Talk presented: 1st European PVM Users' Group Meeting, Rome, October 9-11, 1994

Parallel Back-Propagation for the Prediction of Time Series

Frank M. Thiesing, Ulrich Middelberg, Oliver Vornberger

University of Osnabrück
Dept. of Math. / Computer Science
D-49069 Osnabrück, Germany
frank@informatik.uni-osnabrueck.de
http://brahms.informatik.uni-osnabrueck.de/prakt/prakt.html

Abstract

Artificial neural networks are suitable for the prediction of chaotic time series. A modified back-propagation algorithm with neuron splitting is used to train feed-forward multilayer perceptron networks for prediction. There are two ways of parallelizing: distributing the training set for batch learning or distribute the vector-matrix-operations for on-line training. Three implementation are compaired: PVM on a workstation cluster and PARIX and the new PVM/PARIX on a Transputer system. Results about the quality of forecasting an examplary time series and speedups of the parallel programs are presented.

1 Introduction

Artificial neural networks (ANN) are suitable for the prediction of chaotic time series. They can approximate any function after an amount of training. Especially for prediction ANNs are an alternative to the classical methods. Today ANNs are already applied to the calculation of the demand for electrical power and to the forecasting of economic data [1], [2], [3].

ANNs learn to approximate a function by presenting discrete values of this function to the net. For the prediction of time series values of the past – the so called *training set* – are given to the net. With n successive values in the input layer, the net is trained to calculate the $(n+1)$th value in the output layer (cf. Figure 1).

The learning of the entire training set is repeated until the error is less than a given bound. One path through the training set is called *epoch*. A trained net can be used for the prediction of a time series by presenting the last n known values. Then the net determines the next value for the future.

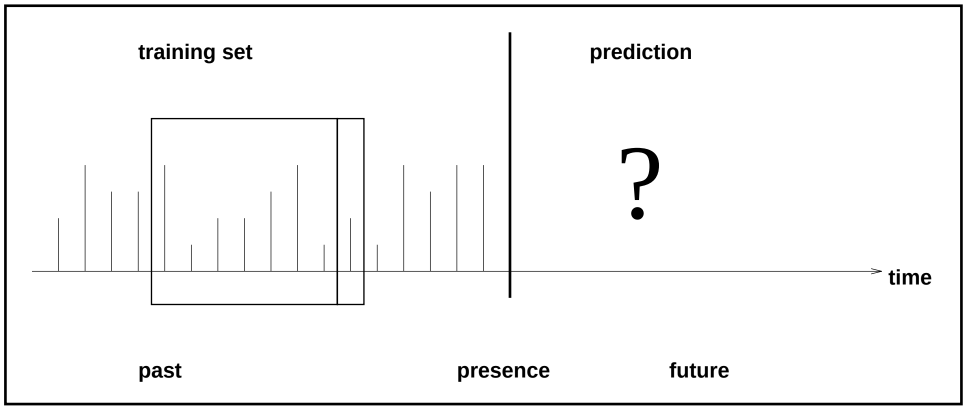


Figure 1: Prediction of Time Series

The *feed-forward multilayer perceptron* (FMP) network is used together with the *back-propagation* algorithm [4]. The error minimization by back-propagation takes an enormous amount of time but the training can be accelerated by efficient parallelizations with PVM and PARIX[1].

2 Modified Back-Propagation

The optimal configuration of a FMP network with its input, hidden and output layers is very difficult. The right number of hidden layers and the number of neurons within each layer is hard to find. Too many hidden neurons lead to a net that is not able to extract the function rule and takes more time for learning. With a lack in hidden neurons it is not possible to reach any error bound. Input and output layers are determined by the problem and the function that is to be approximated.

The back-propagation algorithm is used to minimize the error of the net by modifying the activation weights between the neurons. During the *forward* propagation the error between the nominal and the actual value is calculated. During the *backward* propagation the weights are modified in order to minimize this error. The basis of this method is gradient descent.

Our modified back-propagation algorithm [5] is able to increase the quality of a net by monotonic net incrementation. The training starts with a net of few hidden neurons. Badly trained neurons are split periodically while learning the training set. The old weights are distributed by chance between the two new neurons (cf. Figure 2). This is done until a maximum number of neurons within a hidden layer

[1]PARIX is a trademark of PARSYTEC GMBH

is reached. By training the net with the modified back-propagation algorithm a better minimum of the error is reached in shorter time.

