

## CHAPTER 2

### SYSTOLIC ARRAY METHODOLOGY FOR A NEURAL MODEL TO SOLVE THE MIXTURE PROBLEM

R.M. Pérez, P. Martínez, A. Plaza and P.L. Aguilar

*Departamento de Informática, Universidad de Extremadura  
Campus Universitario s/n 10071 Cáceres, Spain  
E-mail: rosapere,pablomar,aplaza,paguilar@unex.es*

In this paper we present a robust method for determining and quantifying components in a composite spectrum; we assume that the patterns of the individual spectra are known in advance. The proposed method is supported by a linear recurrent neural network based on the Hopfield model (HRNN). The HRNN has very important characteristics in the implementation of low-complexity VLSI structures, since only multiplying and adding operations are required for an inversion process. We propose a systolic array for its implementation. In order to describe the systolic structure, we use a methodology based on the dependence graph. We suggest a sequential algorithm that verifies the unique assignment rule and the locality of the data dependences. The proposed systolic structure is divided into two parts, one rectangular array for implementing the weight matrix determination, and a linear array for obtaining the Threshold vector. Both structures are used for realising the iterative process of the Neural Network. These structures allow us to reduce the computational cost from  $o(N^2)$  to  $o(N)$  so that the weight matrix is obtained, and  $o(N^3)$  to  $o(N)$  in order to realise the iterative process.

#### 1. Introduction

In most signal processing applications such as Remote Sensing, Environmental Control, and so forth, it is necessary to determine and quantify the components in a composite spectrum, which is obtained from a given mixture of elements. In hyperspectral analysis, this issue is known as the determination of end-members and abundances, while in general spectroscopy, it is called the *Mixture Problem*.

Formally, this development may be considered as follows: Under the assumption that we know the spectra of  $K$  elements (*Basic References*), we must determine the unknown composition of a *cocktail* of the mentioned elements by using a radiation spectrum obtained from this mixture.

The main goal of this paper is to explore the possibility of applying a Neural Networks Methodology to achieve a reliable, robust and efficient solution to this problem, based on the inherent parallelism of neural networks.

Adams et al. [1] studied this problem in the context of Surface Mineralogy Prospection, and Lawton [2] proposed conventional digital algorithms for its solution. These approaches are fairly slow because serial computation is implied.

A method based on Optical Neural Networks was presented by Barnard and Cassasent [3]. The main drawback in this approach is the lack of uniqueness for the proposed solution.

The possibility of using the Multiple Regression Theory, thus enabling an optimal solution in terms of uniqueness, has been developed by Díaz et al. [4]. This approach is based on the use of the Pseudo-Inverse Matrix, supported by a Linear Associative Memory, built by implementing Pyle's algorithm [5].

The general solution method proposed here is based on the Hopfield Recurrent Neural Network (HRNN). It is a flexible, efficient and robust approach aimed to solve the problem. The Gradient Method for error reduction is applied to ensure the convergence of the algorithm. The use of this model is fully justified, since the spectrum formation in the *Mixture Problem* is essentially a linear process [6].

One of the most remarkable properties deduced from the algorithmic structure of this method, is related to the possibility of implementing low-complexity VLSI structures. These are suitable to support the algorithm, since multiplications and additions are only required for programming an inversion process [7]. These operations can be implemented through basic processor arrays. This approach matches the VLSI system design principle, according to which, modular, pipelined, and parallel architectures are exploited, and the communication hardware is reduced to local interconnections in most cases [8].

A typical example of VLSI parallel/pipelined architectures is observed in systolic arrays. These introduce many advantages; for instance, the exploitation of pipelining is very natural in networks set up regularly and locally. This saves the cost associated with communication. A second benefit is the good balance

between computation and communication, which is critical to the effectiveness of array computing [8].

In the following sections, a method developed from the HRNN-based (Hopfield model) linear recurrent neural network is explained. Such a development aims to solve the mixture problem and its implementation by means of systolic arrays.

## 2. Algorithmic Method

In order to describe the algorithmic method proposed, we must previously consider that a Composite Spectrum  $\mathbf{y}$  can be seen as an  $N$ -dimensional vector. This vector is built by sorting the emission intensities associated with each energy channel vs. the channel number, where  $N$  is the total number of energy channels:

$$\mathbf{y} = [y_1, y_2, \dots, y_N]^T \quad y_n \geq 0 \quad 1 \leq n \leq N \quad (1)$$

Here,  $y_n$  is the intensity measured as the number of photons whose energy is comprised in the  $n$ -th energy channel interval.

As a result, a *Reference Spectrum* is a spectrum of the same nature, but produced by an *Individual Source*. We denote these spectra as  $\mathbf{r}_k$ , with  $1 \leq k \leq K$ . The set of  $K$  reference vectors is named the *Reference Set*, and it must be evaluated in advance. For the sake of compactness, the *Reference Set* is referred to as the *Reference Matrix*  $\mathbf{R}$  composed by the reference column vectors:

$$\mathbf{R} = [\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_K] \quad (2)$$

In a general sense, the set of Composite Spectra is the range of all possible spectra that may be produced by a linear combination of all elements belonging to the *Reference Set*. When the *Reference Set* is composed by linearly independent  $K$  vectors, the result is a  $K$ -dimensional *Vector Space*, integrated by all the vectors  $\mathbf{y}$ , and explained as:

$$\mathbf{y} = \mathbf{R}\mathbf{c} = \sum_{i=1}^K c_i \mathbf{r}_i \quad (3)$$

where  $\mathbf{c}$  is the *Contribution Vector*, which is defined as:

$$\mathbf{c} = [c_1, c_2, \dots, c_K]^T \quad c_k \geq 0 \quad 1 \leq k \leq K \quad (4)$$

and where every contribution  $c_i$  is a function of the relative intensities of the Composite and Reference Spectra.

