Competitive Strategies for Multilayer Perceptrons' Training using Backpropagation and Parallel Processing

R. L. S. ALVES, A. C. M. L. ALBUQUERQUE, J. D. MELO AND A. D. DÓRIA NETO
Departamento de Engenharia de Computação e Automação
Universidade Federal do RN - Campus Universitário s/n - 59072-970 - Natal - RN - BRAZIL

Abstract : - We present in this article a new approach for multilayer perceptrons' training. It is based on the utilization of parallelism and in the exploration of the inherent competition of this computation form. Multiple copies of the network are trained at the same time, and competitive strategies are used to speed up the convergence of the backpropagation algorithm. Each copy is initialized with a different matrix of synaptic gains, thus allowing a larger exploration of the parameters space and a larger possibility to avoid the local minimums on the error surface. The parallel tasks cooperate with each other to get benefit from the best results. The presented results are sharply superior compared to those obtained with the parallel algorithms published in the literature.

Key-words: - Backpropagation, Parallel, competitive strategies

1 Introduction

It is already well established in the literature the great potential application of the multilayer perceptrons - MLP, as a standard classifier, in the approximation of functions, as well as a universal interpolator. As the knowledge of this kind of network is stored in their synaptic gains, their applicability to complex problems demands topologies more and more dense, either in the number of hidden layers, or in the number of neurons per layer. That increase in the complexity results in processes with very slow training from the computational point of view, mainly when one uses the backpropagation algorithm [1-3]. A good analysis of the involved problems and of the different proposed solutions are found in [4].

Among those more appropriated tools, we can cite the use of parallel processing techniques. There are several used approaches, among them we can stress the exploration of the parallelism of decomposition of the training standards set [5-8]. In this work, this approach will be used as a comparison parameter and a new algorithm will be introduced. The sought goal is to seize not only the possibilities of the parallelism of the sequential algorithm, but also to discover other possibilities based on the exploration of the dispute/independence aspects inherent to the parallelism. Taking into account the complex problems of practical application, we analyze, in a quantitatively and qualitatively way, the performance of the proposed algorithm.

2 Backpropagation parallel algorithm based on the decomposition of the training set

Consider a MLP completely connected, constituted of L layers, with $k_l,\ l = 1,\dots,L$ neurons in each one of them. The connections between neurons of the layers k_l e k_{l-1} are pondered by synaptic gains $w_{ji},\ j = 1,\dots,k_l$ e $i = 1,\dots,k_{l-1}$. The training consists of the adjustment of the synaptic gains so as to minimize the overall mean square error defined by:

$$e_{med}(n) = \frac{1}{2N_P}\sum_{p=1}^{N_P}\sum_{j=1}^{k_L}\left[d_j^p - y_j^{(L)^p}(n)\right]^2 \qquad (1)$$

where $\underline{d}^p,\ p = 1,\dots,N_P$ represents the vector of wished outputs for the standard input $\underline{x}^p$ and $\underline{y}^{(L)^p}$

represents the output produced by the network for the same standard. Figure 1 illustrates the algorithm:

```
*Initialization:*
*Relative data to the network: number of layers, number of neurons per layer, set of training standards ( wished inputs and outputs);*
*Initialization of the synaptic gains matrices relative to all the layers*
**For** *each input* $p \in \{1, \ldots, N_P\}$ *, do*
*relative calculations of the forward phase*
    **Calculate**
    $\underline{y}^{(l)^p}(n) = \underline{f}^{(l)}\left(n, W^{(l)}(n), \underline{y}^{(l-1)^p}(n)\right), \quad \underline{y}^{(0)^p} = \underline{x}^p$
*relative calculations of the phase backward*
    *Calculate*
    $e^p(n) = \frac{1}{2}\sum_{j=1}^{k_L}\left[d_j^p - y_j^{(L)^p}(n)\right]^2 = \frac{1}{2}\sum_{j=1}^{k_L}\left[e_j^p(n)\right]^2$
    *If layer = L, calculate the gradient*
        $\underline{d}^{(L)}(n) = \underline{e}^p(n) \otimes \left[\underline{1}^{(L)} - \left(\underline{y}^{(L)^p}(n)\right)^2\right]$
    *else*
        $\underline{g}^{(l+1)}(n) = \left[\left(\underline{d}^{(l+1)}(n)\right)^T W^{(l+1)}(n)\right]^T$

*Atualizations of the* synaptic *gains*
**For** *each layer* $l \in \{1, \ldots, L\}$ *, do*
    $W^{(l)}(n) = W^{(l)}(n) - h\,\underline{d}^{(l)}(n)\left[\underline{y}^{(l)^p}(n)\right]^T$
***End***
```

