Applied Soft Computing Journal 85 (2019) 105762

Contents lists available at ScienceDirect

Applied Soft Computing Journal

journal homepage: www.elsevier.com/locate/asoc

Recurrent Tensor Factorization for time-aware service recommendation

Yiwen Zhang [a], Chunhui Yin [b], Zhihui Lu [c,d,*], Dengcheng Yan [e], Meikang Qiu [f], Qifeng Tang [g,h]

[a] Key Laboratory of Intelligent Computing & Signal Processing, Ministry of Education, Anhui University, Hefei, Anhui, China
[b] School of Computer Science and Technology, Anhui University, Hefei, Anhui, China
[c] School of Computer Science, Fudan University, Shanghai, China
[d] Engineering Research Center of Cyber Security Auditing and Monitoring, Ministry of Education, Shanghai, China
[e] Institutes of Physical Science and Information Technology, Anhui University, Hefei 230601, China
[f] The Department of Electrical Engineering, Columbia University, New York, NY, 10027, USA
[g] Shanghai Data Exchange Corp., Shanghai, China
[h] National Engineering Laboratory for Big Data Distribution and Exchange Technologies, Shanghai, China

ARTICLE INFO

Article history:
Received 1 May 2019
Received in revised form 12 August 2019
Accepted 30 August 2019
Available online 16 September 2019

Keywords:
Deep Learning
Tensor Factorization
Gated Recurrent Unit
Recurrent Tensor Factorization
Quality of Service (QoS)

ABSTRACT

With the advent of the era "everything is service ", the emergence of Web services on the Internet is experiencing an exponential growth trend. How to recommend services to users that utilize sequential historical records has become one of the most challenging research topics in service computing. Tensor Factorization (TF) and Long Short Term Memory (LSTM) networks are two typical application paradigms for sequential service recommendation tasks. However, TF can only learn static short-term dependency patterns between users and services, ignoring the dynamic long-term dependency patterns between users and services. Although LSTM in Deep Leaning can learn dynamic long-term dependency patterns, it often encounters the trouble of vanishing gradient due to its complex gated mechanism. To address these critical challenges, we develop a novel model based on Deep Learning named Recurrent Tensor Factorization (RTF) with three innovations: (1) Three-dimensional interaction tensor of user–service-time was granulated into three fixed-size embedding dense vectors. (2) Personalized Gated Recurrent Unit (PGRU) and Generalized Tensor Factorization (GTF) simultaneously work on shared embedding dense vectors to memorize the long-term and short-term dependency patterns between users and services respectively. (3) Armed with GTF and PGRU, RTF is competent to predict the unknown Quality of Service (QoS) through comprehensive analysis. Experiments are conducted on real-world dataset, and the results indicate that our proposed method obviously outperforms six state-of-the-art time-aware service recommendation methods.

1. Introduction

With the advent of cloud services, big data, and the Internet of Things (IoT) era, various Web services (e.g., cloud services, mobile services, etc.) are emerging on the Internet, which make it difficult for engineers to select appropriate services when developing Web APIs. In service computing, recommending services to users to meet changing needs over time has become a crucial issue. Quality of Service (QoS) quantifies the behaviors (or preferences) of the users in service-oriented recommenders, which is expressed as services' non-functional attributes (i.e., response time, throughput, and cost). Therefore, predicting the QoS of different non-functional attributes is the most challenging task of service recommendation.

Since Shao et al. [1] applied collaborative filtering (CF) to service recommendation, a large number of CF-based service recommendation methods have been proposed [2–6]. In both industry and academia, CF-based approaches have been widely used to employ users' historical behaviors (or preferences) to predict services that a current user would most likely to prefer. However, CF-based methods have the following two shortcomings: (1) Similarity calculations employed by CF-based methods can only learn low-dimensional and linear features between users and services. (2) The real-world data sparsity problem significantly affects their recommendation performance owing to inadequate feature learning.

* Corresponding author at: School of Computer Science, Fudan University, Shanghai, China.
E-mail addresses: zhangyiwen@ahu.edu.cn (Y. Zhang), yinchunhui.ahu@gmail.com (C. Yin), lzh@fudan.edu.cn (Z. Lu), yandengcheng@gmail.com (D. Yan), qiumeikang@yahoo.com (M. Qiu), keven@chinadep.com (Q. Tang).

https://doi.org/10.1016/j.asoc.2019.105762
1568-4946/

To compensate the shortcomings of CF and further improve the accuracy of recommendation, researchers [7–14] came to integrate contextual information (e.g., location, trust, and time) into similarity calculation of CF-based methods. For instance, Chen et al. [9] utilized location and trust information and proposed a trust-aware and location-based approach to predict the quality of service. Tang et al. [15] incorporated the locations of users and services into similarity calculation, which improved the accuracy of QoS prediction. In addition to location and trust, the influence of time information in service recommendation has also been studied in recent works [10,11,16,17]. For example, Hu et al. [11] proposed an improved time-aware Web service recommendation method. Zhang et al. [17] proposed a time-aware personalized QoS prediction framework named WSPred.

Although existing time-aware service recommendation methods can effectively use time information for service recommendation, the following problems still exist: (1) Lack of effective update mechanism to process new coming data. (2) Normally only learn static short-term dependency mode between users, services and times. In order to solve these problems, Recurrent Neural Networks (RNNs) in Deep Learning (DL), such as Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU), have been widely adopted. For example, Personalized LSTM-based Matrix Factorization (PLMF) is proposed by Xiong et al. [16], which is capable of learning dynamic long-term dependency patterns and provides an effective updating mechanism for new coming data. However, LSTM may cause problems such as vanishing gradient and exploding gradient due to complicated gated mechanism. GRU used in this paper is a variant of LSTM, whose simpler gated mechanism and powerful memory ability make it possible to solve LSTM's shortcomings. The main contributions of this paper are described as follows:

(1) This paper proposes a novel Deep Learning based time-aware service recommendation framework named Recurrent Tensor Factorization (RTF), which combines Tensor Factorization (TF) with Deep Learning (DL) for time-aware service recommendation.

(2) Two sub-methods, Generalized Tensor Factorization (GTF) and Personalized GRU (PGRU), are developed for time-aware service recommendation.

(3) A hybrid loss function is designed to consider both L1 loss and L2 loss for preventing the model's unbalanced fitting problem on multiple evaluation metrics.

(4) Experiments conducted on real-world Web service dataset indicate that our proposed method significantly outperforms six state-of-the-art methods.

The rest of this paper is organized as follows. Related work is discussed in Section 2. Section 3 presents the preliminary knowledge of this work. Section 4 describes the method of the study. Section 5 shows the experimental results and analysis. Finally, brief conclusions are given in Section 6.

2. Related work

Depending on whether time information is used, service recommendation methods can be classified into two categories: time-aware methods and non-time-aware methods.

