Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Table of Contents

Preface

The field of artificial neural networks has come a long way. Depending on one's perspective, one can trace its origins to the research on Mach bands in visual perception and its interpretation in terms of lateral inhibition, to Hodgkin and Huxley's transmission line model of a nerve fiber, to McCulloch and Pitt's model of a neuron, to Hebb's postulate on learning, and to Rosenblatt's Perceptron. A common theme in almost all of the earlier works is the desire to build a model of the brain. The scope of this research began to widen with the publication of McCulloch and Pitt's model of a neuron. It is no accident that this model has an uncanny resemblance to the logic circuits of digital computers.

Although research in the computational capabilities of neural networks suffered a serious setback with the publication of Minsky and Pappert's book on perceptrons, the almost simultaneous development of the back propagation algorithm in the 1970s by several investigators marked the turning point in the direction of neural network research. The publication of Hopfield's paper, in 1983, in the "Proceedings of the National Academy of Sciences" (U.S.A.) almost caused a sensation in some circles. It is well over two decades now since the resurgence of neural network research. This is no longer a fringe area; it is mainstream.

This book is our way of celebrating the success of neural networks in different areas of engineering endeavor. The contributing authors come from many corners of the globe. All these chapters show how the power of neural networks can be exploited in modern engineering applications. Of the ten chapters, the first seven seem to form one group with an emphasis on image processing and industrial or manufacturing slant. Specifically, they touch on issues related to shape recognition, shape from shading, aircraft detection in SAR images, visualization of high-dimensional databases of industrial systems, 3-D object learning and recognition from multiple 2-D views, fingerprint classification and performance optimization in flexible manufacturing systems. The remaining three are applications to the communications area. Surprisingly, one of the earliest applications of neural networks was noise cancellation in telephone networks. That interest continues even today. The last three chapters included here address the issues involved in the exploding area of multimedia communications and in the area of mobile and cellular communications.

The first chapter by Ulgen, Akamatsu and Fukumi focuses on the issue of fast, on-line incremental training in the context of solving geometric shape recognition problems. It is well known that humans correctly recognize shapes independent of the variations in size, orientation and distortions caused by noise. The thesis

of this chapter is that the angular difference between any two consecutive tangent vectors drawn on the borders of a shape, is important in the perceived shape of the object. The authors train a neural net to learn these angular differences and perform the required classification. The authors used some pre-processing to extract features to render the procedure invariant to translations and rotations. They also performed some heuristic fuzzy filtering to impart noise immunity prior to the submission of the training samples to a feed-forward neural network.

The second chapter by Wei and Hirzinger deals with “shape from shading,” an important problem in computer vision. The problem is to infer the surface shape of an object from a single image. This problem can be formulated either as one of solving a partial differential equation or solving the minimization problem of the equivalent variational formulation. This chapter departs from this classical approach and uses a feed-forward neural network to parameterize the object surface and solves an ordinary extremum problem via a conjugate gradient method as well as a radial basis function approach.

The third chapter by Filippidis, Jain and Martin is about the use of neural nets to automatic target recognition with particular emphasis on detection of aircraft from synthetic aperture radar (SAR) images. The focus of this chapter is on fusion of different technologies to improve the target recognition statistics. In addition to the SAR image data, the method also uses texture data and information from the RGB bands from a color photograph and feeds this to a fuzzy “fusion” system.

Self-organizing maps (SOMs) are a powerful tool in clustering, dimensionality reduction where there is a need to preserve the topological relationships in the original data and compressed data. In Chapter 4, Simula, Vasara, Vesanto and Helminen explore its uses in visualizing higher-dimensional data, using the forest industry as a vehicle.

In the fifth chapter, Grossberg and Bradski develop a family of self-organizing neural architectures, called VIEWNET, for learning to recognize 3-D objects from sequences of their 2-D views. VIEWNET architectures use View Information Encoded With NETworks to accomplish this task. VIEWNET incorporates a preprocessor that generates a compressed but 2-D invariant representation of an image, a supervised incremental learning system (Fuzzy ARTMAP) that classifies the pre-processed representations into 2-D view categories whose outputs are combined into 3-D invariant object categories, and a working memory that makes a 3-D object prediction by accumulating evidence over time from 3-D object category nodes as multiple 2-D views are experienced. Fuzzy ARTMAP was modified to enable learning of the object classes. VIEWNET was modified to enable probability learning of object classes. VIEWNET was benchmarked on an MIT Lincoln Lab database of 128×128 2-D views of aircraft, including small frontal views, with and without additive noise. A recognition rate of up to 90% was achieved with one 2-D view and up to 98.5% correct with three 2-D views. The properties of 2-D view and 3-D object category nodes are compared with those of cells in monkey inferotemporal cortex.

Halici, Erol and Ongun, in the sixth chapter, discuss applications of hierarchical networks with particular reference to character recognition and fingerprint classification. Drawing from rich sources pertaining to pre-attentive and attentive levels of human cognition, the authors fashion a hierarchical system where the pre-attentive level is modeled by a self-organizing feature map that makes a rough decision of the eventual outcome by crudely clustering the input patterns. The attentive level is modeled by dedicating to each cluster a recognition module, which in turn is comprised of a feature extraction stage and a classifier stage. Feature extraction is performed using Principal Component Analysis and classification is performed using multilayer feed-forward networks. This method is successfully used for fingerprint classification.

Chapter 7 marks a turning point in the emphasis by leaving behind issues related to classification and focusing on optimization. Identifying the task of maximizing the throughput (i.e., the number of jobs done in a time unit) as the crux of optimization in a flexible manufacturing system, Cavalieri proceeds to formulate the problem in terms of event graphs (a special case of Petri Nets) and an integer linear programming problem. Then, instead of solving this using classical optimization procedures, the author solves the problem using a variation of the Hopfield network.

Wang and Ansari take a turn on the theme in Chapter 8 and address a problem in wireless digital communications, an exciting area of research. The channel assignment problem in mobile communication networks is known to be NP-complete. This problem is solved using mean field annealing (an idea borrowed from statistical mechanics) on an energy function of the Hopfield type.

Sheu, Wang and Young in Chapter 9 continue the theme by addressing a problem on the use of neural networks as baseband maximum likelihood sequence detectors. This technology is likely to play a progressively important role in the burgeoning field of multimedia communications. The emphasis here is on the so-called cellular compact neural networks with potential VLSI implementation possibilities.

Finally, in Chapter 10, Cavalieri and Mirabella talk about the use of Hopfield networks for process scheduling in communication systems, with an emphasis on scheduling for access to a physical channel in a computer network.

Although commendable progress has been made in the use of neural networks in a variety of application areas, the progress is not as rapid in the theoretical underpinnings of the methods. The trend in the use of hybrid methods as a way of improving performance attests to the plausible validity of this observation.

This book will be useful to researchers, practicing engineers and students who wish to develop successful applications of neural networks.

The editors are grateful to the authors for preparing such interesting and diverse chapters. We would like to express our sincere thanks to Berend Jan van der Zwaag, Ashlesha Jain, Ajita Jain and Sandhya Jain for their excellent help in the preparation of the manuscript. Thanks are due to Gerald T. Papke, Josephine Gilmore, Jane Stark, Dawn Mesa, Mimi Williams, Lourdes Franco and Suzanne Lassandro for their editorial assistance.

L.C. Jain, Australia
R. Vemuri, U.S.A.

Table of Contents

HOME SUBSCRIBE SEARCH FAQ SITEMAP CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Previous | Table of Contents | Next

Chapter 1
On-Line Shape Recognition with Incremental Training Using a Neural Network with Binary Synaptic Weights

F. Ulgen
Motorola, Lexicus Division
3145 Porter Drive, Palo Alto, CA 94304, U.S.A.

N. Akamatsu
Department of Information Science, The University of Tokushima
2-1 Minami-Josanjimacho, Tokushima-shi 770 Japan

M. Fukumi
Department of Information Science, The University of Tokushima
2-1 Minami-Josanjimacho, Tokushima-shi 770 Japan

Recognition of hand-drawn shapes is beneficial in drawing packages and automated sketch entry in handheld computers. Fast incremental training, which is performed on-line, is accomplished by the use of the Binary Synaptic Weights algorithm, a one-pass, feed-forward neural network training algorithm. Incremental training offers the advantage of adjusting the recognition capability of the system to the user's drawings. In this chapter, we propose a new approach to on-line geometric shape recognition with incremental training which utilizes a fuzzy function for filtering angular differences and a neural network for classification and on-line training. Instead of recognizing segments of a drawing and then performing syntactical analysis to match with a predefined shape, which is weak in terms of generalization and dealing with noise, we examine the shape as a whole. The main concept of the recognition method is derived from die fact that the angular difference between any two consequent tangent vectors of a shape is important in the perceived shape of outlines. Our application's aim is to recognize elliptic, rectangular, and triangular shapes in a way similar to human cognition of these shapes. Human beings recognize such basic shapes regardless of the variations in size, noise on the shape border, rotation, and in the case of triangles, regardless of the type of the triangle. The key concept is that the neural network learns the angular difference between any two consequent tangent vectors of a shape; therefore, only a few training samples that represent the class of die shape are sufficient. The

results are very successful in the sense that the neural network correctly classified shapes that did not have any resemblance to the shapes ill die initial training set.

1. Introduction

Neural networks, the focus of connectionist artificial intelligence, are characterized by a large number of simple processing elements connected as a network with highly parallel and distributed control. The neural network architectures and training algorithms vary according to the problem they are applied to. Here, we propose a new training algorithm that does not suffer from the drawbacks of the Backpropagation (BP) algorithm. The Binary Synaptic Weights (BSW) algorithm and its application for geometric shape recognition is the focus of this chapter.

By using state-of-the-art MOS fabrication processes, thousands of neurons and a large number of binary synapses can be realized in a single IC chip. Through the learning procedure, the binary synaptic weights can be easily modified and this provides adaptability to a variety of applications. Using the programmable BSW neural chips reconfigurable neural systems with learning capability can be produced. Besides being, easily implementable, the BSW algorithm can provide fast and guaranteed learning.

2. The Binary Synaptic Weights (BSW) Algorithm for Neural Networks

2.1 Motivation

The drawbacks of the Backpropagation algorithm drove researchers to look for new algorithms that do not suffer from the local minima trapping, offer fast training and do not require a trial and error approach to the parameters of the network topology. A number of learning algorithms utilizing binary synaptic weights has been proposed [1].

The BSW algorithm [2], which is implemented on a three layer network, determines die thresholds for the hidden and output layer nodes and the weights of the synaptic links between the layers, in addition to the required number of hidden layer nodes in one feed-forward pass. The main concept of the algorithm can be explained as the separation of globally intermingled patterns within an n-dimensional space through the formation of hyperplanes that separate different classes of patterns at a local region in die space. This in presented in Figure 1.

Each hyperplane, created to distinguish different patterns in the input space, corresponds to a hidden layer node and therefore the number of hidden layer nodes are automatically determined during training. The equation of a hyperplane in n-dimensional space is

$$\sum_{j=1}^{n} w_j i_j = c \qquad (1)$$

where w_j is interpreted as the synaptic link weight between the jth coordinate of the input vector $[i_1, i_2, \ldots, i_n]$ and a hidden layer node, and the constant c of the equation becomes the threshold of the node corresponding to the hyperplane. Therefore, in a network trained using the BSW algorithm, the synaptic link weights and thresholds of the nodes correspond to the pattern space separated by the hyperplanes.

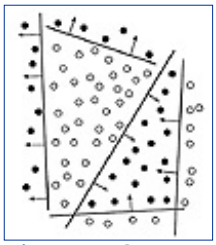


Figure 1 Separation of different patterns with multiple planes.

The activation functions of the nodes are hardlimiter thresholds, and thus a node is activated depending on whether the input pattern is on the positive or negative side of the hyperplane that represents that node.

The input and output pairs presented to the network are binary valued while the weights of the synaptic links between die input and hidden layer nodes are either 1 or -1. Between die hidden and output layers, die weights of the synaptic links are either 1 or 0 depending on die activation level of die connected hidden layer nodes. If a hidden layer node is activated, the corresponding synaptic link to the output node has a weight of 1. The

hidden layer node thresholds are always the constant c of equation (1). The output layer node thresholds are determined by the number of hyperplanes that enclose a pattern type. If there are *P* such planes, then a threshold of (*P* -0.5) is assigned to die output node associated with that pattern. The binary nature of the synaptic weights and the simplicity of the activation function make BSW a very promising candidate for hardware implementation.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

2.2 Separation of distinct patterns by BSW algorithm

In the separation of distinct patterns using the BSW algorithm, a representative input pattern vector, corresponding to a particular class, is chosen. In this discussion we will use the pattern types ● and ○, which were separated by hyperplanes in Figure 1. The sample input pattern vector, which we will refer to a $\vec{x}^{kdy}$, is a vector that is approximately at the center of all the other input pattern vector in the hyperspace for the same class of output. Formally, die calculation of $\vec{x}^{kdy}$ is as follows: Let *I* be the ($m \times n$) input matrix where m is the number of training samples and *n* is the input pattern vector size, and *O* be the ($m \times k$) output matrix where *k* is the output pattern vector size:

$$I = \begin{bmatrix} i_{11} & \cdots & i_{1n} \\ & \cdots & \\ i_{m1} & \cdots & i_{mn} \end{bmatrix} \qquad O = \begin{bmatrix} O_{11} & \cdots & O_{1k} \\ & \cdots & \\ O_{m1} & \cdots & O_{mk} \end{bmatrix}$$

Consider the separation of the pattern type ● in Figure 1. Initially an average vector, $\vec{x}^{av} = \{x_1^{av}, x_2^{av}, \cdots, x_n^{av}\}$ of type ● is calculated:

$$x_q^{av} = \left(\sum_{p=1}^{m} i_{pq} \times O_{pj} \right) \Big/ \sum_{p=1}^{m} O_{pj} \qquad (2)$$

where q = 1, .., *n*, and *j* represents a column of *O*, $1 \leq j \leq k$. If $\vec{x}^{av}$ exists among the ● patterns, it becomes $\vec{x}^{kdy}$. It is likely that $\vec{x}^{av}$ does not exist among the ● patterns, and if so the next step is to search for $\vec{x}^{kdy}$ among all ● patterns, to select the ● pattern which has the minimum Hamming distance to $\vec{x}^{av}$.

$$\vec{x}^{kdy} = \arg \min_{i=1}^{m} \left(HamDist\left[\vec{x}^{ave}, \vec{P}_i \right] \right) \qquad (3)$$

where $P_l = [i_{l1}, i_{l2}, \ldots, i_{ln}]$ is of type ● and *HamDist* represents the Hamming distance between two vectors. Again, using the Hamming distance formula, we determine $\vec{x}^{yes}$, the furthest input pattern vector with die same output classification type as $\vec{x}^{key}$, and $\vec{x}^{no}$, the nearest input pattern vector with an output classification type different than $\vec{x}^{key}$, which is of type ○ in this context.

$$\vec{x}^{yes} = \arg\max_{l=1}^{m}\left(HamDist\left[\vec{x}^{key}, \vec{P}_l\right]\right)$$
$$\vec{x}^{no} = \arg\min_{l=1}^{m}\left(HamDist\left[\vec{x}^{key}, \vec{P}_l\right]\right) \quad (4)$$

If $\text{HamDist}[\vec{x}^{key}, \vec{x}^{yes}] < \text{HamDist}[\vec{x}^{key}, \vec{x}^{no}]$, then the ● and ○ patterns are linearly separable. Otherwise we start the search for a Hamming distance D, initially zero, which will ensure that the majority of the ● type patterns will be included in the separation if we perform the separation at a distance (D + 0.5) from $\vec{x}^{key}$. In other words, at distance (D + 1) from $\vec{x}^{key}$, the number of (●) patterns is less than the number of (○) patterns. We then search for the ○ type patterns which might have been enclosed together with the ● type patterns, and perform further separation using the same procedure, this time starting with ○ type patterns as $\vec{x}^{key}$. This procedure is repeated until all of the patterns have been isolated, as is illustrated in Figure 1.

The hyperplane equation in n-dimensional space given by equation (1), assumes that $\vec{x}^{key}$ is at the origin of the coordinate system. Equation (1) takes the following form when there are components of $\vec{x}^{key}$ which are non-zero;

$$\sum_{q=1}^{n} y_q = k, \quad where\ y_q = \begin{cases} (1 - w_q i_q), iff\ x_q^{key} = 1 \\ w_q i_q, iff\ x_q^{key} = 0 \end{cases} \quad (5)$$

With reference to equation (5), the linear equation for the separation hyperplane to be formed at distance (D + 0.5) from $\vec{x}^{key}$ is:

$$\sum_{q=1}^{n} y_q = (D + 0.5), \quad where\ y_q = \begin{cases} (1 - w_q i_q), iff\ x_q^{key} = 1 \\ w_q i_q, iff\ x_q^{key} = 0 \end{cases} \quad (6)$$

where (D + 0.5) is the constant c of equation (1). Collecting the constants on the right-hand side of the equation (6), we obtain (D + 0.5) - ± which becomes the threshold of the node that represents the newly created hyperplane and where $x_q^{key} = 1$. The synaptic link weights are set as follows:

$$w_{uv} = \begin{cases} -1 \quad iff\ x_q^{key} = 1 \\ 1 \quad iff\ x_q^{key} = 0 \end{cases} \quad (7)$$

where q = 1, ⋯, n andw_{uv} is the synaptic link that connects the input node *u* to the newly created vth hidden layer node.

2.3 Algorithm

The BSW algorithm is presented in pseudo-code below:

Let *I* be the (*n* × *m*) input matrix where *n* is the number of training samples and *m* is the input vector size, and *O* be the (*n* × *k*) output matrix where *k* is the output vector size:

$$I = \begin{bmatrix} i_{11} & \cdots & i_{1m} \\ & \cdots & \\ i_{n1} & \cdots & i_{nm} \end{bmatrix} \qquad O = \begin{bmatrix} O_{11} & \cdots & O_{1k} \\ & \cdots & \\ O_{n1} & \cdots & O_{nk} \end{bmatrix}$$

For each column, j of O, $(1 \leq j \leq k)$ do:

Step 1

1.1: Calculate the vector $Ave = [a_1,a_2,...,a_m]$, where a_q is given by

$$a_q = \frac{\left(\sum_{p=1}^{n} i_{pq} \times o_{pj}\right)}{r} \qquad \text{where} \begin{cases} 1 \leq q \leq m \\ r = \sum_{p=1}^{n} o_{pj} \end{cases}$$

1.2: Calculate the vector $key = [ky_1,ky_2,...ky_m]$, where *key* is given by

$$key = \min_{l=1}^{n}\left(Ham[Ave, R_l]\right) \qquad \text{where} \begin{cases} R_l \text{ is the } l^{th} \text{ row of } I \\ \text{Ham}[X,Y] \text{ is the} \\ \text{Hamming distance} \\ \text{between vectors } X \& Y \end{cases}$$

1.3: Calculate the vector Yes.dis = $[u_1,u_2,...u_m]$, where *Yes.dis* is given by

$$\text{Yes.dis} = \max_{l=1}^{n}\left(\text{Ham}[key, R_l]\right) \qquad \text{iff } O_{lj} = 1$$

1.4: Calculate the vector No.clo = $[c_1,c_2,...c_m]$, where *No.clo* is given by

$$\text{No.clo} = \min_{l=1}^{n}\left(\text{Ham}[Ave, R_l]\right) \qquad \text{iff } O_{lj} = 0$$

1.5: *Dist* :=0

Step 2:

```
if (Ham[key,Yes.dis]<Ham[key,No.clo])
then goto Step 4
else{ Dist :=1;
       goto Step 3}
```

Step 3:

if $\left(\sum_{p=1}^{n} O_{pj} < \sum_{r=1}^{n} O_{rj}\right)$ or (*Dist*>Ham[*key*, Yes.dis])

$$\text{iff} \begin{cases} o_{pj} = 1 \,\&\, o_{rj} = 0 \\ \& \\ \text{Ham}[key, R_p] = \text{Ham}[key, R_j] = Dist \end{cases}$$

```
then goto Step 4;
else {Dist := Dist +1;
         goto Step 3}
```

Step 4:

```
Separation_Plane_Creation[Dist, Dist+1]
if (in created hyperplane there exists R_l where o_lj <> NN(key),
```

```
    (1 d l d n))
    then { key := R_l;
           Dist := 1;
           goto Step 5
         }
     else exit;
```

Step 5:

$$\text{if } \left(\sum_{p=1}^{\hat{}} o_{pj} < \sum_{r=1}^{\hat{}} o_{rj}\right) \qquad \text{iff } \begin{cases} o_{pj} = 1 \,\&\, o_{rj} = 1 \\ \& \\ \text{Ham}[key, R_p] = \text{Ham}[key, R_r] = Dist \end{cases}$$

```
    then { Dist := Dist +1;
           goto Step 5
         }
    else { Separation_Plane_Creation[Dist, Dist+1];
           goto Step 6
         }
```

Step 6:

```
    if(in the created hyperplane there exists R_1 where o_tj =
       0, (1 d l d n))
    then { Dist := Dist +1;
           goto Step 4;
         }
    else exit;
```

Separation_Plane_Creation

```
    if (key = [ky_1, ky_2, ..., ky_m]) where (1 d s d m & ky_s = 0)
```

$$\text{then } \left\{separation\_plane = \sum_{t=1}^{m} x_t = (Dist + Dist + 1)/2\right\}(*)$$

$$\text{else } \left\{separation\_plane = \sum_{t=1}^{m}(1 - x_t) + \sum_{q=1}^{m} x_q = (Dist + Dist + 1)/2\right\}(**)$$

$$\text{iff } \left(ky_t = 1 \,\&\, ky_q = 0 \text{ in the vector } key\right)$$

```
   Threshold of a hidden node = constant RHS of (*) or (**)
   (each hidden node represents one plane)
```

w_{uv} := weight of the synaptic link between input node u and hidden v.

$$:= \begin{cases} 1, iff\ ky_u = 0 \text{ in } key \\ -1, iff\ ky_u = 1 \text{ in } key \end{cases}$$

z_{kl} := weight of the synaptic link between hidden node k and output node l.

$$:= \begin{cases} 1, iff \text{ node } k \text{ gets activated} \\ 0, iff \text{ node } k \text{ is not activated} \end{cases}$$

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

2.4 Example for the BSW algorithm

In this simple example, the different patterns ●,○ in 3-dimensional space will be separated. The input to the neural network is as in Table 1.

Table 1 Training data for the example problem

Input Patterns	Output Patterns
P_1=(0,0,0)	1 ●
P_2=(1,1,0)	1 ●
P_3 =(0,0,1)	0 ○
P_4=(1,0,0)	0 ○
P_5=(1,1,1)	0 ○

These patterns can be represented as in Figure 2(a), coinciding with the corners of a hypercube. As explained in Section 2.2, initially $\vec{x^{av}}$ will be determined. Since there are only two ● patterns, we calculate the x_j^{av} to be the average of the *j*th components of pattern vectors P_1 and P_2, where $(0 \leq j \leq 2)$. Thus x_1^{av} = (0+1)/2, x_2^{av} = (0+1)/2, x_3^{av} = (0+0)/2. As 0.5 falls between 0 and 1, either (0,0,0) or (1,1,0) can become $\vec{x^{av}}$. Let us choose $\vec{x^{av}}$ to be (0,0,0), then *HamDist*[(0,0,0),(0,0,0)] = 0 and *HamDist*[(0,0,0),(1,1,0)] = 2 and according to equation (3) P_1 = (0,0,0) become $\vec{x^{key}}$.

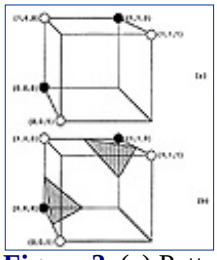


Figure 2 (a) Patterns to be separated; (b) Separation by BSW.

At this point, we classify all patterns in the pattern space according to their Hamming distances to $\vec{x}^{key}$. This is presented in Table 2.

Table 2 Hamming distance classifications of training patterns according $\vec{x}^{key}$

Ham. Dist.= 0	Ham. Dist.= 1	Ham. Dist.= 2	Ham Dist.= 3
(0,0,0) ●	(0,0,1) ○	(1,1,0) ●	(1,1,1) ○
	(1,0,0) ○		

From this representation we can see that $\vec{x}^{yes} = (1,1,0)$, $HamDist[\vec{x}^{key},\vec{x}^{key}]$ = 2 and $\vec{x}^{no} = (0,0,1) or (1,0,0)$, $HamDist[\vec{x}^{key},\vec{x}^{no}]$ = 1. Since $HamDist[\vec{x}^{key},\vec{x}^{yes}]$ is greater than $HamDist[\vec{x}^{key},\vec{x}^{no}]$, we can say that the patterns are not linearly separable. At distance D = 0, there is only one pattern $\vec{x}^{key}$ and at D = 1, the number of ○ patterns is 2, while the number of ● patterns is 0. Therefore, as at distance D = 1, the number of ○ patterns is greater than number of ● patterns, we perform separation at D = 0 and the equation of the separation hyperplane becomes x1+x2+x3 = (0+0+1)/2 = 0.5. The side of the hyperplane which encloses (0,0,0); is represented by $x1+x2+x3 \leqq 0.5$. With respect to Figure 2(b), in order to represent this in a form suitable for threshold logic, we multiply both sides of the inequality by -1 and obtain $-x1-x2-x3 \geqq -0.5$. The threshold of the newly created hidden layer node in -0.5 and the synaptic link weights connecting this node to the input nodes are the coefficients of the x-components in this equation (-1,-1,-1).

The next step is to separate the remaining ● pattern and to accomplish this, P2 = (1,1,0) is selected as $\vec{x}^{key}$. Classification of all patterns according to their Hamming distances to $\vec{x}^{key}$, we determine that the separation plane will be formed at D = 0. With reference to equation (6), the separation plane's equation becomes $(1-x_1)+(1-x_2)+x_3 = (0+0+1)/2 = 0.5$ and if we accumulate all constants on one side the equation becomes $-x_1-x_2+x_3 = -1.5$. Since the side of the plane which encloses (1,1,0) is desired, we set the synaptic links as (1,1,-1) and the threshold as 1.5. Finally, the threshold of the output layer node is determined. The output pattern vector has only one column, which requires only one output node. The number of planes created to enclose either ● pattern was 1; therefore, the threshold of the output node is (1- 0.5) = 0.5. The resultant network topology is shown in Figure 3.

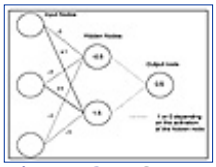


Figure 3 BSW network topology for the example problem of Section 2.4.

3. On-Line Geometric Shape Recognition with Fuzzy Filtering and Neural Network Classification

3.1 Geometric Shape Recognition

The aim of this section is to present a simple method to recognize and correctly redraw hand-drawn regular geometric shapes, such as circles, ellipses, squares, rectangles, and triangles. This would be beneficial in automated sketch entry in software packages to handle drawings in computers. Each shape is recognized as a whole, regardless of size, translation, rotation, or choice of starting point, instead of recognizing the individual segments and then performing syntactic analysis as discussed in [3]. The main concept of the recognition method presented in this section, is derived from the fact that angles are very important in the perception of shapes. Although Davis has also used this concept in his work [4], his method of syntactically analyzing the angles is weak in terms of dealing with the noise along the boundary of the shape. To overcome previous works' demerits and accomplish fast recognition with a high success rate, we have employed scaling/rotation/translation-invariant feature extraction, noise reduction through fuzzy function filtering and classification using a neural network.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Feature extraction is performed on the captured points along the boundary of the shape. Since the captured data for the boundary of a hand-drawn shape invariably includes noise, the fuzzy function is beneficial in elimination of most of the noise and also in detecting portions of the boundary of the shape that exhibit significant angular differences. The features of a shape are represented as a parsable string in terms of significant internal angles and this string is then input to an appropriate type of classifier [5]. The feature extraction and fuzzy filtering that we present in this chapter, prepares data that is invariant in terms of translation, scaling, rotation and in the case of triangles, even invariant of the particular class of the triangle, with a reduced noise level for input to the neural network. Previous work employing neural networks in the classification process usually concentrated on teaching a particular shape to the network and then measuring how well the network performs on noisy versions of that particular shape [6-8]. However, our purpose is to teach the network the definition of a class of shapes, such as triangles or ellipses. In our method, the training set is not necessary to include all kinds of possible shapes, but the training can be accomplished using only a small representative set. The speed with which BSW can be trained enabled us to add an incremental training module to the application.

