Dependable SIMD Architecture for Artificial Neural Networks

Jacek Mazurkiewicz
Wroclaw University of Technology, Poland
Jacek.Mazurkiewicz@pwr.wroc.pl

ABSTRACT

The paper proposed a new methodology for Hopfield and Kohonen neural networks based on systolic array structure. The method proposed is based on pipelined systolic arrays – an example of SIMD architecture. The discussion is realized based on operations which create the following steps of learning and retrieving algorithms. The data which are transferred among the calculation units are the second criterion of the problem. The results of discussion show that it is possible to create the universal structure to implement all algorithms related to Hopfield as well as for Kohonen neural network. The dependability features of the proposed methodology are focused to FTC – necessary calculations can be done using reduced number of elementary processors. The approach can be also easily adopted for other recurrent neural nets – as Hamming neural net for example.

Key Words: SIMD architecture, neural networks, dependability

1. Introduction

The paper is focused on the method of implementation of Hopfield and Kohonen neural network algorithms using pipelined systolic arrays – an example of SIMD architecture. The main assumption is about partial parallel realisation of learning algorithms as well as retrieving phase using the same processing structure. The proposed methodology creates the theoretical basis for hardware realisation of Hopfield and Kohonen neural networks and could easily adopted for other recurrent nets. The methodology is discussed based on the following assumptions:

- the outcome of algorithms realized true to proposed methodology is exactly the same like the outcome of classical neural network algorithms,
- the proposed methodology allows to create a universal structure both for learning algorithms and retrieving phase of defined type of the neural network,
- the systolic structure is realized using only digital elements, input and output data are represented in binary code,
- the number of neurons which create the net is unrestricted, but the maximum number of elementary processors can be limited – as a flexible reaction for failures.

2. Hopfield neural network algorithms

The binary Hopfield net has a single layer of processing elements, which are fully interconnected - each neuron is connected to every other unit. Each interconnection has an associated weight: w_{ji} is the weight to unit j from unit i. In Hopfield network, the weight w_{ij} and w_{ji} has the same value. Mathematical analysis has shown that when this equality is true, the network is able to converge. The inputs are assumed to take only two values: 1 and 0. The network has N nodes containing hard limiting nonlinearities. The output of node i is fed back to node j via connection weight w_{ij}**.**

2.1 Retrieving phase

During the retrieving algorithm each neuron performs the following two steps [8]:

- computes the coproduct:

$$\varphi_p(k+1) = \sum_{j=1}^{N} w_{pj} v_j(k) - \theta_p \qquad (1)$$

w_{pj} - weight related to feedback signal, $v_i(k)$ - feedback signal, Θ_p - bias
- updates the state:

$$v_p(k+1) = \begin{cases} 1 & for \quad \varphi_p(k+1) > 0 \\ v_p(k) & for \quad \varphi_p(k+1) = 0 \\ -1 & for \quad \varphi_p(k+1) < 0 \end{cases} \quad (2)$$

The process is repeated for the next iteration until convergence, which occurs when none of the elements changes state during any iteration. The initial an the end conditions for the iteration procedure require the following equations:

$$\forall_p \quad v_p(k+1) = v_p(k) = y_p \forall_p \quad v_p(0) = x_p \quad (3)$$

2.2 Hebbian learning algorithm

The training patterns are presented one by one in a fixed time interval. During this interval, each input data is communicated to its neighbour *N* times:

$$w_{ij} = \begin{cases} \frac{1}{N}\sum_{m=1}^{M} x_i^{(m)} x_j^{(m)} & for \quad i \neq j \\ 0 & for \quad i = j \end{cases} \quad (4)$$

M - number of training vectors

2.3 Delta-rule learning algorithm

The weights are calculated in recurrent way including all training patterns, according to the following matrix equation:

$$W = W + \frac{\eta}{N}\left[x^{(i)} - Wx^{(i)}\right]\left[x^{(i)}\right]^T \quad (5)$$

$\eta \in [0,7, 0,9]$ - learning rate, *N* - number of neurons, ***W*** - matrix of weights, *x* - input vector

The learning process stops when the next training step generates the changes of weights which are less then the established tolerance ε.

3. Systolic arrays for Hopfield neural network

Systolic arrays are prepared based on proper Data Dependence Graphs - directed graphs that specify the data dependencies of algorithms. In a Data Dependence Graph nodes represent computations and arcs specify the data dependencies between computations.