2.1. Non-time-aware service recommendation

Non-time-aware service recommendation methods utilize the granulated results of the time-aware method for a single time slice regardless of the time information. The CF uses historical QoS values in single time slice to recommend services to users, which is the most representative traditional method. For instance, Wu et al. [18] improved the similarity calculation in CF and proposed a ratio-based method for service recommendation. Sun et al. [19] proposed a Normal Recovery Collaborative Filtering (NRCF) for service recommendation. Chen et al. [20] proposed a heuristic approach named iDiSC for service component deployment in the edge-cloud-hybrid system. Lu et al. [21] proposed an extended IoT Big Data-oriented model named IoTDeM for predicting MapReduce performances. However, CF-based methods are insufficient for learning features from data due to data sparsity obstacle in real-world. Recently, an increasing number of researchers have examined the impact of contextual information, such as location [12,15,22] and trust [9,23,24], to further improve the feature learning ability in CF. Although incorporating contextual information helps to predict QoS values, CF-based approaches only learn the linear relationship between users and services. With the rapid development of Deep Learning technology, many researchers have combined Deep Learning with CF to propose various methods in recent years, which can not only learn the nonlinear relationship between users and services but also alleviate the data sparsity problem. For instance, Xiong et al. [25] combined a fully connected neural network with WSDL text similarity calculation and proposed the Deep Hybrid Collaborative Filtering for Service Recommendation (DHSR). Wu et al. [26] proposed a universal deep neural model (DNM) for making multiple attributes QoS prediction with contexts by combining fully connected neural networks with WSDL text information. However, none of the above research have studied the effects of time information. Time information play a crucial role in service recommendation since QoS performance is highly related to invocation time [27]. Unlike these methods above, we incorporate time information into service recommendation in this work, which solved the limitations in these studies.

2.2. Time-aware service recommendation

Time-aware service recommendation methods utilize the generalized results of non-time-aware methods for multiple time slices by incorporating time information. Time-aware methods extend the user–service 2D matrix in non-time-aware methods to the user–service-time 3D tensor. To utilize time information, many service recommendation methods have been proposed. Hu et al. [11] proposed an improved time-aware CF method that uses the historical QoS records of the first 63 time intervals to predict the QoS of the 64th time interval. Zhao et al. [28] proposed a time-aware service recommendation model named CAPred, which employs existing QoS records on different time slices to predict missing data on each time interval. Yu et al. [29] proposed a time-aware and location-aware CF algorithm for service recommendation. Although existing CF-based methods can achieve the goal of time-aware service recommendation, it reduces the 3-dimensional user–service-time tensor to the two-dimensional user–service matrix and fills in missing values in the users–service matrix at each time interval. This dimension reduction has the following three shortcomings: (1) Cannot learn the long-term dependency pattern between users and services. (2) Cannot afford the task of memorizing time series. (3) Cannot offer an update mechanism for new coming data. Compared with convolutional neural networks (CNNs) [30], neural networks such as LSTM in Deep Learning is a more effective solution to solve these problems. Xiong et al. [16] proposed a personalized LSTM-based (PLMF) method for service recommendation. Although LSTM can learn dynamic long-term dependency patterns, it often encounters the trouble of exploding gradient owing to its complex gated mechanism. Fortunately, in order to solve these problems, many variants of LSTM have been proposed, and the Gated Recurrent Unit (GRU) proposed by Cho et al. [31] has become the most popular one. Related studies [32,33] have demonstrated that GRU outperforms LSTM in time modeling capabilities. GRU used in this paper shows huge improvement over these works.

3. Preliminaries

In this section, we briefly introduce the preliminary knowledge in this study.

3.1. Matrix Factorization (MF)

Since the Nexfix competition, MF has been widely used by many researchers in improving the prediction accuracy of recommenders. MF is often referred to Latent Factor Model (LFM) in other works, which has outstanding performance in data dimensionality reduction, missing data filling (or "sparse data filling") and implicit relationship mining.

Given a two-dimensional user–service QoS matrix $Q_{i,j}$ of size $m \times n$, $q_{i,j}$ represents QoS history true value when user i invokes the service j. When the rank of the matrix satisfies $r = Rank(Q_{i,j}) \ll \min(m, n)$, the prediction $\hat{Q}_{i,j}$ for $Q_{i,j}$ can be written as follows:

$$\hat{Q}_{i,j} = US^T \tag{1}$$

$$\hat{q}_{i,j} = \sum_{k=1}^{K}(u_{ik}v_{jk}) \tag{2}$$

where U is user factor matrix of size $m \times k$, S is service factor matrix of size $n \times k$.

MF requires $Q_{i,j}$at least has one element per column, usually optimized with the following loss function:

$$J = \frac{1}{2}\sum e_{ij}^2 = \frac{1}{2}\sum_{i,j\in s}(q_{i,j} - \hat{q}_{i,j})^2 = \frac{1}{2}\sum_{i,j\in s}(q_{i,j} - \sum_{k=1}^{K}(u_{ik}v_{jk}))^2 \tag{3}$$

Common optimization methods include random gradient descent method and least squares method.

3.2. Gated Recurrent Unit (GRU)

Although MF can simulate patterns between users and services, it is hard for MF to handle patterns that fluctuate over time. Furthermore, too many parameters in full-connected neural networks has risen the possibility of over-fitting. Fortunately, Deep Learning can exploit the useful features of data by proposing more efficient structures. Recently, RNNs have been widely employed in sequential analysis, such as speech recognition, language modeling, and machine translation. RNNs can capture the relationship between the current output of a sequence and previous information. RNNs memorize the previous information and apply it to influence the subsequent output.

Assume that we have a long time series $(x^0, x^1, \ldots, x^t)$. The RNNs generate the output x^t of the next state based on the current input and the hidden state of the previous state h^{t-1}, in which h^t is directly utilized for the output o^t (i.e., $o^t = h^t$):

$$h^t = RNN(h^{t-1}, x^t) \tag{4}$$

where *RNN* represents the current RNNs structure.

Common RNNs include Hopfield network [34], LSTM [35,36], etc. Neural networks such as LSTM is gradually replacing Hopfield network due to implementation difficulties. The LSTM was developed for alleviating the vanishing and exploding gradient problems that traditional RNNs may encounter. Although LSTM can memorize the long-term dependency patterns between users and services, there are still problems such as complex gate structure and large computational load. As discussed in Section 2.2, we use GRU in this paper. As shown in Fig. 1, GRU consists of two gating units: the update gate and the reset gate.

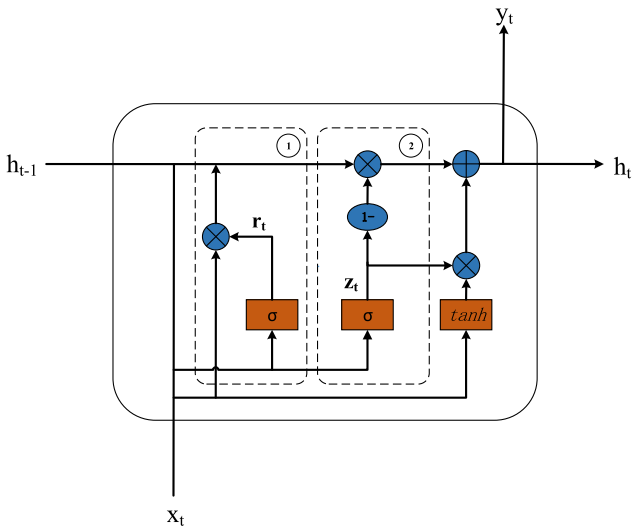

Fig. 1. Gated Recurrent Unit.

The first component in Fig. 1 represents the update gatez$_t$. The update gate z$_t$ determines how much information from the previous time interval is used for next time interval, and the update gate z$_t$ at time step t is calculated as follows:

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1}) \tag{5}$$

where σ_g represents a sigmoid function, and W_z and U_z are weight matrixes.

The second component in Fig. 1 represents the reset gate r_t, which determines how much previous state information is forgotten; r_t is calculated as follows:

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1}) \tag{6}$$

GRU comprehensively examines the update and forgetting gates and generates the candidate state $\tilde{h}_t$ which is defined as follows:

$$\tilde{h}^t = \sigma_h(W_h x_t + U(r_t \circ h_{t-1})) \tag{7}$$

where σ_h represents hyperbolic tangent function, and $\circ$ represents the Hadamard product.

