Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 13 (1): 133-139, 2013
ISSN 1812-5654 / DOI: 10.3923/jas.2013.133.139

Divide and Conquer Approach in Reducing ANN Training Time for Small and Large Data

Mumtazimah Mohamad, Md Yazid Mohd Saman and Muhammad Suzuri Hitam
Department of Computer Science, Faculty of Science and Technology,
University Malaysia Terengganu, Kuala Terengganu, Terengganu, Malaysia

Abstract: Artificial Neural Networks (ANN) are able to simplify recognition tasks and have been steadily improving both in accuracy and efficiency. Classical ANN, as a universal approximator, has been proven to be a more versatile and flexible method compared to modern, high-end algorithms. However, there are several issues that need to be addressed when constructing an ANN used for handling large-scaled data, especially those with a low accuracy score. Parallelism is considered as a practical solution to solve such large workload problems. A comprehensive understanding is needed to generate scalable neural networks in order to achieve an optimal training time for a large network. This study proposed several strategies for distributing data to several network processor structures to reduce the time required for recognition tasks without compromising the achieved accuracy. The initial results obtained indicate that the proposed strategies are able to improve the speed up performance for large scale neural networks while maintaining the accuracy.

Key words: Artificial neural network, back propagation, parallel, large data, low accuracy score

INTRODUCTION

Artificial Neural Networks (ANN) are a popular tool that has tremendously assisted machine learning, especially in recognition tasks in various fields (Alsmadi *et al*., 2009; Yang and Yang, 2008; Negnevitsky, 2002). Recognition is the major area where ANN techniques made their first significant contribution in computer science applications. The techniques have proved to be reliable alternatives to classical methods (Alsmadi *et al*., 2009). In order to reduce the time required to process network parameters and implement learning techniques, it is suggested to use a soft network architecture, instead of a special hardware implementation (De Llano and Bosque, 2010; Turchenko and Grandinetti, 2009). ANN learning methods are extensively used for both pattern and character recognition as they are able to reduce the associated workload and improve both the obtained accuracy and performance (Enachesu and Miron, 2010).

There are various ways to recognize patterns. The most basic way is by using probabilistic methods (Liu and Fujisawa, 2008) while the more advanced machine learning techniques includes support vector machine (Liu and Fujisawa, 2008) and genetic algorithm (Rivero *et al*., 2009) methods. One of the most important types of Multilayer Perceptron (MLP) is the Back Propagation Neural Networks (BPNN). The technique is most widely used learning algorithm (Rumelhart and McClelland, 1986) and proved to be the most universal approximation technique (Hornik, 1991). However, BPNN is not considered to be an optimal classifier (Yang and Browne, 2004). The technique has been considered to be used as a delta rule generalization for nonlinear functions. The motivations for using BPNN methods are its simplicity, availability of the required hardware and reliability (Park *et al*., 2008). The size of a BPNN method is measured by the number of weights. The greater the number of patterns is, the better the training net will be. If the related epochs are too low, the BPNN method does not allow the network to learn. In contrast, if the related epochs are too many, the method can lead to overtraining or over adjustment. Many studies have successfully proven the advantages of BPNN. However, there is room for further improvement in the management of large datasets, such as those used for identification purposes, without compromising the achieved accuracy. With a large MLP network, online BPNN could lead to thousands of epochs that require up to months to process (Ciresan *et al*., 2010; Kattan *et al*., 2010). Therefore, the ability of BPNN to manage large data could be questioned since they need a lot of training and hence, computational time. Here, the multiple processors are used to distribute an ANN computation while maintaining the ANN connectionism.

