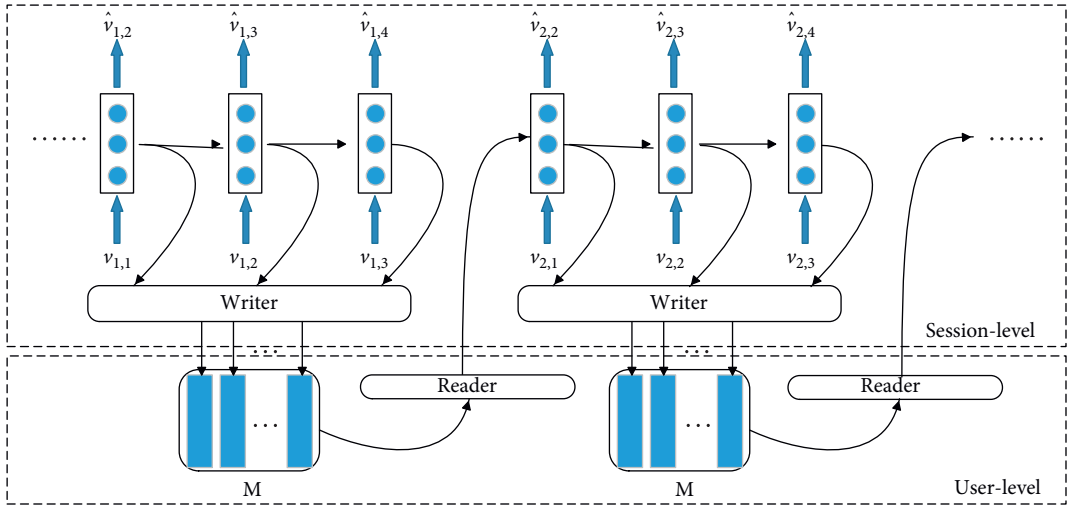

FIGURE 3: The overall structure of the model HNUM^2, which consists of two layers. The first layer is a session-level GRU model, and the second layer is a user-level memory network model.

the next item. The second layer is a user-level memory network model, which stores the entire user's historical information and describes the user's long-term interests. At the beginning of a user's session, the read module reads the memory vector in the memory matrix $\mathbf{M}$ corresponding to the current user and reads the memory as the user's preference vector to initialize the hidden layer of the GRU unit. At the same time, the user's hidden state, short-term interest, and the current stage of the click product are input to the session-level GRU unit. At the end of each time step, the output predicts the item clicks by this user in the next phase and the hidden state of the GRU in the next phase, which is stored into the memory matrix $\mathbf{M}$ by the write module. The same process is performed again when a user's session ends and the next session begins.

In Table 1, we introduce some of the notations used in this paper.

3.2. The Formal Description of HNUM2. Define $U = \{u_1, u_2, \ldots, u_N\}$ as the set of all users, $V = \{v_1, v_2, \ldots\}$ is a set of items, and $S^u = \{s_1^u, s_2^u, \ldots\}$ is the set of the session of user u. $V^s = \{v_1^s, v_2^s, \ldots\}$ is a sequence of interactive items generated in a user's sessions s, in which v_i is one of the interactive items in the whole model, and our goal is to predict the user's next interactive item $\hat{v}_{i+1}^s$. $\mathbf{M}^u = \{m_1^u, m_2^u, \ldots, m_K^u\} \in R^{D\times K}$ is the memory matrix of user u, and $m_k^u \in R^D$ is the k th memory vector of $\mathbf{M}^u$, which is used to store the long-term interests of the user. The size of $\mathbf{M}^u$ depends on the number of memory vectors K and the length D of the vectors in the memory matrix. Among them, K and D are the hyperparameters of the model.

3.3. Memory Reading Module. The read module is mainly responsible for reading the long-term interests of the user in the memory matrix, which is used to guide the training of the session phase. Specifically, set p^u to be the preference embedding of user u, and the interaction item v_i of the current session is used as input; p^u is obtained by reading the memory from $\mathbf{M}^u$. p^u can be expressed as

$$p^u = \text{READ}\left(\mathbf{M}^u, v_i\right). \tag{13}$$

v_i is the embedding vector of the i th interaction item in the current session. Intuitively, the previous i memory vectors will have different effects on the current interest, so the attention mechanism is introduced to assign weight values to different memory vectors.

