Enhanced Parallel Implementation of Backpropagation Algorithm for Biomedical Classification problems

T. Hamsapriya[1], S.Sumathi[2], S.N.Sivanandam[3]

Abstract

Multilayer Perceptron (MLP) with BackPropagation learning algorithm is effective in solving a wide range of bio-medical classification problems. In this paper, the MLP network trained with backpropagation algorithm is implemented on a cluster of workstations using MPI for increased computational speed and effective resource utilization. The proposed work combines data session and training set parallelism to improve the convergence rate and efficiency. This work uses the standard enhancement techniques like momentum factor, adaptive learning rate and adaptive learning rate with momentum factor combined with the standard gradient descent algorithm. The performance of parallel backpropagation algorithm is evaluated for the liver disorder diagnosis and heart disease diagnosis applications. Experimental performance shows that the proposed parallel algorithm has better speedup than the sequential algorithm.

Keywords: Backpropagation algorithm, Data parallelism, training session parallelism, Adaptive-learning rate, Momentum, Message passing interface.

[1]Department of CSE, PSG College of Technology, Coimbatore, India.

[2]Department of EEE, PSG College of Technology, Coimbatore, India.

[3]Department of Computer Science and Engineering, PSG College of Technology, Coimbatore, India.

E-mail : mp_psg@rediffmail.com[1]

1. Introduction

Artificial Neural networks has emerged as a powerful tool for solving various classification problems like heart disease diagnosis, hepatitis, lung disease, cancer detection, liver disease, blood disorders etc. However training these networks is difficult and time consuming. The performance of training algorithms can be improved by introducing parallelism. Parallel processing architectures are unavailable and are very expensive if available. But this barrier can be overcome by the availability of parallel libraries as MPI (Message passing Interface) for communication in a network of heterogeneous /homogeneous workstations.

Of the many Neural Network architectures proposed, Multilayer Perceptron with Backpropagation learning algorithm is found to be effective for biomedical applications. This algorithm is computation intensive and hence is an ideal candidate for parallel formulation. Also, it takes a great deal of time to converge to an acceptable solution. This paper discusses the parallel implementation of the Backpropagation learning algorithm with various enhancements to speedup convergence and to maintain generalized performance.

Section 2 discusses the neural parallel computing and parallelization of feed forward networks. Message passing interface is explained in Section 3. Section 4 describes the proposed work. Standard Backpropagation algorithm and various enhancements are analyzed in Section 5. Section 6 discusses the results of the experimentation.

section 7 summarizes the efficacy of the speedup techniques and presents further directions for research.

2. Neural parallel computing

Parallel computer architectures can be divided into two main categories; SIMD (Single Instruction Multiple Data stream) and MIMD (Multiple Instruction Multiple Data stream). In MIMD, the architecture is classified as tightly coupled systems and loosely coupled systems based on the degree of interaction among the processors.

Two different programming paradigms have evolved from the architectural models:

- Shared memory programming using constructs as semaphores, monitors and buffers (associated with tightly coupled systems)
- Message passing programming using explicit message-passing primitives to communicate and synchronize (associated with loosely coupled systems)

The performance of parallel algorithms can be measured using speedup. Speedup is defined as the ratio between the elapsed time using *m* processors for the parallel algorithm and the elapsed time completing the same task using the sequential algorithm with one processor.

2.1 Parallelization of feed-forward neural networks

The basic terminology in parallelization of BP neural networks is

Training set: It consists of a number of training patterns, each given by an input vector and the corresponding output vector.

Network size: A network comprises N_i input units, N_h hidden units and N_o output units .In short a network is represented as $N_i \times N_h \times N_o$.

Training Iteration: It denotes one representation of the whole training set.

2.1.1 Weight updating strategies

Learning by pattern (*lbp*) updates the weights after each training pattern has been presented.

- Learning by block (*lbb*) updates the weights after a subset of the training pattern has been presented.
- Learning by epoch (*lbe*) updates the weights after all patterns have been presented (i.e., one training iteration).

2.1.2 Weight update interval

The number of training patterns that is presented between weight updates is termed μ. For lbp, μ=1, while for lbe μ=P, where P is the number of training patterns in the training set.

The BP algorithm reveals four different kinds of parallelism as described below.

