A deep learning approach for collaborative prediction of Web Service QoS

Mohammed Ismail Smahi, Fethallah Hadjila, Chouki Tibermacine, Abdelkrim Benamar

Service Oriented Computing and Applications
(will be inserted by the editor)

A deep learning approach for collaborative prediction of Web Service QoS

Mohammed Ismail SMAHI[*,1,2] · **Fethallah HADJILA**[1] · **Chouki TIBERMACINE**[2] · **Abdelkrim BENAMAR**[1]

Received: 3 January 2020

Abstract Web services is the corner stone of many crucial domains, such as cloud computing and the Internet of things. In this context, QoS prediction for Web services is a highly important and challenging issue. In fact, it allows for building value-added processes as compositions and workflows of services. Current QoS prediction approaches, such as collaborative filtering methods, mainly suffer from the problems of data sparsity and cold-start obstacles. In addition, previous studies have not explored in depth the impact of geographical characteristics of services/users and QoS rating on the prediction problem. To address these difficulties, we propose a deep learning-based approach for QoS prediction. The main idea consists of combining a matrix factorization model based on a deep auto-encoder (DAE) and a clustering technique based on the geographical characteristics to improve the prediction effectiveness. The overall method proceeds as follows: First, we cluster the QoS data using a self-organizing map that incorporates the knowledge of geographical neighborhoods; by doing so, we allow for the reduction of the data sparsity while preserving the topology of input data. Besides that, the clustering step effectively handles the cold start problem. Second, for each cluster, we train a DAE that minimizes the squared loss between the ground truth QoS and the predicted one. Third, the missing QoS of a new service is predicted using the trained DAE related to the closest cluster. To evaluate the effectiveness and robustness of our approach, we conducted a comprehensive set of experiments based on a real-world Web service QoS data set. The experimental results showed that our method achieves a better prediction performance compared to several state-of-the-art methods.

Keywords Web Service · QoS Prediction · Deep Autoencoder · Self-Organizing Map.

Mohammed Ismail SMAHI[*]
E-mail: mohamedismail.smahi@univ-tlemcen.dz[*]

Fethallah HADJILA
E-mail: f_hadjila@univ-tlemcen.dz

Chouki TIBERMACINE
E-mail: chouki.tibermacine@lirmm.fr

Abdelkrim BENAMAR
E-mail: a_benamar@univ-tlemcen.dz

[1]LRIT, Computer Science Department, University of Tlemcen, Algeria
[2]LIRMM, University of Montpellier and CNRS, France

1 Introduction

Quality of service (QoS) is an important factor for performing Web service selection and recommendation over the internet [19,48,53]. In this context, QoS prediction constitutes a crucial step in building a recommendation system. In practice, QoS values of services constantly change due to environmental constraints (e.g., IT infrastructure and network load), and this fact makes QoS prediction a challenging task [7]. In addition, we observe that end users may have invoked only a small number of web services; consequently, the user-service QoS data are likely to be sparse [20], which may have a significant impact on the accuracy of QoS prediction.

To address the QoS prediction issue, many existing works have leveraged Collaborative Filtering (CF) [1] to infer the missing data. These approaches are based on the exploration of the historical QoS data of Web services recorded from previous interactions. They typically utilize a user-service QoS matrix to model all historical QoS records. The current CF methods can be divided into two classes: memory-based [6,12,52] and model-based [53,26]. Memory-based approaches consist of two stages: the first one computes the similarity between services (or users) through the use of QoS data, while the second stage computes the missing QoS

using average weighting of the known values of similar services (or users). Model-based approaches learn a set of latent factors from the QoS matrix and make predictions. Matrix Factorization (MF) techniques are the most representative approaches of this class. Despite the advantages and the ease of use of CF methods, most of them do not handle contextual attributes (e.g., geographical information of users/services) to enhance the accuracy of prediction; additionally, they suffer from the cold-start problem [42]. The cold-start problem rises when we want to predict the QoS for new services that have no past QoS records or only few QoS data. The sparsity of the QoS matrix also limits the effectiveness of CF methods and results in a poor accuracy performance. To address the above mentioned issues, we propose an extended CF-based approach that uses both Matrix Factorization (MF) and clustering to handle data sparsity and QoS fluctuations.

This paper is an extension and enhancement of a work originally presented in the European Conference on Service-Oriented and Cloud Computing [36]. The proposed approach extends our previous work through the use of both a DAE and clustering. The contributions of this paper are summarized as follows:

(A) To consolidate the related works, we added new approaches that involve deep learning methods, matrix factorization techniques, and contextual recommendation.
(B) To face the QoS sparsity issue, we perform a clustering of the original QoS matrix and yield a set of sub-matrices that share a subset of contextual characteristics (such as geographic information or service providers). After that, each DAE is trained on a single sub-matrix. The clustering step is realized using a Self-Organizing Map (SOM) [25]. This choice is motivated by the ability of SOM to preserve the topological properties of QoS data (see the motivation Section for more details). The learned neurons of the map are grouped using K-means to create more representative clusters. We notice that the use of SOM allows us to address the cold-start problem (i.e., estimation of QoS for new services or new users). More precisely, we initialize the new service (or new user) with the QoS data of the closest cluster-heads (that share the same contextual attributes or the nearest ones).
(C) To improve the optimization results, the different hyper-parameters of DAEs are fine-tuned through the use of k-fold cross-validation [32].
(D) To reduce the model sensitivity to over-fitting which can harm the final performances, we add a random noise to the input data during the training phase (i.e., we use a denoising autoencoder [39] which is a variant of the standard autoencoder).
(E) To evaluate the effectiveness and the accuracy of the proposed DAE, we perform several experiments by adopting different densities and different hyper-parameters.

The remainder of the paper is structured as follows. We first introduce in Section 2 some background material on both autoencoders and Self Organizing Maps. In Section 3, we present the proposed method for Web service QoS prediction. After that, we show in Section 4 the experiment design, the results and the analysis. In Section 5, we present the related works. Finally in Section 6, we summarize the contribution of the paper and put forward our future work.

2 Background

We introduce the basic background material on both autoencoders and Self Organizing Maps.

2.1 Autoencoders

The autoencoder [33, 18] is an unsupervised neural network. It allows to reproduce the original data from the input layer to the output layer through one or several hidden layers. This unsupervised neural network learns a representation of the inputs that yields the least deformation. Typically, the mechanism consists of the encoding data from the input layer until a central hidden layer, called latent factors layer, through one or more hidden layers. After that, the decoding of the central layer data is performed by a set of decoding layers (having the same size as in the encoding layers).

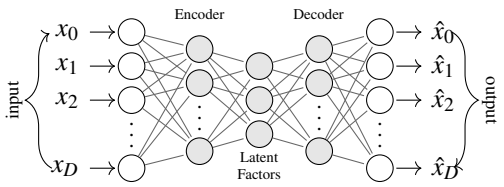

Fig. 1 Deep Autoencoder architecture.

As can be seen in Figure 1, the autoencoder's architecture consists of three parts: the encoder, code (or latent factors) and decoder. The latent factors part represents a single layer of artificial neural network (ANN) where the size should be less than the input layer (which is denoted by K). This hidden layer must be compact and meaningful. The encoder part is represented by a fully connected feed-forward non recurrent ANN. It can be viewed as an *encoding function* F_e that takes a vector $x \in \mathbb{R}^D$ as input data and maps it to a hidden representation $z \in \mathbb{R}^K$ ($z = F_e(x)$). Finally, the decoder part is the reverse mapping of the encoder operation. It is a fully connected feed-forward non recurrent ANN. It is considered as a decoding function F_d that takes the resulting latent representation z then maps it back to a reconstructed vector $\hat{x} \in \mathbb{R}^D$ ($\hat{x} = F_d(z)$). Thus, the output will be:

$$\hat{x} = F_d(F_e(x)) \tag{1}$$

If we assume a basic version of the autoencoder (which uses one hidden layer), the encoding decoding functions will be defined as:

$$\hat{x} = \sigma'(W'z + b') \text{ with } z = \sigma(Wx + b) \quad (2)$$

such that: σ and σ' are transfer functions which can be linear or non-linear[1]. W and W' are two weight matrices having the dimensions (K, D) and (D, K) respectively. b and b' are the offset vectors with K and D dimensions respectively.

In terms of learning, the autoencoder is trained to minimize the dissimilarity function (or the reconstruction error)

$$\underset{W,W',b,b'}{\operatorname{argmin}} \sum_{i=1}^{n} \delta(x_i, F_d(F_e(x_i))) \quad (3)$$

where δ is a dissimilarity function, such as the square loss or the cross entropy loss, x_i is an example that belongs to the learning dataset and n is the size of the dataset.

2.2 Denoising Autoencoders

The denoising autoencoder [39, 15] is a variant version of the basic autoencoder presented above. This kind of autoencoder aims to tackle the noise problem. The goal is to reconstruct the original input data using a corrupted version of it. Note that several corruption types can be applied on the input data, such as: (a) Gaussian noise: where an isotropic Gaussian noise is added to a subset of the input x. (b) Masking noise: A fraction v of the original input x is randomly forced to zero. (c) Salt-and-pepper noise: A fraction v of x is randomly set to their minimum or maximum possible value.