The specific process of READ$(\cdot)$ operation is shown in equations (13)–(15).

$$w_{i,k} = v_i \cdot m_k^u, \tag{14}$$

$$z_{ik} = \frac{\exp\left(\beta w_{ik}\right)}{\sum_j \exp\left(\beta w_{ij}\right)}, \tag{15}$$

where β is an intensity parameter, which can enlarge or reduce the degree of focus. When $\beta = 1$ is a standard softmax, z_{ik} is used as the attention weight to derive the preference vector p^u for user u.

Therefore, the user's historical behavior can be accessed according to the impact of the user's historical behavior on the current item.

$$p^u = \sum_{k=1}^{K} z_{ik} \cdot m_k^u. \tag{16}$$

3.4. Memory Writing Module. The write module is responsible for updating the GRU hidden state into the memory matrix after a time step. Neural turing machine refers to the idea of the update gate of the LSTM:

(1) The input gate is used to determine the information to be added.

(2) The forget gate is used to determine the information to be discarded.

Table 1: Notations.

Symbol	Size	Description
U	$1 \times N$	The set of all users
S^u	$1 \times N$	The set of sessions for user u
V^s	$1 \times N$	The sequence of items that generate interactions in a session s
K		The number of memory vectors of memory matrix
D		The length of memory vector in matrix
$\mathbf{M}^u$	$R^{D\times K}$	The memory matrix of user u
m_k^u	R^D	The k-th memory vector of user u
p^u	K	Preference vector of user u
$\hat{r}_{s,i}$	R	The score of positive samples
$\hat{r}_{s,j}$	R	The score of negative samples

(3) The update gate is used to add or delete the information.

Specifically, the neural turing machine generates an erase vector and an add vector, in which the values of each element range from 0 to 1, indicating the information to be added or removed.

Since the whole process is matrix read and write operations are differentiable, the whole model parameters can be trained by gradient descent. For the erase vector erase_i:

$$\text{erase}_i = \sigma\left(E^T h_i + b_e\right). \tag{17}$$

$\sigma(\cdot)$ is the sigmoid function, E and b are the erase parameters, and h_i is the current hidden state of the user.

Update feature preference memory by attention weight and erase vector.

$$m_k^u \leftarrow m_k^u \cdot \left(1 - z_{ik} \cdot \text{erase}_i\right). \tag{18}$$

z_{ik} is the attention weight of the write phase.

After erasing, update the feature preference memory using the add vector add_i:

$$\text{add}_i = \tanh\left(A^T h_i + b_a\right), \tag{19}$$

$$m_k^u \leftarrow m_k^u + z_{ik}.\text{add}_i, \tag{20}$$

where A and b_a are the parameters in the add operation.

This erase-add updates strategy allows forgetting and reinforcing the learning process for the user preference embedding vector. The model can automatically learn to erase parameters and add parameters to determine which signals need to be weakened or enhanced.

3.5. Loss Function. Classical Bayesian Personalized Ranking (BPR) is a matrix factorization method using pairwise ranking loss [27]. BPR compares the scores of positive samples and negative samples [28]. In the iterative loss calculation process, the scores of the positive items are compared with the scores of the next item in the same batch, and their average value is used as the loss. The loss at a certain point in a session is defined as

$$L_s = -\frac{1}{N_s}\sum_{j=1}^{N_s} \ln\left(\sigma\left(\hat{r}_{s,i} - \hat{r}_{s,j}\right)\right), \tag{21}$$

where N_s is the number of samples, $\hat{r}_{s,i}$ is the score of the positive sample, and $\hat{r}_{s,j}$ is the score of the negative sample.

Both $\hat{r}_{s,i}$ and $\hat{r}_{s,j}$ are the output of GRU through the LeakyReLU activation function, and σ is the sigmoid function.

3.6. Hierarchical Network with User Memory Matrix

(1) We group sessions by the user set $U = \{u_1, u_2, \ldots, u_N\}$, and the sessions of each user u are arranged in chronological order. The sequence of user-item interactions in the session is arranged chronologically.

(2) In the training of the same user, the different sessions are horizontally stitched together to form a triplet ⟨UserId, SessionId, ItemId⟩ and sent into the session-level GRU.

(3) The read module reads the memory matrix $\mathbf{M}$ according to the GRU hidden state h_s of the user's current session s. The memory m_i^u read by the read module is used as the user's preference vector p^u to initialize the hidden layer unit of the GRU.

(4) The memory Write module writes the final state of the session to the memory network when a time step of the GRU ends and updates the memory matrix for training at the user-level.