The GRU obtains a hidden state at time t by linearly interpolating the previous state h^{t-1} and the candidate state $\tilde{h}^t$:

$$h^t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t \tag{8}$$

The GRU combines LSTM's forget gate and input gate into a single update gate and combines the cell state of the LSTM with the hidden state as the new hidden state. The GRU can achieve better performance than the LSTM.

4. Proposed model

In this section, we present our proposed Generalized Tensor Factorization (GTF) and Personalized Gated Recurrent Unit (PGRU) according to Section 3.

4.1. Generalized Tensor Factorization (GTF)

As described in Section 3.1, MF is a popular research topic in the field of recommenders. TF is very similar to MF, since TF is a high-order generalized product of MF. One-dimensional vectors are first-order tensor, and two-dimensional arrays are second-order tensor.

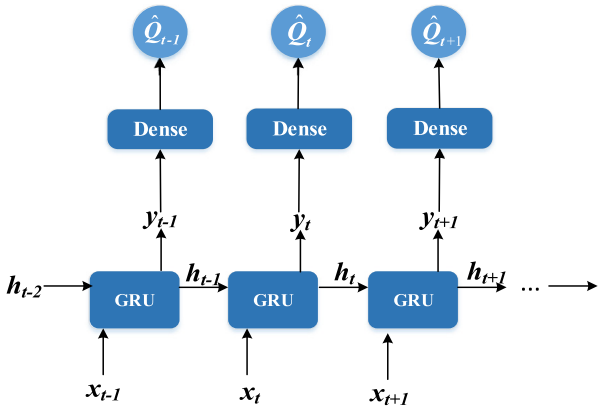

Fig. 2. Personalized Gated Recurrent Unit (PGRU).

Recently, Aggarwal et al. proposed a special TF structure in their study [37]. Inspired by their research, we propose Generalized Tensor Factorization (GTF). Assume that we have user–service-time QoS Tensor Q with size $l \times m \times n$, Q can be further factorized to three two-dimensional tensor (matrix): user factor matrix U of size $m \times k$, service factor matrix S of size $n \times k$ and time factor matrix T of size $d \times k$. Here, T also regarded as context factor matrix since it can be replaced by any other context informations, such as location and reputation. k is the size of the latent factor. The candidate QoS prediction values corresponding to the position (i, j, c) on the third-order tensor are calculated as follows:

$$\tilde{Q}_{i,j,c} = (US^T)_{i,j} + (UT^T)_{i,c} + (ST^T)_{j,c} \tag{9}$$

$$\tilde{q}_{i,j,c} = \sum_{q=1}^{k} (u_{iq} v_{jq} + u_{iq} t_{c,q} + v_{j,q} t_{c,q}) \tag{10}$$

The original TF only learns the linear relationships among users, services and times, which causes insufficient feature learning. To address this problem, we apply the single fully connected perceptron layer to the candidate prediction value. The final prediction value is as follows:

$$\hat{q}_{i,j,c} = f_g(W_g \tilde{q}_{i,j,c} + b_{gtf}) \tag{11}$$

where f_g here, we use Rectified Linear Unit (ReLu) for non-linear transformation. Similar to the loss function of MF, GTF's loss function is usually defined as follows:

$$J = \frac{1}{2} \sum e_{ijc}^2 = \frac{1}{2} \sum_{i,j,c \in s} (q_{i,j,c} - \hat{q}_{i,j,c})^2 \tag{12}$$

Through these, GTF is capable of dealing with user–service-time tensors with nonlinear feature learning.

4.2. Personalized Gated Recurrent Unit (PGRU)

As described in Section 3.2, the input to the main structure of GRU takes into account the input vector x_t from the current time interval t and the hidden state h_{t-1} at the previous time interval $t-1$. At each time interval, the GRU will generate the output hidden state h_t of the current time and use it for the next time $t+1$. Since the operations and variables of the GRU at different time intervals are equivalent, the GRU can theoretically be considered as the result of the same network structure that is being infinitely copied. As CNNs share parameters among different spatial locations, RNNs share parameters among different time intervals, which enables the processing of arbitrary-length time-series data using finite parameters. Expanding the GRU along the time axis, we can obtain the structure shown in Fig. 2.

The expansion of GRU significantly impacts the training speed of RNNs. As shown in Fig. 2, after the GRU is expanded into a time series of length N, it can be regarded as N feedforward neural networks with intermediate layers. Similar to multilayer perceptron, GRUs can be directly trained using backpropagation algorithms. The method of training the GRU is generally referred to as Back-Propagation through Time (BPTT). GRU is expert at processing time series; thus, it can memorize the long-term behavior habits of users. We determined that the output of the original GRU can not be directly employed for personalized service quality prediction. Therefore, we add a single-layer fully connected neural network to the output of different time steps. Here, we define our optimized GRU with a fully connected output layer as the Personalized Gated Recurrent Unit (PGRU). PGRU is a Deep Learning based method for time-aware service recommendation.

4.3. Recurrent Tensor Factorization (RTF)

In this section, we integrate GTF with the PGRU to construct our Recurrent Tensor Factorization (RTF). Specifically, RTF is based on the following four steps: projection, initialization, training and optimization.

4.3.1. Step 1: Projection

Since processing original 3D user–service-time tensors directly consumes precious memory resources, we granulate the tensors into the multiple cubes tensor. As shown in step 1 of Fig. 3, the length, width and depth of each cube represent a user, a service, and a time interval, respectively, and the value of the cube is assigned a true QoS label. Then, we project the user, service and time into the embedding space. Embedding performs one-hot encoding and outputs three fixed-size dense vectors: user latent vector, service latent vector and time latent vector. All latent vectors are projections in latent space, and each vector is a unique one-dimensional tensor that can be employed for neural network training. Similar to Word2vec, Doc2vec, and GloVe, this projection uses dense vectors to represent words or documents, which has been widely used in natural language processing. This process is formulated as follows:

$$u^k = f_E(W_u^T x^u) \tag{13}$$

$$s^k = f_E(W_s^T x^s) \tag{14}$$

where k represents the dimension of the latent vector; u^k, s^k and t^k are the user latent vector, service latent vector, and time latent vector, respectively; and f_E is the embedding activation. We use standard identity functions. W_u and W_s represent the embedding weight matrix, which we initialize according to the Gaussian distribution. W_u and W_s can be separately learned using a model persistence mechanism to initialize. Thus, the goal of accelerating the convergence of the model and improving the expected effect can be achieved. Embedding can extract useful information from the data, which is necessary for feature extraction of neural networks.