In a parallel ANN task, it is important to ensure that the performance of the multiple processors that are using

Corresponding Author: M. Mohamad, Department of Computer Science, Faculty of Science and Technology, University Malaysia Terengganu, Kuala Terengganu, Terengganu, Malaysia
Tel: +60199348822 Fax: +6096155722

classical BPNN can be approximated as the performance of a single processor (Ganeshamoorty and Ranasinghe, 2008). Parallelization of the training stage distributes the workload over a set of processors (Enachesu and Miron, 2010; Yang and Yang, 2008; Kattan *et al*., 2010). The multiple processors of BPNN may help to reduce the training time compared to in sequential training (Babii, 2007; Turchenko and Grandinetti, 2009). However, most of the previous research conducted did not prove that BPNN is able to maintain the achieved accuracy regardless of the size of the network. The motivation of this study is to formulate a scalable learning approach that can accurately recognize patterns and compare its performance with single networks and other similar multi networks. Additionally, the majority of studies conducted to date focus on solutions that emphasize on image or pattern feature extraction recognition and are not based on the original image pixels. The datasets chosen for this experiment are, first MNIST which originally constructed from a special database of the National Institute of Standard and Technology and secondly, Iris dataset from UCI machine learning repository. Therefore, this paper aims to understand whether BPNN can be used to process benchmark large running the computation on homogeneous computer clusters utilizing Message Passing Interface (MPI) tool to communicate within the clusters.

BPNN ALGORITHM

The Back Propagation Neural Network (BPNN) is a gradient descent algorithm that minimizes the error signal between the actual and the target output of a multilayer perceptron (Ganapathy and Liew, 2008; Kattan *et al*., 2010; Park *et al*., 2008). BPNN solves compounding errors between interacting modules by back propagating error information throughout a network (Perwej and Chaturvedi, 2011). BPNN has the ability to learn complex inputs and map into the output by sequential training procedures. BPNN consists of forward propagation, back propagation and a weight update phase. Figure 1 shows the chain rule, represented by arrows, involving heavy output and hidden layers in which the initial weight will be established to issue an error function. The networks adapt and generalize themselves with the data by repeating the three phases until the error signal achieves its target (Ahmad and Zhang, 2005; Grubb and Bagnell, 2010). When the target is achieved, the BPNN is considered has solved the classification problem.

The feed forward phase can be represented by Eq. 1 in order to obtain the network response after the pattern

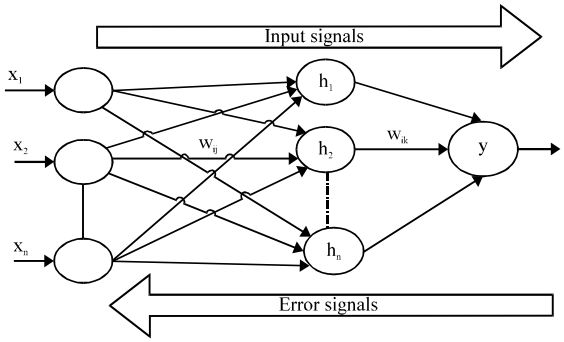

Fig. 1: The BPNN layers based on Negnevitsky (2002)

data is used as the input into the network. This feed forward phase, as in Negnevitsky (2002), are as follows:

$$y = F_3\left(\sum_{j=1}^{N} W_{j3}\left(F_2\left(\sum_{i=1}^{N} W_{ij}x_i - T_j\right)\right) - T\right) \quad (1)$$

where, N is the number of nodes of corresponding errors while w_{j3} is the weight of node j of the hidden layer to the nodes. w_{ij} and x_i are the weight for the input node to the node of the hidden layers and the output values, respectively. T_j is the threshold for the output nodes. The error signal is based on the weights of the neural network (Grubb and Bagnell, 2010) as in Eq. 2 that represent the sum square error of the difference between the output y and the desired output t (Tulunay *et al*., 2004):

$$E = \frac{1}{2}\sum_{p=1}^{P}(t - y)^2 \quad (2)$$

The error is then minimized by calculating the gradient descent with the corresponding weight for each node of layers. The weight update starts from the output layer to the hidden layer by propagating the error to each layer. The update phase is based on Tulunay *et al*. (2004) and is as follows:

$$\Delta w_{ij} = -\eta\frac{\partial E}{\partial w_{ij}} = \sum_{p=1}^{P}\left(-\eta\frac{\partial E_p}{\partial w_{ij}}\right) = \sum_{p=1}^{P}\Delta_p w_{ij} = \sum_{p=1}^{P}\Delta_p w_{ij} = \left(\sum_{p=1}^{P}\delta_{pi}O_{ij}\right) \quad (3)$$