Fig. 1 Backpropagation algorithm

Generally, the possibilities of exploration of the parallelism in a trained MLP neural network using the backpropagation algorithm can be divided in three main kinds: parallelism by decomposition of the training set, parallelism by decomposition of the computation between layers and parallelism by decomposition of the computation in each knot (neuron) of the network [7].

The parallelism by decomposition of the training set $\{x^p, d^p\},\ p = 1, \ldots, N_P$, consists of dividing the set into several subsets, so that each subset is used for the training of a network's copy. All the copies are initialized with the same synaptic gains. Therefore, each parallel task is constituted of a subset of the training standards and of a copy of the network to be trained.

Initially, it is accomplished in each task the processing regarding the forward and backward computation stages of the backpropagation algorithm, in such a way that in the end each one of them has the variations of the synaptic gains associated to its accumulated training subset. Each task sends then the matrix with the variations accumulated for a coordinating task and this makes the update of the set of synaptic gains. The new data are transmitted back for each one of the tasks and it is initiated a new iteration of the process of local update of the synaptic gains. This process is repeated until is reached a pre-determined value for the global mean square error.

From the point of view of parallel processing, the algorithm introduces a fork-and-join communication structure. During the expansion phase, a great number of tasks can be created, while during the contraction phase, only a task is executed. This forces the utilization of global communications of the kind one-for-all, and all-for-one, which are not recommended in an efficient algorithm.

3 Competitive strategies for the improvement of the performance

Although the algorithm introduced in the preceding section is effective in decreasing the time used in training the network, it just distributes among tasks the existing computation in the problem. It is actually an adaptation of the classical sequential algorithm to allow the exploration of the parallelism.

Analyzing that fact, a new approach was developed with the aim to use not only the possibilities of the parallel sequential algorithm, but also to discover other possibilities based on the exploration of the concurrency/independence aspect inherent to the parallelism.

The new algorithm is based on a parallel vision of the problem and, with this, some advantages were obtained. In the classical parallel algorithm, the multiple tasks have their synaptic gains initialized all in the same way, which corresponds to a single point of the parameter's space (synaptic gains) under the error surface. In practice, even though the tasks initialized the matrices of synaptic gain in different points (with no apparent advantage), after the presentation of the first epoch, they would have to leave from a single common point of the parameters space, because the matrices of synaptic gains, for all the tasks, are obligatorily the same.

The new algorithm tries to keep some characteristic of the classical algorithm, such as:

- Each parallel task executes the training of a copy of the network;
- The whole training set is used during the learning phase, although in one given iteration each task uses only a partition of it.

Beyond these, new characteristics are added to the algorithm, such as:

- Each task uses its own matrices of synaptic gains;
- The training set is introduced in its totality to each one of the copies of the network.
- After a pre-determined number of iterations, the tasks exchange information on the course of their learning processes, taking decisions about the continuity of them;
- The communication structure is simplified, avoiding communications of the kind all-to-one and one-to-all.

The utilization of multiple initializations of the matrices of synaptic gains allows that each task explores different regions from the search space [12-13], which permit the cover of the biggest possible area of the parameters space under the error surface. This, in practice, accelerates the convergence, because the initialization of the matrices of synaptic gains in different points, can generate in the tasks different trajectories in the search of the minima of the problem.

In the classical parallel algorithm, as the matrices of synaptic gains are identical for all the tasks in every training process, we have a much smaller probability to avoid minima locations. On the other hand, the competitive parallel algorithm works with different matrices of synaptic gains in the initialization and during the training for each task, enlarging the search possibilities of the global minimum of the system, and avoiding, with a larger probability of success, the minima locations.

