Journal of Engineering and Technology for Industrial Applications

ITEGAM-JETIA

Manaus, v.12 n.58, p.799-807. March/April, 2026.
DOI: https://doi.org/10.5935/jetia.v12i58.3043

ISSN ONLINE: 2447-0228

RESEARCH ARTICLE OPEN ACCESS

WEB SERVICE QOS PREDICTION BASED ON AUTOENCODER WITH MINI-BATCH GRADIENT DESCENT

Le Van Thinh[1]

[1]Faculty of Informatics and Economics, Mientrung Industry and Trade College, Vietnam.

[1]https://orcid.org/0009-0007-5394-0409

Email: thinhcdcn@gmail.com

ARTICLE INFO

Article History
Received: December 03, 2025
Reviewed: January 04, 2026
Accepted: March 10, 2026
Published: April 30, 2026

Keywords:
Web service,
Autoencoder,
Mini-Batch Gradient Descents

ABSTRACT

Reliability prediction of Web services has become very important in related research communities. Especially, predicting the Quality of Service (QoS) for active users has been a hot issue of research and application. On the other hand, with the rapidly growing in number of service providers and users, resulting in a large number of data sets. It has a big impact to the QoS such as managing and monitoring for describing functional and non-functional characteristics of Web services. Therefore, we will certainly struggle at processing large data sets in the future, unless the issues are resolved quickly before it happens. In that context, QoS prediction on big data set is an urgent problem to be solved. In this paper, we present a new model for handling this problem based on autoencoder, it is called autominibatch. We use this model to cope with large data sets by using Backpropagation and Mini-batch gradient descent for predicting QoS values of Web services. This also is a new method for evaluating prediction of the field of web service quality. Our experiments were performed on two data sets in the WS-DREAM data set and the experimental results have proved the effectiveness of the proposed model.

Copyright ©2026 by authors and Galileo Institute of Technology and Education of the Amazon (ITEGAM). This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).

I. INTRODUCTION

Web services are a system which is designed applications to allow the computer to communicate with each other via network. Service providers were established based on the needs of users, with increasing demands of users and a highly profitable business environment with considerable potential for growth. As a result, many Service providers tend to move in the same direction as business and marketing plans, which provides the direction on the potential market environment, leading to multiple providers may offer functionally identical services. On the other hand, QoS also depends on many other factors such as transmission on the network, implementation mechanism and hardware. Therefore, the user desires to know how to choose a QoS from service providers such as latency, availability and reliability. However, almost service providers nowadays focus on the profit and growth. Moreover, in e-commerce and other online environments, the QoS properties of each service are pre-determined by the service providers, but because of its promotion purpose, they often exaggerate to attract more users. Hence, QoS is one of the important factors to build service-oriented applications [1-3].

The QoS property may have different values to different users (e.g., response time, availability, throughput etc.), to measure the efficiency of QoS properties, it depends on the user who uses that web service. If a recommender system can exploit the information of web service providers based on the relationships between the consumers, services and service providers, the accuracy of the forecast will be remarkably improved. But almost service providers only introduce the web services to the consumer without warning anything about their service quality provided by them to the consumer, if any, it is mainly for promotion purposes. Therefore, many studies have focused on solving this problem as Collaborative Filtering (CF) approach [4-6] and some other studies such as a Bayesian network based QoS assessment model [7], decision tree [8] and latent semantic models [9]. Some recent research publications about the use of machine learning focus on prediction for web service as learning with multi-agent techniques [10] and Latent Factor models [9]. However, all of them have not mentioned for processing large data sets. This is one of new challenges which we should have solution to deal.

Currently, the traditional algorithms are not able to handle efficiently for large and sparse data sets. Therefore, the need of new models which automatically extracts the necessary information to users with accurate and effective ways in the continuous increase of growing data is necessary. To overcome the drawbacks described above, we propose Autominibatch model. The model is inspired by previous applications of Deep Learning in a wide range of pattern recognition tasks as applications including image classification, processing and generation [11-13], application to exploit and extract the document [14] and sound modeling [15]. Additionally, it also has been applied for different missions such as: image noise reduction, number identification, document recovery. From these practical applications, we have constructed algorithms based on the Back propagation and mini-batch gradient descent in order to be appropriate for predicting the QoS of Web service and meet the performance requirements on processing big data and we have also proven the convergence for our model. Here are our contributions in this paper.

- We introduce a new model (autominibatch) based on autoencoder for predicting the QoS values.
- Especially, we have used mini-batch gradient descents to improve the convergence time for processing large data sets effectively.

The remainder of this paper is organized as follows: Section 2 introduces to motivation for our approach. In Section 3 we present the model. In Section 4, we present the learning the model. Section 5, we apply the proposed approach to experimental evaluation and report the reliability prediction results. Finally, we conclude our work and some future directions in Section 6.

II. MOTIVATION

Big data is a broad term of a too big or complex data set while traditional algorithms are often not designed to process. Recently, some research have been expanded from different methods to solve large data sets. However, the current approach only focus on the processing without considering other important functions. E.g. processing-data time and the speed of algorithm convergence must meet the processing time in the context of a very large data set. With this reason we have a desire to find out how to improve the data processing speed fast and effectively, this is the main motivation for our approach. Authors [16] have proposed a method based on mini-batch training gradient descent, this method has two advantages: Firstly, After just b examples, can improve the parameters. Second, don't have to wait until we cycled through all the data, and don't have to update parameters after every example. This is very important for dealing with large datasets and this is also the first time we have applied mini-batch Gradient descent to predict the QoS in Web field.