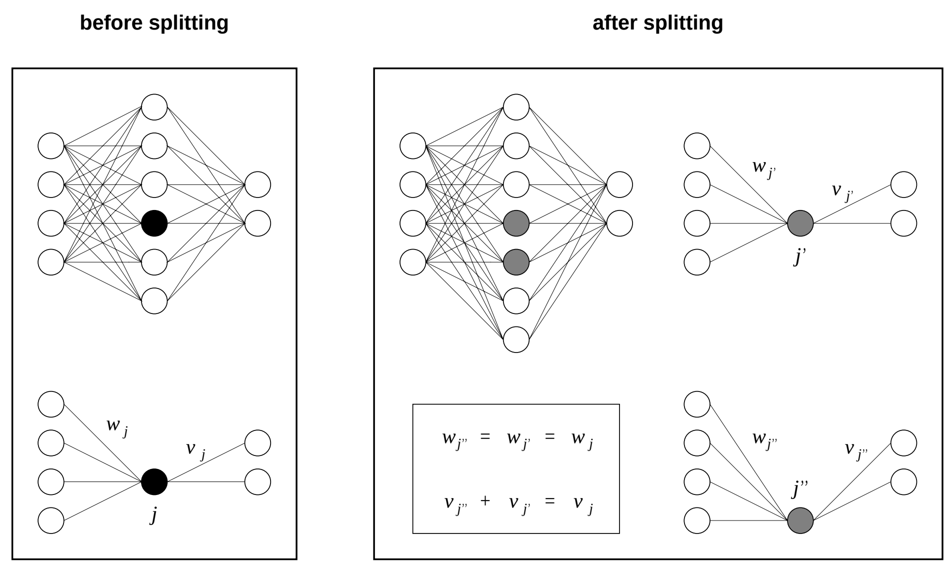


Figure 2: Modified back-propagation with neuron splitting

3 Parallelization

Because it takes very long to train a FMP, a parallelization of the algorithm is worthwhile. There are two completely different methods for training a MFP with consequences for the parallelization: *on-line* training and *batch* learning.

3.1 On-Line Training

The on-line training changes all the weights within each backward propagation after every item from the training set. Here the parallelization is very fine-grained. The vector-matrix-operations have to be calculated in parallel. This needs a lot of communication.

To reduce communication between the processors we use the idea of [6]. For each parallel calculated neuron its receptive and projective weights are stored on the responsible processor. Figure 3 shows the distribution of the neurons and the weight matrices under three processors.

3.2 Batch Learning

The alternative to on-line training is batch learning. For parallel batch learning the training set is divided and learned separately with some identical copies of the net in parallel. The weight corrections are summed up and globally corrected in all nets after each epoch (cf. Figure 4).

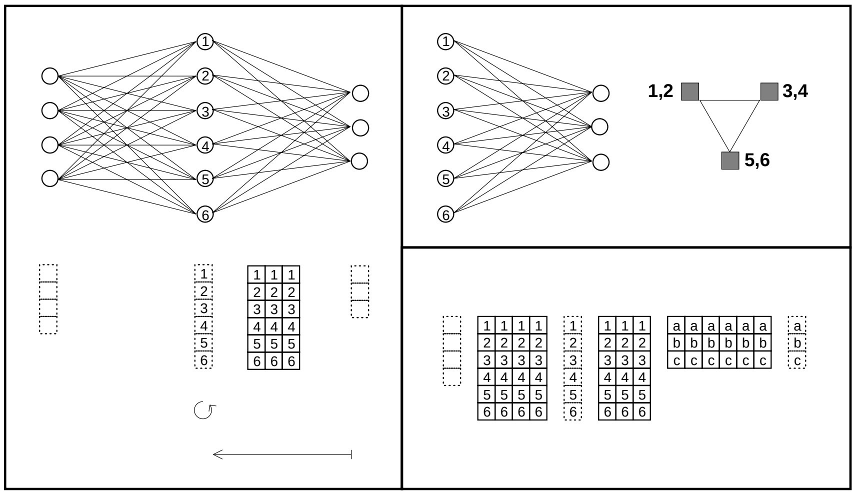


Figure 3: Parallel on-line backward propagation

Communication is only necessary for the calculation of the global sum of the weight corrections after each epoch. In addition to this a global broadcast has to be performed after the master node has calculated the random numbers for the new weights after splitting, but this happens very rarely.

