A MapReduce Implementation of the Genetic-Based ANN Classifier for Diagnosing Students with Learning Disabilities

Tung-Kuang Wu[1], Shian-Chang Huang[2], Hsiu-Ting Kao[3], Hsu Chang[4]
Dept. of Information Management, NCUE
Changhua City, Taiwan
[1]tkwu@im.ncue.edu.tw, [2]shhuang@cc.ncue.edu.tw, [3]b9456005@gmail.com, [4]zx1986@gmail.com

Ying-Ru Meng
Dept. of Special Education, NHCUE
HsinChu City, Taiwan
myr321@mail.nhcue.edu.tw

***Abstract*—Diagnosis of students with learning disabilities (LD) is a difficult procedure that requires extensive man power and takes a long time. Fortunately, through genetic-based (GA) parameters optimization, artificial neural network (ANN) classifier may be a good alternative to the above procedure. However, GA-based ANN model construction is computation-intensive and may take quite a while to process. Accordingly, parallel processing such as multi-core programming and grid computing have been used to speedup the process. In this study, we setup a Hadoop min-cloud environment with virtualized hosts so that we may take full advantage of the current multi-core CPU technology. The GA-based ANN LD classifier is then re-programmed based on the MapReduce programming model and ported to this mini-cloud environment. Some implementation issues and considerations regarding the process will be discussed in the paper. Although the preliminary results may not show significant breakthrough over our previous studies, yet we do gain some experience through this process and see the potential of the MapReduce model in our future applications.**

Keywords-learning disabilities; MapReduce; neural network; virtualization; cloud computing

I. INTRODUCTION

The term "learning disabilities" (LD) was first used in 1963 [1]. However, experts in this field have not yet completely reach an agreement on the definition of LDs and its exact meaning [2]. In fact, a person can be of average or above average intelligence, without having any major sensory problems (like blindness or hearing impairment), and yet struggles to keep up with people of the same age in learning and regular functioning. Due to such implicit characteristics of learning disabilities, the identification of students with LDs has long been a difficult and time-consuming process. In the United States, the so called "Discrepancy Model" [3], which states that a severe discrepancy between intellectual ability and academic achievement has to exist in one or more of these academic areas: (1) oral expression, (2) listening comprehension (3) written expression (4) basic reading skills (5) reading comprehension (6) mathematics calculation, is one of the commonly adopted criteria to evaluate whether a student is eligible for special education services.

In Taiwan, the diagnosis procedure pretty much follows the "Discrepancy Model". The sources of input parameters required in such prolonged process include information from parents, general education teachers, students' academic performance and a number of standard achievement and IQ tests. To guarantee collection of required information regarding students suspected with LD, usually checklists of some kind are developed to assist parents and regular education teachers. The Learning Characteristics Checklists (LCC), a Taiwan locally developed LD screening checklist [4], is commonly used in most counties of Taiwan. Among the standard tests, the Wechsler Intelligence Scale for Children, Third or Fourth Edition (WISC III or IV) plays the most important role in this LD diagnosis model. WISC-III consists of 13 sub tests [5]. The scores of the sub-tests are then used to derive 3 IQs, which include Full scale IQ (FIQ), Verbal IQ (VIQ), Performance IQ (PIQ), and 4 indexes, which include Verbal Comprehension Index (VCI), Perceptual Organization Index (POI), Freedom from Distractibility Index (FDI), Processing Speed Index (PSI). There are also a number of locally developed standard achievement tests (AT), which typical consist of reading, math, and fields that are related to students' academic achievement.

Diagnosis of students with LDs then involves mainly interpreting the standard test scores and comparing them to the norms that are derived from statistical method. As an example, in case the difference between VIQ and PIQ is greater than 15, representing significant discrepancy between a student's cultural knowledge, verbal ability, etc, and his/her ability in recognizing familiar items, interpreting action as depicted by pictures, etc, is a strong indicator in differentiating between students with or without LD [5]. A number of similar indicators together with the students' academic records and descriptive data (if there is any) are then used as the basis for the final decision. Confirmed possible LD students are then evaluated for one year before admitting to special education. However, it is important to note that a previous study reveals that the certainty in predicting whether a student is having a LD using each one of the currently available predictors is in fact less than 50% [6].