Our goal is to estimate  $\mathbf{c}$ , as we assume that  $\mathbf{R}$  and  $\mathbf{y}$  are known. For the mixture described by an estimation of  $\mathbf{c}$ , called  $\mathbf{c}'$ , the difference between the measured spectral vector  $\mathbf{y}$  and its re-constructed version  $\mathbf{y}'$ :

$$\boldsymbol{\varepsilon} = \mathbf{y} - \mathbf{y}' = \mathbf{y} - \sum_{i=1}^K \mathbf{c}'_i \mathbf{r}_i \quad (5)$$

is called *Estimation Error*, which gives us a measure of how well the estimation of  $\mathbf{c}$  has been accomplished. This error is exploited to optimise the estimation process by means of a *Least Mean Square (LMS)* minimization procedure. In relation to  $\mathbf{c}$ , this is laid out to minimise the *Measure Function*  $F(\mathbf{c})$ , being defined as:

$$F(\mathbf{c}) = \|\boldsymbol{\varepsilon}\|^2 = \|\mathbf{y} - \mathbf{R}\mathbf{c}\|^2 \quad (6)$$

To solve this problem, we apply an iterative process, supported by the *Linear Hopfield Minimization Procedure*, that is basically a progressive refinement of the *Contribution Vector*:

$$\mathbf{c}'(t+1) = \mathbf{c}'(t) + \Delta \mathbf{c}'(t) = \mathbf{c}'(t) + \lambda \mathbf{R}^T [\mathbf{y} - \mathbf{R}\mathbf{c}'(t)] \quad (7)$$

where we have used an estimation of the *Gradient of*  $F(\mathbf{c})$  in relation to  $\mathbf{c}$ , exposed as:

$$\nabla_{\mathbf{c}} F(\mathbf{c}) = -2\mathbf{R}^T [\mathbf{y} - \mathbf{R}\mathbf{c}] \quad (8)$$

In compact notation, we can formulate (7) as:

$$\begin{aligned} \mathbf{c}(t+1) &= \lambda \mathbf{q} + \mathbf{P}\mathbf{c}(t) & \text{with} & & \mathbf{q} &= \mathbf{R}^T \mathbf{y} & \mathbf{P} &= \mathbf{I} - \lambda \mathbf{R}^T \mathbf{R} \\ \text{where } 1 \leq i, j \leq K & & p_{ij} &= -\lambda \sum_{p=1}^N r_{pi} r_{pj} & & & i \neq j \end{aligned} \quad (9)$$

$$p_{ii} = 1 - \lambda \sum_{p=1}^N r_{pi} r_{pi} \quad q_i = \sum_{p=1}^N r_{pi} y_p$$

in which  $\lambda$  is a parameter dependent on the trace of  $\mathbf{R}^T \mathbf{R}$ . This controls the speed of convergence<sup>6</sup>, whereas  $p_{ij}$  denotes the weight from the  $i$ -th node to the  $j$ -th

node. This method simply requires multiplying and adding operations to solve the *Mixture Problem*. The HRNN structure proposed according to this algorithm can be seen in Fig. 1 and Fig. 2.

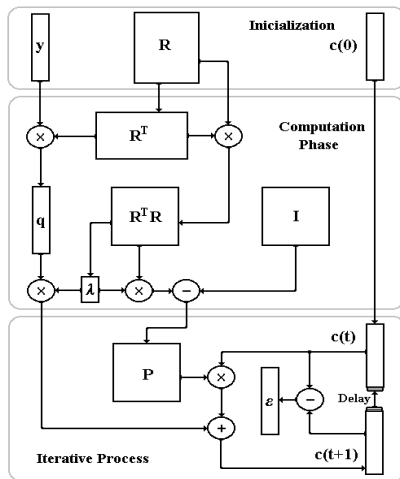


Fig. 1. HRNN Algorithm.

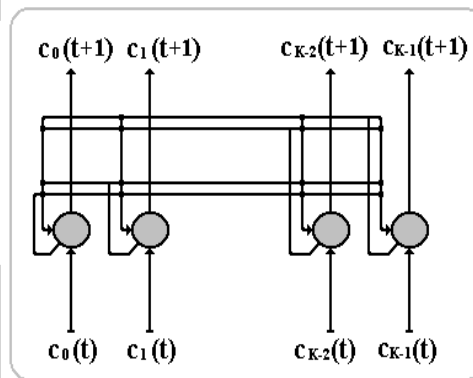


Fig. 2. HRNN Iterative Process.

The HRNN presents significant features, such as a reasonable computational cost, robustness versus increasing levels of noise in the composite spectrum, and an inherent ability to prevail in quite uneven mixtures, as in the high disproportion of 1:1000 [6], [7], [9], [10].

### 3. VLSI Implementation by Systolic Arrays

VLSI technology has made one thing clear: Simple regular interconnections lead to cheap implementations and high densities, and this implies both high performance and low overhead for support components. Due to these traits, an interesting design is that of parallel algorithms that have simple and regular data flows. It is also noteworthy to use pipelining as a general method for implementing these algorithms in hardware [11].

The proposed algorithmic method in the section previous can be implemented in terms of matrix-matrix and matrix-vector multiplications. These

tasks can be classified as compute-bound computations, thereby requiring high computing speed. For applications involving matrix calculations, low-cost architectures that are specialized in terms of algorithm and high-performance, such as the systolic array processors, have been conceived [12].

In the systolic background, VLSI devices are formed by arrays of interconnected processing cells having a high degree of modularity. Each processor operates on a string of data that flows regularly over the network [13].

In order to derive the systolic arrays for the proposed algorithm, we follow a systematic mapping methodology based on dependence graphs. The mapping procedure is summarised into key tasks [13]: To pipeline all the variables in the algorithm; to find the set of data dependence vectors; to identify a valid transformation for the data dependence vectors and the index set; and to map the algorithm onto the hardware.

In the following parts of this section, previous concepts are applied to obtain systolic array structures for the HRNN. We consider three phases: First, the determination of the rectangular systolic array for the weight matrix configuration, to which the reference spectra set  $\mathbf{R}$  data is applied; secondly, a linear systolic array is set to achieve the threshold of the neural network, for which the transpose of the matrix  $\mathbf{R}$  and the composition vector  $\mathbf{y}$  are developed; finally, the structure for the iterative process is designed by exploiting the systolic system described in the previous phases.