III. THE MODEL

Let X be a vector of m visible units $(X = (X_1 + X_2 + \cdots + X_m))$ where consisting of m service user and n web service, called the user-service matrix. Every entry $r_{i,k}$ in matrix represents the a vector of QoS values (e.g., response time, failure-rate, etc.), that is observed by the service user i on the Web service k. Let F denotes the number of h hidden units $(h = (h_1, h_2, \ldots h_F))$, it can be thought of as representing stochastic binary features that have different values for different users. An important proplem in applying autoencoder to cope with the missing QoS values efficiently. In this paper, we constructed the model for predicting the QoS values, the model use a different autoencoder for every user (as shown in Figure 1). Each service user is considered as a single training case for an autoencoder which had a softmax visible layer symmetrically connected to a set of binary hidden layer. Each hidden unit could then learn to model a significant dependency between different values. Each autoencoder has a single training case, but all of the corresponding weights and biases are tied together, so if two users have the same values, their two autoencoders must use the same weights between the softmax visible units for that web service item and the hidden layer.

Let a_i^l denotes the activation of the total weighted sum of inputs and the bias term to unit i in layer l. We have

- For $l = 1$, we have $a_i^{(1)} = r_{ij}$ to denote the $i = th$ input.
- For $l = 2$, we have:

$$\begin{Bmatrix} a_1^{(2)} = f(W_{11}^{(1)} r_{ij}^{(1)} + W_{12}^{(1)} r_{ij}^{(2)} + \cdots + W_{1(m\times n)}^{(1)} r_{ij}^{(K)} + b_1^{(1)} \\ a_2^{(2)} = f(W_{21}^{(1)} r_{ij}^{(1)} + W_{22}^{(1)} r_{ij}^{(2)} + \cdots + W_{2(m\times n)}^{(1)} r_{ij}^{(K)} + b_2^{(1)} \\ \vdots \\ a_F^{(2)} = f(W_{z1}^{(1)} r_{ij}^{(1)} + W_{z2}^{(1)} r_{ij}^{(2)} + \cdots + W_{z(m\times n)}^{(1)} r_{ij}^{(K)} + b_F^{(1)} \end{Bmatrix} \quad (1)$$

And

$$\begin{Bmatrix} h_{W,b}(X_1) = a_1^{(3)} = f(W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)} + \cdots + W_{1F}^{(2)} a_F^{(2)} + b_1^{(2)} \\ h_{W,b}(X_2) = a_2^{(3)} = f(W_{21}^{(2)} a_1^{(2)} + W_{22}^{(2)} a_2^{(2)} + \cdots + W_{2F}^{(2)} a_F^{(2)} + b_2^{(2)} \\ \vdots \\ h_{W,b}(X_m) = a_F^{(3)} = f(W_{F1}^{(2)} a_1^{(2)} + W_{F2}^{(2)} a_2^{(2)} + \cdots + W_{FF}^{(2)} a_F^{(2)} + b_F^{(2)} \end{Bmatrix} \quad (2)$$

Similar to $l = 3; l = 4; \ldots; l = s_l$
The following general formula

$$a_i^{(l)} = f(\sum_{j=1}^{F} W_{ij}^{(l-1)} a_j^{(l-1)} + b_i^{(l-1)}) \quad (3)$$

And calculating in the encoding step of the autoencoder as follows

$$h_{W,b}(X) = a_i^{(l+1)} = f(\sum_{j=1}^{F} W_{ij}^{(l)}.a_j^{(l)} + b_i^{(l)}) \quad (4)$$

Equation (4) may be rewritten as:

$$h_{W,b}(X) = a_i^{(l+1)} = f(\sum_{j=1}^{F} W_{ij}^{(l)}.f(\sum_{j=1}^{F} W_{ij}^{(l-1)}a_j^{(l-1)} + b_i^{(l-1)}) + b_i^{(l)}) \quad (5)$$

We can rewrite the general formula as follows

$$h_{W_1,W,b,b_1(X)=f(W_1.f(W.a+b)+b_1)} \quad (6)$$

where $f(.)$ is *activation functions.* Here, we set the activation function f to be the sigmoid function.

Table 1: Notations of model.

Notations	Explanation
S_l	denotes the number of layers in the model and we called label l as L_l
L_1	is the input layer
L_{sl}	is the output layer
W, b	denote parameters
W_{mn}^l	denotes the weight associated with the connection between unit n in layer l, and unit m in layer $l+1$
b_m^l	is the bias associated with unit m in layer $l+1$.
$h_{W,b}(X)$	is a hypothesis that outputs a real number
a_i^l	Activation (output) of unit i in layer i of the network.

Source: Author, (2026).

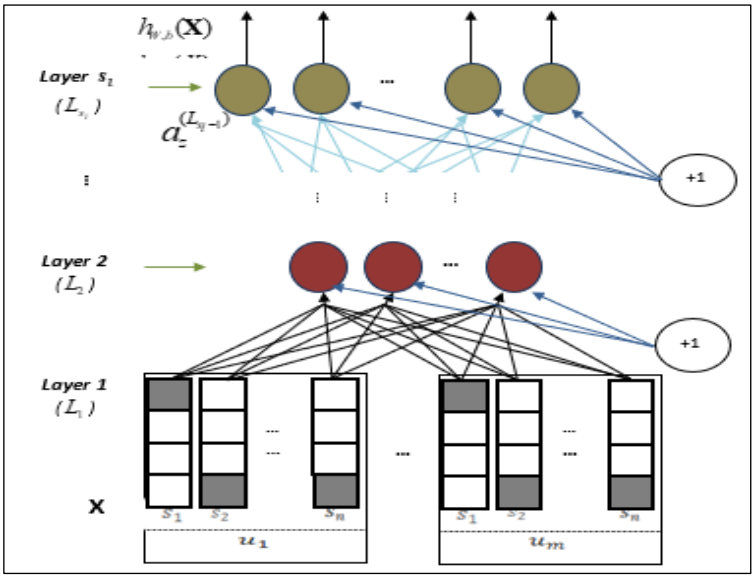

Figure 1: The Autominibatch model based on autoencoder (AWS).
Source: Author, (2026).

