Research Article

General Collaborative Filtering for Web Service QoS Prediction

Wenming Ma , Rongjie Shan , and Mingming Qi

School of Computer and Control Engineering, Yantai University, Yantai 264005, China

Correspondence should be addressed to Wenming Ma; mwmytu@126.com

Received 30 May 2018; Revised 17 October 2018; Accepted 20 November 2018; Published 3 December 2018

Academic Editor: Ioannis T. Christou

Copyright © 2018 Wenming Ma et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

To avoid the expensive and time-consuming evaluation, collaborative filtering (CF) methods have been widely studied for web service QoS prediction in recent years. Among the various CF techniques, matrix factorization is the most popular one. Much effort has been devoted to improving matrix factorization collaborative filtering. The key idea of matrix factorization is that it assumes the rating matrix is low rank and projects users and services into a shared low-dimensional latent space, making a prediction by using the dot product of a user latent vector and a service latent vector. Unfortunately, unlike the recommender systems, QoS usually takes continuous values with very wide range, and the low rank assumption might incur high bias. Furthermore, when the QoS matrix is extremely sparse, the low rank assumption also incurs high variance. To reduce the bias, we must use more complex assumptions. To reduce the variance, we can adopt complex regularization techniques. In this paper, we proposed a neural network based framework, named GCF (general collaborative filtering), with the dropout regularization, to model the user-service interactions. We conduct our experiments on a large real-world dataset, the QoS values of which are obtained from 339 users on 5825 web services. The comprehensive experimental studies show that our approach offers higher prediction accuracy than the traditional collaborative filtering approaches.

1. Introduction

In recent years, web services have become one of the most popular techniques to build large distributed systems with interoperable machine-to-machine interaction over Internet. By automatic selection, any business organizations' internal applications or services can seamlessly integrate the services of others. Web service selection in terms of functional properties has been extensively studied for many years [1–3], but finding services that satisfy the functional requirements is not enough. Even if web services provide the same functionality, they always differ in QoS (Quality-of-Service) properties, due to their dependence on the circumstances of users and web services. With the increasing web services that have identical or similar functionalities, selection process must also aim at the services best meeting the customers' requirements in terms of QoS. Due to the notable importance of QoS in building reliable and user-friendly distributed applications, QoS-based web services selection has been gaining much attention from both the academia and industry [4–6].

Web service QoS refers to nonfunctional properties, such as availability, price, failure probability, response time, and

throughput. It is crucial to know the exact QoS values to select the best services. Unfortunately, evaluating the QoS values of the real-world web services is difficult and sometimes even impractical. It is time-consuming and resource-consuming for each user to evaluate the QoS values of all the service candidates. Moreover, commercially valuable web services are usually charged, hosted by large business organizations or companies, which cannot be operated by the unpaid users.

An attractive way is that we can predict the QoS value of a service by taking advantage of the past web service usage experiences of a small amount of the other users who have evaluated this service. Therefore, collaborative filtering (CF) techniques have been widely studied in recent years for web service QoS prediction, which already have been adopted for the recommender systems for a decade [7, 8]. In general, there are two types of methods commonly used in collaborative filtering recommender systems, which are referred to as neighborhood based CF and model based CF. Neighborhood based collaborative filtering algorithms include user-based algorithms and item-based algorithms, but such techniques do not work well for sparse rating data. Among various model

based collaborative filtering approaches, matrix factorization method is considered to be state of the art in recommender systems [9–11].

The core idea of matrix factorization is that it assumes the rating matrix is low rank. That means many rows and columns are linearly correlated. It project users and services into a shared latent space, and then the interaction between a user and a service is modeled as the inner product of their latent vectors.

Many researchers have provided various matrix factorization methods for web serviced QoS prediction, simply borrowing the similar models already studied in recommender systems. However, we found that they did not recognize that the QoS often takes continuous values with very wide range, but the user-item rating in recommender system is usually a discrete value belonging to a limited set, such as a integer value ranging from 1 to 5. The low rank assumption would incur high bias for QoS prediction due to the low capacity of matrix factorization models. Furthermore, when QoS value matrix is extremely sparse, matrix factorization models also have high variance due to many kinds of solutions. To reduce the bias, we need more complex models with high capacity. To reduce the variance, we need more complex regularization techniques.

Recently, deep neural networks have yielded remarkable success on many applications, especially on the computer vision, speech recognition, and natural language processing. The exploration of deep learning technologies on recommender systems or QoS prediction has received relatively less attraction. Deep neural networks have very high capacity, but they also have very flexible and practical regularization methods. One of the most popular regularization methods is dropout, which is widely used in real-world applications.

Some recent new novel studies, such as Google's Wide and Deep model and Microsoft's Deep Crossing model, have tried to employ deep learning for recommendation, but they primarily model auxiliary information [12–14]. Other studies, such as neural collaborative filtering (NCF), a novel approach learning user-item interactions using neural style, outperformed the stated-of-the-art methods by large margin [15]. However, NCF models focused on implicit feedback and items ranking, which is not appropriate for QoS prediction. Our work, inspired by the above approaches, presents some novel neural architectures for web service QoS prediction. The main contributions of our work are as follows:

- (i) We formalize the high bias and high variance problem of traditional matrix factorization models (short for MF-HBV problem) for web service QoS prediction.
- (ii) We propose a novel neural network based framework for web service QoS prediction, termed as GCF (general collaborative filtering). The traditional matrix factorization models are special cases of this framework. We also provide four different implementations of GCF, including NLMF (Non-Linear Matrix Factorization), DNN-CF (Deep forward Neural Network Collaborative Filtering), ResNet-CF (Residual

Network Collaborative Filtering), and HyResNet-CF (Hybrid ResNet-CF).

- (iii) We perform experiments for a large real-world dataset, and the extensive experimental investigations are conducted to study the QoS value prediction accuracy of our GCF approaches.

The remainder of this paper is organized as follows: We present our GCF framework in Section 2. Section 3 presents several GCF implementations for web service QoS prediction. Section 4 describes our experiments. Section 5 introduces the related work, and Section 6 concludes this paper.

2. GCF Framework

2.1. QoS Prediction Task. QoS prediction problem is closely related to matrix completion, which is a process of adding entries to a matrix which has some unknown missing values. Figure 1 shows an example of QoS prediction scenario, including 6 users and 6 web services from three different locations. Each user stores response time of several services measured by itself, but does not measure the response time of all the 6 services. For example, the user u_1 in the location 1 only knows the QoS values of service s_4 in the location 3 and services s_5, s_6 in the location 2. All the measured QoS values constitute a sparse matrix, some entries of which are not defined. The QoS task is evaluating the missing values in the matrix by using the observed values.

However, the QoS matrix is always very large and sparse. The missing values take a large proportion of the matrix. Therefore, the traditional missing value estimation methods need specialized adaptations.

In fact, the QoS prediction task is a special regression problem. Usually, a user or a service does not have a concrete feature. Although we can use hand-designed features, such as users' demographic characteristics or services' text descriptions, we only have interaction data most of the time. We must learn abstract features for the users or items.

Generally speaking, when modeling QoS prediction task, we need to solve the following problems:

- (i) We learn the latent vector for each user and service. For the i th user and the j th service, the latent vectors are defined as $\mathbf{u}_i$ and $\mathbf{s}_j$, respectively.
- (ii) We then need to define a prediction function $\hat{r}_{i,j} = \mathcal{F}_{\Theta}(\mathbf{u}_i, \mathbf{s}_j)$.
- (iii) A loss function should be defined, such MSE (Mean Square Error) or MAE (Mean Absolute Error).
- (iv) A learning algorithm is designed to train the function $\mathcal{F}_{\Theta}(\mathbf{u}_i, \mathbf{s}_j)$ using the observed values in the matrix. Note that $\mathbf{u}_i, \mathbf{s}_j, \Theta$ usually are learned jointly in the same algorithm.

