11

SIMD Architecture Approach to Artificial Neural Networks Realisation

Jacek Mazurkiewicz
Institute of Computer Engineering, Control and Robotics
Wroclaw University of Technology
ul. Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, POLAND

1. Introduction

A new methodology for artificial neural networks implementation based on SIMD architecture is described. The method proposed is based on pipelined systolic arrays. The partial parallel realisation of learning and retrieving algorithms using unified processing structure is assumed. The discussion is realized based on operations which create the following steps of weight tuning and answer generation 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, Kohonen and Hamming neural networks. Theoretical foundations for synthesis of digital devices related to neural network algorithms are the main goal. The number of neurons is unlimited, but necessary calculations can be done using reduced number of elementary processors. This way the dependability features of the proposed methodology are focused to Fault Tolerant Computing approach. The evaluation of efficiency measures related to the proposed structures is the final step of described work.
The elaboration can be divided into following stages:
- accommodation of analysed algorithms to partial parallel realisation in systolic array;
- synthesis of constructional unified systolic structures and data flow description for analysed algorithms;
- proposals of functions for processing elements;
- evaluation of proposed solutions.

Proposed systolic arrays are based on the set of assumptions:
- results obtained as the output of structures are convergent to theoretical description of proper algorithms of artificial neural nets;
- proposed structures are realised based on digital elements only;
- number of neurons limits maximum number of elementary processors used in structure.

The proposed solutions are characterised by very universal approach with partial parallel information processing which reduces calculation time in significant way. The work can be used as base point both for dedicated circuits to implement artificial neural networks and for structures which realise the net by ready-to-use elements with point-to-point communication rules as PLD for example.

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 (Mazurkiewicz, 2003b) (Ferrari & Ng, 1992):
- computes the coproduct:

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

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

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

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: (Mazurkiewicz, 2004)

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

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:

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

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:

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

$\eta \in [0,7,\ 0,9]$ - learning rate, N - number of neurons, $\boldsymbol{W}$ - 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. (Ferrari & Ng, 1992) (Mazurkiewicz, 2004)

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).

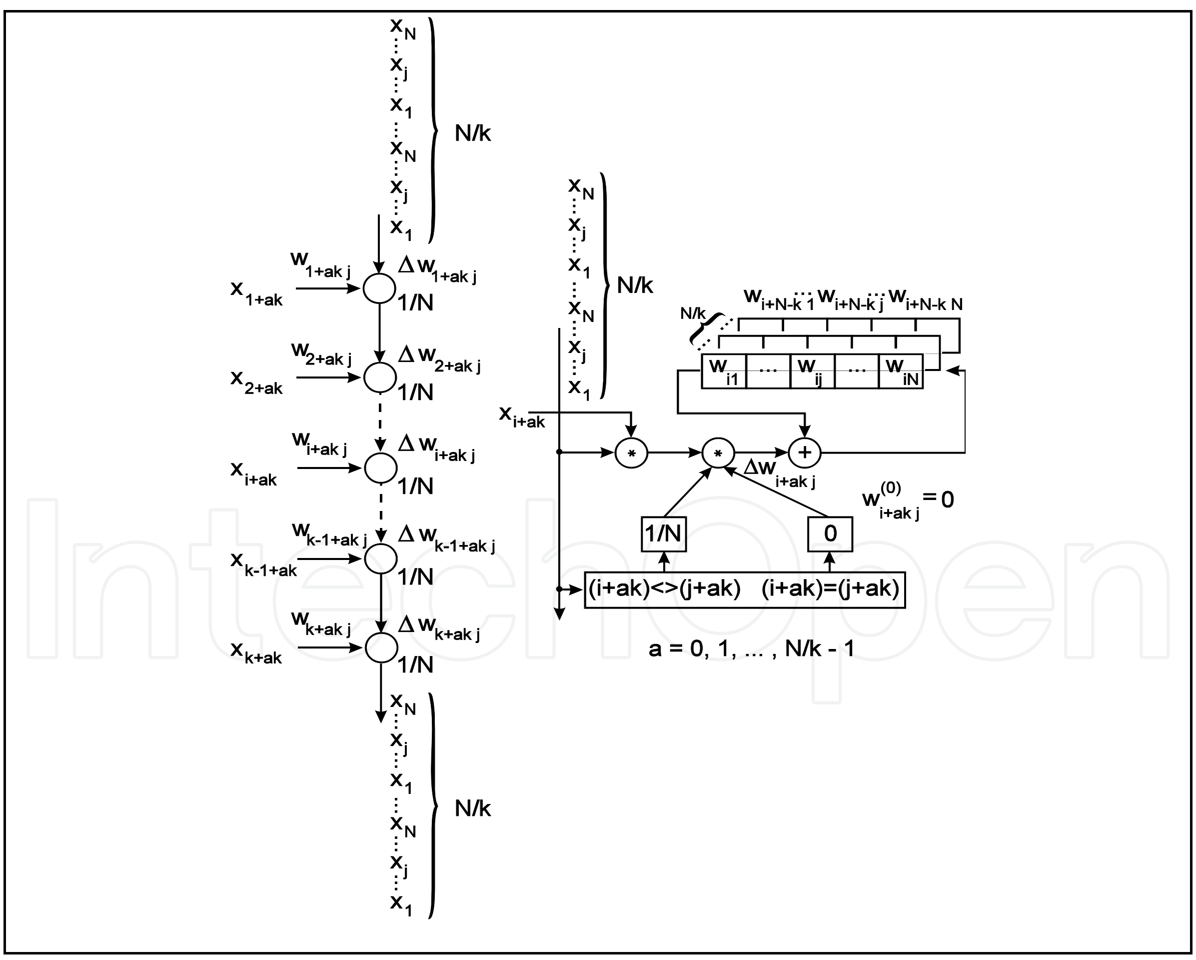


Fig. 1. Idea of systolic array for Hebbian learning algorithm

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 (Kung, 1993). 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).