The pseudocode of the HNUM2 execution process is shown in Algorithm 1.

4. Experiments

4.1. Datasets. (1). MovieLens-25M. MovieLens-25M (hereafter referred to as MovieLens) is a dataset provided by the MovieLens website developed by the GroupLens group at the University of Minnesota in the United States. MovieLens-25M is a publicly available dataset and is widely used in movie recommendations [29]. The version of the dataset used in this paper contains about 25 million rating records on the MovieLens website. To fit the algorithm proposed in this paper, the rating data for each user is sorted by time, and then the data is divided by days. We remove sessions with length less than 5 and we remove users with less than 6. For each user, 80% of sessions are used as training dataset and 20% as testing dataset.

input: triple < UserId, SessionId, ItemId >,
output: the prediction score $\widehat{V}^s = \{\widehat{v}_1^s, \widehat{v}_2^s, \ldots, \widehat{v}_m^s\}$.
(1) group the session by users into $U = \{u_1, u_2, \ldots, u_N\}$.
(2) initialize memory-matrix: **M**
(3) **for** i in epoch:
(4) **for** j in user u_i:
(5) //Session-level
(6) read **M** by reader into m_k^u as preference vector p^u
(7) **if** new session
(8) use p^u to initialize GRU hidden state h_s
(9) z_{ik} as the weight of user interest attention
(10) $m_k^u \leftarrow m_k^u \cdot (1 - z_{ik} \cdot \text{erase}_i)$, erase_i by equation (18)
(11) //User-level
(12) when the end of a time step
(13) write state to **M** by writer
(14) $m_k^u \leftarrow m_k^u + z_{ik}.\text{add}_i$, add_i by equation (20)
(15) computer the loss according equation (21)
(16) end for
(17) end for

Algorithm 1: HNUM^2.

(2). Adressa. Adressa [30] is a news dataset published in the RecTech item, which contains the contextual information about the user and details such as the headline and content of the news [31]. For registered users in the dataset, their historical behavior records can be obtained based on their IDs. The experiment in this paper needs to obtain user's long-term historical behavior information, so the registered users in the dataset can be selected as the experimental data. The dataset provides information such as the type of user's equipment and location [32]. There are start symbols and stop symbols of the session in the dataset, and the session can be divided accordingly. There are two versions of the dataset, one is a large dataset with 20 million reading behaviors with 10 weeks of traffic on the Adresseavisen news portal, and the other is a small dataset with 2 million reading behaviors with only one week of traffic. In this paper, we use a large dataset containing 20 million reading behaviors and filter out users with at least 5 sessions and at least 6 session lengths. We use 80% of these users as the training dataset and 20% as the testing dataset.

4.2. Evaluation Standard. **Recall@K**: Since the recommender systems can only recommend several items simultaneously, the actual items that users may choose should be in the first few items in the list. Therefore, the first evaluation metric of this paper is Recall@K, which indicates the proportion of required items among the top-K items in all test cases. In some scenarios, Recall does not consider the actual ranking of the items, while the absolute order is not important [33]. The traditional calculation formula of Recall is as follows:

$$\textbf{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{22}$$

where TP represents the number of positive samples predicted as positive samples, FN represents the number of positive samples predicted as negative samples, and Recall measures that multiple positive samples are divided into positive samples. In the personalized ranking task of the recommender systems, the calculation of Recall is defined as follows:

$$\textbf{Recall} = \frac{\sum_{u \in U} |R(u) \cap T(u)|}{\sum_{u \in U} |T(u)|}, \tag{23}$$

where $R(u)$ refer to the list of N items recommended for user u and $T(u)$ refer to the set of items preferred by user u in the testing dataset.

The work in this paper used the method of calculating Recall used in [7], which regarded session-based recommendation as a task of the item-by-item recommendation. There is only one target item in the current stage of the session. The final Recall score is the average of all users.

MRR@K: The second evaluation metric used in the experiment is Mean Reciprocal Rank (MRR), which is the average of the reciprocal rank of the required items. If the rank is greater than K, the reciprocal rank is set to 0. MRR considers the ranking of the items, which is very important in focusing on recommendations. The calculation formula is as follows:

$$\textbf{MRR} = \frac{1}{Q} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}, \tag{24}$$

where $|Q|$ indicates the number of items of interest to the users, and rank_i indicates the ranking of items that the users are interested in, in the recommendation list. When the rank of the real value is greater than the set cut-off value, the inverse of the rank is set to 0. MRR better reflects the quality of the recommendation in the ranking problem, because people tend to pay more attention to the first few items in the recommendation list [34]. When the rank of the real value is very low, even if the real value is in the recommendation list, it cannot be considered a high-quality recommendation result.

