Parallel Approach to Learning of the Recurrent Jordan Neural Network

Jarosław Bilski and Jacek Smoląg

Częstochowa University of Technology,
Częstochowa, Poland
{Jaroslaw.Bilski,Jacek.Smolag}@iisi.pcz.pl

Abstract. This paper presents the parallel architecture of the Jordan network learning algorithm. The proposed solution is based on the high parallel three dimensional structures to speed up learning performance. Detailed parallel neural network structures are explicitly shown.

1 Introduction

The Jordan network is an example of dynamical neural networks. Dynamical neural networks have been investigated by many scientists in the last decade [6], [7]. To train the dynamical networks the gradient method was used, see e.g. [15]. In the classical case the neural networks learning algorithms are implemented on serial computer. Unfortunatelly, this method is slow because the learning algorithm requires high computational load. Therefore, high performance dedicated parallel structure is a suitable solution, see eg. [2] - [5], [13], [14]. This paper presents a new concept of the parallel realisation of the Jordan learning algorithm. A single iteration of the parallel architecture requires much less computation cycles than a serial implementation. The efficiency of this new architecture is very satisfying and is explained in the last part of this paper. The structure of the Jordan network is shown in Fig. 1.

The Jordan network has K neurons in the hidden layer and M neurons in the network output. The input vector contains N input signals and M previous outputs. Note that previous signals from output are obtained through unit time delay z^{-1}. Therefore, the network input vector

$$\left[1, x_{1}^{(1)}(t), ..., x_{N}^{(1)}(t), x_{N+1}^{(1)}(t), ..., x_{N+M}^{(1)}(t)\right]^{T} \tag{1}$$

in the Jordan network takes the form

$$\left[1, x_{1}^{(1)}(t), ..., x_{N}^{(1)}(t), y_{1}^{(2)}(t-1), ..., y_{M}^{(2)}(t-1)\right]^{T} \tag{2}$$

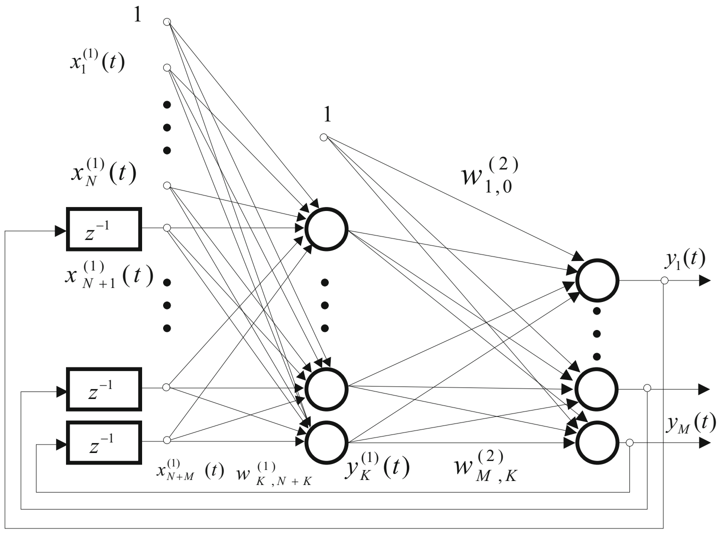


Fig. 1. Structure of the Jordan network

In the recall phase the network is described by

$$\begin{aligned}
s_i^{(1)} &= \sum_{k=0}^{N+M} w_{ik}^{(1)} x_k^{(1)} \\
y_k^{(1)}(t) &= f(s_k^{(1)}(t)) \\
s_j^{(2)} &= \sum_{k=0}^{K} w_{jk}^{(2)} x_k^{(2)} \\
y_j^{(2)}(t) &= f(s_j^{(2)}(t))
\end{aligned} \tag{3}$$

The parallel realisation of the recall phase algorithm uses architecture which requires many simple processing elements. The parallel realisation of the Jordan network in recal phase is depicted in Fig. 2a and its processing elements in Fig. 2b. Four kinds of functional processing elements are used in the proposed solution. The aim of the processing elements (PE) A is to delay outputs signals, so that values of signals appear on inputs of the network from previous instances. Processing elements of type B create matrix which includes values of weights of the first layer. The input signals are entered for rows elements parallelly, multiplied by weights and received results are summed in columns. The activation function for each neuron in the first layer is calculated after determination of product $\mathbf{w}_i^{(1)}\mathbf{x}^{(1)}$ in processing element of type D. The outputs of neurons in the first layer are inputs to the second layer simultaneously. The product $\mathbf{w}^{(2)}\mathbf{x}^{(2)}$ for the second layer is obtained in processing elements of type C similarly.

