Using PVM for the Selection of Neural Networks Parameters

R. Baron, H. Paugam-Moisy
Laboratoire de l'Informatique du Parallélisme, URA CNRS 1398
ENS Lyon, 46 Allée d'Italie
69364 Lyon Cedex 07, France.
e-mail: `rbaron@lip.ens-lyon.fr`

October 7, 1994

1 Introduction

Neural networks are well known as universal approximators [HSW89]. In order to solve a given problem, a specific neural network must be trained during a learning phase. But the performances of the network depends on several parameters. There are parameters defining the neural network architecture (number of hidden layers, number of cells per layer,...), and parameters concerning the learning phase (learning rate, stop criterion,...).

Finding the best parameters has no theoretical answer. Neural network users are forced to try different sets of parameters, and they choose the best among these sets. This development phase is time-consuming, and parallel implementation may be of interest, as proposed in [PM93]. In this paper we first present the neural networks we are interested in, called *wavelet networks*. Then the application of PVM to the selection of good parameters for such networks is developed.

2 Wavelet Neural Networks

An artificial neural network is a group of cells connected together with a given architecture. In feed-forward networks, cells from a layer are connected to other cells from the immediately adjacent layer, and the signal propagates from input layer to output layer through hidden layers. The number of hidden layers, and the number of cells per hidden layer defines the architecture of the neural net. For the computation performed by each cell, the most commonly used transfer function is a sigmoid.

Recently, the theory of *wavelet decomposition* has been related to the neural network area [BZ92]. A function f can be approximated by $\tilde{f}(x) = \sum_{i=1}^{N} w_i \cdot h(D_i(x - t_i))$, where h is a *wavelet function*. This expression can be seen as the output of a neural network with one hidden layer, using wavelets as transfer functions.

In this paper, a wavelet neural network is used for approximating functions from $\mathbb{R}^n$ to $\mathbb{R}$. A feed-forward architecture with one hidden-layer is chosen. Then, the output of the network is computed by

$$g(x) = \sum_{i=1}^{N} w_i \cdot \psi(D_i(x - t_i)) + \overline{g} \tag{1}$$

where

- D_i is a dilation matrix, which is a diagonal matrix
- t_i is a translation vector
- $\overline{g}$ is a parameter introduced to help dealing with non-zero mean functions

In formula 1, t_i and $\overline{g}$ appear as biases. Hence they can be considered as weights, as well as w_i and D_i.

3 Learning Algorithm

From an input vector x, the neural network gives an output signal $s(x)$. Assuming f is the function to be approximated, the network performs well, if the network output $s(x)$ "is not too far" from the desired output $f(x)$ (according to an appropriate distance). An interest of neural networks is that the network may be trained to give a better output value. During the *learning phase*, the network is repeatedly presented a learning set $\mathcal{S}$. Each element of $\mathcal{S}$ is an example, i.e. an input/output pair $(x_k, f(x_k))$, where f is the function to be approximated. Weights $(w_i, D_i, t_i$ and $\overline{g})$ are modified by a *learning algorithm*, so that the output $s(x)$ becomes nearer from $f(x)$. For every weight, the learning rule is a gradient descent algorithm, like in back-propagation learning. The cost function to be minimized at each step is

$$c(x_k, y_k) = \frac{1}{2}(s(x_k) - f(x_k))^2$$

After each presentation of a pair $(x_k, f(x_k))$, the algorithm updates each weight in the opposite direction of the gradient of the functional c. At epoch k, each weight X (X stands for w_i, D_i, t_i *or* $\overline{g}$) of the network is modified, according to the formula

$$X_k = X_{k-1} - \gamma \ grad_X \ c(x_k, y_k) \tag{2}$$

Finally, a measure of the performance of a neural network after the learning phase is given by

$$c_T \ = \ \frac{1}{2|\mathcal{S}|} \sum_k (s(x_k) - f(x_k))^2$$

The less c_T is, the better the performance of the network is.

4 Parameter Space

There are several parameters to be defined, for a neural network to solve a specific problem. Assuming the network has only one hidden-layer, its architecture is fixed by the number C of cells in this hidden layer. This parameter C has a large influence on the network capabilities. The accuracy of the learning phase also depends on the number of presentations of examples to the neural network. This parameter is the number of epochs I of the learning phase. Finally, another effective parameter is the γ coefficient, which controls the gradient descent in formula 2. In this paper, γ will be constant, and the same for all X weights.

