The OWE Artificial Neural Network and its Parallel Implementation

Nicolas Pican* **and Yannick Lallement****

* *Computing and Electrical Engineering Department*
Ocean Systems Laboratory, Heriot-Watt University
Riccarton, Edinburgh EH14 4AS, UK
Email: ceenvp@cee.hw.ac.uk
** *Human Computer Interaction Institute, Carnegie Mellon University*
5000, Forbes Avenue, Pittsburgh, PA 15213-3891, USA
Email: yannick@cmu.edu

RÉSUMÉ. *Les perceptrons multi-couches (ou perceptrons) sont largement utilisés pour l'approximation de fonctions, mais sont très demandeurs en temps de calcul. Malheureusement, l'implantation parallèle des perceptrons est un problème complexe ; dans cet article, nous proposons une nouvelle méthode efficace pour leur parallélisation. Nous présentons l'architecture OWE (Orthogonal Weight Estimator), dont les performances sont au moins aussi bonnes que celles d'un perceptron classique avec le même nombre de paramètres libres. L'architecture OWE consiste en un perceptron principal dans lequel chacun des poids est calculé par un perceptron indépendant (un OWE). Le nombre d'OWEs est égal au nombre de poids dans le perceptron principal. Les calculs dans chaque OWE peuvent être menés de façon indépendante, ce qui permet une parallélisation aisée des phases d'apprentissage et de relaxation. Nous décrivons notre implantation de cette architecture sur une machine Intel Paragon, et nous la comparons avec son implantation sur une machine séquentielle.*

ABSTRACT. *Multi-layer perceptrons (MLPs) are a useful and widely used tool for function approximation, but they are very demanding on computer time, hence the need for parallelization. Unfortunately, the parallel implementation of MLPs is not straightforward. In this paper, we propose a new efficient way to parallelize MLPs. We present the OWE (Orthogonal Weight Estimator) architecture, which performs at least as well as a classical MLP with the same number of free parameters. The OWE architecture consists of a main MLP in which the value of each weight is computed by another MLP (an OWE). The number of OWEs is equal to the number of weights of the main MLP. The computation in each OWE can be done independently, therefore the training and relaxation phases can easily be parallelized. We report our implementation of this architecture on an Intel Paragon parallel computer and the comparison with its implementation on a sequential computer.*

MOTS-CLÉS : *Réseaux de neurones, perceptrons, implantation parallèle, performance.*

KEYWORDS: *Neural nets, perceptrons, parallel implementation, performance.*

1. Introduction

Neural networks are very useful tools for tasks such as classification and function approximation. They are of interest both to artificial intelligence and industry: more and more applications of both domains are dealt with using connectionist networks. Their most interesting feature is that they can learn to approximate or to classify *from examples*. This capability enables neural networks to be used when there is little or no formal knowledge about the application. Among neural networks, the more commonly used ones are the multi-layer perceptrons (MLPs). Their usefulness has been demonstrated many times on several kinds of applications [CHA 95, HER 92].

Neural networks, and especially MLPs, suffer a major drawback: they are very demanding of computing power, hence the need for parallel implementations. Many such implementations have been proposed so far, but mostly on SIMD computers (for example, Hypercube [KER 92, MAL 95], Connection Machine [SIN 90], MasPar [KUS 90]), or on some specially built, dedicated architectures [KOT 93, MÜL 94]. The current evolution of the supercomputing market shows a tendency towards general purpose scalable MIMD computers, like the Intel Paragon or the Cray T3D (recently T3E). Unfortunately, as we will see, the classical MLP architecture is not very well suited for these types of computers. In this paper, we present a new architecture derived from the MLP one, and show how easily it can be parallelized on these new supercomputers, allowing neural networks to run at a much faster pace.

In the following sections, we first give the basics of MLPs, then we explain why the computing power resource is critical for their development and their usage. We review the most common parallelization methods that have been proposed so far and show their limitations. We propose a new connectionist architecture, called OWE, that performs at least as well as a classical MLP with the same number of free parameters, and that can be implemented on a parallel computer in a straightforward manner. Finally, we present the implementation we have done of this new architecture on an Intel Paragon computer and the results we obtained on a function approximation test.

2. Basics of Multi-Layer Perceptrons

Multi-layer perceptrons (see [HER 92] for a complete introduction) are constituted of several consecutive layers of neurons (see Figure 1). A neuron is a unit that computes a weighted sum of its inputs: The output Y_i of the neuron i is given by $Y_i = F_i(\sum_j w_{i,j} Y_j)$, where Y_j is the output of the neuron j, $w_{i,j}$ is the weight between the neurons i and j (see Figure 1), and F_i is the transfer function (usually a sigmoid function). The neurons of a given layer are connected to the neurons of the next one (feed-forward topology), or to the neurons of all the next ones (full feed-forward topology). A MLP works in two phases: the *training phase*, in which the weights are tuned to approximate a target function, and the *relaxation phase*, in which the network simply computes its output for a given input. In the relaxation phase, the output of the network (that is, of the output neurons) is obtained by successively computing the outputs of each layer.