4.3.2. Step 2: Initialization

In supervised learning, the initialization of model parameter is the key to the training process. PGRU and GTF require different inputs and therefore need to be initialized separately. As shown in Fig. 3, for PGRU, we concatenate the user latent vector with the service latent vector to acquire the input vector required by PGRU at time interval t. We initialize the hidden state at the

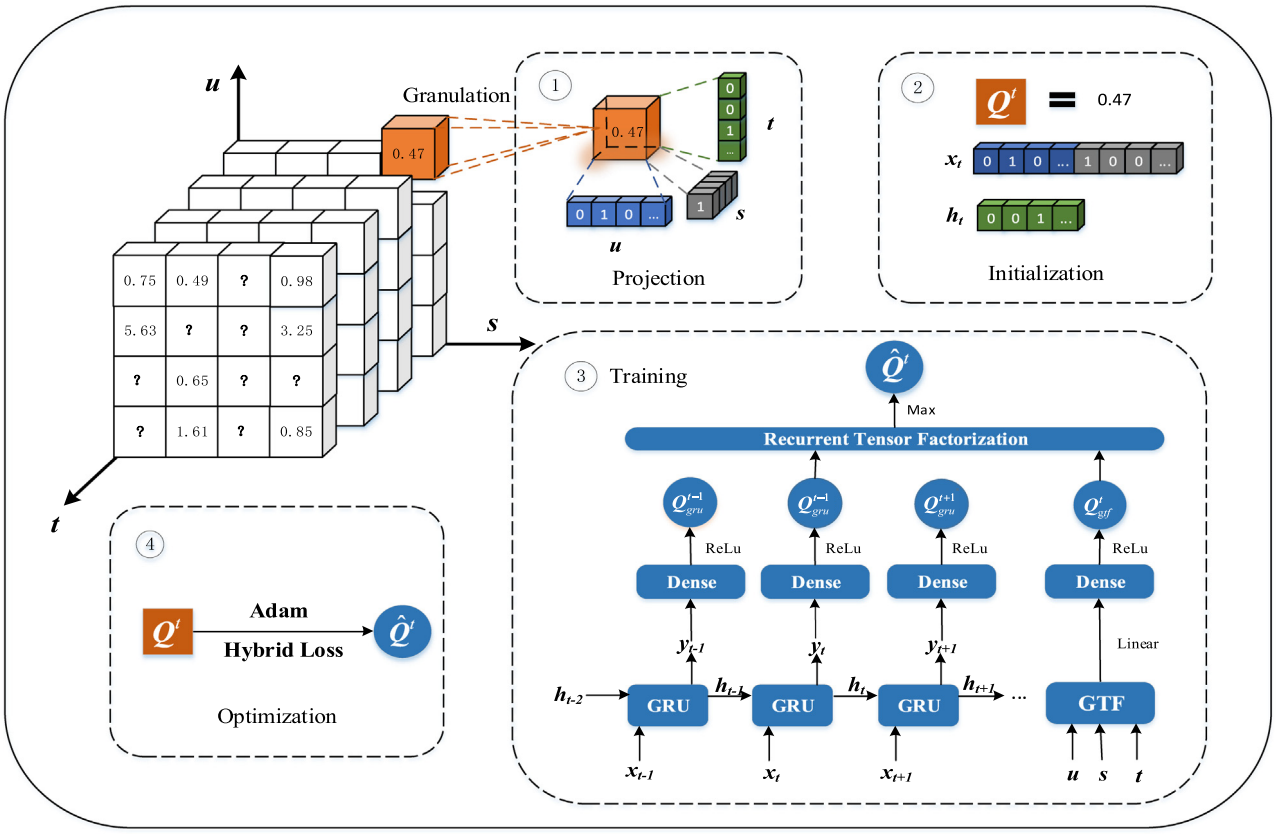

Fig. 3. Overall architecture.

current time interval to time embedding vector. Since RTF models patterns between users, services and times in two paths. Inspired by [38,39], RTF directly connects the outputs of these two paths. The formula defines as follows:

$$x_t = \varphi(u^k, s^k) = \left(u^k, s^k\right) \tag{15}$$

$$h_t = t \tag{16}$$

where φ represents concatenation, h^t represents the hidden state at time t.

4.3.3. Step 3: Training

After unfolding through the time, RNNs can be regarded as the feedforward neural network of N intermediate layers. Step 3 in Fig. 3 describes the forward propagation process of the RTF. Recurrent Tensor Factorization (RTF) proposed by us uses the maximum function to connect GTF and PGRU. Formula defined as follows:

$$\tilde{Q}^t_{pgru} = f_P(W_P y_t + b_P) \tag{17}$$

$$\tilde{Q}^t_{gtf} = f_G(W_G y'_t + b_G) \tag{18}$$

$$\hat{Q}^t_{rtf} = max(\tilde{Q}^t_{gtf}, \tilde{Q}^t_{pgru}) \tag{19}$$

where $\tilde{Q}^t_{pgru}$, $\tilde{Q}^t_{gtf}$, and $\hat{Q}^t_{rtf}$ represent the outputs of PGRU, GTF and RTF, respectively; f_P and f_G the ReLu activation; y_t and y'_t the outputs of GRU and TF, respectively.

4.3.4. Step 4: Optimization

The neural-network-based model optimizes the target parameter by defining the loss function. The choice of the loss function plays a crucial role in the final effect of the model. To address different usage scenarios, supervised learning tasks can be divided into classification and regression. In classification tasks, commonly employed loss functions are cross-information entropy loss, zero loss, logistical loss and hinge loss. In the model used in classification tasks, the output is often the probability that the current input belongs to each class. Unlike the classification problem, the regression problem solves the prediction of specific values. In regression tasks, commonly employed loss functions are the least absolute loss (L1), least squared loss (L2), and Huber loss [40]. L1 measures the sum of the squared error between the predicted value and the true value but lacks robustness for outliers in the data. L2 measures the absolute value of the error between the predicted value and the true value which is robust to outliers; however, the optimal solution can be easily missed due to the fixed gradient. The Huber loss combines L1 with L2 using hyper-parameter and offers the following idea: L1 loss is employed when the error is large, and L2 is employed when the error is small. However, it is difficult to determine the hyper-parameter. To tackle these obstacles, we proposed the hybrid loss that combines L1 loss and L2 loss using a weighting factor to share similar spirit with Huber loss. In this way, hybrid loss helps bridge the gap between L1 loss and L2 loss, and is defined as follows:

$$L1 : J = L_1(Q^t, \hat{Q}^t) = \omega(\theta)\left|\hat{Q}^t - Q^t\right| \tag{20}$$

$$L2 : J = L_2(Q^t, \hat{Q}^t) = \omega(\theta)\left(\hat{Q}^t - Q^t\right)^2 \tag{21}$$

$$Hybrid\ Loss = \delta L1 + (1-\delta)L2 \tag{22}$$

where $\omega(\theta)$ represents true value weight. δ represents the weighting factor used to balance L1 loss and L2 loss, of which the default value is set to 0.5.

In addition to the loss function, another concept that is critical to neural networks is the optimization algorithm. Currently, the mainstream neural network optimization algorithms include the backpropagation algorithm and the gradient descent algorithm. The gradient descent algorithm focuses on optimizing the value of a single parameter, while the backpropagation algorithm uses gradient descent for all parameters. Thus, the loss of the neural network model using training data is as small as possible. In RNNs, the most extensively employed update method of the model parameters is the Backpropagation-Through-Time (BTT) algorithm. The optimizer is the container of the optimization algorithm. We have experimented with optimizers such as RMSprop [41], Adadelta [42], Adam [43], Nadam [44], among which the Adam optimizer performs best. Thus, we choose Adam as the Optimizer. Due to the space setting, the derivation formula for the optimization algorithm is omitted here. To make the model more robust in the test data line, we adopted a regularization mechanism, dropout mechanism, and sliding average mechanism to prevent overfitting of the model. The L2 regularization adds a bias term to the network weight. Dropout refers to randomly disconnecting the input neurons by a certain probability (rate) each time the parameter updates. The sliding average mechanism enables the learning rate to decay each time the parameter updates. The sliding average mechanism is defined as follows:

$$l' = l \times (1 + decay \cdot epoch)^{-1} \tag{23}$$

where l and l' are the learning rate before update and the learning rate after update, respectively; *decay* is the decay factor, and epoch is the current iteration number. The above formulas are all progressive.

5. Experiments

In this section, we conduct experiments to evaluate our proposed method and compare it with the six most advanced methods. We answer the following questions:

(1) Can our proposed method be superior to the state-of-the-art service recommendation methods? Does time information impacts recommendation accuracy?

