Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Sciences 171 (2005) 273–287
INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL
www.elsevier.com/locate/ins

Optical implementation of the Kak neural network

A. Shortt [a], J.G. Keating [a,*], L. Moulinier [b], C.N. Pannell [b]

[a] *Department of Computer Science, National University of Ireland, Maynooth, Maynooth, Co. Kildare, Ireland*

[b] *Building of Physical Sciences (Room 116), University of Kent, Canterbury, CT2 7NR, UK*

Received 12 January 2002; received in revised form 27 February 2003; accepted 28 February 2004

Abstract

We show that the Kak neural network is suitable for optical implementation using a bipolar matrix vector multiplier. We demonstrate how the *CC*4 algorithm, with suitable modifications to the structure and training algorithm, may be used to build an optical neural network implementing *N*-Parity.

Keywords: Kak neural network; Optical neural network; Optical computing

1. Introduction

Kak proposed an efficient three-layer feedforward neural network architecture, with several associated learning algorithms [1,2], that maps a set of p n-dimensional binary vectors $\{\boldsymbol{\xi}_\mu | \mu = 1, 2, \ldots, p\}$ into p m-dimensional binary vectors $\{\boldsymbol{\upsilon}_\mu | \mu = 1, 2, \ldots, p\}$ using a constructive procedure $f(\cdot)$ for the mapping

$$\boldsymbol{\upsilon}_\mu = f(\boldsymbol{\xi}_\mu) \tag{1}$$

* Corresponding author.
E-mail addresses: ashortt@cs.may.ie (A. Shortt), john.keating@may.ie (J.G. Keating).

0020-0255/$ - see front matter
doi:10.1016/j.ins.2004.02.028

where $\mu = 1, 2, \ldots, p$. The training algorithms known as *Corner Classification* (*CC*) *Algorithms*, are examples of prescriptive learning, and train the network by isolating the corner in the n-dimensional cube of the inputs represented by the input vector being learned. Corner classification is based on two novel ideas, one enabling learning and one enabling generalization:

- Learning is achieved by mapping training vectors $\{\boldsymbol{\xi}_\mu\}$ into the corners of a multidimensional cube, isolating each corner of the cube, and associating each corner with a hidden neuron. Systematic recombination of the hidden neuron outputs produces the corresponding target output.
- Generalization is accomplished by invoking the concept of *radius of generalization*, which enables the network to classify any input vectors within a user-specified Hamming distance from a stored vector as belonging to the same output class as that stored vector.

To date, four learning algorithms [1,2,3] have been proposed including *CC*1 which obtained the weights using the perceptron algorithm, *CC*2 where the weights were obtained by data inspection but did not provide generalization, *CC*3 which is a modified version of *CC*2 that provides excellent generalization [4], and *CC*4 which is a fast prescriptive learning algorithm also with enhanced generalization capabilities. Two other corner classification algorithms *ALG*1 and *ALG*2, have been proposed for the architecture [5]. These algorithms are based on *CC*2 but demonstrate improved generalization for a variety of benchmark simulations. Although the scalability of the Kak network has been criticized [6] due the large number of hidden neurons in the network, there can be no doubt that the prescriptive learning, guaranteed convergence properties, superb generalization, and binary-weight architecture are the reasons the architecture has recently warranted attention by the electronic hardware community [7]. In a later section of this paper, we describe an optical architecture suitable for implementing the Kak neural network.

The Kak network is a feedforward, three-layer network of binary neurons, where the input-layer merely distributes data. Training involves the calculation of two integer-valued weight matrices ($\boldsymbol{w}^h$ and $\boldsymbol{w}^o$) and is generally achieved using the the most recent *CC*4 learning algorithm. $\boldsymbol{w}^h$ is the weight matrix between the input, or distribution, layer and the hidden layer and $\boldsymbol{w}^o$ is the weight matrix between the hidden layer and the output layer. The output of a neuron in the hidden or output layer is given by

$$o(x) \equiv \begin{cases} 1 & \text{if } x > \tau \\ 0 & \text{if } x \leqslant \tau \end{cases} \tag{2}$$

where x is the sum of weighted inputs to that neuron from the preceding layer and τ is some threshold. In practice, an additional constant value called a bias, is input to each hidden-layer neuron and τ is replaced with the value zero. This

may also be achieved by providing an extra input $\xi_{\mu,n+1} = 1$ for each input pattern $\mu = 1, 2, \ldots, p$.

