Hand Written Digit Recognition Using Backpropagation Neural Network on Master-Slave Architecture

J.V.S.SRINIVAS
DVR College of Engineering & Technology,
Hyderabad. India

N. SHIRISHA
DRK Institute of Science & Technology,
Hyderabad, India

Abstract:- **The objective of this work is to identify the hand written digits represented by rectangular box of 16x16 in a gray scale of 256 values. Backpropagation neural network (BPN) is one of the simplest models of supervised training multi layer neural networks. In this paper we design an BPN and train it with a set of hand written data. We also implement BPN on Master – Slave architecture to minimize the learning time. The performance parameters are evaluated for both sequential implementation and parallel implementation on Master – Slave architecture.**

Keywords:- Backpropagation, Master-Slave architecture, Learning rate and Semeion Handwritten Digit Data Set.

I. INTRODUCTION

In the recent years handwriting number recognition is one of the challenging research problem. Many approaches such as minimum distance, decision tree and statistics have been developed to deal with handwriting number recognition problems. Alceu de Britto *et al*.[11] proposed an approach for recognizing the handwritten numeral strings that relies on the two-stage HMM-based method. The main objective for our system was to recognize isolated Arabic digits exist in different applications. The studies were conducted on Semeion Handwritten Digit Data Set taken from UCI machine learning repository. The dataset contains 1593 handwritten digits from around 80 persons that were scanned, stretched in a rectangular box 16x16 in a gray scale of 256 values. Then each pixel of each image was scaled into a Boolean (1/0) value using a fixed threshold. Each person wrote on a paper all the digits from 0 to 9, twice.

In general, parallel processing can be achieved in two different ways – data set parallelism and algorithmic parallelism. Algorithmic parallelism offers two possible schemes for the MLP: (i) mapping each layer of the network to a processor and (ii) mapping a block of neurons to a processor. A special case of second scheme is to map each neuron of the network to a processor so that the parallel computer becomes a physical model of the network.

The proposed parallel implementation avoids recomputation of weights and requires less communication cycle per pattern. The communication of data among the processors in the computing network is also less. We obtain the performance parameters like speed-up, optimal number of processors and processing time for both sequential implementation and parallel implementation on Master – Slave architecture. The analytical and experimental results show that the proposed parallel implementation performs better than the sequential implementation.

II. MATHEMATICAL MODEL

In this paper, we consider a fully connected MLP network trained using BP algorithm with momentum. We consider MLP with three layers (l=0,1,2) having N_l neurons in each layer. The different phases in the learning algorithm and corresponding processing time are discussed in the following sections.

A. Sequential Back – Propagation Algorithm

The BP algorithm is a supervised learning algorithm, and is used to find suitable weights, such that for a given input pattern (X^0), the network output (Y^2_i) should match with the desired output (d_i). The algorithm is divided into three phases, namely, forward phase, error BP phase, and weight update phase. The details on each of these phases and the time taken to process are discussed below.

Forward phase**:** For a given input pattern X^0, the activation values of hidden and output layer are computed as follows:

$$Y_i^l = f(\sum_{j=1}^{Nl-1_2} w_{ji}^{l-1} X_i^{l-1}), \qquad i = 0,1,2,3,.....N_l; l = 1,2; X^1 = Y^1$$

where w_{ji}^{l-1} is the weight of the synapse from the i[th] neuron in the (l-1) layer to the j[th] neuron in the l[th] layer, y_j^l is the output of the j[th] neuron that belongs to the l[th] layer, and f(.) is the activation function (bipolar sigmoid function).

Let t_m, t_a, and t_{ac} be time taken for one floating point multiplication, addition, and calculation of activation value respectively. The time taken to complete the forward phase (T_1) is given by

$T_1 = N_1 (N_0 + N_2) M + (N_1 + N_2) t_{ac}$ where $M = t_a + t_m$

Error back propagation phase: In this phase, the deviation between the network output and the desired value is back – propagated to all the neurons through output weights. The δ^2_i term for i[th] neuron in output layer is given by

$\delta^1_i = (Y^2_i - d_i) f`(.)$ i= 1,2,3, …..N_2

where f ′(.) is the first derivative of the activation function.