(2) Can RTF capture the complex dependency patterns between users, services and times? Does the integration of GTF and PGRU contribute to recommendation performance?

(3) Can our proposed hybrid loss balance the global performance of the model across multiple evaluation metrics? Does it improves the fitting ability of the model compared to other loss functions?

(4) Why we have to use dropout? Can overfitting be alleviated by dropout to some extent?

5.1. Dataset

Our experiments are conducted on the public dataset WS-Dream QoS dataset#3 provided by Zheng et al. [45], which is the most representative dataset and has been widely used in related works [46–48]. This dataset describes real-world QoS measurements from 142 users on 4532 Web services over 64 consecutive time intervals (each interval is 15-minute). Each QoS measurement includes two QoS nonfunctional attributes: response time and throughput. We can extract two sets of user–service-time tensors with response time or throughput attributes of $142 \times 4532 \times 64$ from the dataset, where the first dimension of the tensor represents the user, the second dimension represents the service, and the third dimension represents the time interval.

5.2. Pre-processing

To make our experiment more realistic, we randomly remove some entries from the original user–service-time tensor and compare the predicted values of the method with the original values to produce the tensors at different densities. For example, 5% indicates that only 5% of the entries are reserved for the training set of the model, and 95% of the entries that are randomly removed from the original tensor serve as the test set for the model. The detailed description is described as follows: assuming that the current time interval is t (t varies from 0 to 63), the original $142 \times 4532 \times 64$ QoS tensor can be sliced according to the time dimension to obtain a two-dimensional tensor of 142×4532 in 64 dimensions. According to different densities, it can be further divided into the training set Q^t_{train} and the test set Q^t_{test}, which can obtain 64 two-dimensional QoS tensors $Q^0_{train} Q^1_{train} \ldots, Q^t_{train}$ for training and 64 two-dimensional QoS tensors $Q^0_{test} Q^1_{test} \ldots, Q^t_{test}$ for testing. Our method will directly apply to each of these tensors.

5.3. Evaluation metrics

The mean absolute error (*MAE*) can offset the problem that the errors cancel each other; thus, it can accurately reflect the prediction error. The root mean squared error (*RMSE*) is very sensitive to very large or small errors in a set of test cases and is an adequate reflection of the precision of the prediction. We adopt*MAE* and *RMSE* to evaluate the recommendation performance of the method. *MAE* is defined as follows:

$$MAE = \frac{\sum_{u,s} \left| Q^t_{u,s} - \hat{Q}^t_{u,s} \right|}{N} \tag{24}$$

where $Q_{u,s}$ is the QoS original value of user u who invoked the service s at time interval t; $\hat{Q}_{u,s}$ is the predicted QoS value of user u who invokes the service s at time interval t; and N is the total number of QoS.

The *RMSE* is defined as follows:

$$RMSE = \frac{1}{N} \sum_{u,s} \left(Q^t_{u,s} - \hat{Q}^t_{u,s} \right)^2 \tag{25}$$

5.4. Comparison methods

To further verify the effectiveness of our proposed method, we compared the following six most advanced methods:

(1) **UIPCC** (User-based and Item-based CF) [49]: This method combines the similar users and similar web services adopted in UPCC and IPCC for non-time-aware service recommendation.

(2) **PMF**(Probability Matrix Factorization) [50]: This method incorporates probabilistic factors into MF for non-time-aware service recommendation.

(3) **CLUS** [51]: This method predicts the reliability for the ongoing service invocation for time-aware service recommendation.

(4) **WSPred** [17]: This method uses past invocation record for time-aware service recommendation.

(5) **NTF**(Non-negative Tensor Factorization) [52]: This method employs the non-negative tensor factorization approach for time-aware service recommendation.

(6) **PLMF**(Personalized LSTM based MF) [16]: This method combines LSTM with MF for time-aware service recommendation.

To make the experiments more convincing and comprehensive, we choose these most representative comparison methods to demonstrate the superiority of our approach. Among these methods, (1)–(2) are popular non-time-aware service recommendation methods, (3)–(6) are the most advanced time-aware service recommendation methods, and (6) is a Deep Learning based service recommendation method.

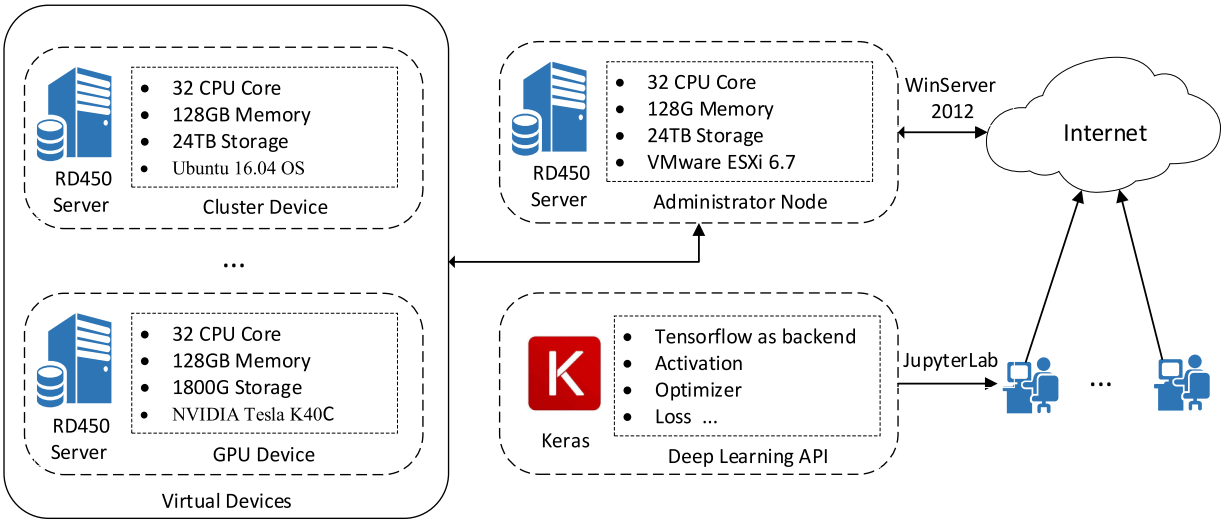

Fig. 4. Topology diagram of experimental environment.

Table 1
Performance comparison (***RQ1***).

Metrics	Methods	Response-time				Throughput			
		Tensor density				Tensor density			
		5%	10%	15%	20%	5%	10%	15%	20%
MAE	UIPCC	1.043	0.961	0.912	0.879	9.909	9.305	8.936	8.388
	PMF	1.015	0.934	0.895	0.867	6.571	5.981	5.831	5.700
	CLUS	0.919	0.886	0.856	0.830	5.628	4.769	4.198	4.171
	WSPred	0.793	0.768	0.756	0.765	4.379	4.166	4.125	4.087
	NTF	0.751	0.741	0.738	0.736	4.369	4.164	4.178	4.109
	PLMF	0.788	0.758	0.741	0.739	4.315	4.123	4.083	4.032
	RTF	**0.668**	**0.630**	**0.612**	**0.535**	**4.139**	**4.025**	**3.874**	**3.799**
RMSE	UIPCC	2.094	1.975	1.920	1.880	43.890	41.525	40.193	38.669
	PMF	2.497	2.244	2.095	1.996	40.291	36.005	33.847	32.493
	CLUS	2.223	2.263	2.249	2.217	34.549	31.087	28.260	26.502
	WSPred	1.817	1.788	1.774	1.786	23.612	22.365	22.031	22.161
	NTF	1.742	1.730	1.726	1.724	24.216	23.043	22.128	21.862
	PLMF	1.815	1.767	1.742	1.736	25.635	24.523	23.845	22.102
	RTF	**1.732**	**1.666**	**1.634**	**1.636**	**22.781**	**20.809**	**20.166**	**19.897**

5.5. Experimental settings

5.5.1. Environmental settings

The experimental topology diagram is shown in Fig. 4. In the hardware environment, we use VMware ESXi to manage virtual appliances (Spark cluster nodes). Each cluster node is configured with 32 CPU cores, 128GB memory and 24TB disk storage, and Ubuntu 16.04 is deployed on each node. Specifically, we use the GPU-equipped device through the WinServer jump server. We experimented on the node configured with NVIDIA Tesla K40C (11 GB memory). In the software environment, we implement our method on JupyterLab platform by using Keras API (TensorFlow as the backend). Keras is an open-source Deep Learning library. For traditional methods, we use their public C++ version, and for PLMF, we choose its PyTorch version.