The procedure provides an upper-bound on the number of hidden neurons, with the number of input and output neurons being n and m, respectively. For *CC*4 the number of hidden neurons is equal to the number of training samples, with each hidden neuron corresponding to one training sample. The output is the binary step function given in Eq. (2). Input and output layer weights are prescribed by inspection of the training samples with a weight of 1, assigned if the input neuron receives a 1 and −1 otherwise. The weights from the bias input is calculated as follows [3]: Let s be the number of 1s in the training vector, excluding the bias input, and let r be the desired radius of generalization. The choice of r depends on the nature of the generalization sought, and a full discussion on the criteria for particular choices of r has been fully researched and presented for a range of problems [4]. The weight between the bias and the hidden neuron corresponding to the training vector is $r - s + 1$. Therefore for any training vector $\boldsymbol{\xi}_\mu$ of length $n + 1$ bits including bias, we may write the weight assignment matrix $\boldsymbol{w}^h$ from the input, or distributive, layer to hidden layer as

$$w_{ij}^h = \begin{cases} 1 & \text{if } \xi_{\mu,j} = 1 \\ -1 & \text{if } \xi_{\mu,j} = 0 \\ r - s + 1 & \text{if } j = n \end{cases} \tag{3}$$

where $\xi_{\mu,j}$ is the jth element of the training vector $\boldsymbol{\xi}_\mu$. The weight between the ith hidden neuron and jth output neuron w_{ij}^o is determined using the rule

$$w_{ij}^o = \begin{cases} 1 & \text{if } \upsilon_{\mu,j} = 1 \\ -1 & \text{if } \upsilon_{\mu,j} = 0 \end{cases} \tag{4}$$

where $j = 1, 2, \ldots, m$ and $i = 1, 2, \ldots, k$ where k is the number of hidden neurons, and $\upsilon_{\mu,j}$ is the output of the hidden neuron associated with bit j of training input $\boldsymbol{\xi}_\mu$. The *CC*4 algorithm has been successfully [3] used for the *XOR*-problem, time series prediction (Mackey-Glass chaotic time series), the Twin Spirals Problem (pattern classification), and more recently as the core technology associated with an intelligent metasearch engine [8].

2. Bipolar modification to the *CC*4 algorithm

We are specifically interested in bipolar weight neural network architectures as they are particularly suitable for optical implementation using optical matrix-vector multipliers. With such systems all weights between neuron j in a network layer and neuron i in a preceding layer w_{ij} satisfy the relation $w_{ij} \in \{1, -1\}$. Hardware implementations of such systems are less susceptible to noise than architectures implementing analogue or digital weights. Such

systems typically have larger numbers of neurons per layer than non-discrete weight systems as there are greater constraints on the positioning of hyperplanes separating class boundaries.

We have highlighted the Kak model as an excellent example of a multilayer system capable of acting as a function approximation network or pattern classifier. It is obvious from Eq. (3), however, that the weights from the bias neuron (necessary for network generalization) may not be bipolar as they are dependent on the radius of generalization sought for a particular problem and training set. This results in an integral-valued matrix $\boldsymbol{w}^h$ rather than a bipolar valued matrix. We have investigated several techniques for obtaining a bipolar matrix $\boldsymbol{w}^H$ from matrix $\boldsymbol{w}^h$ and describe one technique here to demonstrate that our bipolar matrix-vector processor is a suitable optical architecture for use with the Kak neural network. In this section we present an alternative methodology for computing the weights from the input layer to the hidden layer thereby ensuring that a bipolar weight matrix is obtained. The approach is similar to *CC*4 as network generalization is accomplished using the rule proposed in Eq. (3). We shall refer to the modified algorithm as *mCC*4 in the rest of this paper.

Consider the case where the input layer is not augmented with a bias neuron. In this case $\boldsymbol{w}^H$ is given by

$$w_{ij}^H = \begin{cases} 1 & \text{if } \xi_{\mu,j} = 1 \\ -1 & \text{if } \xi_{\mu,j} = 0 \end{cases} \tag{5}$$

where ξ_μ is of length n. As generalization is expected of the network a p-element vector $\boldsymbol{\kappa} = (\kappa_1, \kappa_2, \ldots, \kappa_p)$ is computed, and is based on the desired radius of generalization for each training vector $\boldsymbol{\xi}_\mu$ using the following rule

$$\kappa_\mu = r - s_\mu + 1 \tag{6}$$

where $\mu = 1, 2, \ldots, p$ and s_μ is the number of 1s in the training vector $\boldsymbol{\xi}_\mu$. The activation for the vector $\boldsymbol{\upsilon}_h$ of hidden layer neurons for some input vector $\boldsymbol{\xi}$ is given in vector form by

$$\boldsymbol{\upsilon}_h = o\left(\boldsymbol{\xi}^T \boldsymbol{w}^H + \boldsymbol{\kappa}\right) \tag{7}$$

where $\boldsymbol{\xi}^T \boldsymbol{w}^H$ represents the matrix-vector product of the transpose of the input vector $\boldsymbol{\xi}$, and the bipolar weight matrix $\boldsymbol{w}^H$. The output υ of the Kak network for an input vector $\boldsymbol{\xi}$ is given by

$$\upsilon = o\left(\boldsymbol{\upsilon}_h^T \boldsymbol{w}^o\right) \tag{8}$$

where $\boldsymbol{\upsilon}_h$ is transpose of the hidden layer output vector obtained using Eq. (7).