3.1 Idea of systolic array for Hebbian learning algorithm

Each node in Data Dependence Graph for Hebbian training algorithm multiplies two of corresponding input signals x_i and obtains this way the weight w_{ij} which is stored in local memory unit. So it realizes three operations. Each elementary processor is responsible for these three operations. The input signals x_i are passed to the nearest bottom neighbours and the neighbours on the right hand (Fig.1).

3.2 Idea of systolic array for Delta-rule learning algorithm

The Data Dependence Graph for Delta-rule training algorithm we can divide into two parts: *Relation Graph* G_r and *Value Graph* G_w. Each node which belongs to G_r multiplies a corresponding input signal x_i and weight value w_{ij}, then it subtracts the multiplication result from the input signal x_i. Each node in the G_w part of the Data Dependence Graph is responsible for three operations. During the first operation the node multiplies the corresponding result obtained at the end of the calculations related to the G_r part of the Data Dependence Graph and the input signal x_i. During the second operation each node multiplies the obtained values and the fraction: learning rate/number of neurons. At the end the values of weights are upgraded. This way the weights are obtained and next they are stored in local memory unit. The input signals x_i are passed to the nearest bottom neighbours [3]. Operations described by *Relation Graph* G_r and *Value Graph* G_w ought to be realized in sequence – so processor executes five basic operations. (Fig. 2).

3.3 Idea of systolic array for retrieving algorithm

Each node in Data Dependence Graph for retrieving algorithm multiplies the input signal x_i or feedback signals v_i and corresponding weight w_{ij} which is stored in local memory unit. The product of multiplication is passed to the nearest neighbour on the right hand (two basic operations). The φ_i nodes collect the partial