The training phase is done in two steps: the first step is a relaxation phase, after which the output of the network is compared to the target output provided by a teacher.

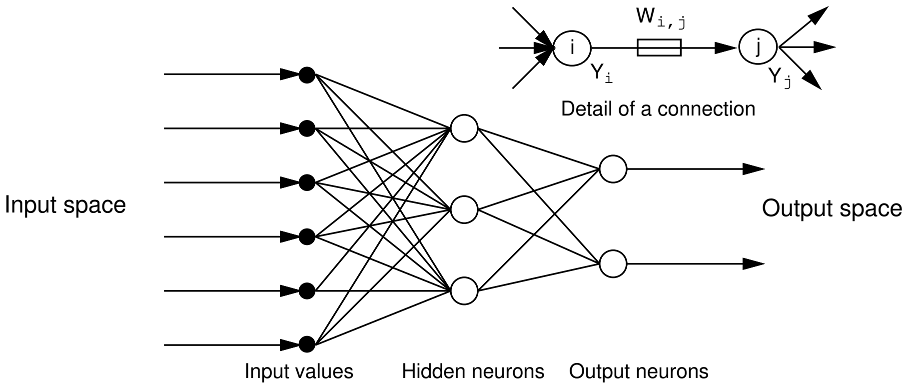

Figure 1: *An example of multi-layer perceptron*

During the second step (back-propagation), the error between the effective and target outputs is back-propagated in the network and the weights are updated in order to make the output closer to the target. The weights can be updated after the presentation of each vector (incremental learning) or only after the presentation of a set of vectors (batch learning). Several parameters control the training phase, and the performance of the network (i.e., the accuracy of the function approximation) will depend on the fine tuning of these parameters.

The network is trained using a *training set* containing input vectors and their corresponding target output. Several presentations of the whole training set (several *training epochs*) are usually required to obtain a good performance from the network. The performance is measured on a *test set*, whose vectors are different from the vectors of the training set.

3. Parallel Implementations of Multi-Layer Perceptrons

It is common for a training phase of an average MLP to last several days on a state of the art workstation. The training of bigger networks can last for weeks. This strongly hinders the use of MLPs since a large number of parameters have to be tuned to get the best possible performance, and the optimal tuning is usually obtained after many tests on different sets of parameters.

Reducing the experimentation time is thus necessary to make the development of MLPs easier, and to allow the development of bigger networks. Parallel computation is an obvious way to achieve this goal. For the reasons mentioned in the introduction, we will consider here only the parallel implementations that can be done on message passing MIMD computers, which are the most widely available parallel architectures. We are currently working on a language intended to make the development of a variety of neural networks on such computers easier [VIA 94, LAL 97], but here we focus on MLPs only.

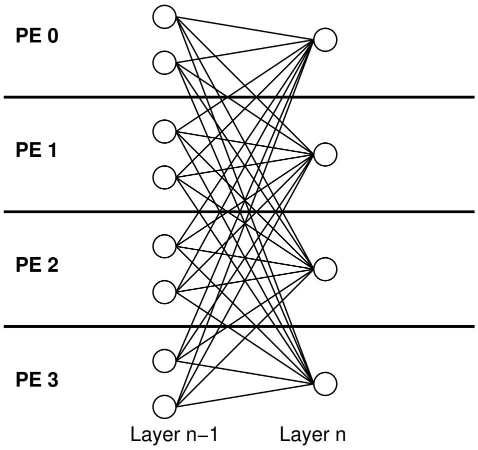

Figure 2: *Mapping of two successive layers of a MLP on 4 PEs*

There are several strategies for parallelizing the computation of MLPs [MÜL 94, NOR 92]. The most straightforward strategy consists of running several MLPs independently on several processing elements (PEs), each MLP having its own set of training parameters; the goal here is to test simultaneously several MLP configurations in order to see which one is the best. This is probably the most commonly used parallelization: there is no need to write explicitly parallel code since there is no communication between the MLPs, and even a network of workstations is enough to exploit this technique. However, this method has two main drawbacks. First, the user still has to wait for a long time before getting any results (even if many results from all the different networks are finally produced); second, the results of one experiment are often needed to start the next one (e.g., considering the current experiment, how should the parameters be modified to obtain better results?).

Another solution is to run several MLPs on several PEs, each MLP performing on a subset of the complete training set. From time to time, the weight updates are exchanged among the MLPs. This kind of parallelization requires explicit parallel programming. It is likely to be efficient on a parallel computer since the amount of data exchanged between the PEs is very small. It is, however, restricted to batch learning which is known to converge significantly slower than continuous learning [HER 92]. A part, if not all, of the speed-up brought by the parallel implementation can then be lost.