The *mCC*4 algorithm is computationally light and does not significantly impact on the time associated with training. Furthermore, the technique reduces the size of the weight matrix by one row as it uses a subtraction of a fixed vector $\boldsymbol{\kappa}$ from the result of the matrix-vector multiplication to obtain the same

Table 1
Kak network implementation for N-parity problem using $CC4$

Inputs				Hidden-layer weights				Hidden-layer summations							Out
ξ_1	ξ_2	ξ_3	s	w_1^h	w_2^h	w_3^h	w_b^h	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	v_0
0	0	1	1	−1	−1	1	0	1	−1	0	−1	0	−2	−1	1
0	1	0	1	−1	1	−1	0	−1	1	0	−1	−2	0	−1	1
0	1	1	2	−1	1	1	−1	0	0	1	−2	−1	−1	0	0
1	0	0	1	1	−1	−1	0	−1	−1	−2	1	0	0	−1	1
1	0	1	2	1	−1	1	−1	0	−2	−1	0	1	−1	0	0
1	1	0	2	1	1	−1	−1	−2	0	−1	0	−1	1	0	0
1	1	1	3	1	1	1	−2	−1	−1	0	−1	0	0	1	1

degree of generalization as $CC4$. The primary significance of $mCC4$, however, is that it is guaranteed to have binary-valued weights which are suitable for optical implementation using our acousto-optic matrix-vector processor, described in the following section.

As an example, we investigate the Kak network's ability to solve the N-Parity Problem which is an illustration of hard learning typically used for benchmarking Artificial Neural Network learning algorithms. The N-parity training set consists of 2^N training pairs, with each training pair comprising an N-length input vector and a single binary target value. The 2^N input vectors

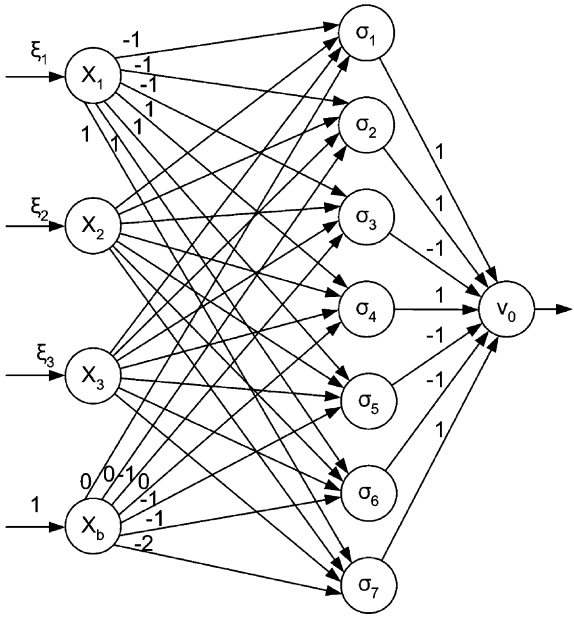

Fig. 1. Kak implementation of the parity network.

Table 2
Modified Kak network implementation for N-parity problem using $mCC4$

Inputs			Hidden weights			Hidden summations							Bias	Out
ξ_1	ξ_2	ξ_3	w_1^H	w_2^H	w_3^H	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	κ_μ	v_0
0	0	1	−1	−1	1	1	−1	1	−1	1	−1	1	0	1
0	1	0	−1	1	−1	−1	1	1	−1	−1	1	1	0	1
0	1	1	−1	1	1	0	0	2	−2	0	0	2	−1	0
1	0	0	1	−1	−1	−1	−1	−1	1	1	1	1	0	1
1	0	1	1	−1	1	0	−2	0	0	2	0	2	−1	0
1	1	0	1	1	−1	−2	0	0	0	0	2	2	−1	0
1	1	1	1	1	1	−1	−1	1	−1	1	1	3	−2	1

represent all possible combinations of N binary numbers. If a given input vector contains an odd number of 1s, the corresponding target value is 1; otherwise the target value is 0. Table 1 shows the input vectors, hidden-layer weights, hidden-layer summations and output targets for 3-parity determined using the $CC4$ algorithm described earlier. Our prototype processor currently can only perform computations on matrices with dimension up to 7×7, therefore for convenience purposes, we have not included the input $\xi_0 = (0,0,0)$ in the table as it always has the output 0. The architecture of the network is given in Fig. 1. Table 2 gives the input vectors, hidden-layer weights, hidden-layer summations, bias vector (κ) and output targets for the same problem using the $mCC4$ algorithm, and the network architecture is shown in Fig. 2.