In terms of learning, the denoising autoencoder is still trained to minimize the same reconstruction error between the cleaned input and its reconstruction:

$$\underset{W,W',b,b'}{\operatorname{argmin}} \sum_{i=1}^{n} \delta(x_i, F_d(F_e(x_i'))) \quad (4)$$

where x' is a copy of x that has been corrupted by some kind of noise cited above. In this case, the denoising training forces the encoding function F_e and decoding function F_d to implicitly learn the structure of the original input data [3].

2.3 Self-Organizing Map

Self-Organizing Map (SOM) or Kohonen feature map [25] is a kind of unsupervised neural network based on a competitive learning paradigm. This method is usually used in performing tasks such as clustering, outlier detection and dimensionality reduction [24]. The objective of this network is to reduce a high-dimensional input data space into a low-dimensional map of neurons (usually two dimensions) through the use of self-organizing neural networks while preserving their topological structure.

[1] The autoencoder is called linear if the transfer functions are linear, otherwise it is non-linear.

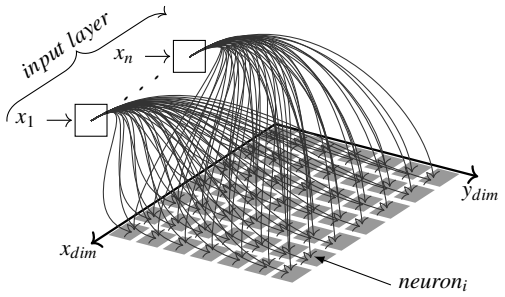

Fig. 2 Kohonen feature map network. (The connections between the input and output layers indicate the relationships between the input and output vectors.)

2.3.1 Self-Organizing Map principles

In terms of architecture (see Figure 2), the network is composed of two layers: the input layer and the map layer; the layers are fully connected. The map layer comprises a set of neurons $M = \{n_1, \ldots, n_k\}$ arranged in a two l-dimensional map of neurons (the map is usually represented graphically as a rectangle with k neurons ($k = x_{dim} \times y_{dim}$)). Each neuron has an associated weight vector W known as the code-book. The code-book describes the typical data profile at a given training step t, where T is the number of training time step instances[2]: $W(t) = \{\overrightarrow{w}_1(t), \ldots, \overrightarrow{w}_k(t)\} \;\; t \in \{1, \ldots, T\}$.

The competitive learning process attempts to construct a nonlinear topology that ensures the mapping of the input vectors $X = \{\overrightarrow{x}(t) | t \in \{1, \ldots, T\}\}$, into a set of neurons M on the map.

In terms of learning, the SOM Algorithm consists of two steps, a competitive and an update step. In the first one, each data vector $\overrightarrow{x}(t)$, for a particular training instance t, is mapped to its best matching neuron on the map : $bm(\overrightarrow{x}(t)) = n_i \in M$. The winning neuron is denoted as:

$$n_i(\overrightarrow{x}(t)) = \underset{j \in 1,\ldots,k}{\operatorname{argmin}} \left\| \overrightarrow{x}(t) - \overrightarrow{w}_j(t) \right\| \quad (5)$$

where each neuron n_i codes the subset of input data space, whose elements are closer to n_i.

In the update step, the weight vectors of both the best match neuron n_i and its whole neighborhood neurons $\Gamma(n_i)$ are adjusted toward the presented input vector $\overrightarrow{x}(t)$ using the following equation: $w_j(t+1) = w_j(t) + \eta(t) \cdot h_{i,j}(t) \cdot (x(t) - w_j(t))$ where $j \in \Gamma(n_i)$, $0 < \eta < 1$ is the learning rate, and $h_{i,j}$ is the neighborhood function. It is important to mention that the neighborhood of the wining neuron (n_i) is progressively reduced until it reaches a set of size one.

A frequently used neighborhood function is the Gaussian:

$$h_{i,j}(t) = \exp\left(-\frac{d(i,j)^2}{2\sigma^2(t)}\right) \quad (6)$$

[2] SOM training stops when the weight vectors are stabilized or the maximum number of iterations is reached.

where $d(i,j)$ is the distance between the neuron n_i and neuron n_j on the map and the radius $\sigma(t) = \sigma_0 \cdot \exp\left(\frac{t}{T_{max}}\right)$ decreases after each iteration to restrict the area of the neighborhood.

2.3.2 Batch learning

A batch algorithm is a useful variant of SOM learning process presented above, which is widely used in parallel implementations [43]. If the whole training data is available beforehand, SOM has a batch formulation of update of all weights:

$$w_j(T) = \frac{\sum_{t'=1}^{T} h_{i,j}(t')x(t')}{\sum_{t'=1}^{T} h_{i,j}(t')} \quad (7)$$

It means that the best matching neurons for all input vectors are selected and hereafter their weights and the weights of their respective neighbors are updated (according to Equation 6).

2.3.3 Initialization Methods

The results and the performances of SOM method are strongly influenced by the initialization of neurons' weights. To this end, two types of initializations can be used. The first one consists of a *random initialization* of the code-book. In this method, random weight vectors are selected from the data points. Technically this method is easy to implement, but its main weakness lies in its high running time and its non deterministic side-effects on results. To tackle the previous problem, a *linear initialization* technique can be used. The initial values of code-book are selected from the subspace spanned by the Principal Component Analysis (PCA) algorithm [22]. The behavior of SOM method using PCA initialization converges more rapidly towards a better clustering and ensures a deterministic clustering result [2].

3 Proposed Method

First, we introduce the architecture model of the proposed approach. Then, we motivate the choices made in the design of this architecture model. At last, we present in depth how QoS prediction is made thanks to this model.

3.1 Architecture model

Figure 3 illustrates the architecture model of the proposed approach. First, we assume that there are m users and n Web services in our system. QoS values are collected from user's invocation of Web services. We use various sources to get this data (such as social networks, a monitoring system, or direct feedbacks). The collected values constitute a $m \times n$ rating matrix $M = \{r_{u,s}\}_{m \times n}$. We assume that QoS matrix contains missing values because it is impossible to invoke all services by all users. In order to predict the missing values (first challenge), we propose an approach that is composed of the following three stages:

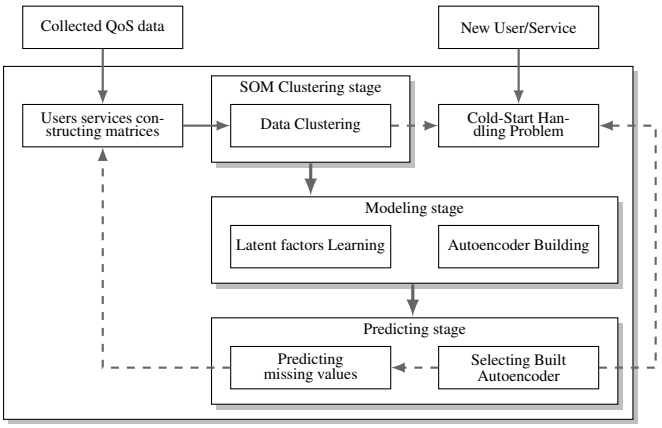

Fig. 3 Overview of the prediction system architecture

1. First, we cluster the initial QoS rating matrix according to the geographical characteristics. More specifically, we will perform a clustering based on SOM Algorithm. As mentioned previously, this method allows us to create clusters whose members are closer in terms of contextual attributes. This step aims to produce a set of clusters that share the same contextual characteristics. Additionally, it allows to decrease the sparsity of the QoS matrix (i.e., the QoS matrix of each cluster is less sparse in comparison to the initial QoS matrix) by considering the top-k clusters.
2. Second, we perform the learning of latent factors for each cluster by leveraging a deep denoising autoencoder. During our experiments, we used a deep autoencoder with three hidden layers (this choice is explained in the experimental study). We assume that all the clusters are trained with the same number of hidden layers.
3. Third, the learned deep autoencoder predicts missing QoS values and stores them in the initial dataset.

The second challenge that interests us is the cold-start problem. Based on the results of the Data Clustering and Autoencoder Building phases, this task describes the arrangements to ensure the best solution for this problem (later in this section, we will describe the detailed mechanism).

3.2 Motivations behind architecture design choices

3.2.1 Autoencoder

Our proposition uses a DAE that allows for predicting QoS values of services. This choice is mainly motivated by the

high accuracy achieved by matrix factorization methods, such as autoencoders [27]. According to [26,47], matrix factorization techniques are more accurate (in terms of prediction performance) than the remaining CF methods. In addition, the flexibility of DAE (e.g., the number of layers and the activation functions) is leveraged to ensure the best performance.
Indeed, it is possible to vary the activation functions of each layer (e.g, $relu$, $softplus$, $leaky\text{-}relu$, ELU) in order to maximize the prediction accuracy.

3.2.2 SOM

To boost the accuracy of our prediction model, we have to address the sparsity issue of the QoS matrix. A higher sparsity results in a lower quality of prediction. To handle this situation, we cluster the QoS data into groups using the SOM method.
We have chosen this method for different reasons, including the following:

- The SOM method has the ability to preserve the density and the topological structure (the form) of the original data;
- SOM-based clustering allows us to address the cold-start problem. More specifically, if we handle a new service (or user) (without QoS values), we first compare its contextual attributes with the context of the cluster-heads; then we initialize the QoS of the new service (or user) with the QoS values of the closest cluster-head;
- SOM network ensures a more faithful representation (in terms of contextual attributes such as country, provider and autonomous systems) of the services (or users), that is, the services (or users) having similar contextual attributes are more likely to belong to the same cluster;
- Their ability to preserve the neighborhood relationships (i.e, two adjacent services/users are represented by two adjacent cluster-heads, and two neighboring cluster-heads share a part of their geographical characteristics while the other properties are different).