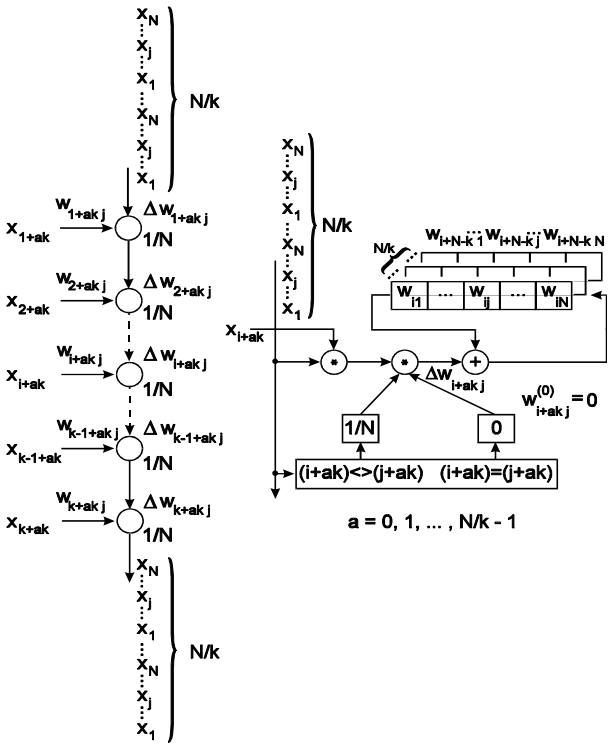

Fig. 1. Idea of systolic array for Hebbian learning algorithm

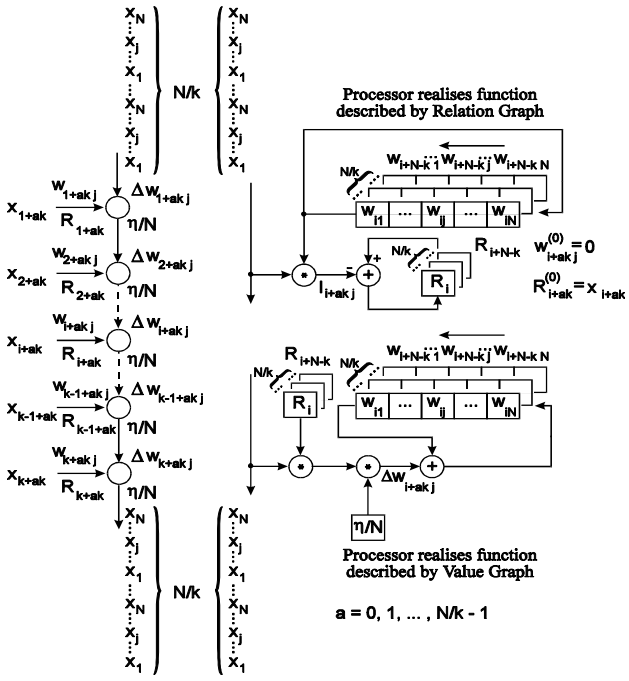

Fig. 2. Idea of systolic array for Delta-rule learning algorithm

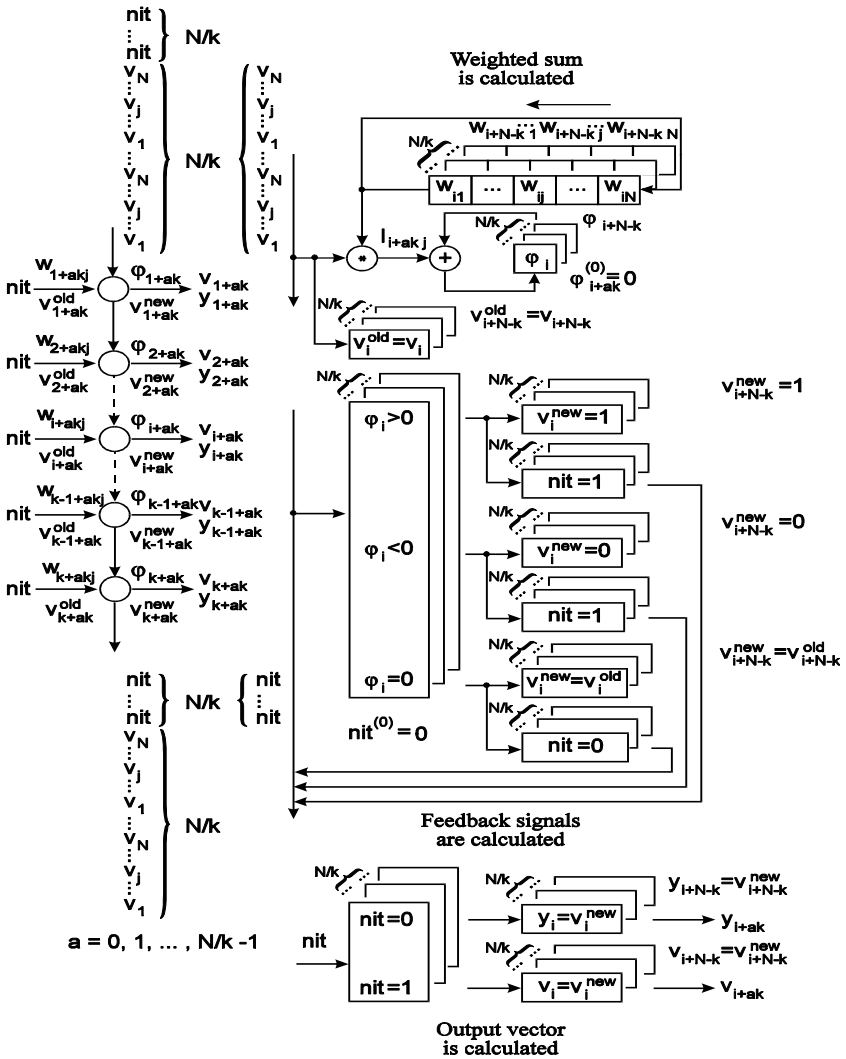

Fig. 3. Idea of systolic array for retrieving algorithm

products and calculate the global value of coproduct. The last nodes on the right are the comparators to check if the next iteration is necessary. (Fig. 3).