The third strategy, which avoids the problems of the two previous ones, is to run only one MLP, spread over a set of PEs. Distributing the computation is not an easy task, since the outputs of the successive layers have to be computed sequentially. Therefore, in order to allow the PEs to perform in parallel, it is necessary to split the layers amongst them. Figure 2 shows how two successive layers can be spread on 4 PEs. To compute the outputs of its part of a given layer, each PE needs to dispose of the complete output of the previous layer. This generates much communication in the

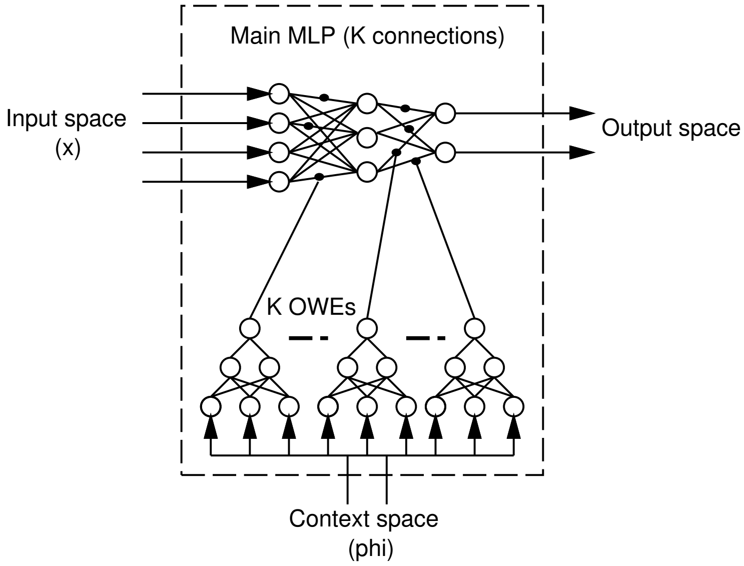

Figure 3: *The OWE architecture*

network of PEs, which will reduce the speed-up obtained by the parallel implementation, especially if there is a large numbers of PEs.

Of these three strategies, none is fully satisfying. This is due to the very structure of MLPs, which is very regular, but strongly interconnected. In the next section, we briefly present a new architecture, called OWE (described in further details in [PIC 94a, PIC 96b]), that performs at least as well as a MLP with the same number of connections — that is, a same amount of computation — and that can be parallelized in a very straightforward way.

4. The OWE Architecture

4.1. *Principle*

Our studies [PIC 94a, PIC 96b] have shown that if we remove some inputs, called *external* or *contextual* inputs $\vec{\varphi}$, of a MLP and train it to model a continuous function $f(\vec{x}, \vec{\varphi})$, the output of the MLP is a function $g(W, \vec{x})$ where each value w_{ij} of the connection matrix W is the expectation of a continuous function $\Omega_{ij}(\vec{\varphi})$ over the external input space Φ:

$$w_{ij} = E_{\Phi}[\Omega_{ij}(\vec{\varphi})] \qquad [1]$$

More precisely, we have shown that the weight w_{ij} of a connection in the MLP is a continuous function depending only on the value of the removed external vector $\vec{\varphi}$. This assertion implies that each weight can be independently estimated by the computation of $\Omega_{ij}(\vec{\varphi})$. In addition, since $\Omega_{ij}(\vec{\varphi})$ are continuous functions, each one can also be modeled by a MLP.

We now consider our neural network architecture, called OWE architecture, that includes a main MLP fed by a vector $\vec{x}$ in which each connection value is computed by another MLP (an OWE) fed by a vector $\vec{\varphi}$. There are thus as many OWEs as the number of weights in the main MLP. All the OWEs are independent from one another. The architecture is presented in Figure 3.

Each OWE computes: The main MLP computes:

$$\begin{array}{rcl} \Omega_{ij} : \Phi & \rightarrow & \Re \\ \vec{\varphi} & \rightarrow & \Omega_{ij}(\vec{\varphi}) \end{array} \qquad\qquad \begin{array}{rcl} g : X \times \Phi & \rightarrow & \Re \\ \vec{x}, \vec{\varphi} & \rightarrow & g(\Omega(\vec{\varphi}), \vec{x}) \end{array}$$

4.2. *Relaxation and Training Phases*

According to the OWE architecture, the relaxation phase, for each presented pattern $(\vec{x}, \vec{\varphi})$, is done in two steps. The first one is the propagation of the external parameters $\vec{\varphi}$ through each OWE to compute each connection value of the main MLP given by the realization of the $\Omega_{ij}(\vec{\varphi})$ functions. The propagations of $\vec{\varphi}$ through each OWE are independent and can be done simultaneously. After the end of this first step all the connection values of the main MLP are known. Then the second step is the classical propagation of the input $\vec{x}$ through the main MLP.

