Self-Adaptive Parallel Processing Neural Networks with irregular Nodal Processing Powers using Hierarchical Partitioning

Rahul Kala*
rahulkalaiiitm@yahoo.co.in

Anupam Shukla
dranupamshukla@gmail.com

Ritu Tiwari
rt_twr@yahoo.co.in

Department of Information Technology,
Indian Institute of Information Technology and Management Gwalior, Gwalior, MP, India

*Corresponding author

Citation: R. Kala, A. Shukla, R. Tiwari (2009) Self-Adaptive Parallel Processing Neural Networks with irregular Nodal Processing Powers using Hierarchical Partitioning, *Neural Network World* 19(6), 657-680.

Final Version Available At: http://www.nnw.cz/obsahy09.html

Abstract
The architecture and working of the Artificial Neural Networks are an inspiration from the human brain. The brain due to its highly parallel nature and immense computational powers still remains the motivation for researchers. A single system-single processor approach is highly unlikely way to model a neural network for large computational needs. Many approaches have been proposed that adopt a parallel implementation of ANNs. These methods do not consider the difference in processing powers of the constituting units and hence workload distribution among the nodes is not optimal. Human Brain not always has equal processing power among the neurons. A person having disability in some part of brain may be able to perform every task with reduced capabilities. Disabilities weaken the processing of some parts. This inspires us to make a self-adaptive system of ANN that would optimally distribute computation among the nodes. The self-adaptive nature of the algorithm makes it possible for the algorithm to taper dynamic changes in node performance. We used data, node and layer partitioning in a hierarchical manner in order to evolve the most optimal architecture comprising of the best features of these partitioning techniques. The adaptive hierarchical architecture enables performance optimization in whatever condition and problem the algorithm is used. The system was implemented and tested on 20 systems working in parallel. Besides, the computational speedup, the algorithm was able to monitor changes in performance and adapt accordingly.

Keywords
Artificial Neural Networks, Artificial Brain, Parallel Processing, Partitioning, Self-Adaptive Systems, Hierarchal Processing, Node Partitioning, Data Partitioning, Layer Partitioning

1 Introduction

Artificial Neural Networks have been applied in numerous domains and problems such as speech recognition, face recognition, character recognition, financial analysis, bioinformatics, credit analysis etc. [16, 2, 34, 47]. Their application over the years has resulted in an increase in the volume and dimensionality of data that require a lot of time to train [30]. The neural networks are inspired from the human brain. The human brain [31] is made up of a number of processing units or neurons working in parallel.

With the rise of multi-processing and grid-computing, more and more systems are now being made based on parallel algorithms. Various models of parallel implementation of Back Propagation Algorithm have thus been proposed. The parallelism introduced in neural networks improves its performance especially during the training stage [8, 15, 17, 18, 21]. Parallel neural networks implement the neural networks in parallel by various kinds of partitioning of data set, nodes or layers called as data set partitioning, node partitioning and layer partitioning respectively [3, 5, 6, 42, 44, 48].

In this paper we introduce the concept of hierarchical partitioning. Every partitioning technique gives an optimal performance under some conditions. Here we apply all partitioning techniques one after the other in a hierarchical manner. First data set partitioning is applied to distribute test cases among node sets. Then layer partitioning is applied in half of the node sets and node partitioning in the other half. The actual distribution of computation is done

in an adaptive manner at runtime. This makes it possible for the most optimal architecture of the system to evolve. This further ensures an optimal distribution of computation among nodes such that no node sits idle for long [25, 34, 36, 47]. This is motivated from the adaptive learning and behavior of brain that can adapt to the internal or external changes [20, 45]. We also introduce the concept of irregular processing powers of the constituent nodes.

This paper is organized as follows. In section 2 we would discuss motivation and the present works. In section 3 we discuss the 3 basic types of partitioning i.e. data set partitioning, layer partitioning and node partitioning available in literature. In section 4 we present the hierarchical nature of the algorithm by introducing hierarchical partitioning. Section 5 talks about the self-adaptive approach of the algorithm. The general algorithm is discussed in section 6. In section 7 we give the results. Conclusion remarks are given in section 8.

2 Motivation

Use of ANNs with BPA is very common in numerous fields [16, 2, 34, 47]. Parallel algorithms due to their computational speedup are another active area of research [14]. With the massive growth in data, the need of parallel implementation of ANN is natural [6, 8, 15, 17, 21]. Node, data, layer, hybrid are commonly used techniques [3, 5, 42, 44]. These partitioning give good speedups and performance benefits as well. Newer areas include the application of Genetic Algorithms [48], hardware support exploitation [39]. BPA has been implemented in various situations [1, 14, 43]. Besides BPA, parallel implementation of Self Organizing Maps (SOMs) [13, 33, 46], Perceptrons, etc. [12] is being done. The need of autonomous and self-adaptive system is on a constant increase due to the various benefits of these systems [25, 34, 36, 47].

Hierarchal approach based partitioning has not yet been applied to the ANNs. Network of Networks (NONs) have been employed to break down the network into modular networks [4, 11, 40]. Also we introduce here the novel concept of self adaptive nature. The motivation for the same comes from the capabilities of the human brain. Here we briefly discuss the human brain as a motivator of the proposed system.

2.1 Motivations from the Human Brain

In this section we study the motivations that we derive from the human brain for implementation in the proposed algorithm. The first and foremost motivation is the architecture itself. Human brain consists of a large number of biological neurons connected by nerve fiber [31]. Information is processed and passed between these neurons in form of electrical signals. For any biological neuron, the dendrites act as receptors of signal. The cell body does the task of processing. The axon carries the outgoing electrical signals [19]. The different neurons work in parallel that gives brain the immense computational power. While the single processor systems correctly implement the brain model at the computational level, these do not exploit the parallelism of the brain functioning. The same has been implemented in the proposed algorithm with sockets playing the role of dendrites and axons; data playing the role of electrical signals; physical wires the role of nerve fiber and processing elements the role of cell body.

The second motivation that comes from the human brain is learning. The memory of the brain is in the form of the connections between neurons and their weights that gives human the intelligence. The learning in brain results in breaking and forming of a large number of connections every second [7]. The BPA is itself an inspiration for implementation of supervised learning of brain.

The third motivation derived from the human brain is evolution. The evolution of life is a perfect example which shows how the various species kept developing and adapting themselves to the surroundings. The classical example of artificial adaptation includes Genetic Algorithms where higher generations adapt themselves better to the situations or problem. As a result the higher generations are more optimized to solve the problem [24]. The human brain also starts as a premature entity of the body in a small child and keeps evolving with time to attain higher complexities and intelligence as the child develops into an adult. The algorithm that we develop would optimize the workload distribution between the various processing elements. It starts with an equitable workload distribution and then keeps modifying the distribution to optimize learning time.

The fourth motivation from the human brain is adaptation to changing situations. The human brain keeps learning continuously in an adaptive manner [20, 45]. Every day we are exposed to some new people, we see many new developments in the society and have good and bad experiences. We are easily able to remember each one of them for varying durations without long learning cycles or efforts. It would be very easy to memorize a number we just saw but very difficult to keep it in memory for 10 years. All this is because of the capability of the brain to adapt

itself in a continuous manner as the situation demands. This is also referred by the term adaptive learning in artificial systems [23, 37, 35]. The other related concepts include plasticity [22, 32] and instantaneous or short term memory [26, 27, 28, 29]. Although we do not implement adaptive learning here, we introduce the same concepts at the performance level. Here we empower the algorithm with adaptive capability to adjust their workload between the various processing elements to have the most optimal workload distribution at any time as per changing environment.

It is well known that different parts of the brain are adapted for different functions [10]. This gives us enough reasons to advocate the computational disparity of the various biological neurons of the brain. This gives us our fifth motivation. The algorithm that we develop accounts for the computational disparity of the various processing elements.

Another related concept that forms our last motivation from the human brain is flexibility. The various biological neurons in the human brain are subjected to change in performance. This may be a sudden change or a gradual change. While the sudden change may happen due to physical illness in brain, the gradual ones may be seen with the reduction in capabilities with age. In any case, the brain continues to perform. This is the flexibility factor of the brain where different parts are able to adjust themselves to continue performing even when the other part fails [9]. The same has been implemented in this algorithm. The irregular nodal processing of the brain is yet to be experimented and studied. It may be guessed that that brain has some adaptive mechanism so that the work of low efficiency or failed nodes can be given to the high efficiency or working nodes. We implement this mechanism of load balancing in this algorithm.