IV. LEARNING MODEL

Suppose we have a fixed training set of m training examples, we train the model using mini-batch gradient descent. In detail, we define the overall cost function to be

$$J(W,b) = \frac{1}{m}\sum_{u=1}^{m}\frac{1}{2}\left\| h_{W_{Ls_l},b_{Ls_l}}(X^{(u)}) - X^{(u)} \right\|^2 + \frac{\lambda}{2}\sum_l \|W_l\|^2 \quad (7)$$

The first term of $J(W,b)$ is an average sum-of-squares error term, The second term is a weight decay term to help prevent over-fitting and decrease the magnitude of the weights. Our goal is to minimize the total cost function $J(W,b)$ as a function of W and b. To train the model, we initialize each parameter $W_{ij}^{(l)}$ and $b_i^{(l)}$ to a small random value near zero (Normal $(0,\varepsilon^2)$ distribution for some small ε, say *0.01*. Then apply algorithm 1 to train Autoencoder such as mini-batch gradient descent. The parameters *W,b* is updated by performing an update for every mini-batch of m training examples:

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b,X^{(u:u+m)}) \quad (8)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J\left(W, b, X^{(u:u+m)}\right) \tag{9}$$

where α is the learning rate. There are two reasons to use the mini-batch for updating the parameters: Firstly, reduces the variance of the parameter updates, lead to more stable convergence. Second, can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries and computing the gradient w.r.t. a mini-batch very efficient. To compute the partial derivatives above, we use the backpropagation algorithm to compute these partial derivatives.

$$\frac{\partial}{\partial W_{ij}^{(l)}} J\left(W, b, X^{(u:u+m)}\right) = \left[\frac{1}{b_1} \sum_{u=k}^{k+(b_1-1)} \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b, X^{(u)})\right] + \lambda W_{ij}^{(l)} \tag{10}$$

$$\frac{\partial}{\partial b_i^{(l)}} J\left(W, b, X^{(u:u+m)}\right) = \frac{1}{b_1} \sum_{u=k}^{k+(b_1-1)} \frac{\partial}{\partial b_i^{(l)}} J(W, b, X^{(u)}) \tag{11}$$

Where b_1 is mini-batch size.

Algorithm 1: One iteration of the mini-batch gradient descent algorithm.

Set $\Delta W^{(l)} := 0, \Delta b^{(l)} := 0$ for all l
Parameter: mini-batch size b_1
k=1;
Repeat {
u:=k;
Repeat {
- Compute $\nabla_{W^{(l)}} J(W, b, X^u)$ and $\nabla_{b^{(l)}} J(W, b, X^u)$
- Set $\Delta W^{(l)} \leftarrow \Delta W^{(l)} + \nabla_{W^{(l)}} J(W, b, X^u)$
- Set $\Delta b^{(l)} \leftarrow \Delta b^{(l)} + \nabla_{W^{(l)}} J(W, b, X^u)$

$u := u + 1;$
Until $u > (k + b_1)$
$k := k + b_1;$
Until $k > m$;
Update the weights and biases (5.8) and (5.9)

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \left[\frac{1}{b_1} \Delta W^{(l)} + \lambda W^{(l)}\right] \quad \text{using the formula (5.8)}$$

$$b_{ij}^{(l)} = b_{ij}^{(l)} - \alpha \left[\frac{1}{b_1} \Delta b^{(l)}\right] \quad \text{using the formula (5.9)}$$

Source: Author, (2026).

To make predictions, we give the training set of one user and a new query service *q*. The input layer for a specific unit represents the probability which service *q* will be predicted the QoS value *k*. As such the expected rating for item *q* is computed as

$$r_q = \sum_k k . \frac{f\left(\sum_{j=1}^{F} W_{ij}^{k(l)} a_j^{(l)} + b_i^{k(l)}\right)}{\sum_k f\left(\sum_{j=1}^{F} W_{ij}^{k(l)} a_j^{(l)} + b_i^{k(l)}\right)} \tag{12}$$

V. EXPERIMENTAL

V.1 DATA SETS DESCRIPTION

We have used public real world Web-service QoS data set for our experiments. This is WS-DREAM (*http://www.ws-dream*) data set, it is a Distributed Reliability Assessment Mechanism for Web Services and allowing users to carry out a reliable and quality assessment web service in a collaborative manner. We conducted experiments the algorithm from two different data sets in WS DREAM. *Dataset 1*[17]: The data set monitor 100 Web services by using 150 distributed computer nodes located all over the world. The obtained results are contained in 150 files, where each file includes 10,000 Web Services invocations on 100 Web services by a service user, there are totally more than 1.5 million Web service invocations and each line in the file is a web service invocation result. We randomly extract 100 files, each file includes 1000 Web service invocations on 100 Web services by a service user, there are totally more than 10.000.000 Web service invocations. *Dataset 2* [18]*:* Real-world QoS evaluation results from 142 users on 4,532 Web services on 64 different times. The obtained results include two files: Response-time (RtRate) and Throughput (TpRate), where each file has collected 4,532 Web services from public sources on the Web. 142 distributed computers from the Planet-Lab are employed for evaluating the QoS performance of Web services in 64 times intervals. Each of the dataset 1 and dataset 2 was divided into two parts. A training set (80%) and a test set (20%). We randomly removed entries 10% on the data set as missing QoS values.

