4T

UDC 681.3

Volodymyr Turchenko
Research Institute of Intelligent Computer Systems, Ternopil National Economic University, Ukraine
vtu@tneu.edu.ua
Center of Excellence of High Performance Computing, University of Calabria, Rende (CS), Italy

Scalability of Parallel Batch Pattern Neural Network Training Algorithm

The development of parallel batch pattern back propagation training algorithm of multilayer perceptron and its scalability research on general-purpose parallel computer are presented in this paper. The model of multilayer perceptron and batch pattern training algorithm are theoretically described. The algorithmic description of the parallel batch pattern training method is presented. The scalability research of the developed parallel algorithm is fulfilled at progressive increasing the dimension of the parallelized problem on general-purpose parallel computer NEC TX-7.

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 in front of 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 its parallelization [2-4]. But the most of the existing parallelization approaches are based on the 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 of NNs parallelization on transputer architectures should be re-designed and its parallelization efficiency should be verified on general-purpose parallel and high performance computers in order to provide its efficient usage within computational Grid systems.

Many researchers already have developed parallel algorithms of NNs training on weights (connections), neuron (node), training set (pattern) and modular levels [6-10]. 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 the 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 training algorithm [6] has good efficiency of 80 % while executing on 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 scalability of parallel batch pattern neural network training algorithm on general-purpose parallel computer in order to form the recommendations for further usage of this algorithm on heterogeneous Grid system. The scalability of parallel algorithm is considered as its ability to maintain the same parallelization efficiency when we progressively increase both the dimension of the parallelization problem and the number of processors of parallel machine [11].

1. Architecture of Multilayer Perceptron and Batch Pattern Training Algorithm

It is expedient to research parallelization of multi-layer perceptron because this kind of NN has the advantage of being simple and provides very good generalized properties. Therefore it is often used for many practical tasks including prediction, recognition, optimization and control [1]. However a parallelization of single multi-layer perceptron with standard sequential back propagation training algorithm does not provide good parallelization efficiency [10] due to high synchronization and communication overhead among parallel processors. Therefore it is expedient to use batch pattern training algorithm, which provides changing neurons' weights and thresholds in the end of each training epoch, i.e. after presenting all training patterns on the input and output of perceptron in the training mode.

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

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

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], [12]. The logistic activation function $F(x) = \dfrac{1}{1+e^{-x}}$ is used for the neurons of the hidden (F_2) and output layer (F_3).

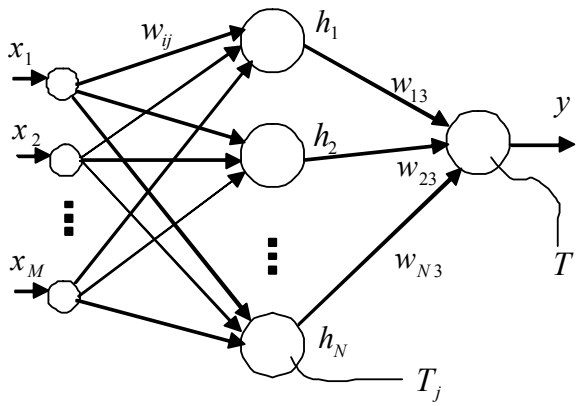

Figure 1 – The structure of three-layer perceptron

The back propagation batch pattern training algorithm consists of the following steps [12]:

1. Set the desired value of total Sum-Squared Error (SSE) to $E_{\min}$ and the number of training iterations t.

2. Initialize the weights and the thresholds of the neurons by values in the range (0,…0,5) [12].

3. For the training pattern pt:

3.1. Calculate the output value $y^{pt}(t)$ using 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 error of the hidden layer neurons $\gamma_j^{pt}(t) = \gamma_3^{pt}(t) \cdot w_{j3}(t) \cdot F_3'(S^{pt}(t))$, 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 the thresholds of all 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.

2. Parallel Back Propagation Batch Pattern Training Algorithm

It is obvious from the analysis of the batch training algorithm from Section 1 above, that sequential execution the points 3.1 – 3.5 for all training patterns in the training set could be transformed to parallel execution, because the sum operations $s\Delta w_{ij}$ and $s\Delta T_j$ are independent on each other. For development of the parallel algorithm it is necessary to divide all computational job among the *Master* (executing assigning functions and calculations) and the *Slaves* (executing only calculations) processors.