The above identification procedure involves extensive manpower and resources. Furthermore, a lack of nationally regulated standard for the LD diagnosis procedure and criteria result in possible variations on the outcomes of diagnosis. In some cases, the difference can be quite significant [7].

With the advance in artificial intelligence (AI) and its successful applications to various classification problems, it is interesting to investigate how these AI-based techniques perform in identifying students with LDs. In our previous study, we have shown that ANN classifier does well in positively identifying students with LDs [7]. In subsequent studies, we combined various feature selection techniques and genetic-based parameters optimization with the ANN classifier, which further improves the overall identification accuracy [8]. However, although ANN-based classifier performs well in LD diagnosis problem, the procedure is computation-intensive and may take quite a while to process. Accordingly, multi-threaded programming and grid-based parallel computing (a parallel distributed genetic algorithm based implementation, will be referred to as PDGA hereafter) have been used to speedup the ANN model training and validation [9, 10, 11].

In this paper, we still focus on the ANN classification model and work on porting the GA-based ANN classifier to the MapReduce programming model. To fit into the new programming model, we have done a number of modifications of the PDGA procedure. The rest of the paper is organized as follows. Section 2 briefly describes the history of applying AI techniques to special education and gives a short introduction to Hadoop related terms that are used in our implementation. Sections 3 and 4 present our experiment settings, design and corresponding results. Finally, Section 5 gives a brief summary of the paper and lists issues that deserve further investigation.

II. Related Work

Artificial intelligence techniques have long been applied to special education. However, most attempts occurred more than one or two decades ago and mainly focused on using the expert systems to assist special education in various ways [7]. There were also numerous classification techniques other than neural networks that were developed and widely used in various applications [12]. Among all the classification techniques, artificial neural networks (ANN) have received lots of attentions due to their demonstrated performance and have gained wide acceptance [13].

An artificial neural network is a mathematical representation that is inspired by the way the brain processes information. Many types of ANN models have been suggested in literature, with the most popular one for classification being the multilayer perceptron (MLP) with back propagation. The goal of this type of network is to create a model that correctly maps the input to the output using historical data so that the model can then be used to predict the outcome when the desired output is unknown. MLP with back propagation is typically composed of an input layer, one or more hidden layers and an output layer, each consisting of several neurons. Each neuron processes its inputs and generates one output value that is transmitted to the neurons in the subsequent layer. Fig. 1 provides an example of an MLP with one hidden layer and one output neuron.

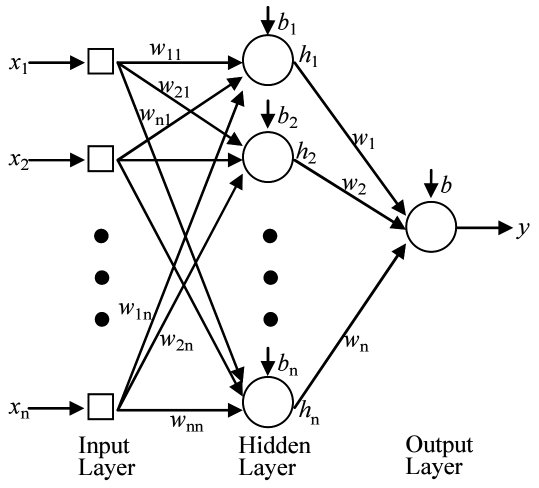

Figure 1. MLP with one hidden layer.

The output of i-th hidden neuron is computed by processing the weighted inputs and its bias term b_i as follows:

$$h_i = f^h\left(b_i + \sum_{j=1}^{n} w_{ij}x_j\right) \tag{2}$$

where w_{ij} denotes the weight connecting input x_j to hidden unit h_i. Similarly, the output of the output layer is computed as follows:

$$y = f^{output}\left(b + \sum_{j=1}^{n} w_j x_j\right) \tag{3}$$

with n being the number of hidden neurons and w_j represents the weight connecting hidden unit j to the output neuron. A threshold function is then applied to map the network output y to a classification label. The transfer functions f^h and f^{output} allow the network to model non-linear relationships in the data. Also note that the number of hidden layer nodes does not need to be the same as the number of input nodes.