### 3.1. Weight Matrix Determination

The HRNN displays a hebbian learning, since the reference set is developed in order to come up with the weight matrix. Proceeding from (9), the following sequential algorithm is devised to compute the values associated with the net connections:

```

For i=1 to K do
  For j=1 to K do
     $P_y = 0;$     $P_u = 1$ 
    For m=1 to N do
       $P_y = P_y + (-\lambda)RT_{jm} R_{mj}$ 
    Endfor
  Endfor
Endfor

```

where  $RT$  is the matrix whose rows have the reference spectra, i.e.,  $RT_{ij} = R_{ji}$ .

Table 1 provides the set index for this algorithm. Each index element is shown as a three-fold tuple  $(i,j,m)$ . Note that for both index elements  $(i,j,1)$  and  $(i,j,2)$ , the same value of  $P(i,j)$  is used; in other words, the value  $P(i,j)$  can be piped in the  $m$  direction. Similarly, the values  $RT(i,m)$  and  $R(m,j)$  can be piped in the  $j$  and  $i$  directions respectively.

Table 1: Index set of the weight matrix algorithm

$i,j,m$	$P(i,j)=$	$P(i,j)$	$RT(i,m)$	$*R(m,j)$
1,1,1	1,1	1,1	1,1	1,1
1,1,2	1,1	1,1	1,2	2,1
...	...	...	...	...
1,1,N	1,1	1,1	1,N	N,1
1,2,1	1,2	1,2	1,1	1,2
...	...	...	...	...
1,2,N	1,2	1,2	1,N	N,2
...	...	...	...	...
1,K,1	1,K	1,K	1,1	1,K
...	...	...	...	...
1,K,N	1,K	1,K	1,N	N,K
2,1,1	2,1	2,1	2,1	2,1
...	...	...	...	...
2,K,N	2,K	2,K	2,N	N,K
...	...	...	...	...
K,1,1	K,1	K,1	K,1	N,1
...	...	...	...	...
K,K,N	K,K	K,K	K,N	N,K

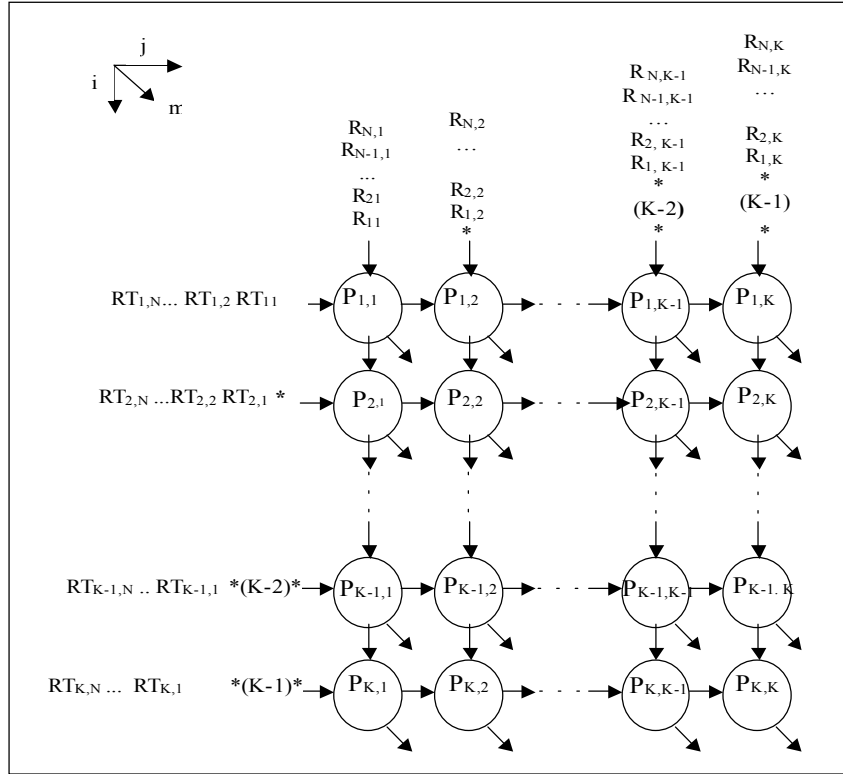
The data set of dependence vectors can be identified by equating indexes of all the possible pairs of generated and exploited variables

$d_p=(001)^T$	$d_{RT}=(010)^T$	$d_R=(100)^T$
---------------	------------------	---------------

The dependence matrix can be expressed as:

$$D = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Figure 3 shows the dependence graph for  $R^T R$ .



**Fig. 3** Dependence graph for the obtaining the weight matrix  $P$ .

This graph is mapped onto the  $m$  direction, for which a transformation  $T$  is sought; this change must be a bijection and monotonic increase of the form:

$$T = \begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{pmatrix} = \begin{pmatrix} \pi \\ S \end{pmatrix}$$

The nodes on an equi-temporal hyperplane cannot be projected to the same PE [12]. This must be done as a valid order, and as a reduction of the turnaround. For such aims, we chose  $\pi$  as:

$$\begin{aligned} \pi * \phi_P = (t_{11} \ t_{12} \ t_{13}) * (0 \ 0 \ 1)^T > 0 \quad \rightarrow \quad t_{13} > 0 \quad \rightarrow \\ t_{13} = 1 \\ \pi * \phi_{RT} = (t_{11} \ t_{12} \ t_{13}) * (0 \ 1 \ 0)^T > 0 \quad \rightarrow \quad t_{12} > 0 \quad \rightarrow \end{aligned}$$

$$\pi = (111)$$

Our choice of  $S$  determines the interconnection of the processors. It is a projection in the  $m$  direction because all nodes in this direction will correspond to the same PE. As a result,

$$S = \begin{pmatrix} 010 \\ 100 \end{pmatrix}$$

Table 2 displays the mapping between original and transformed indexes (by  $T$ ). In this Table, columns  $\pi I$  and  $SxI$  indicate the time and the cell in which the computation indexed by  $I$  will be processed.