According to [5,8,38], QoS values are directly influenced by users and services geographical characteristics. The more the services share the same geographical characteristics the more the QoS values are similar.
In practice, the geographical characteristics of Web services can be categorized into three different groups: *Provider group* ($\mathscr{P}$), *AS*[3] *group* ($\mathscr{A}$) and *Country group* ($\mathscr{C}$). One group is a subset of another (i.e, $\mathscr{P}(i) \subseteq \mathscr{A}(i) \subseteq (i)$, where $i \in \{service, user\}$). This implies that the more the group is restricted, the more the QoS values are likely to be similar.

[3] An AS is either a single network or a group of networks that is controlled by a common network administrator (or group of administrators) on behalf of a single administrative entity (university, a business enterprise, etc.)

As we will see in the experimental evaluation Section, an empirical analysis is conducted to demonstrate the influence of geographical characteristics on the clustering results of SOM.

3.3 Collaborative QoS prediction

The first algorithm (Algorithm 1) enables us to cluster the input data (services S or users U) according to the batch version of SOM principle. Firstly (line 1), we initialize the weight vector for all neurons on the map (code-book initialization). To this end, we use a personalized initialization mechanism to ensure a deterministic behavior and to preserve the neighborhood relationships (more details are given in Section 4.3.3). After that (from line 3 to 5), we determine for each input its winner neuron (BMU) on the map. In next steps (from line 6 to 12), we adjust the weight vectors of all BMUs and their neighborhoods according to Equation 7. The process is repeated as many times as necessary ($epoch$ times as specified in line 2). At the end, we create the final clusters by processing the code-book with a k-means Algorithm [16].

Algorithm 1: Data Clustering According to Batch SOM Algorithm

Inputs : $S = \{s_1, \cdots, s_n\}$, ▷ set of n service vectors
Data : $W = \{w_1, \cdots, w_k\}$ ▷ set of services weight vectors

1 Initialize all $w \in W$
2 **for** $epoch : 1 \rightarrow N_{epochs}$ **do**
3 **foreach** $s \in S$ **do**
4 compute winning node $n(s) = \underset{j \in 1,\ldots,k}{\operatorname{argmin}} \|s - w_j\|$
5 **end**
6 **for** $j : 1 \rightarrow k$ **do**
7 accumulate numerator and denominator in equation 7
8 **end**
9 **foreach** $w \in W$ **do**
10 update w according to equation 7
11 **end**
12 **end**
13 max = 8
14 $< C_1, \cdots, C_{max} >= KmeansCluster(W, max)$ ▷ postprocess the code-books with k-means algorithm
15 **return** $< C_1, \cdots, C_{max} >$ ▷ list of services clusters

In the second algorithm (Algorithm 2), we learn a model that achieves the highest QoS prediction performance. The goal is to infer the best learning model for our deep autoencoder. This algorithm builds a deep autoencoder for each cluster C_i. In line 3, we split the current cluster C_i data into $k\text{-}fold$ folders. Each folder represents a given percentage ($density\%$) of the available cluster data. This parameter represents the ratio between the training dataset and the entire dataset. For example, if density = 20% then the training set represents 20% of all cluster data and the validation set represents the remaining 80% of QoS data. After that in line 7, we learn a deep autoencoder for each folder and with respect

to all training sets ($\forall S \in T$). The goal of this step is to minimize the squared error between the model prediction and the desired values. It is important to mention that the training is controlled by the *noise* parameter. If this parameter is null (i.e, no noise), we use the deep autoencoder architecture. Otherwise, the selected architecture of the model is a denoising autoencoder. In line 8, we calculate the deep autoencoder error achieved on the validation dataset. In line 11, the $k\text{-}folder$ results are averaged to produce a single validation error (*mve*). Finally, we return the best training model as well as the mean validation error.

Algorithm 2: Autoencoder k-fold cross-validation

Inputs : $C = \{C_1, \cdots, C_{max}\}$, *density*, *noise*, $k\text{-}fold$

1 **for** $i \rightarrow max$ **do**
2 $\quad T = C_i$ ▷ training set
3 $\quad < folder_1, ..., folder_{k\text{-}fold} >= Partition(C_i,\ density,\ k\text{-}fold)$
4 $\quad$ **for** $k : 1 \rightarrow k\text{-}fold$ **do**
5 $\quad\quad T = T - folder_k$
6 $\quad\quad V = folder_k$ ▷ validation set
7 $\quad\quad Model^k = \underset{\forall S_m \in T}{\operatorname{argmin}} \left(\sqrt{\frac{1}{|T|} \sum_{m=1}^{|T|} |DAE(S_m, noise, |C_i|) - S_m|^2} \right)$
8 $\quad\quad err^k = \sqrt{\frac{1}{|V|} \sum_{m=1}^{|V|} |Model^k(S_m) - S_m|^2}$
9 $\quad$ **end**
10 $\quad Model^{k^\star} = \underset{\forall k \in k\text{-}fold}{\operatorname{argmin}} (err^k)$ ▷ get the k-th best model in term of err_k
11 $\quad mve = \frac{1}{k\text{-}fold} \sum_{k=1}^{k\text{-}fold} err_k$
12 $\quad \Theta[i] =< Model^{k^\star}, mve >$
13 **end**
14 **return** Θ

In the last algorithm (Algorithm 3), we address the cold-start problem. To solve that, and taking into consideration that QoS values are highly influenced by geographical characteristics (as discussed in the end of Section 3.2.2), we compute the initial QoS values $\mathscr{V}(s) = \{qos_1, \cdots, qos_m\}$ for each new service according to its geographical characteristics. Technically, we check if the actual provider of the new service ($\mathscr{P}(s)$) belongs to the initial set of providers (of our dataset) (line 1). If so, in line 2, we compute the representativeness rate of the actual provider for each service cluster ($\alpha_{c,s}$, $\forall c \in C^S$). Afterward, we average that with respect to all cluster-heads ($\mathscr{H}(c)$, $\forall c \in C^S$). If not, we check if the AS number of the new service ($\mathscr{A}(s)$) is in our initial set of AS numbers (line 3). If this is the case, we calculate the representativeness rate of the actual AS number for each services cluster, and we average the initial QoS values $\mathscr{V}(s)$ with respect to all cluster-heads.
Otherwise, we proceed to compute the representativeness rate of the service country ($\mathscr{C}(s)$) and we average its initial QoS values. In case where the geographical characteristics of the new service are not available, we search the provider having the nearest service (as mentioned in line 8), where $\mathscr{D}$ denotes a geodesic distance function, and we compute its QoS values as mentioned in line 9. This vector ($\mathscr{V}(s)$) is then provided as an input to all the trained DAEs. Finally, we average the prediction results of the initial QoS value for the new service (line 11). Note that the same algorithm is used for a new user ($V(u) = \{qos_1, \cdots, qos_n\}$), with the appropriate changes as well.

Algorithm 3: cold-start Algorithm

Inputs : s ▷ new Service
$C = \{C_1, \cdots, C_{max}\}$ ▷ list of services clusters
$\mathscr{H}(C) = \{\mathscr{H}(C_1), \cdots, \mathscr{H}(C_{max})\}$ ▷ list of cluster-heads
Θ ▷ best training models
Data : $\mathscr{P}$, $\mathscr{A}$, $\mathscr{C}$ ▷ List of Providers, ASNs and Countries from dataset

1 **if** $(\mathscr{P}(s) \in \mathscr{P})$ **then**
2 $\quad \mathscr{V}(s) = \frac{1}{\sum_{c \in C} \alpha_{c,s}} \sum_{c \in C} \alpha_{c,s} \mathscr{H}(c)$ with
$\quad\quad \alpha_{c,s} = \frac{1}{|c|} \sum_{j=1}^{|c|} \begin{cases} 1, & \text{if } \mathscr{P}(s) = \mathscr{P}(s_j) \\ 0, & \text{otherwise} \end{cases}$
3 **else if** $(\mathscr{A}(s) \in \mathscr{A})$ **then**
4 $\quad \mathscr{V}(s) = \frac{1}{\sum_{c \in C} \alpha_{c,s}} \sum_{c \in C} \alpha_{c,s} \mathscr{H}(c)$ with
$\quad\quad \alpha_{c,s} = \frac{1}{|c|} \sum_{j=1}^{|c|} \begin{cases} 1, & \text{if } \mathscr{A}(s) = \mathscr{A}(s_j) \\ 0, & \text{otherwise} \end{cases}$
5 **else if** $(\mathscr{C}(s) \in \mathscr{C})$ **then**
6 $\quad \mathscr{V}(s) = \frac{1}{\sum_{c \in C} \alpha_{c,s}} \sum_{c \in C} \alpha_{c,s} \mathscr{H}(c)$ with
$\quad\quad \alpha_{c,s} = \frac{1}{|c|} \sum_{j=1}^{|c|} \begin{cases} 1, & \text{if } \mathscr{C}(s) = \mathscr{C}(s_j) \\ 0, & \text{otherwise} \end{cases}$
7 **else**
8 $\quad \mathscr{P}(s) = \mathscr{P}(\underset{\forall s_i \in S}{\operatorname{argmin}}(\mathscr{D}(s, s_i)))$ ▷ compute the provider of nearest service
9 $\quad \mathscr{V}(s) = \frac{1}{\sum_{c \in C} \alpha_{c,s}} \sum_{c \in C} \alpha_{c,s} \mathscr{H}(c)$ with
$\quad\quad \alpha_{c,s} = \frac{1}{|c|} \sum_{j=1}^{|c|} \begin{cases} 1, & \text{if } \mathscr{P}(s) = \mathscr{P}(s_j) \\ 0, & \text{otherwise} \end{cases}$
10 **end**
11 $\mathscr{V}(s) = \frac{1}{|\Theta|} \sum_{Model^\star \in \Theta} Model^\star(\mathscr{V}(s))$ ▷ average prediction of the initial QoS values
12 **return** $\mathscr{V}(s)$ ▷ service initial vector