In this work we have proposed a server-client relationship where the server controls the nodes to implement the adaptive features of the algorithm. We know that the brain is not run by such mechanism of server-client relationship. It is rather a system comprising of independent parallel processing neurons [31]. The brain is believed to be a huge network of networks (NoNs) [4, 11]. If we break down the whole brain and come to the basic network, we may find some similar system being implemented there. These may be done using a peer-to-peer architecture [4]. The proposed algorithm is possible using the peer-to-peer architecture as well which we do not implement in the present work. The exact mechanism of such a system in brain is still a big mystery.

3 Partitioning

In this section we discuss the three basic types of partitioning available in literature for the parallel implementation of ANN. These form the basis of our proposed algorithms of hierarchical partitioning and self adaptive implementation. The BPA adjusts the weights and biases to optimize the ANN. In this paper we refer to weights and bias by weights only.

3.1. Dataset Partitioning

This training dataset usually consists of huge amount of data that is trained in a batch processing mechanism in BPA. This type of partitioning divides the dataset into a number of smaller datasets. Each processing element (PE) is given a part which has its own copy of the ANN. The final weights of all PEs are noted. The mean of these weights is taken as the final weight of the ANN. This is given in equation (1).

$$W_{ij}^{k} = \frac{\sum_{l=1}^{N} (W_{ij}^{k})_{l}}{N} \qquad (1)$$

Here W_{ij}^{k} is any weight from any node i to any neural node j of the ANN in layer k.
N is the number of partitions

This type of partitioning is especially very useful in very large data sets. As all the PEs are independent of each other, only the final weights after training are communicated between PEs. This updates the weights of all the PEs and they are ready for another set of iterations. The division of data and then the exchange of weights between the PEs is shown in figure 1.

The communication among PEs needs to be kept at a minimal in order to minimize the communication overheads. For the same reasons peer-to-peer connections among the PEs have been used, where every node is connected to the next node to form a circle. Each PE communicates its weight set to the next PE. It also communicates the weight

sets of previous PEs that it receives to the PE next in the circle. It may be seen that this cycle of communication needs to be carried out till n-1 cycles before all PEs know all weights calculated by all N PEs. They then update their weights using equation (1). This communication is shown in figure 2.

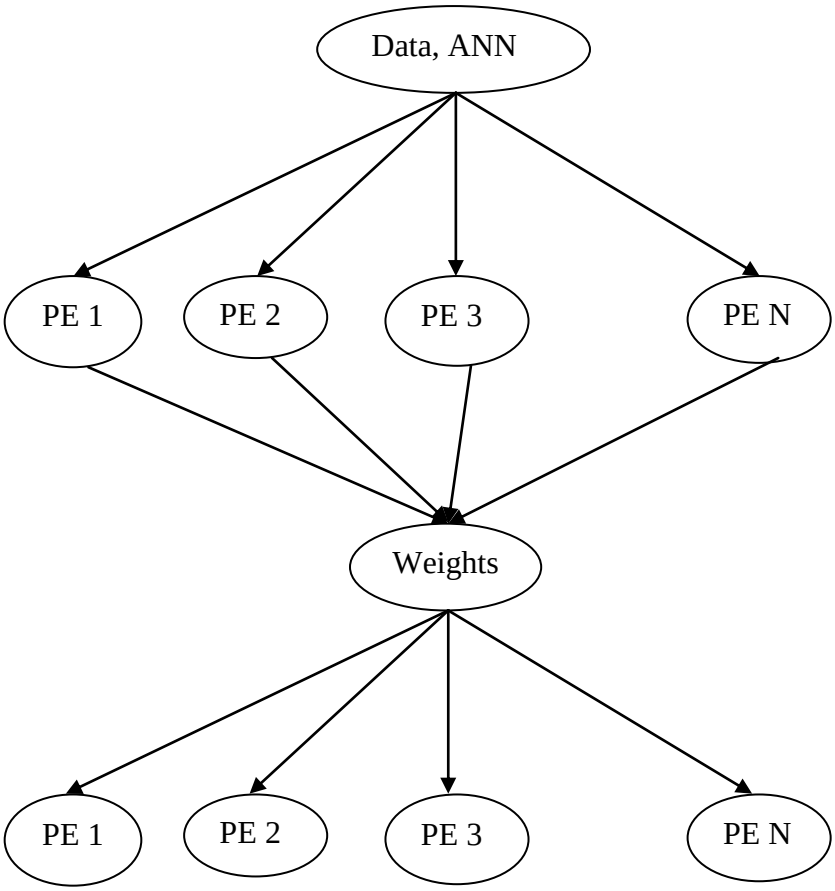

Figure 1: Data Set Partitioning

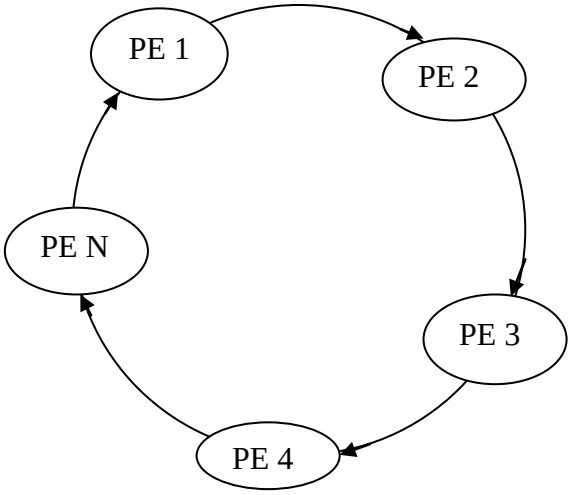

Figure 2: The communication in data set partitioning

3.2 Layer Partitioning

This type of partitioning works on layers. The ANN consists of many layers one after the other. Each layer can work on a data item only when the preceding layer has finished working on it and has supplied its results. The calculations are done by this layer and the results are communicated to the next layer. This happens in both feed forward as well as feedback stages.

The different layers work on different data items in a pipelined manner. Once the first layer completes its calculations for the first data item, it communicates it to the second layer. Now the first layer works on the second data item while the second layer works on the first. A similar working is displayed in the feedback stage. The pipelined architecture facilitates the different layers to work on different data at the same time. The weights are updated at the backward propagation stage. The old weights are temporarily stored during the period the forward and backward stage carry their computation. The concept is shown in figure 3.

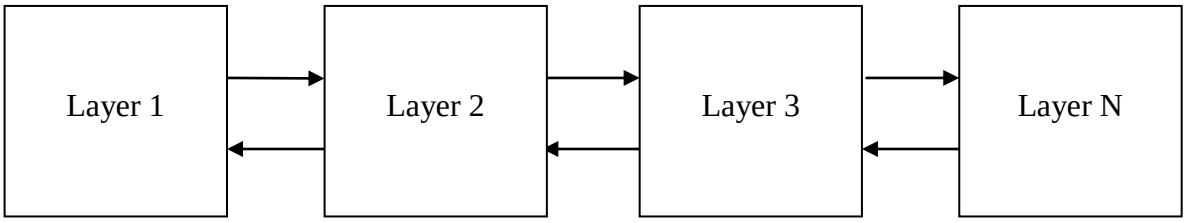

Figure 3: The working of layer partitioning

The general architecture of the layer partitioning is given in figure 4. The computations are transferred from a backward layer to a forward layer (in feed forward stage) and a forward layer to a backward layer (in feedback stage). Hence every PE is connected to the next PE and the previous PE. The communication is shown in figure 3. The mathematical model for this partitioning is given in equation (2) and (3).