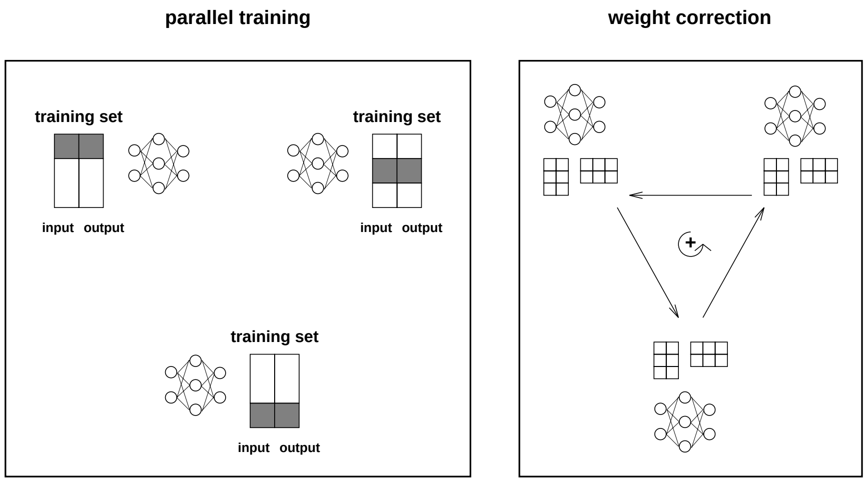


Figure 4: Parallel batch learning

The batch learning is different from the on-line training concerning the convergence speed and the quality of approximation.

4 Implementation

We use PVM 3.3.4 and XPVM 1.0.3 on a cluster of Sun Sparc workstations. Recently we run PVM/PARIX 1.0.1 by Parsytec on our system of T800 Transputers. For batch learning the sequential algorithm runs on every PVM node. Because of the different power of the computation nodes load balancing must be done. The part of the training set for a low-performance workstation has to be less than for a powerful one.

We are implementing the parallelization of the on-line training on our Transputer system with the runtime environment PARIX because of the high communication demands. The following time measurements show the better communication performance of PARIX.

5 Results

For testing the algorithms we use the chaotic series generated by the VERHULST process [7] with $r = 2.8$:

$$\boxed{x_{n+1} = x_n \cdot (1 + r \cdot (1 - x_n))} \qquad \text{(VERHULST process)}$$

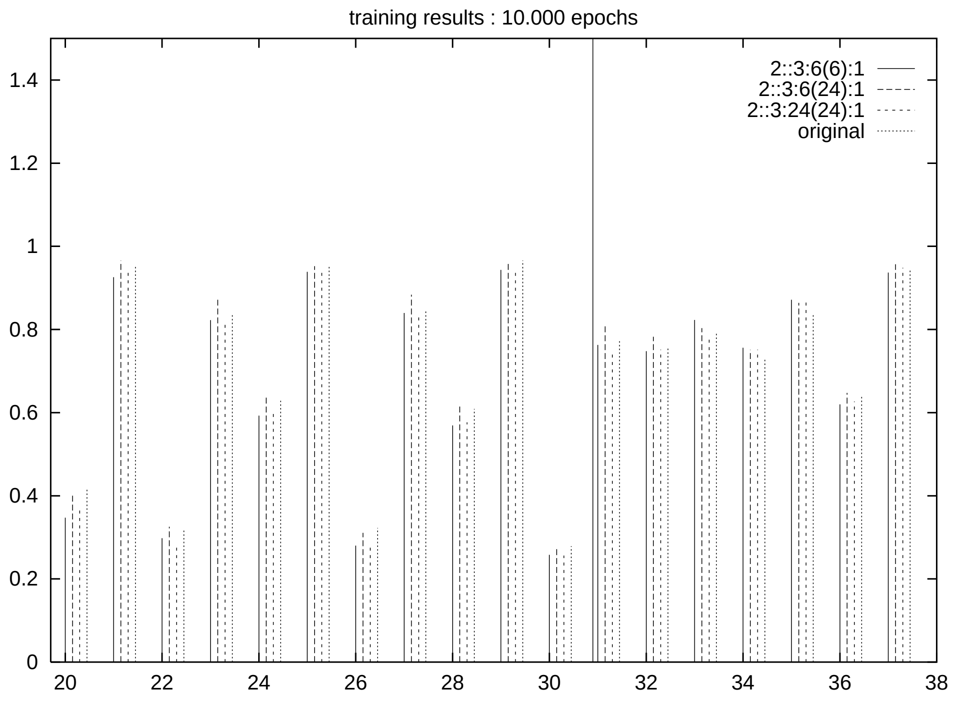


Figure 5: Comparison of training results with and without neuron splitting

The configuration of the FMP net is described by a string that contains topology information:

```
<layers>::<input>:<hidden1>(<max1>): ... :<output>
```