- *Training session parallelism* starts training session with different initial training parameters on different processing elements.
- *Training set parallelism* splits the training set across the processing elements. Each element has a local copy of the complete weight matrix and accumulates weight change values for the given training patterns. The weights are updated using *lbb/lbe*.
- *Pipelining* pipelines the training patterns between the layers, i.e., hidden and output layer on different processors. While the output layer processor calculates output and error values for the present training pattern, the hidden layer processor processes the next training pattern. The forward and backward phase may also be parallelized in a pipeline. Pipelining requires a delayed weight update or *lbb/ lbe*.
- *Node parallelism*, the neurons within a layer is computed in parallel (named neuron parallelism). In this method, the weights can be updated using *lbp*.

In this paper, training session parallelism is employed to determine the optimal values of initial weights and learning rate. A copy of the optimal values are loaded on to the processors and the network is trained using training set parallelism, in which training vectors are partitioned across the workstations and the overall training time is reduced.

2.2 Training Session Parallelism

In training session parallelism, multiple epochs are run, each on its own processing element (PE) simultaneously. The networks are started with different random initial weight values, learning rate, momentum and number of neurons in the hidden layer. Different learning rates allow a training session on any PE to escape a local minimum that the network would otherwise settle in.

Initial Set X **Initial Set Y** **Initial Set Z**

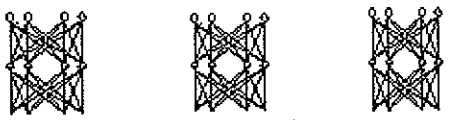

Figure 1 Data session Parallelism

Figure 1 illustrates that all the processing elements have a copy of the same network initialized with different parameters for the same set of training vectors.

2.3 Training Set Parallelism

Each PE has a local copy of the complete weight matrices and accumulates weight change values for the given subset of training patterns. Since the neural network weights must be consistent across all the PEs, the weights are globally updated (learning by block/epoch).

Subset 1 **Subset2** **Subset3**

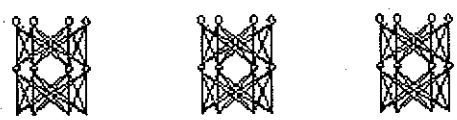

Figure 2 Training set Parallelism

In this paper, learning by epoch strategy is used for weight updation to minimize the communication delay. The weight change values of each PE are summed and used to update the local weight matrices. Figure 2 illustrates the training set parallelism where the training set is partitioned and the subsets are distributed across the processing elements.

3 Message passing Interface (MPI)

Fast learning mechanisms of neural network models have been implemented over multiprocessors on board, or in tightly coupled multiprocessor systems. These systems could be highly expensive or need good programming skill for specific purposes. Various training and learning algorithms for SIMD and MIMD implementation of backpropagation and other neural networks are discussed in [4] and [5]. As the LAN-based fast clusters of PC's or workstations over communication networks have become commonplace, it could be worthwhile to share the computing power among hardware if necessary. In this proposed scheme, the parallel backpropagation algorithm is implemented on a cluster of workstations using MPI. MPI is the de facto standard for explicit message passing on Beowulf-class supercomputers. Each workstation can communicate with other workstations through a MPI library function.

4 Proposed work

The parallel backpropagation neural network is implemented using MPI. It combines both data session parallelism and training set parallelism. In data session parallelism, different initial parameters like the initial weights and the learning rate are assigned to each workstation and the network is trained for the same sample training patterns simultaneously. The root

processor compares the training time of each workstation and finds an efficient parameter set and distributes it to all workstations. A flowchart for data session and training set parallelism is shown in Figure 3.

Training set parallelism involves dividing and distributing the subsets of patterns to processors, after training and gathering the set of weight matrices from the processors. The number of epochs for learning in each processor can vary. The training is terminated when the total error falls below a given error. It is important to equally distribute a subset of patterns to each processor. If a processor learns with the subset of unbalanced and distributed patterns, the training could be useless and it may not give an optimal solution.

After the learning phase in each processor, the root processor gathers the set of weight matrices from all processors and averages them. After initializing the network with the new set of weight matrices, each processor starts to relearn the set of weights. In the latter case, the relearning time can be reduced if the set of weight from participant processors is adaptive to the subset of patterns.

5 Standard Backpropagation Algorithm and its Enhancements

The backpropagation algorithm consists of two phases: A forward phase in which an activity pattern is applied to the input nodes of the network and its effect, propagates through the network layer by layer. Training a backpropagation network involves the following stages.