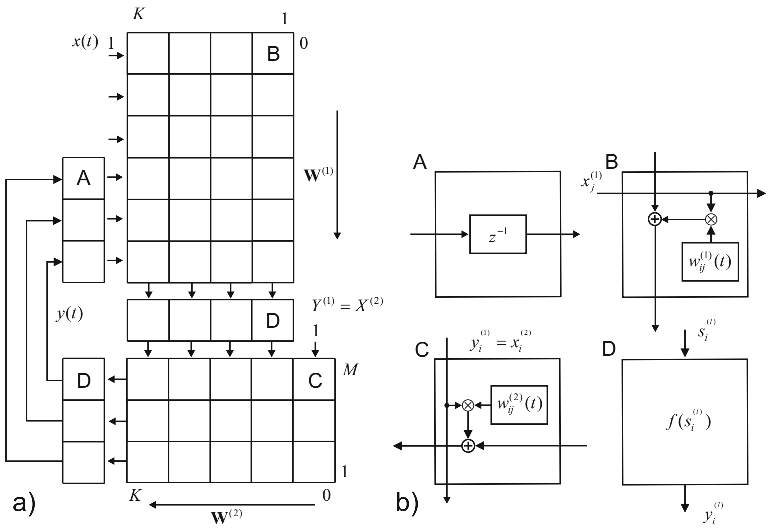


Fig. 2. Recal phase of the Jordan network and the structures of processing elements

The gradient method [15] is used to train the Jordan network. We minimise the following goal criterion

$$J\left(t\right) = \frac{1}{2}\sum\nolimits_{j=1}^{M} \varepsilon_j^{(2)^2}\left(t\right) = \frac{1}{2}\sum\nolimits_{j=1}^{M}\left(y_j^{(2)}\left(t\right) - d_j^{(2)}\left(t\right)\right)^2 \tag{4}$$

were $\varepsilon_j^{(2)}$ is defined as

$$\varepsilon_j^{(2)}(t) = y_j^{(2)}(t) - d_j^{(2)}(t) \tag{5}$$

For this purpose it is nesessary to calculate derivative of the goal funcion with respect to each weight. For weights in the second layer we obtain the following gradient

$$\nabla_{\alpha\beta}^{(2)} J\left(t\right) = \frac{\partial J\left(t\right)}{\partial w_{\alpha\beta}^{(2)}} = \sum\nolimits_{j=1}^{M} \varepsilon_j^{(2)}\left(t\right)\frac{dy_j^{(2)}\left(t\right)}{dw_{\alpha\beta}^{(2)}} \tag{6}$$

and after some calculations we obtain derivative

$$\begin{gathered}\frac{dy_j^{(2)}(t)}{dw_{\alpha\beta}^{(2)}} = \\ \delta_{j\alpha}\frac{dy_\alpha^{(2)}(t)}{ds_\alpha^{(2)}} y_\beta^{(1)}\left(t\right) + \frac{dy_j^{(2)}(t)}{ds_j^{(2)}}\sum\nolimits_{i=1}^{K} w_{ji}^{(2)}\frac{dy_i^{(1)}(t)}{ds_i^{(1)}}\sum\nolimits_{k=1}^{M} w_{i,k+N}^{(1)}\frac{dy_k^{(2)}(t-1)}{dw_{\alpha\beta}^{(2)}}\end{gathered} \tag{7}$$