4.3. Experimental Design. Firstly, we introduce the software and hardware platform used in the experiment. In this paper, we use the Tensorflow framework to build the model, and experiments are carried out on the hardware platform Tesla P100. During the training process, RMSProp is used as an optimizer to optimize the model, and the batch_size is set to 128. For the experimental environment, the better balance between performance and efficiency can be achieved when the batch_size is 128. The parameters of the model are initialized by the normal distribution, which has a mean of 0 and a standard deviation of 0.01. The initial learning rate is 0.001, and the attenuation coefficient of the learning rate is 0.96. To avoid overfitting, the parameter keep_prob of dropout is 0.8 and the number of GRU units is 100. It is found that, due to the complex structure of the network, the saturation of the activation function often occurs when using Tanh as the activation function, resulting in falsely high experimental results. Therefore, LeakyReLU is used as the activation function after the output layer of the GRU unit [35]. The formula of LeakyReLU function is as follows:

$$y_i = \begin{cases} x_i & \text{if } (x_i \geq 0) \\ a_i x_i & \text{if } (x_i < 0) \end{cases}, \tag{25}$$

where $a_i \in (0, 1)$. The LeakyReLU function does not produce saturation and avoids neuron death [36]. In the traditional memory matrix, the number of memory vectors is usually set within 2–15, and its length is set to 100. All hyperparameters are the optimal choices obtained after adjustment based on experimental results.

4.4. Analysis of Experimental Results

4.4.1. The Effectiveness of the Algorithm. To explore the recommendation performance of the HNUM^2 model, we compared the proposed model with the HGRU model and the GRU4REC model for experiments. The HGRU model and GRU4REC model are described below.

The GRU4REC model [7] is a classic session recommendation model based on deep learning, which uses GRU to capture the user's interests in the session and then generates a recommendation list according to the user's interests. This model is a common baseline algorithm model in the field of session recommendation.

The HGRU model [8] is a hierarchical session recommendation model in which both layers of the model use GRU units to capture user's interests. Throughout the session, the model evolves potential hidden states on RNN endpoints across sessions and uses hidden states on GRU to represent user's historical interests.

To compare the parameter settings of the experiments, the HNUM^2 model performs best when the number of memory vectors is 20 in our experiments, and we set the number of memory vectors to 20. The GRU4REC model uses 100 GRU units and the batch_size is set to 128. For the HGRU model, the number of GRU units in the session-level and the number of GRU units in the user-level are both set to 100.

As can be seen in Tables 2 and 3, the HGRU and HNUM^2 models have generally better recommendation results than the GRU4REC model in the session-based recommendation algorithm. The GRU4REC model does not consider the user's historical behavior information and captures the user's interests in the current session, whereas both the HGRU model and the HNUM^2 model utilize the user's historical behavior, so it has better recommendation performance. The HGRU model preforms weaker than HNUM^2 on Recall for the same dataset. Because the HGRU model compresses user's interests into the hidden states of GRU units when portraying user's long-term interests, this approach is not conducive to the dynamic of historical states. The proposed model using memory network avoids this situation. The experimental results show that the performance of each algorithm on the MovieLens dataset is worse than that on the Adressa dataset. MovieLens is not a dataset for session-based recommendations, and the dataset does not show that the chronological sequence of ratings is related to the viewing order. Therefore, the performance of the session-based recommendation algorithm on the MovieLens dataset is not ideal.

Compared with the baseline algorithms GRU4REC and HGRU, the HNUM^2 algorithm has better performance on Recall and MRR, which validates the effectiveness of the proposed algorithm.

4.4.2. Exploration of Long-Term Memory Ability. In order to explore the memory ability of the model to remember users' long-term interests, experiments were designed to compare the different performances of the model when the number of sessions was 10 and the number of sessions was 5. The two datasets were divided into a dataset with 5 sessions and a dataset with 10 sessions. The experiment compares the performance improvement ratio of the HGRU model and the HNUM^2 model when the number of sessions increases.

The main comparison is the memory ability of multiple sessions before a user, while the GRU4REC model only considers sessions and not users, so we do not compare GRU4REC.