Table 2 Mapping the original index set onto the transformed index set

$T$	$i,j,p$	$T(I)$	$P(i,j)=$	$P(i,j)$	$-\lambda RT(i,p)$	$*R(p,j)$	$\pi I$	$SI$
T	1,1,1	3,1,1	1,1	1,1	1,1	1,1	3	1,1
T	1,1,2	4,1,1	1,1	1,1	1,2	2,1	4	1,1
...	...	...	...	...	...	...	...	...
T	1,1,N	N+1,1,1	1,1	1,1	1,N	N,1	N+2	1,1
T	1,2,1	4,2,1	1,2	1,2	1,1	1,2	4	2,1
...	...	...	...	...	...	...	...	...
T	1,2,N	N+3,2,1	1,2	1,2	1,N	N,2	N+3	2,1
...	...	...	...	...	...	...	...	...
T	1,K,1	K+2,K,1	1,K	1,K	1,1	1,K	K+2	K,1
...	...	...	...	...	...	...	...	...
T	1,K,N	N+K+1,K,1	1,K	1,K	1,N	N,K	N+K+1	K,1
T	2,1,1	4,1,2	2,1	2,1	2,1	2,1	4	1,2
...	...	...	...	...	...	...	...	...
T	2,K,N	N+K+2,K,2	2,K	2,K	2,N	N,K	N+K+1	K,2
...	...	...	...	...	...	...	...	...
T	K,1,1	K+2,1,K	K,1	K,1	K,1	N,1	K+2	1,K
...	...	...	...	...	...	...	...	...
T	K,K,N	N+2K,K,K	K,K	K,K	K,N	N,K	2K+N	K,K

Therefore, to obtain the weight matrix,  $K \times K$  PEs are needed, while required computation time is  $2K+N$ . The interconnection between those processors is defined as:

$$Sd_i = \begin{bmatrix} x \\ y \end{bmatrix}$$

where  $x$  and  $y$  refer to the movement of the variable along the directions  $j$  and  $i$  respectively. For our case,

$$S \cdot d_p = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad S \cdot d_{RT} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$S \cdot d_R = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

thus mean that variables  $\mathbf{P}$  do not travel in any direction, and are updated in terms of time; and that  $\mathbf{RT}$  and  $\mathbf{R}$  move with a speed of one grid per unit time along the direction  $j$  and  $i$  respectively.

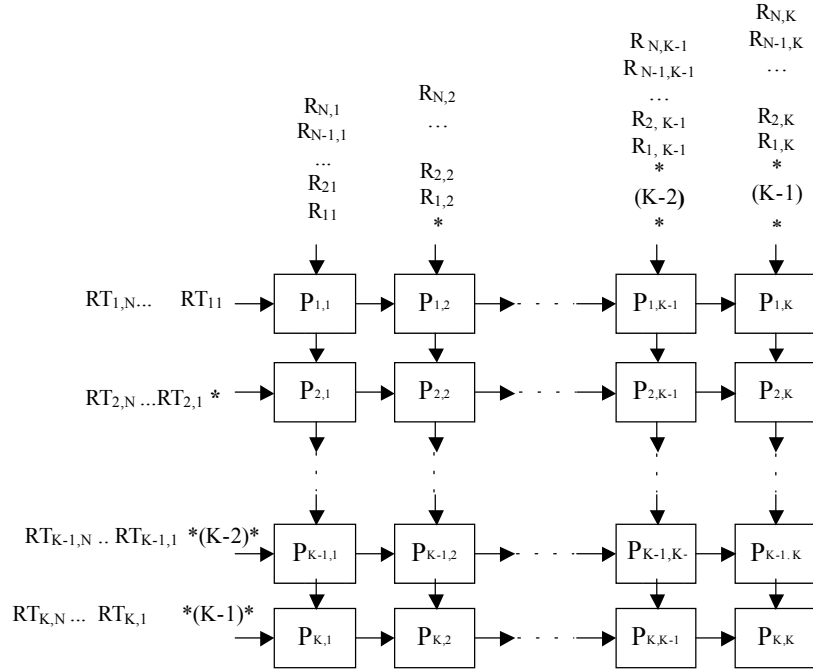


Fig 4. Systolic array for the weight matrix  $\mathbf{P}$ .

Figure 4 reproduces interconnections between PEs. As can be observed, this architecture has a rectangular array design. The elements of matrix  $RT$  are fed from the left side of the array, while matrix  $R$  is pipelined into the array from the upper edge.

At time 3 (first clock cycle of computing)  $R_{11}$  and  $RT_{11}$  enter  $PE_{11}$ , which contains the variable  $P_{11}$  -- at previous times all  $P_{ij}$  are initialised as  $P_{ii}=1$ ;  $P_{ij}=0$  and the value of the learning parameter  $\lambda$  are stored in each PE. These perform multiplying and adding operations, and they accumulate the partial results, as is shown in Figure 5. The process continues until all the values are computed.

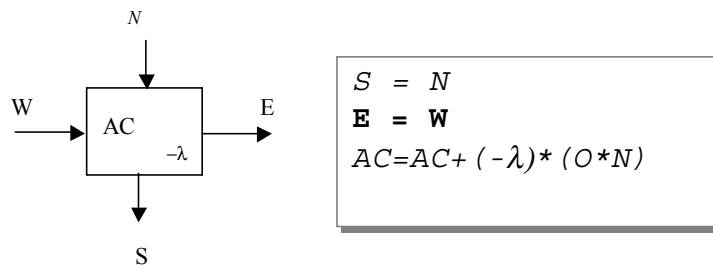


Fig. 5 Operations performed by Fig. 4 PEs

### 3.2. Threshold Vector Computation

The aim is to solve the mixture problem, as shown in (9) above. Thus, for each mixture spectrum  $y$ , we have a threshold vector with  $K$  components,  $q$ . For computing this vector, a matrix-vector multiplication is required; the following algorithm presents the needed conditions:

```

For i=1 to K to do
  For j=1 to N to do
     $q_i = q_i + \lambda RT_{ij} y_j$ 
  End For
End For

```

where  $\lambda$  is the convergence parameter.