4. Efficiency of systolic arrays for Hopfield neural network

4.1 Computation time and Block period

This is time between starting the first computation and finishing the last computation of problem. Given a coprime schedule vector $\vec{s}$, the computation time equals [3], [11]:

$$T = \max_{\vec{p},\vec{q}\in L}\left\{\vec{s}^{\,T}\left(\vec{p}-\vec{q}\right)\right\}+1 \qquad (6)$$

L - is the index set of the nodes in the Data Dependence Graph

Block Period is time interval between the initiation of two successive blocks of operations [3], [11]. In the presented architectures for all algorithms the schedule vector is defined as: $\vec{s}=[1,1]$. For Hebbian training implementation - taking number of basic operations into account - we can calculate the computation time and block period as:

$$T_{systol} = 3N\left(\frac{N-1}{K}+1\right)\tau M$$

$$T_{block} = 3N\left(\frac{N-1}{K}+1\right)\tau \qquad (7)$$

In fact the Data Dependence Graph for this algorithm is combined by two independent structures of operations. The computation time for both parts of algorithm isn't the same because the number of basic operations is different, number of nodes and the topology of them is the same. Additionally the estimated computation time ought to be modified by number of iterations related to single training pattern:

$$T_{systol} = 5N\left(\frac{N-1}{K}+1\right)\tau M\beta$$

$$T_{block} = 5N\left(\frac{N-1}{K}+1\right)\tau\beta \qquad (8)$$

τ - processing time for elementary processor, M - number of training patterns, β - number of iterations for single training pattern, K - number of elementary processors.

The retrieving algorithm also requires multiple presentation of each pattern but single retrieving procedure doesn't require all patterns at the same time. The computation time we can describe using the following equations:

$$T_{systol} = 2N\left(\frac{N+1}{K}+1\right)\tau\beta$$

$$T_{block} = 2N\left(\frac{N+1}{K}+1\right)\tau \qquad (9)$$

In all equations we can find parameter denoted as K – the number of elementary processors. This way the FTC parameter of the structure can be discussed. The definition of SIMD architecture guarantees the unique construction and function of processors – so if we can observe the changes of efficiency parameters related to the number of used processors we can say a lot of the results of the failures. But we have to remember that using only single processor it is possible to realize the whole calculation process. Of course the efficiency parameters will be very poor, but the structure is still working.

4.2 Pipelining period

This is the time interval between two successive computations in a processor. As previously discussed, if both $\vec{d}$ and $\vec{s}$ are irreducible, then the pipelining period equals: $\alpha = \vec{s}^{\,T}\vec{d}$ (10)

The pipelining period is the same for all algorithms - equals: $\alpha = 1$ - is as short as possible [3].

4.3 Speed-up and Utilization rate

Lets define the speed-up factor as the ratio between the sequential computation time $\boldsymbol{T_{seq}}$ and the array computation time $\boldsymbol{T_{systol}}$ and the utilization rate as the ratio between the speed-up factor and the number of processors [3].

$$speed-up = \frac{T_{seq}}{T_{systol}}, \quad utilization\ rate = \frac{speed-up}{K} \qquad (11)$$

Sequential computation time for Hopfield neural network algorithms – taking number of neurons, number of weights and number of basic operations into account - equals:
- for Hebbian learning:

$$T_{seq} = 3N^2 \tau M \quad (12)$$

- for Delta-rule learning:

$$T_{seq} = 5N^2 \tau M \beta \quad (13)$$

- for retrieving algorithm:

$$T_{seq} = 2N(N+2)\tau\beta \quad (14)$$

Based on values of array computation time calculated in chapter 4.1. we can evaluate:
- for Hebbian learning:

$$speed-up = \frac{NK}{N-1+K}$$

$$utilization\ rate = \frac{N}{N-1+K} \quad (15)$$

- for Delta-rule learning:

$$speed-up = \frac{NK}{N-1+K}$$

$$utilization\ rate = \frac{N}{N-1+K} \quad (16)$$

- for retrieving algorithm:

$$speed-up = \frac{(N+2)K}{N+1+K}$$

$$utilization\ rate = \frac{N+2}{N+1+K} \quad (17)$$

This classical parameters calculated for presented architecture are also related to the number of active processors. We can model the speed-up and utilization rate in function of not-failed processors – we can control the efficiency in case of FTC features of proposed architecture.