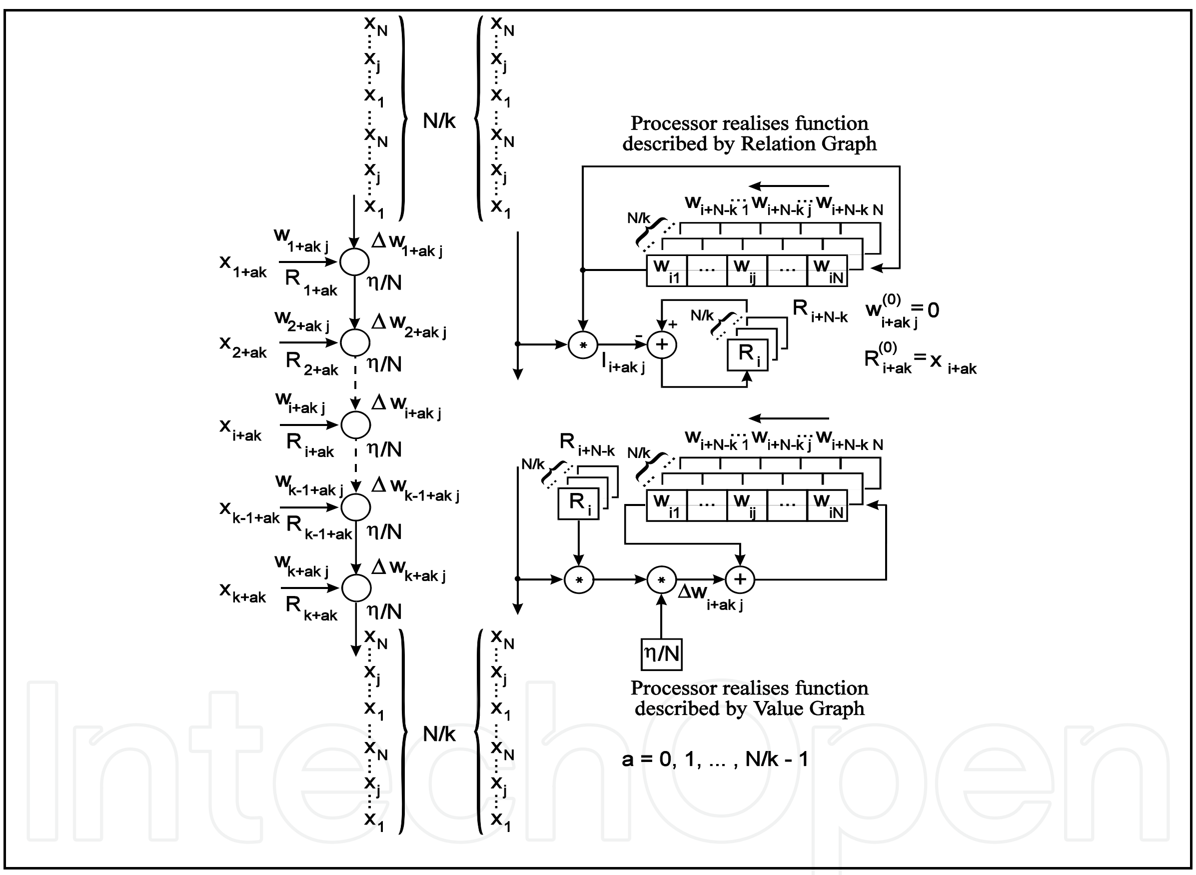


Fig. 2. Idea of systolic array for Delta-rule learning algorithm

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 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).

