Minimal Architecture and Training Parameters of Multilayer Perceptron for its Efficient Parallelization

Volodymyr Turchenko and Lucio Grandinetti

Department of Electronics, Informatics and Systems, University of Calabria
via P. Bucci 22B, 87036, Rende (CS), Italy

Abstract. The development of a parallel algorithm for batch pattern training of a multilayer perceptron with the back propagation algorithm and the research of its efficiency on a general-purpose parallel computer are presented in this paper. The multilayer perceptron model and the usual sequential batch pattern training algorithm are theoretically described. An algorithmic description of the parallel version of the batch pattern training method is introduced. The efficiency of the developed parallel algorithm is investigated by progressively increasing the dimension of the parallelized problem on a general-purpose parallel computer NEC TX-7. A minimal architecture for the multilayer perceptron and its training parameters for an efficient parallelization are given.

1 Introduction

Artificial neural networks (NNs) have excellent abilities to model difficult nonlinear systems. They represent a very good alternative to traditional methods for solving complex problems in many fields, including image processing, predictions, pattern recognition, robotics, optimization, etc [1]. However, most NN models require high computational load, especially in the training phase (up to days and weeks). This is, indeed, the main obstacle to face for an efficient use of NNs in real-world applications. Taking into account the parallel nature of NNs, many researchers have already focused their attention on their parallelization [2-4]. Most of the existing parallelization approaches are based on specialized computing hardware and transputers, which are capable to fulfill the specific neural operations more quickly than general-purpose parallel and high performance computers. However computational clusters and Grids have gained tremendous popularity in computation science during last decade [5]. Computational Grids are considered as heterogeneous systems, which may include high performance computers with parallel architecture and computational clusters based on standard PCs. Therefore, existing solutions for NNs parallelization on transputer architectures should be re-designed. Parallelization efficiency should be explored on general-purpose parallel and high performance computers in order to provide an efficient usage within computational Grid systems.

Many researchers have already developed parallel algorithms for NNs training on weights (connections), neuron (node), training set (pattern) and modular levels [6-10]. The first two levels are a fine-grain parallelism and the second two levels are a coarse-grain parallelism. Connection parallelism (parallel execution on sets of weights) and node parallelism (parallel execution of operations on sets of neurons) schemes are not efficient while executing on a general-purpose high performance computer due to high synchronization and communication overhead among parallel processors [10]. Therefore coarse-grain approaches of pattern and modular parallelism should be used to parallelize NNs training on general-purpose parallel computers and computational Grids [9]. For example, one of the existing implementation of the batch pattern back propagation (BP) training algorithm [6] has efficiency of 80% while executing on a 10 processors of transputer TMB08. However, the efficiency of this algorithm on general-purpose high-performance computers is not researched yet.

The goal of this paper is to research the parallelization efficiency of parallel batch pattern BP training algorithm on a general-purpose parallel computer in order to form the recommendations for further usage of this algorithm on heterogeneous Grid system.

2 Architecture of Multilayer Perceptron and Batch Pattern Training Algorithm

It is expedient to research parallelization of multi-layer perceptron (MLP) because this kind of NN has the advantage of being simple and provides good generalizing properties. Therefore it is often used for many practical tasks including prediction, recognition, optimization and control [1]. However, parallelization of an MLP with the standard sequential BP training algorithm does not provide efficient parallelization due to high synchronization and communication overhead among parallel processors [10]. Therefore it is expedient to use the batch pattern training algorithm, which updates neurons' weights and thresholds at the end of each training epoch, i.e. after the presentation of all the input and output training patterns, instead of updating weights and thresholds after the presentation of each pattern in the usual sequential training mode.

The output value of a three-layer perceptron (Fig. 1) can be formulated as:

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

where N is the number of neurons in the hidden layer, w_{j3} is the weight of the synapse from neuron j of the hidden layer to the output neuron, w_{ij} are the weights from the input neurons to neuron j in the hidden layer, x_i are the input values, T_j are the thresholds of the neurons of the hidden layer and T is the threshold of the output neuron [1, 11]. In this study the logistic activation function $F(x) = 1/(1+e^{-x})$ is used for the neurons of the hidden (F_2) and output layers (F_3), but in general case these activation functions could be different.