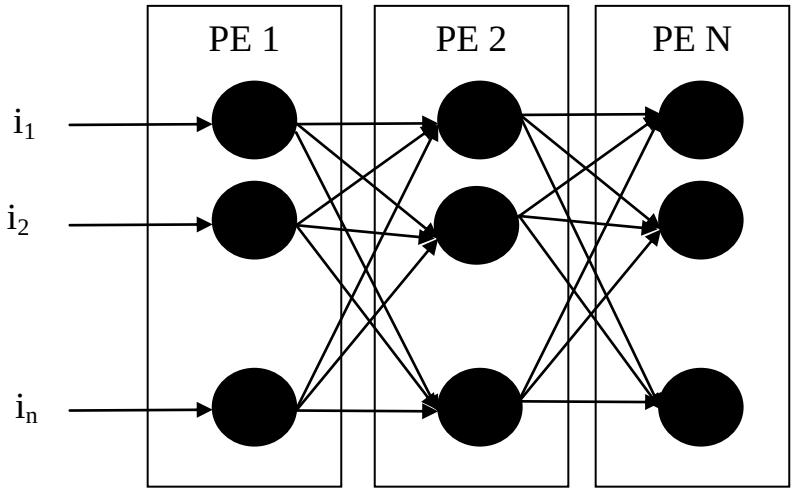

Figure 4: The layer breakup of a neural network

For forward phase we have

$$O_l = W^l * I \tag{2}$$

Here O_l is the output of the l^{th} layer
W^l is the weight matrix of the l^{th} layer ($W=[w]_{mxn}^{\ l}$)
I is the input given to the layer

$$I = \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ \vdots \\ i_n \end{bmatrix}$$

The output of any layer becomes the input for next layer. Similarly in the backward phase we propagate the array of desired valued backward. The error E that is propagated from one layer to previous layer is denoted by equation (3).

$$E = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ \vdots \\ e_n \end{bmatrix} \quad (3)$$

3.3 Node Partitioning

This type of partitioning partitions the various nodes horizontally. Each partition is handled by a separate PE. The partitioned ANN consists of nodes from all layers. In this the computation is slightly more complex. The computation has to precede layer-by-layer. Hence at the start, all nodes at first layer of various PEs may start functioning. Once this is done, then only the nodes at second layer will get a chance to operate. This way the processing propagates simultaneously among all PEs, from one layer element to the other. The computation in the backward phase also follows the same principles in the reverse direction. The general structure of this computation is given in figure 5.

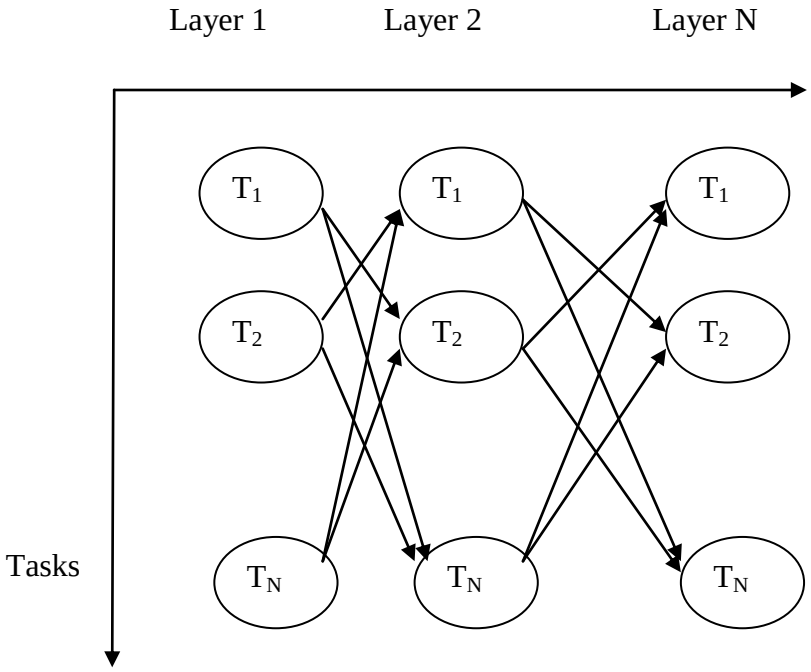

Figure 5: The working of node partitioning

The general structure of this partitioning is given in figure 6. The weights are usually stored reputedly to save computation time. Due to the number of connections, the total communication time is usually large. The communication of data is from one PE to all the PEs. The communication is shown in figure 7. The mathematical model of this type of partitioning is given in equation (4) and (5).

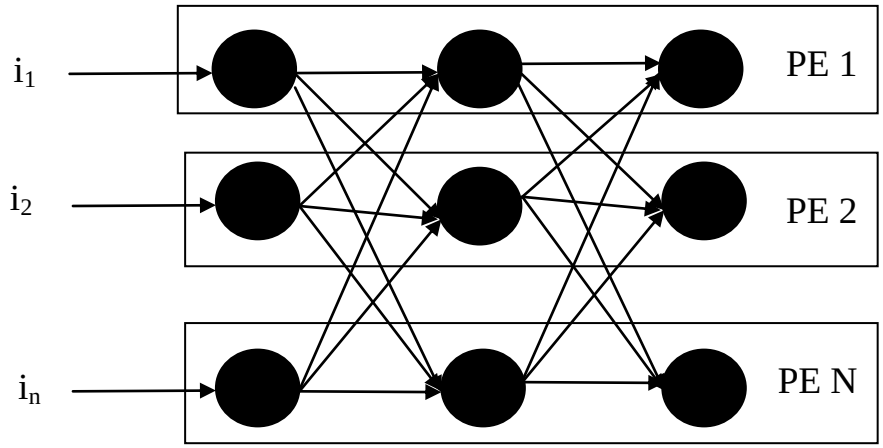

Figure 6: The structure of node partitioning

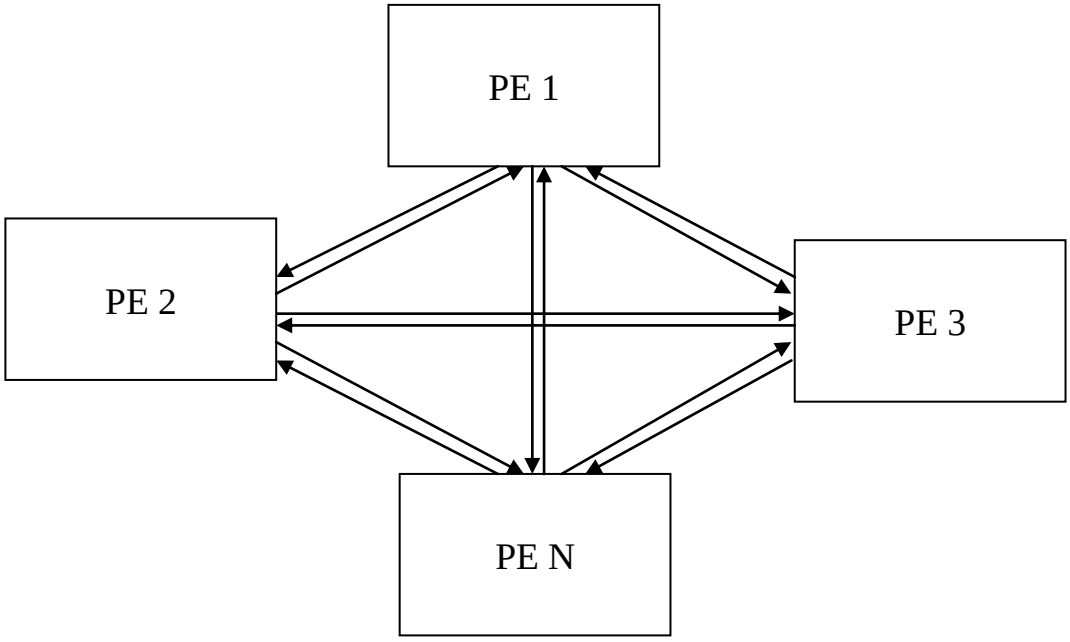

Figure 7: Communication in node partitioning

For forward phase we have

$$O_l = W^l * I \quad (4)$$

Here O_l is the output of the l^{th} layer
W^l is the weight matrix of the l^{th} layer ($W=[w]_{mxn}^{l}$)
I is the input given to the layer

$$I = \begin{bmatrix} i_{11} \\ i_{12} \\ \vdots \\ i_{1n} \\ \hline i_{21} \\ i_{2n} \\ \vdots \\ i_{2n} \\ \hline \cdots \\ \hline i_{N1} \\ i_{N2} \\ \vdots \\ i_{Nn} \end{bmatrix}$$

Similarly in the backward phase we propagate the array of desired valued backward. The error E that is propagated from one layer to previous layer is denoted by equation (5).

$$E = \begin{bmatrix} e_{11} \\ e_{12} \\ \vdots \\ e_{1n} \\ \hline e_{21} \\ e_{2n} \\ \vdots \\ e_{2n} \\ \hline \cdots \\ \hline e_{N1} \\ e_{N2} \\ \vdots \\ e_{Nn} \end{bmatrix} \quad (5)$$

The equation (4) and (5) show the input vector (I) and the error vector (E) respectively divided into N parts that are needed for the functioning of any node in a PE. The PE gets N-1 inputs and errors from other PEs and 1 from its own from a node at the previous or forward layer. Here we have assumed that any PE carries n neurons in a single layer.