For each copy of the network to be trained in an efficient form, one should use the complete set of the training standards. However, if for each epoch is introduced the complete set, the time of processing for each iteration is going to grow, and could make unfeasible the increase in the convergence speed. To keep the dimension of the training set identical to that of the classical algorithm, it was used the following procedure: for each iteration, one chooses a partition of the training set to be introduced to the network. In a new iteration, this partition is modified, in such a way that the whole set can be introduced to the network.

For each iteration of the backpropagation algorithm, the matrices are up-to-date locally following the gradient method. Moreover, after a number of pre-determined iterations, the tasks change information on the trajectories explored by each one of them and take decisions about the path to follow, procedure known in the literature as diffusion [12]. Inside the fundamental characteristic of the algorithm, which is the exploration of different trajectories by each task, after taking the decision, each one of them should continue the learning with their own matrix of synaptic gains.

Two aspects should be considered in the new algorithm: (i) the definition of the number of iterations after the tasks change information and (ii) the decision rules for the alteration of the matrices of synaptic gains. To solve the first aspect, two approaches are possible:

- Accompany the evolution of the error to change information whenever the training process of the network in one of the tasks starts to diverge. As the convergence is not always monotonic, the error can oscillate during a certain period and start to decrease from a given iteration. This fact imposes difficulties to define exactly when the training diverges. One alternative is to consider the error's average on a given horizon (number of iterations), as a form of eliminating the oscillations. Other difficulty is related to the parallel execution of the algorithm. Since the task detects that its training process is diverging, it should notify to all the others the need of modify their matrices of synaptic gains. That imposes the need of synchronization points in the algorithm, jeopardizing its performance;

- A second approach is to determine the number of iterations. The problem in this case is to avoid that the information exchange is too early or too late.

With regards of the definition of the decision rules, it should consider that after a certain number of iterations, each task owns matrices of distinct gains, modified by the gradient method. Based on this set, it was adopted the use of different competitive heuristics, as listed below:

- Heuristic 1: In a new iteration one of the tasks uses the best set W of synaptic gains_(the one that presents the least error for the samples of the training set). In this case, it preserves the best result obtained in the exploration of the trajectories by the tasks;
- Heuristic 2: In a new iteration, one of the tasks uses the average of all the sets of synaptic gains of the tasks. As in the training by epoch it uses the average of the local gradients to update the gains, the general average of all the sets reinforces that procedure;
- Heuristic 3: In a new iteration some tasks use the best set W of synaptic gains with some kind of modification. These modifications can be of two kinds: increase a portion of the set of local gains or utilize a perturbation in the elements of the better set (introduction of a virus). The goal is to dislocate the parameters set of the minimum point, as a form of avoiding the convergence for local minima.

An important aspect to be observed is that the decision process can be locally done for each task. However, to be possible, all of them should have knowledge of all the set of synaptic gains obtained by each of them individually. That defines the communication structure of the algorithm.

To identify which is the best set of gains, it becomes necessary to calculate the associated errors to each one of these set. Adopting a partition strategy of the training set so that, for each iteration:

$$\bigcup_{i=1}^{N_T} X_i = X \quad e \quad \bigcap_{i=1}^{N_T} X_i = \varnothing \tag{2}$$

where N_T is the number of parallel tasks, X_i is the partition used by the task T_i and X is the training set, we can calculate the error by the expression:

$$e_{med}(W_{T_i}) = \frac{1}{N_P}\sum_{i=1}^{N_T} e^{(X_i)}(n, W_{T_i}) = \frac{1}{2N_P}\sum_{i=1}^{N_T}\sum_{j=1}^{|X_i|}[e_j^{(X_i)}(n, W_{T_i})]^2 \tag{3}$$

where W_{T_i} represents the set of synaptic gains associated to the task T_i, $i = 1,\ldots,N_T$ and $|X_i|$ the number of partition elements X_i, i.e., $|X_i| = {|X|}/{N_T}$ supposing $|X|$ multiple of N_T. The least minor error associated to the different sets is given by:

$$e_{med_{\min}} = \min e_{med}(W_{T_i}),\ i = 1,\ldots,N_T \tag{4}$$

As each task accomplishes its training on a distinct partition X_i and, at the end of the fixed number of iterations has a W_{T_i}, also distinct, it becomes necessary to circulate among the tasks all the pairs (X_i, W_{T_i}). To do that, we can use communications of the kind all-to-all, but these are onerous enough from the point of view of time of execution of the algorithm. We opted therefore by the iterative form of communication over a ring.