- Feedforward of the input training pattern.
- Backpropagation of the associated error.
- Weight adjustment

During feedforward, each input unit X_i receives an input signal and broadcasts this signal to each of the hidden units $Z_1,\ldots Z_p$. Each hidden unit then computes its activation and sends its signals z_j to each output unit. Each output unit Y_k computes its activation y_k to form the response of the net for the given input pattern.

The net input to Z_j is denoted by Z_in_j:

$$Z\_in_j = \sum_i x_i v_{ij},$$

The output signal (activation) of Z_j is denoted by z_j:

$$z_j = f(Z\_in_j),$$

The net input to Y_k is denoted by y_in_k:

$$y\_in_k = \sum_j z_j w_{jk},$$

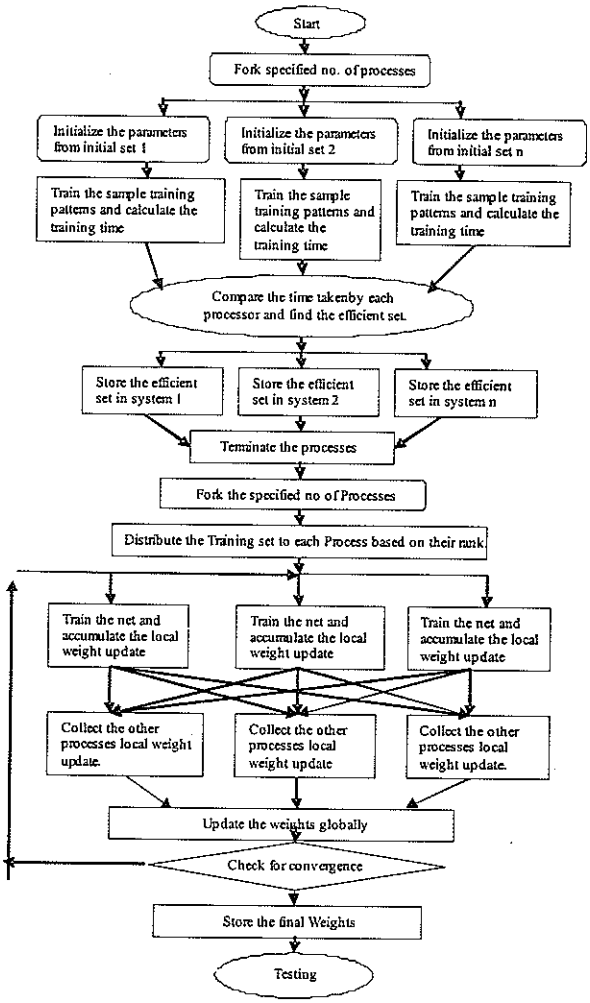

Figure 3 Flowchart for parallelism

The output signal (activation) of Y_k is denoted by y_k:

$$y_k = f(y\_in_k),$$

During training, each output unit compares its computed activation y_k with its target values t_k to determine the associated error for that pattern with that unit. Based on this error, the factor δ_k (k = 1 ...m) is computed. δ_k is used to distribute the error at output unit Y_k back to all units in the previous layer (the hidden units connected to Y_k). It is also used (later) to update the weights between the output and the hidden layer. In a similar manner, the factor δ_j (j = 1 ... p) is computed for each hidden unit Z_j. It is not necessary to propagate the error back to the input layer, but δ_j is used to update the weights between the hidden layer and the input layer.

Error information from each output unit:

$$\delta_k = (t_k - y_k)\, f'(y\_in_k),$$

Its weight update is calculated as follows:

$$\Delta w_{jk} = \alpha\, \delta_k z_j$$

Delta inputs from each hidden unit:

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk},$$

multiplies by the derivative of its activation function to calculate its error information term.

$$\delta_j = \delta\_in_j f'(z\_in_j),$$

Its weight update is calculated as follows:

$$\Delta v_{ij} = \alpha\, \delta_j x_i,$$

After all the δ factors have been determined, the weights for all layers are adjusted simultaneously. The adjustment to the weight w_{jk} (from hidden unit Z_j to output unit Y_k) is based on the factor δ_k and the activation z_j of the hidden unit Z_j of the hidden unit Z_j. The adjustment to the weight v_{ij} (from input unit X_i to hidden Z_j) is based on the factor δ_j and the activation xi of the input unit.