The training of a neural network is the process of presenting the network with sample data and modifying the weights to approximate the desired function. In particular, an epoch indicates one iteration through the process of providing the network with a sample input and updating the network's weights. Let N_i, N_h and N_o respectively represent input feature size, number of hidden and output nodes, the total order of complexity is then $O(N_i \times N_h \times N_o + N_h \times N_o)$ for one epoch [14]. Since a typical ANN training process usually takes 500 epochs, the computation complexity for training of an ANN model is roughly equal to $500 \times N \times O(N_i \times N_h \times N_o + N_h \times N_o)$, where N represents the size of input samples for training.

In the field of special education, ANN has been used in a number of applications [7]. To improve the ANN classification accuracy, genetic algorithms have been used in the training and constructing of ANN model [15]. However, the GA optimization procedure may require numerous applications of the above ANN training process (depending on the number of chromosomes and evolution generations), and thus usually takes quite a long time to process [7]. Accordingly, researches have been applying parallel processing, which may provide affordable computational power, to speedup the time-consuming process [16]. For network connected cluster or grid environment, message passing interface (MPI) is usually used to coordinate computing nodes for completing a common task. On the other hand, to take full advantage of the currently available multi-core processor technology, OpenMP may be used explicitly to direct multi-threaded, shared memory parallelism [9].

With the advance of the cloud computing, a number of distributed computational models have also been developed. Among them, the MapReduce, together with GFS and GigTable were developed by Google in 2003. MapReduce is a programming model for large-scale data processing problems, which may separate the original problem from the details of parallelization. However, other than the related documents and algorithms, Google did not release their source codes. Fortunately, Hadoop, developed by Apache foundation that originally includes HDFS, HBase and MapReduce, is an open-source alternative for Google's implementation [17].

HDFS (Hadoop Distributed File System) is designed to operate upon low-cost hardware with high fault-tolerance and provide high throughput access to applications that have massive data sets. An HDFS cluster operates in a master-slave setup consisting of a name-node (master) and a varying number of data-nodes (slaves). The name-node maintains the metadata for all the files and directories in the file system. It also knows the data-nodes on which all the blocks for a given file are located [17].

MapReduce, operating upon the HDFS, is a distributed programming model that may work on a cluster of tremendous computational nodes and is suitable for processing problems with massive data sets. In MapReduce programming model, a computation is specified by two functions: Map and Reduce. The underlying MapReduce library then proceeds to parallelize the computation, while hiding issues such as data distribution, load balancing and fault tolerance from the programmers. Accordingly, MapReduce programmers may thus be able to concentrate on the programming logic in solving the problems.

A MapReduce job, which consists of input data, MapReduce program, and configuration information, is divided into map and reduce tasks. The job-tracker and a varying number of task-trackers control the job execution process, with the job-tracker coordinating all the jobs on the system and the task-trackers running tasks and sends job progress to the job-tracker [17]. The configuration of various roles in a Hadoop cluster environment can be shown in Fig. 2. As can be seen, the master can be a job-tracker / name-node and a task-tracker / data-node at the same time, while a slave can only be a task-tracker and a data-node.

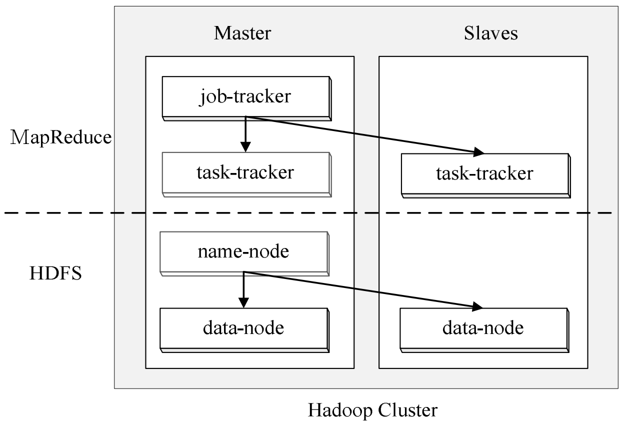

Figure 2. Various roles of Hadoop cluster nodes (revised from [18]).

In this study, we will work on porting the GA-based ANN classifier for LD identification [9] to the emerging cloud computing paradigm.