Table 3 supplies the set index for this algorithm. In this Table, each index element is shown as a two tuple  $(i,j)$ . Based on this, data dependence can be deduced in a similar manner to the previous case.

Table 3: Index set of threshold vector

$i,j$	$q(i)=$	$q(i)$	$+RT(i,j)$	$*y(i)$
1,1	1	1	1,1	1
1,1	1	1	1,2	2
...	...	...	...	...
1,N	1	1	1,N	N
2,1	2	2	2,1	1
...	...	...	...	...
2,	2	2	2,N	N
...	...	...	...	...
K,1	K	K	K,1	1
...	...	...	...	...
K,N	K	K	K,N	N

Thus, the data dependence vectors are:

$$\begin{array}{l} q(i,j)=q(i,j-1) \\ y(i,j)=y(i-1,j) \end{array} \quad \begin{array}{l} D_q=(01)^T \\ D_y=(10)^T \end{array}$$

So, the matrix dependence is:

$$D = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

If the transformation T is selected as:

$$T = \begin{pmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{pmatrix} = \begin{pmatrix} \pi \\ S \end{pmatrix} \quad T = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Then:

$$S \cdot d_q = (1 \ 0) \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = (0) \quad S \cdot d_y = (1 \ 0) \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = (1)$$

It means that variable  $q$  does not travel in any direction and is updated in time; and that variable  $y$  moves to the right with a speed of one per time unit. Table 4 presents the mapping between original and transformed indexes.

Figure 6 reproduces the interconnections between the PEs. As can be examined, this architecture has a linear array design. The elements of vector  $y$  are fed from the left side of the array and transmitted among neighbouring PEs.

The arithmetic processing capabilities of each PE should include multiplication and accumulation operations (see Figure 7).

Table 4: Mapping the original index set onto the transformed index set

$I=(i,j)$	$T(I)$	$q(i)=$	$q(i)$	$+RT(i,j)$	$*y(j)$	$\pi I$	$SI$
1,1	2,1	1	1	1,1	1	2	1
1,2	3,1	1	1	1,2	2	3	1
...	...	...	...	...	...	...	...
1,N	N+1,1	1	1	1,N	N	N+1	1
2,1	3,2	2	2	2,1	1	3	2
...	...	...	...	...	...	...	...
2,N	N+2,2	2	2	2,N	N	N+2	2
...	...	...	...	...	...	...	...
K,1	K+1,K	K	K	K,1	1	K+1	K
...	...	...	...	...	...	...	...
K,N	K+N,K	K	K	K,N	N	K+N	K

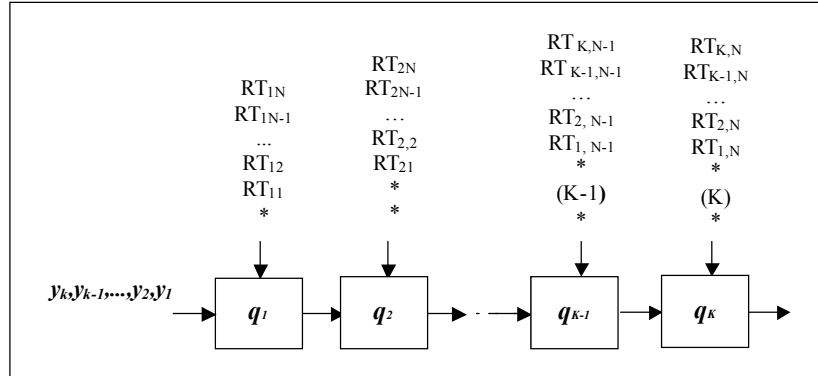


Fig. 6 Required interconnections among PE's for the systolic array in the computation of the threshold vector  $q$ .

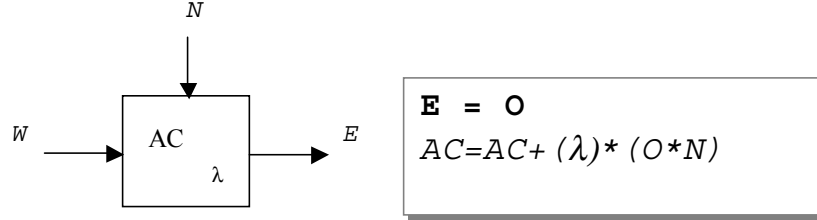


Fig. 7 Operations performed by PEs in Figure 6.

### 3.3. Contribution Achievement

The previous sub-sections have shown how the weight matrix and threshold vector can be computed by means of systolic structures. Next, we should be able to solve the mixture problem with the iterative process (Figure 2) given for (9), and expressed as:

$$c(t+1) = \left( \sum_{j=1}^N p_{ij} c_j(t) + q_i \right) \quad (10)$$

where  $c(t+1)$  and  $c(t)$  represent two consecutive network states. Therefore, when the neural model converges, the contribution of components in a mixture spectrum is obtained and each  $c_i$  determines the proportion of the correspondent reference spectrum  $r_i$  in the mixture. The algorithm describing the process given in (10) can be expressed as:

```

For t=1 to tmax do
  For i=1 to K do
    c'i(t)=qi
    For j=1 to K do
      c'i(t)=c'i(t)+Pijcj(t)
    Endfor
    ci(t+1)=c'i(t)
  Endfor
Endfor

```

where  $c'$  represents the net state at the  $(t+1)$  time.

Table 5 represents the set index for this algorithm. In this Table, each index element is shown as a three tuple (t,i,j). Based on this, data dependence can be located in a similar manner to the previous case.