Weights are updated according to the steepest descent algorithm as follows

$$w_{\alpha\beta}^{(2)}\left(t\right) = w_{\alpha\beta}^{(2)}\left(t-1\right) - \eta\nabla_{\alpha\beta}^{(2)} J\left(t\right) \tag{8}$$

For the first layer we have

$$\delta_j^{(2)}(t) = \varepsilon_j^{(2)}(t) \frac{dy_j^{(2)}(t)}{ds_j^{(2)}} \tag{9}$$

$$\varepsilon_i^{(1)}(t) = \sum\nolimits_{j=1}^{M} \delta_j^{(2)}(t)\, w_{ji}^{(2)} \tag{10}$$

and we obtain the gradient

$$\begin{gathered}\nabla_{\alpha\beta}^{(1)} J(t) = \\ \frac{\partial J(t)}{\partial w_{\alpha\beta}^{(1)}} = \sum\nolimits_{j=1}^{M} \varepsilon_j^{(2)}(t) \frac{dy_j^{(2)}(t)}{ds_j^{(2)}} \sum\nolimits_{i=1}^{K} \left[\frac{dy_i^{(1)}(t)}{dw_{\alpha\beta}^{(1)}} w_{ji}^{(2)} \right] = \sum\nolimits_{i=1}^{K} \frac{dy_i^{(1)}(t)}{dw_{\alpha\beta}^{(1)}} \varepsilon_i^{(1)}(t)\end{gathered} \tag{11}$$

After a few calculations we get

$$\begin{gathered}\frac{dy_i^{(1)}(t)}{dw_{\alpha\beta}^{(1)}} = \\ \delta_{i\alpha} \frac{dy_\alpha^{(1)}(t)}{ds_\alpha^{(1)}} x_\beta^{(1)}(t) + \frac{dy_i^{(1)}(t)}{ds_i^{(1)}} \sum\nolimits_{k=1}^{M} w_{i,k+N}^{(1)} \frac{dy_k^{(2)}(t-1)}{ds_k^{(2)}} \sum\nolimits_{l=1}^{K} w_{kl}^{(2)} \frac{dy_l^{(1)}(t-1)}{dw_{\alpha\beta}^{(1)}}\end{gathered} \tag{12}$$

and the weights can be updated by

$$w_{\alpha\beta}^{(1)}(t) = w_{\alpha\beta}^{(1)}(t-1) - \eta \nabla_{\alpha\beta}^{(1)} J(t) \tag{13}$$

The task of suggested parallel structure will be realisation of all calculations described by equations (6) - (8) and (11) - (13).

2 Parallel Realisation

In order to determine the derivative in the second layer it is required to know its previous values. Derivative values will be stored in E PE Fig. 4a. These elements will create 3D matrix of the dimension $M \times M \times (K+1)$, see Fig. 3. They will be useful for realizing inner sum in equation (7). Presented E PE multiply the respondent elements of derivative matrix $\frac{dy_j^{(2)}}{dw_{\alpha\beta}}$ by corresponding to them weights of the first layer, see Fig. 3. Then, received produtcts in the entire column are added to each other. The weights are delivered by columns. The first column is moved to the extreme right position (as a result of the rotation to the left) $W^{(1)}$ matrix. After a rotation of columns the previous actions are repeated. These operations are repeated K times until the first column of the matrix will revert to the original place. At the same time, the obtained results are sent to the upper 3D matrix of (F) PE (see Fig. 3 and Fig. 4b), multiplied by $w_{ji}^{(2)} \frac{dy_i^{(1)}}{ds_i^{(1)}}$ and $\frac{dy_j^{(2)}}{ds_j^{(2)}}$ and accumulated. The obtained results - calculation of equation (7) - are sent back to the lower 3D matrix. In the next step it is necessary to calculate