3. A bipolar matrix-vector multiplier

Let $\boldsymbol{\delta} = (\delta_1, \delta_2, \ldots, \delta_n)$ where $\delta_i \in \{0,1\}$ be a Boolean vector representing the n-element input to a neural network or the output of some n-element hidden layer in the network. Let $\boldsymbol{w}$ where $w_{ij} \in \{-1,0,1\}$ be an $n \times p$ bipolar interconnection weight matrix between two consecutive neural network layers of length n and p, respectively. The matrix-vector product $\boldsymbol{\sigma} = \boldsymbol{\delta w}$ gives the activation for the neurons in the succeeding layer. For the Kak neural network described earlier, the neuron outputs are determined using Eq. (2).

The purpose of our matrix-vector processor, shown in Fig. 3, is to optically compute the activation vector $\boldsymbol{\sigma}$ in a single step, thereby exploiting the inherent parallelism of the optical processing device to speed up the training and operation of neural networks. The optical arrangement (discussed in detail in a later section) combined with the properties of the AOU (Acousto-Optic Unit) and a recording (CCD) camera implement the optical Matrix-Vector Multiplier (MVM). The activation weights are represented using the LCD (Liquid Crystal Display) panel shown in Fig. 3, and the input vector is represented using the

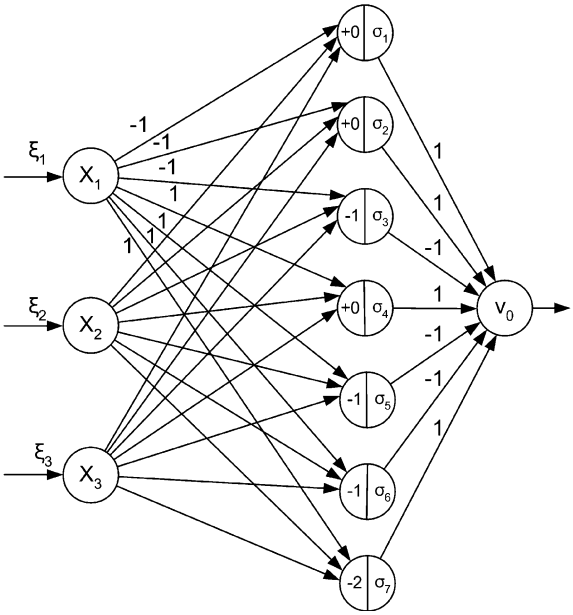

Fig. 2. Modified Kak implementation of the parity network.

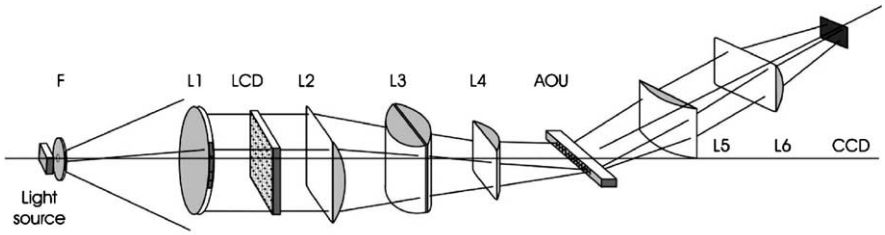

Fig. 3. Optical arrangement for the matrix-vector processor.

AOU. Although, the magnitude of the weight elements may be represented using intensity modulation, it is not possible to represent the sign of the element in this manner. For such optical systems it is typical to compute

$$\boldsymbol{\sigma} = \boldsymbol{\delta w}^{+} - \boldsymbol{\delta w}^{-} \tag{9}$$

where $\boldsymbol{w}^{+}$ and $\boldsymbol{w}^{-}$ are matrices of the magnitudes of positive and negative weights of $\boldsymbol{w}$, respectively, computed using

$$w_{ij}^{+} = \begin{cases} 1 & \text{if } w_{ij} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{10a}$$

$$w_{ij}^{-} = \begin{cases} 1 & \text{if } w_{ij} = -1 \\ 0 & \text{otherwise} \end{cases} \tag{10b}$$