5.5.2. Parameter settings

We establish different tensor densities to construct training sets and testing sets from 5% to 20% with a step size of 5%. Both the embedding layer and the GRU layer of our method use the standard Gaussian distribution to initialize parameters, and for the output fully connected layer, the parameters are initialized with the LeCun Gaussian distribution initializer. We set the model learning rate to 0.001, and the optimizer uses the Adam Optimizer. We set the activation of the GRU to a hyperbolic tangent; the activation function to use for the recurrent step is hard sigmoid. We set the dimension of the latent vector to 1024, which enables the neural network to learn more information. For other comparison experiments, we set the parameters according to the original text, and no changes are made here. For PLMF and our RTF, we run iterations 50 times and take the optimal value after fitting the model.

5.6. Experimental results and analysis

In this section, we conduct substantial experiments to evaluate our method.

5.6.1. Performance comparison (RQ1)

Experiments that compare our approach with six other methods under four tensor densities (%5∼20%) have been conducted and the corresponding results on response-time and throughput are shown in Table 1.

In the response-time comparison, our RTF method has achieved the best performance on both MAE and RMSE. When evaluating *MAE*, UIPCC performs worst, followed by PMF and CLUS. PMF, CLUS are non-time-aware service recommendation methods. Note that WSPred, NTF and PLMF is significantly better than UIPCC, PMF and CLUS, indicating that time information plays a key role in service recommendation. Moreover, we can further discover that the more data we provide, the more accurate the recommendation. Since data sparsity in real-world may heavily

Table 2
Comparison between different losses (***RQ3***).

Loss	Response time		Throughput	
	MAE	*RMSE*	*MAE*	*RMSE*
L1 loss	0.631	1.751	3.692	27.403
L2 loss	0.718	1.709	4.808	21.660
Hybrid loss	0.682	1.719	4.206	21.728

restrict the features learning ability, it is worthy embedding context information such as time into service recommendation.

In the throughput comparison, our method is also significantly better than the other six methods. Specifically, time-aware methods (i.e., CLUS, WSPred, NTF, PLMF and RTF) are superior to non-time aware methods (i.e., UPCC and PLMF), which reveal the crucial impact of time information in service recommendation. By comparing the time-aware methods, we may see that PLMF outperforms CLUS, WSPred, which indicates the superiority of Deep Learning. What is more, our RTF is largely superior to PLMF which proves our previous conjecture that GRU is better than LSTM. From the above discussion, we may conclude that RTF is superior to the six most advanced methods, and time information plays a key role in service recommendation.

5.6.2. Impact of integration (RQ2)

In Section 4, we proposed three time-aware service recommendation methods: GTF, PGRU and RTF, and here we will explore the correlations between them. The experimental results performed under four tensor densities (5%∼20%) are shown in Fig. 5.

In response time comparison, the performance of GTF is the worst for all tensor densities, the RTF performs best, and the performance of PGRU is in middle. After analysis, we believe that GTF only remembers short-term historical patterns between users, services and times, which may cause insufficient feature learning. PGRU is capable of remembering long-term dependency patterns so that it is better than GTF. RTF combines the advantages of both GTF and PGRU, thus RTF performs significantly better than GTF and PGRU.

In throughput comparison, RTF achieves best performance, followed by PGRU and GTF. PGRU is better than GTF when the tensor density is only 5%, which indicates that RNNs can exploit more useful information and alleviate the sparseness of actual data. From what has been discussed above, we may conclude that RTF can capture complex dependency patterns between users, services and times by integrating GTF and PGRU.

*5.6.3. Impact of hybrid loss (**RQ3**)*

To prevent the unbalanced fitting problem, the hybrid loss function defined as Eq. (22) is innovatively designed. In this part, we have conducted substantial experiments within 5% tensor density to compare the performance of using the hybrid loss function with using the L1 loss (i.e., absolute error loss) and with using L2 loss (i.e., squared error loss) functions. The experimental results on response-time and throughput are summarized in Table 2.

In the comparison between the L1 loss and the L2 loss functions, the unbalanced fitting problem can be easily found. The L1 loss function wins on *MAE* but loses on *RMSE*, while the L2 loss function wins on *RMSE* but loses on *MAE*. Through theoretical analysis, we believe that using single loss may lead to an unbalanced fitting problem, because L1 loss and L2 loss are highly relevant to *MAE* and *RMSE,* respectively. By comparing our hybrid loss with other losses, the results indicate that the hybrid loss function can solve the unbalanced fitting problem well.

5.6.4. Impact of dropout (RQ4)

As shown in Fig. 6, the experiments are conducted on single time interval with a tensor density of 5%. Since these experiments are independent of the time information, the conclusions drawn from single time interval are the same as those obtained at multiple time intervals. Fig. 6(a) shows the over-fitting phenomenon, from which we can see asymmetric U-shaped curves when evaluating on response time. The U-shaped curves mean that the model over-learns the characteristics of the training data, but lacks generalization ability for the new coming data.

The over-fitting discussed above may lead to a decline in the generalization ability of the model. To prevent these, dropout was used in our experiments. During the learning process, the partial co-dependence of neurons is reduced by zeroing the partial weight or output of the hidden layer, thereby reducing the risk of over-fitting. Fig. 6(b) shows impact of dropout on our model, and it is easy to find that all three curves are stable, which indicates that dropout significantly alleviates over-fitting problem in Fig. 6(a).

6. Conclusion

This paper innovatively proposes a Recurrent Tensor Factorization (RTF) framework based on Deep Learning for time-aware service recommendation, which aims to capture complex dependency patterns between users and services. Under this framework, Generalized Tensor Factorization (GTF) and Personalized Gated Recurrent Unit (PGRU) collaborate to learn the short-term and long-term dependency patterns between users and services. On this basis, RTF can not only memorize the dynamic temporal behavior of users, but also significantly alleviate the problem of data sparsity in real-world.

Our future work will focus on two aspects: (1) use other context information, such as location, reputation, etc., to further validate the effectiveness of our method; (2) use more complex neural network based on LSTM model to further improve the accuracy of services recommendation.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.asoc.2019.105762.

Acknowledgments

This work is partly supported by the National Natural Science Foundation of China (No. 61872002, No. 61873309, No. 61572137 and No. 61728202), and the Natural Science Foundation of Anhui Province of China (No. 1808085MF197). The work of this paper is also supported by Shanghai Innovation Action Plan Project, China under Grant (No. 19510710500, No. 18510760200, and No. 18510732000), and 2017 Research Projects of Shanghai Science and Technology Commission, China under Grant No. 17DZ1101000.

References

[1] L. Shao, J. Zhang, Y. Wei, J. Zhao, B. Xie, H. Mei, Personalized QoS prediction for web services via collaborative filtering, in: IEEE International Conference on Web Services, ICWS, Salt Lake City, USA, July 9–13, 2007, pp. 439–446.