III. Environment Setup and Implementation Issues

A virtualized 12-node mini-cloud environment, established on top of 2 multi-cores PCs running Ubuntu server, is set up for the experiment. The hardware details of the PCs and the mini-cloud setup are shown in Table I and Fig. 3. Note, virtualization (through kernel-based virtual machine: KVM) is adopted in this study so that we may take full advantage of the current multi-core CPU technology.

TABLE I. Hardware details of the PCs in our study

	CPU	No. of cores	Memory
PC 0	Intel (R) Core (TM) i7 (2.7 GHz)	4 physical cores (8 logical cores)	12 GB
PC 1	AMD Phenom (TM) II (3.3 GHz)	6 physical cores	8 GB

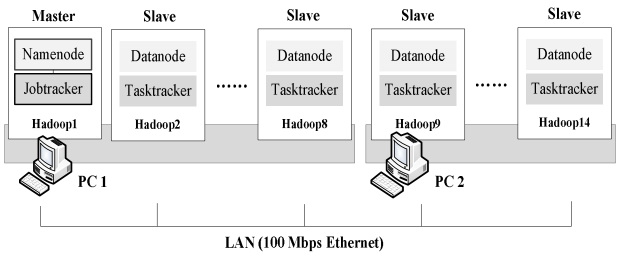

Figure 3. The mini-cloud setup in our study.

To map the regular genetic algorithm to the MapReduce model, we re-arrange the order of the GA procedure as shown in Fig. 4. The most computation-intensive step, which would be the fitness function calculation (ANN model construction and validation), is implemented in the Map stage, while the other GA processes such as selection, cross-over, and mutation are organized in the Reduce stage. Note there is only one Reducer in our implementation, which means only the most computation-intensive fitness function is parallelized while the GA processes are executed sequentially. Although this may somewhat degrade the overall performance (in terms of execution time), yet the implementation is much simpler and we may also avoid the GA procedure converging to some local maxima when each reduce task is distributed with too few chromosomes in a multiple Reducers setup [10]. Furthermore, as our input data is much smaller than the HDFS block size (64 MB), we manually split the input data for each map task to avoid potential overhead in managing the splits and map task creation when it is done by Hadoop [17].

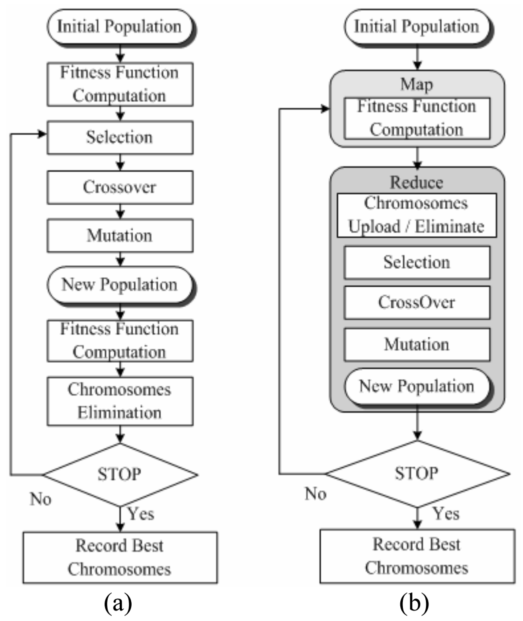

Figure 4. (a) a regular GA operation process that we adopted in our earlier PDGA implementation [9, 10, 11], and (b) its mapping to the corresponding MapReduce programming model.

In addition, as shown in Fig. 5, in each generation the reduce task would preserve at most *N* best chromosomes in the HDFS.

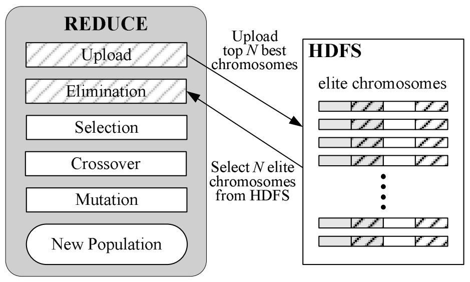

Figure 5. Elite chromosomes preservation and distribution.

Note, these *N* chromosomes also have to be better, in terms of accuracy, than a threshold value (which would be the average of all the elite chromosomes stored in HDFS) to be preserved. Those accumulated elite chromosomes may later be randomly selected to replace the *N* worst chromosomes in consecutive generations. In all of our experiments in this study, *N* is set to 5.