3.2 Feature Extraction

Preprocessing of the data for the purpose of feature extraction is performed prior to the application of recognition algorithms. In other words, the purpose of preprocessing is the creation of an intermediate representation of the input data, in which information that does not help in the discrimination between different classes is reduced and desired or ‘useful’ information is enhanced [9]. In the case of handwritten shapes, undesirable information includes noise due to the limiting accuracy of the tablet, noise introduced in the digitizing process, variations in the capture speed of the mouse and erratic hand motion while drawing with the stylus or dragging the mouse. In the presence of undesirable information in the input during shape recognition, we have to distinguish between essential and nonessential curvature changes along the boundary. In order to accomplish this without a significant loss in useful information, our feature extraction process involves a number of steps. We are proposing two approaches to feature extraction designated as Method1 and Method2.

3.2.1 Method1

Method1 involves the steps of resampling the input data, calculation of the center of the shape and extraction of significant points.

3.2.1.1 Resampling

The raw input data obtained from the contact between the stylus with the tablet, or the mouse with the pad, which is the input media used for the application purposes of this chapter, has enormous variations in drawing speed and therefore the points that represent the shape are not equally spaced. In order to obtain equally spaced points along the trajectory of the stylus or the mouse, we need to resample the input data [9].

Simple linear interpolation between captured points within every pen-down/mouse button-down and pen-up/mouse button-up portions of the drawing serves the purpose of resampling. Linear interpolation is the interpolation between given two points (x_0,f_0) and (x_1,f_0) through the formulae [10]:

$$P_1(x) = f_0 + (x - x_0) f[x_0, x_1]$$
$$f[x_0, x_1] = \frac{f_1 - f_0}{x_1 - x_0} \qquad (8)$$

Figure 4 Rectangle drawn with unordered strokes; N is the total number of boundary points.

The application of interpolation is feasible only between subsequent pen-down/pen-up portions of the capture data because of the possible variations in the order of strokes while drawing a shape as seen in Figure 4.

3.2.1.2 Calculation of center of the shape

In the case of handwritten character recognition, in order to obtain a representation which is scale and translation invariant, the center point (x_c, y_c) of the characters is calculated with the following formulae:

$$x_c = \frac{x_{max} + x_{min}}{2} \quad and \quad y_c = \frac{y_{max} + y_{min}}{2} \qquad (9)$$

However, in shape recognition, calculation of the center of the shape with this method does not always bring the correct center of the shape, as can be seen in Figure 5. Therefore, we chose to calculate the center point of a shape through the center of gravity method which defines:

$$x_c = \frac{\sum_{i=0}^{N-1} x_i}{N} \quad and \quad y_c = \frac{\sum_{i=0}^{N-1} y_i}{N} \qquad (10)$$

where N is the total number of captured data points. The drawing speed, i.e., the speed at which the stylus is moved or the mouse is dragged, makes a difference in the number of points captured in a certain part of the shape, which may lead to shifts in the location of the center of gravity. For this reason, the calculation of (x_c, y_c) is performed after resampling of the captured data points.

Figure 5 Calculation of the center of the shape.

3.2.1.3 Extraction of Significant Points

After resampling and the calculation of the center of the shape (x_c, y_c), we would like to extract L points, which will sufficiently represent the shape, out of the total of N input data points. Figure 6 depicts L quantized vectors that are angularly equispaced, meaning there is an angular difference of (2 À /L) between every two successive vectors. The L sample points $[p_x[i], p_y[i]), i = 1, \cdots, L]$, will be chosen to be the intersections of the shape boundary with L quantized direction vectors originating from $(x_c\ y_c)$ as seen in Figure 7. These sample points possess the circularity property as $P_{L+k} = (p_x[L+k], p_y[L+k]) = P_k = (p_x[k], p_y[k])$ for any integer $k, 1 \leq k \leq L$. This method of sample point extraction makes the resultant shape representation a candidate for the circular auto-regressive (CAR) model [10]. This is desirable as the parameters of CAR

models are invariant to transformations on the boundary, such as translation, choice of starting point and rotation over angles that are multiples of (2 À /*L*).

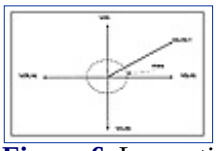


Figure 6 L quantized vectors that are angularly equispaced (theta = 2 À /*L*).

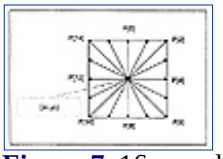


Figure 7 16 sample points are extracted on a shape (*L* = 16).

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

There are cases where a significant point in a shape falls between two sample points as depicted in Figure 8. For the purposes of this method of recognition, any point (x), which stands for the significant point, must be taken into consideration. In addition to this, we must also keep the number of sample points a constant, as the sample points will become the input to a neural network classifier after further manipulation.

The method we use in obtaining significant points such as (x) in Figure 8, is based upon the method used for obtaining line approximations of the boundary of a shape in [11]. This method involves finding the longest line segments that have the minimum sum of absolute errors along the digitized shape boundary. The equation of a line that originates in $P_i = (x_i, y_i)$ and ends in $P_j = (x_j, y_j)$ is:

$$(y_j - y_i)x - (x_j - x_i)y + x_i y_j - x_j y_i = 0 \qquad (11)$$

The points that lie in between P_i and P_j add towards an error term depending on how well they fit the equation (11). This error term (E = sum of absolute errors) for each line segment $[P_i, P_j]$ is calculated as follows;

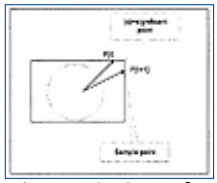


Figure 8 Significant point between two sample points.

$$E = \frac{\sum_{k=i+1}^{j-1} \left| (y_j - y_i)x_k - (x_j - x_i)y_k + x_j y_i - x_i y_j \right|}{\alpha \left\{ (x_j - x_i)^2 + (y_j - y_i)^2 \right\}^{1/2}} \qquad (12)$$

where $k = i+1, \cdots, j-1$ are the points along the boundary of the shape that lie in between P_i and P_j and $\pm$ is the constant associated with the sensitivity of the input device. Since our purpose is to find the longest line segment with the minimum E, we maximize (LS - E) where LS is the length of the line segment under consideration.

$$LS - E = \left\{(x_i - x_j)^2 + (y_i - y_j)^2\right\}^{1/2} - E \quad (13)$$

Maximizing the equation (13) corresponds to maximizing LS and minimizing E. We select a sample point on the boundary and assign it to P_i. We must then normalize the coordinates of the remaining points such that P_i becomes the origin. Each consecutive point on the boundary is tested until the first line segment that satisfies the maximization of the equation (13) is found. This point becomes the new P_i and the process is repeated until the boundary has been traversed. Eventually we obtain a list of significant points in addition to the previously calculated sample points, which need to be merged to maintain a constant number of representative points. The merging process involves shifting the sample points toward the nearest significant point in the case where a significant point falls in between two sample points is illustrated in Figure 8.

3.2.2 Method2

In the second method of feature extraction, we aim to eliminate crossovers, both internal and external, and finally obtain L sample points as in Method1. The L sample points, which are the beginning and end points of the vectors $\{(P_x[i], P_y[i]), i = 1, \cdots, L\}$, are chosen to be the intersections of the input shape boundary with the L vectors originating from (x_c, y_c), as shown in Figure 9(a). Figures 9(b) and (c) present a rectangle and circle, respectively, that have been drawn by hand. As can be seen in the figures, these hand-drawn shapes contain a large amount of undesirable information such as internal and external extensions, and concave segments, which should be removed. In addition to removing the undesirable information, we would like to detect information which will aid in recognition, such as intersection points or corners, and also those input points which contribute significant information to the class of the shape. In order to produce the intermediate representation of the input data as shown in Figure 9(a), from hand-input shapes such as those of Figures 9(b) and (c), without losing significant information, the feature extraction process involves a number of steps. These include the extraction of a set of sample points which contribute significant information to the shape, calculation of the center point of the shape, detection of intersection points, removal of internal and external extensions, determining the convex hull of the shape, determining sample points along the shape boundary, and the formation of tangent vectors between these points. These steps are discussed in the following subsections.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

3.2.2.1 Extraction of Significant Sample Points

Of the total N input data points, we would like to extract the J points, which capture only the significant changes along the shape boundary, without losing any significant information. Also, we wish to process each point on-line, as it is received from the mouse or stylus. This will reduce the total amount of data which must be passed to later stages of the preprocessor and also allows the overlapping of the input and processing of the received data. As in Method1, we have adapted the method proposed in [11], which determines the optimal polygon from a digital curve using the L_1 norm, for use in an on-line recognition environment. This method determines the longest line segments that have the minimum sum of absolute errors along the digitized shape boundary. The equation of a line that originates at $P_i = (x_i, y_i)$ and ends in $P_j = (x_j, y_j)$ is the same as in equation (11). The points (x_k, y_k), $k = i+1, i+2, \cdots, j-1$, which are on the boundary of the shape between points P_i and P_j contribute to the sum of the absolute errors E, for the line segment representing the boundary $[P_i, P_j]$, as in equation (12). The variable M is the sensitivity parameter that is determined as a function of the input device type and the speed with which the input device is moved while data is being captured, such as ± = f(Æ ,t), where t is a real number heuristically determined for each type of input device and Æ is a real number determined according to the acceleration and deceleration in the strokes of the drawing. Since our purpose is to find the longest line segment with the minimum E, we maximize $(L_s - E)$, where L, is the length of the line segment under consideration:

$$\begin{aligned} M_s &= L_s - E \\ &= \left\{(x_j - x_i)^2 + (y_j - y_i)^2\right\}^{1/2} - E \\ &= \left\{(x_j - x_i)^2 + (y_j - y_i)^2\right\}^{1/2} - \frac{\sum_{k=i+1}^{j-1} \left|(y_j - y_i)x_k - (x_j - x_i)y_k + x_j y_i - x_i y_j\right|}{\alpha\left\{(x_j - x_i)^2 + (y_j - y_i)^2\right\}^{1/2}} \end{aligned} \quad (14)$$

Maximizing M_s in equation (14) corresponds to maximizing L_S, and minimizing E. Given the set of points representing a stroke, $P = \{p_0, p_1, ..., p_{n-1}\}$ input by mouse or stylus, where n is the number of points, our on-line approach to significant point detection proceeds as follows: The first point p_0 is accepted as a significant point and is assigned to (x_i, y_i) of equation (14). Each time a new point, p_R $\{p_1, ..., p_{n-1}\}$ is received from the mouse or stylus, it is assigned to (x_j, y_j) and M_R is calculated using equation (14). If $M_R \geq$

M_{R-1}, then P_{R-1} is not significant and the process will be repeated when a new P_R is received. If $M_R < M_{R-1}$, then p_{R-1} is significant, and therefore it is saved and it is also assigned to be the new (x_i, y_i). This process is repeated until all of the input points for the stroke have been received, after which the end point p_{n-1} is saved as a significant point.

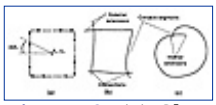


Figure 9 (a) Shape represented by equispaced vectors;(b) hand-drawn rectangle;(c) hand-drawn circle.

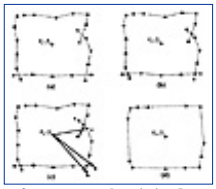


Figure 10 (a) Significant points and stroke segment input data.(b) Detection of intersection points.(c) Removal of extensions.(d) Shape after application of convex hull algorithm.

3.2.2.2 Calculation of the Center of Gravity

As explained in the section on resampling for Method1, due to the variations in the drawing speed, the points which represent the shape will not be equally spaced. In addition, the distance between significant points obtained in the previous section will exhibit significant variation in spacing. Those areas of the figure in which the rate of change in the direction of the line segments, formed by the points, is small, will result in far less significant points than in those areas in which the rate of change is high. According to the center of gravity method, the center point (x_c, y_c) is determined by equation (10) as in Method1. By considering the effects of capture speed and significant point spacing, the center of gravity calculation becomes:

$$x_c = \frac{\sum_{i=0}^{M-1} \text{length}(segment_i) \times X_{ave}(segment_i)}{\sum_{i=0}^{M-1} \text{length}(segment_i)} = \frac{\sum_{i=0}^{M-1} X_{ave}(segment_i)}{\sum_{i=0}^{M-1} \text{length}(segment_i)}$$

$$y_c = \frac{\sum_{i=0}^{M-1} \text{length}(segment_i) \times Y_{ave}(segment_i)}{\sum_{i=0}^{M-1} \text{length}(segment_i)} = \frac{\sum_{i=0}^{M-1} Y_{ave}(segment_i)}{\sum_{i=0}^{M-1} \text{length}(segment_i)} \quad (15)$$

where M is the number of line segments, and the functions length( ), X_{ave}( ) and Y_{ave}( ) return, respectively, the length, average x-value, and average y-value of a segment. As each new significant point, p_R is added to the data set, the cumulative length, cumulative X_{weight}, and cumulative Y_{weight} are updated for the segment $[p_{R-1}, p_R]$, further reducing the processing which must be carried out once input is complete.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

3.2.2.3 Residual Preprocessing of the Input

Once the strokes which define the input shape have been entered, we further process the significant point data. The center of gravity is calculated using the cumulative length, cumulative X_{weight} and cumulative Y_{weight} according to (15). Intersections between any of the segments which form the strokes that define the shape are determined and saved. Figure 10(a) presents a shape defined by its significant points and segments between consecutive points in each of four strokes. In Figure 10(b), the intersection points of the segments have been detected. To remove the extensions present in the shape, a ray is fired from the center of gravity to each significant point as is shown in Figure 10(c). If the ray intersects with any other segment, then the point lies outside of the shape and can be discarded; otherwise, the point is saved. With respect to Figure 10(c), this is correct for points a and b; however, the presence of the internal extensions d and e will result in point c in being erroneously discarded, while points d and e are saved. Taking this into consideration, the convex hull algorithm is applied to the points of the shape, points d and e are removed, and the resultant shape is shown in Figure 10(d).

3.3 Formation of Tangent Vectors Along Shape Boundary

A constant number of sample points that can sufficiently represent the shape of the boundary is necessary for input to the neural network. In order to obtain the *L* representative points along the boundary of the shape for the formation of the tangent vectors illustrated in Figure 9(a), rays are fired from the center of gravity to the boundary of the shape at an angular spacing of (2 À /*L*). However, in the case where a significant point falls between two of the equispaced rays, as in Figure 8, this may result in the loss of a corner point from a shape. To ensure that this does not occur, the *J* significant points, determined by the significant points process discussed in Sections 3.2.a and 3.2b, are merged with the ray intersection points by shifting the ray intersection points toward the nearest significant point. Once *L* representative points along the boundary are obtained, we calculate the tangent vectors $t\,v_i$ between the *i*th and the (*i* + 1)th points *i* = 0, ⋯,*L* -1. To form a closed shape, we also calculate $tv_L = \frac{y_0 - y_L}{x_0 - x_L}$. After *L* tangent vectors are obtained, we want to apply an approximation to these tangent vectors to match each vector with a quantized direction vector. The quantized direction vectors are equally spaced to be (2 À /*G*) degrees apart for any number of vectors G. In our implementation, we have chosen *G* to be equal to *L*. As *G* increases, the presence of noise on the shape becomes more accented, whereas very low values of *G* result in a significant loss in information. Figure 11

depicts quantized vector differences of boundary tangent vectors.

3.4 Fuzzy Function Filtering

We propose that the information provided by the tangent vectors be analyzed by one of the following two methods:

- **The quantized vector differences**: The difference between any two consequent tangent vectors along the boundary of the shape is the clockwise difference between the orderings of the vectors as seen in Figure 11. The difference $d_i = q\,v_d[t\,v_i, t\,v_{i+1}] = k$, where k is an integer.
- **The angle between two tangent vectors**: This method actually utilizes the outcome of the first method and obtains the angle between $t\,v_i$ and $t\,v_{i+1}$ by

$$angle[i] = qvd[tv_i, tv_{i+1}] \times (2\pi/L) \qquad (16)$$

Figure 12 depicts the angular difference between any two consequent tangent vectors of a shape. The second method is utilized as input to our fuzzy function depicted in Figure 13. Since angular differences along the tangents of the handwritten shape boundary are susceptible to noise, we devised a fuzzy function and tuned it by a heuristic analysis of a large number of sample shape boundaries. This fuzzy function helps reduce the effect of noise by attenuating the effect of small angular deviations which it introduces. It also brings out the portions of the boundary that exhibit significant changes in the internal angles between consecutive tangents.

The angle between two consecutive tangent vectors is assigned a degree of membership between 0 and 1, for each of the fuzzy sets illustrated in Figure 13. These fuzzy sets broadly classify an angle into a category such as straight, obtuse, right, acute, or reflex. For further detail, we have divided obtuse angle set into two; right-obtuse, and wide-obtuse, depending on how close the angle is to 90°. In the cases where an angle has more than one non-zero membership value, for example, right and right-obtuse, the maximum membership value determines the winner [12]. Once we have only a few meaningful categories to classify the angles into, rather than 360 individual values, we can count the frequency of their occurrence along the shape boundary. The frequency count of significant angles will be the input to the neural network.

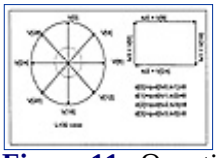


Figure 11 Quantized vector differences of boundary tangent vectors.
$q\,v_d$ = quantized vector difference function
V[i] = i-th direction vector, i = 0,..., L = 31
d[j] = j-th quantized vector difference, j = 0,⋯,M-1,
M = #tangent vectors
$t\,v\,k$ = tangent vector k, k = 0,⋯,M-1

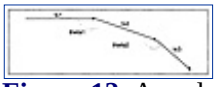


Figure 12 Angular differences between tangent vectors.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

3.5 On-line Shape Classification and Training by a BSW Network

The fuzzy function supplies the information on the detected angles. The four most significant types of angles that determine a shape (with reference to Figure 13) are wide-obtuse, right-obtuse, right, and acute. This information is supplied as input to the neural network. Formally, the input that the neural network receives is as follows:

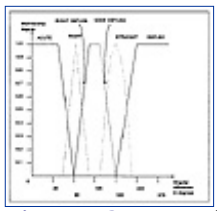


Figure 13 Fuzzy function for filtering angular differences.

$$\#atype = \sum_{i=0}^{L} \max\left(fuzzy\ membership\left(angle[i]\right)\right) \qquad (17)$$

where *angle*[*i*] is determined according to equation (16) and *#atype* represents the sum of a specific type of angle, such as #wide-obtuse, #right-obtuse, #right, or #acute, as shown in Figure 13.

The neural network's task is to find the relationships between the fuzzy filtered internal angles of a geometric shape and its desired classification, while maintaining its generalization ability for recognizing noisy shapes. The desired classifications are circular shapes (circle, ellipse), rectangular shapes (square and rectangle), and triangular shapes (any triangle). The speed with which a neural network can be trained by BSW algorithm makes it a perfect candidate for including on-line training in the system, where the user can retrain the system to classify a shape into a certain category.

In classification of shapes, the output of the fuzzy function supplies the information on the detected angles, and the input that the neural network receives is prepared as in equation (17). BSW networks accept binary input; therefore, we convert these integer values to binary digits. However, since the BSW distinguishes different patterns according to their Hamming distances, the conventional binary-radix integer representation would create conflicts, as can be observed in Table 3. In order to make the Hamming distances of patterns

equal to their Euclidean distances, we use Serial Coding, which is depicted in Table 4. In a system where *L* tangent vectors are utilized, the range of values that *#atype* can assume is 0 to (*L*-1). Therefore, each node that represents an integer valued *#atype* is expanded into L nodes after serial coding. In this experiment, 31 nodes are required for each *#atype* to be represented using Serial Coding. Therefore, the input layer consists of 31 × 4 = 124 nodes.

Table 3 Conventional Binary Coding

Integer	Binary vector	Hamming Distance From 0	Euclidean Distance From 0
0	00000	0	0
1	00001	1	1
2	00010	1	2
3	00011	2	3
4	00100	1	4
5	00101	2	5
.	.	.	.
.	.	.	.
.	.	.	.

Table 4 Serial Coding

Integer	Binary vector (Serially Coded)	Hamming Distance From 0	Euclidean Distance From 0
0	00000	0	0
1	00001	1	1
2	00011	2	2
3	00111	3	3
4	01111	4	4
5	11111	5	5
.	.	.	.
.	.	.	.
.	.	.	.

The network was initially trained with a total of 7 sample shapes drawn by mouse which are presented in Figures 14(a) through (c). The resultant network topology had three hidden nodes created during initial training, as shown in Figure 15. The initial training, set is kept small on purpose, so that the user can incrementally train the system according to his/her style of drawing. Therefore, initially, the network is given enough samples to roughly classify elliptic, rectangular, and triangular shapes. For testing, geometric shapes are again drawn by mouse and on-line recognition is performed. Four different users were given the same initial configuration and told to draw a total of 50 figures. Each set of figures was to include elliptic, triangular, and rectangular shapes. They were told that they could add samples that are not recognized by the network and train the network on-line. It was left to the discretion of the user whether the unrecognized figure should be included in the training set as a representative of that particular class of shapes. Thus, it was possible that the user may add a noisy figure which may create conflicts, such as the new input vector being the same as a previously included vector that had a different output classification. However, there is a simple verification mechanism, so that after the user adds a training sample and performs on-line training, he/she can test the network for the recognition of the newly added sample. If more than one classification is chosen, the user is informed and has the option of removing the conflicting data from the training set.

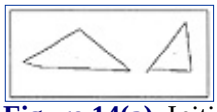


Figure 14(a) Initial training samples for triangular shapes

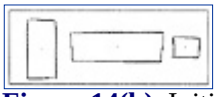


Figure 14(b) Initial training samples for rectangular shapes

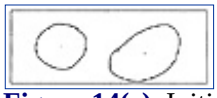


Figure 14(c) Initial training samples for circular shapes

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Table 5 displays the additions to the training set made by the four users and the resultant number of hidden layer nodes. For reference, we are including the figures drawn by users *User*1 and *User*3, in Figures 16(a),(b),(c) and 17(a),(b),(c). Also the network topology at the end of the *User3*'s session is given in Figure 18. With reference to Figures 16 and 17, the order in which the shapes were entered into the system was from left to right and top to bottom. The shapes enclosed in boxes are the additional samples which were added to the system on-line. In this experiment, none of the additional shapes input by the users caused a classification conflict. The initial network exhibited very good performance by recognizing shapes that were not in the training set and had noise along the shape boundary. Thus, in this example, our network has learned the underlying properties of the class of triangular shapes, rather than a set of specific triangles. Variety rather than the number of shapes recognized clearly illustrates this point. For the addition of a new shape to the training set, the desired category information was interactively supplied by the user and the network was retrained, which took less than 10 seconds on a 200MHz-clock PC. The resultant networks had varying configurations depending upon the input vectors of the additional training samples.

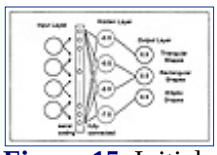


Figure 15 Initial network topology.

Table 5 Results of training sessions by four users

	Number of additional triangular training samples	Number of additional rectangular training samples	Number of additional elliptic training samples	Total number of additional training samples	Number of resultant hidden layer nodes
User1	2	4	4	10	13
User2	2	2	3	7	8
User3	3	3	1	7	8
User4	0	1	1	2	5

Figure 16(a) Rectangular shapes drawn by *User*1.

Figure 16(b) Elliptic shapes drawn by *User*1.

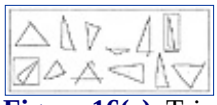


Figure 16(c) Triangular shapes drawn by *User*1.

Figure 17(a) Rectangular shapes drawn by *User*3.

Figure 17(b) Elliptic shapes drawn by *User*3.

Figure 17(c) Triangular shapes drawn by *User*3.

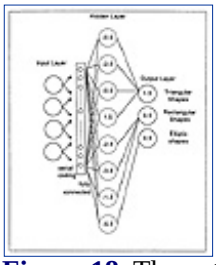


Figure 18 The network configuration after User3's session.

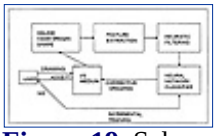


Figure 19 Schematic representation for the customizable figure recognition system Binary Synaptic Weight Neural Network).

3.6 Results

The total recognition time taken from the completion of shape input, until the corrected shape is redrawn, is less than 0.03 seconds on a Pentium PC (clock: 100MHz). The process of correction is illustrated in Figures 20(a) through (c) which present an input shape, the results of feature extraction, and the redrawn figure, respectively.

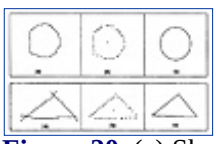


Figure 20 (a) Shape drawn by mouse;(b) Feature extraction and preprocessing results;(c) Recognized and redrawn shape.

The BSW algorithm, used for the training purposes of the neural network, was fast enough to allow us to include on-line training in the system. With this type of system, the user can customize the recognition of shapes to his/her drawing style by initially training the system with a few samples and then incrementally training as unrecognized shapes emerge in drawings. However, caution must be taken in order not to teach a shape that cannot be clearly judged to be of a specific category, as this would confuse the recognition ability of the system.

The result for the application of the presented BSW algorithm has proved that this algorithm is an effective pattern classifier. It has a number of desirable merits such as fast and guaranteed learning and it can be easily implemented in hardware. Therefore, it is recommended for use as an alternative neural network training algorithm for many problems.

4. Conclusion

In this chapter, we have presented a new method to recognize basic geometric shapes and provide on-line training through the use of feature extraction, a heuristic filtering function, and a fast training neural network. The overall structure of our current system, depicted schematically in Figure 19, takes an on-line hand-drawn shape, input mouse, and performs feature extraction, heuristic filtering, and neural network classification, presenting a corrected redrawing of the recognized shape on the output device. The process of correction is illustrated in Figures 20(a) through (c), which present an input shape, the result of feature extraction, and the redrawn figure, respectively.

The feature extraction process is important since it prepares input that is invariant in terms of scaling, translation, and rotation, which is a major problem in computer recognition of images. The heuristic function is very beneficial in reducing the noise and accenting the significant curvature maxima. The neural network brings the ability of dynamically expanding the knowledge of the system for the classification of shapes. Shape recognition by most neural network application is performed by training the system with a particular shape and trying to recognize its noisy or rotated versions rather than trying to capture the definition of that class of shapes. With this approach, our objective was to mimic the flexibility of the human eye in the recognition of three basic geometric shapes. In order words, the neural network learned the underlying definition of a category of three classes of geometric shapes in terms of their internal angles instead of learning to recognize individual shapes. The neural network, therefore, performed the task of extracting the relationship between the significant internal angles of a shape and its classification. The initial training set supplied to the neural network was only a few representative shapes for each category and the network's ability to recognize a wide variety of shapes is enhanced on-line by the user addition of new samples and retraining if necessary.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

The BSW algorithm, used for the training purposes of the neural network, provides fast training and this enabled us to include on-line training in the system. With this type of a system, the user can customize the recognition of shapes to his/her drawing style by initially training the system with a few samples and then incrementally training as unrecognized shapes emerge in drawings. However, caution must be taken in order not to teach a shape that cannot be clearly judged to be of either category, as this would result in confusing the recognition ability of the system. Our future work will involve increasing the variety of shapes which the system can manipulate.

References

1 Andree, H.M.A., et.al., (1993), "A Comparison Study of Binary Feedforward Neural Networks and Digital Circuits," Neural Networks, vol.6, pp.785–790.

2 Ulgen, F., (1992), Akamatsu, N, "A Fast Algorithm with Guarantee to Learn: Binary Synaptic Weight Algorithm on Neural Networks, " Proceedings of SIMTEC'92, Houston, Texas.

3 Pavlidis, T., (1978), "A Review of Algorithms for Shape Analysis," Computer Graphics and Image Processing, vol.7, pp.243–258.