The term (δ^1_i) for i[th] hidden neuron is given by

$$\delta^1_i = f'(.) \sum_{j=1}^{N_2} w_{ji}^2 \delta_i^2, \qquad i = 1,2,3,.....N_1$$

The time taken to complete the error back propagation phase is represented by T_2 and is calculated as

$T_2 = (1 + N_1) N_2 M + (N_1 + N_2) t_{ac}$

Weight update phase: In this phase, the network weights are updated and the updation process of any w^l_j depends on the value of δ^l_j and y^{l-1}_j.

$\Delta w^l_{ji} = \eta \delta^l_i y^{l-1}_j + \mu \Delta w^{l-1}_{ji}$ l = 1, 2

Where η the learning is rate and μ is the momentum factor. The time taken to update the weight matrix between the three layers is represented by T_3 and it is equal to

$T_3 = N_1 (N_0 + N_2) M + (N_0 + N_2) N_1 t_{ac}$

Let $t_{ac} = \beta M$. The total processing time (T_{seq}) for training a single pattern in one iteration is the sum of the time taken to process the three phases and is given as

$T_{seq} = T_1 + T_2 + T_3 = (N_1 K + N_2 \gamma) M + N_1 (N_0 + N_2) t_{ac}$

Where $k = 2N_0 + 3N_2 + 2\beta$ and $\gamma = 1 + 2\beta$

B. Parallel Implementation:

In this implementation, the hidden layer is partitioned using vertical parallelism and weight connections are partitioned on the basis of synaptic level parallelism. The Master Slave architecture of the MLP network used in the proposed scheme is shown in the fig.1. In this architecture there is master and ‘m’ slave processors. The output layer is placed on front – end processor, hidden layer is partitioned into N_1/ m neurons and the partitions are placed on slave processors. The input neurons are placed on all the slave processors.

In this architecture the communication is only between the master and m Slave processors. Each of the Slave processor with master executes three phases of BP training algorithm. Parallel execution of the three phases and the corresponding processing time for each phases are calculated.

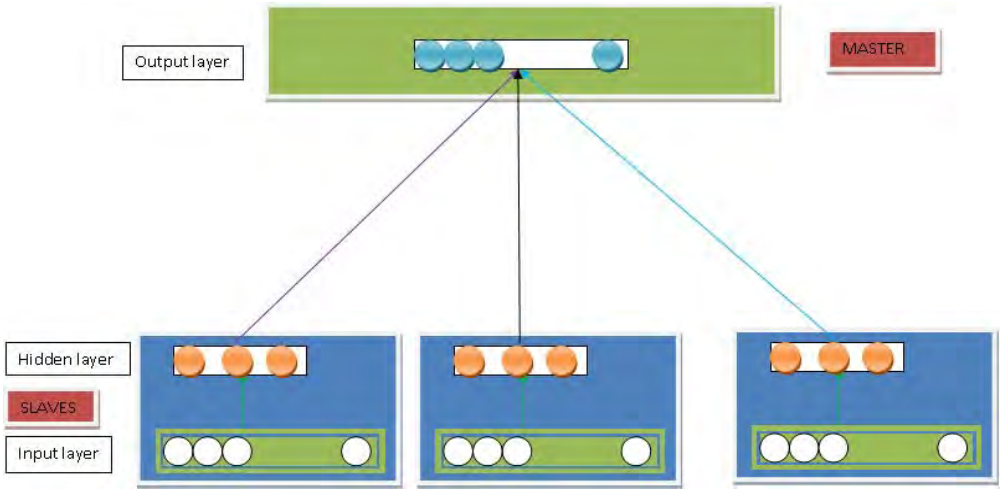

Fig: 1

Forward phase: In this phase, the activation values of the neurons are calculated. The activation value for the hidden neurons is calculated at the slave processors. But, the activation value for the output neurons is calculated at the master. The activation values and the weight connections of neurons present in slave processors are send to master and then activation values of output neurons are calculated. The approximate time taken to process the forward phase (T_1) is given by.

$$T_1 = \frac{[N_0 + N_2]N_1 M}{m} + [\frac{N_1}{m} + N_2]\, t_{ac} + N_2\, m\, t_a + t^1_{com} \quad \text{where} \quad M = t_a + t_m$$