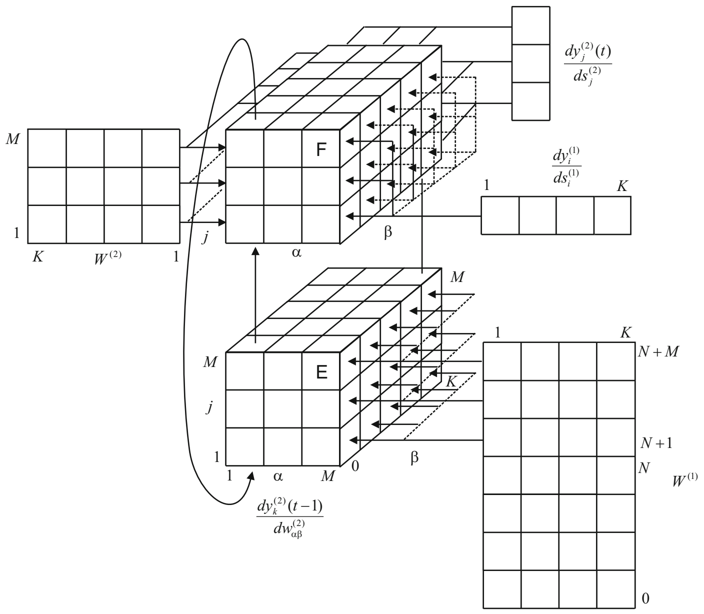

$\frac{dy_j^{(2)}(t)}{ds_j^{(2)}}$
M
F
$\frac{dy_i^{(1)}}{ds_i^{(1)}}$
1
K
1
K
$W^{(2)}$
1
j
α
β
M
M
E
1
K
j
1
1
α
M
0
β
N+M
N+1
N
$W^{(1)}$
0
$\frac{dy_k^{(2)}(t-1)}{dw_{\alpha\beta}^{(2)}}$

Fig. 3. Idea of learning the second layer

gradients eq.(6). This is depicted in Fig. 5. The element for weights updating in eq.(8) is shown in Fig. 6.

Suggested solution leads to acceleration of calculations, but it is not optimal solution yet. It results from the fact that after multiplication of lower 3D matrix and the weights matrix, serial summation follows. In this case multiplication and addition is realized in $M \times K$ steps. It is easily seen that changing manner

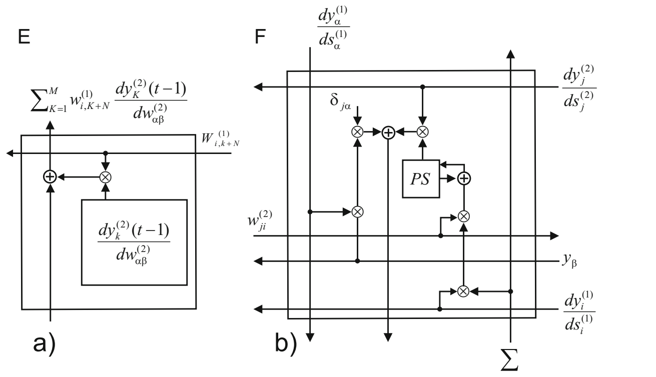

E
F
$\sum_{K=1}^{M} w_{i,K+N}^{(1)} \frac{dy_K^{(2)}(t-1)}{dw_{\alpha\beta}^{(2)}}$
$W_{i,k+N}^{(1)}$
$\frac{dy_k^{(2)}(t-1)}{dw_{\alpha\beta}^{(2)}}$
a)
$\frac{dy_\alpha^{(1)}}{ds_\alpha^{(1)}}$
$\frac{dy_j^{(2)}}{ds_j^{(2)}}$
$\delta_{j\alpha}$
PS
$w_{ji}^{(2)}$
y_β
$\frac{dy_i^{(1)}}{ds_i^{(1)}}$
b)
Σ