The weights and the learning gradients are calculated in each cycle as in Eq. 3. δ_{pi} represents the error term of each layer and η represents the learning rate. The activation function of the hidden layer nodes should be a nonlinear function to a linearly separable problem, the case which BPNN could be best utilized. In this case, a hyperbolic tangent activation function is used because it is faster than a sigmoid activation function when used to

handle a large size of inputs (Negnevitsky, 2002; Tulunay *et al*., 2004). The changes of the weight will slightly induce bigger changes compared to a sigmoid activation function. The advantage of using BPNN is that the method can be carried out using parallel processing. Therefore, the development BPNN-based model provides an accurate prediction of the execution time through different network setups.

BPNN TRAINING DATA SETS

The first data set is an Iris plant data set which the best known database is created by R.A. Fisher, obtained from the UCI Machine Learning Repository. The Iris dataset classifies three different classes of Iris plants with 150 instances. The normalization of Iris involves mapping the four inputs into a range of [-1,1]. The Modified National Institute of Standards and Technology (MNIST) dataset is a standard handwritten digit classification and recognition benchmark. It is originally developed by the National Institute of Standards and Technology. The dataset contains the original black and white images of digits 0 to 9. To date, there are 60 000 training samples and 10 000 test samples, with a resolution of 28×28 pixels and a depth of 256 bit that could be used to test and verify the effectiveness of various machine learning algorithms. These images are normalized using bipolar method while preserving their aspect ratio. The dataset is stored in vectors and multidimensional matrices. The integer data are stored in the most significant bit first (high-endian) format used by most non-Intel processor.

A simple program is developed to read the MNIST dataset format by flipping the byte header in the IDX file format of the dataset. The image pixels are organized row-wise with values ranging from 0 to 255. 0 represent the background (white) while 255 represent the foreground (black). The network is trained on bipolar values instead of grey scale values. Bipolar in this case refers to the normalization on both the network input and output. The pixel intensity of the original MNIST gray scale images, ranging from 0 (background) to 255 (maximum foreground intensity), are mapped to their real values range [-1, 1] (Ciresan *et al*., 2010).

PARALLEL BPNN TRAINING

Parallel processing is a collection of interconnected computers that are independent and are able to simultaneously process a task (Buyya, 1999) by distributing the computational workload among a set of computers (Ciresan *et al*., 2010). The high level of parallelization of the deployed architecture, provided by the insertion of additional computing nodes, allows substantial time reduction. This architecture needs a communication network to connect all the processors used. The single processor computation could be made faster and more efficient by using a number of processors connected using ANN (Buyya, 1999). Many previous studies emphasize on a success of simultaneous training tasks but did not show a clear connection between the accuracy obtained with the different scale size of the particular ANNs. Most of them focused on special purpose hardware implementations that facilitate high-level parallelism such as the visual shared memory architecture (Ganeshamoorty and Ranasinghe, 2008; Kattan *et al*., 2010; De Llano and Bosque, 2010), the cluster system (Ganeshamoorty and Ranasinghe, 2008) and networks of stations (Dahl *et al*., 2008). Parallel BPNN has been shown to reduce the training time required in sequential ANN (Suresh *et al*., 2005). ANN's learning pyramid structure can be used together with the advantages of parallel processing. Parallel learning has an important role as the size of the dataset is expected to grow much faster than the capacity of the calculation (Babii, 2007). By distributing the training task to more than one processor, a large scale yet economical and reliable computational method can be achieved. In short, parallel ANN can minimize the training time in sequential ANN.