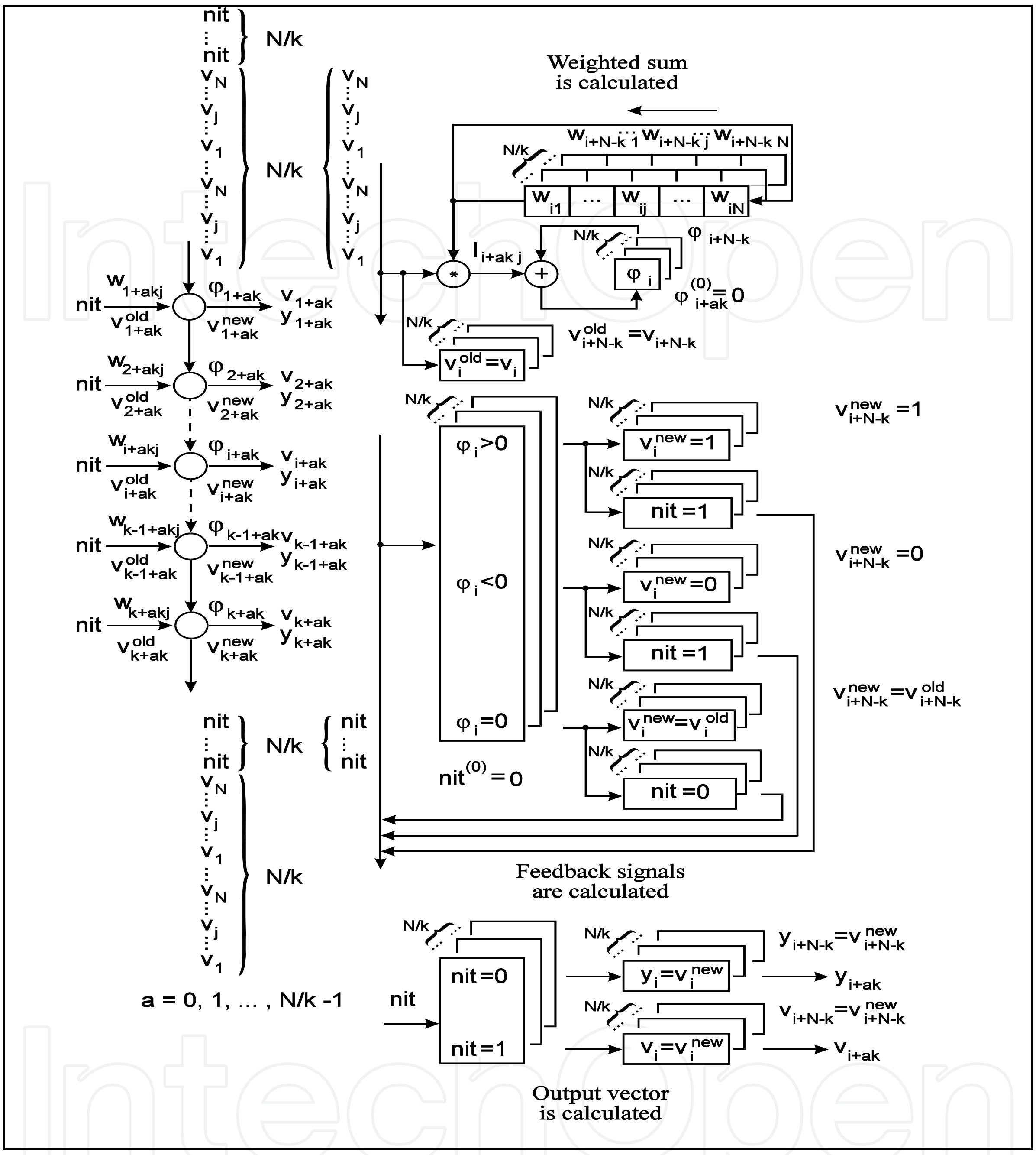


Fig. 3. Idea of systolic array for retrieving algorithm

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 (Kung, 1993), (Zhang, 1999):

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

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 (Kung, 1993), (Zhang, 1999). 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:

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

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:

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

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:

$$T_{systol}=2N\left(\frac{N+1}{K}+1\right)\tau\beta \quad T_{block}=2N\left(\frac{N+1}{K}+1\right)\tau \tag{9}$$

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 to:

$$\alpha=\vec{s}^{\,T}\vec{d} \tag{10}$$

The pipelining period is the same for all algorithms – equals to: $\alpha = 1$ - is as short as possible (Kung, 1993) (Ferrari & Ng, 1992) (Shiva, 1996).

4.3 Speed-up and Utilization rate

Lets define the speed-up factor as the ratio between the sequential computation time T_{seq} and the array computation time T_{systol} and the utilization rate as the ratio between the speed-up factor and the number of processors (Kung, 1993).

$$speed-up = \frac{T_{seq}}{T_{systol}} \quad utilization\ rate = \frac{speed-up}{K} \tag{11}$$

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:

$$T_{seq} = 3N^2 \tau M \tag{12}$$

- for Delta-rule learning:

$$T_{seq} = 5N^2 \tau M \beta \tag{13}$$

- for retrieving algorithm:

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

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

$$speed-up = \frac{NK}{N-1+K} \quad utilization\ rate = \frac{N}{N-1+K} \tag{15}$$

- for Delta-rule learning:

$$speed-up = \frac{NK}{N-1+K} \quad utilization\ rate = \frac{N}{N-1+K} \tag{16}$$

- for retrieving algorithm:

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

The 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 (Mazurkiewicz, 2005a) (Mazurkiewicz, 2005b). All weights are modified according to the following equation:

$$w_{lij}(k+1) = w_{lij}(k) + \eta(k)\Lambda(i^w, j^w, i, j)(x_l - w_{lij}(k)) \tag{18}$$

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 (Asari & Eswaran, 1992):

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

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

$$r = \sqrt{\left(i^w - i\right)^2 + \left(j^w - j\right)^2} \tag{20}$$

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 (Mazurkiewicz, 2003a).

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 (Mazurkiewicz, 2005b) (Mazurkiewicz, 2003a). Each neuron from Kohonen map calculates the output value according to the classical weighted sum:

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

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.