The training phase begins with a relaxation phase and is followed by a classical backpropagation of the error gradient through the main MLP. When the gradients are known for all the main MLP connections we use each gradient value as the realization of the output error that is backpropagated through the corresponding OWE. These OWE trainings, called *on-line learning algorithm* [PIC 94b] are also independent and can be done simultaneously.

4.3. *Comparison with Classical MLP*

In this section, we compare the performance of the OWE architecture to the performance of a classical MLP with the same number of free parameters (weights) on the task of approximating a test function. The comparison is given in terms of number of epochs needed for training and of quality of the approximation. The chosen test function in the boundary space $[-5, 5] \times [-5, 5]$ (Figure 4) is defined by:

$$f(x, y) = xye^{\cos^2 x + \sin^2 y} + \frac{(x+y)^2}{10} - \log(|x+y|+1) + \sqrt{x^2+y^2} \qquad [2]$$

This function has many non-linearities and can be viewed in a 3-dimensional space, thus constitutes a good test example. The training set is composed of $101 \times 101 = 10,201$ patterns $(x, y, f(x, y))$ taken from x and $y = \{-5, -4.9, \ldots, 4.9, 5\}$. The test set is composed of $21 \times 21 = 441$ patterns $(x, y, f(x, y))$ taken from x and $y = \{-5, -4.5, \ldots, 4.5, 5\}$. y has been arbitrarily chosen as *external* inputs ($\vec{\varphi} = y$).

The internal structure of OWE architecture is a 1x12x1 LFF+B (local feed-forward MLP with bias neuron [HER 92]) for the main MLP and a 1x6x1 LFF+B (19 connections) for each OWE. There are 37 connections in the main MLP so there are 37 OWEs. Thus, this OWE architecture includes 703 (37x19) free parameters. y feeds

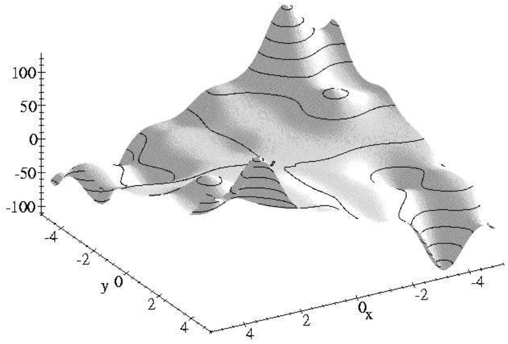

Figure 4: *The test function $f(x, y)$*

the OWEs and x feeds the main MLP. The internal structure of the classical MLP used for comparison, equivalent in terms of number of free parameters (weights), is a 2x30x20x1 LFF+B (731 connections), fed with x and y.

After 5000 training epochs the mean error left for the classical MLP is 1.13 against 0.83 for the OWE architecture, showing that the OWE performs slightly better than the MLP. Moreover, the speed of convergence during the first training epochs is higher for the OWE architecture than for the MLP. The OWE architecture reaches a mean error of 1.13 after only 2400 training epochs when the MLP reaches a mean error of 5.4. Other experiments have confirmed these favorable results of the OWE architectures compared to the MLPs [PIC 96a].

5. Parallel Implementation of the OWE Architecture

Taking advantage of the independence of the OWEs, a very straightforward parallel implementation of the OWE architecture distributes the computation of the $\Omega_{ij}(\vec{\varphi})$ functions by distributing the OWEs on several PEs.

Let P be the number of PEs used in a parallel computer, N the number of connections in the main MLP (the number of OWEs). Let T_{Main} and T_{OWE} be the execution times of the main MLP and of an OWE for one relaxation phase. Let T'_{Main} and T'_{OWE} be the execution times of the main MLP and of an OWE for one training phase. We assume that $N > P$ so that each PE runs at least one OWE.

We use one PE as a *master* PE for the computation of the main MLP and the management of the OWE message transfers. The other PEs (called *slaves*) are used to manage the OWE computations. Let T_R be the execution time of one relaxation (R) step, T_T be the execution time of one backpropagation (T) step. The global execution time of the OWE architecture for one relaxation phase (T) and one training phase (T') can be approximated by:

$$\begin{aligned} T = T_R &= T_{Main} + \beta T_{OWE} + \mu \\ T' = T_R + T_T &= T_R + T'_{Main} + \beta T'_{OWE} + \mu' \end{aligned}$$

$\beta = \frac{N}{P-1}$ is the number of OWE per PE, μ and μ' are the time consumed for message transfers. For a quick succession of training and relaxation phases, the test and training sets must be loaded in memory rather than read from a disk when necessary. In order to minimize μ, the X sets are loaded into the master PE memory and the Φ sets are loaded into each slave PE memory. During the training phase, the two sets must accessed randomly; to make this possible, at the beginning of a training epoch the master PE sends only the same random integer value to each slave. This integer is used by each PE to initalize its random generator, to make sure that all the OWEs will always work on the same pattern as the main MLP.