The parallel model used in this paper is based on Single Instruction, Multiple Data (SIMD) with distributed memory architecture. Several approaches used for the algorithm mostly use MPI to provide parallelism over the processors (Babii, 2007; Lotric and Dobnikar, 2005; Turchenko, 2006). The parallel strategies can be achieved using training session parallelism, network partitioning, pattern partitioning or a combination of network and pattern partitioning (Dahl *et al*., 2008). The first technique involves processors that run with different initial training parameters where the most suitable factor can be identified (Dahl *et al*., 2008). For the second technique, network partitioning, the nodes and weights of ANN are partitioned among the different processors (Dahl *et al*., 2008). The third technique involves distributing the training set while keeping a complete copy of the network locally (Turchenko and Grandinetti, 2009). This paper implements the latter technique because it directly shows the relativity of the connections.

Batch learning is used because it allows faster convergence and has been proven to be better than online learning, especially for a large dataset (Chronopoulos and Sarangapani, 2002). The learning rate used is between 0 and 1 in order to ensure that the optimal weight converges. Parallel training involves partitioning the training session that is implemented based on

sequential back propagation algorithm (Nakama, 2009). The parallelization techniques utilize the divide and conquer method. The training dataset T is partitioned into equal parts T_1, T_2, T_3, .., T_N, where N is the numbers of parallel processors. Each processor has the whole copy of the network and their own different pattern block for own training cycle as in (3).The result of individual processor represent a component gradient of a training pattern batch as summarized in Eq. 4 and 5. After each batch pattern is presented, the master will synchronize the component gradient as in Eq. 4:

$$\Delta B^{\bar{w}}(n+1) = \eta \sum_{p \in B_q} \frac{\partial E_p}{\partial_{\bar{w}}} + \alpha \Delta B^{\bar{w}}(n) \quad (4)$$

where $\bar{w}$ represents the set of all weights. The $\Delta B\bar{w}$ represents weight changes in batch B to the corresponding sum of component gradient $\bar{g}q$:

$$\bar{g}^{<q>} = \sum_{p \in B_q} \frac{\partial E_p}{\partial \bar{w}} = \eta \sum_{q=0}^{p-1} \bar{g}^{<q>} + \alpha \Delta B^{\bar{w}}(n) \quad (5)$$

The modification of weights as in Eq. 5 involved only once while performing the collection and summation of all the component gradient of each pattern in the sequential algorithm. The component gradient applied as Eq. 4 in batch B communicates using neighbor configuration of P processor as in Eq. 5. Each P processor will send its component gradient and will receive P-1 processor's component gradient and implicitly exchange the component gradient with its predecessor. This process is repeated until the sum square value is smaller than the threshold defined.

The parallelization tools used in the implementation is MPI. MPI is a standard application of parallelism achieved using message-passing paradigm. It provides portability and flexibility in both the management and implementation of parallel algorithms with optimized code for all parallel architectures.

EXPERIMENTAL SETUP

A prototype simulator has been developed to simulate the training of BPNN on both single and parallel computers. BPNN was first trained on a single computer by using a standard online learning method and then by using batch learning with a fixed step size with a sequenced and random distribution. This fixed step method is important in showing a significant difference in the result obtained from online and batch training. The next arrangement, batch training, was used in sequence and random chunks. By tuning some of the parameters, the result shows better recognition results for our prearranged/chunked batch learning method. Table 1 shows two cases with different network parameter setups.

