Parallel Realisation of the Recurrent Multi Layer Perceptron Learning

Jarosław Bilski and Jacek Smoląg

Department of Computer Engineering,
Częstochowa University of Technology,
Częstochowa, Poland
{Jaroslaw.Bilski,Jacek.Smolag}@kik.pcz.pl

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

1 Introduction

The RMLP network is an example of dynamical neural networks. Dynamical neural networks have been investigated by many scientists for the last decade [4], [5]. To train the dynamical networks the gradient method was used eg. [8]. 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, eg. [1] - [3], [6], [7]. This paper contains a new concept of the parallel realisation of the RMPL 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 satisfing and is explained in the last part of this paper. The structure of the RMPL network is shown in Fig. 1.

The RMLP network has K neurons in the hidden layer and one neuron in the network output. The input vector contains input signal, its N previous values and M previous outputs. Note, the previous signals from input and output are obtained through unit time delay z^{-1}. Therefore, the network function is

$$y^{(2)}(t+1) = f\left(\begin{array}{c} x^{(1)}(t), x^{(1)}(t-1), \ldots, x^{(1)}(t-(N-1)), \\ y^{(2)}(t-1), \ldots, y^{(2)}(t-M) \end{array}\right) \tag{1}$$

In the recall phase the network is described by

$$\begin{array}{l} s_i^{(1)} = \sum_{j=0}^{N+M} w_{ij}^{(1)} x_j^{(1)} \\ y_i^{(1)} = f\left(s_i^{(1)}\right); \quad x_i^{(2)} = y_i^{(1)} \\ s^{(2)} = \sum_{i=0}^{K} w_i^{(2)} x_i^{(2)} \\ y^{(2)} = f\left(s^{(2)}\right) \end{array} \tag{2}$$