To conduct the QoS prediction task, a lot of matrix factorization (MF) based models have been proposed. Section 2.2 will analyze the limitation of matrix factorization and then our GCF framework is outlined in Section 2.3.

FIGURE 1: Web services QoS prediction problem.

2.2. Formulation of MF-HBV Problem. In recommender systems, matrix factorization models use the dot product of two latent vectors as the predicted value, which is defined as

$$\hat{r}_{i,j} = \mathbf{u}_i \cdot \mathbf{s}_j \quad (1)$$

Suppose the size of the true rating matrix $\mathbf{R}$ is $m \times n$. The length of $\mathbf{u}_i$ or $\mathbf{s}_j$ is usually less than m and n . $\mathbf{R}$ can be approximated by the dot of two low-rank matrices: $\mathbf{R} \approx \mathbf{U}\mathbf{S}^T$. The i th row of $\mathbf{U}$ is $\mathbf{u}_i$, and the j th row of $\mathbf{S}$ is $\mathbf{s}_j$. $\mathbf{U}$ and $\mathbf{S}$ are learned from observed entries of $\mathbf{R}$.

This might work well because the user-item rating is usually a discrete value belonging to a small set. However, the QoS often takes continuous values with a very wide range, so the low rank assumption might incur high bias.

Denote $\mathbf{r}_i$ as the total ratings of user i for n services or items. $\mathbf{r}_i$ is the i th row of matrix $\mathbf{R}$. Cosine similarity is a common measure of similarity between two non-zero vectors. The cosine similarity between $\mathbf{r}_p$ and $\mathbf{r}_q$ is defined as

$$\cos(\mathbf{r}_p, \mathbf{r}_q) = \frac{\sum_{j=1}^n r_{p,j} r_{q,j}}{\sqrt{\sum_{j=1}^n r_{p,j}^2} \sqrt{\sum_{j=1}^n r_{q,j}^2}} \quad (2)$$

Suppose the ratings are normalized and standardized. When the ratings for the each service j are very close, $\mathbf{r}_p$ and $\mathbf{r}_q$ can approximately be considered as linearly correlated. Here, we use the expected value of the sum of absolute difference of ratings to measure the credibility of linear correlation (CLC) of the two vectors:

$$CLC = \mathbb{E} \left(\sum_{j=1}^n |r_{p,j} - r_{q,j}| \right) \quad (3)$$

Small value of CLC means the high credibility of linear correlation.

Suppose the ratings are chosen from a set $\mathbf{r} = \{r_1, r_2, \dots, r_k\}$, where $r_1 < r_2 < \dots < r_k$ and $r_{i+1} - r_i = r_i - r_{i-1} = d$. The value of $l = |r_{p,j} - r_{q,j}|$ lies in 0 to $(k-1)d$. If this value is uniformly chosen from d to $(k-1)d$, then the expected value of $l = |r_{p,j} - r_{q,j}|$ is as follows:

$$\mathbb{E}(|r_{p,j} - r_{q,j}|) = d \sum_{l=1}^{k-1} \frac{1}{k} l = d \frac{1}{k} \frac{k(k-1)}{2} = \frac{d(k-1)}{2} \quad (4)$$

Now, we can conclude that

$$\mathbb{E} \left(\sum_{j=1}^n |r_{p,j} - r_{q,j}| \right) = \frac{nd(k-1)}{2} = \frac{n(r_k - r_1)}{2} \quad (5)$$

Therefore CLC is determined by the number of services (or items) and the difference between the maximum and minimum of ratings. Usually, in recommender systems, ratings are discrete values with limited range, such as 1 to 5. The CLC would be $2n$. However, QoS may takes continuous value. The response time always is in a range of 0 to 19 (s), and the throughput is often from 0 to 2000 (kb/s). The CLC is so large in QoS prediction applications that many rows or columns cannot be simply considered as linearly correlated. That is why low rank assumption incurs high bias for QoS prediction.

Furthermore, usually, QoS rating matrix is extremely sparse. Suppose we know the value of the true latent vector $\mathbf{s}_j$

for each service j , and now we want to learn $\mathbf{u}_i$ for a specific user i .

$$\begin{aligned} u_{i,1} * s_{j,1} + u_{i,2} * s_{j,2} + \dots + u_{i,d} * s_{j,d} &= r_{i,j} \\ u_{i,1} * s_{p,1} + u_{i,2} * s_{p,2} + \dots + u_{i,d} * s_{p,d} &= r_{i,p} \\ &\vdots \\ u_{i,1} * s_{q,1} + u_{i,2} * s_{q,2} + \dots + u_{i,d} * s_{q,d} &= r_{i,q} \end{aligned} \quad (6)$$

If the number of rows is less than the number of columns, we would have infinite solutions for $\mathbf{u}_i$. However, in practice, we need to learn both users' latent vectors and services' latent vectors. If the observed ratings are extremely sparse, there are infinite ways of matrix factorization. That is why low rank assumption incurs high variance.

One might consider some regularization methods for matrix factorization. However, this only reduces the variance, but cannot reduce the bias, and sometimes might increase the bias. Therefore, we need more complex models with high capacity to reduce the bias and then use complex regularization technologies to reduce the variance. The next section will introduce the details of our GCF framework, which is more general and flexible and can learn the user-service interactions more accurately.

2.3. General Collaborative Filtering. In fact, the process of learning prediction function can be classified into three categories:

- (i) Predefine features for users and items (u_i, s_j) and only learn the weights Θ .
- (ii) Predefine the weights Θ and learn the features for users and items (u_i, s_j).
- (iii) Learn features and parameters simultaneously.

Generally speaking, the first type of learning process is related to content-based recommender algorithms, while the other two types are related to collaborative filtering recommender algorithms.

The second type of learning process mainly involves matrix factorization models, where the predicted QoS value $\hat{r}_{i,j}$ is usually defined as

$$\hat{r}_{i,j} = \mathbf{u}_i \cdot \mathbf{s}_j = \mathbf{1}^T (\mathbf{u}_i \odot \mathbf{s}_j) \quad (7)$$

where $\odot$ denotes the element-wise product of vectors. In this case, the weights are predefined as a vector $\mathbf{1}$.

Matrix factorization models use the dot product of two vectors as the predicted value. However, the underlying relationship between the QoS value and latent vectors might be too complex to be captured well by using only the dot product.

In this paper we present a *general collaborative filtering* (GCF) framework that uses the third type of learning process. Suppose each user has a unique index value ranging from 1 to $|U|$, and each service has a unique index value ranging from 1 to $|S|$. Here, we use the integer i to represent the index value of

the user u_i and use the integer j to represent the index value of service s_j . Formally, we can say that $i = I_u(u_i)$ and $j = I_s(s_j)$. The prediction function of GCF is as follows:

$$\hat{r}_{i,j} = \mathcal{F}_{\Theta}^{gcf} (\mathbb{T}_k(I_u(u_i)), \mathbb{T}_k(I_s(s_j))) \quad (8)$$

where $\mathbb{T}_k(\cdot)$ is a function of transforming index values to a shared k -dimensional latent space: $\mathbf{u}_i = \mathbb{T}_k(I_u(u_i))$, $\mathbf{s}_j = \mathbb{T}_k(I_s(s_j))$.