Table 1: Neural network setup

		BPNN Configuration				
Data sets	N	Inputs	Outputs	Hidden nodes	Hidden layer No.	Weights
Iris	150	4	1	5	1	761
MNIST	50000	784	1	30	1	23581

MNIST: Modified National Institute of Science and Technology

Both datasets were normalized and randomly distributed from all classes related to its class to obtain the generalized results. The dataset were divided into two groups; the training (70%) and the testing datasets (30%). Training parameter values, such as the learning rate and the momentum are fixed throughout training in this comparison. The cluster consists of up to 6 nodes and uses Intel Dual Core Machines. The algorithm is written in C while the MPI is implemented using MPICH2 and were compiled in Visual Studio 2005. The performance of BPNN using parallel computation can be measured by both the speedup and the efficiency factors. The speedup factor is analyzed in Eq. 6. where t_s is the execution time on a single processors and t_n is the execution time on parallel computers:

$$s(n) = \frac{t_s}{t_n} \quad (6)$$

RESULTS AND DISCUSSION

Sequential and parallel training algorithms have been successfully developed and are able to run in sequential and parallel algorithms for MNIST and Iris datasets. Table 2 shows that there is no significant difference between the sequential and parallel learning recognition rates for Iris dataset while for the MNIST dataset, the recognition rates are slightly lower. This result is similar to what has been reported by Bhagat and Deodhare (2006) where the divide and conquer likely affects the recognition rates. The best result of divide and conquer strongly depends on data partitioning strategy used in solving a large number of inputs (Parvin *et al*., 2011). The execution time recorded for the Iris dataset shows no difference between sequenced and paralleled learning, but for the MNIST data, a reduction in the execution time for parallel learning is observed. This finding showed the similar execution time for small data in (Dahl *et al*., 2008). However, the large input data influenced the computational complexity with the increases of the neuron at hiding layer (Turchenko, 2006; Babii, 2007).

Table 2: Recognition rates and training time

Parameters	Processors 1	2	3	4	5	6
Recognition rates (%)						
Iris	95.5	95.1	95	95	94	94
MNIST	90.3	87	86	85	84	84
Training time (h)						
Iris	0.0132	0.014	0.013	0.0117	0.01139	0.009
MNIST	365.047	69.89	22.84	19.76	15.82	13.48

MNIST: Modified National Institute of Science and Technology

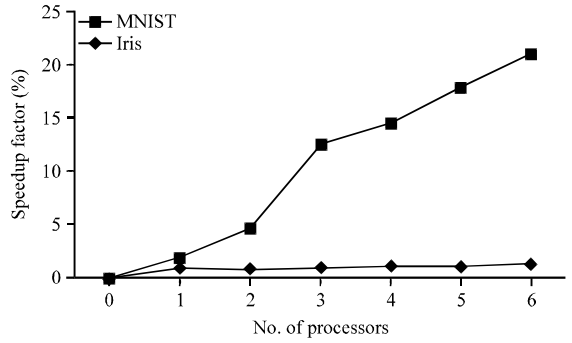

Fig. 2: Result of speedup factors BPNN layers, MNIST: Modified National Institute of Science and Technology

Figure 2 shows the speedup factor of the 6 processors used in the study. The greater number of processors shows the higher speed up factor for the MNIST dataset while for the Iris dataset, no significant relation is observed. This finding is according to the fact that the speedup of both datasets applied in parallel BPNN is non-linear (Turchenko, 2006) and it increases when the input neuron and training grew. Each processor runs as a sequence of a single processor and are independent from each other except for the exchange of the delta of weights similarly proved by (Turchenko and Grandinetti, 2009) without comparing the size of the networks. Therefore, in this study, several hypotheses are answered. Firstly, the performance of BPNN of multiple processors can be approximated as similar as the performance of the single processor. This opinion is equivalent to what had been proved by (Ganeshamoorty and Ranasinghe, 2008; Dahl *et al*., 2008; Babii, 2007).

Secondly, parallel training is better than sequential training in case of large or small case data. However, there are an overhead of MPI communications. A sufficiently large number processor may affect the performance gain but in this case, the accuracy is acceptable. The finding shows similarly efficiency as (Babii, 2007) where the efficiency looses during weight updates. The exchange of delta of weight adds to the workload of the parallel processor as demonstrated, thus requiring additional computational time. Thirdly, the best result for MNIST BPNN training can be obtained by data normalizing task before training as demonstrated by Ciresan *et al.* (2010) and Kattan *et al.* (2010). In short, this test shows that large distributed systems help to accurately measure the systems' performance in a diverse environment. The divide and conquer method used in parallel computing indirectly decreases the accuracy as the number of processors gets bigger. In the future, the ensemble training could be employed to preserve the recognition rates.