L. Rutkowski et al. (Eds.): ICAISC 2012, Part I, LNCS 7267, pp. 12–20, 2012.

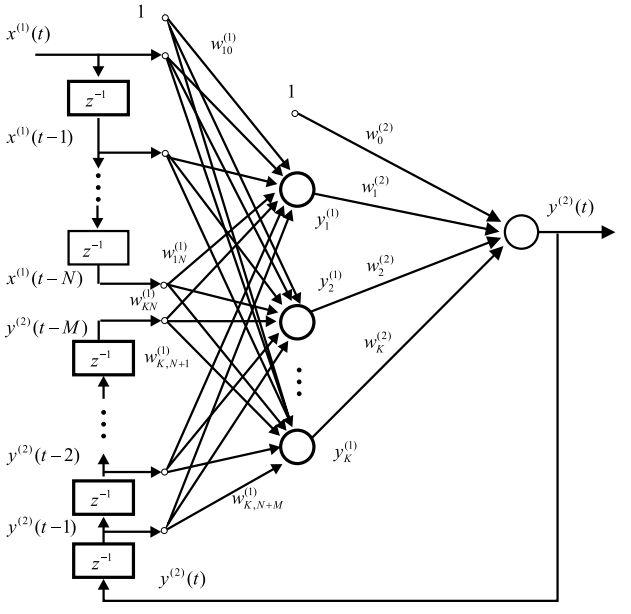

Fig. 1. Structure of the RMLP network

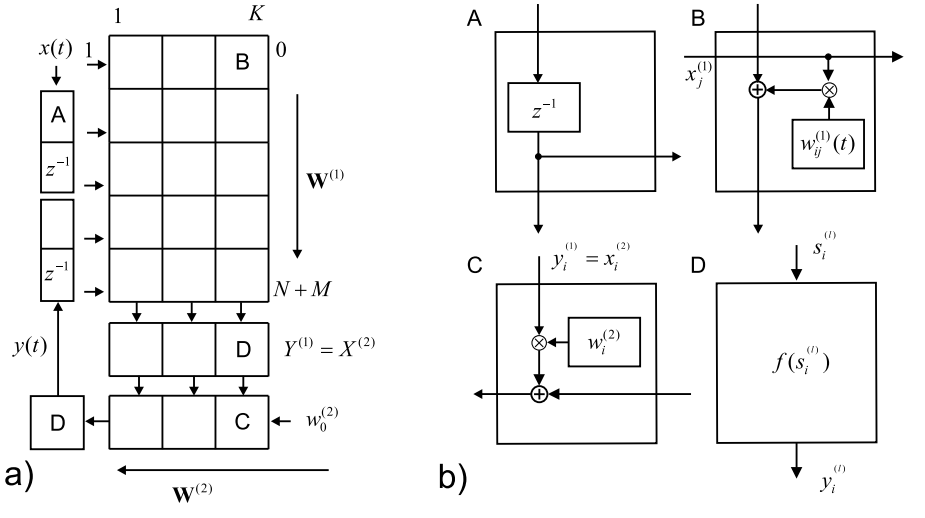

Fig. 2. Recal phase of the RMLP network and the structures of processing elements

Parallel realisation of the recall phase algorithm uses architecture which requires many simple processing elements. The parallel realisation of the RMLP network in recal phase is depicted in Fig. 2a and its processing elements in Fig. 2b. Four kinds of functional processing elements take put in the proposed solution. The aim of PE (A) is to delay inputs and outputs signals, so that values of signals appear on inputs of network from previous instances. 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 un the first layer is calculated after calculation of product $\mathbf{w}_i^{(1)}\mathbf{x}^{(1)}$ in element of type (D). The outputs of neurons in the first layer are inputs the second layer simultaneously. The product $\mathbf{w}^{(2)}\mathbf{x}^{(2)}$ for the second layer is obtained in elements of type (C) similarly.

The gradient method is used to train the RMLP network. 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 derivative

$$\begin{array}{l} \frac{dy^{(2)}(t)}{dw_\alpha^{(2)}} = \\ \frac{df\left(s^{(2)}(t)\right)}{ds^{(2)}(t)} \left[y_\alpha^{(1)}(t) + \sum_{i=0}^{K} w_i^{(2)} \frac{df\left(s_i^{(1)}(t)\right)}{ds_i^{(1)}(t)} \sum_{j=1}^{M} w_{i,j+N}^{(1)} \frac{dy^{(2)}(t-M-1+j)}{dw_\alpha^{(2)}} \right] \end{array} \tag{3}$$

Weights are updated according to the steepest descent algorithm as follows

$$\Delta w_\alpha^{(2)} = -\eta \left(y^{(2)}(t) - d^{(2)}(t) \right) \frac{dy^{(2)}(t)}{dw_\alpha^{(2)}} = -\eta \varepsilon^{(2)}(t) \frac{dy^{(2)}(t)}{dw_\alpha^{(2)}} \tag{4}$$

For the first layer we obtain the derivative

$$\begin{array}{l} \frac{dy^{(2)}(t)}{dw_{\alpha\beta}^{(1)}} = \\ \frac{df\left(s^{(2)}(t)\right)}{ds^{(2)}(t)} \sum_{i=1}^{K} w_i^{(2)} \frac{df\left(s_i^{(1)}(t)\right)}{ds_i^{(1)}(t)} \left[\sum_{j=1}^{M} w_{i,j+N}^{(1)} \frac{dy^{(2)}(t-M-1+j)}{dw_{\alpha\beta}^{(2)}} + \delta_{i\alpha} x_\beta^{(1)}(t) \right] \end{array} \tag{5}$$

and the weights can be updated by

$$\Delta w_{\alpha\beta}^{(1)} = -\eta \left(y^{(2)}(t) - d^{(2)}(t) \right) \frac{dy^{(2)}(t)}{dw_{\alpha\beta}^{(1)}} = -\eta \varepsilon^{(2)}(t) \frac{dy^{(2)}(t)}{dw_{\alpha\beta}^{(1)}} \tag{6}$$

The task of suggested parallel structure will be realisation of all calculations described by equations (3), (4) and (5), (6).

