An ELM-Based Privacy Preserving Protocol for Cloud Systems

Masato Hashimoto, Yuya Kaneda, Qiangfu Zhao
University of Aizu
Aizu-Wakamatsu, Fukushima 965-8580, Japan
Email: {m5201129, d8161108, qf-zhao}@u-aizu.ac.jp

***Abstract*—In this paper, we propose a privacy preserving protocol for cloud system utilization based on extreme learning machine (ELM). The purpose is to implement aware agents (A-agents) on portable/wearable computing devices (P/WCD). The proposed protocol is useful to reduce the calculation cost on the P/WCD. The basic idea of the protocol is to divide an ELM-based A-agent into two parts, one containing the weights of hidden layer(s) and the other containing the weights of the output layer. The former is implemented in the remote server and the latter is implemented in the P/WCD. In addition, the input data are first encrypted in the P/WCD using transposition cipher, and then sent to the server. Because the server can only "see" random weights and encrypted data, the user intention and privacy can be protected. In addition, since part of the computations is executed on the server, the cost for implementing A-agents in the P/WCD can be reduced. Experimental results on several public databases show that the proposed protocol is useful if the dimension of the input data is high.**

***Index Terms*—extreme learning machine, privacy preserving cloud-based computing, portable computing device.**

I. Introduction

Since around 2010, portable/wearable computing devices (P/WCD) like smartphones, have become more and more popular over the world. A variety of applications for P/WCDs has been developed and used by many people. The goal of our research is to implement high performance aware agents (A-agent) on P/WCDs. The A-agents are applications that can provide useful information and help the users to solve problems in their daily lives. Machine learning is often used for developing the A-agents. With the support of different A-agents, people can use the P/WCDs more easily. However, it is difficult to implement the A-agents in a P/WCD if they require too many computing resources. This is mainly because P/WCDs are usually powered by small batteries and their computing performance is lower than desk-top computers. That is, implementation of A-agents in P/WCDs poses a performance-cost dilemma.

To resolve the above dilemma, we may consider two methods. One is to develop methods that can design compact and high performance machine learning models, and the other is to use a cloud server for data processing. For the former method, we have proposed an algorithm called decision boundary making (DBM) [1]. In DBM, a compact model reproduces a decision boundary generated by a high performance model. Multi-layer perceptron (MLP) [2] is used as the compact model, and support vector machine (SVM) [3] is used as the high performance model. Using DBM, we can obtain compact and high performance models for A-agents and implement them in P/WCDs. The concept of "compactness", however, is relative. Compared with the SVM-based high performance model, the MLP-based model obtained using DBM can be smaller, but may not be small. That is, if the given problem is complex, we may not be able to obtain a model small enough for P/WCD implementation.

Using the second method, we can conduct all expensive computations in a cloud server, and the P/WCD is used only for data collection. However, if we put the A-agents in the cloud server, the server will be able to know the user intention easily. In addition, if we send all data to the server for processing, the user privacy will be visible to the server. That is, the second method is not "trust-able" to the users.

To solve the problem, in this paper we propose a privacy preserving protocol for cloud system utilization using extreme learning machine (ELM). ELM is a special type neural network proposed by Huang [4], [5]. In this study, we focus on that how to implement the ELM-based A-agent on P/WCD. The basic idea of the protocol is as follows. The neural network (NN) is divided into two parts: 1) the random weight matrix between input layer and hidden layer, and 2) the weight matrix between hidden layer and output layer. The hidden layer is calculated by a server, and the output layer is calculated by the P/WCD. So, the computation of NN has communication between the server and the client (P/WCD). Additionally, before sending an input datum to the server, it is encrypted using a transposition cipher. Therefore, the server can only see random weights in the hidden layer and the encrypted data, and will not be able to understand the privacy of the user.