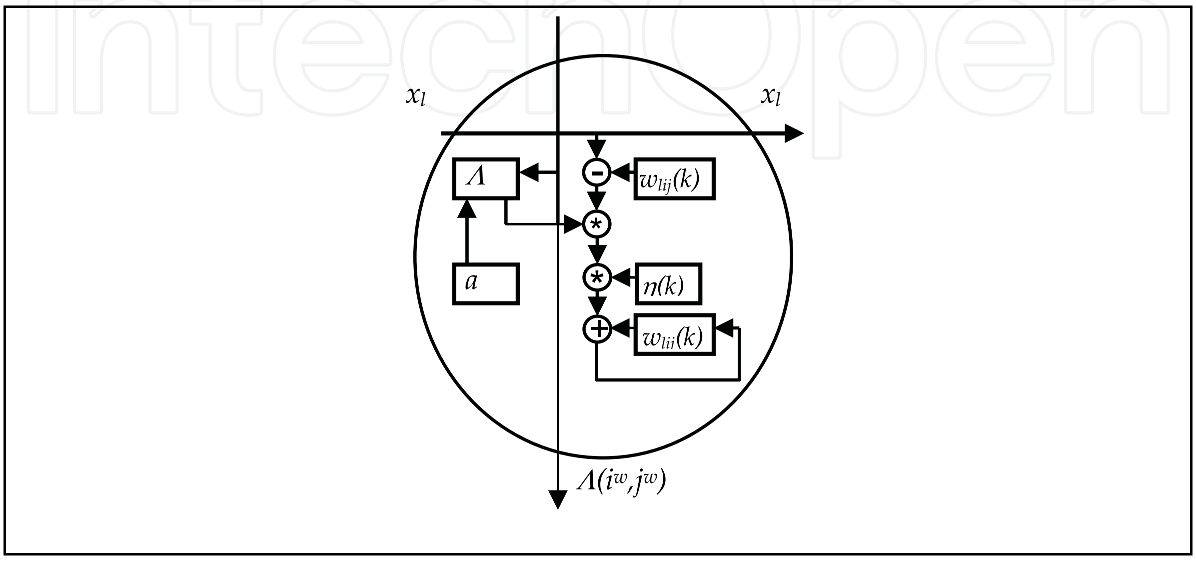


Fig. 4. Single node of Data Dependence Graph for learning algorithm

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.