Each output unit (Y_k, k = 1...m) updates its weights (j =0...p):

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk},$$

Each hidden unit (Z_j, j = 1...p) updates its weights (i = 0.., n):

$$v_{ij}(new) = v_{ij}(old) + \Delta v_{ij},$$

5.1 Enhancements

The performance of the backpropagation algorithm can be improved in some situations. The enhancement techniques discussed below changes the weight update procedure to improve biological plausibility and computational power.

5.1.1 Backpropagation with Momentum

In backpropagation with momentum, the weight change is in a direction based on the current gradient and the previous gradient. This scheme is a modification of gradient descent whose advantages arise chiefly when some training data are very different from the majority of the data (and possibly even incorrect). Convergence is sometimes faster if a momentum term is added to the weight update formula. In this model, new weights for training step t+1 are based on the weights at the training step t and t-1.

The weight updation for backpropagation network with momentum factor is given as

$$\Delta w_{jk}(t+1) = \alpha\, \delta_k z_j + \mu\, \Delta w_{jk}(t),$$

and

$$\Delta v_{ij}(t+1) = \alpha\, \delta_j x_i + \mu\, \Delta v_{ij}(t),$$

where the momentum parameter μ is constrained to be in the range from 0 to 1, exclusive of the end points.

Momentum factor allows the net to make reasonably large weight adjustments as long as the corrections are in the same general direction for several patterns, while using a smaller learning rate to prevent a large response to the error from any of the training pattern. It also reduces the likelihood that the net will find weights that are local,

but not global, minimum. When using momentum, the net is proceeding not in the direction of the gradient, but in the direction of a combination of the current gradient and the previous direction of weight correction.

5.1.2Backpropagation with adaptive learning rate

This model improves the speed of training for backpropagation by changing the learning rate during training. Each weight has its own learning rate. Two heuristics are used to determine the appropriate changes in the learning rate for each weight. If the weight change is in the same direction (increase or decrease) for several time steps, the learning rate for that weight should be increased. The weight change will be in the same direction if the partial derivative of the error with respect to that weight has the same sign for several time steps. However if the direction of the weight change (i.e., sign of the partial derivative) alternates, the learning rate should be decreased. The delta-bar-delta rule consists of weight update rule and a learning rate update rule.

For each output unit "delta" is defined as,

$$\Delta_{jk} = - \delta_k z_j;$$

and for each hidden unit:

$$\Delta_{ij} = - \delta_j x_i;$$

The delta-bar-delta rule uses a combination of information about the current and past derivative to form a "delta-bar" for each output unit:

$$\Delta'_{jk}(t) = (1- \beta) \Delta_{jk}(t) + \beta \Delta'_{jk}(t-1);$$

and for each hidden unit:

$$\Delta'_{ij}(t) = (1- \beta) \Delta_{ij}(t) + \beta \Delta'_{ij}(t-1);$$

The user specifies the value of the parameter β ($0 < \beta < 1$).

The heuristic that the learning rate should be increased, if the weight changes are in the same direction on successive steps, is implemented by increasing the learning rate (by a constant amount) if $\Delta'_{ij}(t-1)$ and $\Delta_{ij}(t)$ are in the same sign. The learning rate is decreased if $\Delta'_{ij}(t-1)$ and $\Delta_{ij}(t)$ are of the opposite sign.

The new learning rate is given by

$$\delta_{ij}(t+1) = \begin{cases} \alpha_{jk}(t) + \varepsilon & \text{if } \Delta'_{ij}(t-1)\,\Delta_{ij}(t) > 0 \\ (1-\gamma)\,\alpha_{jk}(t) & \text{if } \Delta_{ij}'(t-1)\,\Delta_{ij}(t) < 0 \\ \alpha_{jk}(t) & \text{otherwise} \end{cases}$$

5.1.3 Backpropagation with Momentum and adaptive learning rate

The learning rate parameter is used to determine how fast the BP method converges to the minimum solution. A large learning rate leads to bigger step and faster convergence. But if the learning rate is too large, the algorithm will become unstable. On the other hand, if the learning rate is too small, the algorithm will take long time to converge. To speed up the convergence time and to avoid struck to the local minimum, the weight vector is adjusted with a varying /adaptive learning rate in the gradient descent with momentum.