2 Parallel Realisation

In order to determine derivative in the second layer it is required to know its previous values. Derivative values will be stored in (E) PE Fig. 3b. These elements will create matrix of the dimension $M(K+1)$ Fig. 3a. They will be useful

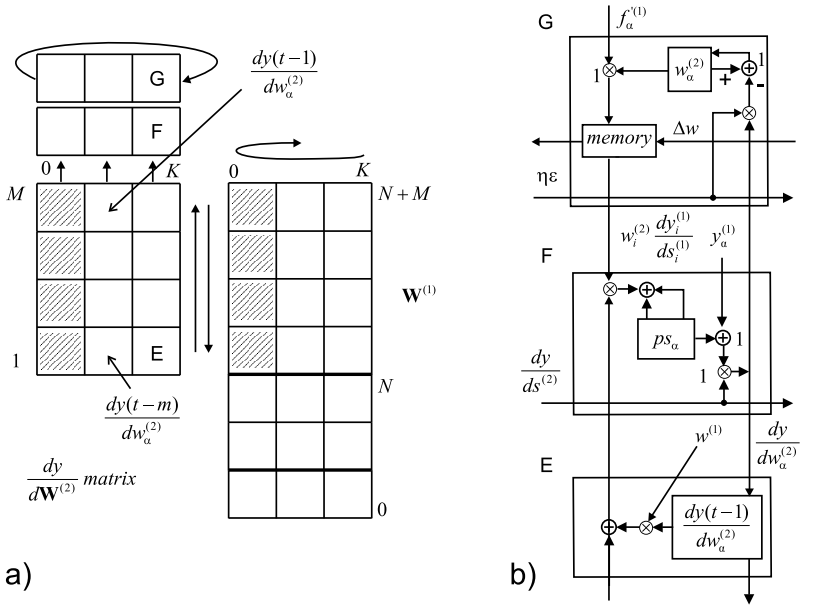

Fig. 3. Idea of learning the second layer and the processing elements

for realizing operations given by equations (5) and (6). Presented idea relies on multiplication of respondent elements of derivative matrix $\frac{dy}{dw_\alpha}$ by corresponding to them weights of the first layer Fig. 3a. Then, received produtcts in the entire column are added to each other. At the same time, the result obtained is multiplied by $w_i^{(2)} \frac{dy_i^{(1)}}{ds_i^{(1)}}$ and accumulated. In the next step, 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 $(K + 1)$ times until the first column of the matrix will revert to the original place. The value of $y_\alpha^{(1)}$ is added to accumulated value and next the sum is multiplied by derivative $\frac{dy}{ds^{(2)}}$. In this way the new value of the derivative $\frac{dy}{dw_\alpha^{(1)}}$ is obtained.

The calculated value of the derivative is placed in the top row of the array, and then is moving down. This newly calculated derivative is used in PE (G) to update the second layer weights according to the equation (4). Suggested solution leads to acceleration of calculations, but it is not optimal solution yet. It results from the fact that after multiplication of both matrices, serial summation follows. In this case multiplication and addition is realized in $M(K + 1)$ steps. It is easily seen that changing manner 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 weights entering is presented in Fig. 4. The multiplication is realised only for elements depicted by the thick line. In the first step only last row is taken into