One can notice that if only one PE of the available parallel computer has a large memory, it is also possible to load the $X \times \Phi$ sets into this node, which will send the $\vec{\varphi}$ vectors to each OWE at each relaxation. The communication times will, in this case, be higher.

μ and μ' can be approximated by:

$$\begin{aligned} \mu &= S_{s \to m}(P-1, \beta); \\ \mu' &= S_{m \to s}(P-1, \beta) \end{aligned}$$

where $S_{m \to s}(n,k)$ (resp. $S_{s \to m}(n,k)$) is a continuous increasing function depending on the target hardware architecture, expressing that the time taken by the transmission of one message of size k bytes from the master to n slave nodes (resp. from n slave to the master node) is an increasing function of n and k. The importance of the term μ decreases with the size of the OWE neural networks. As the size of the OWEs increases, the communication time decreases in proportion to total execution time. This is illustrated in the next section.

6. Parallel Performances

In this section, we measure the OWE architecture performances on the test function and the training and test sets presented above, in terms of execution time on different numbers of nodes.

6.1. *The Experiments*

We used nine different OWE architecture sizes, with different numbers of free parameters (N_W) as listed in Table 1. The parallel implementation and the tests have been realized on the Intel Paragon computer at IRISA (Rennes, France). This computer has 59 PEs with: two PEs with 32 Mb RAM for I/O, one boot PE with 32 Mb RAM, and 56 usable PEs with 16 Mb RAM. The training and test sets occupy less than the 16 Mb RAM of the PEs, eliminating swapping.

In the next paragraph, we study how our OWE parallel implementation running on a variable number of PEs compares to our sequential implementation of the same OWE running on a single PE of the Paragon. The tests are done for each OWE architecture presented in Table 1. For each test, we study the training and the relaxation phases separately, since they form two distinct parts in the functioning of a neural network.

#	Main MLP	OWEs	N	N_W
#1	1x12x1	1x6x1		703
#2	1x12x1	1x6x3x1	37	1,369
#3	1x12x1	1x12x6x1		4,033
#4	1x12x6x1	1x6x1		2,071
#5	1x12x6x1	1x6x3x1	109	4,033
#6	1x12x6x1	1x12x6x1		11,881
#7	1x24x12x1	1x6x1		6,859
#8	1x24x12x1	1x6x3x1	361	13,357
#9	1x24x12x1	1x12x6x1		39,349

Table 1: *The nine OWE architectures: size of the main MLP, size of each OWE, number of OWEs and number of free parameters*

6.2. *Results*

6.2.1. *Execution times, speed-up and efficiency*

The execution times of the training and relaxation phases for various numbers of nodes $P-1 = \{1, ..., 50\}$ are shown in Figure 5. The relaxation and training times (T and T' defined above) are given for one vector. They are obtained by dividing the total training time for one epoch (resp. relaxation time) by the size of the training set (resp. test set). In each phase, there is an optimum number of PEs for which the execution time is minimal. This optimum number grows with the size of the architecture, as the importance of computation grows with regard to the communication overhead, which remains constant. This behavior is typical, and can be observed in many parallel applications in the presence of communication overhead [FLA 89, ZOR 89]. In our case, this behavior is not a problem since we intend to run large network architectures.

We measured the speed-ups of the nine OWE architectures. The speed-up is the execution time on one PE divided by the execution time on n PEs. The results for the training and relaxation phases are given in Figure 6. Here again, the maximum speed-up grows with the size of the architecture, which is also a suitable behavior for our purpose.

We also measured the efficiency (speed-up divided by the number of processors) of the nine OWE architectures: Figure 7. The efficiency decreases when the number of PEs increases, which is a typical behavior. The efficiency is higher and decreases more slowly for large architectures, which is suitable; for the largest one (#9), the efficiency remains above 50% for up to 50 PEs, showing that a reasonably efficient use is made of the parallel computer.