V.2 EVALUATION METRIC

To evaluate the proposed model, we use both the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE) metrics to measure the prediction missing QOS value of our model. The evaluation rules are given by the following forms:

$$MAE = \frac{\sum_i [r_i - p_i]}{N} \tag{13}$$

$$RMSE = \sqrt{\frac{\sum_i (r_i - p_i)^2}{N}} \tag{14}$$

Where p_i is the standard deviations of the predicted values, r_i is real QoS value, N is the sum of the absolute difference over all pairs.

To prove the effectiveness of the proposed model, we compare our model with the following approaches:

***RBM* [19]** describe a class of two-layer undirected graphical models based on Restricted Boltzmann machines (RBM), it show that learning can be performed efficiently by using Contrastive Divergence (CD).
***AutoRec* [20]** This paper presents a novel autoencoder framework for CF, it is called AutoRec, the model is used backpropagation algorithm and fine-tuning by gradient descent.

V.3 EXPERIMENTAL RESULTS

We first set the parameters in the AutoMinibatch model with mini-batch size 300, number of hidden layers in the model (*Layer=150*), learning rate of *0,01* and tuned the regularisation strength $\lambda \in \{0.001; 0.01; 0.1; 10; 100\}$. For RBM, we train the RBM with $F = 100$, the weights were updated using a learning rate of *0.01*, momentum of *0.9* and CD learning was started with $T = 1$. For AutoRec, we tuned the regularization strength $\lambda \in \{0.001; 0.01; 0.1; 10; 100; 1000\}$ and the appropriate latent dimension $k \in \{10; 20; 40; 80; 100; 200; 300; 400; 500\}$. We selected training all models for between 200 and 300 passes (epochs) through the entire training two data sets. The values of these parameters were found above, empirically evaluating was found to yield quite good results. The RMSE and MAE values of all methods on both datasets are shown in Table 2. The results show that AutoRec have performance better RBM and AutoMinibatch obtains smaller MAE and RMSE values consistently, which indicates better prediction accuracy.

Table 2. Prediction quality on both datasets.

Methods	***Dataset 1***		***Dataset 2***			
			Response-time		*Throughput*	
	MAE	*RMSE*	*MAE*	*RMSE*	*MAE*	*RMSE*
RBM	0.3708	0.6111	0.3520	0.5839	0.4321	0.7583
AutoRec	0.3609	0.6028	0.3382	0.5812	0.4191	0.7335
AutoMinibatch	**0.3355**	**0.5780**	**0.3134**	**0.5591**	**0.4011**	**0.7190**

Source: Author, (2026).

Figure 2 and Figure 3 are the results of the models (RBM, autoRec and AutoMinibatch) on two datasets. The x-axis shows the number of epochs and the y-axis displays the MAE/RMSE of the models. We observed a faster rate of convergence for RBM when using the CD learning with T=2. This also shows that learning CD significantly improves performance of the RBM model. However, compared to AutoRec, RBM has slower convergence speed. Especially, AutoMinibatch has the fastest convergence speed, this indicates that AutoMinibatch give the lowest MAE, RMSE values on two datasets.

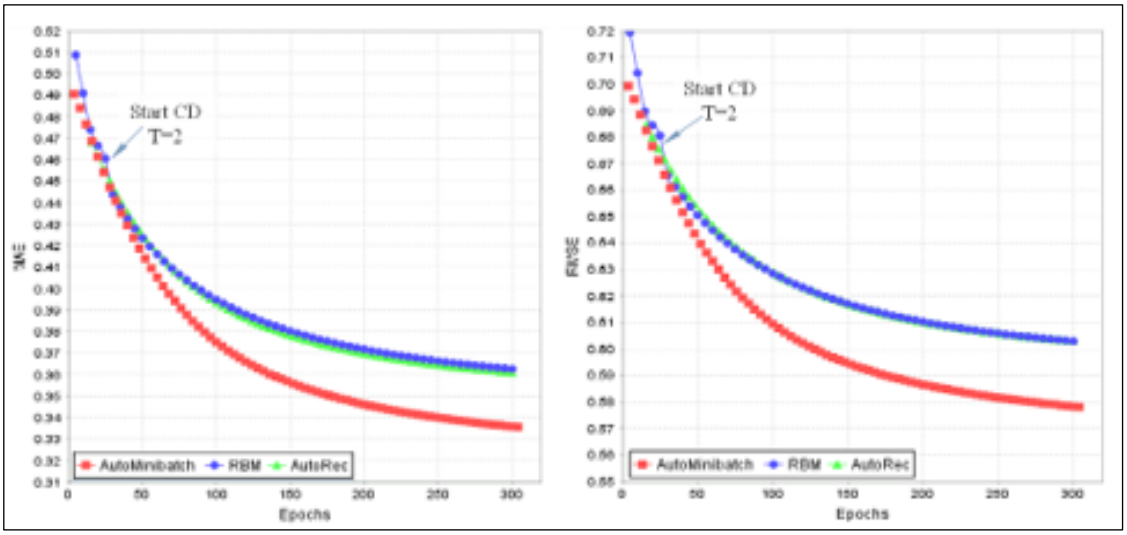

Figure 2: Performance of various models on dataset 1.
Source: Author, (2026).

The y-axis displays MAE or RMSE and the x-axis shows the number of epochs, or passes through the entire training dataset.