Comparing the data in Tables 4 and 5 with the data in Table 2, it can be seen that both HGRU and HNUM^2 show improvements in Recall and MRR when the number of sessions is selected as 10, but the degree of improvement is different. Figures 3 and 4 compare the percentage performance improvement of the two models with 10 sessions versus 5 sessions.

As shown in Figures 5 and 6, it can be seen that the HNUM^2 model can obtain the improvement of recommendation effects when facing longer number of sessions. The reason is that the memory network can store long sequence of information, and more sessions can bring more user information, which can be stored in the memory network.

Table 2: Results of Recall@K and MRR@K on the Adressa dataset with 5 sessions.

	Recall@5	Recall@10	Recall@20	MRR@5	MRR@10	MRR@20
GRU4REC	0.1023	0.1854	0.3074	0.0620	0.0761	0.0846
HGRU	0.1607	0.2897	0.4529	0.0906	0.1104	0.1224
HNUM^2	**0.1638**	**0.2935**	**0.4550**	**0.0931**	**0.1129**	**0.1249**

Table 3: Results of Recall@K and MRR@K on the MovieLens-25M dataset.

	Recall@5	Recall@10	Recall@20	MRR@5	MRR@10	MRR@20
GRU4REC	0.0213	0.0450	0.0831	0.0090	0.0146	0.0173
HGRU	0.0247	0.0571	0.0958	0.0112	0.0191	0.0298
HNUM^2	**0.0286**	**0.0623**	**0.1034**	**0.0134**	**0.0227**	**0.0326**

Table 4: Results of Recall@K on the Adressa dataset with 10 sessions.

	Recall@5	Recall@10	Recall@15	Recall@20
HGRU	0.1682	0.3027	0.3976	0.4693
HNUM^2	**0.1775**	**0.3146**	**0.4096**	**0.4831**

Table 5: Results of MRR@K on the Adressa dataset with 10 sessions.

	MRR @5	MRR@10	MRR@15	MRR@20
HGRU	0.0954	0.1159	0.1240	0.1271
HNUM^2	**0.1018**	**0.1225**	**0.1307**	**0.1353**

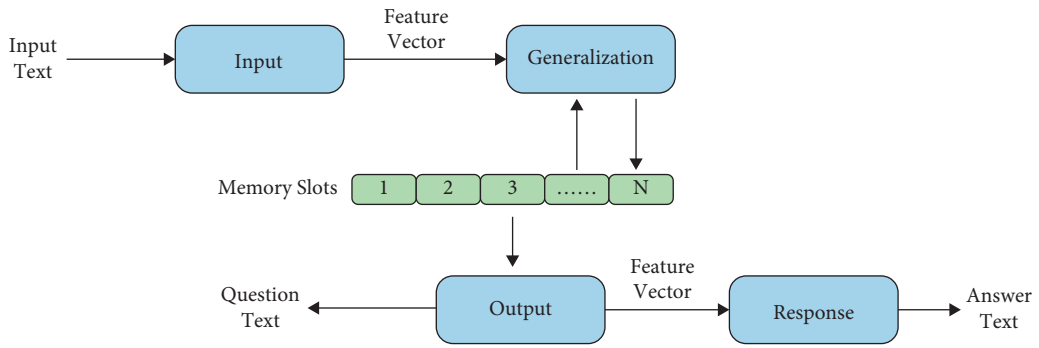

Figure 4: The general structure of the memory network, which consists of 5 components: Memory module, Input module, Generalization module, Output module, and Response module.

4.4.3. Effect of Parameter K on the Model. To explore the effect of different number of vectors K in the memory matrix on the model performance, a comparison experiment was designed. The experimental dataset was selected as the Adressa dataset that better fits the session recommendation model. The value of K for fixed TOP-K is constant at 20, while a dataset with 10 sessions is used. Different memory vector numbers K were selected to calculate the Recall and MRR values of the model. As shown in Figures 7 and 8, the values of Recall and MRR vary continuously with the value of K. The value of K determines the number of memory vectors, which in turn affects the memory ability of the model. As the number of memory vectors increases, the model can capture the user's long-term interests.