4 Davis, L.S., (1977), "Understanding Shape: Angles and Sides," IEEE Trans. on Computers, vol.C-26, pp.125–132.

5 Montas, J., (1987), "Methodologies in Pattern Recognition and Image Analysis-A Brief Survey," Pattern Recognition, vol.20, pp.1–6.

6 Gupta, L. et. al., (1990), "Neural Network Approach to Robust Shape Classification," Pattern Recognition, vol.23, pp.563–568.

7 Perantonis, S.J., Lisboa, P.J.G., (1992), "Translation, Rotation, Scale Invariant Pattern Recognition by Higher-Order Neural Networks and Moment Classifiers," IEEE Trans. on Neural Networks, vol.3, pp.243–251.

8 Khotanzad, A., Lu, J., (1990) "Classification of Invariant Image Representations Using a Neural Network," IEEE Trans. on Acoustics, Speech and Signal Processing, vol.38, pp.214–222.

9 Guyon, I., et. al., (1991), "Design of a Neural Network Character Recognizer for a Touch Terminal," Pattern Recognition, vol.24, pp.105–119.

10 Kashyap, R.L.(1981), Chellappa, R., "Stochastic Models for Closed Boundary Analysis:

Representation and Reconstruction," IEEE Trans. on Information Theory, vol.IT-27, pp.627–637.

11 Ray, B.K., (1993), Ray, K.S., "Determination of Optimal Polygon from Digital Curve Using L1 Norm," Pattern Recognition, vol.26, pp.505–509.

12 Kosko, B., (1992), "Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence," Prentice Hall.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Chapter 2
Neural Network Approaches to Shape from Shading

Guo-Qing Wei, Gerd Hirzinger
Institute of Robotics and System Dynamics
German Aerospace Research Establishment
FF-DR/RS, DLR
82234 Oberpfaffenhofen
Germany

The multilayer feed-forward network has been used for learning a nonlinear mapping based on a set of examples of input-output data. In this chapter, we present a novel use of the network, in which the example data are not explicitly given. We consider the problem of shape from shading in computer vision, where the input (image coordinates) and the output (surface depth) satisfy a known differential equation. We use the feed-forward network as a parametric representation of the object surface and reformulate the shape from shading problem as the minimization of an error function over the network weights. The stochastic gradient and conjugate gradient methods are used for the minimization. Boundary conditions for either surface depth or surface normals (or both) can be imposed by adjusting the same network at different levels. It is further shown that the light source direction can be estimated, based on an initial guess, by integrating the source estimation with the surface estimation. The method is further extended by using the Radial-Basis-Function (RBF) network in the place of the multilayer perceptron, with increased accuracy achieved. The efficiency of the proposed methods is verified by examples of both synthetic and real images.

1. Introduction

The problem of shape from shading is to infer the surface shape of an object based on a single intensity image. Under some simplified assumptions, the surface depth function satisfies a partial differential equation [10]. Most existing solutions, except for the local methods [6, 16, 27], are based on a variational reformulation of the problem [1, 2, 7, 11, 14, 38], which usually involves a regularization term of smoothness due to the ill-posedness in the discrete solution of the problem. Horn [11] noted, however, that during the iteration, the weighting factor of the smoothness term should be gradually reduced to zero, to avoid the

algorithm from walking-away from the correct solution. Too rapid a reduction would cause instabilities (raggedness) in the surface. There is also a lower limit, in the presence of noise, on the value of the smoothness weight, beyond which the solution becomes again unstable [11, p. 65]. Since surface depth and surface gradients were treated as separate variables [1, 2, 11, 14, 38], the integrability constraint has to be imposed, by either using subspace projection [7] or adding a penalty term in the variational integrand [1, 11, 38]. In the latter case, the surface depth and surface gradients can be adjusted simultaneously [11, 38]. The weighting factor of the integrability term, however, should also be reduced [11] to an appropriate value during the iteration. It remains an open problem to determine the optimal values of both the smoothness weight and the integrability weight [34].

In this chapter, we present a new solution of the shape from shading problem based on using a multilayer feed-forward network as the parameterization of the object surface. Due to this parametric and continuous (smooth) representation, both the height (depth) and the gradients can be described by the same set of variables: the network weights. This avoids, in a natural way, the *explicit* use of both the smoothness and the integrability terms. The determination of shape from shading can then be converted to an ordinary extremum problem in which a cost function is to be minimized with respect to the network weights. This conversion also represents a new way of applying the feed-forward network: usually, a set of example data of the network input-output should be given for the learning of a nonlinear mapping [30, 33]; while in our case, the example data are not available; we only know that the network output satisfies a given *differential* equation. Thus, we are treating a more general problem than the learning of a nonlinear mapping from examples. Regarding the use of neural networks in shape from shading, we are aware of only one previous work (Lehky and Sejnowski [18]): under the framework of learning from examples, a multilayer feed-forward network was used to learn from thousands of (small) images the curvature magnitude and the curvature orientation at *one* surface point in the image center; with our method, we are able to learn from one image the complete surface. The learning is undertaken in the pixel level, instead of the picture level [18]. Here, learning also means parameter identification (of the object surface). It is further shown in this chapter that boundary conditions about surface depth and surface gradients can be learned by the same network, too. We demonstrate further how *a priori* knowledge about the surface can be utilized to estimate the illuminant direction by integrating the source estimation with the surface estimation, resulting in improvements of both the estimates. Extensions of the proposed method to the case of the RBF-network are developed, in order to increase the approximating capability in modeling complex surfaces.

The chapter is organized as follows. In Section 2, we describe the perceptron-based shape-from-shading method, assuming a known illuminant direction. In Section 3, we show how to estimate the illuminant direction by integrating with the surface estimation. We report some experimental results in Section 4. In Section 5, we extend the method to the RBF-network case, with some further experiments. In Section 6, we draw some conclusions.

2. Height and Shape Estimation

2.1 The Problem

Suppose the surface of an object is represented by $z = z(x, y)$ in a camera coordinate system x-y-z, with the x-y plane coinciding with the image plane, and the z axis coinciding with the optical axis of the camera. Assuming orthographic projection and the Lambertian reflectance model, the image intensity at position (x, y) of the image plane can then be computed as [10]:

$$I(x,y) = \eta \mathbf{n} \cdot \vec{L} = R(p,q) = \eta \frac{-pl_1 - ql_2 + l_3}{\sqrt{p^2+q^2+1}} \qquad (1)$$

where $\cdot$ is the composite albedo, $\mathbf{n}$ is the surface normal at $(x, y, z(x, y))$,

$$\mathbf{n} = (n_1, n_2, n_3)^T = \left(\frac{-p}{\sqrt{p^2+q^2+1}}, \frac{-q}{\sqrt{p^2+q^2+1}}, \frac{1}{\sqrt{p^2+q^2+1}}\right)^T \qquad (2)$$

$$p = \frac{\partial z}{\partial x}, q = \frac{\partial z}{\partial y}, \qquad (3)$$

p and q are called the surface gradients at (x, y); $\vec{L} = (l_1, l_2, l_3)$ is the illuminant direction. Equation (1) is called the image irradiance equation, which is a nonlinear partial differential equation of the first order on z; $R(p, q)$ is called the reflectance map.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Suppose in the following that the parameters · and $\vec{L}$ are known. The shape-from-shading problem can thus be stated as the determination of $p(x, y)$, $q(x, y)$, and $z(x, y)$ from a given image $I(x, y)$. Horn [11] reformulated the problem as the minimization of

$$\iint_D \{\underbrace{[(I(x,y) - R(p,q)]^2}_{\text{intensity error}} + \lambda \underbrace{(p_x^2 + p_y^2 + q_x^2 + q_y^2)}_{\text{smoothness}} + \mu \underbrace{[(z_x - p)^2 + (z_y - q)^2)]}_{\text{integrability}}\} dx\, dy \qquad (4)$$

with respect to $p(x, y)$, $q(x, y)$ and $z(x, y)$. In (4), the first term is the intensity error, the second term a smoothness measure of the surface, and the third term a penalty term of non-integrability; the parameters » and ¼ are the relative weights of the smoothness term and the integrability term, respectively. The above minimization can be performed by solving the corresponding Euler equations [3]. The finite difference method was used by Horn [11] to iteratively adjust both the height z and the gradients (p, q) on a discrete grid of points.

2.2 The Forward Network Representation of a Surface

Instead of solving for p, q, and z only at a finite number of points, we shall recover them on the continuous domain. The first step to achieve this is to represent z, and thus p and q, parametrically. Since no *a priori* knowledge about the shape of the surface is assumed available, the representation should be flexible enough to approximate *any* surface. With this criterion, we resort to the recent discovery about the multilayer feed-forward network that a 3-layer (one hidden layer) forward network whose input and output units are linear and whose hidden units are semilinear is able to approximate any real-valued continuous function over a compact set (bounded closed subset) up to any accuracy [5, 8, 13]. This justifies us to model the object surface $z = z(x, y)$ in terms of a feed-forward network as

$$z = \sum_{i=1}^{N} w_i \phi(a_i x + b_i y + c_i) + w_0 \qquad (5)$$

where N is the number of units in the hidden layer, $\{w_i, a_i, b_i, c_i\}$ are the network weights, Æ() is the semilinear function, which is the sigmoid function of the form $Æ(s) = 1/(1+ e^{-s})$ in our case, with s being a dummy

variable. Figure 1 shows the structure of a network representation of a surface by using 3 hidden units (here the bias unit, i.e., the unit with a constant input of 1, is not counted in this number.)

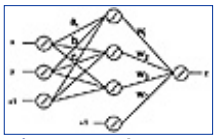


Figure 1 A network structure of size 2 × 3 × 1.

Although a 3-layer network is theoretically able to approximate any surface, it may require an impractically large number of hidden units in the case of complex surfaces [22]. It has been found, however, that an adequate solution can be obtained with a tractable network size by using more than one hidden layer [22, p. 133]. Thus, in the following, we will not limit ourselves to the case of only one hidden layer.

2.3 Solving the Image Irradiance Equation

If we rewrite (5) as

$$z = z(x, y, \vec{W}) \qquad (6)$$

where $\vec{W}$ represents the vector of all the weights, we can analytically compute the surface gradients through (3) as , $p = z_x(x, y, \vec{W})$ and $q = z_y(x, y, \vec{W})$; here, the subscripts represent the partial derivatives. Based on this parametric form, the image irradiance equation can be rewritten as

$$I(x,y) = \eta \frac{-z_x(x,y,\vec{W})l_1 - z_y(x,y,\vec{W})l_2 + l_3}{\sqrt{z_x(x,y,\vec{W})^2 + z_y(x,y,\vec{W})^2 + 1}} \qquad (7)$$

which turns out to be an equation in the network weights $\vec{W}$. Since the equation is satisfied at each image position (x, y), we get a system of equations. Due to the nonlinearity of these equations, we cannot expect the solution for $\vec{W}$ to be unique. Thus, we will just seek one of the solutions satisfying the image irradiance equations. But as long as the surface is intrinsically unique for the given problem, the different solutions for $\vec{W}$ should result in the same surface z. The uniqueness issue was discussed by Oliensis in [24, 25].

A least squares solution for $\vec{W}$ is obtained by minimizing the total intensity error E_I:

$$E_I = \sum_{i \in D_I} E_{I,i} = \sum_{i \in D_I} (I_i - R(z_x(x_i, y_i, \vec{W}), z_y(x_i, y_i, \vec{W}))^2 \qquad (8)$$

where D_I is the index set of all image points; (x_i, y_i) are the image coordinates at pixel i; I_i is the corresponding image intensity, and $E_{I,i}$ is the intensity error there. According to (8), the estimation of shape from shading has been converted to an ordinary extremum problem in which only a finite number of parameters $\vec{W}$ are to be determined. Note that it is not a *functional* minimization problem anymore (compare with (4)), and there is no smoothness term nor integrability term in it.

To minimize (8), we recall the process in learning an input-output mapping using a multilayer network [33]. The analogy motivates us to use the stochastic gradient method [20] to *learn* the weights. Note that in learning an input-output mapping by neural networks [30, 33], the surface values z_i's at coordinates $\{(x_i,y_i)\}$ should be assumed known. The purpose there is to build a network which maps the coordinates $\{(x_i, y_i)\}$ to the given values $\{z_i\} = \{z\ (x_i, y_i)\}$. In our problem, however, we are required to *determine* the z_i's, knowing only that they satisfy a *differential* equation. If still viewed in the mapping framework, our network is to map the (x_i, y_i)'s onto a surface which generates the given image intensities I_i's. Note also that an explicit one-to-one correspondence between a surface value z_i and the intensity value I_i does not exist physically. In terms of the neural networks terminology, we will call the pairs $\{(x_i, y_i; I_i), i \in D_I\} = \{\mathcal{T}_{I,i}\} = \mathcal{T}_I$ the *intensity training patterns*. According to the stochastic gradient method, we repeatedly present the training patterns to the network and modify the network weights after each presentation of a training pattern $\mathcal{T}_{I,i}$, in

the direction opposite to the gradient of the corresponding error $E_{I,i}$. That is, the change of the weights after the presentation of $\mathcal{T}_{I,i}$ is made by

$$\Delta \vec{W} = -\beta \frac{\partial E_{I,i}}{\partial \vec{W}} = \beta (I_i - R_i)(R_p z_{x\vec{W}} + R_q z_{y\vec{W}}) \qquad (9)$$

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

where ² is the learning rate, R_i is the computed intensity at position i using the current weights, R_p = ðR(p, q)/ðp, R_q = ðR(p, q)/ðq, $z_{x\vec{W}} = \partial^2 z/\partial x \partial \vec{W}$, and $z_{y\vec{W}} = \partial^2 z/\partial y \partial \vec{W}$ are quantities all evaluated at position i using the current weights. Similar to [33], we can also introduce a momentum term ± to avoid possible oscillations. That is, the weights will be adjusted according to

$$\vec{W}(n+1) = \vec{W}(n) - \beta \frac{\partial E_{I,i}}{\partial \vec{W}(n)} + \alpha \Delta \vec{W}(n-1) \qquad (10)$$

where n indexes the number of presentations. Typical values for ² and ± in our experiment are ² = 0.3 and ± = 0.6, respectively. When the iteration is getting closer to the solution, however, the value of ² should be reduced. We automatically reduce it by 5% of the current value whenever the total error E_I computed by accumulating the individual errors $E_{I,i}$'s shows any oscillations.

We will in the following, also call the image irradiance equation, when it is to be satisfied at a particular point, the *intensity constraint* at that point, for the convenience of coupling it with boundary conditions.

2.4 Boundary Conditions

We know that a partial differential equation possesses multiple solutions. To single out a specific individual solution, additional conditions have to be added to the differential equation. Prescribing the values of the solution on the boundary of the region of interest (the image boundary) is one way to constrain the solution. In the variational formulation of the shape from shading problem, one specifies the boundary condition by giving gradient values p and q on the image boundary [11] (more exactly, on the boundary of the region of integration in the functional). When nothing is known about the boundary gradients, the *natural boundary condition* has to be assumed, which, together with the Euler equations, serves as the necessary conditions for the functional minimization [3, p. 208]. The natural boundary condition, however, appears as an artifact of the variational reformulation of the original shape from shading problem (which is a first-order partial differential equation), since different variational formulations of the same shape from shading problem may require different natural boundary conditions [3]¹In our formulation of the shape from shading problem, the imposition of a natural boundary condition is not necessary. Besides, any other *a priori* knowledge about the surface can be used to constrain the solution. These constraints can be either known depth values or known

gradient values *anywhere* in the image (not necessarily on the image boundary). Compared with the *initial values* of the characteristic curves used in the solution of a first-order partial differential equation [4, 10, 11], our constraints are more flexible, since we do not require both the depth and the gradients to be given at the same image points. This property also facilitates the integration of shape from shading with stereo vision where usually only sparse depth, not necessarily also the surface normals, is available.

[1]In Zheng and Chellapa's formulation [38], a different natural boundary condition from that in Horn [11] should be derived. They simply used the same natural boundary condition as Horn [11] did.

Despite the above discrepancies, we will, for convenience, still call our constraints boundary conditions. In the following, we will also specialize them by using either the term *gradient constraints* or the term *depth constraints*, according to whether the gradient or depth is given.

In the case of gradient constraints, suppose we are given the surface gradients $\{(p_j, q_j)\}$ at a set of points $\{(x_j, y_j)\}$ indexed by the set D_g. We form the total gradient error E_g as

$$E_g = \sum_{j \in D_g} E_{g,j} = \sum_{j \in D_g} ((n_{1,j} - \hat{n}_{1,j})^2 + (n_{2,j} - \hat{n}_{2,j})^2 + (n_{3,j} - \hat{n}_{3,j})^2) \quad (11)$$

where $E_{g,j}$ is the surface gradient error at position j. The vectors $\mathbf{n}_j = (n_{1,j}, n_{2,j}, n_{3,j})^T$ and $\hat{\mathbf{n}}_j = (\hat{n}_{1,j}, \hat{n}_{2,j}, \hat{n}_{3,j})^T$ are the given and the computed surface normals at position j, respectively; the latter is a function of the network weights (refer to (6), (3), and (2)).

In the case of depth constraints, suppose we are given depth values $\{z_k\}$ at a set of points $\{(x_k, y_k), k \in D_d\}$ with D_d as the index set. The total depth error is defined as

$$E_d = \sum_{k \in D_d} E_{d,k} = \sum_{k \in D_d} (z_k - \hat{z}_k)^2 \quad (12)$$

where $E_{d,k}$ is the depth error at position k, $\hat{z}_k$ is the computed depth at position k by (6).

The exertion of boundary conditions is then equivalent to minimizing (8), (11), and (12) simultaneously with respect to the network weights. If we treat both the gradient constraints and the depth constraints as a new kind of training pattern, we can then adjust the network weights, based on $\left\{\frac{\partial E_{g,j}}{\partial \vec{W}}\right\}$ and $\left\{\frac{\partial E_{d,k}}{\partial \vec{W}}\right\}$, in a similar way to that in (10) of the last section.

Due to the unknown range of the depth error, which is usually much larger than that of the intensity error (we normalize the intensity values to the range [0,1]), we have to introduce a weighting factor w_d to the depth error, so that the weight adjustment due to the depth constraints would not overwhelm the contributions from the intensity constraints. We determine the weight by

$$w_d = \sum_{i \in D_I} E_{I,i} / \sum_{k \in D_I} E_{d,k} \quad (13)$$

The value of w_d thus determined is usually underestimated. Thus, during the training, we increase w_d by 10% of its current value, whenever the total depth error shows a tendency to increase. The same weighting process can be used for the gradient constraints. But since we have used the surface normals (which are of the range [0, 1]) instead of the original gradients for the training, a constant weighting factor of 2 has been found to work well.

2.5 Computational Steps

We have seen that the network weights can be learned from three kinds of constraints: intensity constraints, gradient constraints, and depth constraints. Figure 2 shows the complete training diagram. Thanks to the feed-forward structure of the surface representation, we are able to compute the gradients $\frac{\partial E_{I,i}}{\partial \vec{W}}$, $\frac{\partial E_{g,j}}{\partial \vec{W}}$, and $\frac{\partial E_{d,k}}{\partial \vec{W}}$ in a similar (but not the same) way as in the back-propagation algorithm [33] (see [36] for

details).

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

The following summarizes the steps in the learning of network weights in the presence of all the introduced constraints.

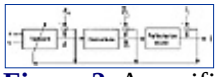


Figure 2 An unified training framework.

1.) Collect all the intensity constraints, the gradient constraints and the depth constraints to form the intensity training patterns $\mathcal{T}_I = \{(x_i, y_i; I_i)\}$, the gradient training patterns $\mathcal{T}_g = \{(x_j, y_j; p_j, q_j)\}$, and the depth training patterns $\mathcal{T}_d = \{(x_k, y_k; z_k)\}$. The entire training patterns are $\mathcal{T} = \mathcal{T}_I \cup \mathcal{T}_g \cup \mathcal{T}_d$.

2.) For all the intensity patterns in $\mathcal{T}_I$, repeat the network training using the stochastic gradient method, until the total intensity error drops below a prescribed value.

3.) For all the training patterns in $\mathcal{T}$, repeatedly do the network training, until the total error, as the sum of the intensity error, the weighted gradient error and the weighted depth error, ceases to decrease.

The initial network weights are set by random values in [-0.5, 0.5]. Step 2 is to provide a good initial surface for imposing boundary conditions. For images with singular points (see the definition in the next section), most parts of the surface can be established before adding boundary conditions [24, 25]. If only boundary gradient constraints are present, i.e., no boundary depth constraints, we can ignore Step 2, since we don't have to determine the weighting factor of the gradient constraints dynamically.

When the iteration gets closer to the solution, the conjugate gradient method [32] can be switched on to speed up convergence and to improve accuracy. But it is not advisable to use the conjugate gradient method at the very beginning of Step 2 or 3, since the process may get into local minima. The stochastic gradient method, however, has been found to rarely get stuck in local minima [33].

2.6 Miscellaneous

In this subsection, we address some issues related to input-output normalization, the multiresolution implementation, and the choice of network size.

Normalization: To make the range of the network weights independent of image size, we normalize the network input (image coordinates) to the range [0, 1] through

$$\tilde{x} = x/N_x; \quad \tilde{y} = y/N_y \tag{14}$$

where N_x and N_y are the image size in the x and y directions, respectively. Using the normalized coordinates $\tilde{x}$ and $\tilde{y}$, the quantities $z_{\tilde{x}}, z_{\tilde{y}}, z_{\tilde{x}\vec{W}}$, and $z_{\tilde{y}\vec{W}}$ computed using (5) (see [36] for details) for the weight adjustment should be denormalized to give

$$z_x = \frac{\partial z}{\partial \tilde{x}} \cdot \frac{\partial \tilde{x}}{\partial x} = z_{\tilde{x}}/N_x z_y = \frac{\partial z}{\partial \tilde{y}} \cdot \frac{\partial \tilde{y}}{\partial y} = z_{\tilde{y}}/N_y \tag{15}$$

$$z_{x\vec{W}} = z_{\tilde{x}\vec{W}}/N_x, z_{y\vec{W}} = z_{\tilde{y}\vec{W}}/N_y. \tag{16}$$

Notice that if we use the sigmoid instead of the linear function in the *output* layer, as is usual in the learning of an input-output mapping [33], there should be a normalization and denormalization of the network output z, and accordingly, a modification of (15) and (16). In that case, however, we should also know *a priori* the depth range of the true surface, so that the normalized depth matches the output range [0, 1] of the sigmoid function.

Multiresolution: The multiresolution scheme has been used in the solution of many computer vision problems for the purpose of reducing computational cost. Through subsampling of the original image, the calculation is first done in the lowest level of resolution. Then one propagates the solution to a higher lever as an initial state of the computation at that level. For the shape from shading problem, one propagates the depth and gradients by an appropriate interpolation scheme [38]. With our neural network method, the interpolation is automatic due to the continuous solution. We subsample the original image as usual, but keep the values of the sampled image coordinates the same as in the original image. Then we normalize the sampled coordinates by the original image size. The training is done as usual. When we transfer from a lower resolution to a higher one, we simply add new training patterns. No change has to be made to the existing network in the transition process.

Network size: Due to the reason stated in Section 2.2, we choose to use two hidden layers instead of one. As for the size of each hidden layer, we have no analytical methods for its determination. As an alternative, we employ a heuristic approach. First, an initial size for the two hidden layers is estimated as $N_1 = N'_x/3$ and $N_2 = N'_y/3$, where N'_x and N'_y are, respectively, the subsampled (reduced) image size in the x-and y-directions, chosen so that the image does not lose details. This estimation is empirical and subjective. We denote this network by $\mathcal{N}_0$ and will later refer to it as the *basis network*. After the training of $\mathcal{N}_0$, we obtain a surface $\mathcal{S}_0$ represented by $z_0(x, y, \vec{W}_0^*)$, where $\vec{W}_0^*$ is the converged weights of $\mathcal{N}_0$. If the rms residual error $e_0^{(I)}$ of the image intensity of $\mathcal{S}_0$ is less than a predefined value, $\mathcal{S}_0$ is regarded as the solution $z(x, y) = z_0(x, y, \vec{W}_0^*)$. Otherwise, we build another *refining network* $\mathcal{N}_1$ to improve the surface estimate as:

$$z(x, y) = z_0(x, y, \vec{W}_0^*) + z_1(x, y, \vec{W}_1) \tag{17}$$

where $z_1(x, y, \vec{W}_1)$ is the surface represented by $\mathcal{N}_1$, and $\vec{W}_1$ is the weight vector of $\mathcal{N}_1$. To determine $\vec{W}_1$, we use the surface representation (17) to solve the image irradiance equation again by the stochastic gradient method. The size of $\mathcal{N}_1$ should be larger than that of $\mathcal{N}_0$ to ensure the increased approximating ability of (17). We set the size of $\mathcal{N}_1$ as being, say, 20% larger than that of $\mathcal{N}_0$. The role of $\mathcal{N}_0$ is to reduce the computational cost in the training of $\mathcal{N}_1$. Since $z_0(x, y, \vec{W}_0^*)$, and thus $z_{0x}(x, y, \vec{W}_0^*)$ and $z_{0y}(x, y, \vec{W}_0^*)$ are fixed quantities for each pixel, they can be prestored in a table. By setting the initial output-layer weights of $\mathcal{N}_1$ to zero, the starting surface in the adjustment of $\vec{W}_1$ is exactly $\mathcal{S}_0$ (refer to (17) and (5)). During the training of $\mathcal{N}_1$, we adjust the weights only for the pixels whose intensity errors are larger than the rms intensity error of the current surface, which is $e_0^{(I)}$ for the first iteration. This kind of

filtering keeps only around 20% of all the pixels in consideration. Although the sigmoid function is global, its derivatives are relatively local, which are the quantities involved in the weight adjustment for the intensity patterns. This ensures the stability of the above training scheme for $\mathcal{N}_1$. This surface refinement can be continued until the desired accuracy is reached.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

3. Estimating the Illuminant Direction

In the last section, we assume the direction of the light source to be known prior to the surface estimation. In practice, this illuminant direction can only be roughly estimated before the surface estimation, since most source estimation algorithms [16, 28, 38] are based on assumptions about *statistical* properties of the surface. Without the use of additional constraints, we could not expect the image alone to allow for an accurate estimation of the illuminant direction. This can be understood by noting that even when the exact illuminant direction is given, we are not guaranteed to arrive at a unique surface estimate. So when both are unknown, the problem is even more difficult. Notice also that there is an inherent two-way ambiguity (concave/convex) in the combination of the surface shape and the illuminant direction [2].

In this section, we will show how to use some *a priori* knowledge about the surface, e.g., boundary conditions, together with a rough guess of the illuminant direction to arrive at an accurate estimate of the illuminant direction. We consider especially the case in which the *a priori* knowledge alone, e.g., boundary depth, is unable to provide such an accurate estimate of the illuminant direction, but could be integrated with our surface estimation procedure to achieve this. We present two methods of this kind.

The first method is based on representing the illuminant direction by the surface normal at a singular point (the brightest point). At singular points, the surface normals have the same orientation as the illuminant direction. Suppose the surface gradient at a singular image point (x_s, y_s) is (p_s, q_s). The illuminant direction can then be expressed as

$$\vec{L} = (l_1, l_2, l_3)^{\mathrm{T}} = \frac{1}{\sqrt{p_s^2 + q_s^2 + 1}}(-p_s, -q_s, 1)^{\mathrm{T}} \qquad (18)$$

where

$$p_s = \frac{\partial z(x_s, y_s, \vec{W})}{\partial x}, q_s = \frac{\partial z(x_s, y_s, \vec{W})}{\partial y} \qquad (19)$$

Inserting (18) and (19) into (7), we get an equation with the network weights as the only unknowns:[2]

[2]Here we assume that the albedo · has been estimated by using either the singular point or existing methods [38].

$$I(x,y) = \eta \frac{z_x(x,y,\vec{W})z_x(x_s,y_s,\vec{W}) + z_y(x,y,\vec{W})z_y(x_s,y_s,\vec{W}) + 1}{\sqrt{z_x^2(x,y,\vec{W}) + z_y^2(x,y,\vec{W}) + 1}\sqrt{z_x^2(x_s,y_s,\vec{W}) + z_y^2(x_s,y_s,\vec{W}) + 1}}. \quad (20)$$

Equation (20) allows us to use the same learning method as in the last section to determine the network weights, and thus, both the object surface and the illuminant direction. We first use an initial guess of the illuminant direction to recover only the surface through minimizing the intensity error (8), in the same manner as in Step 2 of Section 2.5. This establishes an initial surface. Then, a new intensity error is formed, based on (20), as the sum of the squared residuals of (20) evaluated at each image position. This new intensity error together with the errors of the boundary conditions (11) and/or (12) are then minimized with respect to the network weights, in a similar way to Step 3 of Section 2.5. We call this method of source estimation the singular point method.