Overall, batch training should be more efficient than online training for large datasets as suggested in (Nakama, 2009). In fact, in our case, batch learning might be one of the reasons that yielded faster convergence in parallel training including the bipolar normalization, number of hidden nodes, learning rate and the momentum. Thus, in the future, it is recommended that these training algorithms deploy mini batch training that combines online and batch training. This can be achieved by dividing the training data into small batches and training is done using these batches instead of a single large batch.

CONCLUSIONS

The parallel approach is shown to be successful in speeding up the learning process for BPNN when the MNIST benchmark is used. It is understood that effort can be made to achieve a higher accuracy while preserving minimal execution time for ANN. The use of parallel computing in our case reduced the learning time with the optimization of network parameters and pre-processing phase and distributed data training. This approach is not limited to our test cases only and could be potentially used to solve visual or any other pattern recognition problems. Alternative solutions include the use of other parallel strategies of ensemble neural learning for dynamic load balancing in order to find the best method to reduce the computational time while preserving accuracy in large datasets.

ACKNOWLEDGMENTS

We thank JASKOM of Faculty of Science and Technology, University Malaysia of Terengganu, fellow postgraduates and all publication support personnel who have provided helpful comments on previous versions of this document.

REFERENCES

Ahmad, Z. and J. Zhang, 2005. Combination of multiple neural networks using data fusion techniques for enhanced nonlinear process modelling. Comput. Chem. Eng., 30: 295-308.

Alsmadi, M.K.S., K.B. Omar and S.A. Noah, 2009. Back propagation algorithm the best algorithm among the multi-layer perceptron algorithm. Int. J. Comput. Sci. Network Secur., 9: 378-383.

Babii, S., 2007. Performance evaluation for training a distributed backpropagation implementation. Proceedings of the 4th International Symposium on Applied Computational Intelligence and Informatics, May 17-18, 2007, Timisoara, pp: 273-278.

Bhagat, S. and D. Deodhare, 2006. Divide and conquer strategies for Mlp training. Proceedings of the International Joint Conference on Neural Networks, July 16-21, 2006 IEEE, Vancouver, pp: 3415-3420.

Buyya, R., 1999. High Performance Cluster Computing: Architecture and System. Vol. 1, Prentice Hall, Melbourne, Australia.

Chronopoulos, A.T. and J. Sarangapani, 2002. A distributed discrete-time neural network architecture for pattern allocation and control. Proceedings International Parallel and Distributed Processing Symposium, April 15-19, 2002, IEEE Computer Society, pp: 204-211.

Ciresan, D.C., U. Meier, L.M. Gambardella and J. Schmidhuber, 2010. Deep big simple neural nets excel on handwritten digit recognition. Neural Comput., 22: 3207-3220.

Dahl, G., A. McAvinney and T. Newhall, 2008. Parallelizing neural network training for cluster systems. Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks, February 12-14, 2008, ACTA Press Anaheim, pp: 220-225.

De Llano, R.M. and J.L. Bosque, 2010. Study of neural net training methods in parallel and distributed architectures. Future Gener. Comput. Syst., 26: 267-275.

Enachesu, C. and C.D. Miron, 2010. Handwritten digits recogntion using neural computing. Proceedings of the International Conference Interdiciplinarity in Engineering, November 12-13, 2009, Romania, pp: 95-104.

Ganapathy, V. and K.L. Liew, 2008. Handwritten character recognition using multiscale neural network training technique. Proceedings of World Academy of Science, Engineering and Technology, May 2008, Dubai, UAE, pp: 29-37.

Ganeshamoorty, K. and D.N. Ranasinghe, 2008. On the performance of parallel neural network implementations on distributed memory architectures. Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid, May 19-22, 2008, Lyon, pp: 90-97.