The batch pattern BP training algorithm consists of the following steps [11]:

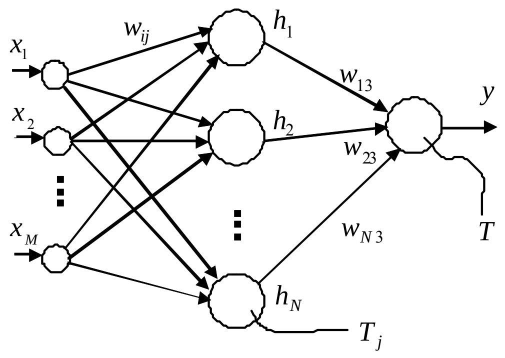

Fig. 1. The structure of a three-layer perception.

1. Set the desired error (Sum Squared Error) SSE=$E_{\min}$ and the number of training iterations t;
2. Initialize the weights and the thresholds of the neurons with values in range (0…0.5) [12];
3. For the training pattern pt:
   - 3.1. Calculate the output value $y^{pt}(t)$ by expression (1);
   - 3.2. Calculate the error of the output neuron $\gamma_3^{pt}(t) = y^{pt}(t) - d^{pt}(t)$, where $y^{pt}(t)$ is the output value of the perceptron and $d^{pt}(t)$ is the target output value;
   - 3.3. Calculate the hidden layer neurons' error $\gamma_j^{pt}(t) = \gamma_3^{pt}(t) \cdot w_{j3}(t) \cdot F_3'(S^{pt}(t))$, where $S^{pt}(t)$ is the weighted sum of the output neuron;
   - 3.4. Calculate the delta weights and delta thresholds of all perceptron's neurons and add the result to the value of the previous pattern $s\Delta w_{j3} = s\Delta w_{j3} + \gamma_3^{pt}(t) \cdot F_3'(S^{pt}(t)) \cdot h_j^{pt}(t)$, $s\Delta T = s\Delta T + \gamma_3^{pt}(t) \cdot F_3'(S^{pt}(t))$, $s\Delta w_{ij} = s\Delta w_{ij} + \gamma_j^{pt}(t) \cdot F_2'(S_j^{pt}(t)) \cdot x_i^{pt}(t)$, $s\Delta T_j = s\Delta T_j + \gamma_j^{pt}(t) \cdot F_2'(S_j^{pt}(t))$, where $S_j^{pt}(t)$ and $h_j^{pt}(t)$ are the weighted sum and the output value of the j hidden neuron respectively;
   - 3.5. Calculate the SSE using $E^{pt}(t) = \frac{1}{2}\left(y^{pt}(t) - d^{pt}(t)\right)^2$;
4. Repeat the step 3 above for each training pattern pt, where $pt \in \{1,...,PT\}$, PT is the size of the training set;
5. Update the weights and thresholds of neurons using $w_{ij}(PT) = w_{ij}(0) - \alpha(t) \cdot s\Delta w_{ij}$, $T_j(PT) = T_j(0) + \alpha(t) \cdot s\Delta T_j$, where $\alpha(t)$ is the learning rate;
6. Calculate the total SSE $E(t)$ on the training iteration t using $E(t) = \sum_{pt=1}^{PT} E^{pt}(t)$;
7. If $E(t)$ is greater than the desired error $E_{\min}$ then increase the number of training iteration to $t+1$ and go to step 3, otherwise stop the training process.

3 Parallel Batch Pattern Back Propagation Training Algorithm

It is obvious from analysis of the batch pattern BP training algorithm in Section 2 above, that the sequential execution of points 3.1-3.5 for all training patterns in the training set could be parallelized, because the sum operations $s\Delta w_{ij}$ and $s\Delta T_j$ are independent of each other. For the development of the parallel algorithm it is necessary to divide all the computational work among the *Master* (executing assigning functions and calculations) and the *Slaves* (executing only calculations) processors.

The algorithms for *Master* and *Slave* processors functioning are depicted in Fig. 2. The *Master* starts with definition (i) the number of patterns *PT* in the training data set and (ii) the number of processors *p* used for the parallel executing of the training algorithm. The *Master* divides all patterns in equal parts corresponding to number of the *Slaves* and assigns one part of patterns to himself. Then the *Master* sends to the *Slaves* the numbers of the appropriate patterns to train.