In the training phase, the protocol generates n different keys (i.e., $2 \leq n \leq N_f!$ where N_f is the number of features) for the transposition cipher. The training datasets are encrypted using each key, and an ELM-NN is trained for each encrypted dataset. Here, the same random weight matrix can be shared by the hidden layers of all models. We only generate n output weight matrices. After training, the hidden weight matrix is saved in the server. The n keys and the corresponding output weight matrices are stored in the P/WCD. In the classification phase, the P/WCD sends the data to the server, that is encrypted by the k-th encryption key for transposition cipher. The number k is determined at random

$(1 \leq k \leq n)$. The server then calculates the hidden layer from the encrypted feature vector, and sends back the result to the P/WCD. Finally, the P/WCD calculates the output layer using the k-th output weights, and find the class label. Theoretically, this protocol can reduce calculation time of P/WCD if the computation cost in the hidden layer is large. In addition, the user's privacy information (the original data and the trained A-agent models) can be preserved without reducing the accuracy.

In the previous study about ELM on cloud system, Jiarun Lin et al, proposed a secure and practical outsourcing mechanism, named partitioned ELM [6]. Their paper focused on running the training process of ELM on a cloud server to reduce the training cost and make it fast. On the other hand, we focus on reducing classification cost on P/WCD.

The structure of this paper is as follows. Section II introduces detail of the privacy preserving protocol, and discusses its security. Section III, and Section IV provide the experimental results and discuss the performance of the protocol. Section V draws some conclusions and suggests some topics for future work.

II. PRIVACY PRESERVING PROTOCOL FOR CLOUD SYSTEM

A. Extreme Learning Machine

The ELM is one of machine learning algorithms based on single hidden layer feedforward neural networks. Training for ELM is faster than traditional gradient-based learning methods, because the weights of hidden layer for ELM need not be tuned. Output function of ELM with binary classification is shown below.

$$f(\boldsymbol{x}) = \mathrm{sign}(\mathbf{h}(\boldsymbol{x}) \cdot \boldsymbol{\beta}) \tag{1}$$

where

$$\mathbf{h}(\boldsymbol{x}) = \boldsymbol{G}(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b}) \tag{2}$$

$$\boldsymbol{G}(\boldsymbol{z}) = \begin{pmatrix} g(z_1) \\ \vdots \\ g(z_n) \end{pmatrix} \tag{3}$$

$$\boldsymbol{W} = \begin{pmatrix} w_{11} & \dots & w_{1N_f} \\ \vdots & \ddots & \vdots \\ w_{N_h 1} & \dots & w_{N_h N_f} \end{pmatrix} \qquad \boldsymbol{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_{N_h} \end{pmatrix} \tag{4}$$

Sigmoid function is often used as the activation function.

$$g(z) = \frac{1}{1 + \exp(-\lambda \cdot z)} \tag{5}$$

In the above equations, N_h is the number of hidden neurons, $\boldsymbol{W}$ is the weight matrix for the hidden layer, $\boldsymbol{b}$ is the bias vector, $\boldsymbol{\beta}$ is the output weight vector, and λ is a positive real number. Generally the ELM has high performance when N_h is large enough. In this study, we use the basic training algorithm of ELM proposed in [5].

B. Methodology

To implement ELMs on a P/WCD, normally we can use two methods. One is to conduct all computations in the P/WCD, and the other is to conduct all computations in the server. In the latter case, P/WCD is used only for data collection and for displaying the results. But these methods have problems. The former has running time and resource problems. Because the computation/energy resources of a P/WCD are less than normal computers, high performance A-agents that require high computational cost, cannot be implemented completely in the P/WCD. The latter has security and privacy problems. If we put the A-agents in the server, and send the original data for processing, the user privacy and intention cannot be protected. To solve the problems, we propose a privacy preserving protocol for cloud system using ELM. The proposed protocol can reduce the computation cost and at the same time, improve the trust-ability of the system.

The main points of proposed protocol are to divide the ELM-NN into two parts, and applying a transposition cipher. Fig. 1 shows the neural network diagram for this protocol. In the protocol, a feature vector (training datum) which is a plain text, is first encrypted using the transposition cipher. The transposition cipher is a basic algorithm for encryption. The idea of transposition cipher is permutation. When the feature vector for encryption is $\boldsymbol{v}$ and a key is defined $\boldsymbol{k} = (k_1, k_2, ..., k_{N_f})$ where N_f is dimension of $\boldsymbol{v}$, if $k_i = j$, the j-th element of $\boldsymbol{v}$ is used as the i-th element of the encrypted vector.

Fig. 2 and Fig.3 provide sequence flow diagrams of training phase and classification phase for the protocol.

The steps of training phase are as follows.