Error propagation phase: In this phase, each processor calculates the error terms. The approximate time taken for processing error back-propagation phase (T^M1) includes the error calculation at the FEP and at the slave proc4essors and is

$$T_2 = (N_1 + m)\, N_2\, M/m + \{N_1/m + N_2\}\, t_{ac} + t_{com}$$

Weight update phase: In this phase, the weight connections between the neurons of the output layer and hidden layer are updated (we also update the weight connections between hidden and input layers). To update the weight connection between the i^{th} hidden neuron and j^{th} input neuron, we need activation values at the jth input neuron and δ_i^1 term in the ith hidden neuron. Since both the activation value as well as the error term are present in the processor, no communication is required. The processing time to update the weight connections (T^M_3) is given by

$$T_3 = (N_o + N_2)\, N_1\, M/m + ((N_o + N_2)\, N_1 / m)\, t_m$$

Let T_M be the time taken to train a single pattern in proposed HP scheme, and it is equal to the algebraic sum of time taken to process all the three phases.

$$T_P = M[2(N_0 + N_2)\, N_1 + (N_1 + m)\, N_2]/m + [\frac{N_1}{m} + N_2]\, t_{ac} + (N_o + N_2)\, N_1\, t_m / m$$
$$+ M \log m \{2\sigma + \mu[(N_1/m) + N_2]\}$$

Here $t_{com} = \log p\, [T_{ini} + T_c\, D]$ where p is the number of processors, D is data size(number of words), T_{ini} is startup time and T_c is per word transfer time.[11]

C. Analytical Performance Comparison:

Maximum number of processors: In general number of processors in parallel implementation in the worst case is equal to number of hidden neurons. Increase in the number of processors does not improve the performance after a certain stage.

Speed up analysis: Speed-up for m-processor system is the ratio between the time taken by uniprocessor to the time taken by parallel algorithm in m-processor network.

The speed-up ratio for parallel implementation can be formulated as

$$S = T_{seq} / T_P$$

$$= [(N_1 k + N_2 \beta)M + N_1 (N_0 + N_2)\, t_{ac}]\ \{M[2(N_0 + N_2)\, N_1 + (N_1 + m)\, N_2]/m + [\frac{N_1}{m} + N_2]\, t_{ac}$$
$$+ (N_o + N_2)\, N_1\, t_m / m + M \log m \{2\sigma + \mu[(N_1/m) + N_2]\}\}^{-1}$$

If the network size is extremely larger than the number of processors m, then the speed up ratio will approach m. This is due to extra computation required in weight update phase and extra communication in exchanging the hidden neurons activation values.

Optimal number of processors: From the expression of T_P , it is clear that if we increase the number of processors, the time taken to communicate will also increase and the time taken for computation will decrease. The total processing time will decrease first and then increase after a certain number of processors. So, there exists an optimal umber of processors m* for which processing time is minimum. We calculate the closed form expression for optimal number of processors by partially differentiating the training time expression T_P with respect to m and then equating it to zero.

Difference between processing times:

From the expressions of T_{seq} and T_P the difference in processing time is calculated as follows.

$$T_{seq} - T_P = [(N_1 k + N_2 \beta)M + N_1 (N_0 + N_2) t_{ac}] - \{M[2(N_0 + N_2) N_1 + (N_1 + m) N_2]/m + [\frac{N_1}{m} + N_2] t_{ac} + (N_o + N_2) N_1 t_m / m + M \log m \{2\sigma + \mu[(N_1/m) + N_2]\}\}$$

Assume that $N_1 = N_2 = N_3 = m = N$. Then

$$T_{seq} - T_P \cong$$

$$[5N^2 - 5N(\sigma + \mu) - 7 + \beta(2N^2 + 4N - 4)]t_a + [5N^2 + N - 5N(\sigma + \mu) - 10 + \beta(2N^2 + 4N - 4)] t_m > 0$$

From the above equation it is clear that the time taken by parallel implementation is less than the time taken by the sequential algorithm.

III. EXPERIMENTAL RESULTS

In this section, we present the performance of the proposed parallel algorithm on master-salve architecture and compare the results with the sequential approach. Both the algorithms are implemented in Sparc-II processor based sun workstations connected through Ethernet. There was a substantial decrease in the learning time of the networks in parallel implementation. The results are shown graphically in fig-2.