The second method is based on the alternative adjustment of the surface normals and the illuminant direction [2]. First, we establish an initial surface in the same way as in the singular point method. Then, the obtained surface is used to solve for the illuminant direction $\vec{L}$ in terms of (7), which can be done in linear computation. With the obtained illuminant direction, we adjust the network weights (the surface) for one cycle in terms of both the residual error of (7) and the errors of the boundary conditions. The obtained surface is then used to re-estimate the illuminant direction. This process repeats until the convergence is reached. We call this estimation process the iterative LS method.

The two methods have been found to perform complementarily. The singular point method has a wider range and a quicker rate of convergence, while the iterative LS method is less sensitive to noise and does not depend on the identification of a singular point. Thus we propose to combine the two methods if a singular point is available. We can first use the singular point method to quickly find a rough solution and then use the iterative LS method to refine the estimate.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

4. Experiments

4.1 Examples

Example 1: Test of accuracy. The first example is a synthetic sine surface (Figure 3(a)), $z = -2cos(\pi\sqrt{x^2 + y^2}/32)$, with the origin at the image center. To specify the illuminant direction $\vec{L}$, we employ the two-parameter description: the tilt angle Ä and the slant angle ³ [38]. Figure 3(b) is a synthetic image of size 32 × 32 with the illuminant direction being at Ä = 30° and ³ = 10°. The image was generated by orthographic projection, and no noise was added. In this and the next example, we used a network of size 2 × 10 × 10 × 1. We also compared our method with Horn's [11] and Leclerc and Bobick's [15] algorithms under different boundary conditions. Figure 3(c) shows the recovered surface by our algorithm without any boundary conditions, after 1500 cycles of gradient searches plus 100 cycles of conjugate gradient searches. Figure 3(d) shows the improved surface by adding *boundary depths* (known depth values along the image boundaries) to Figure 3(b), after 800 more cycles of gradient searches and further 200 cycles of conjugate gradient searches. Here, one cycle means a complete pass of the training patterns. Figures 3(e) and (f) show, respectively, the recovered surfaces, under the same boundary condition, by Horn's method after 10380 cycles of iterations (with » being reduced from 1.0 to 0.0001 and ¼ set to 0.1), and by Leclerc and Bovick's method after 1547 cycles of conjugate gradient searches. The average error, deviation, and maximum error of the surface depth in pixel units, and of the surface normals in degrees for the three methods are shown in Table 1. Figures 3(g), (h), (i) show the respective error surfaces, magnified by a factor of 4. Table 1 also lists the depth errors and normal errors for our method and for Horn's method under *boundary gradient* constraints. (The method of Leclerc and Bovick cannot consider boundary gradients.) Under only boundary gradient constraints, the absolute depth error has no meaning, since the surface can only be determined up to a shift. To investigate the noise sensitivities of the methods, we added ± 5% random noise to the image intensities. The results are also shown in Table 1. From the table,[3]we can see that our method is able to reconstruct a more accurate and more stable surface than the others are.

[3]For Horn's and Leclerc & Bovick's algorithms in the presence of image noise, we have picked up the best results from those with different stopping values of the smoothness weight », i.e., different lower bounds of ».

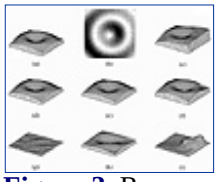


Figure 3 Recovery of a rotated sine surface: (a) true 3-D surface; (b) the input image; (c) recovered surface by our algorithm without boundary conditions; (d) recovered surface by our algorithm with boundary depth; (e) recovered surface by Horn's algorithm with boundary depth; (f) recovered surface by Leclerc and Bobick's algorithm with boundary depth; (g) our error surface; (h) Horn's error surface; (i) Leclerc and Bobick's error surface.

Table 1 The accuracy comparison of different methods under different boundary conditions

Methods		Boundary depth		Boundary gradient	
		depth error	norm. error (°)	depth error	norm. error (°)
		av.,dev.,max.	av.,dev.,max.	av.,dev.,max.	av.,dev.,max.
Ours	0% noise	0.038,0.032,0.272	1.2,1.3,17.3	-,0.064,-	1.3,0.9,5.9
	5% noise	0.093,0.081,0.357	3.1,1.9,15.8	-,0.125,-	2.6,1.4,6.7
Horn's	0% noise	0.196,0.264,0.700	3.5,4.5,26.2	-,0.278,-	1.1,1.3,17.1
	5% noise	0.482,0.476,2.163	5.2,3.4,29.6	-,0.481,-	3.1,1.72,14.2
L. & B.'s	0% noise	0.249,0.372,1.472	6.5,5.9,31.7	—	—
	5% noise	0.472,0.545,2.240	8.1,7.2,35.6	—	—

Example 2: Normal discontinuities and the integrability weight. The second example is a concave spherical cap behind a plane (Figure 4(a)). The sphere radius is 15 pixels, and the circle radius 14 pixels at the plane of intersection. This surface contains discontinuous surface normals. Figure 4(b) is a synthetic image of size 32 × 32 with the illuminant direction at Ä = 150° and ³ = 10° (no image noise). Figures 4(c) and (d) show, respectively, the recovered surfaces of Horn's and Leclerc & Bovick's algorithms without boundary conditions. Figures 4(e)-(h) show, respectively, the sequence of surfaces reconstructed by our algorithm (also without boundary conditions) after 150, 300, and 800 cycles of gradient searches and 600 more cycles of conjugate gradient searches. We can see from this sequence how the surface builds itself. It begins from the area with the highest contrast and then propagates toward areas of weaker contrast. The same has been observed with Horn's and Leclerc & Bovick's algorithms. This experiment demonstrates that the surface propagation in our algorithms in the presence of normal discontinuities is more efficient than that in the other algorithms.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

From Figure 4, it can be seen that our method recovers the correct surface even without boundary conditions (for this example). With Horn's method (and also with Leclerc & Bovick's one), only when boundary conditions (either boundary normals or boundary depth) are added, can the correct solution be recovered. The value of the integrability weight ¼ in Horn's method, however, should be chosen carefully. In this example, we set ¼ = 1 and keep it unchanged during the iteration. To see the influence of the integrability weights on the solution in the presence of boundary conditions, we tried to set ¼ = 0.1. In the case of boundary depth, we still get the correct solution; whereas in the case of boundary gradients, we get a solution which is similar to that without boundary conditions. This means that without an appropriate choice of the integrability weight (and an appropriate way of reducing it, as Horn [11] proposed to do so), the solution may deviate from the correct one.

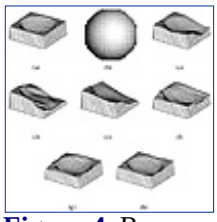


Figure 4 Recovery of a concave spherical cap behind a plane: (a) the true 3-D surface; (b) the image; (c) recovered surface by Horn's algorithm without boundary conditions; (d) recovered surface by Leclec and Bobick's algorithm without boundary conditions; (e)-(h) sequences of surfaces recovered by our algorithm without boundary conditions.

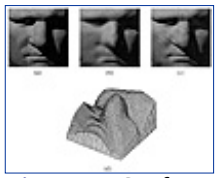


Figure 5 Surface recovery of the Agrippa statue: (a) the image; (b) the learned image by the basis network; (c) the learned image by adding a refining network; (d) the recovered 3-D surface.

Example 3: Real images. Here, we present results of our algorithm on two real images. The first image is the Agrippa statue, see Figure 5(a). The image size is 128 × 128. The illuminant direction is manually estimated to be Ä = 135° and ³ = 50°. The network size is chosen as 2 × 20 × 20 × 1. We use 3 levels in the

multiresolution training. No boundary conditions are assumed to be known. Shown in Figure 5(b) is the learned image by the network. It can be seen that near the lip, the learned image (and therefore the corresponding 3-D structure) is too smooth.[4] This is because of both the inappropriate choice of the network size and the relatively small size of the lip in the whole image: a small network can only learn the rough structure. To remedy this, a refining network of size 2 × 25 × 25 × 1 is added to the basis network in terms of the strategy described in Section 2.6. The refining network successfully learned the fine structure. The learned image and the recovered 3-D surface are depicted in Figures 5(c) and (d), respectively. Figure 6(a) is another real image, the David statue. The illuminant direction and the initial network size are the same as that for the Agrippa statue. Similarly, the basis network fails to recover the details of the eye, see the learned image in Figure 6(b). Adding a refining network of size 2 × 25 × 25 × 1 successfully recovers the fine structure. Figures 6(c) and (d) show, respectively, the learned image and the recovered 3-D shape of the David statue. It was also found that in the initialization of the refining network, setting the initial weights of each layer (except for the output layer) to random values in the same range as the corresponding layer of the basis network can speed up convergence.

[4]It is important to point out that with Horn's algorithm [11], this correspondence of smoothness between image and 3-D structure is not true if » and ¼ were not appropriately chosen. This means that it is not always possible to evaluate the reconstructed 3-D shape by merely checking the reconstructed 2-D image.

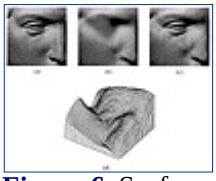


Figure 6 Surface recovery of the David statue: (a) the image; (b) the learned image by the basis network; (c) the learned image by adding a refining network; (d) the recovered 3-D surface.

Example 4: Source estimation. We tested our source estimation methods in Section 3 on the synthetic image of the sine surface in Figure 3(b) with various initial guesses of the illuminant direction. We intensively investigated the case of boundary depth. The case of boundary gradients is trivial, since the boundary normals together with the corresponding image intensities are sufficient to determine the illuminant direction in a closed form (see (1). We found that the singular point method converges within a deviation range of 40° (even 50°) from the correct illuminant direction, whereas the iterative LS method converges within a range of 30°. For both methods, the illuminant direction can be estimated within 0.5°.

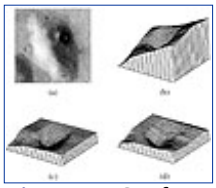


Figure 7 Surface recovery on a moon image: (a) the image; (b) the recovered surface by our algorithm without boundary conditions, under Zheng & Chellapa's Illuminant direction; (c) the improved surface by our algorithm in terms of partial boundary normals and a re-adjustment of the illuminant direction; (d) the recovered surface by our algorithm without boundary conditions, under our estimate of the illuminant direction.

Next, we show an example on a real image. Figure 7(a) is a sub-image of the moon. The image size is 128 × 128. Since there is no *a priori* knowledge about the illuminant direction, we use the method of Zheng and Chellapa [38] to obtain an estimate. This gives Ä = 26° and ³ = 27°. Figure 7(b) shows the recovered surface by our algorithm without using any boundary conditions under this estimated illuminant direction. The recovered surface is a little bit oblique. This is due to the errors in the estimate of the illuminant direction. Nevertheless, we can improve both the estimates of the surface and the illuminant direction by using the knowledge that the normals at the upper and right boundaries of the image are toward the viewer. Shown in Figure 7(c) is the improved surface by our singular point method. The surface is more consistent with the human perception. The improved estimate of the illuminant direction is at Ä = 14.10° and ³ = 51.20°. To see whether the errors in Figure 7(b) are really due to the erroneous illuminant direction, we use our estimate of the illuminant direction to re-estimate the surface by dropping the boundary conditions. The resulting surface is shown in Figure 7(d). The improvement over Figure 7(b) can be clearly seen.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

4.2 Several Comments

1) Multiple Solutions: In the absence of any boundary conditions, there will be at least two solutions to a shading cue, which correspond to concave/convex surfaces with opposite illuminant directions (in the image plane); for example, Figure 4(b) may also be perceived as being convex with the illuminant direction at Ä = -30°. When the illuminant direction is known (or approximately known, as is in our source estimation), however, this two-way ambiguity does not exist, except that the illuminant direction coincides exactly with the viewing direction. (Almost all shape-from-shading computations assume the known illuminant direction.) Other solution ambiguities, which are intrinsic to differential equations, should be resolved by boundary conditions; for example, the cheek of David could appear either as being toward or as being away from the viewer without violating the intensity constraint if no boundary conditions were applied (refer to Figure 6).;

2) Local Minima: This deals with the ability of the gradient search algorithms to find the correct solution of the problem, i.e., the global minima of the cost function. When the learning rate ² and the weighting factor w_d of the depth constraint are chosen appropriately (e.g., ² d 0.3), our gradient algorithm has not been found to get stuck into local minima. Local minima might occur if ² and w_d were set too large. With the conservative estimate of w_d in Section 2.5, we get lower convergence rates, but not local minima. Concerning the conjugate gradient algorithm in the aspect of local minima, see Section 2.5.

3) Computational Complexity: This depends on the image size and the network size. A complex surface may require a larger network than a simple surface does. As an example, the learning of the sine surface of Example 1 in the case of gradient constraints took 15 minutes in a SGI workstation. (Horn's method took 5 minutes, but with reduced accuracy.) For complex surfaces like the Agripa statue, our algorithm takes longer time in finding the fine structure, due to the globalness of the sigmoid function. This problem can be dealt with by using locally tuned radial basis function (RBF) networks (to be in the next section).

5. Extension to the Case of RBF Network

Due to the globalness of the sigmoid function in the depth representation, high-frequency components of the depth surface become difficult to model. According to Section 2.6, the network may have to grow to a very large size in order to represent a complex surface. This makes the computational cost hard to bear. In this section, we use the radial basis function in place of the multilayer perceptron to derive a more flexible method of shape from shading.

5.1 The Method

The whole framework for the solution of the shape from shading problem remains unchanged. The RBF network has the same property of universal approximation [26]. There are several types of radial basis functions used in interpolations [31]. We choose the Gaussian radial basis function. With Gaussian radial basis functions, the surface depth $z(x, y)$ can be modeled as

$$z = \sum_{k=1}^{N} w_k \phi(\mathbf{x}, \mathbf{t}_k, \mathbf{s}_k) + w_0 \qquad (21)$$

where N is the number of Gaussian functions, and $\mathbf{x} = (x, y)$; parameters $\mathbf{t}_k = (t_{x,k}, t_{y,k})$, $\mathbf{s}_k = (\tilde{A}_{x,k}, \tilde{A}_{x,k})$, and w_k are, respectively, the center, width (size), and the weight of the k-th Gaussian function:

$$\phi(\mathbf{x}, \mathbf{t}_k, \mathbf{s}_k) = e^{-\left(\frac{(x-t_{x,k})^2}{\sigma_{x,k}^2} + \frac{(y-t_{y,k})^2}{\sigma_{y,k}^2}\right)} \qquad (22)$$

Each Gaussian has a receptive field, approximately at $[x \quad t_{x,k} \pm 3\tilde{A}_{x,k}, y \quad t_{y,k} \pm 3\tilde{A}_{y,k}]$, beyond which its contribution can be ignored. This finite support property, in contrast to the sigmoid functions, is the very property we need for the incremental estimation introduced later.

If we use the vector $\vec{W} = (\{\mathbf{t}_k\}, \{\mathbf{s}_k\}, \{w_k\})$ to represent all the Gaussian parameters in the specification of depth z, a similar set of equations as (6) and (7) can be obtained. However, due to the locality of the Gaussian neurons, a slight different cost function from (8) will be used:

$$\begin{aligned} E_{\mathrm{IS}} &= E_{\mathrm{I}} + \lambda E_{\mathrm{S}} = \sum_{i \in D_{\mathrm{I}}} E_{\mathrm{IS},i} \\ &= \sum_{i \in D_{\mathrm{I}}} \{[I_i - R(z_x(x_i, y_i, \vec{W}), z_y(x_i, y_i, \vec{W}))]^2 \\ &\quad + \lambda[S_1(x_i, y_i) z_{xx}^2(x_i, y_i, \vec{W}) + 2S_2(x_i, y_i) z_{xy}^2(x_i, y_i, \vec{W}) \\ &\quad + S_3(x_i, y_i) z_{yy}^2(x_i, y_i, \vec{W})]\} \end{aligned} \qquad (23)$$

where E_{I} and E_{S} are the intensity error and the smoothness error, respectively; » is a global smoothness weight; D_{I} is the index set of all the image points; and $\{S_j(x, y), j = 1, 2, 3\}$ are local smoothness weights:

$$S_1(x, y) = (1 - |I_x(x, y)|)^2 \qquad (24)$$

$$S_2(x, y) = (1 - \frac{\sqrt{2}}{2}(|I_x(x, y)| + |I_y(x, y)|)^2 \qquad (25)$$

$$S_3(x, y) = (1 - |I_y(x, y)|)^2 \qquad (26)$$

where the partial derivatives I_x and I_y are computed by finite differences. The intuition for choosing the local smoothness factors as above is based on the fact that smoother images (corresponding to smaller I_x's and I_y's) should have been produced by smoother surfaces (corresponding to larger S_j's), assuming no albedo variations.

Although the smoothness constraint is used, in comparison to the case of perceptron, it can be eventually dropped out without causing instabilities in the solution. Due to image noises, we fix the global smoothness weight at » = 0.1. The introduction of the locally varying smoothness weights $\{S_j(x, y), j = 1, 2, 3\}$ is for more accurate shape recovery; a single global weight tends to treat surface parts of different smoothness the same. Note that here we do not have to normalize the image coordinates, as we did for the multilayer perceptron.

We can also apply the prior knowledge to the solution of shape from shading by minimizing error functions similar to (11) and (12). For the gradient constraints, the error function is:

$$\begin{aligned} E_{\mathbf{g}_{\mathrm{S}}} &= E_{\mathbf{g}} + \lambda E_{\mathrm{S}} = \sum_{j \in D_{\mathrm{n}}} E_{\mathbf{g}\mathrm{S},j} \\ &= \sum_{j \in D_{\mathrm{n}}} [\|\mathbf{n}_j - \hat{\mathbf{n}}_j\|^2 + \lambda(S_1 z_{xx}^2 + 2S_2 z_{xy}^2 + S_3 z_{yy}^2)|_j] \end{aligned} \qquad (27)$$

where S_1, S_2, and S_3 are the same as in (8); symbol $|_j$ means the computation performed at the j-th point [refer

to (8)]. For the depth constraints, the error function is:

$$E_{dS} = E_d + \lambda E_S = \sum_{k \subset D_d} E_{dS,k}$$
$$= \sum_{k \in D_d} [(z_k - \hat{z}_k)^2 + \lambda(S_1 z_{xx}^2 + 2S_2 z_{xy}^2 + S_3 z_{yy}^2)|_k] \quad (28)$$

Equations (8), (27), and (28) are minimized in the same way as in the perceptron case. Experiment indicates that in the case of RBF network, we do not have to compute the weight w_d as in (13); it can be simply set at 1.0.

It was further shown that qualitative knowledge can also be used to constrain the same from shading solution [37].

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

One problem remains to be clarified; that is, how to choose the initial values of the Gaussian parameters. In interpolation problems, the starting values for $\{\boldsymbol{t}_i, \boldsymbol{s}_i, w_i\}$ in the iteration can be either obtained by analyzing the given 3-D data [9, 21], or just set to random values. In shape from shading, 3-D data are not available; and the use of random values was found to be inefficient both in computations and in reconstructing a reasonable surface. Here, we employ a heuristic approach. We let the initial centers of the Gaussians regularly distributed in the image plane and let the initial widths cover different scales. This is similar in style to the use of hierarchical basis functions [34] or scale-space [35]. However, the centers and widths (scales) in our case are subject to continuous adjustment in accordance with the image data. This makes the number of scales very small in comparison to that in [35].

Suppose $2^M \times 2^M$ is the image size. We set the maximum number of levels in the hierarchy to be H, e.g., $H = M - 1$. The number of Gaussians in hierarchy h is chosen to be $2^h \times 2^h$, $h = 1, 2, .., H$. Within a hierarchy, we set the initial positions $\{\mathbf{t}_i\}$ to be equally spaced in the image plane, and the initial widths to be equal to the spacing; that is, at hierarchy h, the initial spacing and width are $2^M / 2^h$ in both the x and y directions. The initial weights of all the Gaussians are set to zero.

Similar to the incremental increase of the network size in the perceptron case, the same can be done with the RBF network. Due to the local support of the radial basis function, more efficient schemes can be devised. First, the Gaussians at the lowest hierarchy $h = 1$ are used to minimize the errors of the objective functions. After the convergence at the current hierarchy, we add Gaussians at the higher hierarchy. For each new Gaussian to be added, e.g., the i-th one, we check the average intensity error e_I, depth error e_d, and normal error e_g within its *main* receptive field defined as $[x \quad t_{x,i} \pm 1.5\tilde{A}_{x,i}, y \quad t_{y,i} \pm 1.5\tilde{A}_{y,i}]$. If all the errors e_I, e_d, and e_n are below the respective thresholds, the i-th Gaussian will not be created. When new Gaussians are generated, we do not freeze the old ones; that is, the old Gaussians are still subject to further adjustment. This avoids possible local minima, according to our experience. The incremental scheme spares lots of unnecessary Gaussians (up to 50%).

5.2 Further Examples

In Section 4, we gave several examples to show the validity of the perceptron-based method. For images of more complex objects, the perceptron-based method is inefficient; one may have to use a very large network for the representation. In this section, we show that by using a RBF network, there is no limitation on the complexity of the objects to reconstructed.

Figure 8 shows the results of the RBF network-based method on the David image (size 64 × 64). We use two levels of image resolutions, which correspond to the image sizes of 32 × 32 and 64 × 64, respectively. Five hierarchies of Gaussians are used, which are distributed over the two image resolutions. At the 32 × 32 image resolution, hierarchies of Gaussians corresponding to the initial positioning in the forms of 2 × 2, 4 × 4, 8 × 8, and 16 × 16 are generated; while at the 64 × 64 image resolution, only the hierarchy containing Gaussians initially spaced in the form of 32 × 32 is generated (see Section 4.1). As has been mentioned, the initial Gaussians at each hierarchy are equally spaced in the *original* 64 × 64 image plane. Using the incremental scheme described in Section 4, a total of 691 Gaussians have been generated after 455 cycles of iterations when convergence is reached. (This takes about 10 minutes on an Indigo2 Silicon Graphics workstation; the running time could be reduced by about a factor of 5 with a looser convergence criterion, without affecting the accuracy too much.) Figure 8(b) shows the reconstructed image, and Figure 8(c) depicts the distribution of the Gaussians after convergence. It can be easily seen that at smoother area, the Gaussians are sparely distributed; while at structured area, the Gaussians are densely distributed. The Gaussians have been deformed both in positions and in shapes. The recovered 3-D shape illuminated from another source direction is shown in Figure 8(d), and the 3-D plot in Figure 8(e).

In this and the following examples, we have kept the smoothness weight » fixed at 0.1. We found that after convergence, the smoothness term could be entirely removed for further iterations without destroying the result, seemingly due to the presence of global Gaussians which stabilize the solution; that is, the result is insensitive to the lower limit of the smoothness weight. This is not the case with variational approaches. In Horn's method [11] (and in others), removing the smoothness term (i.e., setting » = 0) may cause raggedness in the recovered surface, even though the intensity error remains at a very small level (results not shown here).

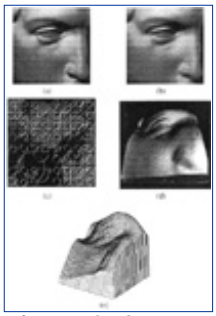


Figure 8 An example of our parametric shape-from-shading method: (a) the input image (David); (b) the reconstructed image; (c) the equal-height contours at 90% of the maxima of each Gaussian; (d) the reconstructed 3-D shape illuminated from another direction.

Figure 9 The left Agrippa image: (a) the input image; (b) the reconstructed surfaces by the RBF network

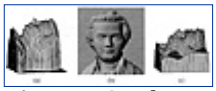


Figure 10 The Mozart image: (a) the 3-D ground truth; (b) the input image; (c) the reconstructed surfaces by RBF network.

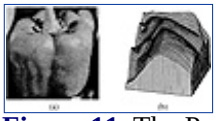


Figure 11 The Pepper image: (a) the input image; (b) the reconstructed surfaces by RBF network, viewed from Northwest.

The results on the Agrippa statue by using the RBF network are shown in Figure 9. For both the David and the Agrippa images, the RBF network produced more accurate results than the perceptron. Two further examples are given in Figures 10 and 11.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

6. Conclusions

We have in this chapter presented a new way of applying the multilayer perceptron and the RBF network. For the first time, we use these networks as a general parameterization tool to solve the partial differential equation in shape from shading. An attractive feature of such a solution is that *a priori* knowledge of different kinds (e.g., depth and normals) can be integrated into the shape-from-shading computation in a unified computational scheme. Two new methods for the estimation of the illuminant direction are also presented. Experiments have verified the efficiency of our method in both the surface and the source estimations.

Acknowledgment

The authors are very thankful to Dr. Q. Zheng at the University of Maryland, Mr. P.S. Tsai at the University of Central Florida, and Dr. K.M Lee previously at the University of Southern California for providing test images. Dr. Zheng and Dr. Hiesh, the latter at the Academia Sinica, Taiwan, kindly provided their original Fortran- and C-codes. Dr. Nelson Max of the University of California, Davis, read the manuscript and made helpful comments.

References

1 Brooks, M.J., Horn, B.K.P., (1986), "The variational approach to shape from shading," *Computer Vision, Graphics and Image Processing*, Vol.33, pp. 174-208.

2 Brooks, M.J., Horn, B.K.P., (1985), "Shape and source from shading," *Proc. Int. Joint Conf. Artificial Intelligence*, Los Angeles, pp.932-936.

3 Courant, R., Hilbert, D., (1953), *Methods of Mathematical Physics*, Vol.I, Interscience Publishers, New York.

4 Courant, R., Hilbert, D., (1962), *Methods of Mathematical Physics*, Vol.II, Interscience Publishers, New York

5 Cybenko, G., (1989), "Approximation by superposition of a sigmoidal function," *Mathematics of Control, Signals and Systems*, Vol.2, pp.303-314.

6 Ferrie, F.P., Levine, M.D., (1989), "Where and why local shading analysis works," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 11, No.2, pp. 198-206.

7 Frankot, R.T., Chellapa, R., (1988), “A method for enforcing integrability in shape from shading algorithms,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 10, No.4, pp.439-451.

8 Funahashi, K.-I., (1989), “On the approximate realization of continuous mappings by neural networks,” *Neural Networks*, Vol.2, pp. 183-192.

9 Goshtasby, A., O’Neill, W.D., (1994), “Curve fitting by a sum of Gaussians,” *CVGIP: Graphical Models and Image Processing*, Vol.56, No.4, pp.281-288, 1994.

10 Horn, B.K.P., (1975), “Obtaining shape from shading information” in: P.H. Winston, Ed., *The Psychology of Computer Vision*, McGraw Hill: New York, pp. 115-155.

11 Horn, B.K.P., (1990), “Height and gradient from shading,” *Int. Journal of Computer Vision*, Vol.5, No. 1, pp.37-75.

12 Horn, B.K.P., Brooks, M.J., (Eds), (1989), *Shape From Shading*, MIT Press, Cambridge, MA.

13 Hornik, F., (1989), “Multilayer feedforward networks are universal approximators,” *Neural Networks*, Vol.2, pp.359-363.

14 Ikeuchi, K., Horn, B.K.P., (1981), “Numerical shape from shading and occluding boundaries,” *Artificial Intelligence*, Vol. 17, pp. 141-184.

15 Leclerc, Y.G., Bobick, A.F., (1991), “The direct computation of height from shading,” *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, Hawaii, pp.552-558.