T.1 Generate n keys ($2 \leq n \leq N_f!$) for transposition cipher (the key length equals to N_f).

T.2 Encrypt training data using each key by the transposition cipher and create n different encrypted training datasets.

T.3 Train n different ELMs using the encrypted data.

T.4 Save the weight matrix $\boldsymbol{W}$ for hidden layer and the bias $\boldsymbol{b}$ to the cloud sever, and save $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, ..., \boldsymbol{\beta}_n$ and key 1, key 2, ... , key n to P/WCD.

In T.3, the weight matrix $\boldsymbol{W}$ and bias $\boldsymbol{b}$ can be shared by all models, different matrices are not needed, because the weights are random numbers. Fig.4 shows the diagram for training phase. In this study, we assume that training is conducted in a personal computer (PC).

The steps for classification are as follows:

C.1 In P/WCD, encrypt a feature vector $\boldsymbol{x}$ using transposition cipher with key k, and k is defined at random ($1 \leq k \leq n$).

$$\boldsymbol{x}' = transposition(\boldsymbol{x}, k) \tag{6}$$

C.2 Send the encrypted feature vector $\boldsymbol{x}'$ from the P/WCD to the server.

C.3 Calculate hidden neurons in the server.
C.4 Send the hidden neurons from the server to the P/WCD.
C.5 Calculate output neurons using β_k on the P/WCD, and output class label.

The flow for classification is shown in Fig. 5. The proposed protocol can reduce calculation time of P/WCD, and protect the user's privacy (feature vector, trained ELM model, and calculation) without decreasing accuracy of ELM.

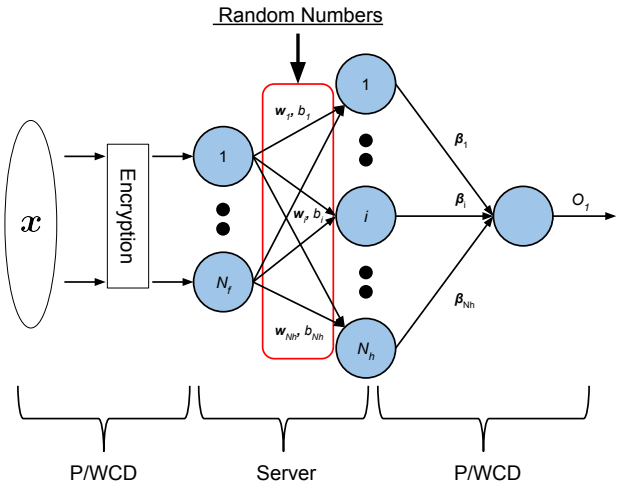

Fig. 1. How to separate calculations of ELM with encryption

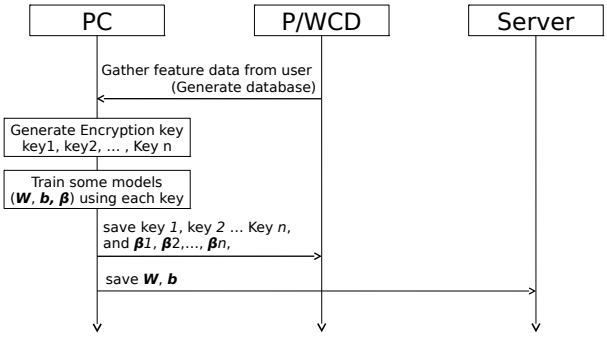

Fig. 2. Sequence flow of training phase

C. Discussion on Security

The length of keys depends on N_f which is the number of features. So the encryption space can be $N_f!$. It is assumed that the dataset whose N_f is large enough, is suitable for this method. When a malicious third person tries to find the original feature vector using brute-force attack, he/she needs to check at most $N_f!$ patterns. The plain text (feature vector) is a sequence of real numbers. So there is no way to distinguish the correct or not, if the third person finds a deciphered feature vector.