We provide three types of definitions for $\mathcal{F}_{\Theta}^{gcf}$:

$$\begin{aligned} \mathcal{F}_{\Theta}^{gcf} (I_u(u_i), I_s(s_j)) &= \mathbf{1}^T (\Phi(\mathbf{u}_i) \odot \Phi(\mathbf{s}_j)) = \mathbf{1}^T \mathbf{h}_{\odot} \\ \mathcal{F}_{\Theta}^{gcf} (I_u(u_i), I_s(s_j)) &= \mathbf{w}^T \left(\Phi \left(\begin{bmatrix} \mathbf{u}_i \\ \mathbf{s}_j \end{bmatrix} \right) \right) + b \\ &= \mathbf{w}^T \mathbf{h}_{\cup} + b \end{aligned} \quad (9)$$

$$\mathcal{F}_{\Theta}^{gcf} (I_u(u_i), I_s(s_j)) = \mathbf{w}^T \left(\begin{bmatrix} \mathbf{h}_{\odot} \\ \mathbf{h}_{\cup} \end{bmatrix} \right) + b$$

where $\mathbf{h}_{\odot}$ and $\mathbf{h}_{\cup}$ are two different transformed interactive features of $\mathbf{u}_i$ and $\mathbf{s}_j$:

$$\begin{aligned} \mathbf{h}_{\odot} &= \Phi(\mathbf{u}_i) \odot \Phi(\mathbf{s}_j) \\ &= [\Phi(\mathbf{u}_i)_1 \Phi(\mathbf{s}_j)_1, \dots, \Phi(\mathbf{u}_i)_d \Phi(\mathbf{s}_j)_d] \\ \mathbf{h}_{\cup} &= \Phi \left(\begin{bmatrix} \mathbf{u}_i \\ \mathbf{s}_j \end{bmatrix} \right) = \left[\Phi \left(\begin{bmatrix} \mathbf{u}_i \\ \mathbf{s}_j \end{bmatrix} \right)_1, \dots, \Phi \left(\begin{bmatrix} \mathbf{u}_i \\ \mathbf{s}_j \end{bmatrix} \right)_d \right] \end{aligned} \quad (10)$$

Note that we can use either $\mathbf{h}_{\odot}$ or $\mathbf{h}_{\cup}$ or both of them in our model. $\mathbf{h}_{\odot}$, $\mathbf{h}_{\cup}$, and $[\mathbf{h}_{\odot}^T \ \mathbf{h}_{\cup}^T]^T$ are termed as *p-Factor*. The traditional matrix factorization method is just a special case of the first type of GCF framework, which only uses $\mathbf{h}_{\odot}$ and an identical feature transformation that $\Phi(\mathbf{u}_i) = \mathbf{u}_i$, $\Phi(\mathbf{s}_j) = \mathbf{s}_j$.

To learn GCF models, we need to solve the following optimization problem:

$$\min_{\mathbf{R}_0} \frac{1}{|\mathbf{R}_0|} \sum_{r_{i,j} \in \mathbf{R}_0} |\mathcal{F}_{\Theta}^{gcf} (I_u(u_i), I_s(s_j)) - r_{i,j}| \wedge \mathbb{C} \quad (11)$$

$\mathbf{R}_0$ is the set of ratings observed in $\mathbf{R}$. Note that " $\wedge \mathbb{C}$ " means the regularization policy we adopt. Theoretically, we can use any kind of regularization, but we mainly use *Dropout* in our models [16, 17].

Now, there are two remaining problems to be solved:

- (i) How to implement the function $\mathbb{T}_k(\cdot)$ to transform the index values to latent factors?
- (ii) How to implement the function $\Phi(\cdot)$ to transform latent factors to interactive features?

The next section will give the implementation details of how to solve these two problems. However, the GCF we proposed in this paper is a general framework, and anyone can design different strategies to solve the two problems above. We believe our GCF framework would inspire more researchers to present various algorithms for different applications.

FIGURE 2: Neural feature embedding.

3. Implementation

In this section, we use neural network architecture to implement the GCF models. We first present a solution of implementing the function $\mathbb{T}_k(\cdot)$ and then elaborate some strategies on how to design $\Phi(\cdot)$. Finally, we present some important algorithms.

3.1. Neural Latent Feature Embedding. In fact, each user or service has a unique identifier in the real-world system. We suppose that each identifier is a unique integer number. If the identifier of a user or a service is equal to i , it refers to the i th user or service.

Now let us use a one-hot vector to represent a user or a service. Suppose we have at most m users and n services. The i th user and the j th can be expressed as

$$\begin{aligned} \mathbf{u}_i^r &= [u_{i,1}^r, u_{i,2}^r, \dots, u_{i,m}^r], \\ \mathbf{s}_j^r &= [s_{j,1}^r, s_{j,2}^r, \dots, s_{j,n}^r] \\ u_{i,l}^r &= \begin{cases} 1, & \text{if } l = i, \ 1 \leq l \leq |\mathbf{u}_i^r| \\ 0, & \text{if } l \neq i, \ 1 \leq l \leq |\mathbf{u}_i^r| \end{cases} \\ s_{j,l}^r &= \begin{cases} 1, & \text{if } l = j, \ 1 \leq l \leq |\mathbf{s}_j^r| \\ 0, & \text{if } l \neq j, \ 1 \leq l \leq |\mathbf{s}_j^r| \end{cases} \end{aligned} \quad (12)$$

where the superscript “r” refers to the “raw” feature. Our neural latent feature embedding is defined as follows:

$$\begin{aligned} \mathbf{u}_i &= \mathbb{T}_k(I_u(u_i)) = \mathbf{W}^u \mathbf{u}_i^r \\ \mathbf{s}_j &= \mathbb{T}_k(I_s(s_j)) = \mathbf{W}^s \mathbf{s}_j^r \end{aligned} \quad (13)$$

where $\mathbf{W}^u$ is a $k \times m$ matrix and $\mathbf{W}^s$ is a $k \times n$ matrix. Expanding the formulas $\mathbf{W}^u \mathbf{u}_i^r$ and $\mathbf{W}^s \mathbf{s}_j^r$, we can see that

$$\begin{aligned} \mathbf{W}^u \mathbf{u}_i^r &= \begin{bmatrix} w_{11}^u & w_{12}^u & \cdots & w_{1m}^u \\ w_{21}^u & w_{22}^u & \cdots & w_{2m}^u \\ \vdots & \vdots & \cdots & \vdots \\ w_{k1}^u & w_{k2}^u & \cdots & w_{km}^u \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} w_{1i}^u \\ w_{2i}^u \\ \vdots \\ w_{ii}^u \\ \vdots \\ w_{ki}^u \end{bmatrix} \\ \mathbf{W}^s \mathbf{s}_j^r &= \begin{bmatrix} w_{11}^s & w_{12}^s & \cdots & w_{1n}^s \\ w_{21}^s & w_{22}^s & \cdots & w_{2n}^s \\ \vdots & \vdots & \cdots & \vdots \\ w_{k1}^s & w_{k2}^s & \cdots & w_{kn}^s \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} w_{1j}^s \\ w_{2j}^s \\ \vdots \\ w_{jj}^s \\ \vdots \\ w_{kj}^s \end{bmatrix} \end{aligned} \quad (14)$$

Therefore, $\mathbf{u}_i$ is the i th column of matrix $\mathbf{W}^u$, and $\mathbf{s}_j$ is the j th column of matrix $\mathbf{W}^s$. Actually, the tradition matrix factorization predicted QoS matrix can be expressed as $\hat{\mathbf{Q}} = (\mathbf{W}^u)^T \mathbf{W}^s$ by using our architecture. $\mathbf{u}_i$ and $\mathbf{s}_j$ are termed as embedding layers in the neural network architecture. Figure 2 shows the architecture of the neural latent feature embedding. Above on embedding layers, we can design various other neural layers.

3.2. Learning High Level Interactive Features. After embedding users and services as latent features, we need to transform the latent features to more complex representations.

Here, we use a series of nonlinear functions to transform $\mathbf{u}_i, \mathbf{s}_j$, or $[\mathbf{u}_i^T \ \mathbf{u}_i^T]^T$

$$\Phi(\cdot) = \phi_d(\cdots \phi_2(\phi_1(\cdot))) \quad (15)$$

FIGURE 3: Crossing residual unit.