16 Lee, C.-H., Rosenfeld, A., (1985), “Improved methods of estimating shape from shading using the light source coordinate system,” *Artificial Intelligence*, Vol.26, pp. 125-143.

17 Lee, K.M., Kuo, C.C.J., (1993), “Shape from shading with a linear triangular element surface model,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 15, No.8, pp.815-822.

18 Lehky, S.R., Sejnowski, T.J., (1988), “Network model of shape-from-shading: neural function arises from both receptive and projective fields,” *Nature*, Vol.333, June, pp.452-454

19 Livesley, R.K., (1983), *Finite Elements: An Introduction for Engineers*, Cambridge University Press.

20 Ljung, L., Söderström, T., (1983), *Theory and Practice of Recursive Identification*, MIT Press, Cambridge, MA.

21 Moody, J., Darken, C.J., (1989), “Fast learning in networks of locally-tuned processing units,” *Neural Computation*, Vol. 1, pp.281-294.

22 Hecht-Nielsen, R., (1990), *Neurocomputing*, Addison-Wesley Publishing Company.

23 Norrie, D.H., de Vries, G., (1978), *An Introduction to Finite Element Analysis*, Academic Press, Orlando, FL.

24 Oliensis J., (1991), “Uniqueness in shape from shading,” *Int. J. Computer Vision*, Vol.6, No.2, pp.75-104.

25 Oliensis, J., (1991), “Shape from shading as a partially well-constrained problem,” *Computer Vision, Graphics and Image Processing*, Vol.54, No.2, pp. 163-183.

26 Park, J., Sandberg, I.W., (1991), “Universal approximation using radial-basis-function networks,” *Neural Computation*, Vol.3, pp.246-257.

27 Pentland, A.P., (1984), “Local shading analysis,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.6, pp. 170-187.

28 Pentland, A.P., (1982), “Finding the illuminant direction,” *Journal of Optical Society of America A*, Vol.72, No.4, pp.448-455.

29 Poggio, T., Torre, V., Koch, C., (1985), “Computational vision and regularization theory,” *Nature*, Vol.317, pp.314-319.

30 Poggio, T., Girosi, F., (1990), “Networks for approximation and learning,” *Proc. IEEE*, Vol.78, No.9, pp. 1481-1497.

31 Powell, M.J.D., (1987), “Radial basis functions for multivariate interpolation: A review,” In *Algorithms for Approximation* , J.C. Mason, M.G. Cox, Eds., Oxford, pp. 143-167.

32 Press, W.H. et al., (1992), *Numerical Recipes in C: The Art of Scientific Computing*, 2nd edition, Cambridge University Press.

33 Rumelhart, D.H., Hinton, G.E., Williams, R.J., (1986), “Learning internal representation by error propagation,” In D.E. Rumelhart, and J.L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol.I, pp.318-362, MIT Press, Cambridge, MA.

34 Szeliski, R., (1991), “Fast shape from shading,” *Computer Vision, Graphics and Image Processing:*

Image Understanding, Vol.53, No.2, pp. 125-153.

35 Jones, A.G., Taylor, C.J., (1994), "Robust shape from shading," *Image Vision Computing*, Vol. 12, No.7, pp.411-421.

36 Wei, G.Q., Hirzinger, G., (1996), "Learning shape from shading by a multilayer network," *IEEE Trans. Neural Networks*, Vol. 17, No.4, pp.985-995.

37 Wei, G.Q., Hirzinger, G., (1997), "Parametric shape-from-shading by radial basis functions," *IEEE Trans. Pattern Analysis and Machine Intell.*, Vol. 19, No.4, pp.353-365.

38 Zheng, Q., Chellapa, R., (1991), "Estimation of illuminant direction, albedo, and shape from shading," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 13, No. 7, pp.680-702.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Chapter 3
Neural Networks and Fuzzy Reasoning to Detect Aircraft in SAR Images

A. Filippidis
Land Operations Division, Defence Science Technology Organisation
P.O. Box 1500, Salisbury SA 5108, Australia

L.C. Jain
Knowledge-Based Intelligent Engineering Systems Centre
University of South Australia, Adelaide
The Mawson Lakes SA 5095, Australia

N.M. Martin
Weapons Systems Division, Defence Science Technology Organisation
P.O. Box 1500, Salisbury SA 5108, Australia

Fuzzy reasoning through fuzzy rule combination is used to improve the accuracy of the automatic detection of aircraft in Synthetic Aperture Radar (SAR) images using *a priori* knowledge derived from color aerial photographs. A combination of the Fuzzy Automatic Target Recognition system with neural networks is used in this chapter to fuse object identity attribute data derived from a SAR and a Color Aerial photo image. The images taken by the two different sensors have quite different resolutions and are taken at different times. The aim of this study is to automatically detect ground-based aircraft in the SAR image with greater certainty. Using the fusion techniques, we have correctly detected five of the six aircraft (83% accuracy rate) and reduced the number of false alarms from 40 to 8 when compared to the output of the background discrimination algorithm.

1. Introduction

The aim of this chapter is to investigate the use of a fuzzy rule-based fusion technique to automatically detect targets, such as ground-based aircraft, in a SAR image with greater certainty by using all available

information possible. To improve the results, we have made use of some *a priori* information in the form of an aerial photographic image taken of the background (i.e., taxiways, buildings, roads etc.), a few months before the SAR image was taken.

Automatic Target Recognition (ATR) involves extraction of critical information from complex and uncertain data for which the traditional approaches of signal processing, pattern recognition, and rule-based artificial intelligence (AI) techniques have been unable to provide adequate solutions [1]. Target recognition of one or more fixed signatures in a stationary background is a straightforward task for which numerous effective techniques have been developed [1]. If the target signatures and the background are variable in either a limited or known manner, more complex techniques such as using rule-based AI (e.g., expert systems) methods can be effective. However, rule-based AI systems exhibit brittle rather than robust behavior (i.e., there is great sensitivity to the specific assumptions and environments). When the target signatures or backgrounds vary in an unlimited or unknown manner, the traditional approaches have not been able to furnish appropriate solutions [1].

A number of ATR systems, based on various methods, have been developed and tested on a very limited data set, and good classification performance has been reported [2, 3]. However, in practice, these efforts have been only partially successful and have produced high false-alarm rates. Some of the key reasons for this are the nonrepeatability of the target signature, competing clutter objects having the same shape as the actual targets, experience with a very limited database, obscuration of targets, and a limited use of *a priori* information.

Neural network technology provides a number of tools, which could form a basis for a potentially fruitful approach to the ATR problem. For ATR, one needs methods to recognize targets and backgrounds that are both sufficiently descriptive yet robust to signature and environmental variations. One also needs the ability to adapt the procedure for additional targets and environments. The existence of powerful learning algorithms is one of the main strengths of the neural network approach. ATR needs to construct a compact set of maximally discriminating target features. In the latter sections of this chapter, these features will be described as target identity attributes. We demonstrate in this chapter that ATR performance can be enhanced by use of *a priori* knowledge about target signatures and backgrounds, in this case *a priori* knowledge of aircraft size and the type of ground surfaces on which aircraft were more and less likely to be found.

Some current target detection techniques on SAR images have been using background discrimination algorithms described in references [4] and [5]. A fusion system is presented which improves the detection process by using all available information to improve target detection and reduce false alarm rates. Figure 1 shows a block diagram of the improved ATR fusion system used in our experiments. It consists of a preprocessing stage which derives four identity attributes (A, B, C, D) of the aircraft targets from a current (synthetic aperture radar) SAR image (aircraft positions shown in Figure 2) and the red, green, and blue bands of a color image (as in Figure 3) of the same scenery (but different targets) taken a few months before the radar image. Image I1 (Figure 4) is the output of preprocessing using neural network NN1 classifier and the image shown in Figure 5, is the output of the background-discrimination algorithm [4, 5]. Neural network NN2 classifies the red, green, and blue bands (which are registered to the SAR image) to derive identity attributes C and D, which represent *a priori* knowledge on the likely locations of ground-based aircraft.

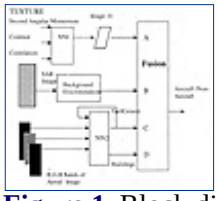


Figure 1 Block diagram to show the fusion of identification attribute information in SAR and color aerial images to identify aircraft targets.

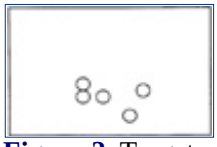


Figure 2 Target positions in circles in a SAR Image. [Registered to Optical.]

Figure 3 Aerial image (red band) of the same area as the SAR image taken at a different time.

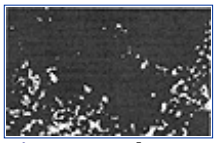


Figure 4 The output of neural network NN1 image I1.

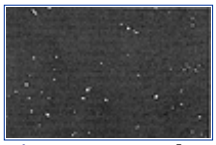


Figure 5 Results of the background discrimination algorithm showing man-made targets derived from the SAR image.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

2. Multilayer Perceptron for Classification

2.1 Structure

A multilayer perceptron (MLP) used in the ATR architecture of Figure 1 (i.e., NN1 and NN2) is considered to be a feed-forward network constructed of interconnected nodes that perform a nonlinear function of the weighted sum of their inputs.

A single node's output, x', is defined by

$$x' = f(w^T \mathbf{x}) \qquad (1)$$

where f is a nonlinear function, $\mathbf{x}$ is the augmented state vector which includes a constant component plus all inputs x' to the node:

$$\mathbf{x} = (1, x_1, \ldots x_i, \ldots x_N)^T$$

and w is the weights vector for this node:

$$\mathbf{w} = (w_0, w_1, \ldots w_i, \ldots w_N)^T$$

where w_i is the weight applied to the ith input and w_0 acts as a threshold for this node. A typical node is depicted in Figure 6 [6, 7].

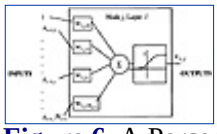


Figure 6 A Perceptron node [6].

The nonlinear function f in this case is the sigmoid, and the network is trained using the gradient descent method known as back-propagation.

$$f(x) = \frac{1}{1+e^{-x}} \quad (2)$$

The basic network structure of an MLP comprising a feed-forward network with each node performing the function described in equation (1) is shown in Figure 6. For L layers, and J_l nodes in each layer, the output of the jth node in layer l becomes

$$x_{lj} = f\left(\mathbf{w}_{lj}^{T}.\mathbf{x}_{l-1}\right) \quad (3)$$

In this, $\mathbf{x}_0$ is the augmented vector of inputs; that is, the threshold is included as one of the components of the input vector.

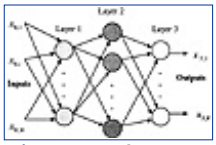


Figure 7 An MLP example with 3 active layers (or 2 hidden layers, L = 3) [6].

2.2 Classification and Training

Classification in the artificial neural network context can be defined as a process that gives an output belonging to a specified class when a pattern is presented to the input of the network. The general form of the output is a multicomponent continuous-valued vector. The output of an MLP may be interpreted as a probability function [8], and in our case in the ATR architecture, this value is fed in as a fuzzy value into the fusion process shown in Figure 1.

Experiments described in this work use back-propagation (BP) for adapting weights. The MLP has been defined as a special case of feed-forward network. Back-propagation will be used as a method for determining the error signal which provides information on how each weight contributes to the output error, and generalized gradient descent will be used to describe the way in which the error signal is used to adjust the weights [6, 9].

For an L layer network, training proceeds by applying an augmented input vector x_0 to the input of the network, and comparing the resultant output vector x_L with the required output vector d. Weights are adjusted to minimize a cost function $E(W)$ relating the errors at the output of the MLP to the values of the internal parameters W. A typical, but not exclusive, definition of E is the L^2 norm of the output error summed over all training patterns [6]:

$$E(W) = \sum_{p\in P}\left\{E_p(W)\right\} = \sum_{p\in P}\left\{\sum_{i=1}^{M}\left(x_{Li}^{p} - d_i^{p}\right)^2\right\} \quad (4)$$

where $x_{L,i}^{p}$ is the ith output of the MLP when presented with training pattern p, d_i^{p} is the corresponding desired output, and P is the set of all training patterns.

It can be shown that by using a recursive procedure (BP) [10], a weight update rule due to Widrow and Hoff shown in reference [11], could be extended to a generalized delta rule which gives adjustments for weights in hidden layers. The simple delta rule is given by:

$$\Delta w_{ji} = \beta\sum_{p\in P}\left(d_j^{p} - x_{Lj}^{p}\right)x_{0i}^{p} \quad (5)$$

where ”w_{ji} is the change made to the weight on the connection from the ith node in layer L-1 to the jth node in layer L, x_{0i}^{p} is the ith element of the input pattern p, and ’ is a carefully chosen gain factor, not necessarily constant during training. This rule (simple delta) requires that the error contributions from all patterns be measured before a weight update is made and was devised for networks with no hidden layers.

While the Widrow-Hoff rule [11] requires that the weight adjustments are made with respect to $E(W)$, provided ’ is small, gradient descent can be approximated closely by making adjustments with respect to $E_p(W)$. This allows a weight adjustment after presentation of each pattern and simplifies the requirements for local memory at each node. Hence, the simplified delta rule becomes [12]

$$\Delta_p w_{ji} = \eta\left(d_j^p - x_{Lj}^p\right)x_{0i}^p = \eta\delta_{Lj}^p x_{0i}^p \qquad (6)$$

where $\Delta_p w_{ji}$ is the change made to the weight from the *i*th to the *j*th node after presentation of the training pattern *p*, · is a small constant gain value factor, and $\delta_{Lj}^p = d_j^p - x_{Lj}^p$ is called the error signal. There is considerable discussion about the validity of this approximation among researchers.

For an MLP with a nonlinear activation function (e.g., sigmoid), it is desirable to at least approximate gradient descent in weight space with an equally uncomplicated weight rule.

Consider the intermediate value *y* (the input to the activation function) formed by the weighted sum within an MLP node. The forward transfer function of node *i* in layer *l* can be rewritten (dropping the *p* superscript for convenience):

$$y_{li} = \sum_{j=0}^{N_{l-1}} w_{lij} f\left(y_{l-1j}\right) \qquad (7)$$

where *f* is the activation function, w_{lij} are the weights from node *j* in the (*l*-1)th layer to node *i* in the *l*th layer, and N_{l-1} is the number of nodes in layer (*l*-1). The error function at the output layer can be written:

$$E(w) = \sum_{i=0}^{N_L}\left(f(y_{Li}) - d_i\right)^2 \qquad (8)$$

Suppose the values y_{ij}, for all *i*, are inputs to a truncated network starting at the *l*th layer. Then, the sensitivity of the network to the input y_{ij} is given by $\frac{\partial E}{\partial y_{li}}$. Moreover, if we know $\frac{\partial E}{\partial Y_{L+1i}}$ for all *i*, we have [6]

$$\frac{\partial E}{\partial y_{li}} = \sum_{j=0}^{N_L} \frac{\partial E}{\partial y_{l+1i}} \frac{\partial y_{l+1i}}{\partial y_{li}}$$

$$\frac{\partial E}{\partial y_{li}} = \sum_{j=0}^{N_L} \frac{\partial E}{\partial y_{l+1i}} w_{l+1ji} f'\left(y_{li}\right)$$

$$\frac{\partial E}{\partial y_{li}} = f'\left(y_{li}\right)\sum_{j=0}^{N_L} \frac{\partial E}{\partial y_{l+1i}} w_{l+1ji} \qquad (9)$$

Let us defined

$$\partial_{li} = \frac{\partial E}{\partial y_{li}}$$

$$\partial_{li} = f'\left(y_{li}\right)\sum_{j=0}^{N_L} \partial_{l+1j} w_{l+1ji} \qquad (10)$$

Since w_{l+1ji} directly influences input yij into the truncated network [6], we have

$$\frac{\partial E}{\partial w_{l+1\,ji}} = \frac{\partial E}{\partial y_{li}} \frac{\partial y_{li}}{\partial w_{l+1\,ji}}$$

$$= \delta_{li} f\left(y_{li}\right)$$

$$= \delta_{li} x_{li} \qquad (11)$$

For output layer nodes, calculation of the error signal follows by taking

$$\frac{\partial E}{\partial x_{Lj}} = \left(d_j - x_{Lj}\right)$$

hence,

$$\delta_{Lj} = \left(d_j - f\left(y_{Lj}\right)\right) f'\left(y_{Lj}\right) \qquad (12)$$

For a network in which each node has a sigmoid activation function, calculating the derivatives gives the error signal for output layer node *j*:

$$\delta_{Lj} = \left(d_j - x_{Lj}\right) x_{Lj} \left(1 - x_{Lj}\right) \qquad (13)$$

and the error for a node *j* in an arbitrary layer *l* is given by recursive calculation (error back-propagation):

$$\delta_{lj} = x_{lj} \left(1 - x_{lj}\right) \sum_{i=1}^{N_{L+1}} \delta_{l+1i} w_{l+1ij} \qquad (14)$$

While it is common to use a sigmoidal activation function, any monotonic increasing, continuously differentiable function would be desirable [6].

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

An untrained network has all weights set to randomized values. To learn from the presentation of a known pattern, we adjust the weights into node *i* of layer *l* according to the generalized delta rule:

$$\Delta w_{lij} = \eta \delta_{li} x_{l-1\,j} \qquad (15)$$

where · is the "learning rate." The value of the learning rate must be chosen empirically as a compromise between speed of learning and probability of instability. Because we are going to allow a weight update after each pattern presentation, it is desirable to add a smoothing term to prevent single points from having undue inluence on the weights, giving

$$\Delta w_{lij}(n) = \eta \delta_{li} x_{l-1\,j} + \alpha \Delta w_{lij}(n-1) \qquad (16)$$

where ± is a term known as "momentum" and *n* is a counter, incremented after each weight adjustment. Again, it is necessary to select ± and · empirically, and they are generally held constant during training [6].

2.3 Classification of Multispectral Images

We address the task of supervised classification of the multispectral (red, green, and blue bands) processed by the recurrent neural network NN2 (4 input, 6 hidden layer, and 2 output nodes are used) and the texture measures derived from the SAR image processed by the feed-forward network NN1 (3 input, 6 hidden layer, and 1 output node used) shown in Figure 1. The recurrent network NN2 feeds the output of the tar/cement classification back into the input of the network. Large homogeneous areas in multispectral images such as the tar/cement areas of the color aerial image have been shown to produce slightly higher accuracy rates (as indicated in reference [6]) when using a recurrent network. Buildings accounted for a smaller and more fragmented portion of the image and hence a recurrent loop input from the buildings output of NN2 would not have resulted in any benefit in accuracy.

The aim of the neural network classifiers is to classify every pixel in the given image into an appropriate class (i.e., tar/cement, buildings, aircraft, etc.). The input to NN2 in Figure 1 is an image comprised of data from a number of spectral bands (indicated by the multilayer block in Figure 8) and the limited knowledge of a (human) supervisor. The supervisor identifies a number of known areas to be used as templates for classifying the whole image. The sets of reference data (training sets) taken from the known areas are used

to generate parameters which characterize the classes (e.g., mean red pixel value or particular texture measure in a 5×5 pixel window). These parameters are then used to classify the whole image. Feed-forward neural network classifier NN1 uses a 5x5 pixel window to process the SAR image producing the three input texture statistics, second angular momentum, contrast, and correlation along every pixel of the image to produce preprocessed image I1 shown in Figure 1. The recurrent neural network NN2 also uses a 5×5 pixel window incremented one pixel at a time calculating the three input mean color pixel values from the red, green, and blue color bands shown in Figure 1.

The fusion algorithm processes five images simultaneously, incrementing through columns in each row one pixel at a time with three different size pixel windows (8×8, 9×9, and 5×5). The center four pixels of the largest size window (9×9) moving across the SAR image (using the background discrimination algorithm) and second largest window moving across the SAR processed image I1 (in Figure 1) lies inside the same position as the smallest size windows (5×5) moving across the red, green, and blue bands of the aerial image. The algorithm processes image I1 (refer to Figure 1) by first calculating the area of an 8×8 pixel window, only if there are at least four pixels at the center of the window (i.e., number of white pixels connected together in the horizontal and vertical directions starting from the center four pixels in the 8×8 pixel window over the entire image). This value is input into the fusion algorithm as attribute “A” shown in Figure 1. From *a priori* knowledge on aircraft size, we know that aircraft pixel size can range from 4 to 22 pixels. The fusion algorithm is simultaneously processing the 9x9 pixel background discrimination window which has to be larger than the size of the targets (approximately fitting into 5x5 pixels window). This output in this case is fed into the fusion algorithm as attribute “B.” Again simultaneously, the mean red, green, and blue pixel values calculated from three more 5x5 pixel windows produce target identity attributes “C” and “D” (in the fusion algorithm) from the outputs of the recurrent classifier NN2.

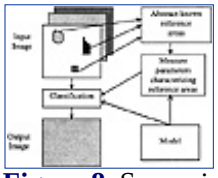


Figure 8 Supervised classification [6].

2.4 Texture for Target Detection

Texture is an important property in detecting targets in SAR images. The texture properties and gray tone in an SAR image are mostly independent properties [18]. In the fusion process, these SAR texture and gray tone properties of the aircraft targets are combined and fed into the fusion algorithm as identity attributes “A” and “B” (as shown in Figure 1). For this reason, even if the target is camouflaged in gray tone, it is difficult to camouflage these properties and the texture simultaneously [18]. There is a large array of texture measures in the literature. The choice of texture features should be based on the application, and on the nature of the images used.

One of the drawbacks of texture feature is that it cannot be used for detecting small targets in general. The object should cover at least a 30 window; in our case, the aircraft targets fit in a 5×5 pixel window.

There are four types of texture measures or features in use [18]: a) co-occurrence, b) fractal, c) Markov random field, and d) multi-channel filter. In reference [17], Ohanian and Dubes have compared the performance of these texture features in classifying objects and found that co-occurrence (spatial dependence) features performed best, followed by fractal dimension measure. Co-occurrence features are very widely used by researchers; they were originally presented by Harlick in reference [15], where he describes 28 features. Some of the more popular are second angular momentum, contrast, correlation, energy, and entropy.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

3. Fuzzy Rule-Based Fusion Method

The classification output of NN1 shown in Figure 4 and the results of the background discrimination algorithm of Figure 5 show that all aircraft targets positions were detected but also produced a large number of false alarms. By combining this information as attributes “A” and “B” in the fuzzy fusion algorithm (refer to Figure 1), together with *a priori* information obtained in attributes “C” and “D,” we have reduced the false alarm rate considerably. The thesis in reference [14] showed that the *fuzzy* rule-based fusion method of combination compared to Dempster’s rule, and the super Bayesian techniques met all the 10 desirable properties of a combination technique such as generality, robustness, convexity, symmetry, complementation, and certainty test, to name a few.

We have implemented a fusion equation [14] which has identified five of the six targets with eight false alarms as shown in Figure 12. The novelty in this method is that it will be used to solve a complex fusion problem involving attributes derived from a current image (SAR), and a past image (optical) taken at a different time to the SAR, together with *a priori* information. The differences of both SAR (9.375 GHz) and optical images is not only their frequencies, resolutions, and time, but also the fact that SAR has the ability to pick up targets at night and during overcast days. Hence the ATR fusion system can operate at night and during overcast days, detecting targets fusion system can operate at night and during overcast days, detecting targets using the *a priori* optical images information obtained at an earlier date to help reduce the false alarm rate.

The implementation of the fuzzy rule-based system [14] is divided into three stages: (1) fuzzification of the real data obtained from the SAR and optical images, (2) fusion, and (3) defuzzification. In this experiment, an SAR image together with three bands (red, green, and blue) of a color image are used.

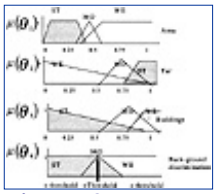


Figure 9 Four membership functions for the four inputs (area, NN2 output based on tar/cement spectral signature, NN2 output based on building spectral signature, and the output from a background discrimination

algorithm) are used in the rule-based fuzzy fusion method. They represent the likelihood of a target (strong, moderate or weak) as the 8×8 window moves across the SAR image and the red, green, and blue color bands.

Fuzzification or modeling uncertainty is the first component in our data fusion system. Generally, the process of fuzzification is a mechanism by which one can transform a non-fuzzy set to a fuzzy set by assigning to each subset in the universe of discourse a fuzzy value. In the fuzzification problem, we are dealing with the combination of four attributes, based on (1) *a priori* information on the targets size, (2) target pixel intensity with respect to the surrounding pixels using a background discrimination windows moving across the five images (refer to Figure 1, i.e., SAR image, target texture segmented image, and red, green, and blue bands), we assign the variable ¸ to measure the strength of the attribute being a target or not. The interpretation of a given attribute is greatly influenced by our *a priori* knowledge, experience, and common sense in identifying the aircraft target.

We can characterize an aircraft target from the strength (normalized within the interval [0,1]) of the four attributes shown in Figure 9. The assigned fuzzy values are: (1) weak likelihood of attribute being an aircraft target (WE), (2) moderate likelihood (MO), and (3) strong likelihood of attribute being an aircraft target (ST). These fuzzy values characterize weak, moderate, and strong portions of the universe of discourse (aircraft target). Figure 9 shows membership functions for the fuzzy subsets of four universes of discourse. The values ¸1, ¸2, ¸3, and ¸4 represent the strength of the attribute representing an aircraft target derived from an aerial image (red, green, and blue bands), and the SAR and preprocessed texture SAR images as shown in Figure 1.

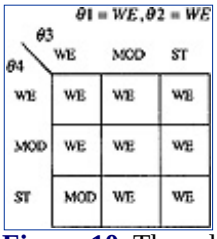


Figure 10 The rule-based matrix shows how 9 of the rules are derived. Rule-based matrices, are used to produce the 81 rules. ¸1, ¸2, ¸3, and ¸4 represent the area, tar/cement, buildings, and background discrimination identity attributes as shown in Figure 1.

Eighty-one rules were derived from 9 rule-based matrices; 9 of the rules are shown in Figure 10. The generated rules must be enough to construct a complete inference system for managing uncertainty. For instance, rule1 derived from the top left-hand row of the matrix is as follows:

$$\theta_1 \oplus \theta_2 \oplus \theta_3 \oplus \theta_4 = \{(\mathrm{WE,WE,WE,WE}) \rightarrow \mathrm{WE}\} = \text{an element of } \theta$$

This is interpreted as if the area, tar/cement, building, and back-discrimination attributes derived from the 8x8 pixel window (moving across all 5 bands as shown in Figure 1) are all weak, then the likelihood of an aircraft target is weak. The character "•" represents the combination of the strengths of the membership values to make up the support for the particular rule derived from the rule-based matrices.

The data fusion function [14] (refer to Figure 11) has been used to combine identity attribute information derived from sensors which have obtained their information from current (SAR) and past images (optical) together with *a priori* information to successfully piece together the likelihood of an aircraft target.

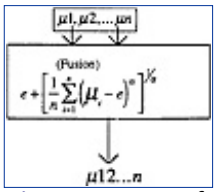


Figure 11 Data fusion function.

Previous | Table of Contents | Next

HOME SUBSCRIBE SEARCH FAQ SITEMAP CONTACT US

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Let ˜ be the sample space which contains a collection of events; i.e., ˜ = {¸}, and assume that ¼$_1$(¸), ¼$_2$(¸), . . . , ¼$_n$(¸) are degrees of truth over ˜ committed to an event ¸ in ˜. The fusion technique described here will handle any piece of evidence with a high efficiency of computation [14]. The combined evidence ¼(¸) for *n* pieces of evidence is given by ¼(¸) = *f*[¼$_1$(¸), ¼$_2$(¸), . . . , ¼$_n$(¸)].