6. Results and Discussion

Two benchmark applications were chosen to evaluate the performance of various enhancement strategies. The models were tested for liver disorder diagnosis (LDD) and heart disease diagnosis (HDD) applications.

In the training session parallelism, the master node initially assigns different parameter sets to different workstations. All the workstations have the same copy of the network architecture. The same set of training patterns are given to all the processors. The master collects the training time from all the workstations. The parameter set of the workstation with the minimum training time is chosen as the optimal parameter set. The parameters of all the workstations are cleared.

The master passes the optimal parameter set and the network architecture to all the workstations. In training set parallelism, the master partitions the training set equally among the workstations. The weights are updated locally in each workstation for the given subset of training set. The master gathers the set of weight matrices from all processors and computes the average. The weights are globally updated on all the workstations and are sent to the master.

The backpropagation learning was experimented with the standard gradient descent learning enhanced with momentum, adaptive learning rate and adaptive learning with momentum. The performance of the parallel implementation was evaluated using speedup for various enhancements. Speedup describes how the performance of a system changes with the enhancements/ improvements made to the system.

6.1 Application -I Liver Disease Diagnosis

Three processors were used to test this application. Each processor has a copy of the same network architecture. The networks are trained with different initial training parameters on different processors. Table 1 shows the training time taken by three parameter sets in the three processors. The parameter set3 consumes minimum amount of training time and is considered as the optimal parameter set for training set parallelism as illustrated in Figure 4.

Table 1 Training time for each initial parameter set in LDD.

Parameter set	Time (msec)
Set 1	1250
Set 2	457
Set 3	220

The optimal parameter set has 18 hidden neurons, learning rate as 0.3, error rate as 0.1and the momentum as 0.7.The training repetition was limited to 35000 epochs.

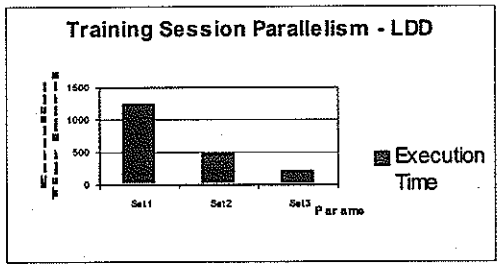

Figure 4 Performance of different initial parameters for LDD.

In the next stage, each processor has a local copy of the backpropagation network architecture and the same parameter set. The training set is partitioned across the network. Each processor maintains a local copy of the complete weight matrices and accumulates weight change values for the given training patterns. The neural network weights must be consistent across all the processors and thus weights are updated in a global operation (learning by block/epoch). The weight change values of each processor is summed and used to update the local weights. The experiment is repeated with enhancements like momentum, adaptive learning rate and momentum with adaptive learning rate and the training time is determined.

Figure 5 and Table 2 show the performance of the standard backpropagation network with and without enhancements. Backpropagation network with adaptive learning rate and adaptive learning rate combined with momentum yields a better performance compared with their counterparts.

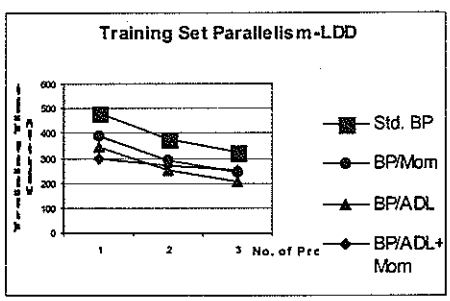

Figure 5 Performance analysis for LDD

Figure 6 illustrates that the speedup improves with the parallel implementation of the Backpropagation algorithm for various enhancements. The effect is prominent for the standard Backpropagation with adaptive learning rate and momentum. Only three processors were used for parallel implementation because the data was too small for the convergence to occur, compared to the complexity of the problem.