Fig. 4. The structure of the processing elements for learning the second layer

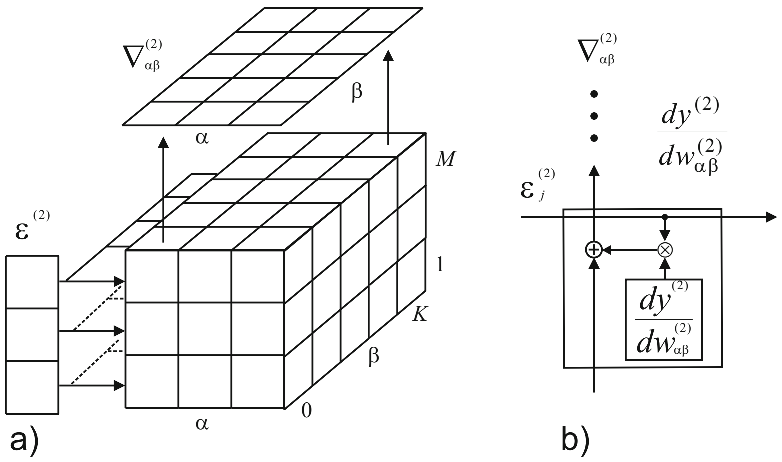

$\nabla_{\alpha\beta}^{(2)}$
β
α
M
$\varepsilon^{(2)}$
1
K
β
0
α
a)
$\nabla_{\alpha\beta}^{(2)}$
$\frac{dy^{(2)}}{dw_{\alpha\beta}^{(2)}}$
$\varepsilon_j^{(2)}$
$\frac{dy^{(2)}}{dw_{\alpha\beta}^{(2)}}$
b)

Fig. 5. Architecture for calculating of the gradient $\nabla_{\alpha\beta}^{(2)}$ for second layer learning and the structure of the processing element

of entering of values from weights matrix to derivatives matrix we can reduce the amount of steps required for execution of the multiplication and addition operations to $M + K - 1$. The manner of these weights entering is presented in Fig. 7. The multiplication is realised only for elements from the first column depicted by the thick line. In the first step only one element from the last row is taken into account. In the next cycles the number of rows is incremented, and the rows that have participated in multiplication are subject to rotation. Rotation is done from step one to the left until all rows reach the starting position. The rows are no longer included in the multiplication. As a result, the proposed modifications in subsequent steps, making the multiplication and summation, as described in the previous scenario. In this case we will receive the sum of the new inner product without waiting the M steps. For the first layer we need to calculate the derivatives (12). Note that equations (12) and (7) have identical structures. Therefore parallel realisation of the first layer learning is analogous to the second layer learning. The architecture of the first layer learning is shown in Fig. 8. Of course in this case the dimensions of the structure and processed data

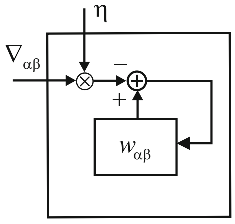


Fig. 6. The weights updating element

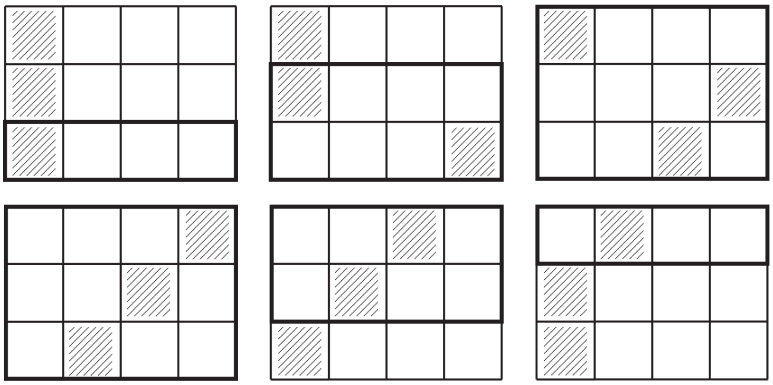
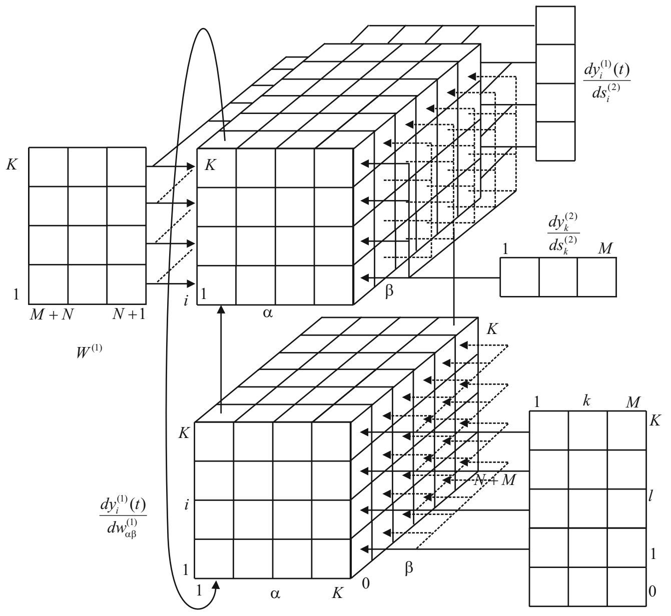