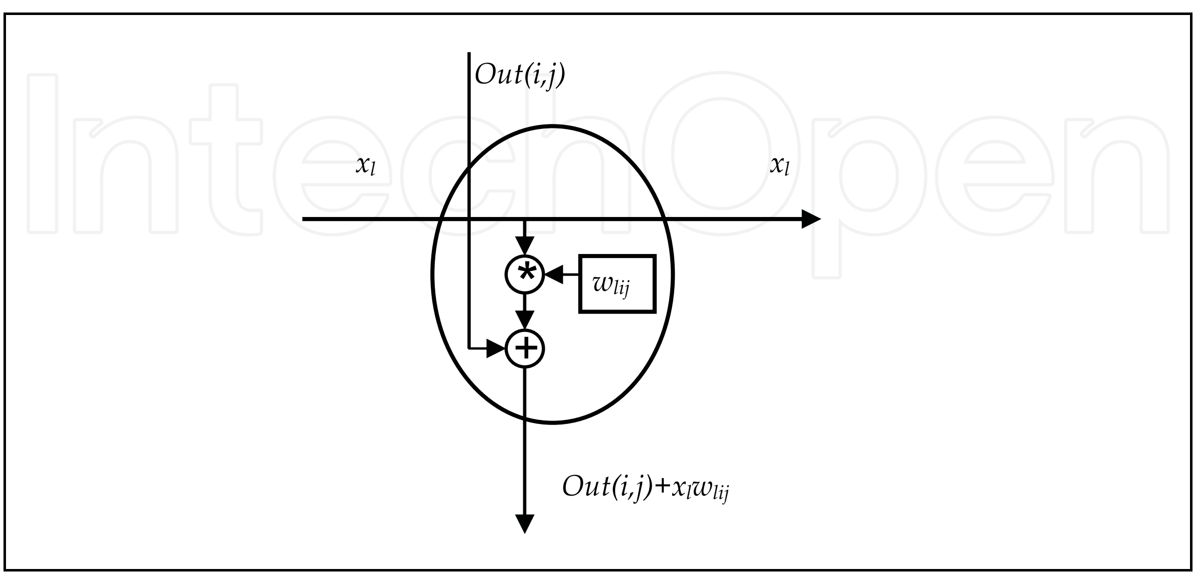


Fig. 5. Single node of Data Dependence Graph for retrieving algorithm

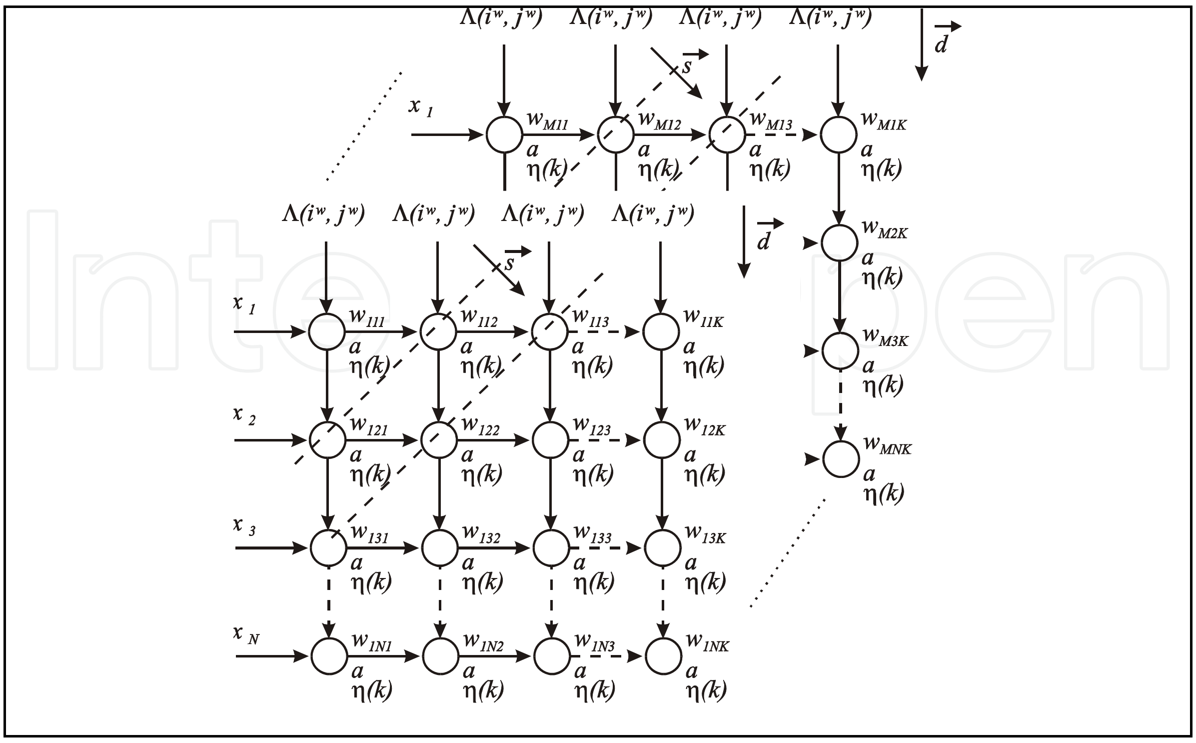


Fig. 6. Data Dependence Graph for learning algorithm of Kohonen map

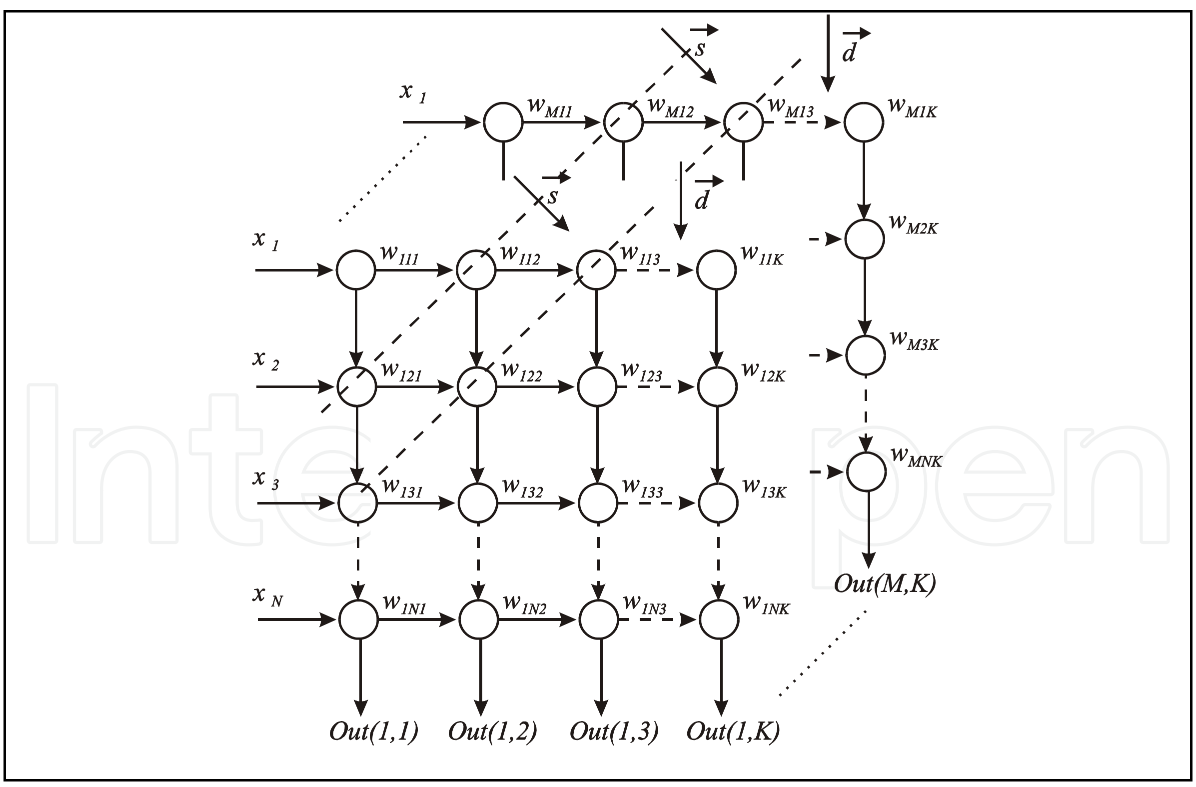


Fig. 7. Data Dependence Graph for retrieving algorithm of Kohonen map

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 (Mazurkiewicz, 2003a). 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 (Asari & Eswaran, 1992) (Mazurkiewicz, 2003a) (Mazurkiewicz, 2003b). 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.) (Kung, 1993) (Asari & Eswaran, 1992). 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 (Zhang, 1999) (Petkov, 1993).

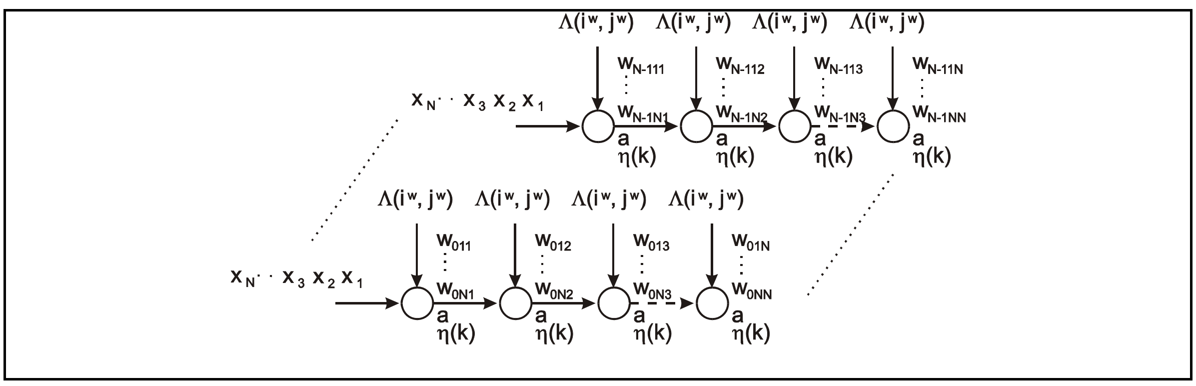


Fig. 8. Systolic array for learning algorithm of Kohonen neural network

8. Efficiency of approach proposed for Kohonen neural network

An efficiency of proposed approach is estimated using the algorithm proposed by Kung (Kung, 1993) and modified for MANTRA computer analysis (Zhang, 1999) (Petkov, 1993). The estimation is based on the dimensions and organization of the Data Dependence Graphs. A computation time for retrieving algorithm equals to:

$$T = (N + K - 1)\tau \tag{22}$$

τ - processing time for elementary processor. The computation time for learning algorithm:

$$T = (N + K - 1)M\eta\tau \tag{23}$$

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:

$$speed - up = \frac{NK}{N + K - 1} \qquad utilization\ rate = \frac{N}{N + K - 1} \tag{24}$$

9. Hamming net description

The Hamming net implements the optimum minimum error classifier when bit errors are random and independent. The performance of the Hamming net is proved in problems such as character recognition, recognition of random patterns and bibliographic retrieval. The feedforward Hamming net maximum likelihood classifier for binary inputs corrupted by noise is presented in (Fig. 9). The Lower Sub Net calculates N minus the Hamming distance to M exemplar patterns, where N is the number of elements in one pattern. The upper sub net selects that node with the maximum output. All nodes use threshold logic nonlinearities, where it is assumed that the outputs of these nonlinearities never saturate. Thresholds and weights in the Maxnet are fixed. All thresholds are set to zero and weights from each node to itself are 1. Weights between nodes are inhibitory with a value of $-\varepsilon$, where $\varepsilon < 1/M$ (Ferrari & Ng, 1992). The connection weights and offsets of the lower sub net are assigned as (Kung, 1993):

$$w_{ji} = \frac{x_i^j}{2} \qquad \Theta_j = \frac{N}{2} \tag{25}$$

for $0 \le i \le N - 1$ and $0 \le j \le M - 1$, Θ_j - the threshold in that node, w_{ji} - the connection weight from input i to node j in the lower sub net. The connection weights in the upper sub net (Maxnet) are fixed as:

$$w_{lk} = \begin{cases} 1 & if \quad k = 1 \\ -\varepsilon & if \quad k \neq 1 \end{cases} \tag{26}$$

for $0 \le l, k \le M$ and $\varepsilon < 1/M$, w_{lk}- the weight from node k to node l in the upper sub net and all thresholds in this max net are kept zero. The outputs of the lower sub net are obtained with unknown input pattern as:

$$\mu_j = \sum_{i=0}^{N-1} w_{ji} x_i - \Theta_j \tag{27}$$

for $0 \le i \le N - 1$ and $0 \le j \le M - 1$. The inputs of the Maxnet are initialized with threshold logic nonlinearities:

$$y_j(0) = f_t(\mu_j) \tag{28}$$

for $0 \le j \le M - 1$. The Maxnet does the maximization:

$$y_j(t+1) = f_t\left(y_j(t) - \varepsilon \sum_{k \neq j} y_k(t)\right) \tag{29}$$

for $0 \le j, k \le M$. This process is repeated until convergence. (Kung, 1993)

10. Data Dependence Graph for Hamming neural network

Each node in Data Dependence Graph responsible for Lower Sub Net multiplies the input signal x_i and corresponding weight w_{ji} which is stored in local memory unit. The product of multiplication is passed to the nearest neighbor on the right hand. The input signals x_i are passed to the nearest bottom neighbors. The μ_i nodes collect the partial products and calculate the global value of coproduct (27). The last *N-1* columns of nodes on the right are responsible for Maxnet. They transmit step by step the partial results among the neurons which belong to the Maxnet. and they calculate the output value (29) (Mazurkiewicz, 2003b).