Grubb, A. and J.A. Bagnell, 2010. Boosted backpropagation learning for training deep modular networks. Proceedings of the 27th International Conference on Machine Learning, June 21-24, 2010, Haifa, Israel, pp: 407-414.

Hornik, K., 1991. Approximation capabilities of multilayer feedforward networks. Neural Networks, 4: 251-257.

Kattan, A.R.M., R. Abdullah and R.A. Salam, 2010. Training feed-forward neural networks using a parallel genetic algorithm with the best must survive strategy. Proceedings of the International Conference on Intelligent Systems, Modelling and Simulation, Jan. 27-29, Liverpool, pp: 96-99.

Liu, C.L. and H. Fujisawa, 2008. Classification and Learning Methods for Character Recognition: Advances and Remaining Problems. In: Machine Learning in Document Analysis and Recognition, Marinai, S. and H. Fujisawa (Eds.). Vol. 90, Springer, Berlin, Heidelberg, pp: 139-161.

Lotric, U. and A. Dobnikar, 2005. Parallel implementations of feed-forward neural network using MPI and C# on. NET platform. http://laspp.fri.uni-lj.si/uros/docs/icannga2005u.pdf

Nakama, T., 2009. Theoretical analysis of batch and on-line training for gradient descent learning in neural networks. Neurocomputing, 73: 151-159.

Negnevitsky, M., 2002. Artificial Intelligence: A Guide to Intelligent Systems. Pearson Education, India.

Park, S.S., W.G. Jung, Y.G. Shin and D.S. Jang, 2008. Optical character recognition system using Bp algorithm. Int. J. Comput. Sci. Network Secur., 8: 118-124.

Parvin, H., H. Alinejad-Rokny and S. Parvin, 2011. Divide and Conquer Classification. Aust. J. Basic Appl. Sci., 5: 2446-2452.

Perwej, Y. and A. Chaturvedi, 2011. Machine recognition of hand written characters using neural networks. Int. J. Comput. Appl., 14: 6-9.

Rivero, D., J. Dorado, E. Fernandez-Blanco and A. Pazos, 2009. A genetic algorithm for ANN design, training and simplification. Proceedings of the 10th International Work-Conference on Artificial Neural Networks, Bio-Inspired Systems: Computational and Ambient Intelligence, Vol. 1, June 10-12, 2009, Salamanca, Spain, pp: 391-398.

Rumelhart, D.E. and J.L. McClelland, 1986. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations, MIT Press, Cambridge, ISBN: 0-262-68053-X, Pages: 611.

Suresh, S., S.N. Omkar and V. Mani, 2005. Parallel implementation of back-propagation algorithm in networks of workstations. IEEE Trans. Parallel Distrib. Syst., 16: 24-34.

Tulunay, Y., E. Tulunay and E.T. Senalp, 2004. The neural network technique-1: A general exposition. Adv. Space Res., 33: 983-987.

Turchenko, V. and L. Grandinetti, 2009. Efficiency Analysis of Parallel Batch Pattern Nn Training on General Purpose Supercomputer. In: Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing and Ambient Assisted Living, Omatu, S., M.P. Rocha, J. Bravo, F.F. Riverola, E. Corchado, A. Bustillo and J.M. Corchado (Eds.). Springer Berlin Heiderberg, USA, pp: 5518.

Turchenko, V., 2006. Fine grain approach to development of parallel training algorithm of multi-layer perceptron. Scient. Theoritical J. Artif. Intell., 1: 94-102.

Yang, F. and F. Yang, 2008. Character recognition using parallel bp neural network. Proceedings of the International Conference on Audio, Language and Image Processing, July 7-9, 2008, Shanghai, China, pp: 1595-1599.

Yang, S. and A. Browne, 2004. Neural network ensembles: Combining multiple models for enhanced performance using a multistage approach. Expert Syst., 21: 279-288.