The subtraction of the two resultant intermediary matrix-vector products in Eq. (10) may be performed either electronically or using software depending on the system architecture. The latter is used at present as this is a prototype system. We utilize an alternative approach whereby we construct a new matrix $\boldsymbol{w}^{\star}$ which is an augmentation of $\boldsymbol{w}^{+}$ and $\boldsymbol{w}^{-}$ and is given by

$$\boldsymbol{w}^{\star} \equiv \left[\boldsymbol{w}^{+} \;\middle|\; \boldsymbol{w}^{-}\right] = \left(\begin{array}{cccc|cccc} w_{11}^{+} & w_{12}^{+} & \dots & w_{1p}^{+} & w_{11}^{-} & w_{12}^{-} & \dots & w_{1p}^{-} \\ w_{21}^{+} & w_{22}^{+} & \dots & w_{1p}^{+} & w_{21}^{-} & w_{22}^{-} & \dots & w_{1p}^{-} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{n1}^{+} & w_{n2}^{+} & \dots & w_{np}^{+} & w_{n1}^{-} & w_{n2}^{-} & \dots & w_{np}^{-} \end{array}\right) \tag{11}$$

and compute a single matrix-vector product

$$\boldsymbol{\sigma}^{\star} \equiv \boldsymbol{\delta w}^{\star} = \left(\sigma_1^{\star}, \sigma_2^{\star}, \dots, \sigma_p^{\star}, \sigma_{p+1}^{\star}, \sigma_{p+2}^{\star}, \dots, \sigma_{2p}^{\star}\right) \tag{12}$$

$\boldsymbol{\sigma}^{\star}$ is a $2p$-element vector containing the positive and negative contributions to the neuron activations, and is obtained using a single optical matrix-vector processor operation. The advantage of our technique is to compute both matrix-vector products, $\boldsymbol{\delta w}^{+}$ and $\boldsymbol{\delta w}^{-}$, in a single pass. The required activation vector $\boldsymbol{\sigma}$ may be obtained from $\boldsymbol{\sigma}^{\star}$ using

$$\boldsymbol{\sigma} = \left(\sigma_1^{\star} - \sigma_{p+1}^{\star}, \sigma_2^{\star} - \sigma_{p+2}^{\star}, \dots, \sigma_p^{\star} - \sigma_{2p}^{\star}\right) \tag{13}$$

For the parity problem described above, we can calculate the hidden-layer weight matrix $\boldsymbol{w}_H^{\star}$ and output-layer weight matrix $\boldsymbol{w}_O^{\star}$ as

$$\boldsymbol{w}_H^{\star} = \left(\begin{array}{ccc|ccc} 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{array}\right); \quad \boldsymbol{w}_O^{\star} = \left(\begin{array}{c|c} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{array}\right) \tag{14}$$

Our prototype processor is currently capable of performing accurate binary calculations using matrices of maximum size 7×15, and vectors of variable length, although we have used vectors of size 7 for these experiments. These

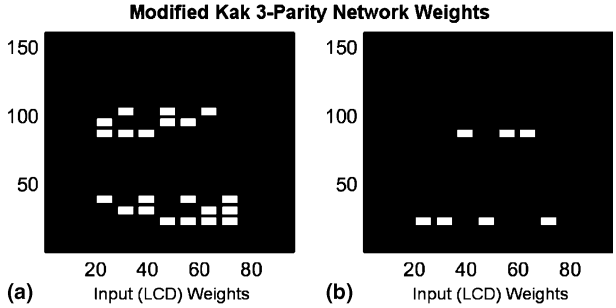

Fig. 4. Weights presented to LCD for the modified Kak 3-parity network for (a) hidden layer and (b) output layer trained using our *mCC*4 algorithm.

constraints are primarily determined by the signal-to-noise ratios and optical components for the current configuration. We have successfully conducted experiments that implement the optical ANN for the 3-parity problem trained with the *mCC*4 algorithm discussed in the previous section. It is, in fact, a *perfect* multiplier as it provided correct MVMs for both hidden and output weight matrices and all of the inputs shown in Table 2. The actual weights displayed on the LCD for the hidden and output layers are shown in Fig. 4(a) and (b), respectively.

For experimental purposes (i.e. the actual data sent to LCD), it was necessary to calculate a 7×15 matrix for both $\boldsymbol{w}_H^{\star}$ and $\boldsymbol{w}_O^{\star}$ where unused weights are set to zero, i.e.

$$\boldsymbol{w}_{LCD}^{\star} = \left(\begin{array}{ccc:c:cccc:c:ccc:cccc} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right). \tag{15}$$

Columns 1–3 of $\boldsymbol{w}_{LCD}^{\star}$ contain the positive weights and columns 9–11 contain the negative weights for the hidden-layer weight matrix $\boldsymbol{w}_H^{\star}$. In our experiments we always set the weight values of column 8 to zero as this column was used as a reference pivot for later subtraction of the negative contributions from the positive contributions in the resulting matrix vector multiplication ($\boldsymbol{\sigma}_{CCD}^{\star}$) accumulated in the CCD detector. Computation of the final matrix-vector