As shown in Figures 7 and 8, the value of K has a certain influence on the model effect, which shows an increasing trend followed by a decreasing trend. This is because more memory vectors can store more user information, and the user's interests that can be described become more accurate. As shown in Figure 7, the Recall of the model zigzags up for K values from 2 to 7, reaching an optimal value of 0.4813 at $K = 10$. As the value of K increases, there is no corresponding improvement in the recommendation performance of the model. However, when the number of

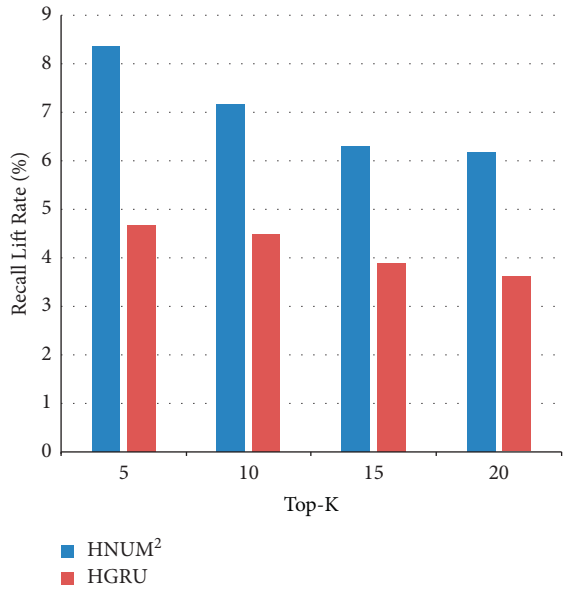

Figure 5: Comparison of long sessions on Recall improvement.

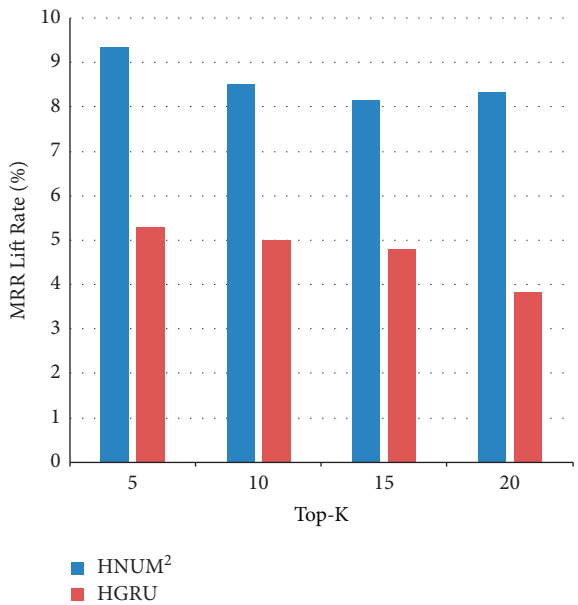

Figure 6: Comparison of long sessions on MRR improvement.

memory vectors increases to more than 10, the recommendation performance of the model begins to decline slightly. The reason is that the memory matrix generates more noise when the number of memory vectors is large. Figure 8 shows that the MRR achieves the optimal value of 0.1353 when the value is 10. Therefore, we can conclude that appropriately increasing the number of memory vectors can improve the memory ability of the memory matrix but also brings problems such as noise. Appropriate control of the number of parameters and the priority of the more important influencing factors is an important way to improve the recommendation performance.

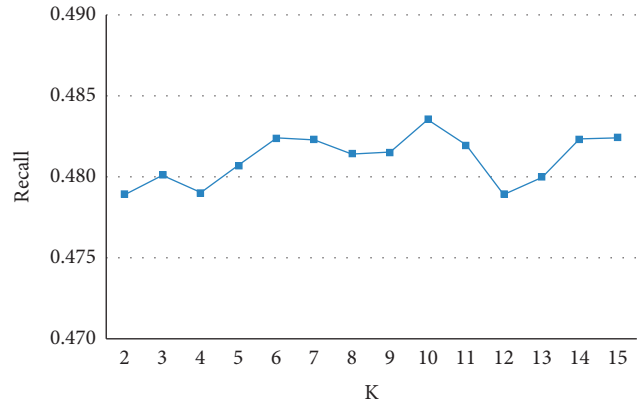

Figure 7: The influence of K on Recall.

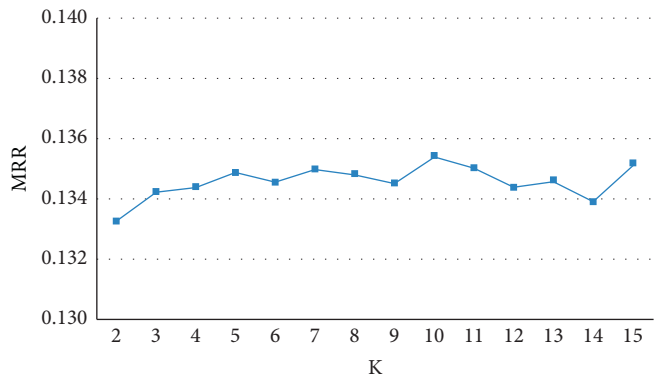

Figure 8: The influence of K on MRR.