These three parameters have a large influence on the performance of the neural network. In order to have the most accurate networks, several sets of parameters must be tested. If N_C, N_I, N_G stands respectively for the number of values for the cells, the epochs and for the parameter γ, then $N = N_C.N_I.N_G$ sets should be tested. This time consuming operation strongly justify the use of a parallel software.

5 A Spy of Neural Network Capabilities

Instead of testing sets of parameters sequentially, several tests can be run at the same time, and the results must be collected. A master-slave architecture suits well. The master process computes the parameters to be tested, and send them to the slave processes. It collects the results from the slave processes, i.e. the performance of each neural network after the learning phase (average error). The slave processes receive the parameters, including one parameter defining the network architecture (number of hidden cells), and two parameters defining the learning algorithm (number of epochs of the learning phase, and γ parameter). Then, each process applies the learning algorithm to its network, and compute the average error. It sends this result back to the master process, and wait for another set of parameters, up to a end signal delivered by the master process.

For the master process to compute the parameter values it has to send to the slave processes, the user must specify, for each parameter, the range to be explored, and the number of different values which should be tested. Since N may be high, one process per set of parameters is not the best strategy. The number of slave processes will be restricted to P, each process dealing with successive sets of parameters. These processes would be distributed on several machines, for parallel processing.

6 Using PVM for Implementing the Spy

The PVM system [GBD+93] allows the management of several processes running at the same time on a heterogeneous network of parallel and serial computers. It provides library routines for initiating processes and for communicating. In this paper, PVM has been used on a network of Sun workstations, but the network can also include parallel machines. A number M of machines are chosen in the local network. A "rup" Unix command is used for selecting machines which are not too much loaded. Once the configuration file of PVM has been created, the master process starts running. It initiates a number P of slave processes per machine in the configuration. Given a number M of machines in the configuration, $M.P$ sets of parameters are tested simultaneously. These processes are initiated by the master process, using the *PvmTaskDefault mode*.

7 Load Balancing Problem

The running time required by a learning phase depends on the number of cells C and on the number of epochs I, but is independent of the value of the γ parameter (fig. 1).

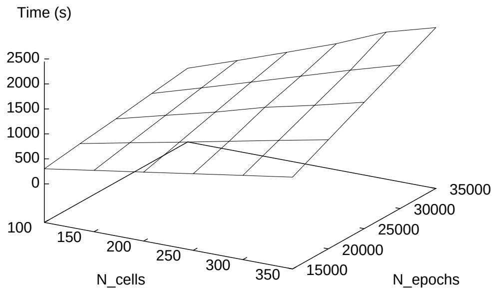

Figure 1: Running time

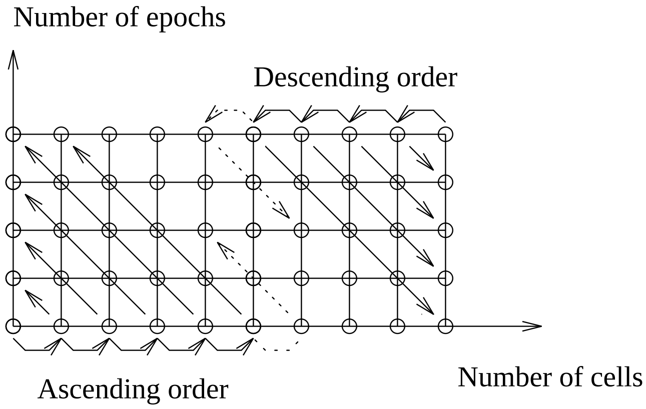

Figure 2: Parameter space exploration

Hence, the upper these two numbers, the longer the time for a slave process to send its results back. If the slave processes are not carefully initiated, a load balancing problem will occur. This drawback can be overcome by sending the parameters in a specific order. A first travel picks the (C, I) pairs in an ascending order, with diagonal $C + I = Constant$, and starting from the smallest values. A second travel picks the (C, I) pairs in a descending order, using diagonal $C + I = Constant$, but starting from the highest values of C and I (fig. 2). Initial sets of parameters are chosen in the first and the second order alternatively. Afterwards, as a slave process answers to the master, it specifies whether its previous parameters has came from the first or from the second order. The master then sends it back new parameters coming from the other order.