5. Kohonen neural network algorithms

5.1 Learning algorithm

The learning algorithm is based on the Grossberg rule [5] [6]. All weights are modified according to the following equation: (18)

$$w_{lij}(k+1) = w_{lij}(k) + \eta(k)\Lambda(i^w, j^w, i, j)(x_l - w_{lij}(k))$$

k - iteration index, η - learning rate function, x_l - component of input learning vector, w_{lij} - weight associated with connection from component of input learning vector x_l and neuron indexed by (i,j), Λ - neighborhood function, (i^w, j^w) - indexes related to winner neuron, (i, j) - indexes related to single neuron from Kohonen map.
The learning rate η we assume as a linear decreasing function. Learning rate function is responsible for the number of iterations - it marks the end of learning process. The presented solution is based on the following description of the neighborhood function [1]:

$$\Lambda(i^w, j^w, i, j) = \begin{cases} 1 & for \quad r = 0 \\ \frac{\sin(ar)}{ar} & for \quad r \in \left(0, \frac{2\pi}{a}\right) \\ 0 & for \quad other\ values\ r \end{cases} \quad (19)$$

a - neighborhood parameter, r - distance from winner neuron to each single neuron from Kohonen map, calculated by indexes of neurons as follow: (20)

$$r = \sqrt{(i^w - i)^2 + (j^w - j)^2}$$

The learning procedure is iterative: weights are initialized by random values; position of winner neuron for each learning vector is calculated by ordinary Kohonen retrieving algorithm using random values of weights; weights are modified using Grossberg rule (18); the learning rate is modified, the neighborhood parameter a (19) is modified and if the learning rate is greater than zero weights are modified by the next learning vector, else the learning algorithm stops [7].

5.2 Retrieving algorithm

During the retrieving phase the Euclidean distance: the weights vector and the output vector is calculated. The winner neuron is characterized by the shortest distance [6] [7]. Each neuron from Kohonen map calculates the output value according to the classical weighted sum:

$$Out(i, j) = \sum_{l=0}^{N-1} x_l w_{lij} \quad (21)$$

$Out(i, j)$ - output value calculated by single neuron of Kohonen map indexed by (i, j)

6. Data Dependence Graphs for Kohonen neural network

6.1 Kohonen learning algorithm

For 1-D Kohonen map neurons are placed is single line, each neuron has two

neighbors, excluding neurons at the ends of line. For such topology there are $(N \times K)$ weights if we assumed N-element input vector and K neurons which create the Kohonen map. 1-D Kohonen map ought to be described by rectangular Data Dependence Graph (Fig. 6.). Each node of the graph is responsible for single weight calculation. using Grossberg rule (18) (Fig. 4.). The current value of the weight is stored in the local memory of each node. The node decreases the learning rate in automatic way. The size of the graph equals to the size of the weight matrix. Each node of the graph is loaded by two signals. The neighborhood function is calculated using sinus function. We propose to place the values of sinus in a table and store them in a local memory of each node. The neighborhood parameter a (19) is also stored in the local memory and is sequential reduced by negative counter.

6.2 Kohonen retrieving algorithm

1-D Kohonen map is described by rectangular Data Dependence Graph (Fig. 7.). Each node of the graph calculates the component of the weighted sum (21) (Fig. 5.). The necessary weight value is stored in a local memory of the node. The size of the graph equals to the size of the weight matrix.