IV. EXPERIMENT DESIGNS AND RESULTS

Our objectives in this study are two-fold: (1) to gain some experience in a mini-cloud environment, and hopefully this may be extended to future application with more input and in a larger scale cloud environment setup, (2) to evaluate how the parallel genetic algorithm performs in constructing the ANN-based LD identification model with the MapReduce programming model as compared to implementations using multi-threaded APIs (OpenMP) and grid-based distributed computing.

The data sets used in this study are summarized in Table II, which together with the corresponding pre-processing (such as normalization and feature selection) are exactly the same as those used in [9].

TABLE II. DATA SETS AND THEIR FEATURES USED IN THIS STUDY

	sample size	number of features
data set 1	652	7
data set 2	125	7
data set 3	159	10

To fulfill the above mentioned objectives, we have design and conducted three experiments. The ANN code (fitness function computation, adopting five-fold cross validation and 500 epochs in each ANN training) is exactly the one used in [9] (in C language), and is invoked by the Map tasks (implemented with Java language) through external procedure call. Three parameters of the ANN classifier (number of hidden nodes, learning rate and momentum), together with random number seeds, which might affect the initial weights and bias of neural network, are encoded into the chromosomes. For genetic algorithm, real-value encoding is adopted with the crossover rate, mutation rate and number of generation set at 0.8, 0.1 and 50, respectively. Furthermore, accuracy in classification is used to evaluate the fitness of populations. A performance index: correct identification rate (CIR) is defined to evaluate the experiment outcomes, as listed in equation 1 below.

$$CIR = \frac{(\text{number of correct LD and non-LD identification})}{(\text{total number of cases})} \quad (1)$$

In the first experiment, we evaluate our MapReduce implementation of the GA-based ANN classifier in terms of CIR and execution time by fixing the population size (number of chromosomes) assigned to each map task to 20 in the PDGA-based ANN classifier, while varying the number of computing nodes (1, 2, 4, 8, and 12, respectively). Accordingly, the overall population size also varies between 20, 40, 80, 160, and 240, respectively. In the second experiment, we fix the overall population size to 200, while varying the number of computing nodes (1, 2, 4, 8, and 12, respectively). In other words, the overall population is evenly distributed to each map task (in case of 12 nodes scenario, each node is assigned 17 chromosomes). The results of the two experiments are shown in Tables III and IV, with all numbers as averaged over twenty consecutive runs.

In general, according to Table III, the CIR improves as the overall population size increases. From Table IV, when adding more computing nodes (and thus reducing population size assigned to each individual node), it is possible to achieve higher CIR and lesser execution time at the same time. The above two findings are reasonable and consistent with our previous studies [9, 10].

TABLE III. PERFORMANCE COMPARISON BY FIXING POPULATION AT EACH NODE TO 20 AND VARYING COMPUTATIONAL NODES (ALL TIME IN SECONDS)

data set / slave node	1		2		3	
	CIR	*execution time*	*CIR*	*execution time*	*CIR*	*execution time*
1	87.9%	4048	84.7%	2390	86.2%	3655
2	87.9%	3689	84.9%	1998	86.4%	3401
4	87.9%	3624	85.4%	2275	86.4%	3594
8	87.9%	4431	85.8%	2584	86.6%	4289
12	87.9%	5145	86.2%	3641	86.9%	4831

TABLE IV. PERFORMANCE COMPARISON BY FIXING THE TOTAL POPULATION TO 200 AND VARYING THE COMPUTATIONAL NODES (ALL TIME IN SECONDS)

data set / slave node	1		2		3	
	CIR	*execution time*	*CIR*	*execution time*	*CIR*	*execution time*
1	88.0%	20368	85.9%	8170	86.3%	16521
2	88.0%	9781	85.8%	4116	86.8%	8807
4	88.0%	6456	85.6%	3046	86.9%	6545
8	**88.1%**	5293	85.8%	2977	86.5%	4386
12	88.0%	4378	85.7%	3407	86.9%	5035

However, in [10], we notice that in the later case (experiment 2) there may be a limit on the trend. It appears the sub-population assigned to each node has to be at least 20 to avoid the possibility that evolutionary process contains too few chromosomes and potentially causes the GA optimization process to be trapped into some local maximum. But we do not see this obvious trend in Table IV. One possible reason may be the sub-population size (17) in the 12-node scenario is very close to the above mentioned threshold (20). However, it is more likely due to our non-parallelized implementation of the Reduce stage where the GA procedure proceeds. In other words, no matter how many nodes are involved, all 200 chromosomes are taking part in the evolutionary phase in one node. Furthermore, we need to note that 88.1% (in Table IV) is the best (average) CIR we have achieved so far with data set 1.