Figure 4 shows an example that explains the principles of the proposed approach: The subplot (A) shows a l by k grid that covers the different clusters of the SOM map; Each line of the map represents a country and involves k neurons, and each neuron is a subset of services. The subplot (B) shows the initialization process of the neurons' weights of a given country, the neurons are assigned in a self-organized mode. The subplot (C) shows the training phase of the different DAEs that are related to the learned clusters. The subplot (D) details the cold start scenario.

4 Experimental Evaluation

In this section, we conduct experiments to evaluate the performance of the proposed approach and its different variants (deep autoencoder, clustered deep autoencoder and Top-k

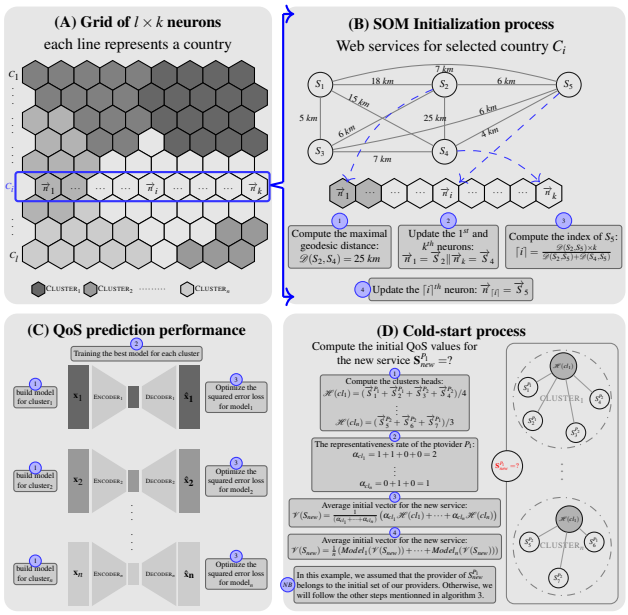

Fig. 4 Illustrative example describing the principles of the proposed approach

clustered deep autoencoder). The objective is to answer the following questions: (1) How sensible are our proposed models to the hyper-parameters? (2) What is the impact of the geographical characteristics on the SOM clustering? (3) How do our approach compare to the state-of-the-art methods under different scenarios?

4.1 Experimental setup

The experiments were conducted on the MESO@LR-Platform of the University of Montpelier, France. To this end, we used 4 nodes (14 cores) with 128 GB of RAM. All the learning programs were implemented in Tensorflow Python. Note that, for the training of the SOM algorithm, we turn to a high-performance implementation called Somoclu[4], which is considered as a massively parallel tool for training a batch formulation of self-organizing maps on large datasets [43].

4.1.1 Data collection

To evaluate the proposed approach, we conducted experiments on a large-scale real-world Web service QoS repository named WS-Dream, released by [54]. This repository contains two datasets. The first one contains 1 974 675 invocations from 339 users on 5825 Web services. The second one contains about 30 million Web service invocation records collected from invocation of 4500 services by 142 users at 64-time intervals. Each time interval t takes 15 minutes. The innovation, on both datasets, considers response time and throughput criteria ($c \in \{rt, th\}$). Table 1 summarizes some characteristics of WS-Dream repository data.

Table 1 Information details of ws-dream repository (RT : response Time, TP: Throughput Time)

Statistics	First Dataset		Second Dataset	
	RT	TP	RT	TP
Scale	0-20 s	0-20 s	0-20 s	0-20 s
Mean values	0.908 s	47.561 s	3.165 s	9.608 s
Users	339	339	142	142
Web services	5826	5826	4500	4500
Times slices	1	1	64	64
All values	1 874 177	1 831 592	30 286 687	30 286 687
Mis. values	100 837	143 422	10 609 313	10 609 313
% mis. values	5.10%	7.26%	25.94%	25.94%

Note that the dataset contains about 26% of missing values, which is a quite high date sparsity rate. This problem specific to this dataset may pose the risk to having an invalid clustering (a cluster with services with null values). In order to reduce this risk, we considered the following two rules for a given service invocation QoS matrix $M_{c,t} = \{QoS_{u,s}\}_{m \times n}$ (for a selected criterion c and at given time slot t):

1. For each $QoS_{s,u}^{c,t}$ value for a given invocation matrix $M_{c,t}$, if this value is invalid, then we replace it by the average of the valid $QoS_{s,u}^{c,t'}$ values on all the previous matrices time slot: $QoS_{s,u}^{c,t} = \frac{1}{t-1} \sum_{t'=1}^{t-1} QoS_{s,u}^{c,t'}$
2. For each $QoS_{s,u}^{c,t}$ value for a given matrix invocation $M_{c,t}$, if this value is invalid and all its previous values (regarding time slots) are invalid, then $QoS_{s,u}^{c,t} = 0$.

Table 2 The results of different improvements on dataset

Attributes	% of missing values (before)	% of messing value (after)
QoS values	26%	23%
AS number	21%	15%
IP Address	24%	13%
Latitude/Longitude	22%	15%

Other changes are made on the first dataset, in order to reduce the missing values on the services attributes in terms of geolocational information (AS number, IP address, latitude and longitude). To do this, we used the geolocational information derived from GeoIP2[5], and IP2Location[6] databases. Table 2 shows the different improvements operated on the initial dataset.

[4] https://github.com/peterwittek/somoclu

[5] Maxmind GeoIP2 Geolocational Databases. Retrieved on May 2019 from http://dev.maxmind.com/geoip/geoip2/geolite2/

[6] IP2Location LITE Databases. Retrieved on May 2019 from http://lite.ip2location.com

4.1.2 Evaluation metrics

To evaluate the performance of the method, we use two well-known metrics frequently used in collaborative filtering: *mean absolute error*: $MAE_V = \frac{1}{|V|}\sum_{(u,s)\in V}|r_{u,s}-\hat{r}_{u,s}|$ and *root mean squared error*: $RMSE_V = \sqrt{\frac{1}{|V|}\sum_{(u,s)\in V}|r_{u,s}-\hat{r}_{u,s}|^2}$, where V represents the validation dataset. $r_{u,s}$ is the actual QoS score for service s given by user u, and $\hat{r}_{u,s}$ the predicted one.
Since we used k-fold cross-validation, we assess the prediction on the average of MAE and RMSE too. The average MAE is defined as follows:

$$MAE_{AVG} = \frac{1}{\left|\bigcup_{allV_{k\text{-}fold}}\right|}\sum_{allV_{k\text{-}fold}}(|V_{k\text{-}fold}|\times MAE_{V_{k\text{-}fold}}) \quad (8)$$

The average RMSE (which is also denoted as *mve* in step 11 of Algorithm 2) is defined as:

$$RMSE_{AVG} = \frac{1}{\left|\bigcup_{allV_{k\text{-}fold}}\right|}\sum_{allV_{k\text{-}fold}}(|V_{k\text{-}fold}|\times RMSE_{V_{k\text{-}fold}}) \quad (9)$$

where $V_{k\text{-}fold}$ represents a validation set in both previous equations.

4.2 Effects of training parameters

Autoencoder models are full of hyper-parameters and setting a proper parameter initialization is a great challenge. This requires expertise and extensive trial and error. In our work, we focused on two important hyper-parameters: the size of each layer in the autoencoder and the choice of the activation function on these layers. In the following, we present a series of comparative experiments conducted in order to obtain the optimal parameters of our model.

4.2.1 Impact of autoencoder layer sizes

The size of the common layer (latent factor layer) is a hyper-parameter that we must set before training the autoencoder. Like any feed-forward networks, it has been proven that deep autoencoders yield much better compression than corresponding shallow or linear autoencoders [18]. So, additional layers can learn complex representations by approximating any mapping from input to code arbitrarily well. For this reason, and aside from the latent factor layer, we used a three layer net for both encoding and decoding networks. This choice is based on the power of three-layer neural network experimentally proven in [17]. The final architecture of our autoencoder is composed of four fully connected layers (three layers + latent factor layer) for both encoding and decoding networks.