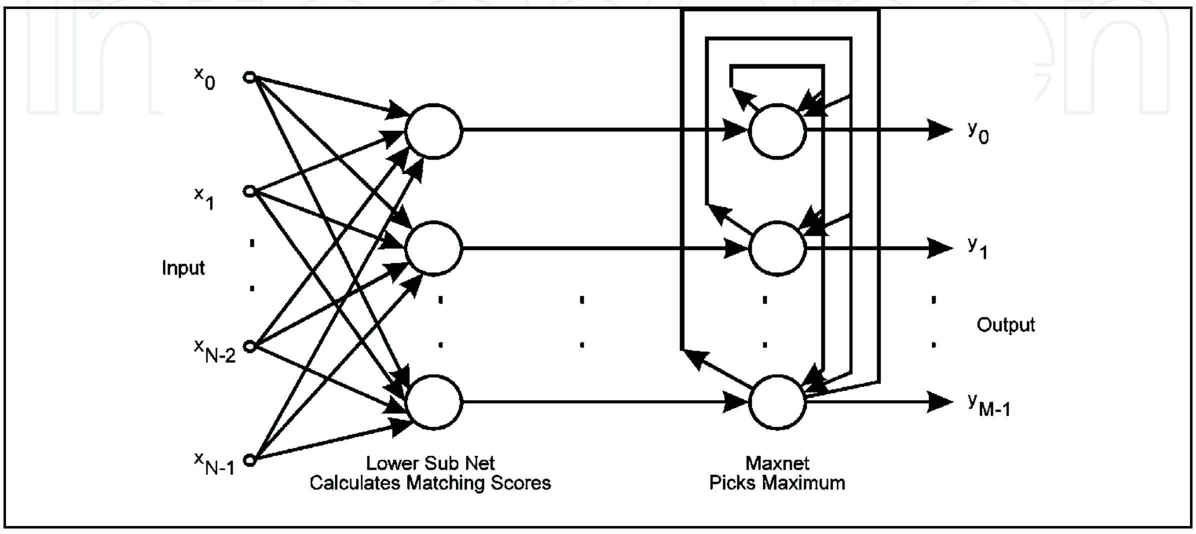


Fig. 9. Hamming neural network

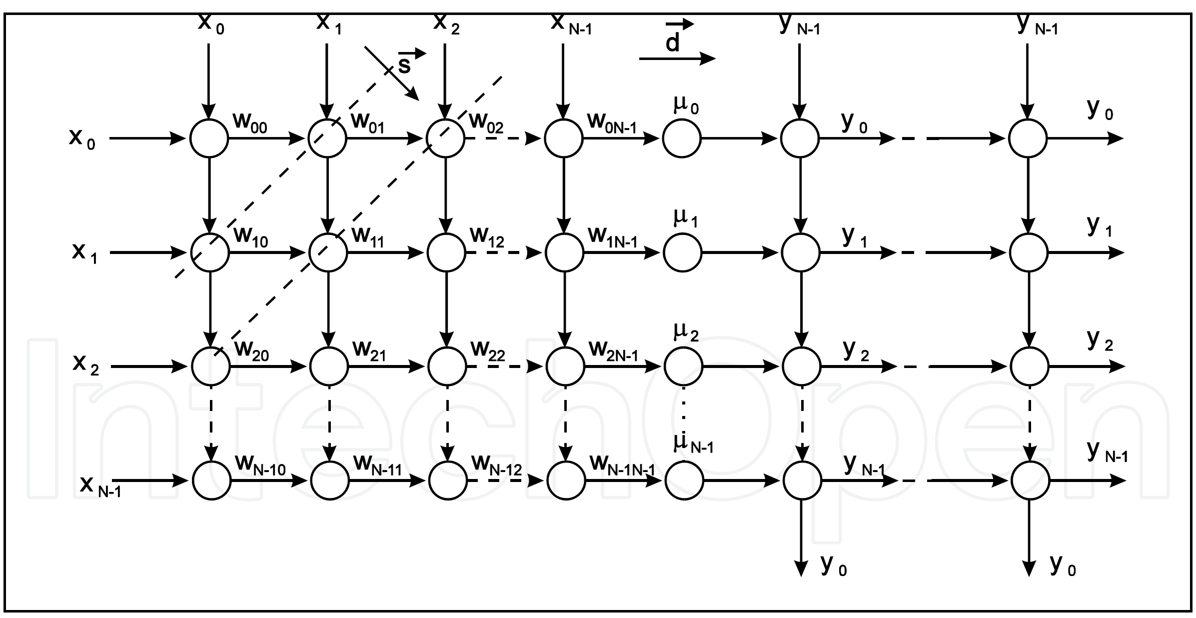


Fig. 10. Data Dependence Graph for Hamming neural network

11. Systolic realisation of Hamming neural network

The Hamming net algorithm is modified for convenience to implement in a parallel systolic architecture. For that the following changes are made: the inputs are considered binary - 0 and 1 - instead of 1 and -1, the weights are made 1 and 0 instead of +0,5 and –0,5 corresponding to a pattern. This will avoid the subtraction of the threshold $\Theta_j = 0,5 * N$ after the summation. The weights are obtained by the following algorithm.

$$w_{ji} = x_i^j \quad for \quad 0 \le i \le N-1 \quad and \quad 0 \le j \le M-1 \tag{30}$$

The activation function of the lower sub net is updated as follows:

$$y_j(k) = \begin{cases} y_j(k-1)-1 & if \quad (x_i \oplus w_{ji}) = 1 \\ y_j(k-1) & if \quad (x_i \oplus w_{ji}) = 0 \end{cases} \tag{31}$$

$y_j = y_j\ (N)$ for $0 \le i \le N - 1$ and $0 \le j \le M - 1$ and $0 \le k \le N$. It is possible to constitute the maximization network with a layer of up counters, whose outputs are changed with the following algorithm:

$$y_j(k) = \begin{cases} y_j(k-1)+1 & if \quad AND\{O_i\} = 0 \\ y_j(k-1) & if \quad AND\{O_i\} = 1 \end{cases} \tag{32}$$

O_i - the output vector, obtained from the MSBs - sign bits - of these counters

11.1 Elementary processor realization of Hamming neural net

(Fig. 11) presents the parallel implementation of the algorithm for Hamming net using only digital circuits. The input vector corrupted by noise is applied to the input register when *ILP - Input Load Pulse* - is present. The down counter is cleared by this signal. The input data is applied sequentially to each processing element with the *Shift Control Pulse SP'*, which is derived from *SP* and *LCP - Load Counter Pulse*: *not(LCP) & SP*. At the same time the weight register is also shifted once by the same signal. The content of the down counter is updated according to the (8) with the presence of the pulse *CP'*, which is derived from *CP* and *ILP*: *not(ILP) & CP*.