Fig. 7. Method of entering weights for the second layer learning

correspond to equation (12). After obtaining the derivatives, the value of $\epsilon_i^{(1)}(t)$ is calculated, see eq. (9) and (10). The structure for this operation is presented in Fig. 9. The gradient in eq. (11) is obtained from the analogous structure like in the second layer (see Fig. 5). All weights are updated in the same time by elements depicted in Fig. 6.



$\frac{dy_i^{(1)}(t)}{ds_i^{(2)}}$
$\frac{dy_k^{(2)}}{ds_k^{(2)}}$
1
M
K
1
$M+N$
$N+1$
$W^{(1)}$
K
i
1
α
β
K
1
k
M
K
l
1
0
$N+M$
K
i
1
$\frac{dy_i^{(1)}(t)}{dw_{\alpha\beta}^{(1)}}$
1
α
K
0
β

Fig. 8. Idea of learning of the first layer

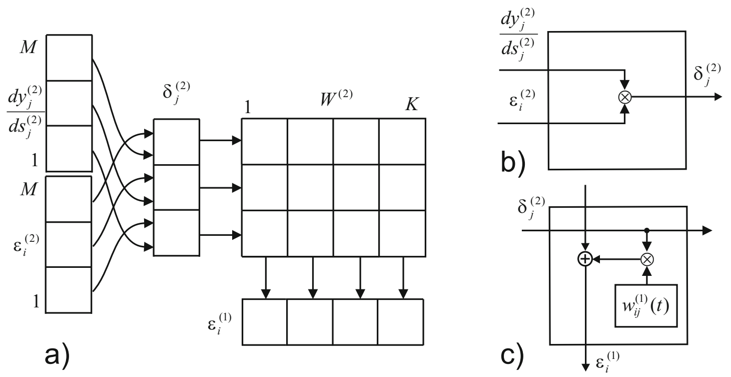


Fig. 9. Structure for calculating $\epsilon_i^{(1)}(t)$ in the first layer (a) and the processing elements (b) and (c)

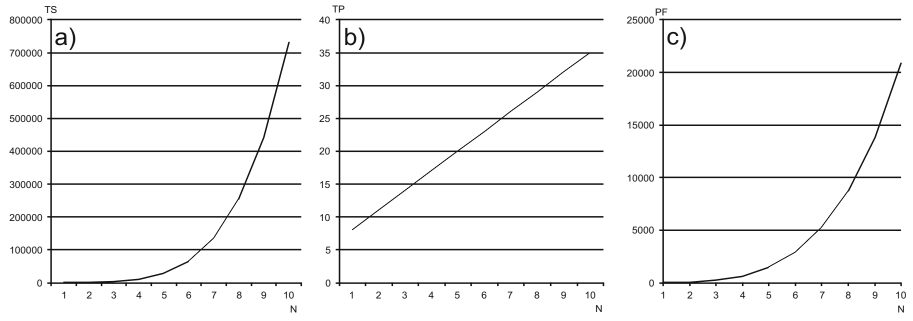


Fig. 10. Number of times cycles in a) classical (serial), b) parallel implementation and c) performance factor