Each *Slave* executes the following operations for each pattern *pt* among the *PT/p* patterns assigned to him:

- calculate the points 3.1-3.5 and 4, only for its assigned number of training patterns. The values of the partial sums of delta weights $s\Delta w_{ij}$ and delta thresholds $s\Delta T_j$ are calculated here;
- calculate the partial SSE for its assigned number of training patterns.

After processing all its assigned patterns, each *Slave* waits for the other *Slaves* and the *Master* at the synchronization point. At the same time the *Master* computes the partial values of $s\Delta w_{ij}$ and $s\Delta T_j$ for its own (assigned to himself) number of training patterns.

The global operations of reduction and summation are executed just after the synchronization point. Then the summarized values of the $s\Delta w_{ij}$ and $s\Delta T_j$ are sent to all the processors working in parallel. Using a global reducing operation and simultaneously returning the reduced values back to the *Slaves* allows a decrease of the time overhead in the synchronization point. Then the summarized values of $s\Delta w_{ij}$ and $s\Delta T_j$ are placed into the local memory of each processor. Each *Slave* and the *Master* use these values $s\Delta w_{ij}$ and $s\Delta T_j$ in order to update the weights and thresholds according to the point 5 of the algorithm. These updated weights and thresholds will be used in the next iteration of the training algorithm. As the summarized value of $E(t)$ is also received as a result of the reducing operation, the *Master* decides whether to continue the training or not.

The software routine is developed using the C programming language with the standard MPI library. The parallel part of the algorithm starts with the call of the *MPI_Init()* function. The parallel processors use the synchronization point *MPI_Barrier()*. The reducing of the deltas of weights $s\Delta w_{ij}$ and thresholds $s\Delta T_j$ is provided by function *MPI_Allreduce()*, which allows to avoid an additional step for sending back the updated weights and thresholds from the *Master* to each *Slave*. Function *MPI_Finalize()* finishes the parallel part of the algorithm.

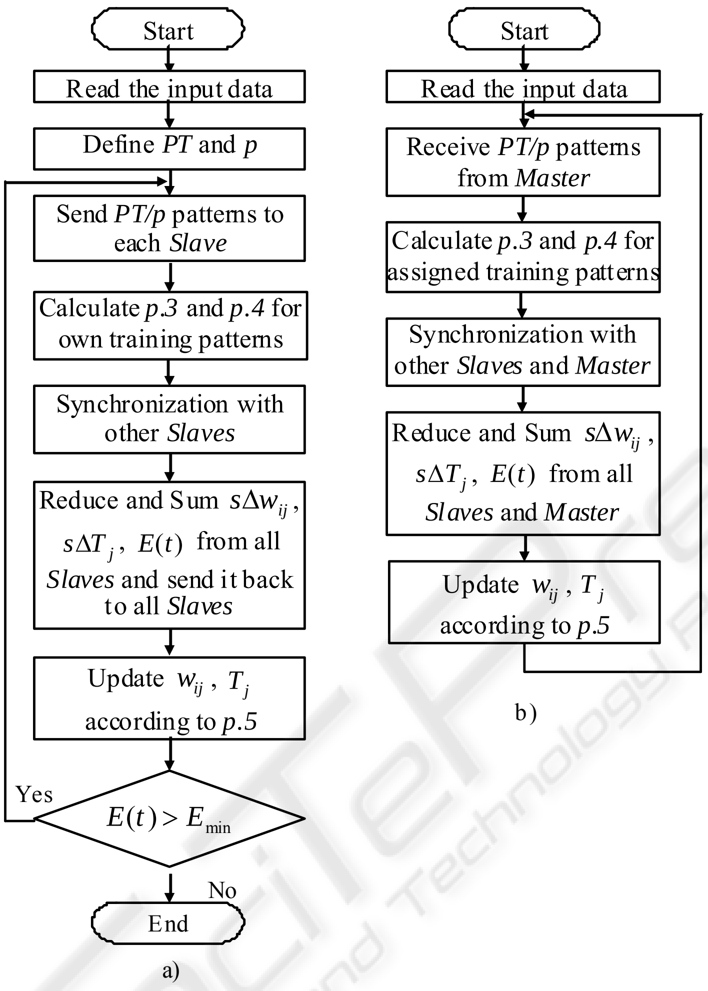
Start
Read the input data
Define PT and p
Send PT/p patterns to each *Slave*
Calculate $p.3$ and $p.4$ for own training patterns
Synchronization with other *Slaves*
Reduce and Sum $s\Delta w_{ij}$, $s\Delta T_j$, $E(t)$ from all *Slaves* and send it back to all *Slaves*
Update w_{ij}, T_j according to $p.5$
Yes
$E(t) > E_{\min}$
No
End
a)
Start
Read the input data
Receive PT/p patterns from *Master*
Calculate $p.3$ and $p.4$ for assigned training patterns
Synchronization with other *Slaves* and *Master*
Reduce and Sum $s\Delta w_{ij}$, $s\Delta T_j$, $E(t)$ from all *Slaves* and *Master*
Update w_{ij}, T_j according to $p.5$
b)