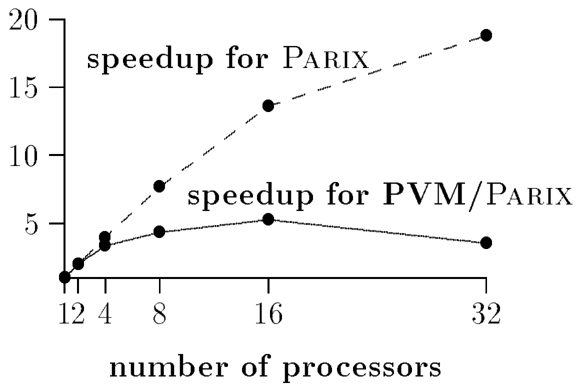
`<layers>` means the number of layers without the input layer. The other values are the numbers of neurons within the hidden layers at the beginning of the training and in parentheses the maximum number for this layer reachable by splitting with the modified back-propagation.

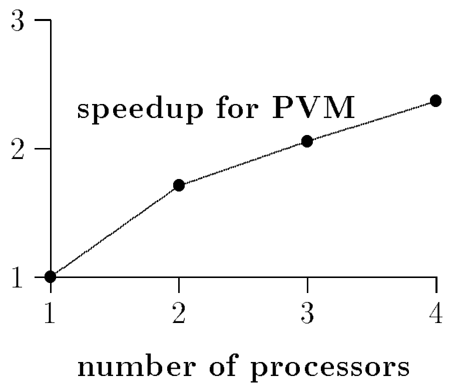
The results in Figure 5 show that the modified back-propagation for the net `2::3:6(24):1` is best in approximating the VERHULST process and predicting its values. The training set consists of the "original" values left from the vertical line. On the right the quality of the forecast can be seen for the different nets.

For time measurements we have trained this net by batch learning. The training times and speedups for different numbers of nodes are presented in the following tables.

Instance 1	Computing time in sec for			
	1 Sparc	2 Sparc	3 Sparc	4 Sparc
PVM	62.3	36.4	30.3	26.3

Instance 2	Computing time in sec for					
	1 T800	2 T800	4 T800	8 T800	16 T800	32 T800
PARIX	159.0	79.6	40.1	20.7	11.7	8.6
PVM/PARIX	159.0	79.9	47.4	36.6	30.2	44.9





Instance 1 has a training set of 100,000 items and instance 2 of 10,000 items that are both learned for 100 epochs. The results are very promising, but can surely be improved in the future.

6 Conclusion and Future Work

Our modified back-propagation algorithm for FMP can adapt chaotic time series very well. Together with the parallelization of batch learning under PVM the enormous amount of trainig time can be significantly reduced.

For the future we will adapt our implementations to the prediction of time series for economical consum processes.

References

[1] E. Pelikán, C. de Groot, D. Würtz. *Power Consumption in West-Bohemia: Improved Forecast with Decorrelating Connectionist Networks.* Neural Network World 2, pp. 701-712, 1992.

[2] R.G. Hoptroff, M.J. Bramson, T.J. Hall. *Forecasting Economic Turning Points With Neural Nets.* IEEE INNS Int. Joint Conf. on Neural Networks, Seattle, Vol. 1, pp. I347-I352, 1991.

[3] B. Widrow, D.E. Rumelhart, M.A. Lehr. *Neural Networks: Applications in Industry, Business and Science.* CACM March 1994, Vol. 37, No. 3, pp. 93-105, 1994.

[4] D.E. Rumelhart, G.E. Hinton, R.J. Williams. *Learning internal representations by error propagation.* In D.E. Rumelhart and J.L. McClelland (Eds.), Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1, pp. 318-362, MIT 1987.

[5] I. Glöckner. *Monotonic Incrementation of Backpropagation Modules and Network Architectures.* Project paper, University of Osnabrück, Dept. of Computer Linguistic und Artificial Intelligence, 1994.

[6] H. Yoon, J.H. Nang, S.R. Maeng. *A distributed backpropagation algorithm of neural networks on distributed-memory multiprocessors.* Proceedings of the 3rd symposium on the Frontiers of Massively Parallel Computation, pp. 358-363, IEEE 1990.

[7] H.-O. Peitgen, P.H. Richter. *The Beauty of Fractals.* Springer-Verlag, 1986.