Table 5: Index for the iterative process

t,i,j	c'(i)=	c'(i)	+P(i,j)	*c(j)
1,1,1	1	1	1,1	1
1,1,2	1	1	1,2	2
...	...	...	...	...
1,1,K	1	1	1,K	K
...	...	...	...	...
1,K,1	K	K	K,1	1
...	...	...	...	...
1,K,K	K	K	K,K	K
2,1,1	1	1	1,1	1
2,1,2	1	1	1,2	2
...	...	...	...	...
2,K,1	K	K	K,1	1
...	...	...	...	...
2,K,K	K	K	K,K	K
t <sub>max</sub> ,1,1	1	1	1,1	1
t <sub>max</sub> ,1,2	1	1	1,2	2
...	...	...	...	...
t <sub>max</sub> ,1,K	K	K	1,K	K
t <sub>max</sub> ,2,1	2	2	2,1	1
...	...	...	...	...
t <sub>max</sub> ,2,K	2	2	2,K	K
...	...	...	...	...
t <sub>max</sub> ,K,1	K	K	K,1	1
...	...	...	...	...
t <sub>max</sub> ,K,K	K	K	K,K	K

According to Table 5, data dependence can be obtained as:

In order to account for the previous derivations, the  $\pi$  and  $S$  values are chosen as:

$c'(t,i,j)=c'(t,i,j-1)$	$d_c = (0 \ 0 \ 1)^T$
$c(t,i,j)=c(t,i-1,j)$	$d_t = (0 \ 1 \ 0)^T$
$P(t,i,j)=P(t-1,i,j)$	$d_p = (1 \ 0 \ 0)^T$

$$\pi = (111); \quad S = \begin{pmatrix} 010 \\ 001 \end{pmatrix} \Rightarrow T = \begin{pmatrix} 111 \\ 010 \\ 001 \end{pmatrix}$$

Through this selection, we can apply the designed rectangular array, which calculated and stored one weight in each PE. According to the selected matrix  $T$ , the variable:  $P$  does not travel in any direction,  $c$  and  $c'$  move with a speed of one grid per unit time downwards and rightwards respectively. Note that the first vaule of  $c_i'$  is  $q_i$ , thus implying that the results of the designed linear array must be used, inputted and stored into the first column of PEs.

Table 6 yields the mapping between original and transformed indexes. The Table demonstrates that the involved cycle number to solve the problem is  $t_{\max} + 2K$ .

Table 6: Mapping between original and transformed indexed

$I=(i,j)$	$T(I)$	$c'(i)=$	$C'(i)$	$+P(i,j)$	$*c(j)$	$\pi I$	SI
1,1,1	3,1,1	1	1	1,1	1	3	1,1
1,1,2	4,1,2	1	1	1,2	2	4	1,2
...	...	...	...	...	...	...	...
1,1,K	K+2,K,N	1	1	1,K	K	K+2	K,N
...	...	...	...	...	...	...	...
1,K,1	K*2,K,1	K	K	K,1	1	K+2	K,1
...	...	...	...	...	...	...	...
1,K,K	2K+1,K,K	K	K	K,K	K	2K+1	K,K
2,1,1	4,1,1	1	1	1,1	1	4	1,1
2,1,2	5,1,2	1	1	1,2	2	5	1,2
...	...	...	...	...	...	...	...
2,K,1	K+3,K,1	K	K	K,1	1	K+3	K,1
...	...	...	...	...	...	...	...
2,K,K	2K+2,K,K	K	K	K,K	K	2K+2	K,K
...	...	...	...	...	...	...	...
$t_{\max}, 1,1$	$t_{\max} + 2, 1,1$	1	1	1,1	1	$t_{\max} + 2$	1,1
$t_{\max}, 1,2$	$t_{\max} + 3, 1,2$	1	1	1,2	2	$t_{\max} + 3$	1,2
...	...	...	...	...	...	...	...
$t_{\max}, 1,K$	$t_{\max} + K + 1, K$	K	K	1,K	K	$t_{\max} + K + 1$	1,K
$t_{\max}, 2,1$	$t_{\max} + 3, 2,1$	2	2	2,1	1	$t_{\max} + 3$	2,1
...	...	...	...	...	...	...	...
$t_{\max}, 2,K$	$t_{\max} + K + 2, 2,K$	2	2	2,K	K	$t_{\max} + K + 2$	2,K
...	...	...	...	...	...	...	...
$t_{\max}, K,1$	$t_{\max} + 2K + 1, K,1$	K	K	K,1	1	$t_{\max} + K + 1$	K,1
...	...	...	...	...	...	...	...
$t_{\max}, K,K$	$t_{\max} + 2K, K,K$	K	K	K,K	K	$t_{\max} + 2K$	K,K

Figure 8 captures the interconnections between the PEs. Extra links between the last column ( $PE_{iK}$ ) and the first row ( $PE_{Ki}$ ) have been added for realising the iterative process at the time  $t$ . This graph includes a multiplexer to select among threshold vector  $q_i$  (at the initialisation cycle), 0 (at the resting cycles), and a convergence control circuit.

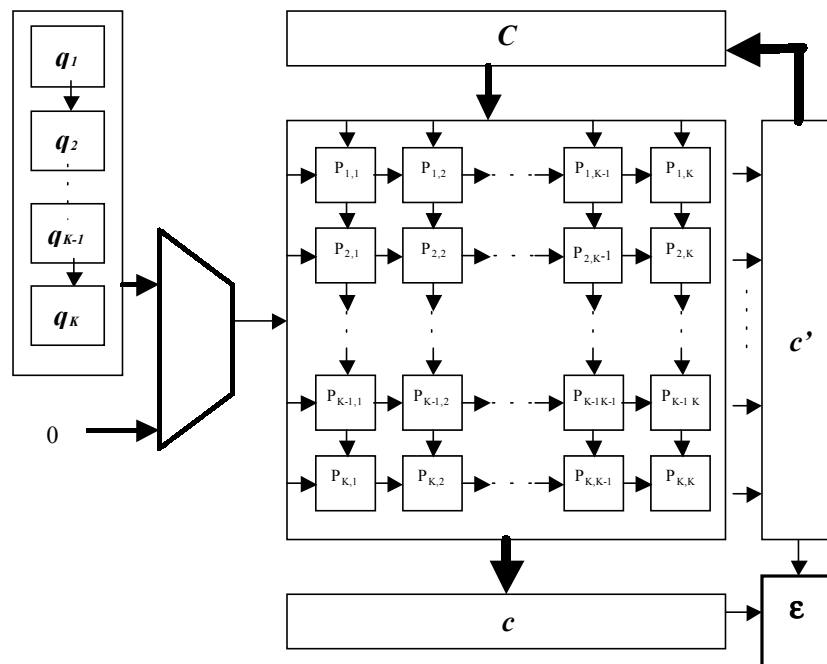


Fig. 8 Systolic structure for the iterative process.