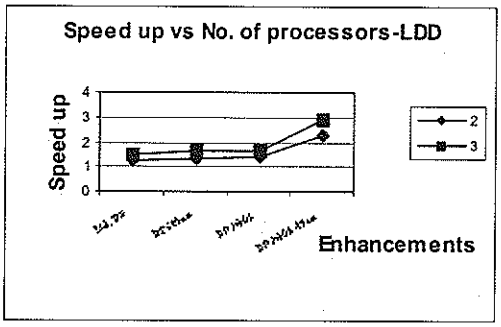

Figure 6 Speedup Analysis for LDD

Table 2 Performance of the models- LDD

Enhancements	No. of Processors			Speedup	
	1	2	3	2	3
Std. BP	483	377	323	1.28	1.49
BP/mom	392	292.5	243.5	1.34	1.61
BP/ADL	342	247.8	203.57	1.38	1.68
BP/ADL+ mom	299	270	250	2.27	2.91

6.2 Application-II Heart Disease diagnosis

The dataset for the heart disease diagnosis is collected from the Cleveland database, UCI MLRepository.html [7]. It has 6 input attributes and two output classes. The network is trained with 303 training vectors. The experiment is conducted with one hidden layer consisting of 20 hidden neurons for both the applications.

Initially, the parameter set is chosen for the backpropagation network using training session parallelism. Table 3 and Figure 7 show that parameter set2 is the ideal candidate for this application as it consumes the minimum amount of training time. Each workstation is supplied with a local copy of the parameter set and the architecture. In this application the input layer consists of 13 nodes and the output layer consists of 2 nodes. The experiment is repeated with the following enhancements to the backpropagation network.

- Standard Backpropagation (Std. BP)
- BP with momentum (BP/Mom),
- BP with adaptive learning rate (BP/ADL),
- BP with both momentum and adaptive learning rate (BP/ADL+ Mom).

Table 3 Training time for each initial parameter set in HDD.

Parameter set	Time (msec)
Set 1	27
Set 2	18
Set 3	460

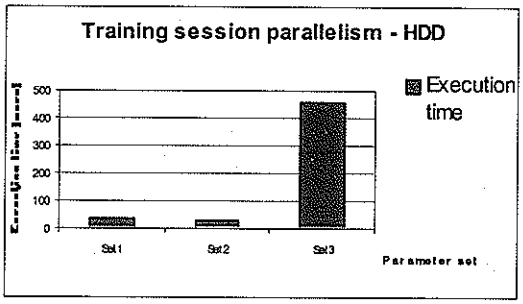

Figure 7 Time comparison of different initial parameters for HDD.

Comparative study shown in Table 4 and figure 8 illustrates that backpropagation with adaptive learning rate performs better in a multiprocessor environment for this application. In a uniprocessor system backpropagation with adaptive learning rate and momentum performs well when compared with other enhancements.

Table 4 Time taken by each model for HDD.

Enhancements	No. of Processors			Speedup	
	1	2	3	2	3
Std. BP	335	261	221	1.28	1.52
BP/mom	87.3	69.9	58	1.24	1.50
BP/ADL	79	63.7	56.2	1.24	1.41
BP/ADL+ mom	72.4	38.7	33.1	1.87	2.18

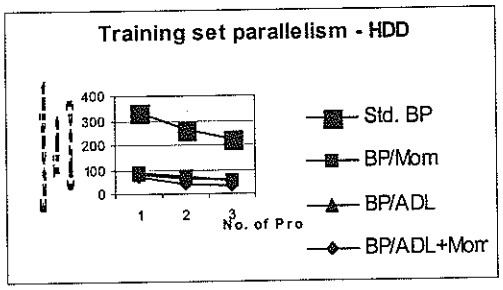

Figure 8 Performance analysis for HDD

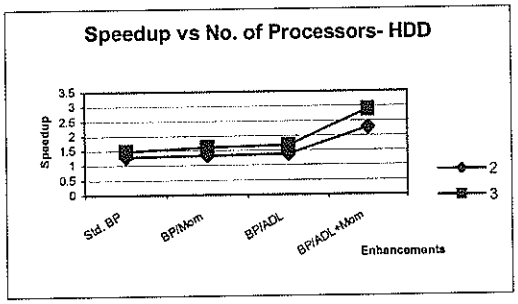

Figure 9 Speedup Analysis for HDD

Figure 9 shows a remarkable speedup in the performance of the network by increasing the number for processors with various enhancements. The maximum number of processors was limited to three as the number of training vectors was less. The training time for the heart disease diagnosis is increasing with the number of processors. This is because the communication delay is higher than converging time. This may be avoided for applications with very large training set. BP with momentum and BP with adaptive learning avoid local minima and took less time compared to standard BP model.