In the equation of Figure 11, let ¼$_1$, ¼$_2$, . . . ,¼$_n$ be the truth values committed to a proposition ¸ 4˜, where 0 d ¼$_i$ d 1, *i* = 1, 2, . . . , *n*. Because the truth region is defined within the interval [0,1], the identity of the truth region *e* (shown in Figure 11 and implemented in equation (17)) is equal to 0.5. Supportive and nonsupportive pieces of evidence are represented by ¼$_i$ (*i* represents the rule number). The parameter ± (shown in Figure 11 and implemented in equation (17)) determines the weight of each piece of evidence, which in turn represents the relative importance of the aggregated bodies of evidence. The desire is that the combination operator shown in Figure 11 can be defined to perform various strengths. That is, the Minkowski averaging operator can yield different aggregation functions, which are defined by different choices of the parameter ± = 1, 3, . . . , . The optimum value of alpha was equal to 3 [14]. The variable *n* (in Figure 11) represents the number of identity attributes, which in this case is equal to 4. Hence, using the fusion formula given in Figure 11, the fusion equations for rules 1, 2, and 3 (out of the nine) as described previously in the rule-based matrix example in Figure 10, are as follows (the four ¼$_{we}$ represent the degree of confidence of ¸$_1$, ¸$_2$, ¸$_3$, and ¸$_4$, respectively as a weak input):

$$\mu 1 = \frac{0.5 + \left[(\mu_{we} - 0.5)^3 + (\mu_{we} - 0.5)^3 + (\mu_{we} - 0.5)^3 + (\mu_{we} - 0.5)^3\right]^{\frac{1}{3}}}{4}$$

$$\mu 2 = \frac{0.5 + \left[(\mu_{we} - 0.5)^3 + (\mu_{we} - 0.5)^3 + (\mu_{mid} - 0.5)^3 + (\mu_{we} - 0.5)^3\right]^{\frac{1}{3}}}{4}$$

$$\mu 3 = \frac{0.5 + \left[(\mu_{we} - 0.5)^3 + (\mu_{we} - 0.5)^3 + (\mu_{sr} - 0.5)^3 + (\mu_{we} - 0.5)^3\right]^{\frac{1}{3}}}{4} \qquad (17)$$

Hence, the likelihood of a target being an aircraft is weak for these three rules according to the rule-based matrix in Figure 10. Only one of the nine rules in the rule-based matrix in Figure 10 will give a moderate likelihood output; all the other eight rules also give a weak likelihood output. Let this rule (with a moderate output) be described as rule 7 below.

$$\mu 7 = \frac{0.5 + \left[\left(\mu_{we} - 0.5 \right)^3 + \left(\mu_{we} - 0.5 \right)^3 + \left(\mu_{we} - 0.5 \right)^3 + \left(\mu_{st} - 0.5 \right)^3 \right]^{\frac{1}{3}}}{4}$$

As it was important to fuzzify data in order to apply nondeterministic approaches, the data was recovered from the output fuzzy consensus. We defuzzify the output of the fusion system in order to recover crisp output values. The defuzzification scheme is based on the centroid method. At each of the fuzzy outputs, the crisp value is computed as follows:

$$\theta = \frac{\sum_{i=1}^{n} \mu_i \mu_c(\theta_{ci})}{\sum_{i=1}^{n} \mu_i} \qquad (18)$$

where n is the number of rules used at a given input, and ¼$_c$ (,$_{ci}$) is the centroid of the fuzzy output at the ith rule.

4. Experiments

Fuzzy reasoning through fuzzy rule combination is used to fuse object identification attributes derived from preprocessing co-registered SAR and color aerial photo images. The maximum number of possible identity attributes used in the fusion process is 4. The identification attributes consist of the area of a moving 8×8 pixel window of a processed SAR image whose output indicates a homogeneous texture similar to that of aircraft targets; the output of a background discrimination algorithm [4, 5] identifying man-made targets derived from the SAR image; and two attribute outputs of a trained recurrent back-propagation neural network which classifies the color aerial photo graphic image as being buildings or tar/cement. These attributes are marked as inputs A, B, C, and D in Figure 1.

When all four attributes A, B, C, and D are fused together using fuzzy reasoning, the resultant output image is shown in Figure 12. The circles in Figure 12 represent the detected aircraft targets, and the squares represent the false alarms.

We assume that we have *a priori* information on the approximate size/area of aircraft targets, and we also know that there is a high likelihood that the aircraft are located on a tar/cement surface or some other known surface.

The first object identification attribute, shown as A in Figure 1, derived from the SAR image, has been obtained through a preprocessing stage and as part of the fusion stage. The preprocessing stage is the classified output image from a back-propagation neural network NN1. The output is the result of supervised texture classification of the SAR image. The texture parameters computed from the co-occurrence matrices [15] are second angular momentum, correlation, and contrast [15]. As shown in Figure 1, these parameters are used as inputs to the neural network NN1, which has 3 input neurons, 10 hidden, and 1 output neuron. NN1 was trained on the texture characteristics of 3 of the 6 aircraft targets shown in Figure 2. It took NN1 20,000 iterations (3 hours on a SPARC20 workstation) to train on the three aircraft targets. Hence, based on the texture features such as second angular momentum, correlation, and contrast computed from the co-occurrence matrices of the 5×5 window, the back-propagation neural network (NN1) classified the given SAR input image to produce image I1 shown in Figure 4. The white areas in image I1 in Figure 4 represent portions of the image for which the texture/context for a 5×5 pixel window are similar to that of the aircraft targets in the SAR target position image shown in Figure 2.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Once preprocessing training is completed, part of the fusion stage of the ATR system extracts the first identity attribute (shown as A in Figure 1), by performing morphological processing on image I1 to calculate the area of the 8×8 pixel window, only if there are at least four pixels at the center of the window (i.e., number of white pixels connected together in the horizontal and vertical directions starting from the center four pixels in the 8x8 pixel window over the entire image). From *a priori* knowledge on aircraft size, we know that the aircraft pixel size can range from 4 to 22 pixels.

The second attribute extracted in the fusion stage of the ATR system, shown as B in Figure 1, is the output of a background discrimination algorithm which rejects natural-clutter and detects man-made targets for a certain intensity threshold value [4, 5]. Refer to the block diagram in Figure 1, and to an example showing all man-made targets for a certain threshold value shown in Figure 5.

The third and fourth object identification attributes extracted in the fusion stage of the ATR system, shown as C and D in Figure 1, are derived from a color aerial photograph. The mean pixel intensity result (out of 25 pixels) obtained from each of the three 5×5 pixel windows from the red, green, and blue bands are input into NN2. The color aerial photograph is classified into two categories (attributes), namely, tar/cement and buildings using recurrent neural network NN2. From this process, each pixel will have two attributes, y(0) and y(1), corresponding to tar/cement and building classes. The recurrent neural network NN2 has four inputs, namely, red, green, and blue bands (Figure 3), and the previous output y(0) via a feedback loop as shown in Figure 1. Unlike neural network NN1, which was trained on the texture of targets derived from a current SAR image, NN2 was trained on the spectral values of tar/cement and buildings of the red, green, and blue bands of an aerial image taken a few months prior to the SAR. The tar/cement areas indicate likely locations of aircraft, and building locations indicate unlikely targets. It took NN2 25,000 iterations (3 hours on a SPARC20 workstation) to train on selected portions of the tar/cement and buildings shown in Figure 3.

5. Discussion of Results

The objective of this chapter was to explore the use of neural networks and the fusion technique to automatically detect targets such as aircraft in a SAR image using *a priori* information derived from all possible sources, to increase the detection rate, and to reduce the false alarm rate. For example, we used *a priori* knowledge of the size or area of targets (which could range from 4 to 22 pixels in this case), *a priori* knowledge of position of existing buildings and tarred/cemented areas in the image to increase our target detection effectiveness.

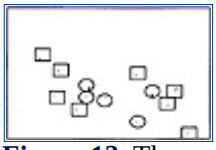


Figure 12 The output image using fuzzy (rule) technique (with attributes A, B, C, and D). Five targets (in circles) and eight false alarms (in squares).

As shown in Figures 4 and 5, the advantages of using the fusion process quickly becomes evident when observing the numerous false targets obtained from outputs of processing just one of the images or using one preprocessing technique. The problems with the classifier outputs NN1 and NN2 could be the number of misclassifications, due to poor training examples or poor selection of inputs or even the fact that the network found a local minimum, and not a global one. The problem with the back-discrimination algorithm could be the size of the window,and the amount of speckle in the SAR image and the selection of the threshold value could all produce varying results.

As shown in Figure 5 for the optimum threshold value in the background discrimination algorithm [5], 46 targets (six of which are aircraft) were detected in the SAR image. Using the fusion technique, we are able to successfully automatically identify 5 of the 6 aircraft shown within circles in Figure 12. Hence by using all the identification attribute information at hand, we have reduced the number of unwanted detections from 40, as obtained in the background discrimination algorithm, to 8 (shown in squares in Figure 12).

We have demonstrated that by using additional information, the performance of target recognition can be significantly improved. Even though there were 6 targets, the fusion algorithm was tested on the 24,066 pixels as the processing window moved across the registered images (126×191, size in rows and columns respectively). In conclusion, even though we achieved 83% accuracy rate using the fusion technique further investigations are necessary using several images before drawing definitive conclusions.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Acknowledgments

The authors are grateful for the comments from Professor Rao Vemuri from the University of California, the enabling research task manager principal research scientist Dr. Bob Seymour, senior research scientist Dr. Anne-Marie Grisogono, and to the Land Operations Division at the Defence Science and Technology Organisation.

References and Further Reading

1 Roth, M.R., (1990), "Survey of Neural Network Technology for Automatic Target Recognition," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 28-33, March.

2 Bhanu, B., (1986), "Automatic Target Recognition: state of the art survey," *IEEE Aerospace Electron. Syst.*, vol. 1, AES-22, no. 4, pp. 364-379, July.

3 Gyer, M., (1988), "Automatic Target Recognition Panel Report," SAIC, San Diego, CA, rep. ET-TAR-002, December.

4 Kreithen, D.E, Halverson, S.D., Owirka, G.J, (1993), "Discriminating Targets from Clutter," *The Lincoln Laboratory Journal*, vol. 6, no. 1, pp. 11-51.

5 Stacy, N., Smith, R., Nash, G., (1994), "Automatic Target Detection for the INGARRA Airborne Radar Surveillance System," D.S.T.O. internal report, Ingara-210, pp. 1-12, August.

6 Whitbread, P., (1992), "Multispectral Texture: Improving Classification of Multispectral Images by the Integration of Spatial Information," Ph.D. Thesis, The University of Adelaide, October.

7 Lippman, R.P., (1989), "Pattern Classification using Neural Networks," *IEEE Communications Magazine*, pp. 47-64, November.

8 Bridle, J.S., (1989), *Probabilistic Interpretation of Feed forward Classification Network Outputs with Relationships to Statistical, Architecture and Applications*, Fougelman-Soulie and Herault, Eds., Springer-Verlag.

9 Jain, L.C., (Ed.), (1997), *Soft Computing Techniques in Knowledge-Based Intelligent Engineering Systems*, Springer-Verlag.

10 Hercht-Nielsen, R., (1990), *Neurocomputing*, Addison-Wesley.

11 Rumelhart, D.E., Hinton, G.E., Williams, R.J., (1986), "Learning Internal Representation by Error Propagation," in *Parallel Distributed Processing*, MIT Press, pp. 318-360.

12 Widrow, G., Hoff, M.E., (1960), "Adaptive Switching Circuits," in *Conventional Record Part 4, IRE Western Electronic Show and Convention*, pp. 96-104.

13 Rumelhart, D.E., Hinton, G.E., Williams, R.J., (1988), "Learning Internal Representations by Backpropagation Errors," *Nature*, Vol. 323, pp. 533-536, October.

14 Abdulghafour, M.B., (1992), "Data Fusion through Fuzzy Reasoning Applied to Feature Extraction from Multi-Sensory Images," Ph.D. Thesis, The University of Tennessee, Knoxville, pp. 41-96, December.

15 Harlick, R.M., Shanmugham, K., Dinstein I., (1973), "Textural Features for Image Classification," *IEEE Transactions on Systems, Man, and Cybernetics, vol*. SMC-3, no.6, pp. 610-621, November.

16 Filippidis, A., Jain, L.C., (1997), "Identity Attribute Information in a Multiband Aerial Photo Image Using Neural Networks to Automatically Detect Targets," *International Journal of Knowledge-Based Intelligent Engineering Systems*, vol. 1, no. 1, pp. 52-56, January.

17 Ohanian, P.P., Dubes, R.C., (1992), "Performance Evaluation for Four Classes of Textural Features," *Pattern Recognition*, vol. 25, pp. 819-833.

18 Shettigara, V.K., Kempinger, S.G., Kruzins, E., (1997), "Fusion of Texture and Intensity Characteristics for Target Detection in SAR Images," *Image Analysis and Information Fusion, Proceedings of the International Workshop IAIF1'97*, Adelaide, South Australia, pp. 495-499.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Previous | Table of Contents | Next

Chapter 4
The Self-Organizing Map in Industry Analysis

Olli Simula, Juha Vesanto
Laboratory of Computer and Information Science
Helsinki University of Technology
P.O.Box 2200, FIN-02015 HUT
Finland

Petri Vasara, Riina-Riitta Helminen
Jaakko Pöyry Consulting
Jaakonkatu 3, FIN-01620 VANTAA
Finland

E-mail: Olli.Simula@hut.fi, Petri.Vasara@poyry.fi,
Juha.Vesanto@hut.fi, Riina-Riitta.Helminen@poyry.fi

The Self-Organizing Map (SOM) is a powerful neural network method for the analysis and visualization of high-dimensional data. It maps nonlinear statistical relationships between high-dimensional measurement data into simple geometric relationships, usually on a two-dimensional grid. The mapping roughly preserves the most important topological and metric relationships of the original data elements and, thus, inherently clusters the data. The need for visualization and clustering occurs, for instance, in the data analysis of complex processes or systems. In various engineering applications, entire fields of industry can be investigated using SOM-based methods. The data exploration tool presented in this chapter allows visualization and analysis of large databases of industrial systems. The forest industry is the first chosen application for the tool. To illustrate the global nature of the forest industry, the example case is used to cluster the pulp and paper mills of the world.

1. Introduction

The Self-Organizing Map (SOM), developed by Professor Kohonen [9], is one of the most popular neural network models. The SOM implements a nonlinear projection from the high-dimensional space of input

signals onto a low-dimensional array of neurons. The array forms an elastic net that during learning folds onto the “cloud” formed by the input data. The net approximates the probability density function of the input data. This means that the neurons tend to drift to where the data are dense, while there are only a few neurons where data are sparsely located. The mapping tends to preserve the topological relationships of the input signal domains. Due to this topology-preserving property, the SOM is able to cluster input information and their relationships on the map. The SOM also has the capability to generalize, i.e., the network can interpolate between previously encountered inputs.

The most important applications of the SOM are in the visualization of complex processes and systems and discovery of dependencies and abstractions from raw data [8, 23, 26]. Especially the latter operation, data exploration or data mining, has recently become important because of the increasing amounts of measured information and data.

In industrial and engineering applications, the most straightforward applications of the SOM are in analysis and monitoring of complex process or machine states, otherwise difficult or even impossible to detect and interpret. The SOM algorithm is based on the unsupervised learning principle, i.e., the training is entirely data-driven and little or no *a priori* information about the input data is required. The SOM can be used for pattern recognition and clustering of data without knowing the class memberships of the input data. The SOM can thus be used to automatically detect features inherent to the problem. This is a clear advantage when compared with artificial neural network (ANN) methods based on supervised learning (e.g., multilayered perceptron (MLP)) which require that the target values corresponding to the data vectors be known.

The SOM has been successfully applied in various engineering applications [10], covering areas such as pattern recognition, full-text and image analysis, financial data analysis, process monitoring and control, and fault diagnosis [17, 18, 21]. The ordered signal mapping property of the SOM has also proven useful in certain telecommunications tasks, e.g., in signal detection [15] and adaptive resource allocation problems [20].

Knowledge discovery in databases (KDD) is an emerging area of research in artificial intelligence and information management. The purpose of KDD is to find new knowledge from databases in which the dimension, complexity or the amount of data has so far been prohibitively large for human observation alone. Some typical tasks of KDD are classification, regression, clustering, summarization and dependency modeling. The algorithms that are employed in these tasks include decision trees and rules, nonlinear regression and classification methods such as feed-forward networks and adaptive splines, example-based methods such as k-Nearest Neighbors (kNN), graphical dependency models, and relational learning [3]. Analysis of major global industries is an example of an application for these techniques.

In this chapter, the applicability of the SOM-based methods in knowledge discovery is discussed. Special emphasis is placed on industrial applications in which a lot of different types of information are available from databases and automation systems. Thorough analysis of the industry field requires, for instance, the integration of knowledge originating from different sources. As a case study, the analysis of the world pulp and paper industry is considered. In the case study, a data mining application, ENTIRE, based on the use of the Self-Organizing Map was utilized as the primary analyzing tool [26].

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

2. The Self-Organizing Map in Knowledge Discovery

2.1 The Self-Organizing Map

The SOM algorithm resembles vector quantization (VQ) algorithms, such as *k*-means [1], and is closely related to principal curves [5]. The important distinction from VQ techniques is that the neurons are organized on a regular grid and, along with the selected neuron, its neighbors are also updated, so that the SOM performs an ordering of the neurons. In this respect, the SOM is a multidimensional scaling method projecting data from input space to a lower, typically two-dimensional output space.

A SOM consists of neurons organized in an array. The number of neurons may vary from a few dozen up to several thousand. Each neuron is represented by an *n*-dimensional weight vector, $m = [m_1, \ldots, m_n]$, where n is equal to the dimension of the input vectors. The neurons are connected to adjacent neurons by a neighborhood relation, which dictates the topology, or structure, of the map. Typically, a rectangular (as in Figure 1) or hexagonal (as in Figure 2) neighborhood is used.

The SOM is trained iteratively. In each training step, one sample vector x from the input data set is chosen randomly, and the distance between it and all the weight vectors of the SOM is calculated using some distance measure, e.g., Euclidean distance. The neuron c whose weight vector is closest to the input vector x is called the Best-Matching Unit (BMU):

$$\|x - m_c\| = \min_i \{\|x - m_i\|\} \qquad (1)$$

where $\|\cdot\|$ denotes the distance measure.

After finding the BMU, the weight vectors of the SOM are updated so that the BMU is moved closer to the input vector in the input space. The topological neighbors of the BMU are also treated in a similar way. This adaptation procedure stretches the BMU and its topological neighbors toward the sample vector as shown in Figure 1. The SOM update rule for the weight vector of the unit i is

$$m_i(t+1) = m_i(t) + h_{ci}(t)\left[x(t) - m_i(t)\right] \qquad (2)$$

where t denotes time. The $x(t)$ is the input vector randomly drawn from the input data set at time t and $h_{ci}(t)$ the neighborhood kernel around the winner unit c at time t. The neighborhood kernel is a non-increasing function of time and of the distance of unit i from the winner unit c. It defines the region of influence that the input sample has on the SOM.

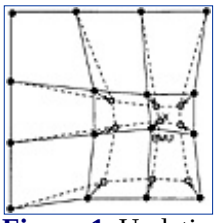


Figure 1 Updating the best matching unit (BMU) and its neighbors toward the input sample marked with x. The solid and dashed lines correspond to the situation before and after updating, respectively.

2.2 Knowledge Discovery Using the SOM

The SOM has several beneficial features which make it a useful methodology in knowledge discovery. It follows the probability density function of the underlying data, it is readily explainable, simple, and — perhaps most importantly — highly visual. The SOM functions as an effective clustering and data reduction algorithm and can thus be used for data cleaning and preprocessing. Integrated with other methods, it can be used for rule extraction [22] and regression [16].

An important property of the SOM is that it is very robust. Naturally, the SOM suffers from any kind of flaws in the data, but the degradation of performance is graceful. An outlier in the data only affects one map unit and its neighbors. The outlier is also easy to detect from the map, since its distance in the input space from other units is large. The SOM can even be used with partial data, or data with missing data component values. Three application areas important in KDD are highlighted in greater detail below.

2.2.1 Visualization

Because of the important role that humans have in KDD, visualization is a data mining method in itself, in addition to being essential in reporting the results, or creating the knowledge [2]. The different SOM visualizations offer information of correlations between data components and of the cluster structure of the data. The illustrations can be used to summarize data sets and to compare them.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

The SOM can be efficiently used in data visualization due to its ability to represent the input data in two dimensions. In the following, several ways to visualize the network are introduced using a simple application example, where a computer system in a network environment was measured in terms of utilization rates of the central processing unit (CPU) and traffic volumes in the network. The SOM was used to form a representation of the characteristic states of the system.

- *The unified distance matrix (u-matrix)* method by Ultsch [23] visualizes the cluster structure of the SOM.

First, a matrix of distances (u-matrix) between the weight vectors of adjacent units of a two-dimensional map is formed. Second, some representation for the matrix is selected, for example, a gray-level image [7]. The u-matrix of the example system is shown in Figure 2a. The lighter the color between two map units, the smaller is the relative distance between them. On the left side, there is a large uniform area, which corresponds to idle state of the computer system. The top right corner forms a clearly separated area, which corresponds to high CPU load in the system.

- *Component plane representation* visualizes relative component values in the weight vectors of the SOM.

The illustration can be considered as a “sliced” version of the SOM, where each plane shows the distribution of one weight vector component. Using the distributions, dependencies between different process parameters can be studied. For example, Goser et al. [4] have used this kind of visualization to investigate parameter variations in VLSI circuit design. The component planes of the example system are presented in Figure 2c. The colors of map units have been selected so that the lighter the color is, the smaller is the relative component value of the corresponding weight vector. It can be seen, for instance, that the components #1, #2, and #6 (read blocks per second, written blocks per second and write I/O percentage of CPU usage, respectively) are highly correlated.

- *Sammon’s mapping* is an iterative algorithm to project high-dimensional vectors in two dimensions [13].

The nonlinear mapping tries to preserve the relative distances between input vectors. The algorithm can be used to visualize the SOM by mapping the values of the weight vectors onto a plane. To enhance the net-like look, the neighboring map units may be connected to each other with lines to show the topological relations. Since the SOM tends to approximate the probability density of the input data, the Sammon’s mapping of the SOM can be used as a very rough approximation of the form of the input

data. The Sammon's mapping of the example system is illustrated in Figure 2d. According to the mapping, the SOM seems to be well ordered in the input space. Sammon's mapping can also be applied directly to data sets, but because it is computationally very intensive, it is too slow for large data sets. However, the SOM quantizes the input data to a small number of weight vectors, which lightens the burden of computation to an acceptable level.

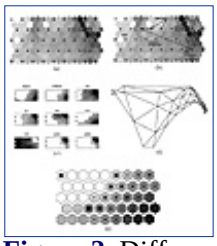


Figure 2 Different visualizations of the SOM. U-matrix presentation (a), trajectory on top of labeled u-matrix (b), planes representation (c), Sammon's mapping (d), and data histogram on top of a component plane (e). In figures (a) and (b), the black spots denote map units.

- *Data histogram* shows how input data are clustered by the SOM.

In other words, it shows how many input vectors belong to clusters defined by each map unit. The histogram is formed using a trained SOM and a data set: for each data set vector, BMU is determined, and "hit counter" of that unit is increased by one. The histograms may be visualized in many ways. In our example, we have used squares of different sizes: the larger the square, the larger the counter value. The data histogram of the example application is shown in Figure 2e.

- *Operating point and trajectory* can be used to study the behavior of a process in time.

The operating point of the process at time *t* is the BMU of the measurement vector *x(t)*. The location of the point on the topologically ordered SOM can be easily visualized and used to determine the current process state. If the history of the process is also of interest, a sequence of operating points in time forming a trajectory can be studied. The trajectory shows the movement of the operating point, which in some cases may be a very useful piece of information. A piece of trajectory of the example system is illustrated in Figure 2b. Also, the key areas of the SOM have been identified and labeled on the map. The trajectory starts from the normal operation area and moves through a disk-intensive phase to high load area.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

2.2.2 Clustering

Clustering is one of the main application areas of the SOM. The neurons of the SOM are themselves cluster centers; but to accommodate interpretation the map units can be combined to form bigger clusters. A significant advantage with this approach is that while the Voronoi regions of the map units are convex, the combination of several map units allows the construction of non-convex clusters.

A common strategy in clustering the units of the SOM is to calculate a distance matrix between the reference vectors and use a high value of the matrix as an indication of a cluster border [11, 23, 24]. In 3-D visualization of such a matrix, e.g., the u-matrix, the clusters will appear as “valleys.” The problem then is how to determine which map units belong to a given cluster. For this, agglomerative and divisive algorithms are typically used, e.g., in [14, 25]. In addition to distance, some other joining criteria can be used, for example, that the joined clusters are required to be adjacent [14].

Another quite interesting option is to use another SOM to cluster the map units. This kind of structure is often referred to as a hierarchical SOM. Usually, a “hierarchical SOM” refers to a tree of maps, the lower levels of which act as a preprocessing stage to the higher ones. As the hierarchy is traversed upward, the information becomes more and more abstract. Hierarchical self-organizing networks were first proposed by Luttrell [12]. He pointed out that although adding extra layers to a vector quantizer yields a higher distortion in reconstruction, it also effectively reduces the complexity of the task. Another advantage is that different kinds of representations are available from different levels of the hierarchy.

The SOM can be used for classification purposes by assigning a class for each reference vector and deciding the class of a sample vector based on the class of its BMU. However, it should be noted that if the class memberships of the training data are known, using the SOM for classification purposes is not sound, since the SOM does not take into account the known class memberships and cannot therefore optimize the class boundaries appropriately. In such cases, the Learning Vector Quantizer (LVQ), a close relative of the SOM, or another method of supervised classification should be used [9].

2.2.3 Modeling

The problem of system modeling is one of high practical importance. A traditional way to approach modeling is to estimate the underlying function globally. In the last decade, however, local models have been a source of much interest because in many cases they give better results than global models [19]. This is especially true

if the function characteristics vary throughout the feature space.

The elastic net formed by the SOM in the input space can be interpreted as an implicit lookup model of the phenomena that produced the training data. The lookup model can be used for sensitivity analysis [6]. An expansion is to fit local models for each map unit. The local models can be constructed in various ways, ranging from using the best example vector to splines and small MLPs. Usually, local models are kept simple, such as weighted averages of the example vectors or linear regression models [26].

3. Requirements for a Forest Industry Analysis Tool

3.1 From the General to the Specific: The Neo-Generalist Between Sectoral Expert, Amateur, and Polymath

In an age other than ours, it was possible to be a polymath. Now, the sheer amount of information in each field makes this description hard to apply. Instead, we have a progression from the British “cult of the amateur” toward the generalist with domain expertise on a broad but shallow basis. A global consulting company such as the Jaakko Pöyry Group[1] is a good laboratory for detecting the appearance of a new type of knowledge worker. The company’s 35-year history is illustrative: a tradition of domain expertise was over the years slowly fused with a dose of management consulting. Then, the management consulting wing split off in a separate company (Jaakko Pöyry Consulting) under the group aegis. Finally, in the latter company the need for “neo-generalist” consultants with a combination of broad but shallow domain expertise and general, meta-level analyst skills is becoming apparent.

[1] A market leader in management and engineering consultancy for the forest industry worldwide.

3.2 Future Direction of Knowledge Discovery in Industry Analysis

Much of the work done at Jaakko Pöyry Consulting can be put under the vague heading of “industry analysis,” involving all relevant aspects of the forest industry, from markets and financial analysis to environmental issues and forestry. The arrival of the new type of consultant/knowledge worker is heralded by new types of problems gaining in relevance. Characteristics of the problems inherent in this type of analysis include:

- Many dimensions of data.

  The number of relevant fields in different Jaakko Pöyry pulp and paper mill data banks, of which there are many, linked types, rarely drops below 30.

- Many categories of dimensions of data which have to be dealt with both separately and in combination.

  When the above-mentioned data banks are used, the output is mostly a combination of at least two categories of information (technical data on equipment, time-dependent production numbers, resource usages, production costs). These are of interest by themselves, but the combination is where the true value lies.

- Non-existent or poor theoretical foundation.

  Rules of thumb abound, and even such a basic component as the pulping process is far from completely understood in the sense of physical modeling.

- Incomplete data.

  Even with an extensive global data collection network, it is nearly impossible to find out many highly relevant pieces of information. Some are protected by commercial interest, others are not measured in the same way or at all across the board, etc. The market shares of individual companies for different products in different countries are among these items, and environmental data is perhaps shrouded in the greatest veil of secrecy of all.