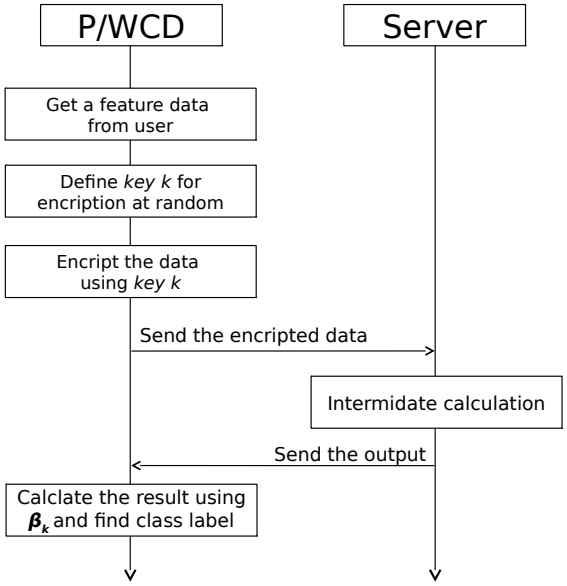

Fig. 3. Sequence flow of classification phase

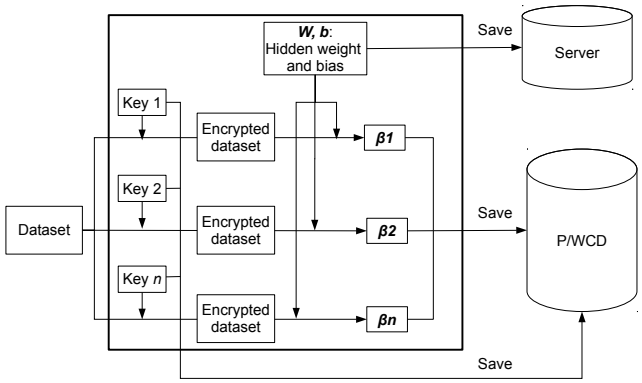

Fig. 4. Flow of training phase of the secure protocol

Additionally, the proposed protocol makes it difficult for the third person to find the meaning of the data, if they are normalized before the training. Because the range of the data can be $[-1, 1]$. Data normalization is basic pre-processing technique for improving classification performance. Therefore, this process can be not only for security but also performance.

For classification phase, the data sent from P/WCD to the server are encrypted (C.2), and all elements of $\boldsymbol{W}$ on the server are random numbers (C.3). The results of calculation on the server are also random data (C.4). Hence the data through the network and the data stored in server, are only encrypted data or random data. Moreover, the key for encryption was switched at random when classification (C.1). Therefore, data analysis can be difficult (except user), if the network transmission and server calculation are published.

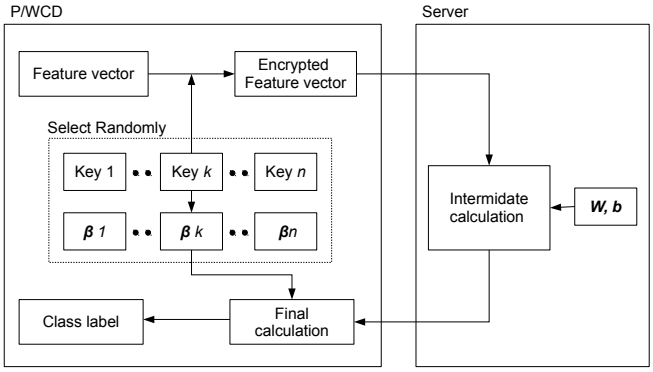

Fig. 5. Flow of classification phase of the secure protocol

There is a point that we need to improve in the future. In T.4, the keys and $\boldsymbol{\beta}$ are saved to P/WCD. However, it will not be safe if the keys or the $\boldsymbol{\beta}$ are leaked. Because, the third person can decipher the encrypted data using leaked keys, and analyze the user's privacy information from the feature vector. We need to protect them especially using another encryption or security technology, this is one of our future work.

III. Experiment

A. overview

The privacy preserving protocol proposed in our study is tested using datasets that are published on UCI Machine Learning Repository [7] and mldata.org [8]. They are shown in Table I.

TABLE I
Dataset features

	class (N_c)	Features (N_f)	Data Count(N_d)
Iris	3	4	150
Australian	2	14	690
Satimage	6	36	6,435
USPS	10	256	9,298

We measured classification time in each dataset. The classification time is defined as the time between the input (feature vector) and the output (class label). The purpose of this experiment is confirmation of reducing calculation time on P/WCD, and find suitable situations.