The array output is obtained sequentially. To feed the upper input, a demultiplexer must be used, and a module K counter is also needed as the control of this circuit. The diagram for this is shown in Figure 9.

The implementation of the convergence control can be seen in Figure 10. This control is formed by two adders and one comparator. Its output is related to the network stability condition.

Finally, the arithmetic processing capabilities of each PE should include multiplying and accumulating operations (see Figure 11).

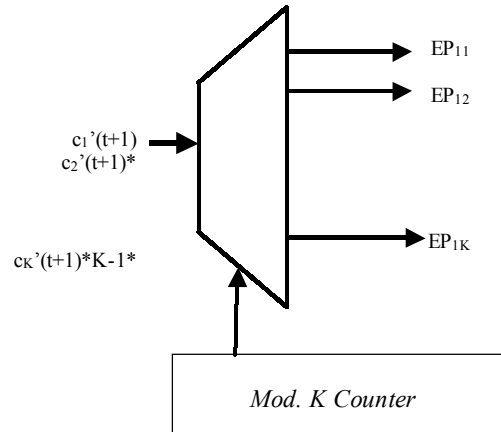


Fig. 9 Diagram of circuit needed for feeding the upper input to the right output.

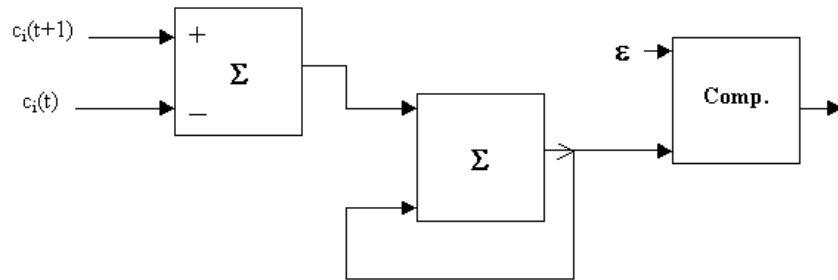


Fig. 10 Diagram of the control convergence process.

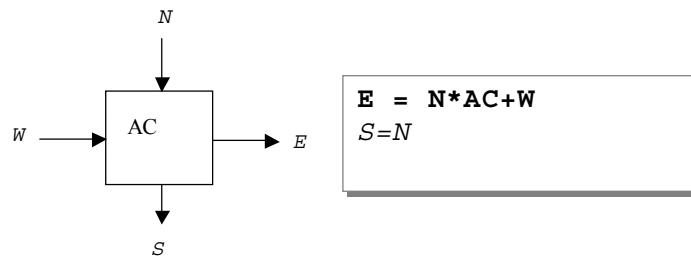


Fig. 11 Operations performed by PEs in Figure 8.

### 3.4. Design Summary

It seems a good idea, at this stage, to sum up the proposed systolic methodology: The HRNN model can be mapped onto a systolic structure formed by a rectangular array of  $K \times K$  PEs and a linear array of  $K$  PEs. The PEs should be programmable due to the varying functionalities involved. For their description, we have distinguished data movements and PE design requirements in both cases.

#### *Data movements within the linear systolic design*

The data movements comprise:

- ✓ The data  $y_j$  are handled sequentially in the first PE, in the original order. They are subsequently propagated downwards to all other PEs.
- ✓ The transpose of the reference spectra matrix,  $RT$  (row by row) in the PEs.
- ✓ When the value  $y$  arrives at the  $i$ th PE (from the top), it is multiplied by the stored data  $RT_{ij}$  to yield a partial sum  $q_i$  to be accumulated in the same PE.
- ✓ The PE changes mode to be ready for the iterative process phase.

#### *Processor element design requirements within the linear systolic design*

The processor elements include the following key components:

- a) Each PE should store a row of the transposition of the reference spectra matrix  $RT$ , i.e., a column of matrix  $R$ .
- b) Data are transmitted in one direction between two neighbouring PEs.
- c) Each PE should support all the arithmetic processing capabilities, including the multiplying and accumulating operations.

#### *Data movements within the rectangular systolic design*

In this case, we have distinguished two main steps: First, the weight matrix computation, and, secondly, the iterative process.

##### Weight Computation

The data movements comprise:

- ✓ The reference spectra  $R$ , as input into the first row of the PE array -- one column for each PE. They are subsequently propagated downwards to all PEs on the same column.

- ✓ The transposition of the reference spectra matrix  $\mathbf{RT}$ , as input on the first column of the PE array -- one row for each PE. They are subsequently propagated rightwards to all PEs on the same row.
- ✓ When the values  $R_{ji}$  and  $RT_{ij}$  arrive at the  $ij$ th PE (from the top and the right positions respectively), both are multiplied to yield a partial sum  $P_{ij}$  that is accumulated in the same EP. This value will be used during the iterative process phase.