[2] J.S. Breese, D. Heckerman, C.M. Kadie, Empirical analysis of predictive algorithms for collaborative filtering, in: Uncertainty in Artificial Intelligence, UAI, San Francisco, CA, USA, July 24–26, 1998, pp. 43–52.

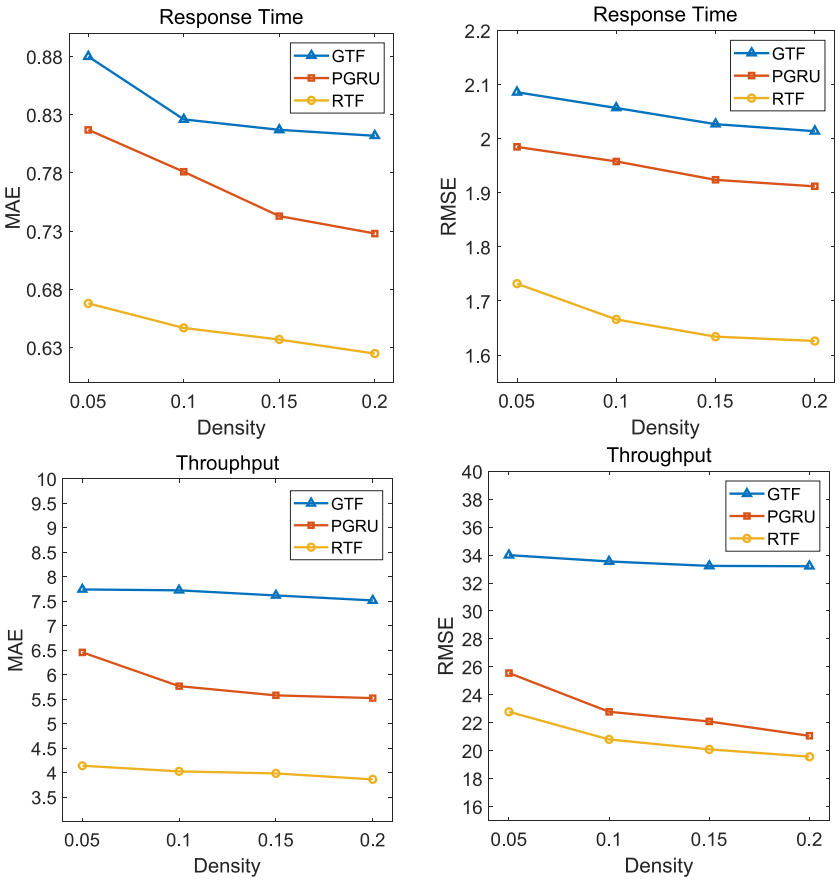

Fig. 5. Comparison between GTF, PGRU and RTF (***RQ3***).

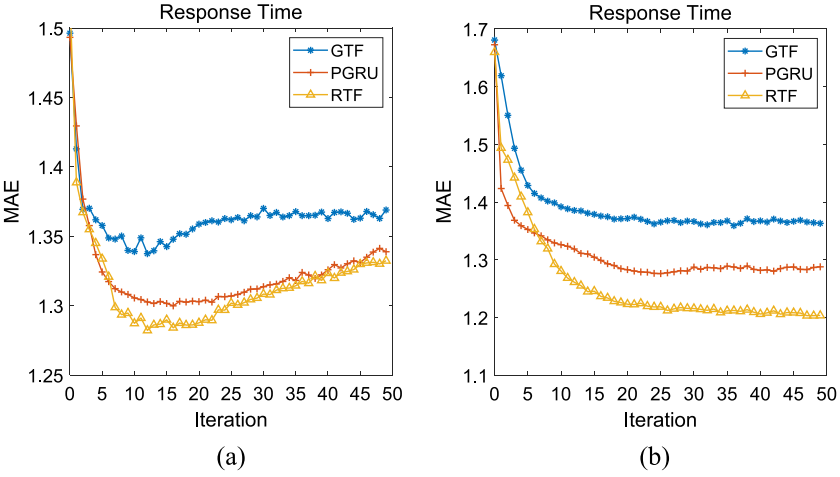

Fig. 6. Impact of Dropout (***RQ4***).

[3] H. Ma, I. King, M.R. Lyu, Effective missing data prediction for collaborative filtering, in: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Amsterdam, The Netherlands, July 23–27, 2007, pp. 39–46.

[4] Z. Tan, L. He, An efficient similarity measure for user-based collaborative filtering recommender systems inspired by the physical resonance principle, IEEE Access 5 (2017) 27211–27228.

[5] X. Zheng, L.D. Xu, S. Chai, QoS recommendation in cloud services, IEEE Access 5 (2017) 5171–5177.

[6] Z. Zheng, X. Wu, Y. Zhang, M.R. Lyu, J. Wang, QoS ranking prediction for cloud services, IEEE Trans. Parallel Distrib. Syst. 24 (6) (2013) 1213–1222.

[7] M.H. Abdi, G.O. Okeyo, R.W. Mwangi, Matrix factorization techniques for context-aware collaborative filtering recommender systems: a survey, Comput. Inf. Sci. 11 (2) (2018) 1–10.

[8] G. Adomavicius, A. Tuzhilin, in: F. Ricci, L. Rokach, B. Shapira (Eds.), Context-Aware Recommender Systems, Recommender Systems Handbook, Springer US, Boston, MA, 2011, pp. 217–253.

[9] K. Chen, H. Mao, X. Shi, Y. Xu, A. Liu, Trust-aware and location-based collaborative filtering for web service QoS prediction, in: IEEE 41st Annual Computer Software and Applications Conference, COMPSAC, Turin, Italy, July 4–8, 2017, pp. 143–148.

[10] X. Fan, Y. Hu, Z. Zheng, Y. Wang, P. Brezillon, W. Chen, CASR-TSE: Context-aware web services recommendation for modeling weighted temporal-spatial effectiveness, IEEE Trans. Serv. Comput. 99 (1939) 1.

[11] Y. Hu, Q. Peng, X. Hu, R. Yang, Time aware and data sparsity tolerant web service recommendation based on improved collaborative filtering, IEEE Trans. Serv. Comput. 8 (5) (2015) 782–794.