As for P/WCD, an Android smartphone was used. The network for this experiment was structured on one Android smartphone, one server, and one router. The router and smartphone are connected with Wi-Fi, and the router and the server are connected with LAN cable. NEC PA-WG1800HP2 was used as the router for this experiment. Specifications of the smartphone and the server are provided in Table II and Table III, respectively.

Activation function was sigmoid function. The number of hidden neurons were 100, 500, and 1,000. We measured the classification time of each dataset for the following two cases.

- All P/WCD: All classification computation of ELM in P/WCD, without encryption.

TABLE II
P/WCD Smartphone Specs and Environments

Machine	Google Nexus6
OS	Android 6.0.1 ART VM
CPU	Qualcomm 2.7GHz quad-core krait 450
Memory	3 GB
Wi-Fi	IEEE802.11b/g/n

TABLE III
Server Specs and Environments

Machine	Dell Precision-WorkStation-T3400
OS	Ubuntu 12.04
CPU	Intel Core2 Duo E8500 (3.16GHz)
Memory	4GB

- Protocol: Proposed privacy preserving protocol.

ELM models ($\boldsymbol{W}, \boldsymbol{b}$, some $\boldsymbol{\beta}$s) and keys were generated and dumped to files. The ELM was implemented in Python 3.5 with Numpy 1.10.4 [9], Scipy 0.17.0 [10] and Scikit-learn 0.17.1 [11]. To implement calculation of the P/WCD, we used Java, which is a basic programming language for developing Android applications.

B. Implementation of All P/WCD

The files for trained weights and keys were serialized to Java object and saved to the device. Also, the datasets were stored in device as serialized Java object. Fig. 7 shows the pseudocode for All P/WCD classification. It does not encrypt the data because we do not need the encryption if we calculate all only on P/WCD. The range of time measurement is between line 3 - line 6 in Fig. 6.

1: Observe a feature vector $\boldsymbol{x}$ from user
(feature vector from dataset in this experiment)
2: $\boldsymbol{x} \Leftarrow$ Normalize the feature vector $\boldsymbol{x}$
3: $\boldsymbol{W}$, $\boldsymbol{b} \Leftarrow$ Load hidden weight and bias
4: $\boldsymbol{\beta} \Leftarrow$ Load output weight
5: $\boldsymbol{H} \Leftarrow \mathrm{G}(\boldsymbol{W} \cdot \boldsymbol{x} + \boldsymbol{b})$
6: label $\Leftarrow \mathrm{sign}(\boldsymbol{H} \cdot \boldsymbol{\beta})$

Fig. 6. Classification pseudocode for all classification on P/WCD

C. Implementation of Protocol

We implemented the system as REST API. The communication method between the P/WCD (Android smartphone) and the server was HTTP. And the feature vector was serialized to JSON for communication.

1) P/WCD client: Fig. 7 shows the pseudocode for P/WCD client. We used OkHttp 2.4.0 [12] as HTTP communication and Jackson 2.6.0-rc3 for JSON serialization on Android. OkHttp and Jackson are well-known library for developing Android applications. The number of keys n was set to 5 in this experiment. The range of time measurement is between line 3 - line 9 in Fig. 7.

1: Observe a feature vector $\boldsymbol{x}$ from user
(feature vector from dataset in this experiment)
2: $\boldsymbol{x} \Leftarrow$ Normalize the feature vector $\boldsymbol{x}$
3: $k \Leftarrow$ Get a random value from $[1, n]$
4: key $\Leftarrow$ Load k-th key
5: $\boldsymbol{\beta} \Leftarrow$ Load k-th output weight
6: $\boldsymbol{x}' \Leftarrow$ Encrypt the feature vector $\boldsymbol{x}$ by the key
7: Send $\boldsymbol{x}'$ to the server
8: $\boldsymbol{H} \Leftarrow$ Receive a response from the server
9: label $\Leftarrow$ sign($\boldsymbol{H} \cdot \boldsymbol{\beta}$)

Fig. 7. Classification pseudocode for P/WCD client

2) Cloud server: The server was implemented in bottle 0.12.9 [13], which is a web framework for Python. The files for hidden weights and biases were serialized to Python objects using pickle library, and saved to the server, The process is only calculation of input vector, random weight matrix and random bias. Fig. 8 shows the pseudocode for cloud server (API server). This program runs when the server get a request from the P/WCD.