TABLE 1: Typical GCF Models.

Model	p-Factors		Transform Functions			Framework Type
	$\mathbf{h}_o$	$\mathbf{h}_{ij}$	$\mathbf{u}_i$	$\mathbf{s}_j$	$[\mathbf{u}_i^T \ \mathbf{s}_j^T]^T$	
UCMF	✓		Identity	Identity		First
NLMF	✓		ReLUs	ReLUs		First
DNN-CF	✓	✓			ReLUs	Second
ResNet-CF	✓	✓			CRUs	Second
HyResNet-CF	✓	✓	ReLUs	ReLUs	CRUs	Third

where ϕ_i can be any kind of simple nonlinear activation function or complex compound function.

For the simple nonlinear activation function, we adopt ReLU [18–20], which is defined as

$$\text{ReLU}(\mathbf{x}) = [h_1, h_2, \dots, h_k]$$

$$h_i = \begin{cases} \mathbf{w}_i^T \mathbf{x} + b, & \text{if } \mathbf{w}_i^T \mathbf{x} + b > 0 \\ 0, & \text{else} \end{cases} \quad (16)$$

Note that we usually add a dropout regularization function for some ϕ .

Suppose $\mathbf{h}_l = \phi_l(\mathbf{h}^{l-1})$; if we use dropout, we need a random vector $\mathbf{r}$, where the length of $\mathbf{r}$ is equal to $\mathbf{h}_l$ and $r_i \sim \text{Bernoulli}(p)$. $1 - p$ is the drop ratio. Now $\mathbf{h}_l$ becomes $\tilde{\mathbf{h}}_l = \mathbf{r} \odot \mathbf{h}_l$

For the complex compound function, we provide a novel crossing residual unit (CRU):

$$\begin{aligned} \mathbf{h}_1^k &= \text{ReLU}(\mathbf{x}_o^{k-1}) \\ \tilde{\mathbf{h}}_1^k &= \text{Dropout}(\mathbf{h}_1^k) \\ \mathbf{h}_2^k &= \mathbf{W}\tilde{\mathbf{h}}_1^k + \mathbf{b} \\ \mathbf{h}_+^k &= \mathbf{h}_2^k + \mathbf{h}_3^{k-1} \\ \mathbf{h}_3^k &= \text{ReLU}(\mathbf{h}_+^k) \\ \tilde{\mathbf{x}}_o^k &= \text{Dropout}(\mathbf{h}_3^k) \end{aligned} \quad (17)$$

Instead of adding $\mathbf{h}_2^k$ to $\mathbf{x}_o^{k-1}$, we add $\mathbf{h}_2^k$ to $\mathbf{h}_3^{k-1}$. Our CRU can reduce the variance when using dropout. Figure 3 gives an intuitive explanation of CRU. If we use l CRUs, $\mathbf{h}_1^l$ in the last CRU is termed as t -Factor, the size of which is important for training.

3.3. Typical GCF Models and Algorithms. Depending on how to choose the p-Factors, transformation functions, and prediction functions, we can get various GCF models with different neural network architectures. Table 1 lists 5 types of typical GCF models, and a general architecture for these different models is shown in Figure 4.

UCMF (Un-Constrained Matrix Factorization) is actually the traditional matrix factorization, and the predicted QoS value is $\hat{r}_{i,j} = \mathbf{1}^T \mathbf{h}_o + 0 = \mathbf{u}_i^T \mathbf{s}_j$. NLMF (Non-Linear Matrix Factorization) is a simple extension of the traditional matrix factorization, which replaces $\mathbf{u}_i$ and $\mathbf{s}_j$ with $\Phi(\mathbf{u}_i)$ and $\Phi(\mathbf{s}_j)$, respectively. The transform functions are a series of ReLUs. UCMF and NLMF belong to the first type of GCF framework. DNN-CF and ResNet-CF belong to the second type of GCF framework. However, DNN-CF uses the ReLUs but ResNet-CF uses the CRUs. HyResNet-CF belongs to the third GCF framework. It uses ReLUs to transform $\mathbf{u}_i$, $\mathbf{s}_j$ and uses CRUs to transform $[\mathbf{u}_i^T \ \mathbf{s}_j^T]^T$. Note that UCMF is just a neural network version of tradition matrix factorization, so GCF models refer to NLMF, CNN-CF, ResNet-CF, and HyResNet-CF in the later discussion, not including UCMF.

Now, we give some important algorithms to implement GCF models. Due to the limited space, we only elaborate the algorithms related to HyResNet-CF, and the algorithms of the other models are similar. We implement the algorithms using Keras framework, so the pseudocodes are similar to Keras codes [21].

Algorithms 1 and 2 show how to build and train the model, respectively. *Embedding* function refers to $\mathbb{T}(\cdot)$, which is implemented according to formula (13). *Concat* function merges two vectors to a long vector. *CRU* function refers to CRU neural unit, which is implemented according to formula (17). *Dense* is a fully connected layer. *Split* function separates users' ids, services ids, and QoS values from the dataset. Given a training set, the training time is mainly determined by the mode size, batch size, learning rate, and the optimization

FIGURE 4: GCF neural network architecture.

function. In this paper, we use the *Adam* as the optimizer, which uses a separated learning rate for each parameter and automatically adapts these learning rates throughout the course of learning.

4. Experimental Results

4.1. Dataset and Evaluation Metrics. We evaluate the proposed methods using the response time values of a publicly large accessible dataset: WS-DREAM dataset#1, obtained from 339 users on 5825 web services [22]. Note that the original dataset is a dense full matrix. We construct training and test sets as follows:

- (i) We first filter out zero values, which are not useful for evaluation, and get the dataset $\mathbf{Q} = \{i, j, r_{i,j} \mid 1 \leq i \leq 329, 1 \leq j \leq 5825, 0 < r_{i,j} \leq 20\}$.
- (ii) We then randomly $k\%$ (sparsity level) of QoS values from $\mathbf{Q}$ as the training set $\mathbf{Q}_R$, and the remaining

values as the test set $\mathbf{Q}_E$. We construct 6 different datasets with different sample densities of 5%, 10%, 15%, 20%, 25%, and 30%.

We adopt two evaluation metrics: Mean Absolute Error (MAE) and Normalized MAE (NMAE). MAE metric is widely employed to measure the QoS prediction quality, which is defined as

$$MAE = \frac{\sum_{(i,j,r_{i,j}) \in \mathbf{Q}_E} |r_{i,j} - \hat{r}_{i,j}|}{|\mathbf{Q}_E|} \quad (18)$$

Since different QoS properties of web services have different value ranges, we also use the Normalized MAE to give a better interpretation of the MAE metric. Here, we define the NMAE as follows:

$$NMAE = \frac{MAE}{\sum_{(i,j,r_{i,j}) \in \mathbf{Q}_E} r_{i,j} / |\mathbf{Q}_E|} \quad (19)$$