In the last experiment, we compare our MapReduce implementation of the GA-based ANN classifier and the grid and OpenMP implementations in terms of CIR and execution time by varying the population size in a fixed 7-node mini-cloud environment. By OpenMP, we mean OpenMP APIs are used to multi-thread the most time-consuming ANN model constructions and verifications in our case. A simple static scheduling that evenly assigns population to the available threads (cores) is adopted. The outcomes are shown in Table V, again with all numbers as averaged over twenty consecutive runs. Note that all three parallel computing environments are built upon PC0 as listed in Table I so that the performance comparison can be meaningful. In addition, the outcomes of the sequential version (depicted as NA) of our ANN classifier implementation are also shown and used as the baseline for comparison.

TABLE V. PERFORMANCE COMPARISON ON VARYING THE POPULATION IN THE 7-NODE SETUP (1 MASTER + 6 SLAVES, ALL TIME IN SECONDS, NA, MR, GRID AND OMP REPRESENT SEQUENTIAL, MAPREDUCE, GRID COMPUTING AND OPENMP IMPLEMENTATIONS, RESPECTIVELY)

data set / population		1		2		3	
		CIR	*execution time*	*CIR*	*execution time*	*CIR*	*execution time*
100	*NA*	87.5%	6302	84.7%	1998	86.9%	4001
	MR	**87.9%**	3935	85.3%	2225	86.6%	3581
	Grid	87.4%	1991	**85.7%**	798	**87.2%**	1122
	Omp	87.3%	1450	84.6%	435	86.7%	847
200	*NA*	87.6%	13460	84.8%	4545	87.0%	7450
	MR	**88.0%**	5600	**86.0%**	2763	86.9%	4974
	Grid	87.3%	3775	85.7%	1513	**87.7%**	2100
	Omp	87.6%	2562	85.0%	908	86.8%	1544
300	*MR*	**88.1%**	6809	86.4%	3128	87.2%	5889

Only MapReduce implementation outcomes are available in the case of 300 population size.

According to Table V, it seems distributed implementations (either MapReduce or grid computing) perform somewhat better in terms of CIR. But when it comes to execution time, the OpenMP version of the PDGA performs the best (with speedup between 4.35 and 5.25), and the grid implementation stands second (with speedup between 2.50 and 3.57), and the MapReduce implementation falls far behind (with speedup between 0.90 and 2.40). The primary cause may be attributed to the sequential operation in the Reduce stage (the GA procedure), which in our measure may take between 25% (population=300) to 50% (population=100) of the overall execution time. Accordingly, the parallelization of the Reduce stage would be our first priority in future research.

In addition, we also note that CPU usage jumps from 23% with sequential implementation to 92% with multi-thread implementation using OpenMP APIs. In cases of MapReduce and grid computing implementations, the CPU usage can be as high as 100%. Apparently, the computing power of the underlying multi-core CPUs has indeed been fully utilized. However, considering the speedup depicted above, there may be quite a lot of work to do in reducing overhead associated with the MapReduce and grid implementations (especially with the former one), which would be another focus of our future study.

V. SUMMARY AND FUTURE WORK

In this study, we modify our grid-based PDGA implementation of the ANN classifier for identifying students with learning disabilities to the MapReduce distributed programming model. Compared with the grid computing model, MapReduce has the advantage of hiding the underlying hardware details and thus allow the programmers to be able to concentrate on the programming logic in solving the problems. The preliminary results show that in 50% of cases, the MapReduce implementation may achieve the best CIR when compared to the other parallel programming models. However, in terms of execution time, the MapReduce model does not show significant breakthrough. But we do see the potential of the MapReduce model in our future applications. For example, increase the population size, which may easily be extended by simply adding more nodes to the Hadoop-based cloud environment, seems to be a good direction to optimize the ANN LD classification model. In addition, more diagnosis data for students with LDs will be collected so that we may explore the processing power of MapReduce upon massive data sets. Finally, a more sophisticated parallelized GA procedure in the Reduce stage is also under development.