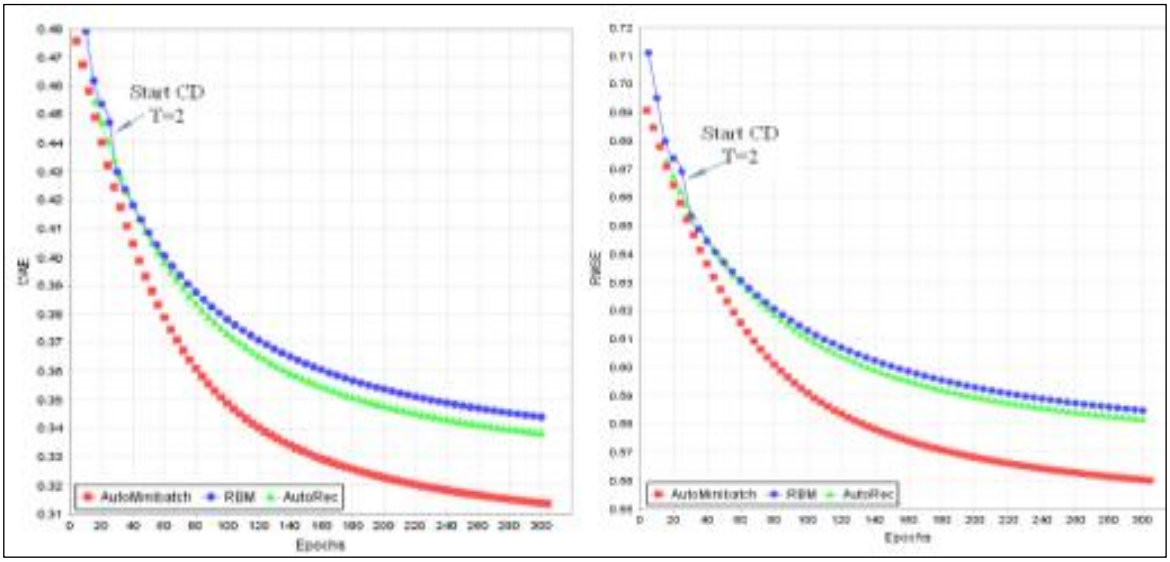

Figure 3: Performance of various models on dataset 2 (Response-time).
Source: Author, (2026).

The y-axis displays MAE or RMSE and the x-axis shows the number of epochs, or passes through the entire training dataset. On the other hand, AutoMinibatch model has the number of free parameters is very large. So, to find a satisfying combination of parameters for the model is not an easy task by the following reasons:

- First, all these parameters are involved in the curve.
- Second, sometimes they reinforce each other or destroy each other or even make it more complex.
- Finally, we must have the time required to evaluate a set of parameter settings, it is not easy to combine the effective parameters.

In that context, we presented a way to add three experiments below with the aim of this study is to clarify the individual effect of each parameter on the model.

Learning rate: We first study the influence of the learning rate for faster convergence, we set three different parameters for learning of $\gamma = (0.01; 0.001; 0.0001)$ and all remaining parameters like in the experiments above. Figure. 4 and Figure. 5 display the results of the learning rate on two datasets and they have a different impact on the convergence. If we set the learning values γ to be too small $\gamma = 0.0001$. The result is almost a straight line and it almost does not affect the learning model, therefore, this value will not be appropriate for the model. The results show that the value $\gamma = 0.01$ the convergence rate is faster than $\gamma = 0.001$, so, we can conclude that the $\gamma = 0.01$ offers best result of the model.

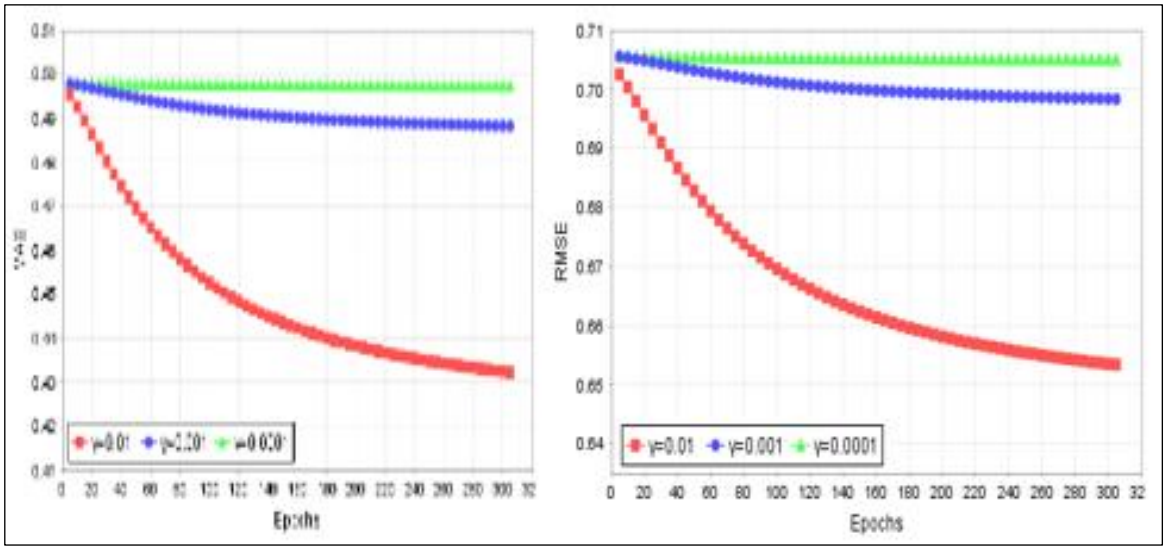

Figure 4: The effect of learning of various models on dataset 1.
Source: Author, (2026).