In the first iteration of the communication process, each task calculates and stores the error regarding its data partition and its set of synaptic gains, transmitting afterwards the set of gains and the value of the error for the neighbor task, in the route path of the ring. In the second iteration, each task calculates the error regarding the set of received training, stores that value and transmits the gain set and all the stored and received errors in the previous iteration. The other iterations on the ring procede in the same way until all the sets of synaptic gains have been sent/received. Observe that, in this process, some errors are calculated in each task and the remaining are transmitted by the other tasks. The algorithm in pseudo code can be seen in Figure 2.

```
While the stop criterion is not reached, do:{
  local initialization of the data;
  shuffled of the training set;
  While n times are not processed, do:{
    Calculation of the standards band for each task;
    If rank==0 {    // coordinating task
      Do the calculation of the number of
      standards of the subset;
    If the number of standards is odd add 1 to the number of
    standards to be trained by rank 0;
} else {  // other tasks
    Do the calculation of the number of standards of the subset;
  }
  For each standard of the subset, do:{
    Calculate the phase forward{
      Calculate the local error;
    }
    Calculate the phase backward{
      Calculate the variations of the synaptic gains;
    }
  }
  Update local synaptic gains by using the Delta-bar-delta rule;
  }
  For each passage of the ring do:{
    Send the matrix of valid gains for the successor;
    Calculate the error for the subsets of local standards and the
    matrix of received gains(); // in the first time the  received
    equal to the current
    Keep the error and the passage of the ring;
    If it is not the first passage of the ring do:{
        Calculate the error for the subsets of the standards
        received  and the local gain matrix;
        Keep the errors and the passage of the ring;
        Send all the calculated and received errors  in the
        previous step of the ring for the successor;
        }
        Receive the  matrix of synaptic gains from the
        predecessor
        If it is not the first passage of the ring do:{
            Receive all the correspondents errors sent by the
            predecessor;
            Keep the received errors;
        }
      }
      It calculates the new initial matrix of synaptic gains  for the
      next n times;
      increase the iteration;
    }
```

Fig. 2. Competitive parallel algorithm

4 Application of the algorithm in the solution of complex problems

The image compression becomes more and more important to solve capacity limitations problems of the communication's channels. It is a complex problem, for which many different techniques have been developed. MLP's utilization appears as an approach with promising possibilities to contribute for the solution of the problem [9].

For implementation of the parallels algorithms we used an Beowulf architecture [10] with double bars and 14 processors, available in the Automation and Computer Engineering Laboratory at UFRN.

To obtain the image's compression using MLP we proceed in the following way: reduce the number of neurons of one of the hidden layers of the network. The training consists of letting the network learn to repeat in its output the standards that are introduced in the input. The compression/decompression procedure of the image is done by the partition of the network in two, one being defined up to the strangled layer and the other from this stage. The compressed image corresponds then to the output of the neurons of the strangled layer.

For the training it was used an image of Lena in black and white, composed of 512x512 pixels or 262.144 points. The segmentation of the data was made starting from windows of 8x8 pixels. Moreover, the values of the data were in the range between 0 and 1 and after that normalized and scaled.

It was used three topologies for the neural network: 64:8:8:8:64, 64:16:16:16:64 and 64:32:32:32:64. The set of complete training was made up of a thousand points (window of 8x8 pixels) of the image.

The results obtained with the classical parallel algorithm for the problem of the training, when used the compression rate of 64:32, are shown in Table 1.