[12] J. Liu, M. Tang, Z. Zheng, X. Liu, S. Lyu, Location-aware and personalized collaborative filtering for web service recommendation, IEEE Trans. Serv. Comput. 9 (5) (2016) 86–699.
[13] H.-N. Dai, Z. Zheng, Y. Zhang, Blockchain for internet of things: A survey, IEEE Internet of Things J. (2019) 1–19, http://dx.doi.org/10.1109/JIOT.2019.2920987.
[14] J. Huang, L. Kong, G. Chen, M. Wu, X. Liu, P. Zeng, Towards secure industrial iot: Blockchain system with credit-based consensus mechanism, IEEE Trans. Ind. Inform. 15 (6) (2019) 3680–3689, http://dx.doi.org/10.1109/TII.2019.2903342.
[15] M. Tang, Y. Jiang, J. Liu, Location-aware collaborative filtering for QoS-based service recommendation, in: IEEE International Conference on Web Services, ICWS, Honolulu, HI, USA, June 24–29, 2012, pp. 202–209.
[16] R. Xiong, J. Wang, Z. Li, B. Li, P.C.K. Huang, Personalized LSTM based matrix factorization for online QoS prediction, in: 2018 IEEE International Conference on Web Services, ICWS, July 2–7, 2018, pp. 34–41.
[17] Y. Zhang, Z. Zheng, M.R. Lyu, WSPred: a time-aware personalized QoS prediction framework for web services, in: IEEE 22nd International Symposium on Software Reliability Engineering, ISSRE, Hiroshima, Japan, Nov 29-Dec 2, 2011, pp. 210–219.
[18] X. Wu, B. Cheng, J. Chen, Collaborative filtering service recommendation based on a novel similarity computation method, IEEE Trans. Serv. Comput. 10 (3) (2017) 352–365.
[19] H. Sun, Z. Zheng, J. Chen, M.R. Lyu, Personalized web service recommendation via normal recovery collaborative filtering, IEEE Trans. Serv. Comput. 6 (4) (2013) 573–579.
[20] X. Chen, S. Tang, Z. Lu, J. Wu, Y. Duan, S. Huang, Q. Tang, iDiSC: A new approach to IoT-data-intensive service components deployment in edge-cloud-hybrid system, IEEE Access 7 (2019) 59172–59184.
[21] Z. Lu, N. Wang, J. Wu, M. Qiu, IoTDeM: An IoT Big Data-oriented MapReduce performance prediction extended model in multiple edge clouds, J. Parallel Distrib. Comput. 118 (Part) (2018) 316–327.
[22] Y. Yang, Z. Zheng, X. Niu, M. Tang, Y. Lu, X. Liao, A location-based factorization machine model for web service QoS prediction, IEEE Trans. Serv. Comput. 1 (1) (2018) 2876532.
[23] Z. Wu, Z. Lu, P.C.K. Huang, S. Huang, Y. Tong, Z. Wang, QaMeC: A QoS-driven IoVs application optimizing deployment scheme in multimedia edge clouds, Future Gener. Comput. Syst. 92 (2019) 17–28.
[24] M. Tang, Y. Xu, J. Liu, Z. Zheng, Trust-aware service recommendation via exploiting social networks, Santa Clara, CA, USA, June 28-July 3, 2013, pp. 376–383.
[25] R. Xiong, J. Wang, N. Zhang, Y. Ma, Deep hybrid collaborative filtering for web service recommendation, Expert Syst. Appl. 110 (2018) 191–205.
[26] H. Wu, Z. Zhang, J. Luo, K. Yue, C. Hsu, Multiple attributes QoS prediction via deep neural model with contexts, IEEE Trans. Serv. Comput. 1 (1) (2018) in pre-print.
[27] C. Yu, L. Huang, Time-aware collaborative filtering for QoS-based service recommendation, in: IEEE International Conference on Web Services, Anchorage, AK, USA, 27 June-2 July, 2014, pp. 265–272.
[28] Y. Zhao, Q. Pi, C. Luo, D. Yan, CAPred: A prediction model for timely QoS, in: IEEE International Conference on Web Services, ICWS, New York, USA, June 27-July 2, 2015, pp. 599–606.
[29] C. Yu, L. Huang, A web service QoS prediction approach based on time- and location-aware collaborative filtering, Serv. Orient. Comput. Appl. 10 (2) (2016) 135–149.
[30] S. Deng, S. Jia, J. Chen, Exploring spatial–temporal relations via deep convolutional neural networks for traffic flow prediction with incomplete data, Appl. Soft Comput. 78 (2018) 712–721.
[31] K. Cho, B.v. Merrienboer, Ç. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder–decoder for statistical machine translation, arXiv (2014) 724–1734.
[32] B. Hidasi, A. Karatzoglou, L. Baltrunas, D. Tikk, Session-based recommendations with recurrent neural networks, in: 4th International Conference on Learning Representations, ICLR, San Juan, Puerto Rico, May 2–4, 2016.
[33] D. Dong, X. Zheng, R. Zhang, Y. Wang, Recurrent collaborative filtering for unifying general and sequential recommender, in: Twenty-seventh International Joint Conference on Artificial Intelligence, IJCAI, Stockholm, Sweden, July 13–19, 2018, pp. 3350–3356.
[34] S. Saratha, A.T.W.A.J.M.A.S. Wan, Logic learning in hopfield networks, 2 (3) (2008), arXiv:0804.4075.
[35] A.J.N.C. Graves, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[36] F. ElSaid, J. El Jamiy, A. Higgins, B. Wild, T. Desell, Optimizing long short-term memory recurrent neural networks using ant colony optimization to predict turbine engine vibration, Appl. Soft Comput. 73 (2018) 969–991.
[37] Aggarwal, C. Charu, Recommender Systems, first ed., Springer International Publishing, New York, Cham, 2016.
[38] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, T.S. Chua, Neural collaborative filtering, in: Proceedings of the 26th International Conference on World Wide Web, Perth, Australia, April 03–07, 2017, pp. 173–182.
[39] H. Xue, X. Dai, J. Zhang, S. Huang, Deep matrix factorization models for recommender systems, in: Twenty-Sixth International Joint Conference on Artificial Intelligence, August 19–25, AAAI Press, Melbourne, Australia, 2017, pp. 3203–3209.
[40] P.J. Huber, Robust Estimation of a Location Parameter, Springer, New York, 1992.
[41] T. Tieleman, G. Hinton, Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, COURSERA: Neural Netw. Mach. Learn. 4 (2) (2012) 26–30.
[42] J.C. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Mach. Learn. Res. 12 (2011) 2121–2159.
[43] D.P. Kingma, J. Ba, 2014. Adam: a method for stochastic optimization, in: 3rd International Conference on Learning Representations, ICLR, San Diego, CA, USA, May 7–9, 2015.
[44] I. Sutskever, J. Martens, G.E. Dahl, G. Hinton, On the importance of initialization and momentum in time aware and data sparsity tolerant web service recommendation based on improved collaborative filtering, in: International Conference on Machine Learning, ICML, Atlanta, GA, US, Jun 16 - Jun 21, 2013, pp. 1139–1147.
[45] Z. Zheng, Y. Zhang, M.R. Lyu, Investigating QoS of real-world web services, IEEE Trans. Serv. Comput. 7 (1) (2014) 32–39.
[46] X. Wang, J. Zhu, Z. Zheng, W. Song, Y. Shen, M.R. Lyu, A spatial–temporal QoS prediction approach for time-aware web service recommendation, ACM Trans. Web 10 (1) (2016) 1–25.
[47] Y. Hu, Q. Peng, X. Hu, R. Yang, Time aware and data sparsity tolerant web service recommendation based on improved collaborative filtering, IEEE Trans. Serv. Comput. 8 (5) (2015) 782–794.
[48] S. Li, J. Wen, F. Luo, G. Ranzi, Time-aware QoS prediction for cloud service recommendation based on matrix factorization, IEEE Access 6 (2018) 77716–77724.
[49] Z. Zheng, H. Ma, R. Lyu, I. King, QoS-aware web service recommendation by collaborative filtering, IEEE Trans. Serv. Comput. 4 (2) (2011) 140–152.
[50] R. Salakhutdinov, A. Mnih, Probabilistic matrix factorization, in: advances in Neural Information Processing Systems, NIPS, Vancouver, B.C. Canada, Dec 8–13, 2007, pp. 1257–1264.
[51] M. Silic, G. Delac, S. Srbljic, Prediction of atomic web services reliability based on k-means clustering, in: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, FSE, Saint Petersburg, Russia, August 18–26, 2013, pp. 70–80.
[52] W. Zhang, H. Sun, X. Liu, X. Guo, Temporal QoS-aware web service recommendation via non-negative tensor factorization, in: Proceedings of the 23rd international conference on World Wide Web, WWW, Seoul, Korea, April 7–11, 2014, pp. 585–596.