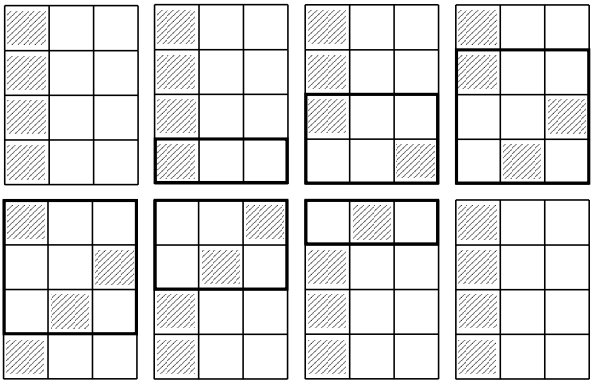

Fig. 4. Method of entering weights for the second layer learning

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 first scenario. In this case we will receive the sum of the new inner product without waiting the M steps. For the first layer the derivatives $\frac{dy(t-j)}{dw_{\alpha\beta}}$ are placed

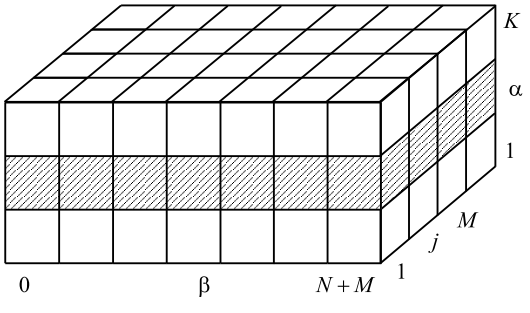

Fig. 5. Cuboid matrix of the derivatives $\frac{dy(t-j)}{dw_{\alpha\beta}}$ for first layer learning

in the cuboid matrix of the processing elements, see Fig. 5. It can be splited into K matrices (Fig. 6) which are parallely processed. For simplicity next figures show structures only for one such matrix. The architecture of processing elements dedicated to realization of first layer lerning is shown in Fig. 7a. In this case weights are given step by step to the derivatives matrix, in which the total sums are calculated according to eq. (5). Then, in the elements (F), see Fig. 3b, above the array new values of derivatives are calculated. These values are sent back to

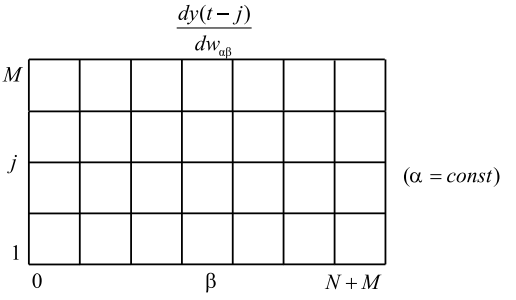

Fig. 6. Single matrix (2D) of the cuboid matrix (3D)

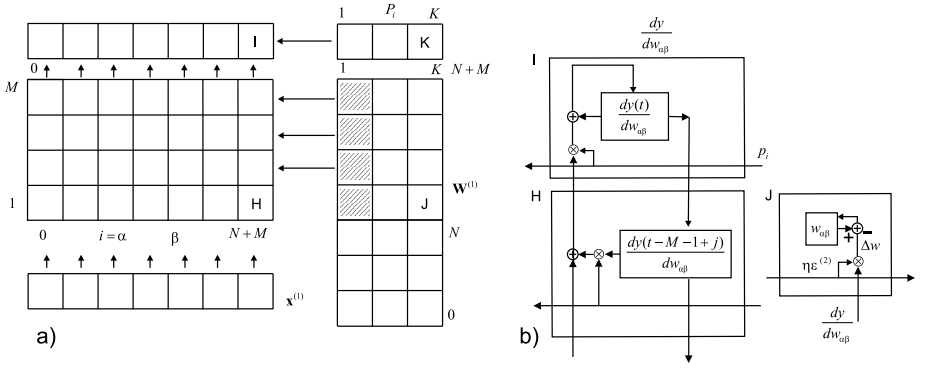

Fig. 7. Idea of learning for first layer and the processing elements

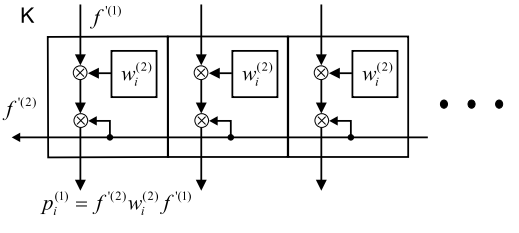

Fig. 8. The auxiliary vector

the derivatives matrix, and in this way the new value of derivatives overrides the previous etc. The newly obtained values of derrivatives from all two dimensional matrices are used to updating weights (6). This is done in elements of the type (J), see Fig. 7b. The use of these elements simplifies the calculation of the vector P see (5) and Fig. 8.