Table 1. Results of execution of the classical parallel algorithm

Number of Proc.	Compression 64:32		
	Exec. Time	Gain	Efficiency
1	32601.81	1.00	1.00
4	8687.311	3.75	0.94
8	4741.956	6.86	0.86
14	3644.731	8.94	0.64

For the other topologies of the network the results were very similar and will be omitted here.

Before the comparison with the competitive algorithm, we tried initially to verify the influence of the initialization in the classical parallel algorithm. For ten different initializations, we verified a variation in the time of execution of the order of 52%. To make comparison, it was used the least training time of all the initializations.

For comparison effect of the performance of the two algorithms, we defined the gain in time through the expression:

$$\text{speed-up} = \frac{\text{usual\_parallel\_execution\_time}}{\text{competitive\_parallel\_execution\_time}}$$

The results obtained in terms of gain and time of execution are shown in the Table 2 below:

Table 2. Results of Gain And Time of Execution

	Competitive Compression		
	64:8	64:16	64:32
Gain	5.17	2.28	7.25
Time	42.027	113.628	502.791

The communication rate used in the competitive parallel algorithm for each training was 50 epochs for each communication. Taking into account that it was not used an average for the several initializations of the classical algorithm but, instead, the best of them, one can prove that the new algorithm reduces efficiently the time of training. Other important aspect to be analyzed is the quality of the obtained solution. Tests were accomplished to verify the generalization capacity of the network, after the training [11].

5 Conclusions

In this work it was introduced competitive techniques for the training of multilayer perceptrons using backpropagation. Allied to the parallel processing, such techniques showed to be very effective in decreasing the time of training of networks with complex topologies. Once the parameters space is explored in different points, one can avoid the problem of the local minima, besides obtaining a sharply superior quality in the results of generalizations generated by the network. Therefore, it is allowed to foresee an enlargement in the utilization of such a kind of neural network.

References:

[1] P. J. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, 1974.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representation of back-propagation errors*, Nature (London), vol.323, 533-536, 1986.

[3] D. R. Hush, B. Horne and J. M. Salas. Error surfaces for Multilayer Perceptrons. IEEE Tr. on Sys., Man and Cybernetics, vol. 22, 1152-1161, 1992.

[4] Y. LeCun, L. Bottou, G. Orr, and K. Muller. *Efficient BackProp, Neural Networks: Tricks of the trade*, (G. Orr and Muller K., eds.), 1998.

[5] V. Kumar, S. Shekhar e M. B. Amin. *A Scalable Formulation of the Backpropagation Algorithm for Hypercubes and Related Architectures*, IEEE Tr. on Par. and Dist. Systems, vol. 5, 1073-1090, 1994.

[6] S. K. Foo, P. Saratchandran e N. Sundararajan. *Parallel Implementation of Backpropagation Neural Networks on a Heterogeneous Array of Transputers*, IEEE Tr. on Sys. Man and Cybernetics, Part B, vol. 27, 118-126, 1997.

[7] J. Torresen. *Parallelization of Backpropagation Training for Feed-Forward Neural Netoworks*, PhD thesis, The University of Trondhein, 1996.

[8] A. Novokhodko e S. Valentine. *Parallel Implementation of the Bach Backpropagation Training of Neural Networks*, Proc. of The Int. Joint Conf. on Neur. Net., IJCNN, July, 2001.

[9] R. C. Gonzalez e R. E. Woods. *Processamento de Imagens Digitais*, vol. 1, page 227, Edgard Blücher, São Paulo, SP , 2000.

[10] T. Sterling e D. Becker. *The Beowulf Project.* www.beowulf.org.

[11]R. L. S. Alves, J. D. Melo, A. D. Dória Neto e A. C. M. L. Albuquerque. New Parallel Algorithms for Back-Propagation Learning, IJCNN2002, Honolulu, Havaí, USA, 2002.

[12] M. Toulouse, T. Crainic and K. Thulasiraman, Global Optimization Properties of Parallel Cooperative Search Algorithms: A Simulation Study, Parallel Computing, vol. 26, 91-112, 2000.

[13] M. Toulouse, T. Crainic and B. Sansò, Systemic Behavior of Cooperative Search Algorithms. Parallel Computing, vol.30, no 1, pp 57-79, 2004.