- Correlations between variables are not obvious.

  The prices for different products or resources (e.g., pulp and recovered paper; pulp and fine paper) have a logical connection but often seem to be the output of a chaotic system. The modern pulp mill is a very complex piece of equipment, extending over a large area and volume, rich in feedback loops, constantly being rebuilt and modified. The effects of changes in one area on another may be great but hard to predict.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

3.3 Characteristics of the Forest Industry

The characteristics listed above are shared by data concerning many industries. Attempting to put the key features of the forest industry at its current stage of development in a nutshell is a hard task. Figure 3 gives not a nutshell but an onion shell view of the forest industry — peeling away layer after layer of complexities.

Figure 3 The forest industry onion shell.

The chosen order of these layers is not self-evident, and the onion shell idea is correct in describing the unfolding levels of complexity but incorrect in eliminating the interplay of factors. However, more and more, the questions asked by the industry are of the type described in the example below:

An executive wants to know the competitiveness of his packaging business units in Europe and North America, competing with local players in France, and contemplating the construction of a new corrugated board mill in the same country because of the supply of recovered paper. The energy consumption of the planned mill is rising in importance with the possibility of EU-wide harmonized CO_2-taxes. Elsewhere, the reduction of the discharges of nitrogen oxides to meet emission limits is proving difficult because of technical problems not easily solvable with current knowledge. The environmental element in the decision-making is complemented by the need to take a stand on the producer responsibility issue: how should the responsibility for paper recovery be shared by different players such as municipalities, collectors, merchants, retail stores, and the paper industry?

If there is a less subtly hidden message in the figure, it is perhaps the location of environmental issues at the core.

3.4 Requirements for a Computerized Tool for a Forest Industry General Analyst

Questions asked by companies with the need for a global overview, to take into account the long lifetime (up to 30 years) of an investment, and to try to assure a long-term supply of fiber, energy, and water put a heavy strain on the resources of the analyst. Computer tools are the basis of most analysis assignments, but the bag of tricks of the "neo-generalist" is of special importance. Presenting a set of possible elements from the field of mathematics and artificial intelligence, we have:

- Extracting production rules.

If "company concentrates on newsprint" AND "company is situated in Canada," then with probability X% "company does not use recovered paper."

- Forecasting with exact numbers.

What is the price of market pulp for the next quarter in Asia?

- Visualization.

When Sweden took the path of making internal changes in the process and Finland chose external treatment of effluents, how did the Scandinavian effluent discharge emission landscape change?

- Classification.

The Finnish major company Enso decided to acquire the Holtzmann paper company in Germany, thus reinforcing its publication paper sector. Find an acquisition analogous in effects on the company-internal geographical and production balance.

- Time dimension/trajectories.

How has the state of the Finnish company UPM-Kymmene changed when it has emerged (as the result of a series of fusions and acquisitions) to become the biggest forest sector company in Europe?

- Cutting across and combining problem dimensions.

When the Swedish company SCA acquired the German company PWA, how did the acquisition change the company, when we combine (a) financial and environmental, (b) environmental and technological, (c) financial and technological, or (d) all three aspects?

- Correlation hunting.

Taking the top 150 pulp and paper companies in the world, is there a link between return on investment and product sortiment?

All of these features are useful. However, exact forecasts are perhaps the most researched area in forest industry analysis — and the predictions are no more correct than elsewhere. Experiments conducted with Jaakko Pöyry Consulting databases resulted in highly stochastic rules with a generous amount of preconditions. Single experiments are of course not conclusive, and rule extraction is a task worth pursuing. Rules could even be extracted from back-propagation networks, thus combining a limited degree of numeric forecasting and rule extraction. The last five features above, however, are intriguing questions rarely asked because of their difficulty. They also describe characteristics of the Self-Organizing Map (SOM).

3.5 A SOM of ENTIRE Tasks: Mapping the Consultant's Workload Trajectory

We can make the characteristics above into a series of questions about one company. The example chosen for the set is the large Finnish company Enso.

- How should Enso's investments in the future be directed to best match the group's product sortiment to market demands?
- How is the result of the merger of two state-owned forest industry companies, Enso and Veitsiluoto, different from the two starting points, when the enviro-techno-economic profile is examined?
- Does the path of formation of today's Enso (acquisitions and fusions) represent a clear strategy or has the company changed strategic direction as a result of circumstances?
- How does the Enso acquisition of the Holtzmann company change its recovered paper consumption profile?
- How well do the emission changes for Enso's pulp mills match the overall trend of changes in, on one hand, Finland and Sweden, and, on the other hand, the Northern hemisphere?

In answering questions, an array of databases is needed, financial, technological, and environmental data are combined in different ways, and both static and dynamic data must be considered. The consultants can use SOMs in each case.

Taking a set of projects completed by consultant teams at Jaakko Pöyry Consulting and classifying them using a suitably chosen descriptive vector (including, e.g., client type, country, project financial overhead) would lead to a meta-SOM of tasks performed using SOMs. The number of projects in the 1990s is around 2000 so far; not all of them are obvious candidates for SOMs, but a suitable set could be found and a Jaakko Pöyry Consulting trajectory "SOMmed."

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

4. Study of Pulp and Paper Technology

In this case study, the SOM is used to analyze data from the world pulp and paper industry. Three data sets were used, including information on over 4000 pulp and paper mills, and over 11,000 paper machines and pulp lines in them. The first one included information on the production capacities of the mills, the second included information on the technology of the paper machines, and the third, the technology of the pulp lines.

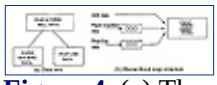


Figure 4 (a) There were three technological data sets: one of mill production capacities, one of paper machines and one of pulp lines. Each mill could contain several paper machines and pulp lines. (b) The hierarchical map structure. Data histograms from the two smaller maps were utilized in the training of the third map. The arrows show which data sets were used in training the maps.

Each mill could contain several paper machines and pulp lines and, therefore, a hierarchical structure of maps was used (see Figure 4). At first, two low-level maps were constructed from the paper machine and pulp line data sets. These maps provided a clustering of the different machine types. The technology map was trained using the mill-specific info in the mill data set and the data histograms from the two low-level maps.

4.1 Paper Machines and Pulp Lines

In the construction of the paper machine map seven features were used: wire width, wire trim width, speed, minimum and maximum grammages, present technical production capacity, and the year the machine was (re)built. For the pulp lines, five features were used: bleaching type, fiber type, main pulp grade type, market capacity as a percentage of the total pulp production, and the total pulp production capacity itself. The units and ranges are given in Table 1. Prior to training, each component was linearly scaled so that its variance was equal to 1.

The vector component values in the map units can be seen from the component plane representations in Figure 5. By visual inspection of the paper machine map, the first three components (wire width, wire trim width, and speed) have a strong correlation, as do the next two components (minimum and maximum grammage). In the pulp line map, no such global correlations can be seen.

The clustering of the maps was based on visual inspection of the u-matrices shown in Figure 6 supplemented by the knowledge of the distribution of component values. The clusters and their descriptions are shown in Table 2. From the paper machine map, six clusters were extracted corresponding to different types of machines. The first cluster corresponds to new, high-capacity machines with high speed and high value for wire width. The second cluster includes slightly older big or average sized machines. The third cluster consists of machines with big paper weight. The fourth cluster resembled the second with the difference that the machines are a bit smaller and about 5 years older. The last two clusters had the largest number of hits. They correspond to smallest and oldest paper machines. Based on the properties of the clusters, the paper machines can be divided into three different types:

1. The new machines with highest capacity and speed and widest wire (cluster 1).
2. The paper machines with big paper weight (cluster 3).
3. The majority of the paper machines belong to the third type. Their capacity and size decrease steadily with increasing age (clusters 2, 4-6).

Table 1 Paper machine feature vector components (top) and pulp line feature vector components (bottom); the second column gives the range of values of the component

Component	Value range
Paper machine data	
Wire width (*mm*)	0-10160
Wire trim width (*mm*)	0-9450
Speed (*m/min*)	0-2800
Grammage, min (g/m^2)	0-999
Grammage, max (g/m^2)	0-9999
Capacity (*1000 t/a*)	0.1-525
(Re)built (*year*)	1863-1996
Pulp line data	
Bleaching code	1, 2, 3
Fiber code	1, 2, 3, 4
Main pulp grade code	1, 2, 3, 4, 5
Market pulp capacity (%)	0-100
Total capacity (*1000 t/a*)	1-1075

Figure 5 The component planes of the paper machine (a) and pulp line (b) maps. The name of the component on the right and the corresponding values in the map units from 1 to 20 in the middle. All components are presented with a grayscale, with black representing the maximum value and white the minimum.

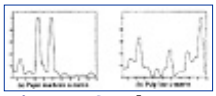


Figure 6 The u-matrix presentations of the paper machine (a) and pulp line (b) maps, with the map unit on the *x*-axis and the value of u-matrix on *y*-axis. Clusters on map are separated by peaks in the u-matrix. Clusters are listed in Table 2.

Table 2 Clusters of the paper machine map (top) and pulp line map (bottom). The second column gives the units that belong to a certain cluster. The third column gives the number of samples that were projected to the cluster. See also Figures 5 and 6.

Cluster	Units	Hits	Description
Paper machine map			
1	1, 2	454	Wide wire and high speed, small paper weight, high capacity, new machines.
2	3-6	1024	Average sized machines, over 10 years old on average.
3	7-9	467	Narrow wire, slow speed, big paper weight, small capacity.

4	10-12	1033	Average wire width and speed, over 15 years old on average.
5	13-16	2648	Small machines, over 15 years old on average.
6	17-20	3139	Small and old machines.
Pulp line map			
1	1-4	557	Pulp made from waste paper.
2	5-11	993	Unbleached pulp, both fiber and grade vary.
3	12-15	762	Bleached pulp, wood or other virgin fibers, chemical main grade.
4	16-18	303	Bleached pulp, mostly wood fiber and chemical main grade, average market ratio, high capacity.
5	19,20	364	Big market ratio, mostly wood fiber and chemical main grade.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

The pulp line map can be divided into five clusters listed in Table 2. The pulp lines in the first cluster make bleached pulp and use recovered paper for fiber. The second cluster corresponds to pulp lines making unbleached pulp with varying fiber and main grade types. The third cluster has pulp lines which produce bleached pulp mostly from wood or other virgin fibers with chemical main grade. The fourth cluster consists of pulp lines with high capacity and an average market ratio. The pulp is bleached, fiber mostly wood, and grade mostly chemical. The last cluster consists of pulp lines with high market pulp ratio. All in all, there are three major pulp line types:

1. Those using recovered paper for pulp.
2. Those making unbleached pulp, further divided into three subtypes:

    a. Wood fiber and semichemical pulp grade
    b. Wood fiber and either chemical or mechanical pulp grade
    c. Other virgin fibers with chemical pulp grade

3. Those using chemical pulpwood to produce mainly bleached pulp from wood fiber. This type could further be divided into three subtypes: those with small capacity, those with high capacity, and those with a high market ratio of pulp production.

4.2 Mill Technology

The mill data set consisted of the total pulp and paper production capacities, the number of paper machines, the number of coaters and the percentage of total production of different pulp and paper types for each mill, for a total of 35 features. They were augmented by the data histograms from the paper machine and pulp line maps. For each mill, the two data histograms were constructed using the paper machines and pulp lines in that particular mill. To emphasize the importance of high capacity, the histograms were weighted by the production capacities of the machines/lines. After this, the histograms were smoothed by convoluting them with a symmetric vector *[0.3, 1, 0.3]*.[2] Finally, the histograms were normalized. All in all, the technology data set consisted of 4205 75-dimensional vectors, each corresponding to one mill. A 40 times 25 -sized SOM was trained with this data.

[2] Since the neighboring units of the SOM have similar data items projected to them, the histograms (1) and (2) in

Figure 7 should be considered more similar than histograms (1) and (3).

Figure 7 On the left side three different histograms 1, 2, and 3. Since in the SOM, similar vectors are projected to close lying map units, histograms 1 and 2 should be more similar than histograms 1 and 3. However, using Euclidean distance between histograms, this is not so. Instead, the quantization errors (QE) are equal. This is corrected by convoluting the histograms with a symmetric matrix, as seen on the right side.

The u-matrix of the mill technology map is shown in Figure 8 and part of the component planes in Figure 9. Several correlations between components can be seen from the component plane representation, such as that the mills with high paper production capacity usually also have a high pulp production and that newsprint production corresponded to high thermomechanical pulp production. Of special interest are correlations between different pulp and paper types. From Figure 9 it can also be seen that for some components, the high values were divided between clearly different types of mills. For example, there are two types of mills that produce chemimechanical pulp: those that also produce wood containing paper and those that concentrate purely on producing pulp.

An analysis of mill types was performed by first dividing the map into 31 initial clusters. After this, all components were separated into three ranges and it was investigated which clusters had component values in the two highest thirds. Finally, each cluster was examined to see what kind of mill type it corresponded to. During this procedure, some of the initial clusters were combined. For example, the SmallInd cluster was combined from several initial clusters corresponding to mills producing various industrial papers. The only truly separating factor between the initial clusters was that they received data from different units of the paper machine map: units 13-20 corresponding to old and small paper machines. The cluster analysis resulted in the 20 different mill types described in Table 3.

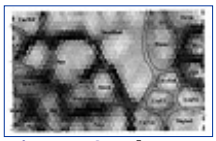


Figure 8 The u-matrix of the mill technology map. Black corresponds to a high value of the u-matrix, white to small value. The mill types have been marked on the map with lines and labeled as in Table 3.

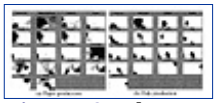


Figure 9 The paper production (a) and pulp production (b) related component planes of the mill technology map. For each component, the black corresponds to the highest value and white to the smallest.

Table 3 Pulp and paper mill type (the clusters refer to the areas marked on the u-matrix of the mill technology map in Figure 8.)

Type	Description
UncWF	Uncoated wood-free paper.
SmallInd	Various industrial papers, machines old and small.
Tissue	Tissue paper, average wire width, and speed.
Dewa	Some tissue paper, but especially high deinked waste paper usage.
CoWF	Many coaters, coated wood-free paper, machines of average capacity and speed.
Pms	Many paper machines, including some high-capacity paper machines, uncoated wood-free but also various industrial papers, unbleached and semibleached sulfite pulp.
Diwa	Cartonboard, linerboard and fluting papers, dispersed waste paper for pulp.
BlWF	Uncoated wood-free paper, bleached chemical pulp from wood fiber.
Big	High capacity, many machines and coaters, wood-free paper or linerboard, pulp is chemical (sulfate), machines are big.
WrLi	Wrapping paper and linerboard, unbleached sulfate pulp, big paper machines.
Mech	Various industrial papers, pulp is unbleached and mechanical: groundwood or rmp.
AveInd	Cartonboard, linerboard, and fluting paper, average capacity machines.

Flut	Fluting paper, semichemical unbleached pulp, average capacity paper machines.
WoodC	Wood containing paper, chemimechanical or mechanical pulp from wood fiber, average capacity paper machines.
News	High paper capacity: newsprint, thermomechanical pulp, high-capacity paper machines and pulp lines.
Pulp	No paper production but large pulp production, high-capacity pulp lines, big market percentage.
Carton	Cartonboard and other papers, big weight in paper machines.
LinFl1	Linerboard and fluting paper, small to average sized machines.
LinFl2	Linerboard and fluting paper, high-capacity machines.
BigInd	Cartonboard, wrapping, tissue and other papers, high-capacity machines.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

A couple of important notices are in place here. First, the set of 20 mill types is by no means definitive. Depending on the desired precision, several other clusters could be extracted from the map. For example, the Pulp cluster can be divided into three subtypes based on the pulp type and the production capacity. Second, the clustering is heavily influenced by the choice and scaling of the components. The mill types in this study reflect only the information present in the data components used. By changing part of the components, or even scaling them differently, a different set of mill types could be obtained. Third, the analysis made so far details the different clusters only in very sketchy terms. To get a better view of the different types, a more detailed in-cluster or in-type analysis would be beneficial.

4.3 Geographical Areas

It is interesting to note that some mill types are typical of certain geographical regions. For the analysis of different geographical areas, the data was separated into 11 sets, each consisting of pulp and paper mills in a certain area. The data sets were projected on the map and, based on the resulting histograms, some conclusions can be drawn for each region, as listed in Table 4. The same approach can be directly used for comparing and analyzing different companies.

Table 4 Different geographical areas and the main mill types they have

Region	Mills	Description
Scandinavia	149	Big capacity mills, newsprint and pulp-only mills but relatively little industrial paper.
Western Europe	1004	Even spread of all mill types, special notice on the many mills using dispersed waste paper.
North America	759	Printing/writing paper production resembles that of Scandinavia, but in addition quite a lot of old SmallInd mills.
Eastern Europe	302	Industrial papers; old SmallInd mills and mills making mechanical pulp. Also some mills in Diwa cluster.
Latin America	533	Even spread of all mill types, special notice of mechanical pulp.
Near and Middle East	65	Industrial papers, mills in Diwa and BigInd clusters.
Africa	106	Mainly industrial papers.

China	370	Many paper machines per mill, wood-free paper, some high-capacity industrial paper mills and several small pulp-only mills.
Japan	221	Even spread of all mill types, many mills using deinked waste paper.
Far Asia	665	Wood-free and various industrial papers, many of them with high-capacity machines.
Oceania	31	Mostly new machines, otherwise an even spread of all mill types, many mills in Diwa cluster.

Four of the histograms are shown in Figure 10: Scandinavia, North America, Far Asia, and China. Scandinavia and North America represent technologically advanced regions. Scandinavia in particular has mostly new, high-capacity mills, the majority of which produce printing/writing papers and pulp. North America has in addition a large number of old and small industrial paper mills. Far Asia, on the other hand, is a growing region with mostly average or small-capacity mills, though the paper machines themselves are big. China is a special case; the mills have many machines and they produce both industrial and printing/writing papers. Both Far Asian and Chinese printing/writing paper is almost exclusively wood-containing.

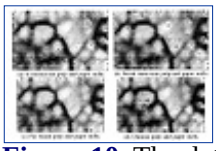


Figure 10 The data set histograms of four different geographical regions on the u-matrix of the pulp and paper mills map. The bigger the square, the more mills were projected to that unit on the map.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

5. Future Directions

The Self-Organizing Map is a versatile tool for exploring data sets. It is an effective clustering method and it has excellent visualization capabilities, including techniques which use the weight vectors of the SOM to give an informative picture of the data space, and techniques which use data projections to compare data vectors or whole data sets with each other. The visualization capabilities of the SOM make it a valuable tool in data summarization and in consolidating the discovered knowledge. The SOM can also be used for regression and modeling or as a preprocessing stage for other methods. The many abilities of the SOM, together with its robustness and flexibility, are a combination which makes the SOM an excellent tool in knowledge discovery and data mining.

At present, our implementation of the SOM, ENTIRE, is clearly an expert tool. An understanding of the SOM fundamentals and a modicum of domain expertise are of essence for efficient utilization of ENTIRE's potential. Thus, it is a tool for a "neo-generalist" with at least a neural network veneer. In developing ENTIRE further, several directions are possible. They can be pursued simultaneously, but the amount of resources available makes a focus necessary. Goals include:

- Further improvements in visualization.

Using 3-D graphics in virtual reality environments is a particular interest.

- Injecting a degree of expertise into the tool.

ENTIRE could be developed in an application-specific direction, resulting in a help desk in the form of small expert systems. This expertise can be used both as a reference during SOM processing and, perhaps more importantly, in the interpretation phase.

- Improving clustering, autolabeling, correlation hunting.

In the more general realm of research, an aid in the visual hunt for correlation between variables, in the form of a primitive "reporter" summarizing links between SOM layers, would speed up analysis and ensure a smaller rate of missed connections. Combining this with an improved autolabeling function and, more fundamentally, autoclustering, would yield benefits to the user. It would simultaneously provide a pivot point for research: an application and an application area to test new concepts on. This type of cross-fertilization between use in industry and research at university is only possible given a consistent vision and enough time for the cooperation.

In the case study, the world pulp and paper technology was investigated. A hierarchical structure of SOMs was used to combine data from the different data sets. Such use of multiple interpretation layers introduces some additional error due to necessary generalizations but, on the other hand, provides a structured solution to data fusion.

A study combining economic, environmental, and technological data has been made to produce a comprehensive view of the whole pulp and paper industry [26]. The case study has added value, as it transforms a long-talked-about idea in the forest industry (combining economy, technology, and environment in decision-making) into a concrete example.

The results achieved so far have been encouraging. However, much work is still needed in the postprocessing stage and the interpretation of results. The development and automated usage of algorithms that cluster the units of the SOM will be an essential part of future work. This may be accomplished by the use of the hierarchical maps or with fuzzy interpretation rules.

The use of ANNs in industrial applications is often simplified due to the availability of large amounts of measurement data. The unsupervised learning principle of the SOM is a desirable property, and noise and distortion in data can partly be compensated by the robustness of the algorithm. In this chapter, the forest industry has been considered as a case study. However, the methods used are applicable to other fields of industry as well. The ENTIRE tool could easily be modified to a SOM of ENTIRE tasks in, e.g., steel or telecommunications industries. In addition, the behavior of industry fields could be simulated, for instance, in changing environments.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

References

1 Bishop, C.M., (1995), *Neural Networks for Pattern Recognition*, Oxford University Press.

2 Brachman, R.J. and Anand, T., (1996), The Process of Knowledge Discovery in Databases, *Advances in Knowledge Discovery and Data Mining*, edited by Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., AAAI Press / MIT Press, California.

3 Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., (1996), editors, *Advances in Knowledge Discovery and Data Mining*, AAAI Press / MIT Press, California.

4 Goser, K., Metzen, S., and Tryba, V., (1989), Designing of Basic Integrated Circuits by Self-Organizing Feature Maps. *Neuro-Nîmes*, Int. Workshop on Neural Networks and Their Applications, Nanterre, France, pp. 225-235.

5 Hastie, T. and Stuetzle, W., (1989), Principal Curves, *Journal of the American Statistical Association*, 84, pp. 502-516.

6 Hollmén, J. and Simula, O., (1996), Prediction models and sensitivity analysis of industrial production process parameters by using the self-organizing map, *IEEE Nordic Signal Processing Symposium Proceedings*, pp. 79-82.

7 Iivarinen, J., Kohonen, T., Kangas, J., and Kaski, S., (1994), Visualizing the clusters on the self-organizing map, *Proc. Conf. on Artificial Intelligence Res. in Finland*, edited by Carlsson, C., Järvi, T. and Reponen, T., Number 12 in Conf. Proc. of Finnish Artificial Intelligence Society, pp. 122-126, Helsinki.

8 Kaski, S., (1997), *Data Exploration Using Self-Organizing Maps*, Ph.D. thesis, Helsinki University of Technology.

9 Kohonen, T., (1995), *Self-Organizing Maps*, Springer, Berlin.

10 Kohonen, T., Oja, E., Simula, O., Visa, A., and Kangas, J., (1996), Engineering applications of the self-organizing map, *Proceedings of the IEEE*, 84(10), pp. 1358-1384.

11 Kraaijveld, M.A., Mao, J. and Jain, A. K. (1995), A nonlinear projection method based on Kohonen's topology preserving maps, *IEEE Transactions on Neural Networks*, 6(3), pp. 548-559.

12 Luttrell, S.P., (1989), Hierarchical self-organizing networks, *Proceedings of the 1st IEE Conf. on Artificial Neural Networks*, Savoy Place, London.

13 Sammon, J.W. Jr., (1969), A nonlinear mapping for data structure analysis, *IEEE Transactions on Computers*, C-18(5), pp. 401-409.

14 Murtagh, F., (1995), Interpreting the Kohonen self-organizing feature map using contiguity-constrained clustering, *Pattern Recognition Letters*, 16, pp. 399-408.

15 Raivio, K., Simula, O., and Henriksson, J., (1991), Improving decision feedback equalizer performance using neural networks, *Electronics Letters*, 27(23), pp. 2151-2153.

16 Ritter, H., Martinetz, T., and Schulten, K., (1992), *Neural Computation and Self-Organizing Maps*, Addison-Wesley Publishing Company.

17 Simula, O., Alhoniemi, E., Hollmén, J., and Vesanto, J., (1996), Monitoring and modeling of complex processes using hierarchical self-organizing maps, *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'96)*, volume Supplement, pp. 73-76.

18 Simula, O. and Kangas, J., (1995), Process monitoring and visualization using self-organizing maps, *Neural Networks for Chemical Engineers*, volume 6 of *Computer-Aided Chemical Engineering*, chapter 14, Elsevier, Amsterdam.

19 Singer, A.C., Wornell, G.W., and Oppenheim, A.V., (1992), A nonlinear signal modeling paradigm, In *Proc. of ICASSP*.

20 Haitao Tang and Simula, O., (1996), The optimal utilization of multi-service scp, *Intelligent Networks and New Technologies*, pp. 175-188, Chapman & Hall.

21 Tryba, V., and Goser, K., (1991), Self-Organizing Feature Maps for process control in chemistry, *Artificial Neural Networks*, edited by Kohonen, T., Mäkisara, K., Simula, O., and Kangas, J., pp. 847-852, Amsterdam, Netherlands.

22 Ultsch, A., (1993), Self-organized feature maps for monitoring and knowledge acquisition of a chemical process, *Proc. ICANN'93 Int. Conf. on Artificial Neural Networks*, edited by Gielen, S. and Kappen, B., pp. 864-867, Springer, London, UK.

23 Ultsch, A. and Siemon, H.P., (1990), Kohonen's self organizing feature maps for exploratory data analysis, *Proc. INNC'90, Int. Neural Network Conf.*, pp. 305-308, Kluwer, Dordrecht, Netherlands.

24 van Gils, M.J., (1995), *Peak Identification in Auditory Evoked Potentials Using Artificial Neural Networks*, Ph.D. thesis, Eindhoven University of Technology.

25 Varfis, A. and Versino, C., (1992), Clustering of socio-economic data with kohonen maps, *Neural Network World*.

26 Vesanto, J., (1997), *Data mining techniques based on the self-organizing map*, Master's thesis, Helsinki University of Technology.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Chapter 5
A Self-Organizing Architecture for Invariant 3-D Object Learning and Recognition from Multiple 2-D Views

Stephen Grossberg[1]

[1]Supported in part by the Office of Naval Research (ONR N00014-95-1-0409 and ONR N00014-95-1-0657).

Department of Cognitive and Neural Systems
and Center for Adaptive Systems
Boston University
Boston, MA 02215

Gary Bradski[2]

[2]Supported in part by the National Science Foundation (NSF IRI-90-24877) and the Office of Naval Research (ONR N00014-92-J-1309).

Department of Cognitive and Neural Systems[3] and
Intel Corporation
Microcomputer Research Labs, RNB 6-35
2200 Mission College Boulevard
Santa Clara, CA 95052-8119

[3]Acknowledgments: The authors wish to thank Diana Meyers for her assistance in preparing the manuscript.

A family of self-organizing neural architectures, called VIEWNET, is developed for learning to recognize 3-D objects from sequences of their 2-D views. VIEWNET architectures use View Information Encoded With NETworks to accomplish this task. VIEWNET incorporates a preprocessor that generates a compressed but 2-D invariant representation of an image, a supervised incremental learning system (Fuzzy ARTMAP) that

classifies the preprocessed representations into 2-D view categories whose outputs are combined into 3-D invariant object categories, and a working memory that makes a, 3-D object prediction by accumulating evidence over time from 3-D object category nodes as multiple 2-D views are experienced. Fuzzy ARTMAP was modified to enable probability learning of the object classes. VIEWNET was benchmarked on an MIT Lincoln Laboratory database of 128 × 128 2-D views of aircraft, including small frontal views, with and without additive noise. A recognition rate of up to 90% was achieved with one 2-D view and up to 98.5% correct with three 2-D views. The properties of 2-D view and 3-D object category nodes are compared with those of cells in monkey inferotemporal cortex.