Learning Rate:0.2. Momentum Factor:0.25

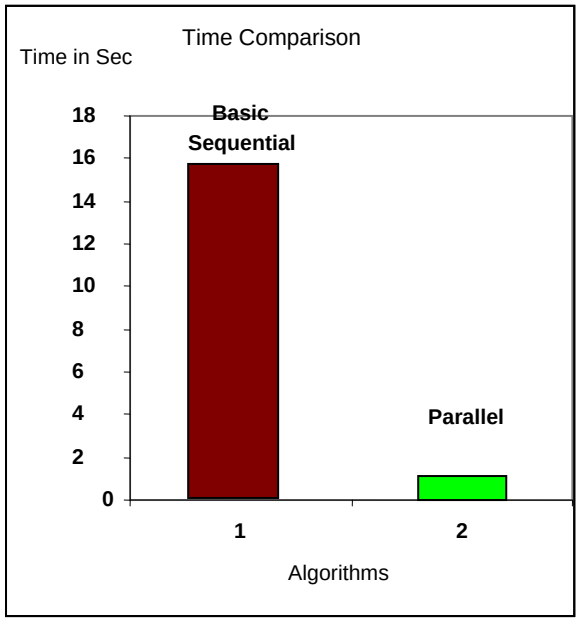

Fig-2

IV. CONCLUSION

In this paper, an implementation of distributed BP algorithm to train MLP network with single hidden layer on a cluster of computers connected over Ethernet LAN is presented. Hybrid partitioning technique is proposed to partition the network. The partitioned network is mapped onto Master-Slave architecture. The analytical performance of the proposed algorithm is compared with the vertical partitioning. Using the hybrid-partitioning scheme, recomputation of weights is avoided and the communication time is reduced. As one

hidden layer is adequate for a large number of applications, in the present project work the algorithm is developed for neural nets with one hidden layer

ACKNOWLEDGEMENTS

We acknowledge UCI machine learning repository for giving the opportunity to use the Semeion Handwritten Digit Data Set to train and test our model.

REFERENCES

[1] Christopher M. Bishop, Neural Networks for pattern recognition, 1995.

[2] Jim Torresen and Shinji Tomit, A Review of Parallel Implementations of Back propagation Neural Networks, citeseer.ist.psu.edu/495048.html.

[3] R. Greenalw, H. Hoover, and W. Ruzzo. Limits to Parallel Computation, Oxford university press, 1995.

[4] V. Kumar, S. Shekar, and M.B. Amin, "A Scalable Parallel Formulation of the Back-Propagation Algorithm for Hypercubes and Related Architectures," IEEE Trans. Parallel and Distributed Systems, vol.5, no.10,pp.1073-1090, Oct. 1994.

[5] Czauderna, T.; Seiffert, U.: Implementation of MLP networks running backpropagation on various parallel computer hardware using MPI. Proceedings of the 5th International Conference on Recent Advances in Soft Computing (RASC), Nottingham, United Kingdom, December 16-18, 2004, pp.116- 121. [ISBN 1842331108]

[6] Han-wook Lee and Chan-ik, An Efficeint Parallel Block Backpropagationn Learning Algorithm in Transputer- Based Mesh-Connected Parallel Computers, IEICE Trans. Inf. & Syst., vol. E83-D, No.8, August, 2000.

[7] Faramarz valafar and Okan K. Ersoy, Parallel Implementation of Back-Propagation Neural Network on MASPAR MP-1

[8] Laurence Fausett, Fundamentals of neural networks; Architectures, algorithms and applications, PEA, 2005

[9] S. Haykins, Neural Networks_A Comprehensive Foundation. PEA, 2004.

[10] Jacek. M.Zurada, Introduction to Artificial Neural systems, 2004.

[11] Alceu de Britto, S., R. Sabourin, F. Bortolozzi and Y. Ching Suen, 2003. The recognition of handwritten numeral strings using a two-stage HMM-based method. Int. J. Docu. Anal. Recog., 5: 102-117. DOI: 10.1007/s10032-002-0085-5

[12] Annanth Grama, Anshul Gupta, George Karypis, Vipin Kumar – Introduction to parallel computing, PEA, 2004.

[13] W. Richard Stevens.Unix Network Programming Prentice Hall.

[14] N. Sundararajan and P. Saratchandran, Parallel Architecture for Artificial Neural Networks, IEEE CS Press, 1998.

[15] V. Sudhakar, C. Siva, and R. Murthy, "Efficient Mapping of Back-Propagation Algorithm onto a Network of Workstations," IEEE Trans. Man, Machine, and Cybernetics-Part B: Cybernetics, vol.28, no.6, pp.841-848, 1998.