5. Conclusion

In order to solve the problem of the insufficient memory ability of traditional recurrent neural networks, we proposed a hierarchical network with a user memory matrix (HNUM^2). In the proposed model, we use a memory network, which can capture the user's long-term interests and combine user's long-term and short-term interests to generate recommendations, which in turn improves the overall recommendation effectiveness of the algorithm. The experimental results show that the proposed model has better performance in session recommendation and better recommendation for problems with long sequences.

With the continuous improvement of information technology, the form of data has changed greatly, from the traditional scoring data to multisource heterogeneous information including images, text, and labels. The following work can further explore the fusion of multisource heterogeneous information. The current graph neural network as a new method for long sequence recommendation has opened up a new direction for sequence recommendation, and future work can apply memory networks to graph neural networks to improve the long sequence memory capability of the model.

Data Availability

All data included in this study are available upon request by contact with the corresponding author.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by Shandong Provincial Natural Science Foundation, China (ZR2020MF147).

References

[1] P. He, H. Wu, C. Zeng, and Y. Ma, "Truser: an Approach to service recommendation based on trusted users," *Chinese Journal of Computers*, vol. 42, no. 4, pp. 851–863, 2019.

[2] S. Wang, L. Hu, Y. Wang, L. Cao, Q. Sheng, and M. Orgun, "Sequential recommender systems: challenges, progress and prospects," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, pp. 6332–6338, Macao, China, July 2019.

[3] S. Wang, Y. Wang, Q. Sheng, M. Orgun, L. Cao, and D. Lian, "A survey on session-based recommender systems," *ACM Computing Surveys*, vol. 9, no. 4, 2021.

[4] L. Huang, B. Jiang, S. Lv, Y. Liu, and D. Li, "Survey on deep learning based recommender systems," *Chinese Journal of Computers*, vol. 41, no. 7, pp. 1619–1647, 2018.

[5] G. Lu and W. Zhang, "Survey of deep learning applied in recommendation system," *Software Engineering*, vol. 23, no. 2, pp. 5–8, 2020.

[6] G.-E. Yap, X.-L. Li, and P. S. Yu, "Effective next-items recommendation via personalized sequential pattern mining," *Database Systems for Advanced Applications*, vol. 7239, pp. 48–64, 2012.

[7] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, Vancouver, Canada, April 2016.

[8] M. Quadrana, A. Karatzoglou, and B. Hidasi, "Personalizing session-based recommendations with hierarchical recurrent neural networks," in *Proceedings of the Conference on Recommender Systems*, pp. 130–137, Como, Italy, August 2017.

[9] J. Liu, Y. Wang, and X. Luo, "Research and development on deep memory network," *Chinese Journal of Computers*, vol. 43, no. 2, pp. 1–52, 2020.

[10] J. Weston, S. Chopra, and A. Bordes, "Memory networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, Vancouver, Canada, November 2015.

[11] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," arXiv preprint arXiv:1410.5401, 2014.

[12] S. Sukhbaatar, A. Szlam, and J. Weston, "End-to-end memory networks," *Advances in Neural Information Processing Systems*, vol. 28, pp. 2440–2448, 2015.

[13] X. Chen, H. Xu, and Y. Zhang, "Sequential recommendation with user memory networks," in *Proceedings of the International Conference on Web Search and Data Mininghttp*, pp. 108–116, Melbourne, Australia, February 2018.

[14] J. Huang, W. Zhao, and H. Dou, "Improving sequential recommendation with knowledge-enhanced memory networks," in *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pp. 505–514, Ann Arobor, MI, USA, June 2018.

[15] Z. Cai, Z. He, X. Guan, and Y. Li, "Collective data-sanitization for preventing sensitive information inference attacks in social networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 4, pp. 577–590, 2018.

[16] R. Ma, Q. Zhang, and J. Wang, "Mention recommendation for multimodal microblog with cross-attention memory network," in *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pp. 195–204, Ann Arbor, MI, USA, June 2018.