8 Results

Figures 3 and 4 show traces of executions, respectively without, and with the use of the ordering strategy. In the case of 6 slave processes on 6 workstations, for very low values of C and I, this toy experiment shows that the load balancing is better, when applying the ordering strategy. The gain would be much larger for realistic values of the parameters.

Table 1 summarizes several realistic experiments. There are $N = 24$ neural networks to be tested, and the number of simultaneous active processes P varies from 6 to 12.

The speed-up $s1$ is related to the parallel execution of 6 processes testing the parameters, and the second one, $s2$, is related to the sequential testing. Notice the parallel execution using 6 processes on 1 machine takes more time (33109 s) than the sequential testing, which takes 29880s. This is due to both the fact that parallel processes work in a time-sharing mode, and the fact that there is time lost in the message-passing. The all tests take about 9 hours, and the longest one takes 2450 s. Thus, the best reachable speed-up is given by the ratio of this two running times, i.e. 12.

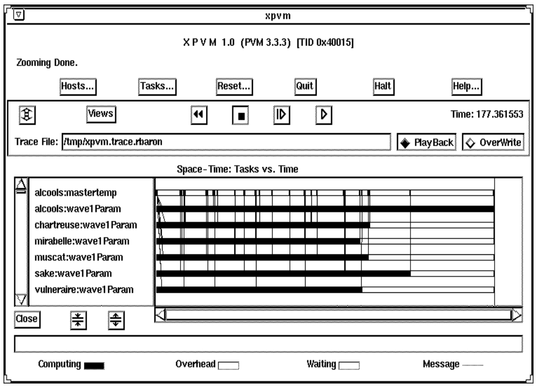

Figure 3: Execution without caution

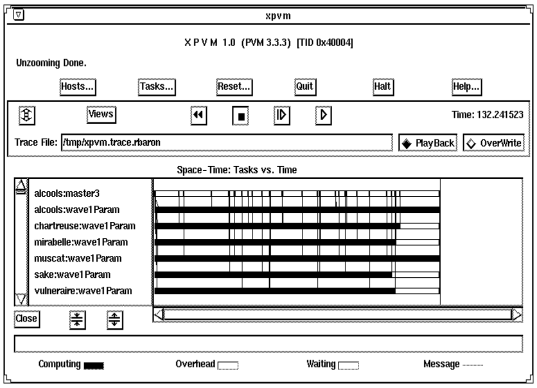

Figure 4: Execution with ordering strategy

# simultaneous processes	# machines	# processes/machines	time (s)	speed-up s1	speed-up s2
6	1	6	33109	1	0.90
	2	3	22440	1.47	1.33
	3	2	15067	2.19	1.98
	6	1	5837	5.67	5.11
12	12	1	3929	8.42	7.60

Table 1: Speed-up for several parallel configurations

The speed-up increases with the number of workstations in the network. However, this notion of speed-up is different from speed-up obtained on parallel multiprocessor machines. First, the network is composed of different workstations, with different computing capabilities. Second, computing capabilities are related to the workstation load. This load may evolve during slave processes running time, due to the multi-task and multi-user operating system. At the present time, a load measurement during running time is not implemented.

9 Conclusion

A distributed software written in PVM helps solving the neural network users problem of the choice of parameters. The above experiment allows us to select the best parameters, obtained for 300 cells and 30000 iterations. Future works include the study of the importance of the workstation network configuration. The parameters distribution could be improved to solve the load balancing problem. The master process should also be developed, in order to select the best parameters, and continue the exploration of the parameters space around these best parameters values [PM93]. Implementing this software on both parallel multi-processor machines (such as Paragon) and workstations, will be the next step of our work.

References

[BZ92] A. Benveniste and Q. Zhang. Wavelet networks. *IEEE Transactions on Neural Networks*, 3(6):889–898, November 1992.

[GBD+93] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM3 User's Guide and Reference Manual*. Oak Ridge National Laboratory, Oak Ridge, Tennesse, May 1993.

[HSW89] K. Hornik, S. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[PM93] H. Paugam-Moisy. Parallel neural network computing based on network duplicating. In I. Pitas, editor, *Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks*, pages 305–340. John-Wiley, 1993.