Fig. 2. The algorithms of the *Master* (a) and the *Slave* (b) processors.

4 Experimental Researches

Our experiments were carried out on a parallel supercomputer NEC TX-7, located in the Center of Excellence of High Performance Computing, University of Calabria, Italy (www.hpcc.unical.it). NEC TX-7 consists in 4 identical units. Each unit has 4 Gb RAM, 4 64-bit processors Intel Itanium2 with a clock rate of 1 GHz. This 16th-processor computer with 64 Gb of total RAM has a performance peak of 64 GFLOPS. The NEC TX-7 is functioning under the Linux operation system.

As shown in [12], the parallelization efficiency of parallel batch pattern BP algorithm for MLP does not depend on the number of training epochs. Parallelization efficiencies of this algorithm are respectively 95%, 84% and 63% on 2, 4 and 8 processors of the general-purpose NEC TX-7 parallel computer for a 5-10-1 MLP with 794 training patterns and an increasing number of training epochs from 10^4 to 10^6.

As shown in [7], parameters such as the number of training patterns and the number of adjustable connections of NN (number of weights and thresholds) define the computational complexity of the training algorithm and, therefore, exert influence on its parallelization efficiency. Therefore, research efficiency scenarios should be based on these parameters. In this case the purpose of our experimental research is to answer the question: what is the minimal/enough number of MLP connections and what is the minimal/enough number of training patterns in the input data set for the parallelization of batch pattern BP training algorithm to be efficient on a general-purpose high performance computer?

The following architectures of MLP are researched in order to provide the analysis of efficiency: 3-3-1 (3 input neurons × 3 hidden neurons = *9* weights between the input and the hidden layer + *3* weights between the hidden and the output layer + *3* thresholds of the hidden neurons and *1* threshold of the output neuron = 16 connections), 5-5-1 (36 connections), 5-10-1 (71 connections), 10-10-1 (121 connections), 10-15-1 (181 connections), 15-15-1 (256 connections), 20-20-1 (441 connections). The number of training patterns is changed as 25, 50, 75, 100, 200, 400, 600 and 800. It is necessary to note that such MLP architectures and number of training patterns are typical for most of neural-computation applications. During the research the neurons of the hidden and output layers have logistic activation functions. The number of training epochs is fixed to 10^5. The learning rate is constant and equal $\alpha(t) = 0.01$.

The parallelization efficiency of the batch pattern BP training algorithm is depicted in Figs. 3-5 on 2, 4 and 8 processors of NEC TX-7 respectively. The expressions $S=Ts/Tp$ and $E=S/p\times 100\%$ are used to calculate a speedup and efficiency of parallelization, where Ts is the time of sequential executing the routine, Tp is the time of parallel executing of the same routine on p processors of parallel computer. It is necessary to use the obtained results as the following: (i) first to choose the number of parallel processors used (Fig. 3 or Fig. 4 or Fig. 5), (ii) then to choose the curve, which characterizes the necessary number of perceptron's connections and (iii) then to get the value of parallelization efficiency from ordinate axes which corresponds to the necessary number of training patterns on abscissa axes. For example, the parallelization efficiency of the MLP 5-5-1 (36 connections) is 65% with 500 training patterns on 4 processors of NEC TX-7 (see Fig. 4). Therefore the presented curves are the approximation characteristics of a parallelization efficiency of the certain MLP architecture on the certain number of processors of a general-purpose parallel computer.