1: Get $\boldsymbol{x}'$ with a post request from client
2: $\boldsymbol{W}$, $\boldsymbol{b} \Leftarrow$ Load hidden weight and bias.
3: $\boldsymbol{H} \Leftarrow \boldsymbol{G}(\boldsymbol{W} \cdot \boldsymbol{x}' + \boldsymbol{b})$
4: return $\boldsymbol{H}$ to P/WCD client

Fig. 8. Classification pseudocode for Server

IV. Result and Discussion

First, we confirmed the accuracy of ELM to investigate how many hidden neurons are needed to get enough accuracy score. Table IV shows the averaged accuracy of 10 times 5-fold cross validation for three different N_h values.

TABLE IV
Accuracy of ELM (%)

	N_h		
	100	500	1,000
Iris	80.4	**82.2**	80.6
Australian	**85.2**	60.6	54.4
Satimage	87.2	91.2	**93.0**
USPS	88.4	93.7	**94.9**

Bold-faced numbers are the highest scores in each dataset. From these results, we found that proper N_h is needed for each dataset to get a high accuracy score.

Fig. 9 through Fig. 12 show the classification time for each dataset. The vertical axis of each graph indicates the classification time (second), and the horizontal axis is the number of hidden neurons N_h. Each bar shows the average of classification time of 30 trials. The error bars represent the standard deviations.

It is shown that with larger N_f or N_h takes longer time. In Iris dataset which has relatively small N_f, all P/WCD method can classify faster than the proposed protocol when $N_h = 100$.

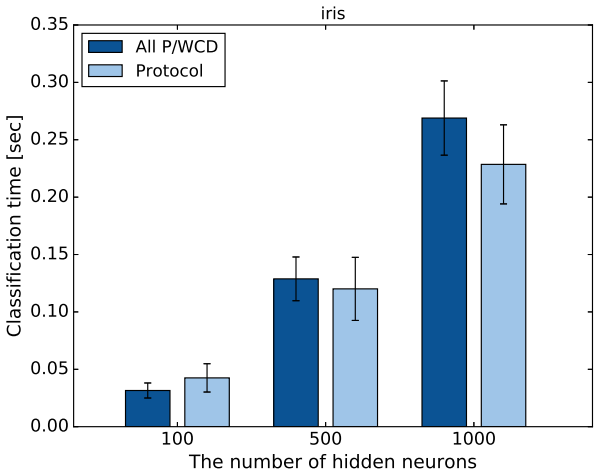

Fig. 9. Classification time of Iris dataset

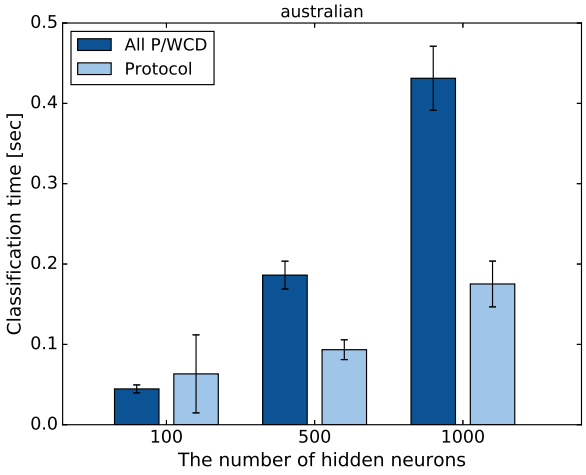

Fig. 10. Classification time of Australian dataset

This is because loading the weights and calculation of the hidden layer, are faster than encryption and communication. But in other N_h or datasets, the proposed protocol can classify faster. From Eq. (4), the hidden weight matrix size depends on N_f and N_h. So, the loading and calculation time also depend on N_f and N_h. Hence the protocol is unsuitable for small N_h or small N_f like Iris dataset.

Also, in Australian and Iris dataset, there was no big difference of classification time between the proposed protocol and all P/WCD method. But it has big difference in the result of Satimage and USPS datasets that has large N_f. The data with large N_f requires longer time for loading weight matrices and classification. Additionally the classification of data with large N_f are more secure because of transposition cipher in the proposed protocol. Therefore the proposed protocol is suitable for dataset having large N_f. From these results, it was confirmed that the proposed protocol can reduce classification time for A-agent on P/WCD.