6.2.2. *Optimal performances*

Figure 8 shows the best speed-ups for each architecture, and (at the bottom of each box) the corresponding number of PEs. The mean efficiencies obtained are 42% for

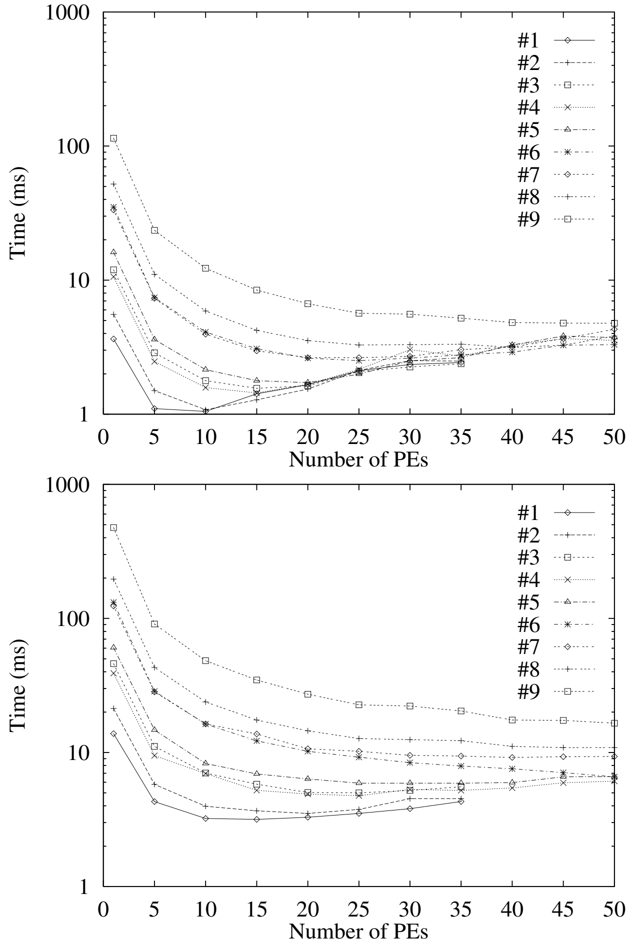

Figure 5: *T (relaxation time) vs number of nodes for one vector (top) and T' (training time) vs number of nodes for one vector (bottom)*

the relaxation phase (min 32%, max 52%), and 52% for the training phase (min 43%, max 58%). If efficiency is an important criterion, significantly higher ones can be obtained at the cost of slightly lower speed-ups. These two diagrams, summarizing the measurements presented above, show that the best results in terms of speed-up and efficiency are obtained for large networks, which are precisely the ones we target. Therefore, we can conclude that our proposed architecture is well adapted to implement large neural networks on large parallel computers.

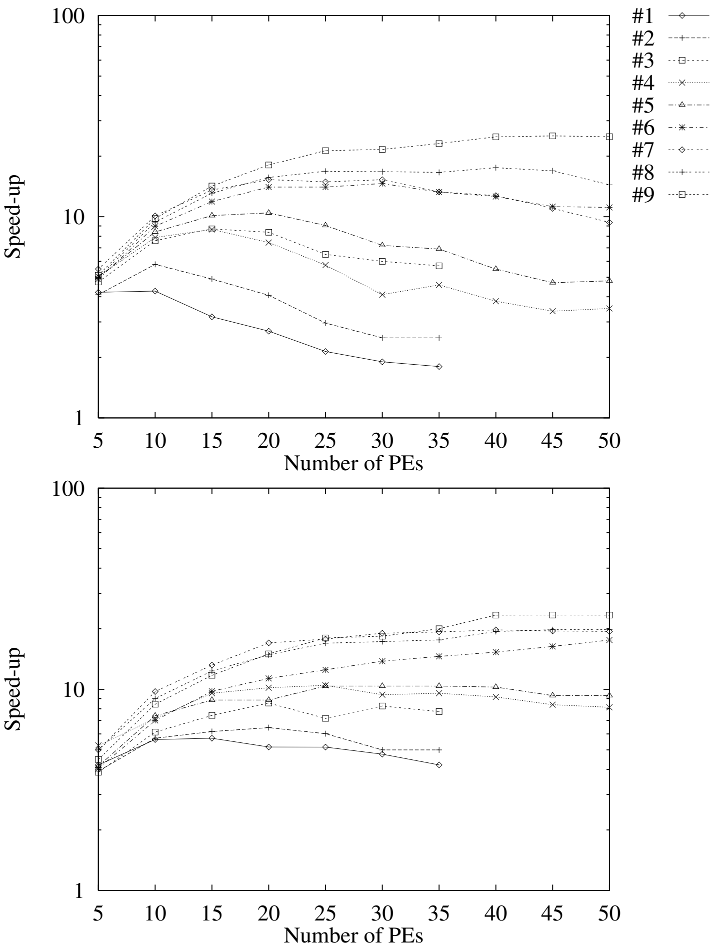

Figure 6: *Speed-up vs number of nodes for the relaxation phase (top) and the training phase (bottom)*

7. Discussion

We have presented the results of the parallel implementation of the OWE architecture, where each connection weight of a main MLP is computed independently by another MLP (an OWE). The OWE architecture is interesting from three points of view. Firstly, its intrinsic performance is at least as good as that of a classical MLP with an equal number of weights. Secondly, its parallel implementation is very straightforward, which is valuable in case of large neural networks. The obtained performances and the scalability are correct; the application can make efficient use of several tens of PEs (Section 6). Therefore, our primary goal of speeding up the execution of large neural nets is reached. Thirdly, the speed of convergence is higher for an OWE ar-