As it is seen from the Figs. 3-5, the parallelization efficiency is increasing when the number of connections and the number of the training patterns is increased. However, the parallelization efficiency is decreasing for the same scenario at increasing the number of parallel processors from 2 to 8. The analysis of the Figs. 3-5 allows defining the minimum number of the training patterns which is necessary to use for efficient parallelization of the batch pattern training algorithm at the certain number of MLP connections (Table 1).

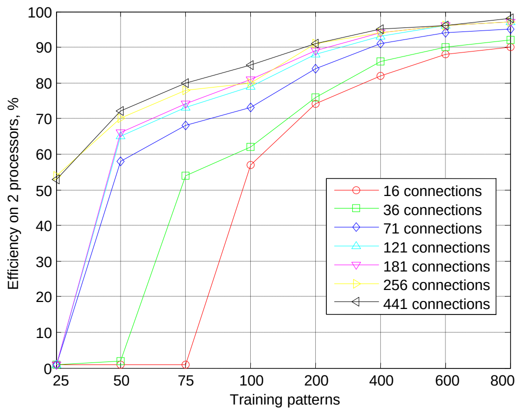

Fig. 3. Parallelization efficiency on 2 processors of NEC TX-7.

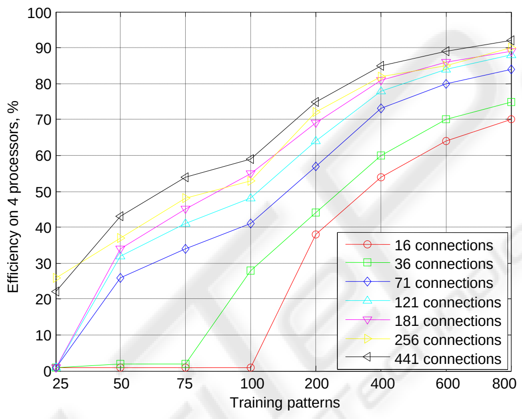

Fig. 4. Parallelization efficiency on 4 processors of NEC TX-7.

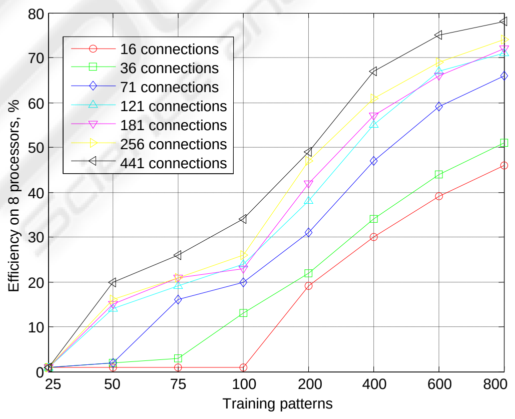

Fig. 5. Parallelization efficiency on 8 processors of NEC TX-7.

For example, the Table 1 shows that the number of training patterns should be 100 and more (100+) for efficient parallelization of MLP with the number of connections more than 16 and less and equal than 36. As it is seen from the Table 1, it is necessary to use more training patterns in a case of small MLP architectures. The minimum number of the training patterns is increasing in a case of parallelization on the bigger number of parallel processors.

Table 1. Minimum number of training patterns for efficient parallelization on NEC TX-7.

2 processors		4 processors		8 processors	
Connections number, C	Training patterns	Connections number, C	Training patterns	Connections number, C	Training patterns
$16 < C \leq 36$	100+	$16 < C \leq 36$	200+	$16 < C \leq 36$	200+
$36 < C \leq 71$	75+	$36 < C \leq 71$	100+	$36 < C \leq 71$	100+
$71 < C \leq 256$	50+	$71 < C \leq 256$	50+	$71 < C \leq 121$	75+
$C > 256$	25+	$C > 256$	25+	$C > 121$	50+