The algorithms of functioning the *Master* and the *Slave* processors are depicted in Fig. 2a and Fig. 2b respectively. 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 parallel executing the training algorithm. The *Master* divides all patterns in equal parts corresponding to number of *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 of *pt* patterns:
– calculation the points 3.1 – 3.5 of the algorithm from Section 1 above, the point 4 is executed only for assigned number of training patterns. The values of partial sums of delta weights $s\Delta w_{ij}$ and delta thresholds $s\Delta T_j$ are calculated as a result of this step;
– to calculate partial SSE for assigned number of training patterns.

After processing all assigned patterns each *Slave* is waiting other *Slaves* and the *Master* in the synchronization point. At the same time the *Master* executes own (assigned to himself) number of training patterns and calculates own partial values of delta weights $s\Delta w_{ij}$ and delta thresholds $s\Delta T_j$.

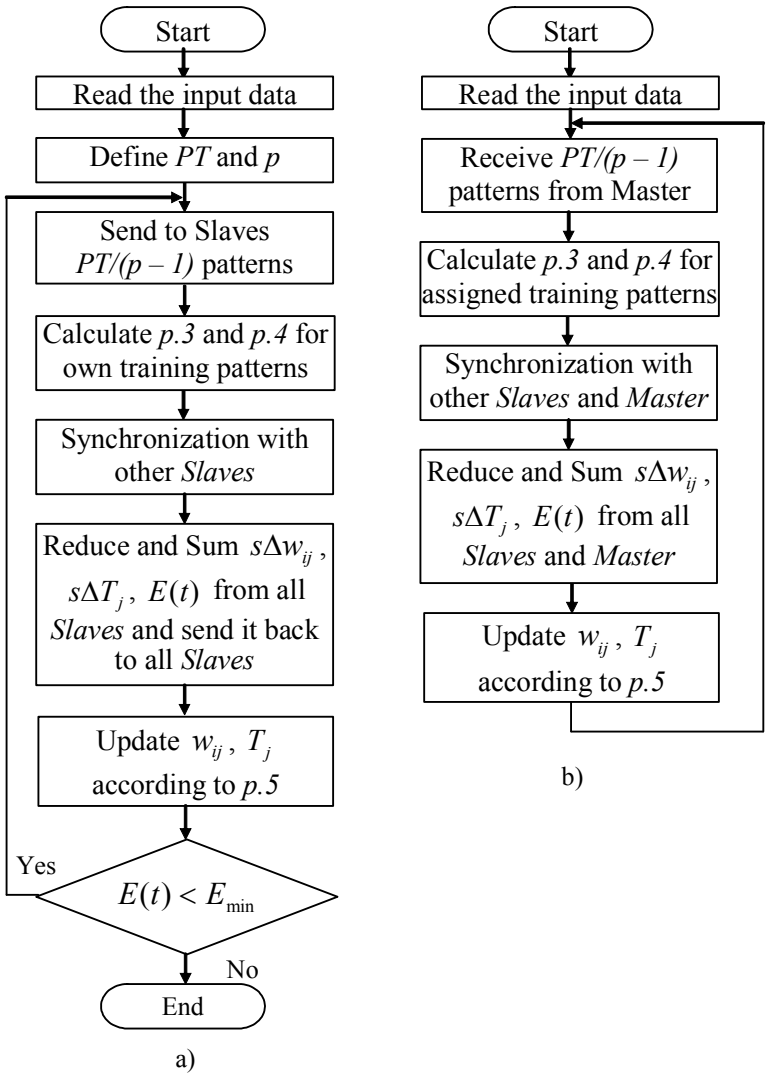

Figure 2 – The algorithms of *Master* (a) and *Slave* (b) processors

The global reducing operation with summation is executing just after synchronization point. Then the summarized values of $s\Delta w_{ij}$ and $s\Delta T_j$ are sending to all processors working in parallel. Using global reducing operation with simultaneous returning the reduced values back to the senders allows decreasing 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 on the next iteration of the training algorithm. Since the summarized value of $E(t)$ also is received in a result of reducing, the Master executes the operation from the point 7 of the algorithm, i.e. decides to continue the training or not.

The software routine is developed using C programming language using 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 $s\Delta T_j$ are provided by function *MPI_Allreduce()* which allow to avoid additional step of sending the updated weights and thresholds from the *Master* to each *Slave* back. Function *MPI_Finalize()* finishes the parallel part of the algorithm.