The y-axis displays MAE/RMSE and the x-axis shows the number of epochs, or passes through the entire training dataset

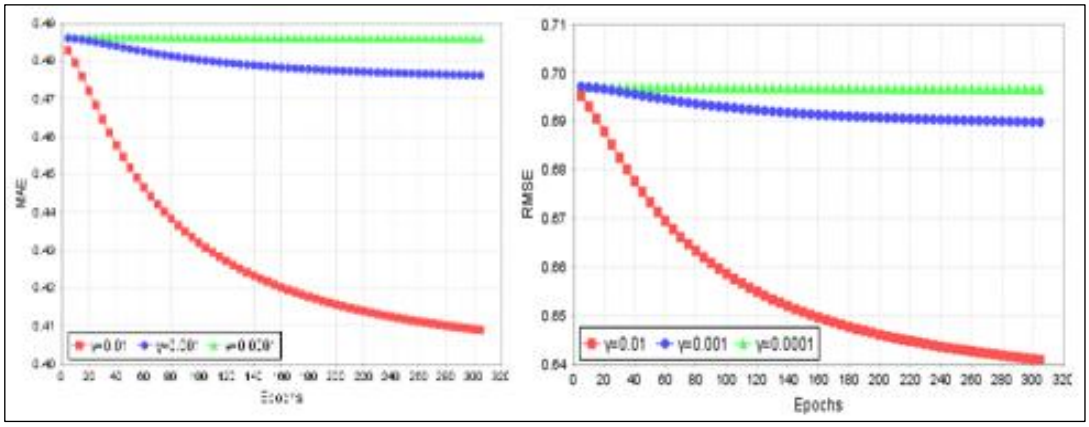

Figure 5: The effect of learning of various models on dataset 2 (Response-time).
Source: Author, (2026).

The y-axis displays MAE/RMSE and the x-axis shows the number of epochs, or passes through the entire training dataset.

The number of hidden layers: The determination of the number of layers and the number of nodes in each layer are a very important part in deciding model architecture. Therefore, we firstly examines how to determine them for effectiveness. There are two issues related to select the number of layers and the number of nodes in each layer: First, the hidden nodes are not directly interacting with the external environment, but they have a tremendous influence on the final output. Second, depending on the training data that we can choose the number of different hidden nodes, but if we choose too many hidden layers and hidden nodes, it can lead to over-fitting. To demonstrate the influence on the number of hidden layers and hidden nodes in each layer in the model, we choose three different values: 50 layers (Layer=50), 100 layers (*layer=100*) and 150 layers (*layer=150*) and each layer we choose 200 hidden nodes. The results which are illustrated in Figure. 6 and Figure. 7 show that there is a huge difference in the speed of convergence among various hidden layers. With *layer=150*, the convergence rate is the fastest.

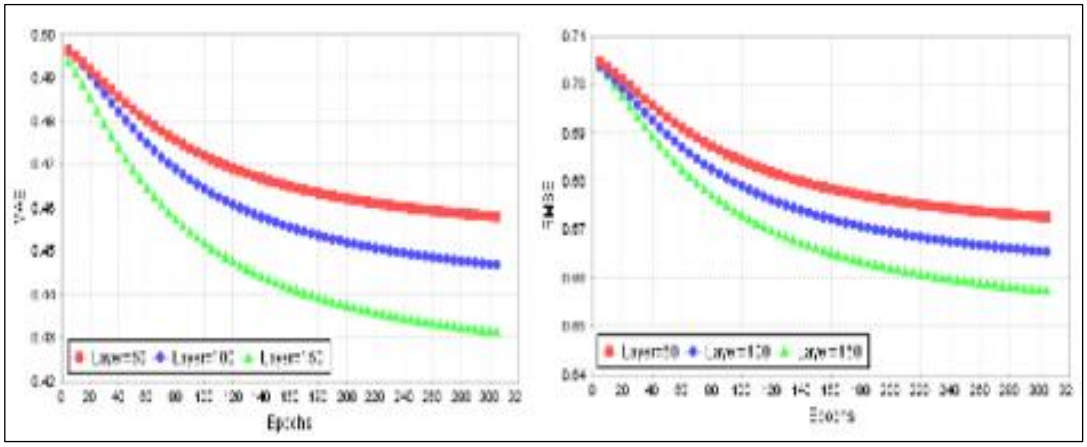

Figure 6: The effect of the number of hidden layers on dataset 1.
Source: Author, (2026).

The y-axis displays MAE/RMSE and the x-axis shows the number of epochs, or passes through the entire training dataset.

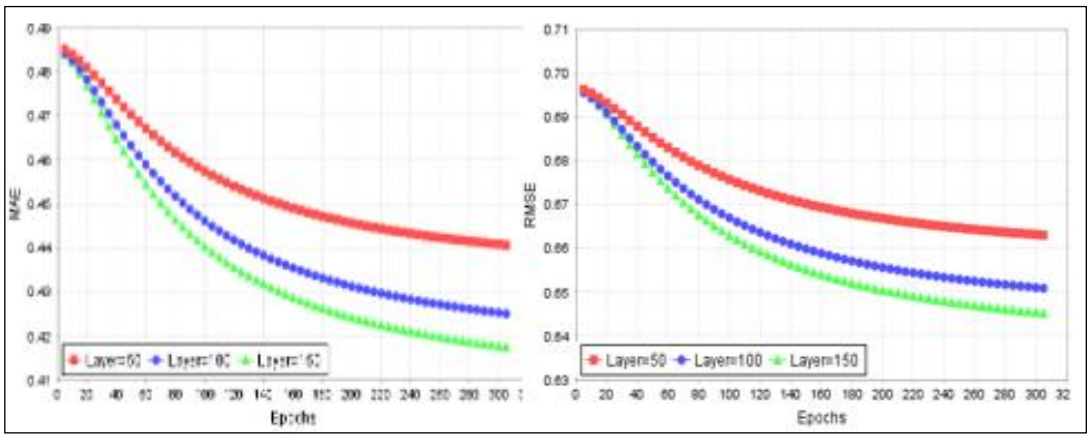

Figure 7: The effect of the number of hidden layers on dataset 2 (Response-time).
Source: Author, (2026).