In order to determine the ideal number of neurons for each layer, many rules-of-thumb have been taken into account [31]. The code size should be: (1) between the size of the input layer and the size of the output layer; (2) two-thirds the size of the input layer, plus the size of the output layer; (3) less than twice the size of the input layer. We considered these three rules as a starting point for determining the upper bound on this parameter.
Figure 5(A) shows the results of four different values that operate on the layers sizes of the autoencoder. As we can see, the configuration that performs best on our 4-layer-model is 1024, 512, 256 and 128. Note that this configuration depends on the dimension of our input (with 4500 input services) for the autoencoder model. For the clustered autoencoder, this configuration is updated by the rule of three on the input size (which represents the cluster size).

4.2.2 Impact of activation functions

The second important hyper-parameter used in the training of our autoencoder is the target activation. To examine the influence of this parameter, we empirically used some of the most popular choices in deep learning. So, we used a sigmoid such as standard activation function and exponential linear units [11] (ELU), standard rectified linear unit (ReLU) [37] and finally, a leaky rectified linear unit (Leaky ReLU) [46] known as non-standard activation functions.
The effect of the different activation functions on RMSE training metric is shown in Figure 5(B). In order to accelerate the convergence speed we opted for a non-saturated activation function: Leaky ReLU.

4.3 Results and discussion

In deep learning, it has been experimentally proven that the classical model of autoencoders degenerate into identity networks and they fail to learn the latent relationship between data [13]. To tackle this issue, we used a variant model of

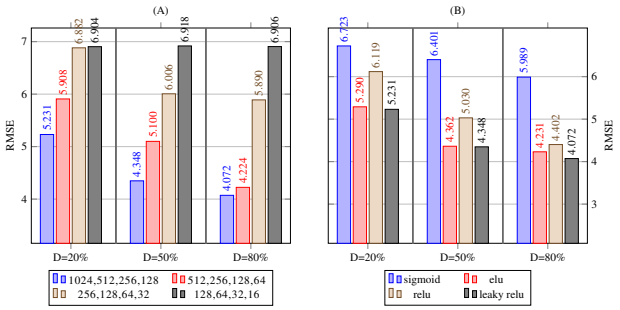

Fig. 5 Training RMSE per $mini-batch$ on the second dataset, with $learningrate = 0.05$, $epoch = 20$, for an optimal parameters selection. (A) Influence of layers size, (B) influence of activation functions.

deep autoencoder by corrupting the inputs, training the model to denoise the final outputs.

To enhance the robustness of the model, we adopted a masking noise technique to set a random fraction of the input to zero. For this reason, we explore four rates of noise: 0% (i.e., no noise), 20%, 50% and 80%, applied on these models with the whole dataset in its input. Note that the same training process is applied on experiments for each clustered deep autoencoder after SOM clustering phase on all input data.

4.3.1 Deep autoencoder

To extract the latent factor from the dataset, we used a 4-layer encoder network with the following configuration on the layers sizes: 1024-512-256-128. The same symmetric configuration is used for the decoder network to reconstruct the input data.

To ensure more robust prediction results, we lead our experiments with a multiple folders according to the k-fold cross-validation principle. Indeed, the experiments are performed using five partitions (folders) with different density values. Each partition represents a given percentage (*density%*) of the available dataset. The *density* represents the ratio between the training and the validation datasets. To this end, we explore three possibilities for density ratio: 20%, 50% and 80%. Finally, the validation results are averaged over the 5-folders through the application of equations 8 and 9.
In Figure 6 (A), we illustrate the average RMSE/MAE after five runs operated on three different data densities. For each density value, we applied four noise variations on the input data. Note that the QoS prediction accuracy increases continuously when the matrix density increases (due to the increase of the noise rate).
In Figure 6 (B), we can see that the training converged for all the possibilities, after fewer than 100 epochs. Also, we can easily observe that there is a stability (no increase after converging), and no overshoot (no increase before converging).

4.3.2 Clustered deep autoencoder

In order to provide yet a better prediction accuracy, we used a Clustered Autoencoder. This autoencoder variant is designed after the partition of all services into eight clusters that are homogeneous with regard to their geographical characteristics. As mentioned above, we used the same architecture of our deep autoencoder (4-layer-model configuration: 4-layer sizes, and the activation functions are Leaky ReLU). The main difference is that the size of the four layers that make up each clustered autoencoder is depending on the size of the inputs (number of services in the cluster). The layer sizes are updated by the rule of three on the initial configuration (1024-512-256-128) according to the cluster size (input size). To do so, we performed eight trainings for each cluster. We applied the same process on each cluster under various input data densities (20%, 50% and 80%). For each density case, we applied a set of noise variations (0%, 20%, 50% and 80%). Thereafter, we average the validation over those clusters. As shown in the results from Figure 7(A) the

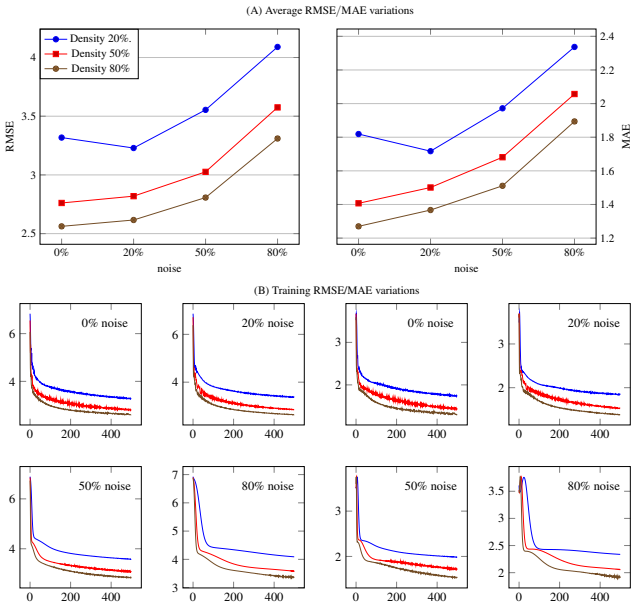

Fig. 6 The results with a training/testing set of (*density%*)/(100 − *density%*) for a **deep autoencoder** prediction (on various noises and densities). (A) The average *RMSE* and *MAE* results, and (B) The training performance graph.

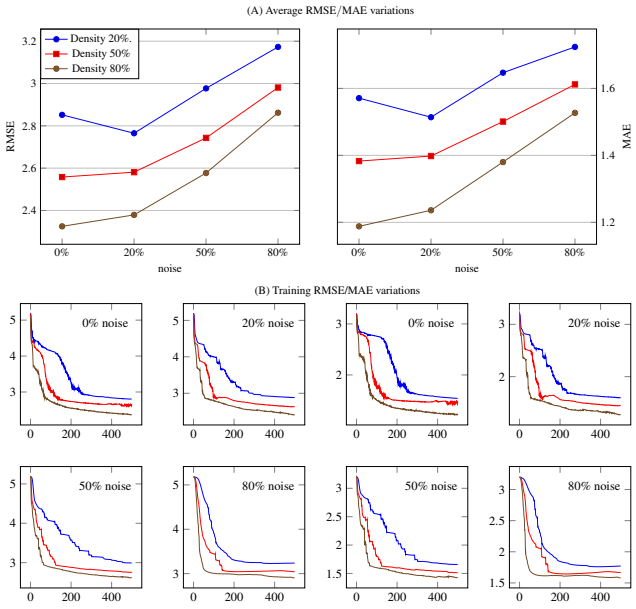

Fig. 7 The results with a training/testing set of (*density%*)/(100 − *density%*) for a **clustered deep autoencoder** prediction (on various noises and densities). (A) The average *RMSE* and *MAE* results, and (B) The training performance graph.

average RMSE/MAE for different densities increases continuously when the rate of corrupted input data increases (except for the case when density is 20%). In Figure 7 (B), we notice that the training converges a little slower than the deep autoencoder model.

4.3.3 SOM Initialization

This subsection demonstrates the performance of SOM as a method for geographical clustering instead of using other alternative methods like k-means with Euclidean distance, Hierarchical clustering, etc.
As already noted, the final results of clustered autoencoder (see Figure 7 (A)), are the average result for the training of eight different autoencoders. Each autoencoder is built according to its input vector size (number of services). The histogram depicted in Figure 8 (A) shows the detailed results of training autoencoders for each cluster separately. In fact, this graph represents the results in terms of RMSE and MAE metrics of eight autoencoders. The average RMSE and average MAE are plotted with blue and red lines respectively. Note that the x-axis corresponds to the size of input data for those autoencoders.

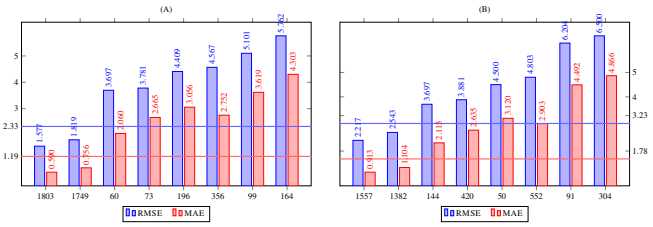

Fig. 8 Detailed results of clustered autoencoder with $density = 80\%$ and $noise = 0\%$. SOM clustering results: (A) based on Geographical characteristics initialization. (B) with random initialization

As we can see, the best results are for the clusters with the highest number of services (clusters with 1803 and 1749 services). Their results are far less than the average result. However, the third and the fourth ranked autoencoders are for the clusters with the smallest number of services (60 and 73 respectively). For the remaining clusters, despite the fact that they contain a considerable number of services, we note that their results are significantly higher then the average result.