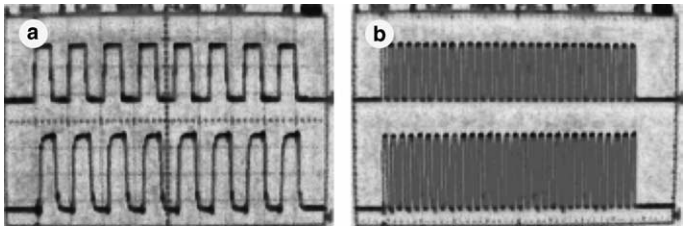

Fig. 5. Sample input vectors (lower signal) presented to the AOU: (a) an 8-bit vector (b) a 60-bit vector. The upper pulse train is used for system synchronization.

product ($\boldsymbol{\sigma}_{CCD}$) was accomplished using a "MATLAB" peak-finding and normalization function. This will be illustrated in Fig. 6, to be discussed below. At a later stage the CCD will be replaced with a linear photodetector array and perform the necessary subtractions in electronic hardware.

The optical arrangement is such that the values given in the matrix shown in Eq. (15) are rotated 90° counterclockwise before presentation to the LCD panel (i.e. the first column in the matrix becomes the bottom row of displayed weights) as shown in Fig. 4. Vector input to the multiplier was accomplished by sending a pulse train (0 represented by 0V and 1 represented by 1V) of

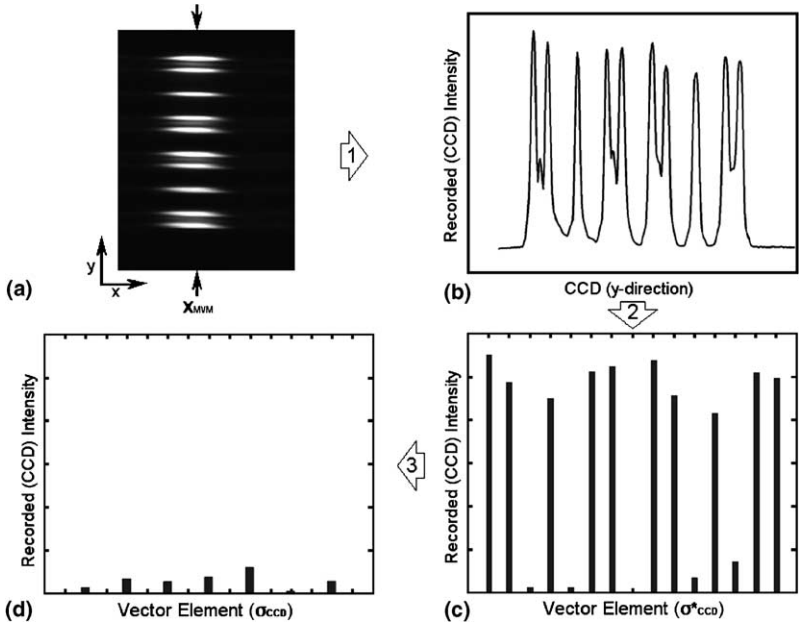

Fig. 6. Process involved in computing σ_{CCD}: (a) 2D MVM recorded on CCD, (b) 1D (sampled) MVM, (c) normalization and extraction of σ^{*}_{CCD} and (d) calculation of σ_{CCD}.

appropriate length to the AOU device. An eight-element vector representing all 1*s* is shown in Fig. 5(a). Using software control, it is possible to have longer vectors as shown in Fig. 5(b), however, the modified Kak 3-parity neural network experiments just required a seven-bit pulse train.

Fig. 6(a)–(d) shows the steps in the process to compute σ_{CCD} for a typical recorded MVM obtained using our optical arrangement. We select a line at a fixed x_{MVM} from the recorded CCD image and produce an intensity plot. Using the "MATLAB" peak-finding and normalization function, mentioned previously, we compute the 15-element vector $\boldsymbol{\sigma}^{*}_{CCD}$ which is then used to compute the 7-element vector $\boldsymbol{\sigma}_{CCD}$ that contains the result of the MVM. Irregularities in the LCD panel mean that the intensity distribution will not be constant over the modulating plane, therefore, and contributes to small inaccuracies in the final vector $\boldsymbol{\sigma}_{CCD}$ (theoretically $\boldsymbol{\sigma}_{CCD} = \mathbf{0}$ for the example shown). It is possible to compensate for these and other noise-related errors with post-processing, however.

4. Experimental arrangement and system evaluation

Fig. 3 shows a schematic of the apparatus used in our experiments. The light source *S* is focused through a spatial filter *F* in order to get a clean beam as homogeneous as possible in the transverse plane. The light source is placed at the focal distance of the lens L1 in order to produce a wide collimated beam incident on the LCD panel used to contain the network weights as described in the previous section. The set of three lenses (L2, L3 and L4) following the rectangular LCD panel are used to modify the shape of the beam in order to sufficiently illuminate the acousto-optic unit (AOU) that essentially performs the multiplication. The first of the three lenses orients the light beam in the vertical plane (or *y*-direction). The AOU is placed at the focal distance of the lens L2 but in order to improve homogeneity in the diffraction (i.e. the multiplication) the beam has to fill the complete height of the cell, therefore L2 is repositioned to obtain a 3 mm high beam in the AOU.