The y-axis displays MAE/RMSE and the x-axis shows the number of epochs, or passes through the entire training dataset.

Mini-batch size: Mini-batch size will directly affect the number of times that update parameters in the model. To experiment, we also choose different mini-batch size as follows: *mini-batch size=100*, *mini-batch size=200* and *mini-batch size=300*. All other parameters as above experiment. The result in Figure 8 and Figure 9 show that all the mini-batch seem to be almost equal. But if we look carefully, we will see different results, in which *mini-batch size=300* has the best result. Basing on the experimental results above, we shown that the following parameters have the best convergence: Learning rate with $\gamma = 0.01$; The number of hidden layer is 150; and mini-batch size=300. However, depending on the datasets there will be different parameters.

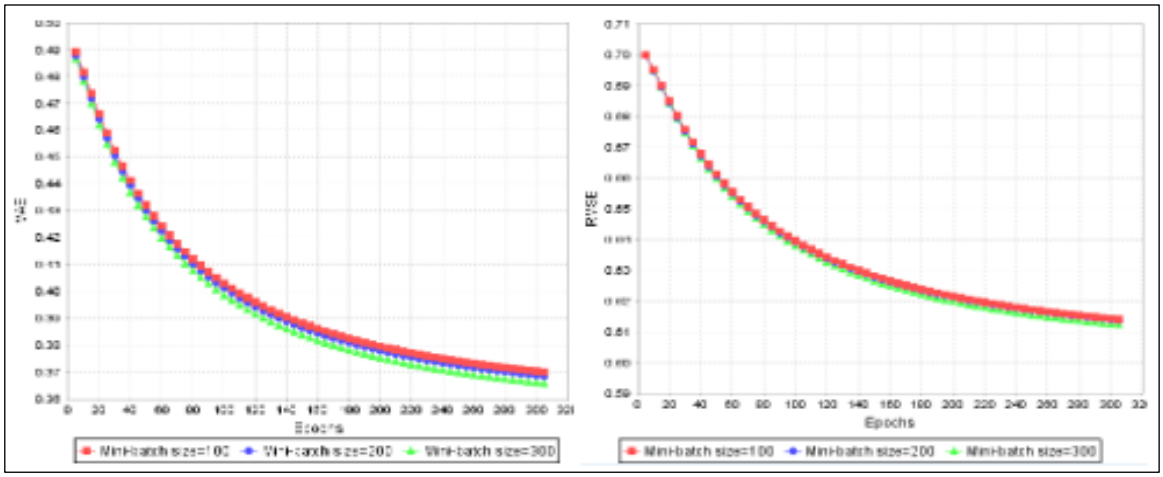

Figure 8: The effect of the mini-batch size on dataset 1.
Source: Author, (2026).

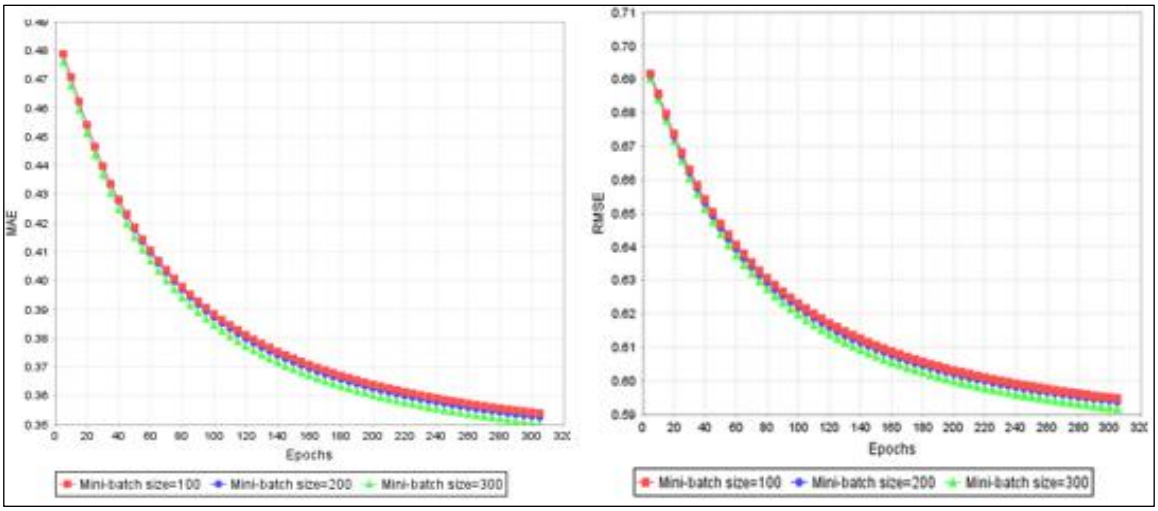

Figure 9: The effect of the mini-batch size on dataset 2 (Response-time).
Source: Author, (2026).