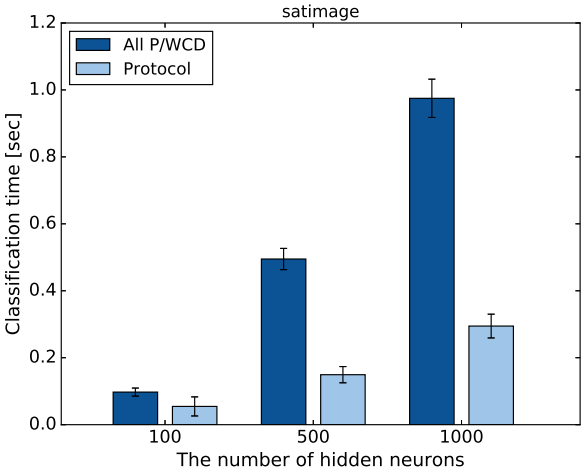

Fig. 11. Classification time of Satimage dataset

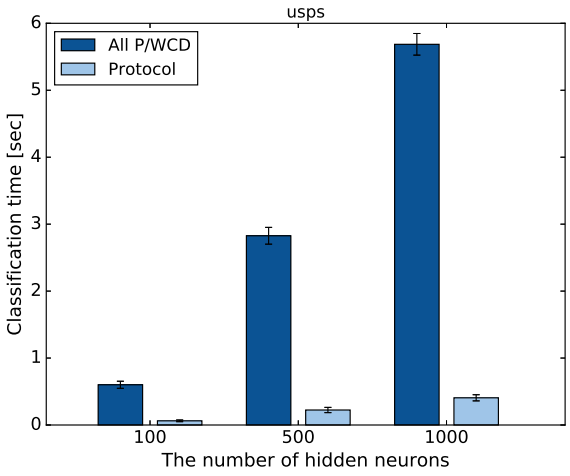

Fig. 12. Classification time of USPS dataset

V. Conclusion

In this paper, we proposed a privacy preserving protocol for cloud system using ELM to implement A-agent on P/WCD, and investigate (the classification times) of the protocol with other methods. In the protocol, the calculation of the server uses only random matrix and encrypted vector. Therefore, the user's privacy (feature vector and models), and the computation intention can be protected. From the experimental result, the proposed protocol can classify faster if the number of features or the number of hidden neurons is large. For implementing high performance A-agent on P/WCD, the proposed protocol can be one solution to reduce calculation time of the A-agent on P/WCD. However, it will not be safe if the keys or the output weight matrices are leaked. For future work, we would like to find a method to protect the keys and the output weights stored in the P/WCD. We would also like to use some public cloud server for high performance classification using deep learning or ensemble learning.

References

[1] Yuya Kaneda and Qiangfu Zhao. "Inducing high performance and compact neural networks based on decision boundary making". In: *The transactions of the Institute of Electrical Engineers of Japan. C* 134.9 (2014), pp. 1299–1309.

[2] DE Rumelhart GE Hinton RJ Williams and GE Hinton. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536.

[3] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[4] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. "Extreme learning machine: theory and applications". In: *Neurocomputing* 70.1 (2006), pp. 489–501.

[5] Guang-Bin Huang et al. "Extreme learning machine for regression and multiclass classification". In: *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 42.2 (2012), pp. 513–529.

[6] Jiarun Lin et al. "A secure and practical mechanism of outsourcing extreme learning machine in cloud computing". In: *IEEE Intelligent Systems* 28.6 (2013), pp. 35–38.

[7] M. Lichman. *UCI Machine Learning Repository*. http://archive.ics.uci.edu/ml. 2013.

[8] *mldata.org*. URL: http://mldata.org/.

[9] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. "The NumPy array: a structure for efficient numerical computation". In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30.

[10] Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. [Online; accessed 2016-04-05]. 2001–. URL: http://www.scipy.org/.

[11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[12] Inc Square. *OkHttp*. http://square.github.io/okhttp/‘. 2014.

[13] Marcel Hellkamp. *bottle*. http://bottlepy.org/. 2014.