3. Experimental researches

The parallel computer NEC TX-7, placed in the Center of Excellence of High Performance Computing, University of Calabria, Italy (www.hpcc.unical.it), is used for experimental research of developed parallel algorithm. NEC TX-7 consists of 4 identical units. Each unit has 4 Gb RAM, 4 64-bit processors Intel Itanium2 with clock rate of 1 GHz. This 16^{th}-processor computer with 64 Gb of total RAM has a peak performance of 64 MFLOPS. Computer NEC TX-7 is functioning under Linux operation system.

It is expedient to form the research scenarios of increasing the dimension of parallelized problem in order to research parallelization efficiency according to these scenarios. The quality of perceptron training is described by achieved value of sum-squared error SSE, which should be provided in the result of training. Therefore the number of training epochs could be considered as an input parameter to form the research scenarios, which provide different SSEs. The task of prediction and predicting multilayer perceptron with 5 input, 10 hidden and 1 output neurons are used for research. The neurons of hidden and output layer have logistic activation function. It is used 794 training patterns in the training data set and 482 patterns in the prediction data set. The number of training epochs is changed from 10000 to 10^6 during the research. The learning rates of perceptron's hidden and output layers are constant and equal $\alpha(t) = 0{,}01$. The parameters of scenarios fulfillment are presented in the Table 1.

Table 1 – Parameters of research scenarios

Scenario	Number of iteration	Reached SSE	Time of sequential execution, seconds	Time of parallel execution on 1 processor, seconds	Relative error of prediction, %
Scenario 1	10000	2,9850	13,71	13,06	12,3
Scenario 2	100000	0,4391	137,08	130,79	4,7
Scenario 3	500000	0,2228	685,49	653	1,0
Scenario 4	1000000	0,1626	1371,00	1307,94	0,1

As it is seen from the Table 1, the perceptron provides good training ability, the SSE is changed from 2,98 till 0,16 and relative error of prediction is changed from 12,3 % till 0,1 %. The difference between the execution time of the sequential routine and the execution time of the parallel routine on 1 processor of the NEC TX-7 is within 5 %. The execution time has linear increasing which is caused by the certain execution time of one training epoch.

The execution time of parallel batch training algorithm on 2, 4 and 8 processors of the NEC TX-7 is presented in Table 2. The speedup $S = Ts/Tp$ and efficiency $E = S/p \times 100\ \%$

of parallelization is researched on 2, 4 and 8 processors, 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.

Table 2 – Execution time, speedup and efficiency of parallelization

Scenarios	Execution time (seconds) on processors		
	2	4	8
Scenario 1	6,85	3,90	2,50
Scenario 2	68,52	39,06	25,01
Scenario 3	342,23	195,31	129,04
Scenario 4	685,30	390,12	250,35
	Speedup on processors		
	2	4	8
Scenario 1	1,9066	3,3487	5,224
Scenario 2	1,9088	3,3484	5,2295
Scenario 3	1,9080	3,3434	5,0604
Scenario 4	1,9086	3,3527	5,2244
	Efficiency on processors, %		
	2	4	8
Scenario 1	95	84	65
Scenario 2	95	84	65
Scenario 3	95	84	63
Scenario 4	95	84	65

As it is seen from the results, the parallel batch back propagation training algorithm of multilayer perceptron provides very good scalability, i.e. provides the same level of parallelization efficiency at increasing the dimension of the parallelization problem. The efficiencies of parallelization of this algorithm are 95 %, 84 % and 63 % on 2, 4 and 8 processors of general-purpose parallel computer NEC TX-7 respectively for the multilayer perceptron 5-10-1 with 794 training patterns.

Conclusions

The parallel batch pattern back propagation training algorithm of multilayer perceptron is developed in this paper. The parallelization efficiency research for the scenarios of increasing the training epochs from 10000 to 10^6 showed very good scalability of parallel algorithm. It means that parallelization efficiency of this algorithm does not depend on the number of the training epochs. The parallelization efficiencies of parallel batch pattern back propagation training algorithm of multilayer perceptron are 95 %, 84 % and 63 % on 2, 4 and 8 processors of general-purpose computer NEC TX-7 respectively for the multilayer perceptron 5-10-1 with 794 training patterns. The provided level of parallelization efficiency is enough for using this parallel algorithm in Grid environment on general-purpose parallel and high performance computers. For future research it is expedient to estimate the parallelization efficiency of developed parallel algorithm on the scenarios of changing the architecture of multilayer perceptron (number of neurons) as well as the number of the training patterns in the input data set.