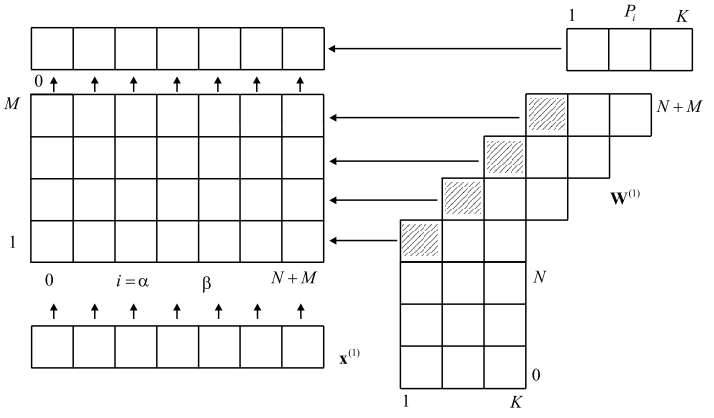

Fig. 9. Practical structure for first layer learning

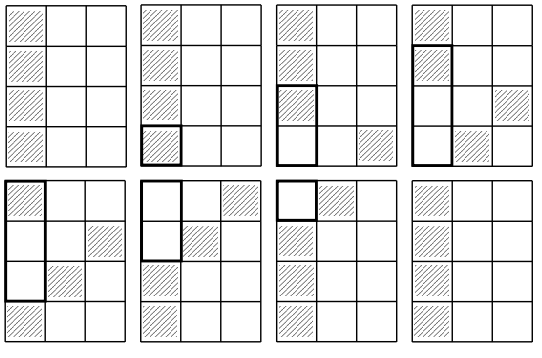

Fig. 10. Method of entering weights for the first layer learning

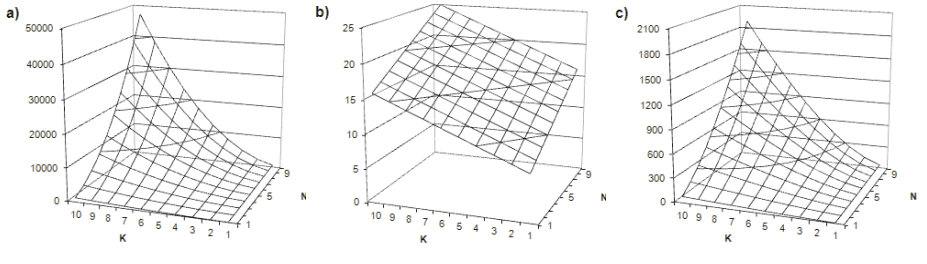

Fig. 11. Number of times cycles in a) classical, b) parallel implementation and c) performance factor classical/parallel

The optimal performance of the structure is obtained by the specific way of sending weight values. The practical structure for the first layer learning is shown in Fig. 9. Weights are sent in the following steps as indicated by the thick line in Fig. 10 The layout of all weights is identical in Fig. 10 and Fig. 4 which means that weights of the first layer, necessary for the calculations in the first and second layers can by sent fully paralelly.

3 Conclusion

In this paper the parallel realisation of the RMLP neural network was proposed. We assume that all multiplications and additions operations take the same time unit. For simplicity of the result presentation we suppose that M=N in the input vector of the network.