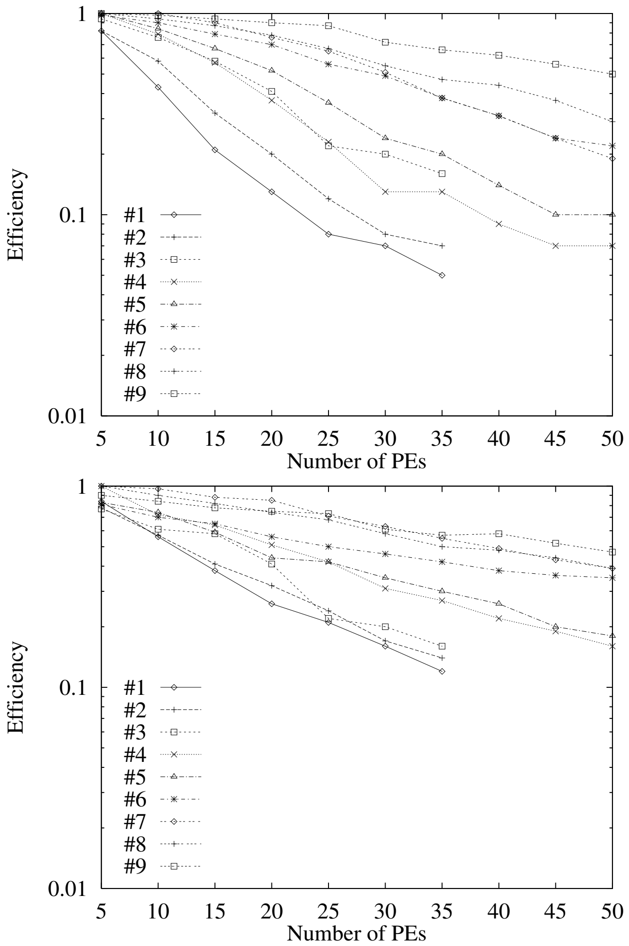

Figure 7: *Efficiency vs number of nodes for the relaxation phase (top) and training phase (bottom)*

chitecture than for the corresponding MLP: less training epochs are needed to reach a certain level of error, which can still diminish the total execution time; this aspect was not taken into account in our measurements.

We did not study the cost-effectiveness of running neural networks on a Paragon type system, because we are interested in experimenting with large neural networks, that would be very impractical (if at all possible) to run on sequential computers. Indeed, we are currently working on a continuous speech recognizer application involving an OWE architecture with a 22x12x6x6 LFF+B for the main MLP and a 44x4x1 LFF+B for each OWE. This architecture has 73,260 connections (about twice that of experiment #9) and the sizes of the training and test sets are respectively 17,000 and 6,000 vectors of dimension 66. 100 training epochs on a sequential computer take 5

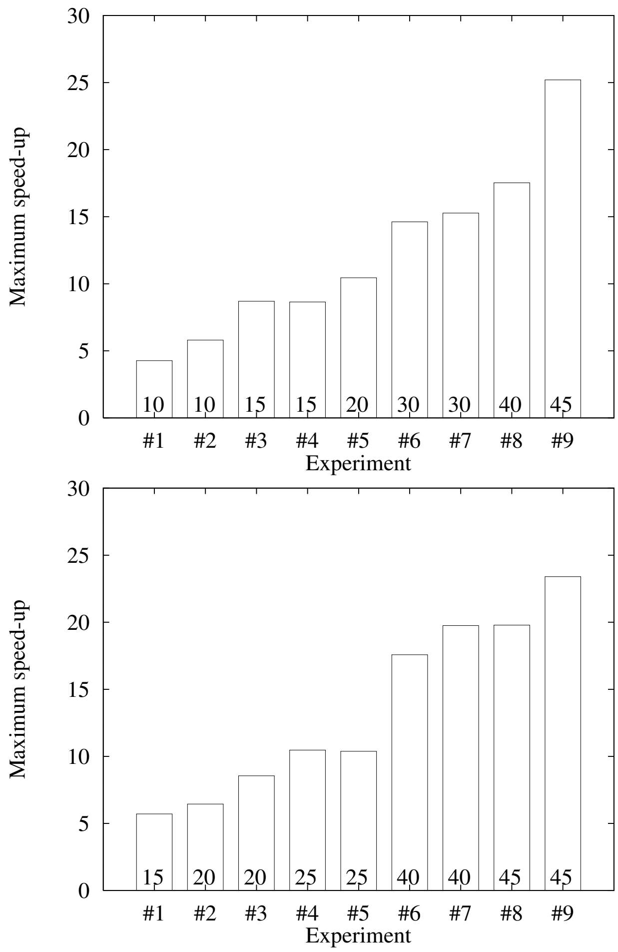

Figure 8: *Maximum speed-up and corresponding number of PEs for the relaxation phase (top) and the training phase (bottom)*

days versus 4 hours and 30 minutes on the Paragon computer with 30 nodes (reaching an efficiency of 88%). We actually need more than 100 training epochs; thus it is the parallel implementation that makes the very realization of this continuous speech recognizer possible [PIC 96c].