Input: m —number of users
 n —number of services
 k —dimension of latent vectors

Output: $\mathcal{M}$ —Model

```

1:  $u = \text{Input}(\text{shape}=(1,), \text{dtype}='int32')$ 
2:  $s = \text{Input}(\text{shape}=(1,), \text{dtype}='int32')$ 
3:  $\mathbf{u}_i = \text{Embedding}(u, k, \text{length}=1)$ 
4:  $\mathbf{s}_j = \text{Embedding}(s, k, \text{length}=1)$ 
5:  $\mathbf{h}_{u,1} = \text{ReLU}(\mathbf{u}_i)$   $\mathbf{h}_{u,1} = \text{Dropout}(\mathbf{h}_{u,1})$ 
6:  $\mathbf{h}_{s,1} = \text{ReLU}(\mathbf{s}_j)$   $\mathbf{h}_{s,1} = \text{Dropout}(\mathbf{h}_{s,1})$ 
7:  $\vdots$ 
8:  $\mathbf{h}_{u,d} = \text{ReLU}(\mathbf{h}_{u,d-1})$   $\mathbf{h}_{u,d} = \text{Dropout}(\mathbf{h}_{u,d-1})$ 
9:  $\mathbf{h}_{s,d} = \text{ReLU}(\mathbf{h}_{s,d-1})$   $\mathbf{h}_{s,d} = \text{Dropout}(\mathbf{h}_{s,d-1})$ 
10:  $\mathbf{h}_{\odot} = \mathbf{h}_{u,d} \odot \mathbf{h}_{s,d}$ 
11:  $\mathbf{h}_{us} = \text{Concat}(\mathbf{u}_i, \mathbf{s}_j)$ 
12:  $\mathbf{h}_{us,1} = \text{CRU}(\mathbf{h}_{us})$   $\mathbf{h}_{us,1} = \text{Dropout}(\mathbf{h}_{us,1})$ 
13:  $\vdots$ 
14:  $\mathbf{h}_{us,l} = \text{CRU}(\mathbf{h}_{us,l-1})$   $\mathbf{h}_{us,l} = \text{Dropout}(\mathbf{h}_{us,l-1})$ 
15:  $\mathbf{h}_{\cup} = \mathbf{h}_{us,l}$ 
16:  $\mathbf{h}_{pred} = \text{Concat}(\mathbf{h}_{\odot}, \mathbf{h}_{\cup})$ 
17:  $\hat{r} = \text{Dense}(1, \text{activation}='linear')(\mathbf{h}_{pred})$ 
18:  $\mathcal{M} = \text{Model}(\text{input}=[u, s], \text{output}=\hat{r})$ 
19: return  $\mathcal{M}$ 

```

ALGORITHM 1: BuildModel(m, n, k).

Input: T —Training set
 m —number of users
 n —number of services
 k —dimension of latent vectors
 r —learning rate
 t —batch size
 L —loss function

Output: $\mathcal{M}$ —Model

```

1:  $\mathcal{M} = \text{BuildModel}(m, n, k)$ 
2:  $\text{opt} = \text{Adam}(\text{lr}=r, \text{beta}_1=0.5)$ 
3:  $\mathcal{M}.compile(\text{optimizer}=\text{opt}, \text{loss}=L)$ 
4: for  $i = 1$  to  $\text{epochs}$  do
5:    $T_{\text{batch}} = \{T_1, T_2, \dots, T_t\} = \text{Sample}(T, t)$ 
6:    $\text{uids}, \text{sids}, \text{qs} = \text{Split}(T_{\text{batch}})$ 
7:    $\mathcal{M}.train\_on\_batch(\text{input}=[\text{uids}, \text{sids}], \text{label}=\text{qs})$ 
8: end
9: return  $\mathcal{M}$ 

```

ALGORITHM 2: TrainModel(T, m, n, k, lr, t, L).

4.2. Performance Comparison. We compare our GCF methods with some traditional collaborative filtering methods: UPCC [23], IPCC [23], UIPCC [24], and UCMF. For UPCC, IPCC, and UIPCC, different trade-off parameters k = 5, 10, 15, 20, 25, 30 (the parameters of top k similar users or services) are tried, and finally we choose k = 10. For UCMF, DNN-CF, NLMF, ResNet-CF, and HyResNet-CF, the sizes of latent factors are also set as 10. For DNN-CF, NLMF, ResNet-CF, and HyResNet-CF, different hidden layers and different hidden unit sizes are tried. Here, the maximum number of

hidden layers is limited to 3. Finally, NLMF uses one hidden layer and the other GCF models use 3 hidden layers. We tested the batch size of [128, 256, 512, 1024], the learning rate of [0.0001, 0.0005, 0.001, 0.005], and the training epoch of [10, 20, 30, 40, 50, 60, 70, 80].

For UCMF, we did not adopt any regularization because we found that there is no help to improve the prediction accuracy. For GCF methods, we adopt dropout regularization. The drop rate is set as 0.5 for the layer of which the size is equal to or larger than 64, and 0.2 for the layer of which the size is smaller than 64.

We conduct 10 experiments for each model and each sparsity level and then average the prediction accuracy values. $\mathbf{s}_{j,l}^r$ is a one-hot vector, the length of which is 5825. Only the j th element is 1. This vector represents the raw feature of a service. The size of weights matrix $\mathbf{W}^s$ is 5825×10 . This matrix is learned jointly with the other parameters of the whole neural network. The output of each hidden layer, $\mathbf{h}$, represents different abstract features learned by the network. The higher level feature is determined by the lower level feature and the corresponding weights matrix $\mathbf{W}$. These matrices are also learned together while training.

The average total training and testing time (from density 5% to 30%) per experiment of each model finally chosen is as follows: UPCC took about 12 minutes, IPCC took about 13 minutes, UIPCC took about 15 minutes, UCMF took about 18 minutes, DNN-CF took about 27 minutes, NLMF took about 19 minutes, ResNet-Cf took about 42 minutes, and HyResNet-CF took about 53 minutes.

The results are reported in Tables 2 and 3 and Figure 5. We can make the following observations:

- (i) With the sparsity level increasing, all the models have much better prediction accuracy (lower MAE and NMAE). When sparsity level is set as 30%, all the models have similar prediction accuracy except UPCC.
- (ii) Our GCF methods outperform the traditional collaborative filtering methods, especially when the QoS rating matrix is extremely sparse. HyResNet-CF has the best prediction performance.
- (iii) DNN-CF, ResNet-CF, and HyResNet-CF are all large networks. Although they get similar prediction performance, we found that DNN-CF has higher variance.
- (iv) UCMF has very high variance when the QoS rating matrix is very sparse, and the variance decreases when the sparsity level becomes larger. However, it still has high bias and has lower prediction performance than the other GCF models.

When the QoS matrix density is set to 5%, the MAE of UCMF is 0.6284, but the MAE of GCF models is 0.5248 (DNN-CF), 0.5176 (NLMF), 0.5154 (ResNet-CF), and 0.5111 (HyResNet-CF), respectively. On the other side, the SD (Standard Deviation) of UCMF is 0.0256, but the SD of GCF models is 0.0078 (DNN-CF), 0.0094 (NLMF), 0.5154 (ResNet-CF), and 0.0070 (HyResNet-CF), respectively. From

TABLE 2: Performance Comparison in terms of MAE.

Methods	Density=5% MAE	Density=10% MAE	Density=15% MAE	Density=20% MAE	Density=25% MAE	Density=30% MAE
UPCC	0.6383±0.0031	0.5562±0.0022	0.5155±0.0014	0.4868±0.0013	0.4670±0.0008	0.4537±0.0012
IPCC	0.6354±0.0021	0.5947±0.0028	0.5096±0.0017	0.4561±0.0014	0.4330±0.0015	0.4155±0.0013
UIPCC	0.6264±0.0021	0.5823±0.0026	0.5010±0.0016	0.4500±0.0013	0.4276±0.0014	0.4109±0.0012
UCMF	0.6284±0.0256	0.5484±0.0318	0.4710±0.0066	0.4447±0.0033	0.4299±0.0061	0.4191±0.0035
DNN - CF	0.5248±0.0078	0.4692±0.0134	0.4427±0.0105	0.4309±0.0116	0.4162±0.0128	0.4159±0.0291
NLMF	0.5176±0.0094	0.4656±0.0036	0.4423±0.0039	0.4287±0.0041	0.4112±0.0037	0.4049±0.0017
ResNet-CF	0.5154±0.0070	0.4642±0.0082	0.4351±0.0077	0.4213±0.0080	0.4089±0.0046	0.3980±0.0032
HyResNet-CF	0.5111±0.0070	0.4548±0.0045	0.4308±0.0042	0.4148±0.0043	0.4028±0.0048	0.3971±0.0031

TABLE 3: Performance Comparison in terms of NMAE.