3 Conclusion

In this paper the parallel realisation of the Jordan neural network was proposed. We assume that all multiplications and additions operations take the same time unit. For simplicity of the result presentation we assume that K=M=N in the network. We can compare computational performance of the Jordan parallel implementation with sequential architectures up to N=M=10 for inputs and outputs and up to 10 neurons (K) in the hidden layer of neural network. Computational complexity of the Jordan learning is of order $\mathcal{O}(K^5)$ and equals $TS = 2K^3M^2 + 2K^3MN + 2K^2M^3 + 2K^3M + 4K^2M^2 + 3K^2MN + 2KM^3 + 7K^2M + 4K^2N + 8KM^2 + KMN + 4K^2 + 6M^2$. In the presented parallel architecture each iteration requires only $TP = K + M + \max(K, M) + 5$ time units (see Fig. 10). Performance factor ($PF = TS/TP$) of parallel realisation of the Jordan algorithm achieves nearly 21000 for N=10 inputs, K=10 neurons in the hidden layer and M=10 of neurons in the output layer and it grows very fast when these numbers grow, see Fig. 10. We observed that the performance of the proposed solution is promising. Analogous parallel aproach can be used for the advanced learning algorithm of feedforward neural networks, see eg. [1]. In the future research we plan to design parallel realisation of learning of other structures including probabilistic neural networks [9]-[11] and various fuzzy structures[8],[12].

References

1. Bilski, J.: The UD RLS Algorithm for Training the Feedforward Neural Networks. International Journal of Applied Mathematics and Computer Science 15(1), 101–109 (2005)
2. Bilski, J., Litwiński, S., Smoląg, J.: Parallel realisation of QR algorithm for neural networks learning. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS (LNAI), vol. 3070, pp. 158–165. Springer, Heidelberg (2004)
3. Bilski, J., Smoląg, J.: Parallel realisation of the recurrent RTRN neural network learning. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2008. LNCS (LNAI), vol. 5097, pp. 11–16. Springer, Heidelberg (2008)
4. Bilski, J., Smoląg, J.: Parallel Realisation of the Recurrent Elman Neural Network Learning. In: Rutkowski, L., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2010, Part II. LNCS(LNAI), vol. 6114, pp. 19–25. Springer, Heidelberg (2010)
5. Bilski, J., Smoląg, J.: Parallel Realisation of the Recurrent Multi Layer Perceptron Learning. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2012, Part I. LNCS(LNAI), vol. 7267, pp. 12–20. Springer, Heidelberg (2012)
6. Kolen, J.F., Kremer, S.C.: A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press (2001)
7. Korbicz, J., Patan, K., Obuchowicz, A.: Dynamic neural networks for process modelling in fault detection and isolation. Int. J. Appl. Math. Comput. Sci. 9(3), 519–546 (1999)
8. Li, X., Er, M.J., Lim, B.S., et al.: Fuzzy Regression Modeling for Tool Performance Prediction and Degradation Detection. International Journal of Neural Systems 20(5), 405–419 (2010)
9. Rutkowski, L.: Multiple Fourier series procedures for extraction of nonlinear regressions from noisy data. IEEE Transactions on Signal Processing 41(10), 3062–3065 (1993)
10. Rutkowski, L.: Non-parametric learning algorithms in the time-varying environments. Signal Processing 18(2), 129–137 (1989)
11. Rutkowski, L.: Generalized regression neural networks in time-varying environment. IEEE Trans. Neural Networks 15, 576–596 (2004)
12. Rutkowski, L., Przybył, A., Cpałka, K.: Novel Online Speed Profile Generation for Industrial Machine Tool Based on Flexible Neuro-Fuzzy Approximation. IEEE Transactions on Industrial Electronics 59(2), 1238–1247 (2012)
13. Smoląg, J., Bilski, J.: A systolic array for fast learning of neural networks. In: Proc. of V Conf. Neural Networks and Soft Computing, Zakopane, pp. 754–758 (2000)
14. Smoląg, J., Rutkowski, L., Bilski, J.: Systolic array for neural networks. In: Proc. of IV Conf. Neural Networks and Their Applications, Zakopane, pp. 487–497 (1999)
15. Williams, R., Zipser, D.: A learning algorithm for continually running fully recurrent neural networks. Neural Computation, 270–280 (1989)