[17] Z. Sun, Y. Wang, Z. Cai, T. Liu, X. Tong, and N. Jiang, "A two-stage privacy protection mechanism based on blockchain in mobile crowdsourcing," *International Journal of Intelligent Systems*, vol. 36, no. 5, pp. 2058–2080, 2021.

[18] X. Xia, H. Yin, J. Yu, Q. Wang, L. Cui, and X. Zhang, "Self-supervised hypergraph convolutional networks for session-based recommendation," *Association for the Advancement of Artificial Intelligence*, https://arxiv.org/abs/2012.06852, 2020.

[19] X. Yue, Y. Liu, and C. Yu, "Session-based multi-rate RNN recommendation model," *Journal of Shanxi University*, vol. 42, pp. 332–339, 2019.

[20] M. Aghdam, N. Hariri, and B. Mobasher, "Adapting recommendations to contextual changes using hierarchical hidden Markov models," in *Proceedings of the 9th ACM Conference on Recommender Systems*, pp. 241–244, Vienna, Austria, September 2015.

[21] W. Gu, S. Dong, and Z. Zeng, "Increasing recommended effectiveness with Markov chains and purchase intervals," *Neural Computing & Applications*, vol. 25, no. 5, pp. 1153–1162, 2014.

[22] Q. He, D. Jiang, and Z. Liao, "Web query recommendation via sequential query prediction," in *Proceedings of the 2009 IEEE 25th International Conference on Data Engineering*, pp. 1443–1454, Shanghai, China, April 2009.

[23] Z. Cai and X. Zheng, "A private and efficient mechanism for data uploading in smart Cyber-Physical systems," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 2, pp. 766–775, 2020.

[24] M. Gao and B. Xu, "Recommendation algorithm based on recurrent neural network," *Computer Engineering*, vol. 45, no. 8, pp. 198–202+209, 2019.

[25] Y. Zhan, X. Luo, and Y. Wang, "Supervised hierarchical deep hashing for cross-modal retrieval," in *Proceedings of the ACM International Conference on Multimedia*, pp. 3386–3394, Seattle, WA, USA, August 2020.

[26] Y. Song and J. Lee, "Augmenting recurrent neural networks with high-order user-contextual preference for session-based recommendation," 2018, https://arxiv.org/abs/1805.02983.

[27] Y. Wang, Z. Cai, Z.-H. Zhan, B. Zhao, X. Tong, and L. Qi, "Walrasian equilibrium-based multiobjective optimization for task allocation in mobile crowdsourcing," *IEEE Transactions on Computational Social Systems*, vol. 7, no. 4, pp. 1033–1046, 2020.

[28] Y. Zhan, Y. Wang, Y. Sun, X. Wu, X. Luo, and X. Xu, "Discrete online cross-modal hashing," *Pattern Recognition*, vol. 122, Article ID 108262, 2021.

[29] Z. Cai and Z. He, "Trading private range counting over big IoT data," in *Proceedings of the 39th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pp. 144–153, Dallas, TX, USA, July 2019.

[30] J. Gulla, L. Zhang, and P. Liu, "The Adressa dataset for news recommendation," in *Proceedings of the International*

Conference on Web Intelligence, pp. 1042–1048, Leipzig, Germany, August 2017.

[31] Y. Wang, Z. Chen, X. Luo, and X. Xu, "High-dimensional sparse cross-modal hashing with fine-grained similarity embedding," in *Proceedings of the Web Conference 2021*, pp. 2900–2909, New York, NY, USA, April 2021.

[32] X. Zheng and Z. Cai, "Privacy-preserved data sharing towards multiple parties in industrial IoTs," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 5, pp. 968–979, 2020.

[33] Z. Cai, Z. Xiong, H. Xu, P. Wang, W. Li, and Y. Pan, "Generative adversarial networks: a survey towards private and secure applications," *ACM Computing Surveys*, vol. 37, no. 4, 2020.

[34] Y. Wang, Y. Gao, Y. Li, and X. Tong, "A worker-selection incentive mechanism for optimizing platform-centric mobile crowdsourcing systems," *Computer Networks*, vol. 171, no. C, 2020.

[35] T. Liu, Y. Wang, Y. Li, X. Tong, L. Qi, and N. Jiang, "Privacy protection based on stream cipher for spatiotemporal data in IoT," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 7928–7940, 2020.

[36] Z. Lu, Y. Wang, X. Tong, P. Wang, C. Mu, and Y. Li, "Data-driven many-objective crowd user selection for mobile crowdsourcing in industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 99, 2021.