4 Hierarchal Partitioning

This is the partitioning that we propose in this paper. The basic aim is to exploit the best features of the various techniques. We first apply data partitioning to divide data among the different PE sets. Later all the PE sets further divide the computation using either node partitioning or layer partitioning. This would enable generation of unique architecture of the network that would be optimal. The hierarchal partitioning helps in partitioning the ANN to the maximum possible extent. This helps us in attaining higher level of parallelism and better performance.

The data set partitioning divides the whole dataset into smaller data sets and assigns each data set to a PE set. We then use either node or layer partitioning in the PEs within the PE set. A general way is to partition half PE sets using node partitioning and the other half using layer partitioning. In layer partitioning each PE represents a layer and the number of such elements is the number of layers. Similarly, in node partitioning each PE represents a set of nodes from all layers.

At the time of training, the data set is first divided and distributed among all the PE sets. Each set behaves independently till the interchange of weights takes place to calculate the ANN architecture. At this time, the circular communication starts and the weights are communicated as discussed in node partitioning.

Internally, each PE set partitions its task using node or layer partitioning. This communication does not affect the communications for the exchange of weights in data set partitioning. The result is computed internally by the applied partitioning technique by the PE set. The final weight vector is calculated and then communicated in the data set partition.

The general structure of this system is given in figure 8. It may be noted here that there is no physical system that acts as the data set partition PE. Rather the extra functionalities of the data set partition are added to the first PE of the node/layer partition. It performs both the tasks of data set partitioning and node/layer partitioning. We use two different sockets for both these communications.

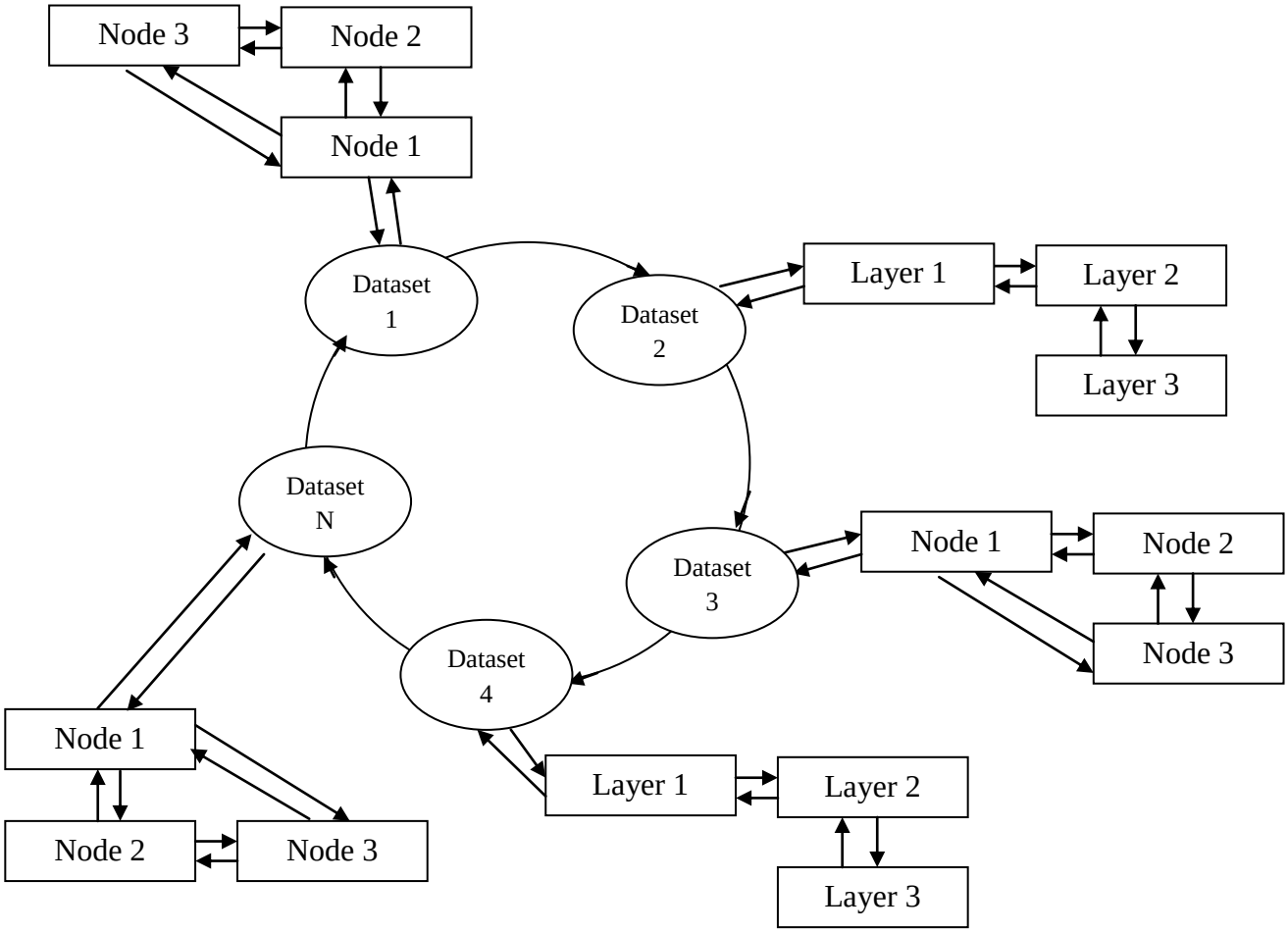

Figure 8: The general structure of hierarchical partitioning

5 Self-Adaptive Approach

This is another concept that we introduce in this paper. As discussed above, all the PEs of the network may not have equal processing powers. Since we are considering that there are a large number of workstations available for partitioning to happen, it is very less likely that they be of the same processing powers. The processing power or configuration may vary drastically. Also if the processing powers are same, it is possible that there may be a temporary load on some PE due to which the performance may be temporarily reduced. Hence performance of a PE is never constant. It keeps changing.

The general algorithms would not give optimal performance in such cases. The self-adaptive nature of the system empowers it to adapt itself towards changing conditions. The adaptation enables performance optimization. The

adaptation is carried by varying the computational load on the PEs. The high performance workstations are given higher computation as compared to the lower performing PEs. The performance is seen in relative terms. It is the performance of a PE compared to other PEs.

For implementation of this concept, we setup a server and client architecture. The server here does the job of load balancing. Load balancing is done on the data set partitioning PEs. The server is responsible for the performance evaluation of all the nodes and then division of computation among them.

A major parameter here is the frequency in which the server monitors and re-allocates the load. If the frequency is too less, it is possible that the system may recover completely from the low performance peaks and the same may go unnoticed. If the frequency is very high, it may result in a huge loss of time in communication. This communication may increase overheads and the total time may shoot up. Hence operating frequency needs to be decided cautiously.

The communication required for this system is carried out in peer-to-peer mode. Each PE reports its performance to the next PE. This goes for n-1 cycles after which all nodes know about the performance of all nodes. While implementing this communication takes place along with the transfer of weights between PEs in data partitioning. Both the information on weights and performance go simultaneously. The performance is communicated by the first node to the server at the desired frequency. The server studies the performances and then redistributes the computation among all PEs.

Every node is provided with a number that denotes the number of data cases that it would be given. If this number is changed, the next time the PE would accept those many numbers of data cases. The circulation of test cases also takes place in the circular queue. The test cases are passed from one PE to another. Each PE picks up the mentioned number of test cases and passes on the others to the next PE. The last PE gets the left number of data cases ($T-\sum n_i$) where T is the total data cases.