7. Conclusion

Despite its inherently parallel nature, an efficient parallel implementation of the backpropagation algorithm on a network of workstations performs better than its sequential counterpart. The training time of a BP algorithm also depends on initial parameters and initial weight values. Training session parallelism selects the optimal initial parameter values and decreases the training time. Training set parallelism further reduces the training time by dividing the training set across the workstations. The experimental results illustrates the marked reduction in the training time of the standard backpropagation network with momentum, adaptive learning rate, momentum with adaptive learning rate results show that adaptive learning rate with momentum, take very less training time and provide better performance, for the biomedical applications compared to other models. The performance of the backpropagation evaluated using speedup illustrates the efficacy of the parallel implementation and enhancements. The work can be extended further by studying the effect of increasing the hidden layers and varying the number of neurons in the hidden layer.

References

[1] Jim Torresen and Shinji Tomita, *"A review of Parallel Implementations of Backpropagation Neural Networks"*, IEEE CS Press, 1998

[2] Cheng-Chang Jeng , I-Ching Yang, *"Practical Implementation of Back-Propagation Networks in a low-Cost PC Cluster"*, Neural Information Processing-Letters and Reviews, Vol.4, No.3, pp33-37 ,September 2004.

[3] N. Sundararajan and P. Saratchandran, *"A Review of Parallel Implementations of Backpropagation*

Neural Networks Parallel Architectures for Artificial Neural Networks", IEEE CS Press, 1998.

[4] Timothy J. Rademacher and James E. Lumpp, *"High-Performance Simulation of Neural Networks"*, Proceedings of the 1997 IEEE-Aerospace Conference, Feb. 1997.

[5] A. Petrowski, G. Dreyfus, and C. Girault, *"Performance analysis of a pipelined backpropagation parallel algorithm"*, IEEE Transactions on Neural Networks Nov. 1993.

[6] S. L. Hung and H. Adeli, *"Parallel backpropagation learning algorithms on Cray Y-MP8/864 supercomputer"*, Neuro-computing, 5:287—302, Nov 1993.

[7] Ernest Istook, Tony Martinez, *"Improved Backpropagation learning in neural networks with windowed momentum"*, In International journal of Neural systems, vol.no.12, No.3&4,pp303-318.

[8] Suresh, Omkar and Mani, *"Parallel Implementation of Backpropagation Algorithm in network of workstations"* IEEE Transactions on Parallel and Distributed Systems, Vol. 16, No.1, , pp 24-34, Jan 2005.

[9] Udo Seiffert, *"Artificial Neural Networks on Massively Parallel Computer Hardware"* ESANN'2002 Proceedings -24-26 April 2002,ISBN 2-930307-02-1, pp 319-330.

[10] Michael J.Quinn *"Parallel Programming in C with MPI and OpenMP"*, Tata McGraw-Hill Edition, 2003.

[11] Adang Suwandi Ahmad, Arief Zulianto,Eto Sanjaya, *"Design and Implementation of Parallel Batch mode Neural Network on Parallel Virtual Machine"*, Proceedings ,Industrial Electronic Seminar, Graha Institut Teknologi Sepuluh Nopember, Surabaya, October 27-28,1999.

[12] Manavendra Misra, *"Parallel Environments for Implementing Neural Networks"*, Neural Computing Surveys Vol 1, 48-60, 1997.

Author's Biography

Ms T Hamsapriya received the ME degree in Communication Systems from PSG College of Technology. She is currently working as an Assistant Professor, Department of CSE, PSGCT. Her research interest includes Parallel and Distributed Computing, Evolutionary Computing, Neural Networks and Data Mining.She has published 10 technical papers in papers in International, National Journals and Conferences.

Dr.S.Sumathi received the Ph.D degree in Computer Science Engineering from PSG College of Technology, Bharathiar University. She is currently working as an Assistant Professor, Department of EEE, PSGCT. Her research interest includes Genetic Algorithms, Evolutionary Computing, Neural Networks and Data Mining. She has published 50 Technical papers in International, National Journals and Conferences. He has published 3 books.

Dr S N Sivanandam, received the PhD degree in Electrical and Electronics Engineering from Madras University, Chennai in 1982. He is currently serving as the Professor and Head of the Computer Science and Engineering Department, PSGCT, Coimbatore. His research interest lie in the area of Control Systems, Neural Networks, Genetic Algorithm, Digital Logic Design. He has published 400 Technical papers in International, National Journals and Conferences. He has published 7 books.