Methods	Density=5% NMAE	Density=10% NMAE	Density=15% NMAE	Density=20% NMAE	Density=25% NMAE	Density=30% NMAE
UPCC	0.7040±0.0036	0.6125±0.0024	0.5677±0.0018	0.5360±0.0017	0.5143±0.0009	0.4996±0.0011
IPCC	0.7008±0.0026	0.6550±0.0034	0.5612±0.0022	0.5023±0.0019	0.4769±0.0008	0.4575±0.0015
UIPCC	0.6909±0.0026	0.6413±0.0031	0.5517±0.0021	0.4956±0.0018	0.4710±0.0017	0.4525±0.0013
UCMF	0.6929±0.0281	0.6041±0.0351	0.5185±0.0075	0.4894±0.0035	0.4732±0.0066	0.4613±0.0036
DNN - CF	0.5786±0.0086	0.51680.0149	0.4875±0.0117	0.4744±0.0129	0.4585±0.0142	0.4577±0.0317
NLMF	0.5706±0.0104	0.5127±0.0040	0.4870±0.0041	0.4709±0.0043	0.4527±0.0043	0.4455±0.0020
ResNet-CF	0.5683±0.0076	0.5113±0.0090	0.4790±0.0084	0.4638±0.0089	0.4502±0.0049	0.4380±0.0033
HyResNet-CF	0.5637 ±0.0078	0.5008 ±0.0050	0.4744 ±0.0047	0.4570 ±0.0056	0.4436 ±0.0051	0.4364 ±0.0035

FIGURE 5: Performance comparison in terms of MAE and NMAE.

the empirical results, we can see that GCF models have lower variance and lower MAE than UCMF. According to the machine learning basic theory, prediction loss is determined by both the bias and variance. Since GCF models have lower prediction loss and lower variance, they obviously also have lower bias. This verifies our assumption in Section 2.3.

To understand why deep GCF models can get better prediction performance, let us show how MAE changes during the training process with different sparsity levels. Here, we compare UCMF and HyResNet-CF methods. The results are shown in Figure 6, and we can make some observations as follows:

- (i) The training MAEs of UCMF decrease sharply at the early stage, then become steady or increase slightly for a while, and then decrease with increasing the training epochs.
- (ii) The test MAEs of UCMF have a knee point when the sample density is set as 5%, 10%, or 15%. If we set the sparsity level as 15%, after the knee point, the MAEs first increase but decrease after a period of time and finally become steady. However, for the density of 20%, 25%, and 30%, the test MAEs of UCMF first increase for a while and then decrease and finally become steady.
- (iii) The training MAEs of HyResNet-CF decrease sharply at the early stage and then decrease relatively more slowly with increasing the training epochs.
- (iv) The test MAEs of HyResNet-CF becomes steady after a few training epochs but is always lower than the test MAEs of UCMF.

Therefore, HyResNet-CF is more robust and efficient than UCMF. Actually, however we choose the sparsity level; HyResNet-CF can always get a better prediction performance using only a small number of training epochs.

To give a more intuitive explanation of the advantages of deep GCF models, we visualize the service latent factors (s_j) of UCMF, NLMF, and HyResNet-CF, using 5% and 30% sparsity levels. We use t-SNE to map the 10-dimensional service latent factor to the 3-dimensional space. From Figure 7, we can see the following:

- (i) If the training set is larger, i.e., the sparsity level is 30%, each model can capture distinguishable structures.
- (ii) When the sparsity level is 5%, the training set is very small. UCMF learns poor latent factors with disordered structure. However, NLMF and HyResNet-CF can still capture more clear structures.
- (iii) NLMF and HyResNet-CF learn different latent factors structures; both of them can get similar prediction performance.

4.3. Impact of Network Depth. To see how network depth influences the prediction performance, we change the number of CRUs of ResNet-CF model and the size of $\mathbf{h}_1^k$ for each CRU. Here, if we use l CRUs, and the size of t-Factor, i.e., $\mathbf{h}_1^l$, is 8, then the size of $\mathbf{h}_1^{l-1}$ is 16, the size of $\mathbf{h}_1^{l-2}$ is 32, and so on. Therefore, we test the prediction performance, given different l and size of t-Factor. The results are shown in Figures 8 and 9 and Tables 4 and 5.

Due to the extremely large size, models with t-Factor size greater than 8 use at most 7 CRU layers. It is too slow to train the more deeper networks in a limited period of time, but 7 CRU layers are sufficient to verify the impact of network depth. Therefore, Figure 8 and Tables 4 and 5 provide the performance of at most 7 CRUs. However, if the size of t-Factor is equal to 8, we need more CRUs to observe the change of performance. Thus, we draw the prediction performance with respect to the number of CRUs for 1 to 10 in Figure 9 for the model of which the t-Factor size is 8. Note that # t-Factor

FIGURE 6: MAE w.r.t. training iteration.

FIGURE 7: t-SNE visualization of services latent factors in 3D space.

FIGURE 8: MAE and NMAE of ResNet-CF with different layers.

FIGURE 9: MAE and NMAE of ResNet-CF with different layers (t-Factor=8).

in Tables 4 and 5 refers to t-Factor size. We can make some observations as follows:

- (i) When adding the number of CRUs, the performance would become much better, but it would decrease when the number of CRUs exceeds some threshold value. Therefore, we should choose appropriate network depth.
- (ii) It seems that models with smaller t-Factor size can get better performance using deeper architecture. When setting t-Factor size as 8, we get the lowest MAE and NMAE using 7 CRUs.
- (iii) If we use shallower architecture, a larger t-Factor size can get better prediction performance.

5. Related Work

Collaborative filtering (CF) algorithms have been widely used for the recommender systems. To select the best favorite items

for the users, CF usually computes the ratings for how the users are interested in the items and then selects the top k items in term of the ratings. Predicting QoS values are mostly the same as predicting ratings in recommender systems. Users only know partial knowledge about the QoS properties for all the candidate services. We can build the user-service matrix just like the rating matrix and apply collaborative filtering to make prediction [25–27].

Memory based CF fits the models by directly estimating the parameters using some heuristic algorithms. In fact, memory based CF models are actually generalized k-nearest-neighbors (KNN) algorithms [28–31]. Generally, there are two types of memory based CF models: user-based CF and item-based CF. User-based CF utilizes the most similar other users to predict how the user potentially likes a specific item, while item-based CF makes a recommendation according to the user's history experience on similar items. Most of improvement on memory based CF models is how to design an appropriate similarity function for a specific task. Xiaokun

TABLE 4: MAE of ResNet-CF with Different Layers.

#t-F	ResNet-1 MAE	ResNet-2 MAE	ResNet-3 MAE	ResNet-4 MAE	ResNet-5 MAE	ResNet-6 MAE	ResNet-7 MAE
8	0.5579±0.0120	0.5300±0.0111	0.5214±0.0079	0.5210±0.0078	0.5121±0.0118	0.5084±0.0129	0.5016±0.0090
16	0.5509±0.0312	0.5259±0.0147	0.5112±0.0048	0.5084±0.0059	0.5096±0.0083	0.5064±0.0107	0.5075±0.0114
32	0.5304±0.0267	0.5196±0.0057	0.5153±0.0072	0.5066±0.0073	0.5029±0.0068	0.5029±0.0068	0.5114±0.0099
64	0.5263±0.0047	0.5131±0.0132	0.5087±0.0189	0.5079±0.0124	0.5099±0.0084	0.5094±0.0080	0.5201±0.0160
128	0.5242±0.0050	0.5515±0.0119	0.5100±0.0010	0.5075±0.0102	0.5201±0.0082	0.5166±0.0162	0.5273±0.0121

TABLE 5: NMAE of ResNet-CF with Different Layers.