The distribution of work load is done by the ratio of the performances reported by the various PEs. The performance here is the actual time taken by the PE in execution, neglecting the time it was waiting for the response from the other PE. The number of data cases n_i^{new} to be given to any node i may be calculated by equation (6).

$$n_i^{new} = \frac{n_i^{old} * p_i}{\sum_{j=1}^{N} p_j} \quad (6)$$

Here n_i^{old} = Number of data cases previously given to node i
n_i^{new} = Number of data cases to be given to node i
N = Total number of PEs
p_i = Performance of node i

Here p_i is calculated as the difference between the time taken by serial execution of the algorithm and processing times of that PE. This is given in equation (7).

$$p_i = S - t_i \quad (7)$$

Here t_i = Processing time of node i
S = time taken by serial execution of the algorithm

The general architecture of this system is shown in figure 9.

6 Algorithm

In the previous section, we discussed all the individual parts of the algorithm. Now we put it all together and present the big picture of the algorithm. The whole algorithm operates in two stages. The first stage is the initialization stage where the entire architecture is built. This stage makes necessary connections between the various PEs and assigns roles to the them as per the intentions of the user. This stage lays the basis for the second stage or the working stage. In this stage each and every step of the BPA is put into practice. We discuss these two stages one by one.

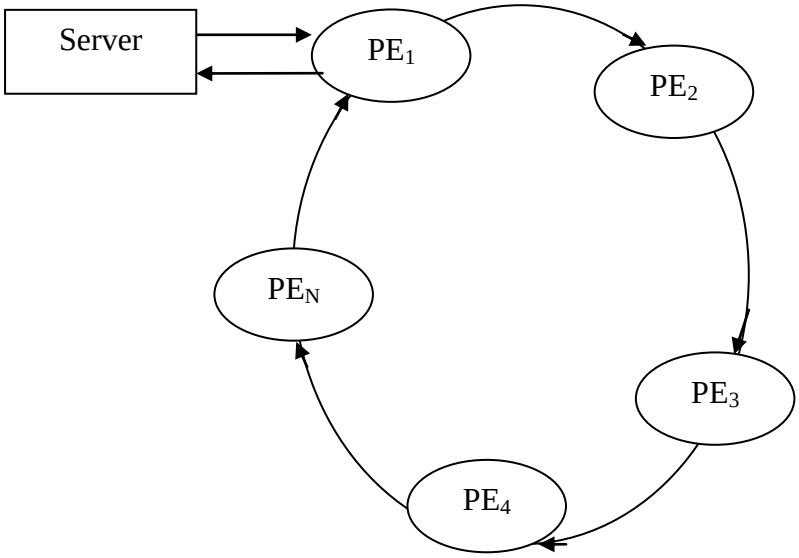

Figure 9: The server/client self adaptive model

6.1 Initialization Stage

Before we even start the training of the ANN, we need to connect the various PEs to each other such that the required transfer of information can take place. This is done using network sockets and socket programming. The basic structure of the nodes and their connections is shown in figure 10. We study the architecture in three levels.

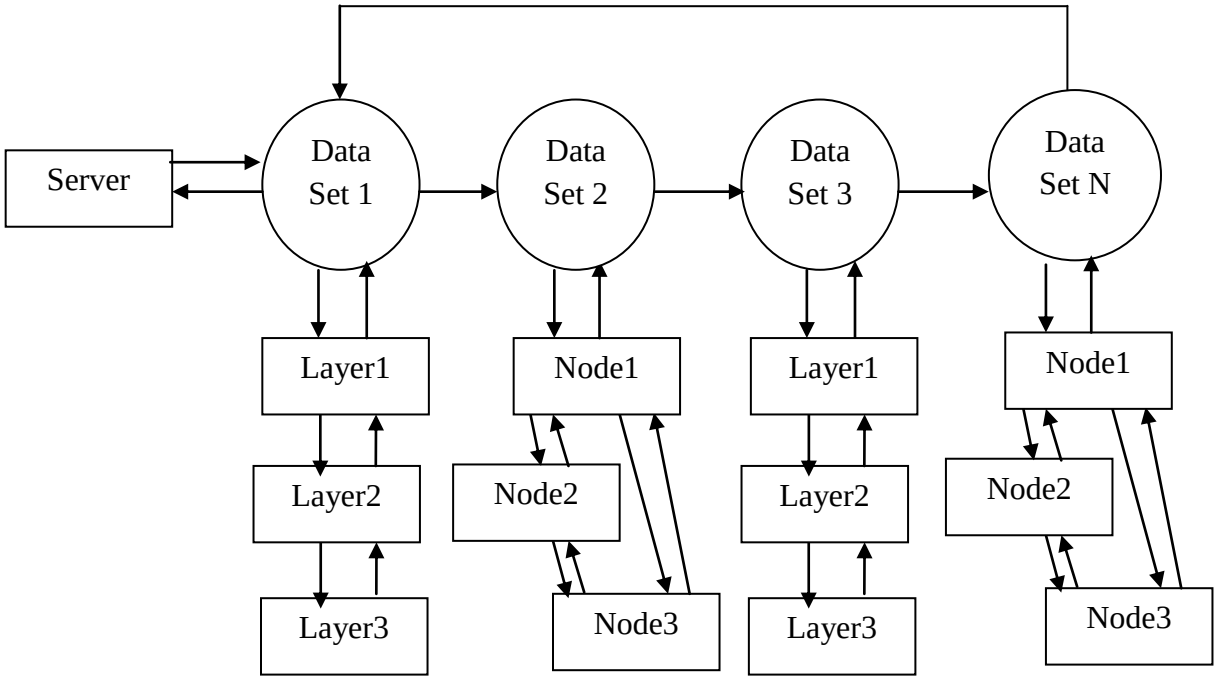

Figure 10: The connections in the algorithm

At the first level the whole architecture of the algorithm is based on a single server-client relationship. The server is responsible for the self-adaptive load balancing approach. For this the server needs a framework to connect to the clients for the passing of information from the PEs and to the PEs to implement load balancing.

At the second level, there is a ring of data set partitioning. Here the various PEs are connected in a circular manner to each other. This enables the various data set partition PEs to pass data set, architecture, initialization, training results and performance information to all the other PEs in a circular distribution mechanism. This ring is connected to the server as the first level.

At the third level the connections are maintained for node or layer partitioning. Here every data set partition is further divided into either layer partitions or node partitions. The layer partitions have a number of nodes connected in series one after the other. This depends on the number of layers in the network and the number of PEs available. The communication of every node is to the forward layer and backward layer. In node partitioning the ANN is divided into various parts in horizontal alignment. These are worked upon by individual PE. The connections in this partitioning are more than in any other partitioning. Any PE needs to be connected to all the other PEs.

When the algorithm starts, the server first sends the initialization instructions using the level 1 connection. This is then communicated between the data set PEs using level 2 connections. Then each data set connection transmits the needed information to all the node or layer PEs using level 3 connections. In this manner all the PEs get the initialization information. The initialization instruction contains two types of information. The first part is the partitioning architecture. These are the roles and duties of every PE participating in the algorithm. It helps the PE to make the needed connections with the other PEs. The second information carried is the ANN architecture. It helps in the initialization of the various PEs. The various PEs allocate memory needed for the algorithm to run. Now the various neurons and corresponding weights have symbolic existence and are initialized by dummy values.

All these connections are established as the first step as soon as the program starts. These connections are maintained throughout the program run. All the communications are carried out using these connections only. When the initialization phase is over, the algorithm enters the second stage. This is the working stage. This stage is also initiated by the server and is communicated to all the PEs.

6.2 Working Stage

The second stage of the algorithm is the working stage. In this stage the algorithm trains the ANN by BPA. The connections established in the previous step are used for communication. The working of the algorithm may also be seen at three distinct levels.

At the topmost or the first level there is a server based load balancing system between the server and the clients. The server interrupts the training process (in synchronous to the completion of all pending computations) and takes the performance analysis of all the PEs. The revised computational load for the PEs is calculated and the same is communicated. This process is repeated in a predefined frequency.

At the second level there is an implementation of the data set partitioning. Here the first job is to pass on the training data set from the server to all the PEs. These PEs take the required number of test cases and pass the rest to the next PE. After this step all the PEs involved in dataset partitioning have a data set with them. Then we start the process of training the ANN independently in all the PEs. Once the training is over the computed weights are passed in a circular manner and every PE updates weights in its copy of the ANN. This process of training for certain number of epochs and then passing on the computed weights is repeated for a number of iterations.