The major significance of these results is on the manner that we operate for the initialization of the code-book. In fact, we opted for a Kohonen feature map network with 80×80 neurons to perform the clustering of 4500 services. Each line on this map contains 80 neurons and represents one country from our dataset. Note that, in ws-dream dataset we detected 81 different countries (we merged two countries). The size of each neuron vector is equal to the number of users available in our dataset (i.e., 142 users). So, each line for a selected $country_i$ on the map represents semantically the maximal geodesic distance (according to their longitude/latitude values) between all services pairs which compose this $country_i$ (i.e., $argmax(\mathscr{D}(s_1,s_2)), \forall s_1,s_2 \in Country_i$), where $\mathscr{D}$ denotes a geodesic distance function. This means that the first neuron is initialized with the vector of s_1 divided by the size of all services which compose the selected country. The same thing is operated for the last neuron with the vector s_2. The rest of services are distributed on the rest of neurons according to their geodesic distance. Note that, if there are several services with the same geographic coordinates, we put on the corresponding neuron the average vector of those services. Finally, the services which do not have the geographical information (i.e., latitude/longitude) are randomly dispatched throughout the whole neurons of their corresponding country line.

In support of our initialization outcomes (Figure 8 (A)), other experiments were performed with a random initialization of the code-book in SOM algorithm. As we can see in Figure 8 (B)), the average results in terms of RMSE and MAE are less than the results of experiments operated on SOM clustering results based on Geographical characteristics initialization.

Other interesting observations on our proposed initialization method are:

Table 3 Information details of ws-dream repository

Cluster size	% of sparsity	% missing geoloc. info.
1803	22,34%	13,81%
1749	23,10%	9,43%
60	19,30%	20,00%
73	24,44%	13,70%
196	28,06%	32,14%
356	26,63%	33,99%
99	22,53%	15,15%
164	26,69%	24,39%

- Firstly, in terms of RMSE and MAE, we observe that about 80% of all services are clustered in the top-3 clusters for clustering based on Geographical characteristics initialization (figure 8 (A)), and only 69% for a random initialization (figure 8 (B)).
- Secondly, we observe that there is a directly proportional relationship between the training results and the sparsity on data of each cluster. The more the density of input data is the better the results are. Table 3 summarizes the sparsity (column two) ratio on each cluster (column one), knowing that the sparsity of all data is about 23%.
- The third important observation is about the percentage of missing geolocational information. We observed that most of services without geographical characteristics are clustered in the latest clusters. The third column in Ta-

ble 3 shows the missing rate according to the clusters size.
- Finally, with our proposed initialization mechanism, we ensured a deterministic behavior of clustering result.

Based on this analysis, and in order to preserve the topological properties of the dataset and reduce the effect of QoS sparsity, we take the advantage of SOM algorithm to create the appropriate clusters with our proposed initialization mechanism. Table 4 summarizes the results in terms of RMSE, MAE and TIME for the three architectures: A deep autoencoder, a clustered autoencoder and clustered* autoencoder with only top-3 clusters.
As already mentioned, the results represent the comparison between the three different approaches that we have implemented, so, the results are grouped according to those methods. The *Deep AE* group concerns the results on test data after training the deep autoencoder without using any clustering. The *Clustered AE* group concerns the average results for eight clusters of the deep autoencoder. The third group (*Clustered* AE*) presents the average testing results on the top-3 best clusters (about 80% of all services are clustered in the top-3 clusters). All the experiments are conducted with three different data density values, and by adding a random noise to the input data for each of them. Note that the technique of adding the noise, as specified in [30], not only helps us to avoid over-fitting, but also can result in lower training loss (especially when density is about 20%).
This clearly demonstrates the interest of using the clustering phase before. Indeed, the results show that the method with SOM clustering behaves better than the first method where the prediction of QoS values is performed on the whole dataset. Consequently, using deep-autoencoder on the top-3 clusters outperforms the other methods. We also note that along with the data densities increasing, the prediction metrics become smaller.

A detailed version of experiments operated on the whole dataset according to k-fold cross-validation principle can be found at the following links:

- Deep Autoencoder: `http://bit.ly/2JrIGSs`
- Clustered deep autoencoder: `http://bit.ly/2XKRMxO`

We additionally observe, that not only the prediction accuracy decreases on the clustered AE, but the training time improves significantly from the deep autoencoder method to the clusterd AE method.

4.3.4 Performance Comparison

With the aim of evaluating the advantages of our method and its variants, we compare them with the following state-of-art baseline QoS prediction methods. *UPCC* [50]: User-based CF using PCC. This method is a user-based model using Pearson Correlation Coefficient for recommendation and prediction of Web services. *IPCC* [34]: Item-based CF using PCC. This method uses similar services for the QoS prediction using Pearson Correlation Coefficient method. *ARIMA* [14]: this method is often considered as the baseline method. It is a statistical method adapted to QoS web service prediction. *WSRec* [51]: It is a hybrid CF algorithm that combines a user-based prediction [50] model with item-based prediction model [34]. *Lasso* [41]: this approach optimizes the recommendation problem by adapting the lasso penalty function. Country-clustered *Autoencoder* [36]: this is our first approach to predict QoS of Web services based on historical data. We used a simple (not deep) autoencoder architecture (with only one hidden layer) to predict the QoS values on sets of clusters based on the country ID. For that, all the experiments for our previous approach have been reconducted in line with the actual execution parameters (number of clusters[7] and number of folders).

We note that our work is not compared with QoS Prediction methods that use a Topk based model (since they do not consider the entire dataset).

Table 5 presents the MAE and RMSE results of different prediction methods on response-time criterion when the training set densities take two different values: 80% and 50%. From these results, we notice the following observations:

1. The prediction accuracy of Lasso is better then Country AE, WSRec and ARIMA methods;
2. The Deep AE method is slightly better then Lasso method for the two density values. However, when the rate of corruption data increases the performance of our Deep AE decreases, compared to Lasso method;
3. Compared with Lasso method, the Clustred AE (when no noise) can obtain as high as 9.6% and 11% improvements in prediction accuracy when data density is 80% and 50% respectively. Furthermore, the performances are almost identical when the input noise is about 50%.
4. We clearly remark that the top-3 clustering deep autoencoders outperform the others methods in all cases (density and noise variations) even when the noise rate is around 80%.
5. Globally, when the noise is lower or equal to 20%, we observed that the results of the three variants of our proposed method outperform other prediction approaches in prediction accuracy in all cases in terms of RMSE metric (except for Deep AE when density is about 80% and noise is equal to 20%), and in the most cases from MAE metric.

[7] For this work, the clustering is performed on country ID. In order to have only eight clusters we grouped some of countries in the same cluster

Table 4 Users accuracy comparison methods on RT criterion (training time is specified in minutes)

Method	Noise	Density=20%			Density=50%			Density=80%		
		RMSE	MAE	TIME	RMSE	MAE	TIME	RMSE	MAE	TIME
Deep AE	0%	3,318	1,819	121,2	**2,761**	**1,407**	144,5	**2,562**	**1,270**	194,2
	20%	**3,229**	**1,717**	97,98	2,819	1,501	128,9	2,616	1,367	167,6
	50%	3,554	1,972	92,56	3,026	1,681	115,8	2,807	1,511	143,4
	80%	4,090	2,337	94,92	3,575	2,057	108,4	3,310	1,894	131,1
Clustered AE	0%	2,852	1,571	53,74	**2,558**	**1,383**	69,29	**2,325**	**1,188**	85,34
	20%	**2,765**	**1,514**	54,16	2,581	1,398	70,20	2,379	1,236	86,26
	50%	2,977	1,647	54,34	2,743	1,501	70,30	2,577	1,380	86,12
	80%	3,173	1,724	53,97	2,981	1,612	69,68	2,862	1,527	85,40
Clustered* AE	0%	**1,908**	**0,840**	42,20	**1,762**	**0,771**	54,70	**1,700**	**0,748**	66,90
	20%	1,997	0,907	42,25	1,806	0,802	55,10	1,731	0,771	67,90
	50%	2,087	0,936	42,32	1,917	0,858	55,20	1,829	0,822	68,00
	80%	2,224	0,956	42,31	2,128	0,945	55,10	2,071	0,930	67,50

Table 5 Performance Comparisons of Prediction methods on RT with two data density values

Method		Density=50%		Density=80%	
		RMSE	MAE	RMSE	MAE
UPCC		3,034	1,470	3,032	1,467
IPCC		2,951	1,396	2.925	1.372
ARIMA		3,401	1,236	2,986	1,028
WSRec		2,945	1,380	2,925	1,372
Lasso		**2,872**	**1,021**	**2,572**	**0,893**
Country AE		3,825	1.892	2,803	1,250
Deep AE	noise = 0%	**2,761**	**1,407**	**2,562**	**1,270**
	noise = 20%	2,819	1,501	2,616	1,367
	noise = 50%	3,026	1,681	2,807	1,511
	noise = 80%	3,575	2,057	3,310	1,894
Cluster AE	noise = 0%	**2,558**	**1,383**	**2,325**	**1,188**
	noise = 20%	2,581	1,398	2,379	1,236
	noise = 50%	2,743	1,501	2,577	1,380
	noise = 80%	2,981	1,612	2,862	1,527
Cluster* AE	noise = 0%	**1,762**	**0,771**	**1,700**	**0,748**
	noise = 20%	1,806	0,802	1,731	0,771
	noise = 50%	1,917	0,858	1,829	0.822
	noise = 80%	2,128	0,945	2,071	0,930

4.4 Cold-start situation

To handle the cold-start problem, we adopt the strategy described in algorithm 3. More specifically, we estimate the initial QoS values of a new service (or new user) as follows: First, we use the geographical characteristics of a service to determine the most representative cluster (in terms of country, autonomous system, or provider); this step is performed by using the first part of Algorithm 3 (from line 1 to line 9). Then, we use the new QoS values (given by the first step) as an input of all trained DAEs and take the average of the returned results (line 11).