As stated in the first part of this paper, due to their huge demand in computer time, the evolution of research in MLPs is related to their parallel implementation: experimentation on large networks is certainly a key trend in the future of this research, and cannot be conducted on sequential computers. By taking advantage of supercomputer power, the OWE architecture and its proposed implementation allow to make a step in that direction.

Acknowledgements

We would like to thank Brandon Nelson for his helpful comments on earlier drafts of this article. We are also grateful to the IRISA (Computer research laboratory, Rennes, France) for access to its Intel Paragon computer. This research is supported in part by the ONR, Cognitive Science Program, Contract Number N00014-89-J-1975N158. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the ONR or the US Government.

References

[CHA 95] CHAUVIN Y. and RUMELHART D.E., editors, *Backpropagation: theory, architectures, and applications*. Lawrence Erlbaum Associates, 1995.

[FLA 89] FLATT H.P. and KENNEDY K., Performance of parallel processors, *Parallel computing*, 12, 1989.

[HER 92] HERTZ J., KROGH A. and PALMER R., *Introduction to the Theory Of Neural Computation*. Addison Wesley, 1992. Lecture notes, vol. 1.

[KER 92] KERCHOFFS E.J.H., WEDMAN F.W. and FRIETMAN E. E., Speeding up backpropagation training on hypercube computer. *Neurocomputing*, 4, 1992.

[KOT 93] KOTILAINEN P., SAARINEN J. and KASHI K., Neural network computation in a parallel multiprocessor architecture. *International Joint Conference on Neural Networks*, 1993.

[KUS 90] KUSZMAUL C., CHINN G., GRAJSKI K.A., CHEN C. and TOMBOULIAN S., Systolic Array Implementations of Neural Nets on the MasPar MP-1 Massively Parallel Processor. *International Joint Conference on Neural Networks*, San Diego, CA, 1990.

[LAL 97] LALLEMENT Y., CORNU T. and VIALLE S., Application development under ParCeL-1. In KITANO H., SUTTNER C. and GELLER J. editors, *Parallel Processing for Artificial Intelligence 3*. Elsevier, 1997.

[MAL 95] MALLUHI Q.M., BAYOUMI M.A. and RAO T. R., Efficient mapping of ANNs oh hybercube massively parallel machines. *IEEE transaction on computers*, 44, 6, June 1995.

[MÜL 94] MÜLLER U.A. and GUNZINGER A., Neural net simulation on parallel computers. *International conference on neural networks*, Orlando, Florida, 1994.

[NOR 92] NORDSTRÖM T. and SVENSSON B., Using and designing massively parallel computers for artificial neural networks. *Journal of parallel and distributed computing*, 14, 1992.

[PIC 94a] PICAN N., FORT J.-C. and ALEXANDRE F., Lateral Contribution Learning Algorithm for Multi-MLP Architecture. *European Symposium on Neural Networks*, Brussels, 1994. D Facto Brussels Belgium.

[PIC 94b] PICAN N., FORT J.-C. and ALEXANDRE F., An On-line Learning Algorithm for the Orthogonal Weight Estimation of MLP. *Neural Processing Letters*, 1, 1994.

[PIC 96a] PICAN N., An Orthogonal Delta Weight Estimator for MLP Architectures. *International conference on neural networks*, Washington, 1996.

[PIC 96b] PICAN N. and ALEXANDRE F., How OWE Architectures encode Contextual Effects in ANNs *Mathematics and Computers in Simulation*, 5-6, 41, 1996.

[PIC 96c] PICAN N., FOHR D. and MARI J.-F., HMMs and OWE Neural Network for Continuous Speech Recognition. *International Conference on Spoken Language Processing*, Philadelphia, 1996.

[SIN 90] SINGER A., Implementations of artificial neural networks on the connection machine. *Parallel Computing*, 14, 1990.

[VIA 94] VIALLE S., CORNU T. and LALLEMENT Y., ParCeL-1, User's Guide and Reference Manual. R-10 (Crin number 94-R-235), Supélec, établissement de Metz, France, 1994.

[ZOR 89] ZORBAS J.R., REBLE D.J. and VANKOOTEN R.E., Measuring the scalability of parallel computer systems. *Supercomputing*, 1989.