The third level contains the implementation of the node or the layer partitioning which is independent for every data set partition PE. The data set partition ANN is trained as a joint effort of all the internal data set or node partitions. There is exchange of data between the various PEs as per the partitioning demands. The internal working of the two partitioning techniques has already been discussed. The whole training data set is passed in one epoch and the result is computed for the entire training of all the elements of that data set for one epoch.

At the end of the required number of epochs, the training is finished. The final list of weights as computed by the data set partition nodes is regarded as the final weight vector and the same is used for the testing purposes. One of

the critical aspects of the algorithm is to main a proper synchronization among the different steps to ensure that the activities take place in the correct order and no activity starts before the previous one has ended.

7 Results

The testing and analysis of the algorithm was done on a simulation engine built by the authors of their own. JAVA with Eclipse IDE was used as a developmental framework for the simulator. The simulator consisted of a central server program which was stored and executed at the central server used for the algorithm execution. This program had a separate module called input module where the inputs and targets could be specified or generated randomly. This is the only module that stores the training data centrally. Another module called network module stores the architecture of the final ANN. This module enables the users to form a network and alter the same as per requirements. This module also does the network formulation and initialization. Another module at the central server called partitioning module stores the partitioning details. It allocates the partitioning architecture and roles of each and every PE in the system. Text files are used for this purpose and they are parsed by a parser at the central server. All the PEs are identified by their IP addresses. The present simulation engine assumes all PEs in the same Local Area Network, although they may be easily generalized to other subnets as well. The server may execute commands to start or terminate the execution of the various PEs. The user only interacts with the central server while interaction with the other PEs for issues related to configuration, initialization, data exchange or control are carried by the server itself. The central server operates at two levels. The first level is the DEBUG level. Here the server prints the information regarding the runtime or any other specified information from time to time. The other level is the RUN level. In this the server only gives the final summary of training.

The other PEs have another piece of program to work as an agent of data set partitioning, layer partitioning or node partitioning. The code to be executed and the architectural framework with which the execution takes place are set at the time of initialization. These PEs also run at two level. At the DEBUG level, they show details regarding the runtime or any other information desired by the user. At the RUN level, there is no output. All the features discussed in the algorithm were implemented using JAVA Socket programming, JAVA Multithreading programming and related technologies [38].

A major concern for the simulator is the synchronization between the threads. This was mandatory for the various tasks to perform the needed operations as per design in the same order and conveying the same meaning. JAVA thread synchronization using fundamentals of the classic Producer-Consumer problem was used for the same. Deadlock prevention cases were explicitly handled [38].

The execution of the algorithm using this simulator followed a very simple cycle. First the user specifies the training inputs and targets in the input module. Then he specifies the network architecture in the network module. He then specifies the parallel partitioning architecture in the partitioning module. The level is specified as DEBUG to enable output of network information. The execution of program is carried out in any order in all the PEs. The user then performs the “start training” event at the server. The server configures all the PEs. All the sockets are initialized among the various PEs as per user specifications. The initialization data is exchanged between them. When all this is over, the server sends the “start train” command to all the PEs to start their working as per the algorithm.

All simulations were done on the indicated number of computers acting as PEs. All these had 256 MB RAM with Pentium 3 processor.

Before we discuss the data used for the testing, it would be wise to discuss the role of ANN with BPA in problem solving. Machine learning problems may be classified in two separate categories. These are functional prediction problems and classificatory problems. ANN with BPA is known for its generalizing capabilities where it tries to regress the available training data into a smooth function. It is hence observed that ANN with BPA is much suited for the functional prediction problems. Classification is more of a localized problem that is well solved by localized networks like Learning Vector Quantization (LVQ), Self Organizing Map (SOM), etc [41]. We hence concentrate our focus on the functional prediction problems for the testing purposes.

In place of taking standard database from a standard repository, we chose to generate a synthetic database. The reason for the same is twofold. Firstly, standard data sets have limited number of training data items which are normally very small. The algorithm needs a very large number of data sets for good performances. Also the standard data sets can be trained using a limited number of layers and neurons. They denote simple problems that are easily

solvable. It is further not advisable to increase the number of layers or neurons to avoid over-learning or over-generalization [41]. The second problem is of complexity. One cannot change the complexity denoted by these problems which is a limitation in the use of standard database. This algorithm works well only for problems with huge data sets and high complexity to require more number of layers and neurons.

The synthetic function we used to generate random data for this purpose is given in equation (8). This exhibits a large complexity as well as can be easily used to generate large number of data sets as per requirements.

$$f(a,b,c,d,e) = \frac{a*100+b*c+\mathrm{Sin}(d)}{e} \quad (8)$$

We also coded a single-system BPA and noted it's time for execution. This was compared with the time taken by the parallel based system.

We calculated the speed-up for all different architectures and number of PEs. The speedup is defined as the ratio of time taken in on a single system with that taken in parallel implementation given in equation (9). A higher speed-up means that the algorithm was able to train itself in lesser amount of time. The speed-up is better if we introduce more number of workstations.

$$\mathrm{Speedup} = \frac{\text{Sequential execution time}}{\text{Parallel execution time}} \quad (9)$$

The first experiment was conducted on randomly generated inputs whose targets were calculated using equation (8). The network architecture used for this consisted of an input layer with 8 neurons, an output layer with 1 neuron and a hidden layer with 15 neurons. The network so formed was allowed to train for 150000 iterations. After every 50000 iterations the server was allowed to redistribute the work load among the PEs using the discussed mechanisms. The number of PEs was varied from 2 to 20. The same experiment with same data was carried out with a single system as well. This gave us the sequential execution time for all cases. Speedups for the various number of PEs were measured. Figure 11(a) shows the rise in speedup with the increase in the number of PEs. The values of speedups for various number of PEs along with the execution times are given in table 1(a).

Here we proposed a server client relation for redistribution of inputs. We tested the algorithm without this feature as well. We found that the speedup reduced by a considerable amount, which proves the importance of the self adaptive nature of the algorithm. The rise in speedup with increase in the number of PEs without use of load balancing feature is given in figure 11(b). The values of speedups for various number of PEs along with the execution times are given in table 1(b). The network, partitioning architecture and data set were similar to our previous experiment apart from the fact that the redistribution of workload after 50000 iterations was not carried out.

In order to study the affect of addition of neurons in the algorithm, we carried out another experiment. A function similar to equation (8) was used. This time there was one input layer with 11 neurons, 1 output layer with 1 neuron and 1 hidden layer with 25 neurons. The learning was carried forward till 150000 iterations. A total of 500 training data cases were generated. Again there were two set of executions done. First run was with the load balancing factor using load balancing after 50000 iterations as in experiment 1. The plot for the speedup for varying number of PEs for this experiment is given in figure 11(c). Table 1(c) gives the corresponding speedup values. The second was without the load balancing factor. Figure 11(d) plots the speedup values and table 1(d) gives the corresponding speedups.

8 Conclusions

In this paper we proposed a novel approach to parallel implementation of the ANN. We also discussed its relevance with the brain that was a motivation behind the work. We proposed a model that we propose is similar in nature to the working of the human brain in special cases of disabilities.

We proposed a multi system (or multi PE) parallel implementation of ANN for PEs with different working capabilities. The parallel implementation helped us attain higher speedups. The system performed faster than the traditional methods in cases of difference in configuration of the PEs. We applied three types of partitioning

techniques: node partitioning, layer partitioning and dataset partitions in a hierarchical manner to get the best out of the system working in parallel. We proposed this hierarchical nature where the different types of partitioning were applied one after the other to get the final output weight vector of trained ANN. The higher speedup of the ANN was responsible due to the very high partitioning that was employed.