Note that, in the learning phase, we use about 60% of entire dataset as examples to fit the parameters of the SOM classifier. In the testing phase, we use about 40% of examples to assess the performance of our fully-specified classifier.

As depicted in Table 6, the RMSE, MAE, and testing TIME, show the postive impact of the contextual characteristics on the resolution of the cold-start problem. For that, we are not only focused on the checking of the most representative cluster in terms of geographical characteristics (step 1), but we also use already the trained DAE (after using Algorithm 2) to average the returned results. We show that the application of DAE (step 2), can significantly improve the results.

Table 6 Cold-start problem (time is specified in seconds)

Cold-Start Steps	RMSE	MAE	TIME
Step 1	3.877	1.951	1.201
Step 2	3.010	1.420	2.194

4.5 Threats to validity

Regarding the recommendations in [44], there are several elements to discuss about the validity of the experiment. For that, we consider the threats to internal and external validity of our study:

- *Internal Validity* threats are intrinsically tied to experimental realism. To select the different parameters for our models, we have tried to be as objective as possible. However, despite the fact that we conduct several pre-trainings for selecting the best configuration of those parameters, other different values may conduct to different results. For the deep autoencoder model, since we split the dataset into training and testing sets with varying proportions and use the k-fold cross-validation, this can alleviate the actual threat. For the clustered deep autoencoder model, and in addition to previous remarks, internal validity threats are mitigated since the number of neurons is proportional with the size of clusters.
- *External Validity* threats are about the ability to generalize our proper findings and conclusions to other contexts. In this study, we performed our experiments on a large dataset (ws-dream), which enables us to train correctly our prediction method. To the best of our knowledge, this is the only large Web service QoS dataset publicly available. Although we used only one dataset, we are quite confident that the impact of using another dataset

is reduced with the use of k-fold cross-validation which operates on multiple data densities. On the other side, the fact of using a denoising autoencoders by corrupting the data by randomly forcing some of the input values to zero, decreases the risk of external validity.

5 Related Works

In [49,35,21], the authors draw up a detailed survey and new perspectives on "deep learning"-based recommendation systems. For the two first articles, the authors treat all categories of recommendation systems. However in the third article, the authors focus only on the Collaborative Filtering recommendation algorithms.

In this paper, we focused on the Collaborative Filtering methods using a deep learning mechanism and/or exploiting the context information (and more particularly geographical information) for clustering Web services. For this reason, we present recent related works according to these two different aspects:

5.1 Clustering-based Methods in Collaborative Filtering

In the literature, several clustering-based works have been carried out [38,10,8,9,29,28]. Those studies are trying to discover a set of clusters based on the neighborhood characteristics of data.

In [29], the authors focus on Cloud Services Selection according to users' non-functional requirements. Their method is based on time series QoS data. In order to identify the user clusters, they use the double *Mahalanobis* distances to improve the similarity measurement of QoS cloud models during multiple periods. They assume two challenges: Challenge in exactly identifying the neighboring users for a current user and Challenge in selecting the appropriate cloud service with optimal QoS meeting user's period preferences.

In the work presented in [28], the authors assume that most existing CF approaches ignore the influence of task similarity among different users on QoS prediction results and vice versa (i.e. the influence of similar users on different tasks). To address this problem, the authors proposed a novel clustering-based and trust-aware method for personalized and reliable QoS values prediction. For that, they combine two contributions to make a more personalized QoS prediction. For the first one, they develop a clustering-based algorithm to identify a set of similar users from the point of view of task similarity. The tasks similarities are computed by incorporating both explicit textual information and rating information as well as implicit context information. For the second one, they made assumptions that the QoS values may be contributed by unreliable users. For that, they design a trust-aware CF approach by merging local and global trust values, to reconstruct trust network of the clustered users. A series of experiments on two real-world datasets were conducted to evaluate the proposed approach.

The work presented in [8] aims to improve QoS prediction accuracy. To this end, the authors have considered the factor of geographical neighborhood to perform both matrix factorization and service clustering. The authors built a set of service clusters based on geo-neighbors similarity. The clusters are identified using both a bottom-up hierarchical neighbor-hood and the contextual attributes (like the country and service provider). Accordingly, the authors leveraged a neighborhood-based term to decompose the QoS matrix and predict the missing values.

5.2 Deep Learning Methods in Collaborative Filtering

Recently, deep learning-based models are rapidly developed in the era of recommendation. A wide range of Collaborative Filtering approaches have been proposed like in [23,45,4, 40]. We focus next on two works that we consider as very related to our work.

In [45], the authors proposed a hybrid deep learning approach by combining Matrix Factorization with Content filtering to improve Web service recommendation. To predict the probability of a particular service to be invoked by a particular web application or mashup (user in our case), they used two feed-forward neural networks for both Collaborative filtering and Content filtering components. After that, these two components are combined by concatenating their last hidden layers to compose a third multi-layer perceptron to learn the interaction between them as well as their functionalities (it is well known that the interaction treated with Collaborative filtering and Content filtering is about multiple factors such as invocation history and functionalities). In order to evaluate their method, the authors conducted several experiments using a real-world Web service dataset. However, despite the fact that for those experiments the training data is randomly selected under different percentages of training data (20% to 80% with a step of size 10%), they kept only 20% in test data for each experiment round. The second remark is about the lack of information on their choices of the number of layers and the size of each of them and how those choices can affect the final results. A final remark is about the learning time, which is relatively high due to the fact that they try to learn three different neural networks.

In [4], the authors used an autoencoder architecture to learn low-dimensional representations and perform feature extraction. To this end, they build a deep learning framework to recommend long-tail Web services. To tackle the problem of the unsatisfactory quality of the low-quality of the content description, they used a stacked denoising autoencoder to perform feature extraction for long-tail Web

services with severe sparsity on both their content description and their historical usage data. The proposed approach is tested on a real-world dataset and compared with several state-of-art baselines. In summary, this framework can be considered as a content-based recommendation system. However, from the viewpoint of matrix factorization and collaborative filtering paradigm, the interaction between the applications (mashups) and services is not addressed here.

5.3 Positioning of the approach

In contrast to the aforementioned works, our contribution leverages both contextual attributes and QoS data to perform clustering. By doing so, we limit the chances of having sparse or small-sized clusters. More specifically, the initialization of the cluster-heads (or weights of neurons) is performed using contextual attributes; in addition, this initialization is accomplished in a self-organized manner (i.e., the neighboring neurons of the map are also neighbors in the physical network). Besides that, the training of the SOM model exploits the QoS data (instead of contextual attributes) to update the weights of the cluster-heads.

To the best of our knowledge, our contribution is the first work that designs a deep autoencoder architecture (and its variant denoising autoencoder) for QoS prediction Web services.

6 Conclusion

QoS is a key factor for building successful service-oriented applications. Most existing QoS-predicting works do not handle data sparsity, the cold-start problems, and the contextual information of services and thus are likely to perform worse in real scenarios.

In this paper, we tackled the aforementioned issues by adopting a DAE architecture. To optimize prediction capabilities, the hyper-parameters of the DAE are tuned through cross-validation. We also introduced a SOM-based clustering step as a pre-processing to the DAE learning. This step aims to reduce QoS matrix sparsity and improve prediction accuracy. We conducted a set of experiments to evaluate the performance of this prediction method. We explored many variants of this method and compared their performance with the state-of-the-art methods. These experiments showed that our method outperforms the existing methods in terms of accuracy.

As perspectives to this work, we would like to investigate first the use of alternative architectures of deep learning models like stacked autoencoders or deep belief networks. These architectures may ensure more accurate prediction results than the autoencoders used in this work. Second, we plan to use deep learning in predicting Web service Quality of Experience (QoE). We intend to use some powerful text mining tools to analyse the users' feedbacks in order to estimate QoE scores for Web services. We argue that the combination of QoS and QoE will contribute to improve Web service recommendation systems.

Acknowledgment

This work has been performed with the support of the High Performance Computing Platform MESO@LR, financed by the Occitanie / Pyrénées-Méditerranée Region, Montpellier Mediterranean Metropole and the University of Montpellier, France.