Acknowledgement

This research was supported by a Marie Curie International Incoming Fellowship grant 221524 “PaGaLiNNeT – Parallel Grid-aware Library for Neural Networks Training” within the 7th European Community Framework Programme. This support is gratefully acknowledged.

References

1. Haykin S. Neural Networks. – New Jersey : Prentice Hall, 1999.
2. Mahapatra S. A parallel formulation of back-propagation learning on distributed memory multiprocessors / Mahapatra S., Mahapatra R., Chatterji B. // Parallel Computing. – 1997. – Vol. 22, № 12. – P. 1661-1675.
3. Hanzálek Z. A parallel algorithm for gradient training of feed-forward neural networks / Hanzálek Z. // Parallel Computing. – 1998. – Vol. 24, № 5-6. – P. 823-839.
4. Murre J.M.J. Transputers and neural networks: An analysis of implementation constraints and performance / Murre J.M.J. // IEEE Transactions on Neural Networks. – 1993. – Vol. 4, № 2. – P. 284-292.
5. Dongarra J. Clusters and computational grids for scientific computing / Dongarra J., Shimasaki M., Tourancheau B. // Parallel Computing. – 2001. – Vol. 27, № 11. – P. 1401-1402.
6. Topping B.H.V. Parallel training of neural networks for finite element mesh decomposition / Topping B.H.V., Khan A.I., Bahreininejad A. // Computers and Structures. – 1997. – Vol. 63, № 4. – P. 693-707.
7. Rogers R.O. Using the BSP cost model to optimise parallel neural network training / Rogers R.O., Skillicorn D.B. // Future Generation Computer Systems. – 1998. – Vol. 14, № 5. – P. 409-424.
8. Parallel implementations of feed-forward neural network using MPI and C# on .NET platform / B. Ribeiro, R.F. Albrecht, A. Dobnikar [et al.] // Proceedings International Conference on Adaptive and Natural Computing Algorithms. – Coimbra (Portugal). – 2005. – P. 534-537.
9. Turchenko V. Computational Grid vs. Parallel Computer for Coarse-Grain Parallelization of Neural Networks Training / Turchenko V. // Lecture Notes in Computing Science LNCS 3762. – 2005. – P. 357-366.
10. Turchenko V. Fine-Grain Approach to Development of Parallel Training Algorithm of Multi-Layer Perceptron / Turchenko V. // Artificial Intelligence. – 2006. – № 1. – P. 94-102.
11. Parallel algorithms to solve two-stage stochastic linear programs with robustness constraints / P. Beraldi, L. Grandinetti, R. Musmanno, C. Triki // Parallel Computing. – 2000. – Vol. 26. – P. 1889-1908.
12. Golovko V. Neural Networks: training, models and applications / V. Golovko, A. Galushkin. – Moscow : Radiotechnika, 2001. – 256 p.

В. Турченко
Масштабированность параллельного группового алгоритма обучения нейронной сети
Разработка параллельного группового алгоритма обучения обратного распространения ошибки многослойного персептрона и исследование его масштабированности на параллельном компьютере общего назначения представлены в этой статье. Модель многослойного персептрона и групповой алгоритм его обучения описаны формализованным образом. Параллельный групповой алгоритм обучения представлен в алгоритмическом виде. Исследование масштабированности разработанного параллельного алгоритма осуществлено для пропорционально увеличивающегося размера задачи параллелизации на параллельном компьютере общего назначения NEC TX-7.

В. Турченко
Масштабованість паралельного групового алгоритму навчання нейронної мережі
Розробка паралельного групового алгоритму навчання зворотного поширення помилки багатошарового персептрону та дослідження його масштабованості на паралельному комп’ютері загального призначення розглянуті в цій статті. Модель багатошарового персептрону та груповий алгоритм його навчання описані формалізованим чином. Паралельний груповий алгоритм навчання представлено в алгоритмічному вигляді. Дослідження масштабованості розробленого паралельного алгоритму здійснено для пропорційно збільшуваного розміру задачі паралелізації на паралельному комп’ютері загального призначення NEC TX-7.

Статья поступила в редакцию 24.03.2009.