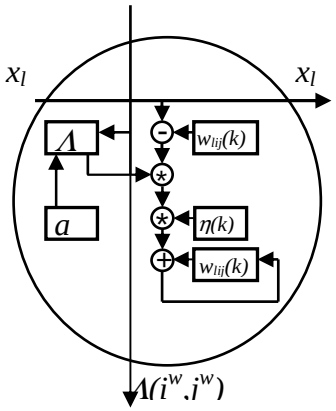

Fig. 4. Single node of Data Dependence Graph for learning algorithm

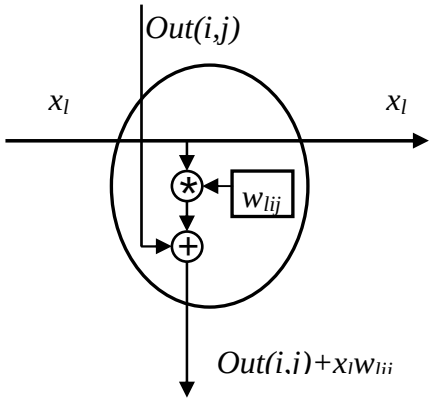

Fig. 5. Single node of Data Dependence Graph for retrieving algorithm

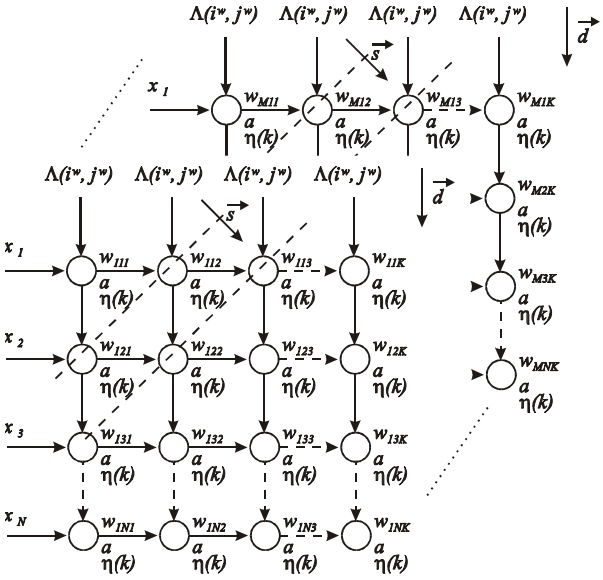

Fig. 6. Data Dependence Graph for learning algorithm of Kohonen map

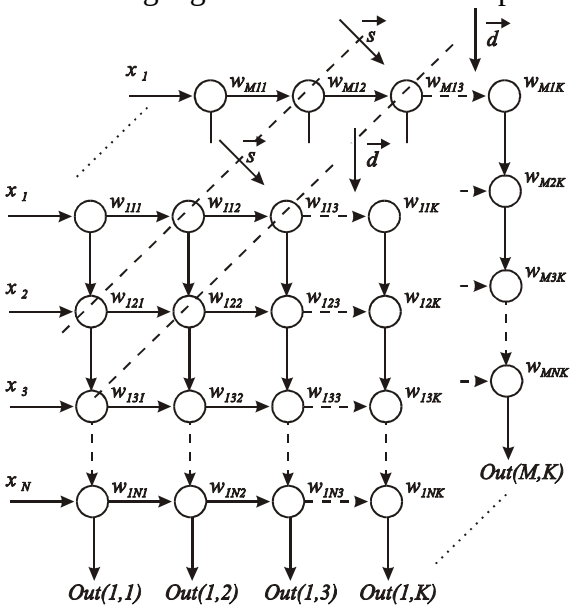

Fig. 7. Data Dependence Graph for retrieving algorithm of Kohonen map

7. Mapping Data Dependence Graphs onto systolic array

The Data Dependence Graphs for retrieving and learning algorithms are local and composed by the same number of nodes. The single neuron operations are described by the column of the graph [7]. Multi-dimensional Kohonen map is described by the set of 1-D Data Dependence Graphs (Fig. 6.) (Fig. 7.). It means that the slabs work in parallel [1] [7]. The graphs can be converted to an universal structure able to implement learning algorithm as well as retrieving algorithm using processors with switched functions (Fig. 8.) [3] [1]. The systolic arrays are the result of the linear projection of Data Dependence Graphs onto lattice of points, known as processor space. The elementary processor combines operations described by nodes taken from single vertical line of the graph [11].

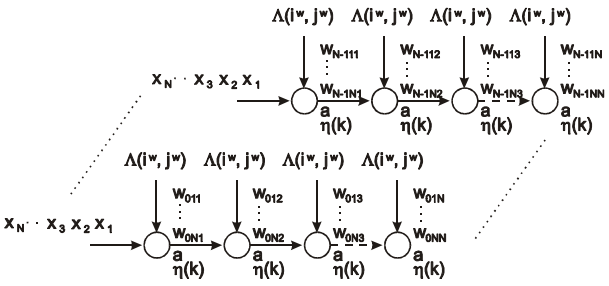

Fig. 8. Systolic array for learning algorithm of Kohonen neural network

8. Efficiency of approach proposed for Kohonen neural network

An efficiency of proposed approach is estimated using the algorithm proposed by Kung [3] and modified for MANTRA computer analysis [11]. The estimation is based on the dimensions and organization of the Data Dependence Graphs. A computation time for retrieving algorithm equals:

$$T = (N + K - 1)\tau \qquad (22)$$

τ - processing time for elementary processor.

The computation time for learning algorithm:

$$T = (N + K - 1)M\eta\tau \qquad (23)$$

M - number of learning vectors.