#t-F	ResNet-1 NMAE	ResNet-2 NMAE	ResNet-3 NMAE	ResNet-4 NMAE	ResNet-5 NMAE	ResNet-6 NMAE	ResNet-7 NMAE
8	0.6151±0.0132	0.5843±0.0123	0.5750±0.0086	0.5745±0.0085	0.5647±0.0129	0.5606±0.0142	0.5531±0.0098
16	0.6075±0.0344	0.5798±0.0162	0.5637±0.0053	0.5607±0.0065	0.5619±0.0088	0.5583±0.0118	0.5596±0.0126
32	0.5850±0.0293	0.5728±0.0065	0.5682±0.0079	0.5587±0.0082	0.5546±0.0076	0.5545±0.0077	0.5640±0.0108
64	0.5804±0.0047	0.5658±0.0147	0.5609±0.0208	0.5601±0.0137	0.5621±0.0094	0.5616±0.0080	0.5734±0.0175
128	0.5781±0.0056	0.5640±0.0132	0.5626±0.0121	0.5596±0.0111	0.5734±0.0091	0.5696±0.0179	0.5813±0.0132

Wu et al. computed the similarity between users or between items by comparing the attribute values directly. Huifeng Sun et al. proposed a similarity measure named normal recovery (NR), unifying similarity of the scaled user vectors (or item vectors) in different multidimensional vector spaces. Other studies also provide their own similarity measure based on different assumptions and constraints. Due to lack of ability of making complex assumptions and integrating side information, the memory based CF usually cannot get lower prediction accuracy than the model based CF.

Model based CF, also as a special machine learning algorithm, actually is a generalized regression or classification method, which estimates the coefficients from data. Unlike the typical supervised machine learning problems, there are no obvious raw features for data represented by the rating matrix. Therefore, matrix factorization has become the main technology to handle such data [32–38]. The main difference between traditional matrix factorization approaches is how to represent the latent vectors. Probabilistic Matrix Factorization (PMF) supposes that the rating given a specific user and a specific item obeys a normal Gaussian distribution, and the user latent vector and item latent vector have zero-mean Gaussian distribution [39]. Non-negative Matrix Factorization (NMF) learns the optimal nonnegative latent factors from data, which usually deals with the task of rating prediction on explicit feedback [40]. Generally speaking, non-negative matrix factorization is used to deal with implicit feedback, where the dataset only includes whether the users are interested in some items but does not include whether the users dislike some items. For explicit feedback, the ratings are usually normalized by subtracting the mean, so non-negative matrix factorization is not suitable. Our task is predicting QoS values, which belong to explicit feedback. Furthermore, our GCF framework is a deep neural network architecture. Although we can add a constraint that the output of embedding layer is non-negative, the output of the above layers could be any value again. So the non-negative constraint has little influence on the GCF framework. However, we can easily use dropout to avoid overfitting for the GCF models. Some studies also integrate matrix factorization with memory-based CF algorithms. Even if matrix factorization CF algorithms have the state-of-the-art performance, only capturing linear interaction cannot get further significant improvement.

To efficiently find nonlinear interaction, some recent studies have provided some deep learning based collaborative filtering technologies. Two remarkable examples of such technologies are Google's Wide and Deep [12] and Microsoft's Deep Crossing [13]. Both of them are generalized linear regression/classification models. The interaction between features is represented by deep neural networks, such as MLP or Residual Network. However, they are designed for tasks with a lot of features, and the interaction of them includes not only the user and the item. Neural Collaborative Filtering (NCF) is designed purely for user and item interactions [15]. It creatively combines the linear interaction and nonlinear interaction, by applying the embedding technology and multiplication of embedding latent vectors. However, the above three studies focus on classification task. Our work is

motivated by NCF, but we are focused on regression tasks, which have different evaluation protocols. Our models indeed made an obvious improvement in service QoS prediction.

Another way of modeling nonlinear interaction is using kernel matrix factorization, which is similar to NLMF method in our GCF framework [41, 42]. However, kernel matrix factorization is a shallow architecture, which has limited learning capacity. NLMF is more flexible, which can adjust the network depth to increase or reduce the learning capacity. Furthermore, kernel matrix factorization needs matrix inverse operation for each training iteration, which is hard to train for the large dataset. Finally, we can benefit from the newest regularizer technologies, such as dropout and batch normalization when using NLMF, which are more powerful than L1 or L2 regularizer used in the kernel matrix factorization.

6. Conclusion

Due to the wide range of QoS values, traditional collaborative filtering methods, especially the matrix factorization models, cannot capture the complex structures of user-service interactions. We give some formal description of the MFHBV problem of traditional matrix factorization models and then provide a more general collaborative filtering framework, which is called GCF framework. Traditional matrix factorization is just a special case of GCF framework. Instead of using the dot product of latent factors to make predictions, we use nonlinear transformation functions to get high level interactive feature to increase the capacity of models. At the same time, we use some complex regularization technologies to reduce the variance of models.

There are three types of GCF framework, using different interactive features to predict QoS values. We design 5 types of instances of these three GCF frameworks, using different neural network architectures, including the traditional matrix model. Particularly, we present a novel neural unit, called crossing residual unit (CRU). The generalization power is usually estimated using the prediction performance on the test set. In this paper, we can see that two GCF models, ResNet-CF and HyResNet-CF, get the lowest MAE and NMAE on the test set. Therefore, by using CRUs, models would have better generalization power. We use dropout as the regularization method in GCF models. Finally, we conduct extensive experiments on a real-world dataset. The experimental results show that our GCF models (not including UCMF) outperform the traditional collaborative filtering methods. Furthermore, we give some intuitive explanations on why GCF models have lower test error and why they can capture the user-service interaction better. To verify whether deeper neural networks can get better performance, we design several ResNet-CF architectures. It seems that adding more layers can improve the prediction accuracy but the number of layers cannot exceed some threshold value.

However, the experiments only focus on how to predict response time. The values are from 0 to 19s, and most of the values are around 3s. The future study will conduct more

experiments on QoS dataset, the values of which have much wider range.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work is supported by the National Nature Science Foundation of China (No. 61602399 and No. 61502410) and Shandong Provincial Natural Science Foundation, China (No.ZR2016FB22).