VI. CONCLUSION AND FUTURE WORK

In parallel with the development of e-commerce, service oriented systems have become one of the most dynamic research areas. The objective of the systems is to recommend QoS to users, but to identify whether a service provider is good or bad, it depends on many different factors. If we want to predict the Web service reliability for the current user or helping users find appropriate Web service providers, we'll force them to collect more detailed information about each user and maintain data for all users. On the other hand, the skyrocketing number of users and service providers lead to a huge volume of data. To contribute to the solving of the above problems, in this paper, we introduced a new model called AutoMinibatch with two goals: First, adding a new method for evaluating prediction the field of web service quality. Second, opening for predicting QoS on large data sets based on an autoencoder.

The model used backpropagation and fine-tuned by mini-batch gradient descent. Experimental results obtained are very satisfying and better than what we expected. AutoMinibatch has faster final convergence than RBM and AutoRec. In addition, we also demonstrate how to choose the parameters (Mini-batch size, the number of hidden layer and learning rate) for the best efficiency. The best results obtained from experiments: Mini-batch size=300, the number of hidden layer is 150 and learning rate is 0.01. In the future, we will orient some work that needs to be done here. First, We will finish building a system of collecting information of QoS values from different service users and predicting the missing QoS values for active users based on the Autominibatch model. Second, algorithms typically do not take into account any form of implicit feedback for making inferences. Normally this feedback information has only a small number of ratings in the training set, but it is a very useful source of information for the other users. So, the architecture of the AutoMinibatch model has been modified to take all this extra information into account and called a conditional Autominibatch model.

VII. AUTHOR'S CONTRIBUTION

Conceptualization: Le Van Thinh.
Methodology: Le Van Thinh.
Investigation: Le Van Thinh.
Discussion of results: Le Van Thinh..
Writing – **Original Draft**: Le Van Thinh.
Writing – **Review and Editing**: Le Van Thinh.
Approval of the final text: Le Van Thinh.

VIII. REFERENCES

[1] Serhani A, Dssouli R, Hafid A and Sahraoui H. "A QoS broker based architecture for efficient web services selection", Proceedings of the IEEE International Conference on Web Services, In ICWS, vol 1, 113-120, 2005.

[2] Geoffrey E.H. "A Practical Guide to Training Restricted Boltzmann Machines". Neural Networks: Tricks of the Trade Lecture Notes in Computer Science. 599-619, 2012.

[3] Hassan I, Chadi A, Mourad D and Sujoy R, "QoS-aware middleware for web services composition: a qualitative approach", Enterprise Information Systems, vol 3, 449-470. 2009.

[4] Zibin Z and Michael R L, "Collaborative Reliability Prediction for Service-Oriented Systems", In Proceedings of the ACM/IEEE 32nd International Conference on Software Engineering (ICSE2010), Cape Town, South Africa, 35-44, 2010.

[5] Zibin Z, Hao M, Michael R L and Irwin K, "Qos-A ware Web Service Recommendation by Collaborative Filtering", IEEE Transactions on services computing, 4(2), 13-120, 2011.

[6] Zibin Z and Michael R L, "Neighborhood-Based QoS Prediction of Web Services", QoS Management of Web Services Advanced Topics in Science and Technology in China, 41-64, 2013.

[7] Lou S and Rong J, "Flexible mixture model for collaborative filtering", Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), 704-711, 2003.

[8] John S B, David H and Carl K, "Empirical analysis of predictive algorithms for collaborative filtering", Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence, 43-52, 1998.

[9] Thomas H, "Collaborative filtering via guassian probabilistic latent semantic analysis", Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, 259-266, 2003.

[10] HongBing W, Xin C, Qin W, Qi Y, Zibin Z and Athman B, "Integrating On-policy Reinforcement Learning with Multi-agent Techniques for Adaptive Service Composition", 12th International Conference on Service Oriented Computing (ICSOC), 154-168, 2014.

[11] Jyri J. K and Christopher W. "Multiple Texture Boltzmann Machines", Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics(AISTATS), 638-646, 2012.

[12] Hogo L and Yoshou B, "Classification using discriminative restricted Boltzmann machines", Proceedings of the 25th international conference on Machine learning, ACM, 536-543, 2008.

[13] Nicolas L R, Nicolas H, Jamie S and John W. "Learning a generative model of images by factoring appearance and shape", Neural Computation 23, 593-650, 2011.

[14] Eric P X, Rong Y and Alexander G H, "Mining associated text and images with dual-wing harmoniums", In Conference on Uncertainty in Artificial Intelligence,633-64, 2005.

[15] Thomas H, "Collaborative filtering via guassian probabilistic latent semantic analysis", Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, 259-26, 2003.

[16] Geoffrey H and Terrence J.S, "Unsupervised Learning", Foundations of Neural Computation. Neural Networks. 19

[17] Jyri J. K and Christopher W, "Multiple Texture Boltzmann Machines", Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics(AISTATS), 638-645, 2012.

[18] Yilei Z, Zibin Z and Michael R L. WSPred, "A Time-Aware Personalized QoS Prediction Framework for Web Services", In Proceedings of the 22th IEEE Symposium on Software Reliability Engineering (ISSRE), 210-215, 2011.

[19] Salakhutdinov R, Mnih A, Hinton G, "Restricted boltzmann machines for collaborative filtering", In proceedings of the 24th International Conference on Machine Learning, 791–79, 2007.

[20] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, Lexing Xie, "AutoRec: Autoencoders Meet Collaborative Filtering", Proceedings of the 24th International Conference on World Wide Web. Pages 111-112, 2015.