Speed-up and processor utilization rate are exactly the same for retrieving and learning algorithms - assuming possible sequential computation time: (24)

$$speed - up = \frac{NK}{N + K - 1} \qquad utilization\ rate = \frac{N}{N + K - 1}$$

9. Conclusion

The comparison of the same criteria for two methods of learning is the most interesting part. Computation Time - if we assume single presentation of each training vector - is less then two times longer for Delta-rule learning. Of course such assumption is true for Hebbian learning but isn't in general true for Delta-rule. Each next presentation of training set makes the Computation Time longer and the dependence is directly proportional.

It is very interesting we can observe exactly the same Speed-Up and Processor Utilization Rate both for Hebbian and Delta-rule learning procedures. The necessary time-period for calculation of Delta-rule procedure is longer than time-period related to Hebbian learning - but elementary processors' using is the same. The results of discussion show that it is possible to create the universal structure to implement all algorithms related to Hopfield neural network. This way there are no barriers to tune the Hopfield net to completely new tasks. The proposed methodology can be used as a basis for VLSI structures which implement Hopfield net or as a basis for set of general purpose processors – as transputers or DSP. Proposed methodology for Kohonen neural is based on classical and not modified algorithms related to Kohonen maps. It is possible to realize the obtained subtasks by software processes, but also using dedicated neuro-computers like MANTRA [11].

Table 1. Efficiency parameters for ring systolic structure related to Hopfield neural network algorithms – possible minimum and maximum values

	Learning		Retrieving phase
	Hebbian rule	Delta-rule	
Computation time T_{systol} (min) / (max)	$3(2N-1)\tau\ M$ / $3N^2\tau M$	$5(2N-1)\tau\ M\beta$ / $5N^2\tau M\beta$	$2(2N+1)\tau\ \beta$ / $2N(N+2)\tau\beta$
Speed-up (min) / (max)	$1 / \frac{N^2}{2N-1}$	$1 / \frac{N^2}{2N-1}$	$1 / \frac{(N+2)N}{2N+1}$
Utilization rate (min) / (max)	$\frac{N}{2N-1} / 1$	$\frac{N}{2N-1} / 1$	$\frac{N+2}{2N+1} / 1$

τ - processing time for elementary processor, M - number of training patterns,
β - number of iterations for single training pattern, N - number of neurons

The minimum values of the efficiency parameters are calculated for the proposed structure with single available (not-failed) processor. The maximum values are related to the optimal number of used processors. This way it is possible to observe the changes of the values as a function of ready to use elementary processors. We can model the influence of the decrease of the number of processors for the global efficiency of the system.

References

[1] K.V. Asari and C. Eswaran, *Systolic Array Implementation of Artificial Neural Networks*, Indian Institute of Technology, Madras, 1992.

[2] A. Ferrari and Y. H. Ng, "A Parallel Architecture for Neural Networks", Parallel Computing'91, Elsevier Science Publishers B. V., 1992, pp. 283 - 290.

[3] S.Y. Kung, *Digital Neural Networks*, PTR Prentice Hall, 1993

[4] J Mazurkiewicz, "Feedforward Neural Network Simulation Based on Systolic Array Approach", NETSS 2004, ACTA MOSIS No. 94, MARQ., Ostrava, 2004, pp. 17 - 22.

[5] J. Mazurkiewicz, "Kohonen Neural Network Learning Algorithm Simulation Based on Systolic Array Approach", MOSIS'05 Conference Modelling and Simulation of Systems, ACTA MOSIS No. 102, Ostrava, 2005, pp. 202 - 207.

[6] J. Mazurkiewicz, "Systolic Realization of Kohonen Neural Network", Artificial Neural Networks: Formal Models and Their Applications – ICANN 2005, LNCS 3697, Springer-Verlag Berlin Heidelberg, 2005, pp. 1015 - 1020.

[7] J. Mazurkiewicz, "Systolic Realisation of Self-Organising Neural Networks", ICSC 2003, 2003, pp. 116 - 123.

[8] J. Mazurkiewicz, "Systolic Simulation of Hamming Neural Network", Advances in Soft Computing, Physica-Verlag Heidelberg 2003, A Springer-Verlag Company, 2003, pp. 867 - 872.

[9] N. Petkov, *Systolic Parallel Processing*, North-Holland, 1993.

[10] S.G. Shiva, *Pipelined and Parallel Computer Architectures*, Harper Collins Publishers, 1996.

[11] D. Zhang, *Parallel VLSI Neural System Design*, Springer-Verlag, 1999.