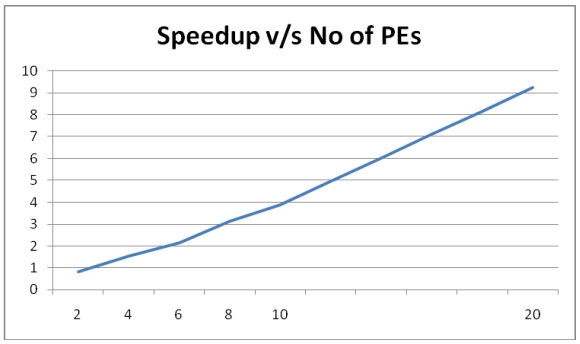

Figure 11(a): Speedup v/s No of PEs (Experiment 1)

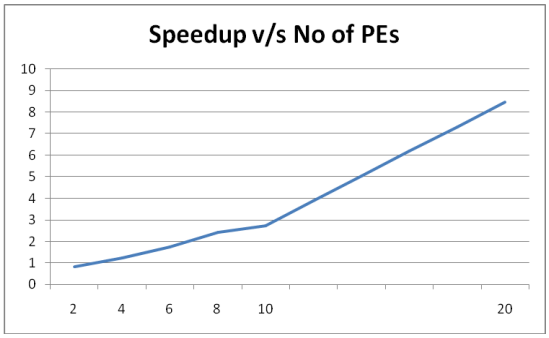

Figure 11(b): Speedup v/s No of PEs (Experiment 1) without self adaptive approach

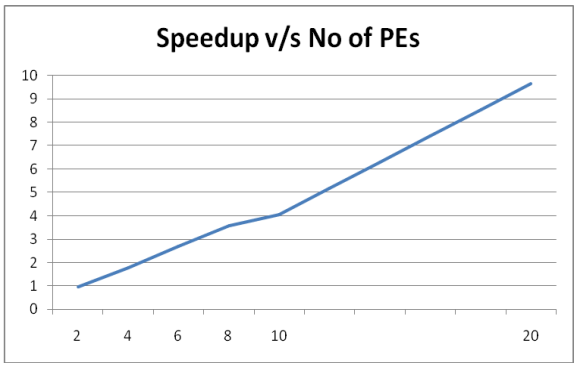

Figure 11(c): Speedup v/s No of PEs (Experiment 2)

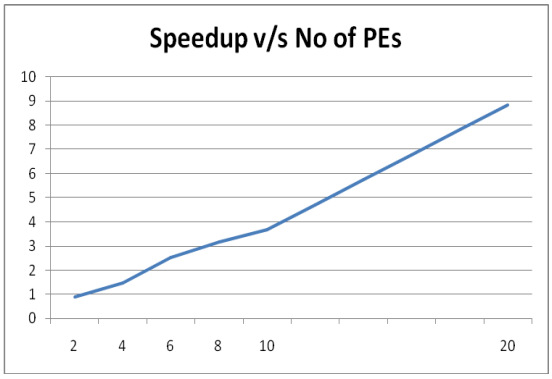

Figure 11(d): Speedup v/s No of PEs (Experiment 2) without self adaptive approach

Table 1(a): Speedups for Experiment 1

No of PEs	No of training data sets	Network Architecture	Epochs	Server Sync after iterations	Time Serial (in ms)	Time Parallel (in ms)	Speedup
2	500	8-15-1	150000	50000	823737	982521	0.838391
4	500	8-15-1	150000	50000	823737	530306	1.553324
6	500	8-15-1	150000	50000	823737	384294	2.143507
8	500	8-15-1	150000	50000	823737	263511	3.126006
10	500	8-15-1	150000	50000	823737	211249	3.899365
20	500	8-15-1	150000	50000	823737	89225	9.232132

Table 1(b): Speedups for Experiment 1 without self adaptive approach

No of PEs	No of training data sets	Network Architecture	Epochs	Server Sync after iterations	Time Serial (in ms)	Time Parallel (in ms)	Speedup
2	500	8-15-1	150000	NA	823737	993483	0.829141
4	500	8-15-1	150000	NA	823737	674358	1.221513
6	500	8-15-1	150000	NA	823737	473899	1.738212
8	500	8-15-1	150000	NA	823737	342174	2.407363
10	500	8-15-1	150000	NA	823737	303214	2.716685
20	500	8-15-1	150000	NA	823737	97180	8.476405

Table 1(c): Speedups for Experiment 2

No of PEs	No of training data sets	Network Architecture	Epochs	Server Sync after iterations	Time Serial (in ms)	Time Parallel (in ms)	Speedup
2	500	11-25-1	150000	50000	1733690	1829312	0.947728
4	500	11-25-1	150000	50000	1733690	984654	1.76071
6	500	11-25-1	150000	50000	1733690	644070	2.691773
8	500	11-25-1	150000	50000	1733690	486311	3.564982
10	500	11-25-1	150000	50000	1733690	429221	4.039155
20	500	11-25-1	150000	50000	1733690	179378	9.665009

Table 1(d): Speedups for Experiment 2 without self adaptive approach

No of PEs	No of training data sets	Network Architecture	Epochs	Server Sync after iterations	Time Serial (in ms)	Time Parallel (in ms)	Speedup
2	500	11-25-1	150000	NA	1733690	1911289	0.907079
4	500	11-25-1	150000	NA	1733690	1175223	1.475201
6	500	11-25-1	150000	NA	1733690	683619	2.536047
8	500	11-25-1	150000	NA	1733690	548465	3.160986
10	500	11-25-1	150000	NA	1733690	471293	3.678582
20	500	11-25-1	150000	NA	1733690	196229	8.835035

The entire system is based on a server-client approach, where the role of the server is to balance the work load among the various PEs. This balancing helped us in controlling the network performance when certain nodes might become less responsive or more responsive in an adaptive manner.

As a result we saw a huge computational speed-up when we applied this over a synthetic database. The high speed-up attained by this algorithm clearly shows that the algorithm has immense potential for improving the speed of training of the neural network especially when different nodes have different processing powers. The speed up was more than any of the traditionally applied techniques.

We have proposed a robust method of implementing BPA across various kinds of nodes. The algorithm so far has no capability to change the architecture of the system. This is something that is believed to happen in the human brain where new connections are made and older ones are destroyed. This would make the system completely dynamic. Such a dynamic system may be designed in future. Also a peer-to-peer architecture may be modeled in place of our approach of server client architecture. This would further boost the speedup.

Acknowledgement
The authors wish to thank Mr. Harsh Vazirani from Indian Institute of Information Technology and Management Gwalior for all his technical inputs and support which largely led to the successful completion of the project.