Although the beam needs to be focused in the *y*-direction, an image of the LCD panel has to be cast onto the AOU in the horizontal plane (or *x*-direction) for the multiplication to take place, i.e. the LCD panel and the acousto-optic cell are conjugate in the *x*-direction. As the AOU is a Bragg deflector the beam must be incident with as little divergence as possible so that the entire beam respects certain optical (i.e. Bragg) conditions. This result is achieved by combining the effects of two lenses L3 and L4, as a single lens would introduce too much divergence in the beam.

The first order of the diffraction pattern emerging from the AOU carries the result of the MVM and this order is collected by the two lenses L4 and

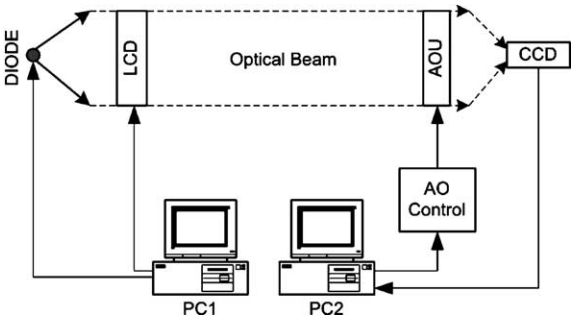

Fig. 7. Experimental control and data acquisition.

L5, ignoring the zeroth order and the little energy located in the higher-diffracted orders. Lens L5 focuses the beam in the x-direction in order to obtain a vector ($\boldsymbol{\sigma}_{CCD}$) representing the MVM. Lens L6 has the same purpose as the lenses L3 and L4, i.e. an imaging action between the acousto-optic device and the output plane defined by lens L5. The result is then acquired by the CCD camera.

Experimental control for our prototype matrix-vector multiplier is accomplished using two computers as shown in Fig. 7. The first computer, PC1, is used to write the network weights to the LCD panel, the second, PC2, delivers the vector to the AOU device and collects the results of the MVM using the CCD camera. Unfortunately, their use cannot be combined at this stage as the full screen of PC1 is necessary to drive the LCD panel (a 640×480 unit in parallel with the PC1's VDU). PC2 is also responsible for providing synchronization information, i.e. the camera needs to be triggered to record as soon as the MVM occurs. Ideally a single PC for control is most desirable as it would then be possible to run the multiplier without manual intervention.

The custom-built AO Control unit is central to the operation of our system, and is necessary because of the unique behavior of the acousto-optic cell and the limitations of the CCD camera. The cell is 25 mm long and as the speed of the acoustic wave carrying the vector signal propagates at 617 ms^{-1}, it takes 40 μs to fill the cell. The instant the cell is filled with the vector signal, the MVM occurs. In this short period, the CCD camera cannot acquire sufficient photons in order to acquire an image. It is necessary, therefore, to repeatedly send the same information until sufficient photons are acquired to obtain the result of the MVM. The AO Control stores a series of pulses (delivered by PC2), with amplitude between 0 and 1 (i.e. 0V and 1V) and repeatedly delivers this waveform to the AOU every 40 μs. Eventually, a photodiode array with a very fast response time could replace the CCD camera and acquire an image with one pulse. Fig. 5 shows a captured

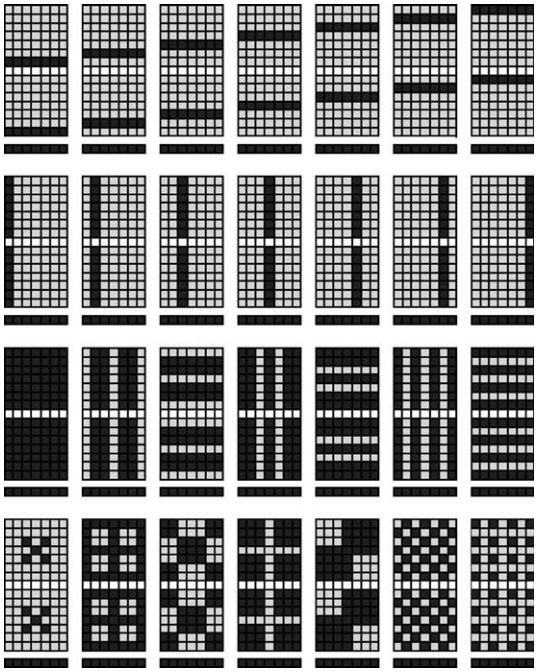

Fig. 8. Weight configurations used for matrix-vector multiplication tests.

oscilloscope image of two sample input vectors presented to the AOU: (a) an 8-bit vector (b) a 60-bit vector. The lower waveforms contain the actual vector signal and upper waveform is used for system synchronization, in particular triggering the CCD.