References

- [1] F. Chen, C. Lu, H. Wu, and M. Li, "A semantic similarity measure integrating multiple conceptual relationships for web service discovery," *Expert Systems with Applications*, vol. 67, pp. 19–31, 2017.
- [2] J. Serrano-Guerrero, J. A. Olivas, F. P. Romero, and E. Herrera-Viedma, "Sentiment analysis: A review and comparative analysis of web services," *Information Sciences*, vol. 311, pp. 18–38, 2015.
- [3] J. J. Samper Zapater, D. M. Llidó Escrivá, F. R. Soriano García, and J. J. Martínez Durá, "Semantic web service discovery system for road traffic information services," *Expert Systems with Applications*, vol. 42, no. 8, pp. 3833–3842, 2015.
- [4] N. Temglit, A. Chibani, K. Djouani, and M. A. Nacer, "A Distributed Agent-Based Approach for Optimal QoS Selection in Web of Object Choreography," *IEEE Systems Journal*, 2017.
- [5] M. Wu, X. Xiong, J. Ying, C. Jin, and C. Yu, "QoS-driven global optimization approach for large-scale web services composition," *Journal of Computers*, vol. 6, no. 7, pp. 1452–1460, 2011.
- [6] Z. Zheng, X. Wu, Y. Zhang, M. R. Lyu, and J. Wang, "QoS ranking prediction for cloud services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1213–1222, 2013.
- [7] X. Yang, Y. Guo, Y. Liu, and H. Steck, "A survey of collaborative filtering based social recommender systems," *Computer Communications*, vol. 41, pp. 1–10, 2014.
- [8] J. Wei, J. He, K. Chen, Y. Zhou, and Z. Tang, "Collaborative filtering and deep learning based recommendation system for cold start items," *Expert Systems with Applications*, vol. 69, pp. 1339–1351, 2017.
- [9] X. Luo, M. Zhou, Y. Xia, and Q. Zhu, "An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1273–1284, 2014.
- [10] A. Hernando, J. Bobadilla, and F. Ortega, "A non negative matrix factorization for collaborative filtering recommender systems based on a Bayesian probabilistic model," *Knowledge-Based Systems*, vol. 97, pp. 188–202, 2016.
- [11] P. Pirasteh, D. Hwang, and J. J. Jung, "Exploiting matrix factorization to asymmetric user similarities in recommendation systems," *Knowledge-Based Systems*, vol. 83, no. 1, pp. 51–57, 2015.
- [12] C. Heng-Tze, K. Levent, H. Jeremiah, and T. Shaked, "Wide & Deep Learning for Recommender Systems," in *Proceedings of the in Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pp. 7–20, 2016.
- [13] Y. Shan, T. R. Hoens, J. Jiao, H. Wang, D. Yu, and J. C. Mao, "Deep crossing: Web-scale modeling without manually crafted combinatorial features," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2016*, pp. 255–262, USA, August 2016.
- [14] S. Zhang, L. Yao, and A. Sun, "Deep Learning based Recommender System: A Survey and New Perspectives," *Computer Science: Information Retrieval*, 2017.
- [15] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua, "Neural Collaborative Filtering," in *Proceedings of the the 26th International Conference*, pp. 173–182, Perth, Australia, April 2017.
- [16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [17] S. Wager, S. Wang, and P. Liang, "Dropout training as adaptive regularization," in *Proceedings of the 27th Annual Conference on Neural Information Processing Systems, NIPS 2013*, USA, December 2013.
- [18] W. Shang, J. Chiu, and K. Sohn, "Exploring normalization in deep residual networks with concatenated rectified linear units," in *Proceedings of the 31st AAAI Conference on Artificial Intelligence, AAAI 2017*, pp. 1509–1516, USA, February 2017.
- [19] K. Hara, D. Saito, and H. Shouno, "Analysis of function of rectified linear unit used in deep learning," in *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2015*, Ireland, July 2015.
- [20] P. Wang, R. Ge, X. Xiao, Y. Cai, G. Wang, and F. Zhou, "Rectified-Linear-Unit-Based Deep Learning for Biomedical Multi-label Data," *Interdisciplinary Sciences: Computational Life Sciences*, vol. 9, no. 3, pp. 419–422, 2017.
- [21] F. Chollet, "Keras: the Python deep learning library," <https://keras.io/>.
- [22] "WS-DREAM: A Package of Open Source-Code and Datasets to Benchmark QoS Prediction Approaches of Web Services," <https://github.com/wsdream>.
- [23] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "WSRec: a collaborative filtering based web service recommender system," in *Proceedings of the IEEE International Conference on Web Services (ICWS '09)*, pp. 437–444, IEEE, July 2009.
- [24] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "QoS-aware web service recommendation by collaborative filtering," *IEEE Transactions on Services Computing*, vol. 4, no. 2, pp. 140–152, 2011.
- [25] X. Wu, B. Cheng, and J. Chen, "Collaborative Filtering Service Recommendation Based on a Novel Similarity Computation Method," *IEEE Transactions on Services Computing*, vol. 10, no. 3, pp. 352–365, 2017.
- [26] J. Liu, M. Tang, Z. Zheng, X. Liu, and S. Lyu, "Location-aware and personalized collaborative filtering for web service recommendation," *IEEE Transactions on Services Computing*, vol. 9, no. 5, pp. 686–699, 2016.
- [27] D. Margaritis, C. Vassilakis, and P. Georgiadis, "An integrated framework for adapting WS-BPEL scenario execution using

- QoS and collaborative filtering techniques,” *Science of Computer Programming*, vol. 98, pp. 707–734, 2015.
- [28] A. Bellogín, P. Castells, and I. Cantador, “Neighbor Selection and Weighting in User-Based Collaborative Filtering,” *ACM Transactions on the Web (TWEB)*, vol. 8, no. 2, pp. 1–30, 2014.
 - [29] Z. Jia, Y. Yang, W. Gao, and X. Chen, “User-based collaborative filtering for tourist attraction recommendations,” in *Proceedings of the IEEE International Conference on Computational Intelligence and Communication Technology (CICIT ’15)*, pp. 22–25, February 2015.
 - [30] H. Sun, Z. Zheng, J. Chen, and M. R. Lyu, “Personalized web service recommendation via normal recovery collaborative filtering,” *IEEE Transactions on Services Computing*, vol. 6, no. 4, pp. 573–579, 2013.
 - [31] J. Wu, L. Chen, Y. Feng, Z. Zheng, M. C. Zhou, and Z. Wu, “Predicting quality of service for selection by neighborhood-based collaborative filtering,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 2, pp. 428–439, 2013.
 - [32] Z. Zheng, H. Ma, M. R. Lyu, and I. King, “Collaborative web service QoS prediction via neighborhood integrated matrix factorization,” *IEEE Transactions on Services Computing*, vol. 6, no. 3, pp. 289–299, 2013.
 - [33] J. Zhu, P. He, Z. Zheng, and M. R. Lyu, “Online QoS Prediction for Runtime Service Adaptation via Adaptive Matrix Factorization,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 10, pp. 2911–2924, 2017.
 - [34] R. Zhu, D. Niu, and Z. Li, “Robust web service recommendation via quantile matrix factorization,” in *Proceedings of the 2017 IEEE Conference on Computer Communications, INFOCOM 2017*, USA, May 2017.
 - [35] Y. Zhang, Z. Zheng, and M. R. Lyu, “Exploring latent features for memory-based QoS prediction in cloud computing,” in *Proceedings of the 30th IEEE International Symposium on Reliable Distributed Systems (SRDS ’11)*, pp. 1–10, IEEE, Madrid, Spain, October 2011.
 - [36] D. Yu, Y. Liu, Y. Xu, and Y. Yin, “Personalized QoS prediction for web services using latent factor models,” in *Proceedings of the 11th IEEE International Conference on Services Computing (SCC ’14)*, pp. 107–114, July 2014.
 - [37] Y. Koren, “Factorization meets the neighborhood: a multifaceted collaborative filtering model,” in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD ’08)*, pp. 426–434, New York, NY, USA, August 2008.
 - [38] W. Lo, J. Yin, S. Deng, Y. Li, and Z. Wu, “An Extended Matrix Factorization Approach for QoS Prediction in Service Selection,” in *Proceedings of the 2012 IEEE International Conference on Services Computing (SCC)*, pp. 162–169, Honolulu, HI, USA, June 2012.
 - [39] R. Salakhutdinov and A. Mnih, “Probabilistic Matrix Factorization,” in *Proceedings of Advances in Neural Information Processing Systems*, pp. 1257–1264, 2007.
 - [40] D. D. Lee and H. S. Seung, “Algorithms for non-negative matrix factorization,” in *Advances in Neural Information Processing Systems*, pp. 556–562, 2000.
 - [41] X. Liu, C. Aggarwal, Y.-F. Li, X. Kong, X. Sun, and S. Sathe, “Kernelized matrix factorization for collaborative filtering,” in *Proceedings of the 16th SIAM International Conference on Data Mining 2016, SDM 2016*, pp. 378–386, USA, May 2016.
 - [42] S. Rendle and L. Schmidt-Thieme, “Online-updating regularized kernel matrix factorization models for large-scale recommender systems,” in *Proceedings of The 2008 ACM Conference on Recommender Systems*, pp. 251–258, 2018.

Submit your manuscripts at
www.hindawi.com