References
[1] Acierno, Antonio d', "Back-propagation learning algorithm and parallel computers: The Clepsydra mapping scheme", *Neurocomputing* Volume 31, Issues 1-4, March 2000, Pages 67-85
[2] Alex, Graves; Santiago, Fernandez; Marcus, Liwicki, Horst, Bunke & Jurgen, Schmidhuber, "Unconstrained Online Handwriting Recognition with Recurrent Neural Networks", *Advances in Neural Information Processing Systems* 20, 2008
[3] Alves, R.Lde.S.; de Melo, J.D.; Neto, A.D.D. and Albuquerque, A.C.M.L, "New parallel algorithms for back-propagation learning", *Proceedings of the 2002 International Joint Conference on Neural Networks*, 2002. IJCNN '02, pp 2686-2691, 2002
[4] Anderson, J. A. & Sutton, J. P., "A network of networks: Computation and neurobiology," World Congr. Neural Networks, vol. 1, pp. 561-568, 1995
[5] Azari, N.G. and Lee, S.-Y, Hybrid partitioning for particle-in-cell simulation on shared memory systems, *Proc. of 11th International Conference on Distributed Computing Systems*, 20-24 May 1991, pp. 526-533.
[6] Babii, Sorin; Cretu, Vladimir; Petriu, Emil M., "Performance Evaluation of Two Distributed BackPropagation Implementations", *Proceedings of International Joint Conference on Neural Networks*, Orlando, Florida, USA, August 12-17, 2007
[7] Bechtel, William & Abrahamsen, Adede, "Connectionism and the Mind – An Introduction to Parallel Processing in Networks", *Basil Blackwell*, pp 22-65
[8] Blas, Andrea Di; Jagota, Arun and Hughey, Richard, "Optimizing neural networks on SIMD parallel computers", *Parallel Computing* 31 (2005) 97–115
[9] Boller, Franço; Jordan, Grafman & Rizzolatti, G, "Handbook of neuropsychology", *Elsevier*, pp 324
[10] Brodie, B.C.; "Pathological and surgical observations relating to injuries of the brain", *The London Medical and Surgical Journal*, Vol 1, No. 4, October 1828, pp 307-318
[11] Caelli, T.; Ling Guan; Wen, W., "Modularity in neural computing," *Proceedings of the IEEE*, vol.87, no.9, pp.1497-1518, Sep 1999
[12] Caminhas, Walmir M.; Vieira, Douglas A. G.; Vasconcelos, Joao A., "Parallel layer perceptron", *Neurocomputing* Volume 55, Issues 3-4, October 2003
[13] Campobello, Giuseppe; Patane, Giuseppe and Russo, Marco, "An efficient algorithm for parallel distributed unsupervised learning", *Neurocomputing*, Volume 71, Issues 13-15, August 2008, Pages 2914-2928
[14] Chang, Li-Chiu and Chang, Fi-John; "An efficient parallel algorithm for LISSOM neural network", *Parallel Computing* 28 (2002) 1611–1633
[15] Cornelis, Hugo and Schutter, Erik De, "Neurospaces:Towards automated model partitioning for parallel computers", *Neurocomputing* Volume 70, Issues 10-12, June 2007, Pages 2117-2121
[16] Draghici Sorin, "A neural network based artificial vision system for licence plate recognition", *international Journal of Network Security, International Journal of Neural Systems*, Vol. 8, No. 1, 1997
[17] Estevez, Pablo A; Paugam-Moisy, Helene, Puzenat & Didier, Ugarte, Manuel, "A scalable parallel algorithm for training a hierarchical mixture of neural experts", *Parallel Computing* 28 (2002) 861–891
[18] Feng, Zhonghui; Zhou, Bing and Shen, Junyi, "A parallel hierarchical clustering algorithm for PCs cluster system", *Neurocomputing* Volume 70, Issues 4-6, January 2007, Pages 809-818
[19] Floreano, Dario & Mattiussi, Claudio, "Bio-Inspired Artificial Intelligence", *MIT Press*, Chapter 3, 2008, pp163-265
[20] Grossberg, Stephen, "The Link between Brain Learning, Attention, and Consciousness", *Consciousness and Cognition*, Volume 8, Issue 1, March 1999, Pages 1-44, ISSN 1053-8100, DOI: 10.1006/ccog.1998.0372.
[21] Hanzalek Zdenek, "A Parallel algorithm for gradient training of feedforward neural networks", *Pattern Computing*, 24(1998), 823-839
[22] Harkness, K.L., & Tucker, D.M., "Motivation of neural plasticity: neural mechanisms in the self organization of depression". M.D. Lewis & I. Granic (Eds.), Emotion, development, and selforganization, *New York: Cambridge University Press*, pp. 186–208
[23] Ho, K.L.; Hsu, Y.-Y.; Yang, C.-C., "Short term load forecasting using a multilayer neural network with an adaptive learning algorithm," *IEEE Transactions on Power Systems*, vol.7, no.1, pp.141-149, Feb 1992
[24] Holland , JH, "Adaptation in natural and artificial systems", *MIT Press* Cambridge, MA, USA, 1992

[25] Jia, Zhen, Balasuriya, Arjuna and Challa, Subhash, " Sensor fusion-based visual target tracking for autonomous vehicles with the out-of-sequence measurements solution", *Robotics and Autonomous Systems* Volume 56, Issue 2, 29 February 2008, Pages 157-176
[26] Kak, S. C., "New training algorithm in feedforward neural networks, First International Conference on Fuzzy Theory and technology", Durham N.C., October 1992. Also in Wang PP(Editor), *Advance in fuzzy theory and technologies*, Durham, N.C. Bookwright Press, 1993
[27] Kak, S. C. & Pastor, J, "Neural Networks and methods for training neural networks", US Patent No 5,426,721, June 20, 1995
[28] Kak, S. C., "A class of instantaneously trained neural networks", *Information Sciences*, 148,97-102,2002
[29] Kala, Rahul; Shukla, Anupam & Tiwari, Ritu; "Fast Learning Neural Network using modified Corners Algorithm", Proceedings of the *IEEE Global Congress on Intelligent System*, ieeexplore, May 2009, Xiamen, China
[30] Kala, Rahul; Shukla, Anupam; Tiwari, Ritu, "Fuzzy Neuro Systems for Machine Learning for Large Data Sets", Proceedings of the *IEEE International Advance Computing Conference*, ieeexplore, pp 541-545, DOI 10.1109/IADCC.2009.4809069, 6-7 March 2009, Patiala, India
[31] Konar, Amit, "Artificial Intelligence and Soft Computing – Behavioral and Cognitive Modeling of the Human Brain", *CRC Press*, Chapter 14, pp 433-434
[32] Nolfi, S.; Miglino, O.; Parisi, D., "Phenotypic plasticity in evolving neural networks," *From Perception to Action Conference*, 1994., Proceedings , vol., no., pp. 146-157, 7-9 Sept. 1994
[33] Ozdzynski, Piotr; Lin, Andy; Liljeholm, Mimi and Beatty, Jackson, "A parallel general implementation of Kohonen's self-organizing map algorithm: performance and scalability", *Neurocomputing* Volumes 44-46, June 2002, Pages 567-571
[34] Pagac, D., Nebot, E. M. and Durrant. W., H., "An evidential approach to map building for autonomous robots," *IEEE Trans. On Robotics and Automation*, vol.14, no.2, pp. 623-629, Aug. 1998.
[35] Pearson, B. J.; White, J. L.; Weinacht, T. C., & Bucksbaum, P. H., "Coherent control using adaptive learning algorithms", Phys. Rev. A 63, 063412 (2001)
[36] Purwin, Oliver; D'Andrea, Raffaello; Lee, Jin-Woo; "Theory and implementation of path planning by negotiation for decentralized agents", *Robotics and Autonomous Systems* Volume 56, Issue 5, 31 May 2008, Pages 422-436
[37] Riedmiller, Martin & Braun, Heinrich, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm", *Proceedings of the IEEE International Conference on Neural Networks*, 1993
[38] Schildt, Herbert, "The Complete Reference Java 2", Fifth Edition, *Tata McGraw Hill Publications*
[39] Seiffert, Udo, "Artificial neural networks on massively parallel computer hardware", *Neurocomputing* Volume 57, March 2004, Pages 135-150
[40] Shields, Mike W. and Casey, Matthew C.; "A theoretical framework for multiple neural network systems", *Neurocomputing* Volume 71, Issues 7-9, March 2008, Pages 1462-1476
[41] Shukla, Anupam; Tiwari, Ritu & Kala, Rahul, "Real Life Applications of Soft Computing", *CRC Press*
[42] Suresh, S.; Omkar, S.N. and Mani, V, "Parallel Implementation of Back-Propagation Algorithm in Networks of Workstations", *IEEE Transactions on Parallel and Distributed Systems*, Vol 16, No 1, pp 24-34, January, 2005
[43] Thomas D. CoatesJr., "Control and monitoring of a parallel processed neural network via the World Wide Web", *Neurocomputing* Volumes 32-33, June 2000, Pages 1021-1026
[44] Torresen, Jim; Shin-ichiro Mori, Nakashima, Hiroshi; Tomita, Shinji; Landsverk. Olav. "Parallel back propagation training algorithm for MIMD computer with 2D-torus network", *Third Parallel Computing Workshop 1994 (PCW'94)*, Kawasaki, Japan
[45] Tyre, Marcie J. & Hippel, Eric von, "The Situated Nature of Adaptive Learning in Organizations", *Organization Science*, Vol. 8, No. 1 (Jan. - Feb., 1997), pp. 71-83
[46] Valova, Iren; Szer, Daniel; Gueorguieva, Natacha; Buer, Alexandre, "A parallel growing architecture for self-organizing maps with unsupervised learning ", *Neurocomputing* Volume 68, October 2005, Pages 177-195
[47] Weitzenfeld, Alfredo, "From schemas to neural networks: A multi-level modelling approach to biologically-inspired autonomous robotic systems", *Robotics and Autonomous Systems* Volume 56, Issue 2, 29 February 2008, Pages 177-197
[48] Yasunaga, Moritoshi; Yoshida, Eiji, and Yoshihara, Ikuo "Parallel Back-propagation Using Genetic Algorithm: Real-time BP Learning on the Massively Parallel Computer CP-PACS," *Proc. The IEEE and INNS Intl. Joint Conf. on Neural Networks*, CD-ROM, July 1999.