#### Iterative process

In this case, the data movements comprise:

- a) At the time  $t$ , the contribution vector  $\mathbf{c}(t)$  as input on the first row of the PE array -- one component  $c_j(t)$  for each  $lj$ th PE. They are subsequently propagated downwards to all PEs on the same column.
- b) The threshold vector, previously calculated,  $\mathbf{q}$ , as input on the first column of the PE array -- one component  $q_i$  for  $il$ th PE. They are stored on all PEs on this first column.
- c) When the values  $q_i$  and  $c_l(t)$  arrive at the  $il$ th PE (from the right and the top sides respectively), the products of the  $P_{il}$  are computed with  $c_l$ , and from  $\lambda$  with  $q_i$  both results are added and routed to the  $il$ th PE. When the respective values arrive at the rest of  $ij$ th PEs, they are multiplied to yield a partial sum to be propagated rightwards to all PEs on the same row.
- d) As soon as the final partial sum is computed at the last PE of each row,  $PE_{iK}$ , can be routed to the corresponding  $PE_{li}$  on each column.

#### ***Processor element design requirements within the rectangular systolic design***

In this case, the processor elements comprise the following key components:

- a) Each  $PE_{ij}$  should store a row and a column of the reference matrix  $\mathbf{R}$ , plus a row of the transposition reference matrix  $\mathbf{RT}$ , and the value of the learning parameter  $-\lambda$ . Moreover, it is necessary that each PE has a memory cell for storing the computed weight  $P_{ij}$ .
- b) Data are transmitted in two directions between neighbouring PEs, downwards and rightwards, and an extra link between the last column and the first row is added for the iterative process.

- c) Each PE should support all the arithmetic processing capabilities including the multiplying and accumulating operations.

#### **4. Conclusions**

In this paper, a recursive Neural Network has been introduced to solve the Mixture Problem, based on the Hopfield Model. This model finds the composition of the mixture by operating with the spectra of the Reference Set. One interesting property, deduced from the algorithmic structure of the method, is related to the low-complexity VLSI structures, amenable for supporting the algorithm, since only multiplications and additions are required for programming an inversion process.

The main goal of this paper has been to demonstrate how this model can be implemented through a systolic array. The proposed methodology is applied as a linear structure for determining the net thresholds, and as a rectangular structure of programmable processor elements to obtain the net weight matrix and to implement the iterative process.

The linear array permits us to reduce the sequential computation time of  $K*N$  order to  $K+N$  clock cycles. This is an important advantage because the dimension of the used spectra is given by 1024 channels. In the weight determination phase, we need  $K*K*N$  cycles for a sequential computation while the systolic structure proposed reduces it to  $2K+N$ . Finally, for realising the iterative process, the reduction is from  $K*K*t_{max}$  (where  $t_{max}$  is the maximum iteration number) to  $2K+t_{max}$ .

The proposed algorithmic method can be applied to deal with several issues in spectroscopy, such as IR and visible spectrum decomposition. In addition, applications can be developed for hyperspectral unmixing through remote sensing on the Earth's surface. The reduction of computation time is an important benefit if a response is wished in real time.

#### **Acknowledgements**

The authors wish to express their gratitude to Dr. Alejandro Curado Fuentes for his linguistic revision of this paper.

## References

- [1] Adams J., Johnson P., Smith M. and George T. "A Semi-Empirical Method for Analysis of the Reflectance Spectra of Binary Mineral Mixtures". *Journal of Geophysic Res.*, **88**, 3557–3561 (1983).
- [2] Lawton W. and Martin M. "The Advanced Mixture Problem-Principles and Algorithms". Technical Report IOM384, Jet Propulsion Laboratory (1985).
- [3] Barnard E. and Cassasent P. "Optical Neural Net for Classifying Imaging Spectrometer". *Applied Optics*, **18**, n. 15, 3129-3133, (August 1989).
- [4] Díaz J.C., Aguayo P., Gómez P., Rodellar V. and Olmos P. "An Associative Memory to Solve the Mixture Problem in Composite Spectra". Proc. of the 35th Midwest Symposium on Circuits and System, (Washington DC August 1992), 422–428.
- [5] Rodellar V., Hermida M., Díaz A., Lorenzo A., Gómez P., Aguayo P., Díaz J.C. and Newcomb R. W. "A VLSI Arithmetic Unit for a Signal Processing Neural Network". Proc. of the 35th Midwest Symposium on Circuits and System, (Washington DC, August 1992), 891–894.
- [6] Pérez R.M., Martínez P., Silva A. and Aguilar P.L. "Influence of the Fixed Point Format on the Accuracy of the Neural Network Solution to the Mixture Problem". Proc. of the 3rd Advanced Training Course: Mixed Design of Integrated Circuits and Systems, (Lodz, Poland, May 1999), 413–418.
- [7] Pérez R.M., Martínez P., Martínez L., Díaz J.C., Rodellar V. and Gómez P. "A Hopfield Neural Network to Solve the Mixture Problem". Proc. of the VI Spanish Symposium on Pattern Recognition and Image Analysis, (Córdoba, Spain, April 1995), 744–745.
- [8] S.Y. Kung. "Digital Neural Networks". Prentice-Hall, Inc., 85–91 (1993).
- [9] R.M. Pérez, P. Martínez, A. Silva and P.L. Aguilar "Adaptive Algorithm to Solve the Mixture Problem with a Neural Networks Methodology". Proc. of 3rd. International Workshop In Image/Signal Processing: Advances In Computational Intelligence, IWISP'96 (Manchester, United Kingdom, November 1996), 133–136.
- [10] Pérez R.M., Martínez P., Silva A. and Aguilar P.L. "An Adaptative Solution to the Mixture Problem with Drift Spectra". Proc of the Third International Conference on Signal Processing, (Beijing, China, October 1996).
- [11] S.Y. Kung "VLSI Array Processors", *IEEE ASSP Magazine*, **2**, n.3, 4–22 (July 1985).
- [12] J.J. Navarro, J.M. Llaberia, and M. Valero. "Partitioning: An Essential Step in Mapping Algorithms Into Systolic Array Processors". *IEEE Computer*, 77–89 (July 1987).

- [13] D.I. Moldovan, "On the Design of Algorithms for VLSI Systolic Arrays".  
Proceedings of the IEEE, **1**, n.1 (January 1983).