ACKNOWLEDGMENT

This work was supported in part by the National Science Council of Taiwan, R.O.C. under Grant NSC 100-2511-S-018-011-MY2.

REFERENCES

[1] S. A. Kirk, “Behavioral diagnosis and remediation of learning disabilities,” Proc. of the Conference on the Exploration into the Problems of the Perceptually Handicapped Child, 1963, pp.1-7.

[2] J. M. Fletcher, W. A. Coulter, D. J. Reschly, and S. Vaughn, “Alternative approach to the definition and identification of learning disabilities: some questions and answers,” Annals of Dyslexia, vol. 54, no. 2, 2004, pp. 304-331.

[3] J. Schrag, “Discrepancy approaches for identifying learning disabilities,” http://www.specialed.us/discoveridea/topdocs/nasdse/discld.pdf, retrieved: June, 2013.

[4] Y.-R. Meng and L.-R. Chen, “On discussing the differences about the learning characteristics of LD,” Bulletin of Special Education, vol. 233， 2002， pp. 75-93. (in Chinese)

[5] C. L. Nicholson and C. L. Alcorn, “Interpretation of the WISC-III and its subtests,” Paper presented at the 25th Annual Meeting of the National Association of School Psychologists, Washington, DC, 1993.

[6] T.-S. Huang, “A Study on the characteristics of WISC-III for students with learning disabilities,” Master thesis, Graduate Institute of Special Education, National HsinChu University of Education, Hsinchu, Taiwan. (in Chinese)

[7] T.-K. Wu, S.-C. Huang, and Y.-R. Meng, “Evaluation of ANN and SVM classifiers as predictors to the diagnosis of students with learning disabilities,” Expert Systems with Applications, vol. 34, no. 3, April 2008, pp. 1846-1856.

[8] T.-K. Wu, S.-C. Huang, and Y.-R. Meng, “Effects of feature selection on the identification of students with learning disabilities using ANN,” Lecture Notes in Computer Science, Springer Berlin/Heidelberg, vol. 4221, 2006, pp. 565 – 574.

[9] T.-K. Wu, S.-C. Huang, Y.-R. Meng, Y.-L. Lin, and H. Chang, “On the parallelization and optimization of the genetic-based ANN classifier for the diagnosis of students with learning disabilities,” Proc. 2010 IEEE Conference on Systems, Man and Cybernetics, 2010, pp. 4263-4269.

[10] T.-K. Wu, S.-C. Huang, Y.-R. Meng, and T.-H. Wu, “Experiences on constructing neural network based learning disabilities identification model with the Amazon elastic compute cloud,” Proc. 2012 International Conference on Internet Study, 2012.

[11] K. Kazunori, M. Hiroshi, and I. Masaaki, “Asynchronous Parallel Distributed GA using Elite Server,” Proc. 2003 congress on evolutionary computation, 2003, vol. 4, pp. 2603-2610.

[12] B. Baesens, T. Van Gestel, S. Viaene, M. Stepanova, J. Suykens, and J. Vanthienen, “Benchmarking state-of-the-art classification algorithms for credit scoring,” Journal of the Operational Research Society, vol. 54, 2003, pp. 627–635.

[13] C. M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, Oxford, UK, 1995.

[14] E. Istook and T. Martinez, “Improved backpropagation learning in neural networks with windowed momentum, International journal of neural systems,” vol. 12, no.3 & 4, 2002, pp. 303-318.

[15] E. Cantú-Paz and C. Kamath, “An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems,” IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, vol. 35, no. 5, 2005, pp. 915-927.

[16] N. Sakamoto, K. Ozawa, and T. Niimura, “Grid computing solutions for artificial neural network-based electricity market forecasts,” Proc. 2006 International Joint Conference on Neural Networks, 2006, pp. 4382-4386.

[17] T. White, Hadoop: The Definitive Guide. O’ Reilly Media, Inc.

[18] Y. Wang and W. Chen, “Introduction to the Hadoop distributed file system,” http://trac.nchc.org.tw/cloud/raw-attachment/wiki/NCHCCloudCourse090331/3.ppt, retrieved: June, 2013.