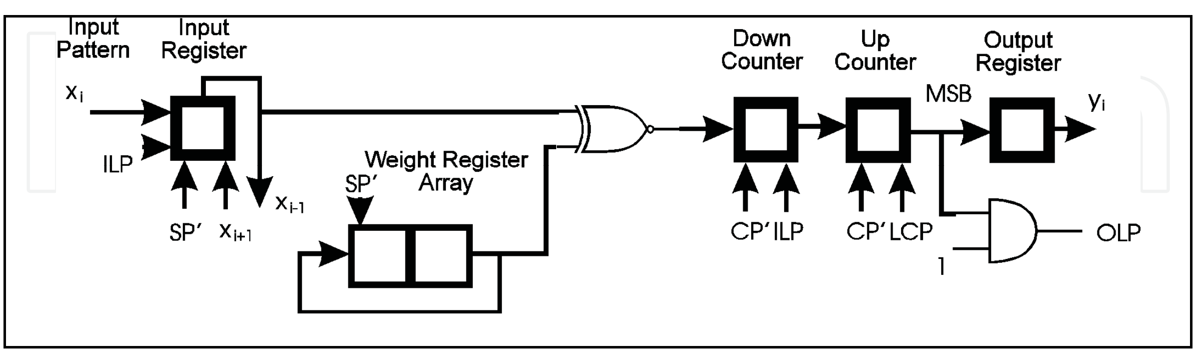


Fig. 11. Single slab implementation of Hamming network

The data obtained in each register after *N* steps will be negative. They are transferred to the up counters when *LCP* signal is present. *LCP* signal is applied only after *N* shift operations of the input register and weight registers are performed. The MSBs of the register contents will be the sign bits, which will be 1 for all the data at this stage. The MSBs give the output vector O_j for $0 \le j \le M - 1$. The up counters constitute the Maxnet, which will count up when *CP'* signal is present. The operation of the up counter is stopped by a control signal *OLP* - *Output Load Pulse* - derived from the output vector O_j. When any one of the sign bits of the registers becomes 0 (its content becomes positive) *OLP* stops the operation of the up counter and thereby the neuron with the largest activation function wins the competition. The output pattern is the inverted output of the vector O_j which is obtained when *OLP* signal is present.

12. Efficiency of Hamming Net Systolic Implementation

12.1 Computation time

The first part of Data Dependence Graph (Fig. 10) is responsible for Lower Sub Net calculation. This part is spread on *N* elements in horizontal point of view. The second part guarantees the data communication within Maxnet. Here we have *N-1* elements in horizontal axis. The total computation time for retrieving algorithm taking into account the computation time for Lower Sub Net and the computation time for Maxnet can be calculated as (τ - processing time for elementary processor):

$$T_{systol} = (4N - 3)\tau \tag{33}$$

12.2 Processor utilization rate

If we assume that all elementary processors need the same time-period to proceed their calculations the speed-up and utilization rate factors can be estimated as:

$$speed - up = \frac{(2N-1)N}{4N-3} \quad utilization\ rate = \frac{(2N-1)}{4N-3} \tag{34}$$

13. 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 (Shiva, 1996). 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 (Zhang, 1999) (Petkov, 1993).

The Hamming net algorithm is modified suitably for the systolic implementation. The systolic algorithm for the computation of the activation values of the lower sub net is developed. The upper sub net which acts as a maximization network is constructed by using simple counters. The main advantage of the proposed architecture is that it can be easily extended to larger networks.

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.

	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)	$\frac{N}{2N-1} \,/\, 1$	$\frac{N}{2N-1} \,/\, 1$	$\frac{N+2}{2N+1} \,/\, 1$

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

14. References

Asari, K.V. & Eswaran, C. (1992). *Systolic Array Implementation of Artificial Neural Networks*, Indian Institute of Technology, Madras

Ferrari, A. & Ng, Y.H. (1992). A Parallel Architecture for Neural Networks, *Parallel Computing'91*, pp. 283–290, Elsevier Science Publishers B. V.

Kung, S.Y. (1993). *Digital Neural Networks*, PTR Prentice Hall

Mazurkiewicz, J. (2004). Feedforward Neural Network Simulation Based on Systolic Array Approach, *NETSS 2004*, pp. 17-22, ACTA MOSIS No. 94, MARQ., Ostrava

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

Mazurkiewicz, J. (2005b). Systolic Realization of Kohonen Neural Network, *Artificial Neural Networks: Formal Models and Their Applications – ICANN 2005*, pp. 1015-1020, LNCS 3697, Springer-Verlag Berlin Heidelberg

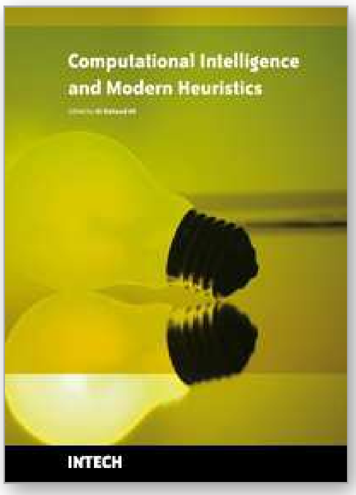
Mazurkiewicz, J. (2003a) Systolic Realisation of Self-Organising Neural Networks, *ICSC 2003*, pp. 116-123, EPI Kunovice

Mazurkiewicz, J. (2003b). Systolic Simulation of Hamming Neural Network, *Advances in Soft Computing*, pp. 867-872, Physica-Verlag Heidelberg, A Springer-Verlag Company

Petkov, N. (1993). *Systolic Parallel Processing*, North-Holland

Shiva, S.G. (1996). *Pipelined and Parallel Computer Architectures*, Harper Collins Publishers

Zhang, D. (1999). *Parallel VLSI Neural System Design*, Springer-Verlag



Computational Intelligence and Modern Heuristics
Edited by Al-Dahoud Ali

ISBN 978-953-7619-28-2
Hard cover, 348 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010

The chapters of this book are collected mainly from the best selected papers that have been published in the 4th International conference on Information Technology ICIT 2009, that has been held in Al-Zaytoonah University, Jordan in the period 3-5/6/2009. The other chapters have been collected as related works to the topics of the book.