5 Conclusions

The parallel batch pattern back propagation training algorithm of multilayer perceptron is developed in this paper. The analysis of parallelization efficiency is done for 7 scenarios of increasing the perceptron's connections (number of weights and thresholds), in particular 16, 36, 71, 121, 181, 256 and 441 and increasing the number of training patterns, in particular 25, 50, 75, 100, 200, 400, 600, 800. The presented results can be used for estimation a parallelization efficiency of concrete perceptron model with concrete number of training patterns on the certain number of parallel processors of a general-purpose parallel computer. The experimental research proves that the parallelization efficiency of batch pattern back propagation training algorithm is (i) increasing at increasing the number of connections and increasing the number of the training patterns and (ii) decreasing for the same scenario at increasing the number of parallel processors from 2 to 8. The results of analysis of minimum number of training patterns for efficient parallelization of this algorithm show that (i) it is necessary to use more training patterns in case of small architectures of multilayer perceptron and (ii) the minimum number of the training patterns should be increased in a case of parallelization on the bigger number of parallel processors.

The provided level of parallelization efficiency is enough for using this parallel algorithm in Grid environment on the general-purpose parallel and high performance computers. For the future research it is expedient to estimate the factors of decreasing the parallelization efficiency of batch pattern back propagation training algorithm at small number of training patterns and small number of adjustable connections of multilayer perceptron.

Acknowledgements

This research is financially supported by a Marie Curie International Incoming Fellowship grant of the corresponding author Dr. V. Turchenko, Ref. Num. 221524 “PaGaLiNNeT - Parallel Grid-aware Library for Neural Networks Training", within the 7th European Community Framework Programme. This support is gratefully acknowledged.
We wish to thank anonymous referees for thoughtful and helpful comments which improved the readability of the paper.

References

1. Haykin, S.: Neural Networks. Prentice Hall, New Jersey (1999).
2. Mahapatra, S., Mahapatra, R., Chatterji, B.: A Parallel Formulation of BP Learning on Distributed Memory Multiprocessors. Parallel Computing. 22 (12) (1997) 1661–1675.
3. Hanzálek, Z.: A Parallel Algorithm for Gradient Training of Feed-forward Neural Networks. Parallel Computing. 24 (5-6) (1998) 823–839.
4. Murre, J.M.J.: Transputers and Neural Networks: An Analysis of Implementation Constraints and Perform. IEEE Transactions on Neural Networks. 4 (2) (1993) 284–292.
5. Dongarra, J., Shimasaki, M., Tourancheau, B.: Clusters and Computational Grids for Scientific Computing. Parallel Computing. 27 (11) (2001) 1401–1402.
6. Topping, B.H.V., Khan, A.I., Bahreininejad, A.: Parallel Training of Neural Networks for Finite Element Mesh Decomposition. Computers and Structures. 63 (4) (1997) 693–707.
7. Rogers, R.O., Skillicorn, D.B.: Using the BSP Cost Model to Optimise Parallel Neural Network Training. Future Generation Computer Systems. 14 (5) (1998) 409–424.
8. Ribeiro, B., Albrecht, R.F., Dobnikar, A., et al: Parallel Implementations of Feed-forward Neural Network using MPI and C# on .NET Platform. In: Proceedings of the International Conference on Adaptive and Natural Computing Algorithms. Coimbra (2005) 534–537.
9. Turchenko, V.: Computational Grid vs. Parallel Computer for Coarse-Grain Parallelization of Neural Networks Training. In: Meersman, R., Tari, Z., Herrero, P. (eds.): OTM 2005. Lecture Notes in Computing Science, vol. 3762. Springer-Verlag, Berlin Heidelberg New York (2005) 357–366.
10. Turchenko, V.: Fine-Grain Approach to Development of Parallel Training Algorithm of Multi-Layer Perceptron. Artificial Intelligence, the Journal of National Academy of Sciences of Ukraine. 1 (2006) 94–102.
11. Golovko, V., Galushkin, A.: Neural Networks: Training, Models and Applications. Radiotechnika, Moscow (2001) (in Russian).
12. Turchenko, V.: Scalability of Parallel Batch Pattern Neural Network Training Algorithm. Artificial Intelligence, the Journal of National Academy of Sciences of Ukraine. 2 (2009).