References

1. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge & Data Engineering **17**(6), 734–749 (2005)
2. Akinduko, A.A., Mirkes, E.M., Gorban, A.N.: Som: Stochastic initialization versus principal components. Information Sciences **364**, 213–221 (2016)
3. Alain, G., Bengio, Y.: What regularized auto-encoders learn from the data-generating distribution. The Journal of Machine Learning Research **15**(1), 3563–3593 (2014)
4. Bai, B., Fan, Y., Tan, W., Zhang, J.: Dltsr: A deep learning framework for recommendations of long-tail web services. IEEE Transactions on Services Computing **13**(01), 73–85 (2020). DOI 10.1109/TSC.2017.2681666
5. Chen, S., Fan, Y., Tan, W., Zhang, J., Bai, B., Gao, Z.: Service recommendation based on separated time-aware collaborative poisson factorization. J. Web Eng. **16**(7&8), 595–618 (2017)
6. Chen, W.Y., Chu, J.C., Luan, J., Bai, H., Wang, Y., Chang, E.Y.: Collaborative filtering for orkut communities: discovery of user latent behavior. In: Proceedings of the 18th international conference on World wide web, pp. 681–690. ACM (2009)
7. Chen, X., Zheng, Z., Yu, Q., Lyu, M.R.: Web service recommendation via exploiting location and qos information. IEEE Transactions on Parallel and Distributed Systems **25**(7), 1913–1924 (2013)
8. Chen, Z., Shen, L., Li, F.: Exploiting web service geographical neighborhood for collaborative qos prediction. Future Generation Computer Systems **68**, 248–259 (2017)
9. Chen, Z., Shen, L., Li, F.: Your neighbors are misunderstood: On modeling accurate similarity driven by data range to collaborative web service qos prediction. Future Generation Computer Systems **95**, 404–419 (2019)
10. Chen, Z., Shen, L., Li, F., You, D.: Your neighbors alleviate cold-start: On geographical neighborhood influence to collaborative web service qos prediction. Knowledge-Based Systems **138**, 188–201 (2017)
11. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289 (2015)
12. Deshpande, M., Karypis, G.: Item-based top-n recommendation algorithms. ACM Transactions on Information Systems (TOIS) **22**(1), 143–177 (2004)
13. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp. 249–256 (2010)

14. Godse, M., Bellur, U., Sonar, R.: Automating qos based service selection. In: 2010 IEEE International Conference on Web Services, pp. 534–541. IEEE (2010)
15. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). http://www.deeplearningbook.org
16. Hartigan, J.A., Wong, M.A.: Algorithm as 136: A k-means clustering algorithm. Journal of the Royal Statistical Society. Series C (Applied Statistics) **28**(1), 100–108 (1979)
17. Hecht-Nielsen, R.: Theory of the backpropagation neural network. In: H. Wechsler (ed.) Neural Networks for Perception, pp. 65–93. Academic Press (1992)
18. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. science **313**(5786), 504–507 (2006)
19. Huang, A.F., Lan, C.W., Yang, S.J.: An optimal qos-based web service selection scheme. Information Sciences **179**(19), 3309–3322 (2009)
20. Huang, Z., Chen, H., Zeng, D.: Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. ACM Transactions on Information Systems (TOIS) **22**(1), 116–142 (2004)
21. Jalili, M., Ahmadian, S., Izadi, M., Moradi, P., Salehi, M.: Evaluating collaborative filtering recommender algorithms: A survey. IEEE Access **6**, 74003–74024 (2018)
22. Jolliffe, I.: Principal component analysis. Springer (2011)
23. Kanagawa, H., Kobayashi, H., Shimizu, N., Tagami, Y., Suzuki, T.: Cross-domain recommendation via deep domain adaptation. In: European Conference on Information Retrieval, pp. 20–29. Springer (2019)
24. Kiang, M.Y., Kulkarni, U.R., Tam, K.Y.: Self-organizing map network as an interactive clustering tool-an application to group technology. Decision Support Systems **15**(4), 351–374 (1995)
25. Kohonen, T.: Self-organization and associative memory, vol. 8. Springer Science & Business Media (2012)
26. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)
27. Li, S., Kawale, J., Fu, Y.: Deep collaborative filtering via marginalized denoising auto-encoder. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, pp. 811–820. ACM (2015)
28. Liu, J., Chen, Y.: A personalized clustering-based and reliable trust-aware qos prediction approach for cloud service recommendation in cloud manufacturing. Knowledge-Based Systems **174**, 43–56 (2019)
29. Ma, H., Hu, Z., Li, K., Zhu, H.: Variation-aware cloud service selection via collaborative qos prediction. IEEE Transactions on Services Computing (2019)
30. Neelakantan, A., Vilnis, L., Le, Q.V., Sutskever, I., Kaiser, L., Kurach, K., Martens, J.: Adding gradient noise improves learning for very deep networks. arXiv preprint arXiv:1511.06807 (2015)
31. Panchal, G., Ganatra, A., Kosta, Y., Panchal, D.: Behaviour analysis of multilayer perceptrons with multiple hidden neurons and hidden layers. International Journal of Computer Theory and Engineering **3**(2), 332–337 (2011)
32. Refaeilzadeh, P., Tang, L., Liu, H.: Cross-validation. Encyclopedia of database systems **5**, 532–538 (2009)
33. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation. Tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science (1985)
34. Sarwar, B.M., Karypis, G., Konstan, J.A., Riedl, J., et al.: Item-based collaborative filtering recommendation algorithms. Www **1**, 285–295 (2001)
35. Singhal, A., Sinha, P., Pant, R.: Use of deep learning in modern recommendation system: A summary of recent works. arXiv preprint arXiv:1712.07525 (2017)
36. Smahi, M.I., Hadjila, F., Tibermacine, C., Merzoug, M., Benamar, A.: An encoder-decoder architecture for the prediction of web service qos. In: European Conference on Service-Oriented and Cloud Computing, pp. 74–89. Springer (2018)
37. Sun, Y., Wang, X., Tang, X.: Deeply learned face representations are sparse, selective, and robust. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2892–2900 (2015)
38. Tang, M., Zheng, Z., Kang, G., Liu, J., Yang, Y., Zhang, T.: Collaborative web service quality prediction via exploiting matrix factorization and network map. IEEE Transactions on Network and Service Management **13**(1), 126–137 (2016)
39. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. Journal of machine learning research **11**(Dec), 3371–3408 (2010)
40. Wang, H., Wang, L., Yu, Q., Zheng, Z., Bouguettaya, A., Lyu, M.R.: Online reliability prediction via motifs-based dynamic bayesian networks for service-oriented systems. IEEE Transactions on Software Engineering **43**(6), 556–579 (2016)
41. Wang, X., Zhu, J., Zheng, Z., Song, W., Shen, Y., Lyu, M.R.: A spatial-temporal qos prediction approach for time-aware web service recommendation. ACM Transactions on the Web (TWEB) **10**(1), 7 (2016)
42. Wei, J., He, J., Chen, K., Zhou, Y., Tang, Z.: Collaborative filtering and deep learning based recommendation system for cold start items. Expert Systems with Applications **69**, 29–39 (2017)
43. Wittek, P., Gao, S., Lim, I., Zhao, L.: somoclu: An efficient parallel library for self-organizing maps. Journal of Statistical Software, Articles **78**(9), 1–21 (2017). DOI 10.18637/jss.v078.i09. URL https://www.jstatsoft.org/v078/i09
44. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering. Springer Science & Business Media (2012)
45. Xiong, R., Wang, J., Zhang, N., Ma, Y.: Deep hybrid collaborative filtering for web service recommendation. Expert Systems with Applications **110**, 191–205 (2018)
46. Xu, B., Wang, N., Chen, T., Li, M.: Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853 (2015)
47. Xu, Y., Yin, J., Deng, S., Xiong, N.N., Huang, J.: Context-aware qos prediction for web service recommendation and selection. Expert Systems with Applications **53**, 75–86 (2016)
48. Yu, T., Zhang, Y., Lin, K.J.: Efficient algorithms for web services selection with end-to-end qos constraints. ACM Transactions on the Web (TWEB) **1**(1), 6 (2007)
49. Zhang, S., Yao, L., Sun, A., Tay, Y.: Deep learning based recommender system: A survey and new perspectives. ACM Computing Surveys (CSUR) **52**(1), 5 (2019)
50. Zhao, Z.D., Shang, M.S.: User-based collaborative-filtering recommendation algorithms on hadoop. In: 2010 Third International Conference on Knowledge Discovery and Data Mining, pp. 478–481. IEEE (2010)
51. Zheng, Z., Ma, H., Lyu, M.R., King, I.: Wsrec: A collaborative filtering based web service recommender system. In: 2009 IEEE International Conference on Web Services, pp. 437–444. IEEE (2009)
52. Zheng, Z., Ma, H., Lyu, M.R., King, I.: Qos-aware web service recommendation by collaborative filtering. IEEE Transactions on services computing **4**(2), 140–152 (2010)
53. Zheng, Z., Ma, H., Lyu, M.R., King, I.: Collaborative web service qos prediction via neighborhood integrated matrix factorization. IEEE Transactions on Services Computing **6**(3), 289–299 (2012)
54. Zheng, Z., Zhang, Y., Lyu, M.R.: Investigating qos of real-world web services. IEEE transactions on services computing **7**(1), 32–39 (2014)