In order to investigate the full capabilities of the system (apart from the Kak 3-parity problem discussed in the previous section) we created many sample binary valued 7×15 matrices, shown schematically in Fig. 8, to test the device. Each of the matrices were arranged so that when multiplied by the vector (1,1,1,1,1,1,1) and post-processed using the procedure shown in Fig. 6 gave the resultant vector (0,0,0,0,0,0,0). The weight configurations were also chosen to investigate the properties of the LCD which, from earlier investigations, was known to be non-linear. The configurations also provided scaling information and aided the choice of spacing between the individual weight representations on the CCD to minimize errors in the MVM due to overlaps. We are pleased to report, as shown by the results presented in Fig. 9, that our acousto-optical multiplier is sufficiently robust to perform accurate matrix vector multiplications using 7×7 bipolar valued $(-1,0,1)$, or 7×15 Boolean (0,1) valued matrices and (7×1) Boolean vectors.

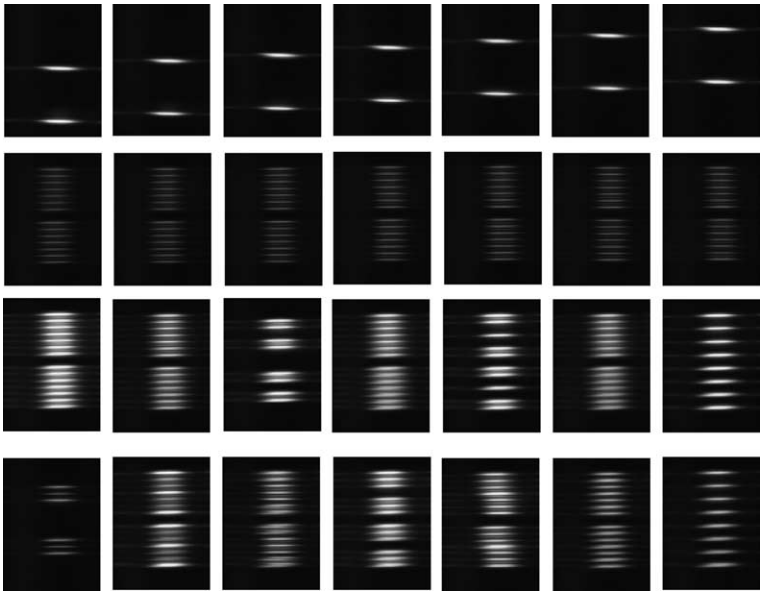

Fig. 9. Matrix-vector multiplications (from CCD) for the test weight matrices.

5. Summary

The guaranteed convergence properties, excellent generalization and instantaneous learning capabilities of the Kak neural network make it very appealing for optical implementation. In this paper, we provided a modification of the Kak CC4 algorithm in order to obtain a guaranteed bipolar weight matrix which was suitable for implementation in hardware using our acousto-optic matrix-vector multiplier. We provided successful results for an optical 3-parity problem based on the Kak neural network using network weights derived using the modified algorithm. Our hardware configuration is a prototype, however, and is currently limited by system noise and low-quality components essentially reducing the size of weight matrices that can be accurately used in matrix-vector multiplications. We plan on investigating these limits of the architecture not that the system is operational. The system is not fully automated and work is currently underway to provide experimental control using a single PC and automate the training and execution of larger, bipolar and continuous-valued neural networks.

References

[1] S. Kak, On training feedforward neural networks, Pramana: A Journal of Physics 40 (1993) 35–42.
[2] S. Kak, New algorithms for training feedforward neural networks, Pattern Recognition Letters 15 (1994) 295–298.

[3] K.-W. Tang, S. Kak, A new corner classification approach to neural network training, Circuits, Systems and Signal Processing 17 (1998) 459–469.
[4] S. Kak, On generalization by neural networks, Information Sciences 111 (1998) 293–302.
[5] K.B. Madineni, Two corner classification algorithms for training the Kak feedforward neural network, Information Sciences 81 (1994) 229–234.
[6] P. Raina, Comparison of learning and generalization capabilities of the Kak and Backpropagation algorithms, Information Sciences 81 (1994) 261–274.
[7] J. Zhu, G. Milne, Implementing Kak neural networks on a reconfigurable computing platform, in: R.W. Hartenstein, H. Gruenbacher (Eds.), FPL 2000, LNCS 1896, Springer-Verlag, (2000) 260–269.
[8] B. Shu, S. Kak, A neural network based intelligent metasearch engine, Information Sciences 120 (1999) 1–11.