We can compare computational performance of the RMLP parallel implementation with sequential architectures up to N=M=10 for inputs and 10 neurons (K) in the hidden layer of neural network. Computational complexity of the RMPL learning is of order $\mathcal{O}(K^4)$ and equals $4M^2K^2 + 6MK^2 + 10MK + K^2 + 2M + 9K + 8$. In the presented parallel architecture each iteration requires only $K + M + 5$ time units (see Fig. 11). Performance factor (see Fig. 11) of parallel realisation of the RMLP algorithm achieves nearly 1900 for N=M=10 inputs and K=10 of neurons in the hidden layer and it grows fast when those numbers grow. We observed that the performance of the proposed solution is promising. In the future research we plan to design parallel realisation of learning of neuro-fuzzy structures [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19].

References

1. 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)
2. 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)
3. 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. LNCS (LNAI), vol. 6114, pp. 19–25. Springer, Heidelberg (2010)
4. Kolen, J.F., Kremer, S.C.: A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press (2001)
5. 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)
6. 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)
7. 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)

8. Williams, R., Zipser, D.: A learning algorithm for continually running fully recurrent neural networks. Neural Computation, 270–280 (1989)
9. Korytkowski, M., Scherer, R., Rutkowski, L.: On Combining Backpropagation with Boosting. In: International Joint Conference on Neural Networks, IEEE World Congress on Computational Intelligence, Vancouver, BC, Canada, pp. 1274–1277 (2006)
10. Korytkowski, M., Rutkowski, L., Scherer, R.: From Ensemble of Fuzzy Classifiers to Single Fuzzy Rule Base Classifier. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2008. LNCS (LNAI), vol. 5097, pp. 265–272. Springer, Heidelberg (2008)
11. Nowicki, R.: Rough Sets in the Neuro-Fuzzy Architectures Based on Monotonic Fuzzy Implications. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS (LNAI), vol. 3070, pp. 510–517. Springer, Heidelberg (2004)
12. Nowicki, R.: Rough Sets in the Neuro-Fuzzy Architectures Based on Nonmonotonic Fuzzy Implications. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS (LNAI), vol. 3070, pp. 518–525. Springer, Heidelberg (2004)
13. Rutkowski, L., Cpałka, K.: A general approach to neuro - fuzzy systems. In: Proceedings of the 10th IEEE International Conference on Fuzzy Systems, Melbourne, December 2-5, vol. 3, pp. 1428–1431 (2001)
14. Rutkowski, L., Cpałka, K.: A neuro-fuzzy controller with a compromise fuzzy reasoning. Control and Cybernetics 31(2), 297–308 (2002)
15. Scherer, R.: Boosting Ensemble of Relational Neuro-fuzzy Systems. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Żurada, J.M. (eds.) ICAISC 2006. LNCS (LNAI), vol. 4029, pp. 306–313. Springer, Heidelberg (2006)
16. Scherer, R.: Neuro-fuzzy Systems with Relation Matrix. In: Rutkowski, L., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2010. LNCS (LNAI), vol. 6113, pp. 210–215. Springer, Heidelberg (2010)
17. Starczewski, J., Rutkowski, L.: Neuro-Fuzzy Systems of Type 2. In: Proc. 1st Int'l Conf. on Fuzzy Systems and Knowledge Discovery, Singapore, vol. 2, pp. 458–462 (2002)
18. Starczewski, J., Rutkowski, L.: Interval type 2 neuro-fuzzy systems based on interval consequents. In: Rutkowski, L., Kacprzyk, J. (eds.) Neural Networks and Soft Computing, pp. 570–577. Physica-Verlag, Springer-Verlag Company, Heidelberg, New York (2003)
19. Starczewski, J.T., Rutkowski, L.: Connectionist Structures of Type 2 Fuzzy Inference Systems. In: Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waśniewski, J. (eds.) PPAM 2001. LNCS, vol. 2328, pp. 634–642. Springer, Heidelberg (2002)