1. Introduction: Transforming Variant Image Data into Invariant Object Predictions

This chapter describes a class of self-organizing fuzzy neural architectures, generically called VIEWNET, wherein spatially and temporally variant, noisy, and incomplete data may be transformed into invariant recognition events. VIEWNET processing stages compute increasingly stable and invariant representations of fluctuating and uncertain data. No one processing stage, in itself, accomplishes the transformation of these data into an invariant code.

The simplest type of VIEWNET architecture, called VIEWNET 1, is capable of learning invariant representations of 3-D objects from sequences of their 2-D views (Bradski and Grossberg, 1993, 1995). VIEWNET architectures may be generalized to accomplish more complex tasks of scene understanding.

The VIEWNET 1 architecture incorporates a preprocessor that generates a compressed but 2-D invariant representation of an image, a supervised incremental learning system that classifies the preprocessed representations into 2-D view categories whose outputs are combined into 3-D invariant object categories, and a working memory that makes a 3-D object prediction by accumulating evidence from 3-D object category nodes as multiple 2-D views are experienced.

The VIEWNET 1 preprocessor includes the CORT-X 2 filter, which discounts the illuminant, regularizes and completes figural boundaries, and suppresses image noise. This boundary segmentation is rendered invariant under 2-D translation, rotation, and dilation by use of a log-polar transform. The invariant spectra undergo Gaussian coarse coding to further reduce noise, compensate for small shifts in log-polar processing, partially overcome 3-D foreshortening effects, and increase generalization.

These compressed codes are input into the classifier, a supervised learning system based on the Fuzzy ARTMAP algorithm. Fuzzy ARTMAP learns compressed 2-D view categories that are invariant under 2-D image translation, rotation, and dilation as well as 3-D image transformations that do not cause a predictive error. Evidence from sequences of 2-D view categories converges at 3-D object nodes that generate a response which is invariant under changes of 2-D view. The properties of these 2-D view and 3-D object category nodes may be compared with those of analogous cells in monkey inferotemporal cortex.

These 3-D object nodes input to a working memory that accumulates evidence over time to improve object recognition. In the simplest working memory, each occurrence (nonoccurrence) of a 2-D view category increases (decreases) the corresponding node’s activity in working memory. The maximally active node is used to predict the 3-D object.

Recognition has been studied with noisy and clean images using both slow and fast learning. Slow learning at the Fuzzy ARTMAP map field is adapted to learn the conditional probability of the 3-D object given the selected 2-D view category. VIEWNET 1 was originally benchmarked on an MIT Lincoln Laboratory database of 128 × 128 2-D views of aircraft with and without additive noise. A recognition rate of up to 90% was achieved with one 2-D view and of up to 98.5% correct with three 2-D views.

2. The Tradeoff Between Preprocessor, Classifier, and Accumulator

Seibert and Waxman (1990a, 1990b, 1991, 1992) pioneered the development, of neural network architectures that self-organize representations of 3-D objects from 2-D view sequences. Since the work on VIEWNET was inspired by their model and uses the same data base to benchmark its results, their model is first reviewed in some detail.

Seibert and Waxman relied on building a neural “cross-correlation matrix” as shown in Figure 1. This network provides a realization of the *aspect graph* concept of Koenerink and van Doorn (1979). An aspect graph represents 2-D views of a 3-D object by nodes of a graph, and legal transitions between nodes by the arcs between them. The Seibert-Waxman cross-correlation matrix was used to learn both 2-D views and 2-D

view transitions and to associate the 2-D views and view transitions with the 3-D objects that produced them. To test their architecture, the 2-D views were taken from 3-D airplane models painted black and digitized against a light background for ease of segmentation.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Sequences of images were obtained by rotating the airplanes 360° for each of the many angles of inclination that the video camera was set at, from 0° (horizontal) to 72° (looking down at the spinning jet). These images were then binarized using a high threshold to remove noise. Points of high curvature and the object centroid were found using a reaction-diffusion process. A log-polar transform around the object centroid was used to remove 2-D rotation and scale variations. The output of this filter was (approximately) invariant under 2-D translations, rotations, and dilations of the image. In order to compress this invariant spectrum and reduce 3-D foreshortening effects, the result was coarse coded (compressed to 5 × 5 pixels from 128 × 128) using Gaussian filters.

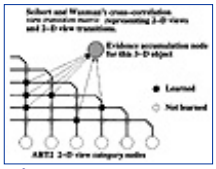


Figure 1 View Transition Matrix from the architecture of Seibert and Waxman computes cross-correlations between its input 2-D view categories. The matrix represents the correlation between the present categorical 2-D view and the decaying activation of the previous view. As preprocessed images enter ART 2, the resulting categorical views excite learned weights in this cross-correlation "aspect graph" devoted to each 3-D object. An evidence node integrates activation from the learned connections in the matrix. Another network (not shown) chooses the evidence node with the highest accumulated activation as the 3-D object most likely being viewed by the architecture. [Reprinted with permission from Bradski and Grossberg, 1995.]

The coarse coded vectors (25 data points) were fed into an ART 2 (Carpenter and Grossberg, 1987) network for unsupervised learning and categorization. These "categorical" 2-D views further compressed the 2-D representation so that a new 2-D view category was chosen only if significant changes occurred in the 2-D appearance of the object that were not invariant under translation, rotation, dilation, or modest foreshortening. How much change was tolerated was controlled by the ART 2 vigilance parameter. These 2-D view categories were then fed into a series of cross-correlation matrices, or view graphs, one for each possible 3-D object, so that views and view transitions could be learned by a 3-D object categorization layer. The 3-D categorization layer incorporated "evidence accumulation" nodes which integrate activations that they receive from learned connections to the correlation matrix. Decay terms in these integrator nodes determine how long they stay active without input support.

Seibert and Waxman's approach of automatically generating aspect graphs directly from the imagery that the architecture experiences vastly simplifies earlier attempts which generated aspect graphs by constructing projections from mathematical descriptions of the 3-D objects. However, using view transition information comes at a cost; given N 2-D views and M objects, the architecture must have the potential to encode on order of $N^2 \times M$ 2-D view transitions and the corresponding adaptive weights. Another potential problem is that an error in identifying a 2-D view may introduce a spurious 2-D view transition. Finally, unless one presumes a 2-D view frame capture rate fast enough to capture the highest, speed movement that an object can make, view transitions may be skipped inadvertently by fast object motion.

As reported by Seibert and Waxman (1992), 75% of the 2-D airplane images were ambiguous to some degree. That is, 75% of the 2-D view categories formed by ART 2 gave evidence for more than one type of airplane. Two possible reasons for this level of ambiguity exist: (1) image preprocessing using high curvature points followed by coarse coding may lump together object features that are needed for unambiguous recognition; (2) the 2-D views were categorized by ART 2 without using any supervised feedback to help correct category boundaries. Although most, 2-D view categories were ambiguous, the transitions between them were used to unambiguously identify a particular 3-D object. Thus, view transitions are critically important in the Seibert and Waxman architecture, which may then incur the cost of needing up to $N^2 \times M$ view transition correlation matrices.

This analysis suggests that a tradeoff exists between the choice of preprocessor, learned categorizer, and evidence accumulation parts of the architecture. If the preprocessor and categorizer generate 2-D view categories that are too coarse or ambiguous, then the evidence accumulation network may have to be enhanced to overcome these limitations. VIEWNET explores this tradeoff by using a different preprocessor and categorizer that generate less ambiguous 2-D view categories. In Fact, VIEWNET can categorize individual views on the Seibert-Waxman data base with high accuracy (up to 90%), in accord with the human experience that many objects can be identified with a single view, except, when they are observed from an unfamiliar perspective or from a perspective that reduces the object's apparent dimension. A computationally less costly evidence accumulation, or working memory, network could then be used, at least on these data. The general problem is to design the optimally balanced preprocessor, categorizer, and working memory networks to handle the largest possible set of images.

3. VIEWNET 1 Heuristics

As diagramed in Figure 2, VIEWNET consists of three parts: an image pre-processor, a self-organizing recognition network that may operate in either unsupervised or supervised modes, and a working memory network to accumulate evidence over multiple views. It is assumed that the figure to be recognized has already been separated from its background. Neural networks for figure-ground separation that use computations consistent with those in the VIEWNET pre-processor were described by Grossberg (1994) and Grossberg and Wyse (1991; 1992). The image figure is then processed by a boundary segmentation network, called the CORT-X 2 filter (Carpenter, Grossberg, and Mehanian, 1989; Grossberg and Wyse, 1991; 1992). CORT-X 2 is a feed-forward network that first compensates for variable illumination, extracts ratio contrasts, and normalizes image intensities. It then suppresses image noise while it completes and regularizes a boundary segmentation of the figure. Thus, the maximal curvature point representation of Seibert and Waxman is replaced by an illumination-compensated, noise-sup pressed boundary segmentation of the entire figure.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

This boundary segmentation is then rendered invariant under 2-D rotation, translation, and scale by a centering, log-polar, centering operation as described by Schwartz (1977). As in Seibert and Waxman, the resulting spectra are then compressed by coarse coding to gain some insensitivity to pixel shift and 3-D deformation effects and to reduce memory requirements. Coarse coding was done by two methods in Bradski and Grossberg (1995) whose performance was compared: a many-to-few pixel simple spatial averaging, and Gaussian spatial averaging. The latter generated somewhat better results for reasons summarized below. The output of this preprocessor is a coarse-coded, invariant spectrum of an illumination-compensated, noise-suppressed boundary segmentation. This representation provides the input vectors to the self-organizing neural network classifier.

Fuzzy ARTMAP (Carpenter, Grossberg, Markuzon, Reynolds, and Rosen, 1992) was used to categorize the output spectra. This architecture is capable of fast, stable learning of recognition categories in response to nonstationary multidimensional data, and of learning to generate many-to-one output predictions from the recognition categories to output labels. Fuzzy ARTMAP runs under either unsupervised or supervised learning conditions. Under supervised conditions, erroneous predictions trigger further hypothesis testing, or memory search, in the input classifier. Fuzzy ARTMAP converts the vigilance parameter of unsupervised ART classifiers, such as ART 2, into an internally controlled parameter. When an erroneous prediction occurs, vigilance is increased just enough to trigger a new bout of hypothesis testing to discover a better category. Multiple 2-D views can be compressed into a single 2-D view category by this mechanism.

This control scheme is called *match tracking* because the vigilance parameter tracks the match value that encodes how well the selected category's prototype matches the input spectrum. Using match tracking, memory search discovers and learns recognition categories that conjointly maximize code compression and minimize predictive error. Fuzzy ARTMAP can hereby use supervised learning to rapidly fit the number, size, and shape of input categories to the statistical demands of the environment. This added power helps Fuzzy ARTMAP to learn 2-D view categories that tend to fit the data better than ART 2. It is this combination of boundary segmentation preprocessing combined with supervised ART learning that achieves correct prediction of up to 90% accuracy in response to a single airplane view.

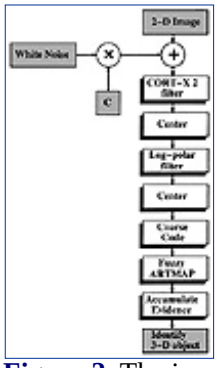


Figure 2 The image processing flow chart of the VIEWNET system. [Reprinted with permission from Bradski and Grossberg, 1995.]

Figure 3 The two left-hand pictures are frontal views of an F-16. The right-hand picture is of an HK-1. [Reprinted with permission from Bradski arid Grossberg, 1995.]

In the VIEWNET 1 architecture, Fuzzy ARTMAP automatically combines the outputs of 2-D view categories at 3-D object category nodes that are invariant under changes of experienced 2-D views. This learned fusion of 2-D view categories into invariant 3-D object categories occurs at the Fuzzy ARTMAP *map field* that is defined below. A similar type of hierarchical organization from 2-D view to 3-D object has been reported in neurophysiological studies of cell responses in monkey inferotemporal cortex, where some cells respond to individual 2-D views whereas others, like 3-D object nodes, respond to a wide range of views (Logothetis *et al*., 1994). These studies were motivated by the regularization networks of Poggio and Girosi (1990) which also add up responses from 2-D views at 3-D object nodes. These networks do not, however, incrementally learn their categories in real-time and have not yet been incorporated into a self-organizing image processing architecture.

Given the high accuracy attained by individual 2-D view categories, the simplest possible working memory was used in VIEWNET 1 to illustrate the tradeoff between preprocessor, categorizer, and working memory. No view transitions were used. In fact, no temporal order information was used. Instead, the working memory simply updated its representation of each 3-D object category every time one of its 2-D views was experienced, and the 3-D object was predicted by voting for the winning 3-D category. Using this scheme, voting with two views achieves up to 94%, and with three views up to 98.5% accuracy. It was thereby shown that the simplest evidence accumulation from multiple views, without view transitions or even a representation of view temporal orders, can lead to high recognition on the Seibert-Waxman database if the preprocessor and classifier are designed as indicated.

The remainder of this chapter describes these operations in more detail. Section 4 describes the airplane database. Section 5 describes the CORT-X 2 boundary segmentation network. The Appendix describes its equations. Section 6 describes the operations to generate an invariant representation of the boundary segmentation. Section 7 describes the coarse coding algorithm. Section 8 describes the Fuzzy ARTMAP network. The Appendices describe the equations. Section 9 describes computer simulation results in response to a single view, using fast or slow learning with or without image noise. The results are robust across all simulation conditions. Section 10 summarizes how multiple views may improve recognition scores. Section 11 provides concluding remarks.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

4. Simulated Database

Bradski and Grossberg (1995) tested VIEWNET 1 on a database consisting of multiple 2-D images of three airplanes: an F-16, F-18, and HK-1. Michael Seibert and Allen Waxman of MIT Lincoln Laboratory kindly provided access to this database. Video images were taken of three models of these airplanes. Each was painted black and suspended by string against a light background to aid in segmentation. The camera was mounted anywhere in an arc around the airplanes that started at 0.0° above horizontal and went in increments of 4.5° to a maximum of 72.0° above horizontal. For each camera angle, the airplanes were spun and frames covering one full revolution (an average of 88 frames) were retained resulting in 1200 to 1400 images per object. The images themselves were 128 × 128 pixel gray scale. The images were then thresholded and binarized into a SUN raster format to form the “raw” database. For our processing, data was turned into a floating point format scaled between 0.0 and 1.0 and an additive noise process was introduced. The noise consisted of a 128 × 128 pixel images with each pixel taken from a uniform distribution between 0.0 and 1.0 scaled by a constant C e 0.0. That is, every pixel in the noise image is multiplied by C to create a “scaled” noise image. Thus, if $C = 0.5$, the noise image would consist of pixels that varied randomly in amplitude with uniform distribution between 0.0 and 0.5. These scaled, 128 × 128 noise images were then added to the 128 × 128 airplane images prior to preprocessing. Thus, both noise-free and noisy 2-D views covering a half-sphere surrounding the 3-D object were collected, keeping their spatial relationships intact.

Even-numbered rotation images from each camera angle were taken as the training set, with the odd-numbered images forming the test set. The system was trained using random walks over the half-sphere of training images. Testing was done using random walks over the half-sphere of test images so that the paths taken and views seen were never the same between the training and test sets. Problem difficulty derives in part from the existence of small ambiguous frontal views in the data base, as in Figure 3.

5. CORT-X 2 Filter

The CORT-X 2 filter was used to preprocess the 128 × 128 airplane images. This is a feed-forward network that detects, regularizes, and completes image boundaries from edge, texture, and shading contrasts, while suppressing noise that does not have an underlying anisotropic structure. The CORT-X 2 filter is an enhancement of the original CORT-X filter (Carpenter, Grossberg, and Mehanian, 1989). It generates better boundary segmentations, deals better with noise, and may also be used for figure-ground separation. The

figure-ground separation properties were not needed in this research because the data images were already separated from their backgrounds. These CORT-X filters are simplifications of the Boundary Contour System (or BCS) for boundary segmentation (Grossberg, 1987; 1994; Grossberg and Mingolla, 1985; Grossberg, Mingolla, and Todorović, 1989; Grossberg, Mingolla, and Williamson, 1994). The BCS includes internal feedback loops to generate coherent boundary completions even over image regions defined by sparse image contrasts. A full BCS system can be inserted into the VIEWNET architecture to handle a wider class of imagery.

The CORT-X 2 filter incorporates a number of features that are useful for processing noisy imagery, including ON and OFF cells and multiple size scales, that respond to images in complementary ways. The CORT-X 2 filter embodies a computational strategy for combining these complementary computations to achieve enhanced image processing. The processing stages of the CORT-X 2 filter are schematized in Figure 4 and summarized below. Equations and parameters are listed in Appendix A.

Step 1: Discount the Illuminant. The first processing stage compensates for variable illumination, and thereby extracts ratio contrasts from the image while normalizing the overall activity level in each image region. Both ON cells and OFF cells process the image in this way, using parallel shunting on-center/off-surround ("ON-C") and off-center/on-surround ("OFF-C") networks. Parameters are set so that the ON-C network has a zero baseline activity and the OFF-C network has a positive baseline activity. The OFF-C filter performs an image inversion because it has a positive baseline activity that is inhibited by positive signal values in the image. Figure 5 shows a noise-free image as well as the ON-C and OFF-C outputs.

Along straight contrast boundaries in an image, both the ON-C and OFF-C networks enhance the contrast. On the other hand, the ON-C network has a stronger response to concave corners of activity in an image than the OFF-C network, while the converse is true at convex corners, as was noted by Grossberg and Todorović (1988). These complementary responses are joined at later processing stages used to build more complete boundary segmentations.

Step 2: Boundary Segmentation. The CORT-X 2 filter transforms the normalized ON-C and OFF-C images into a boundary segmentation using fast feed-forward computations with oriented contrast-sensitive cells. Each processing stage possesses a full range of oriented cells as at each pixel in the image.

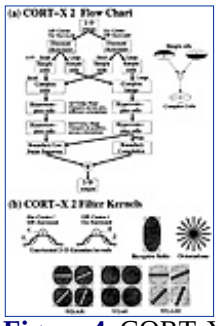


Figure 4 CORT-X 2 flow chart and filter kernels. The image is processed in parallel with small and large scale filters. Grey areas in the kernels are the active regions. All kernels are normalized to have an area equal to 1. [Reprinted with permission from Bradski and Grossberg, 1995.]

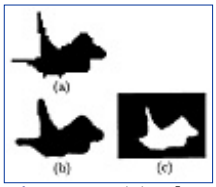


Figure 5 (a) The original F16 image. (b) CORT-X 2 ON-C output. (c) CORT-X 2 OFF-C output. [Reprinted with permission from Bradski and Grossberg, 1995.]

Image processing is done in parallel using two sets of convolution kernels at two different size scales. As noted in Carpenter, Grossberg, and Mehanian (1989), larger scale oriented cells are better able to complete gaps in image boundaries and to suppress noise. On the other hand, smaller scale oriented cells do a better job of boundary localization than do larger scale cells. Interactions between the two size scales are designed to generate boundary outputs that combine good localization, boundary completion, and noise suppression properties.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

The first stage, called the *simple cell layer*, consists of oriented contrast detectors that are sensitive to the orientation, amount, direction, and spatial scale of image contrast at a given image location. The orientation sensitivity results from using an elliptically shaped kernel, or input field, one for each of eight orientations spaced 45° apart that operate in parallel at each position in the image. Sensitivity to direct ion-of-contrast results from a kernel in which one half is excitatory and the other half inhibitory. At each orientation, a pair of detectors sensitive to opposite directions-of-contrast processes the image. The net activity of each detector is rectified, giving rise to a half-wave rectified output signal. Figure 7 shows the results of processing the ON-C (Figure 6a) and OFF-C (Figure 6b) image with the small spatial scale (6 × 3 pixels) simple cell layer. The line lengths in the figure indicate the magnitude of the simple cell responses at each orientation and position.

The next processing stage generates a cell type whose output is insensitive to direction-of-contrast, or contrast polarity. Such an operation enables image boundaries to bridge textured and shaded image regions where contrast polarity reverses. This *complex cell layer* combines outputs from the simple cells at each position as shown in Figure 5. Complex cells perform a full-wave rectification of the image that is sensitive to the orientation, amount, and spatial scale of the contrast in the image, but not to its direction-of-contrast. To achieve this, complex cells sum up the half-wave rectified outputs of like-oriented and scaled simple cells of both directions-of-contrast at each position from both the ON-C and OFF-C networks. This is done in two parallel circuits at both the small and large spatial scales (see Figure 4).

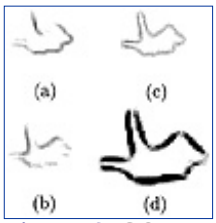


Figure 6 CORT-X 2 processing. (a) Output resulting from the ON-C network, using left-sided elliptical filters (simple cell output) with a small spatial scale (6 × 3 pixels). A “left-sided” filter refers to filters that respond to a vertical left-to-right, high-to-low contrast transition area in the image when the filter is in vertical orientation. A “right-sided” filter is the opposite. Lines in the figure are proportional to the magnitude of the response at each orientation at each position. (b) Output from the OFF-C network using small left-sided elliptical filters. (c) Hypercomplex cell output for the small scale. (d) The final CORT-X 2 output. [Reprinted with permission from Bradski and Grossberg, 1995.]

Complex cells excite hypercomplex cells in the next layer at their position and orientation while inhibiting hypercomplex cells at nearby locations that are not colinear with the complex cell's orientation. This positional competition is called the *first competitive stage*. It positionally sharpens the location of the segmentation, especially in response to textured and shaded images. Figure 6c shows the output of the hypercomplex cell layer for the small scale.

The next layer, called the *second competitive stage*, chooses the hypercomplex cell whose orientation is maximally activated to represent the activity at each position. These orientationally favored hypercomplex cells are often called higher-order hypercomplex cells in the full BCS. Figure 7a displays the small scale output of these higher-order cells.

The final stages of CORT-X 2 involve cooperative interactions between the large and small scale filters to join together the better boundary completion and noise suppression properties of the larger scale cells and the better localization properties of the smaller scale cells. One type of cooperative interaction is used to complete boundaries across boundary gaps caused by image noise. In the full BCS, boundary completion bridges gaps between distant image contrasts using a feedback loop that includes another cell type, called the bipole cell. Lacking a feedback loop, the CORT-X 2 filter uses a simplified interaction that captures the main heuristic for completing boundaries across small image gaps. In particular, cooperative interactions among the higher-order hypercomplex cells activate an inactive cell if enough cells that share the inactive cell's orientation are active on both sides of its oriented axis.

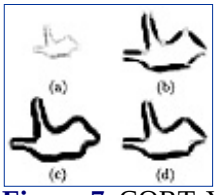


Figure 7 CORT-X 2 final stage outputs. (a) The maximal orientations of the hypercomplex cells (second competitive stage). (b) The long-range boundary completion output. (c) The multiple scale interaction output. (d) The final CORT-X 2 output from the additive combination of the top right and bottom left outputs. [Reprinted with permission from Bradski and Grossberg, 1995.]

Another type of cooperative interaction combines large and small scales in such a way that the better localization properties of the smaller scale filters have an effect only within regions where the larger scales have located a boundary. The output of the boundary-completing cooperative cells is shown in Figure 7b. Figure 7c displays the multiple scale cooperative interaction, and Figure 7d shows the final CORT-X 2 output consisting of the sum of output of the boundary-completion and multiple scale localization interactions. Figure 8 shows the results of processing images with two levels of additive noise: $C = 0.5$ or 50% noise (a), and $C = 1.0$, or 100% noise (b). Because of the relative simplicity of the images being processed, only amplitude information was used. The CORT-X filter also computes potentially useful information about the relative orientation of different object parts, as in Figure 7a. The boundary completion capabilities of the CORT-X filter are also not greatly taxed by these images.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

6. Translation, Rotation, and Scale Invariance

The next processing stages generate a representation, using standard methods, that is invariant under 2-D translation, rotation, and dilation. First, the 2-D boundary segmentation is centered by dividing its 1st moments by its 0th moment to find the figure centroid, subtracting off the center of the image, and then shifting the figure by this amount. A log-polar transform is then taken with respect to the center of the image. Each point (x, y) is represented as $re^{i\theta}$. Taking the logarithm yields coordinates of log radial magnitude and angle. As is well known (Schwartz, 1977), figural sizes and rotations are converted into figural shifts under log-polar transformation. Using these shift parameters to center the log-polar transformed image leads to a figural representation that is invariant under 2-D changes in position, size and rotation. Figure 9 shows the results of processing two F-18 images which are identical except for scaling and rotation. The images become very similar in the centered log-polar domain.

7. Two Coarse-Coding Strategies

Coarse coding reduces memory requirements, as it compensates for modest 3-D foreshortening effects and inaccuracies of figural alignment in the invariant filter. Coarse coding by averaging in the space domain is equivalent to low pass filtering in the frequency domain. Neighboring pixel features are thereby blurred, compensating for slight alignment variations. In addition, 2-D images of 3-D objects suffer from 3-D perspective distortions that cannot be corrected by log-polar transforms. These are viewpoint-specific foreshortening effects and self-occlusions. The blurring associated with coarse coding can help to increase generalization by causing foreshortened and non-foreshortened images to map to nearly the same image. On the other hand, too much blurring can obscure critical input features and thereby harm recognition performance. Our analysis suggests how to balance these effects to maximize the benefits of coarse coding.

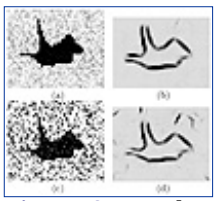


Figure 8 Results of processing noisy images with CORT-X 2. Uniform random noise was added to every

pixel in the original image. The original image (left column) had pixels with activity levels between 0.0 and 1.0. Uniform random noise with pixel values ranging between 0.0 and 1.0 was scaled by *C* and added to the clean image prior to processing by CORT-X 2 with results shown in the right column. At top (a,b), random noise between 0.0 and 0.5 (*C* = 0.5 or 50% noise) was added, on the bottom (c,d), noise between 0.0 and 1.0 (*C* = 1.0 or 100% noise) was added. For example, 100% noise refers to the fact that the magnitude of any given noise pixel can be as large as the largest magnitude pixel in the noise-free image. [Reprinted with permission from Bradski and Grossberg, 1995.]

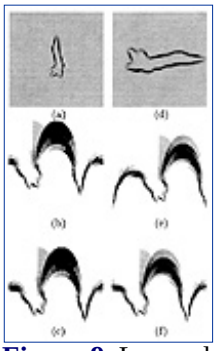


Figure 9 Log-polar transform example. At top (a,d) are the results of processing F-18 images at two different scales and orientations using CORT-X 2. The middle images (b,e) show the results of a log-polar transform of the top images. At bottom (c,f) are the centered log-polar images that have been made more identical in the sense that, overall, many more (*x*, *y*) pixel locations obtain similar values between the two images after processing. In the log-polar images, the ordinate is the log-radial magnitude and the abscissa is the periodic angle axis. [Reprinted with permission from Bradski and Grossberg, 1995.]

Data reduction is an important practical issue in its own right in many realistic pattern recognition problems. Seibert and Waxman's database contains 4300 images of three distinct objects. Each image is made up of 128 × 128 floating point pixels of 4 bytes each for a total of 128 × 128 × 4 × 4300 = 281, 804, 800 bytes. Yet, as reported below, object identification performance based on single 2-D views did well even when images were reduced to just 4 × 4 pixels, yielding a database of just 4 × 4 × 4 × 4300 = 275, 200 bytes. This reduction affords an enormous saving in both computation and memory storage.

Coarse coding of the 2-D images used a spatial averaging method that preserves sufficient information for accurate recognition. This method was selected as follows. Spatial averaging consists of convolving the original image *I* with a function ¨ and then sampling the resultant image with delta functions spaced every *T* pixels: ´(*x* - *nT*, *y* - *kT*). For simplicity, in 1-D this is

$$(I * \Psi) \cdot \sum_{n=-\infty}^{\infty} \delta(x - nT) \qquad (1)$$

If the Fourier transform of *I* is $\hat{I}$, and that of ¨ is $\hat{\Psi}$, then the Fourier transform of equation (1) is

$$(\hat{I} \cdot \hat{\Psi}) * \frac{2\pi}{T} \sum_{k=-\infty}^{\infty} \delta(\Omega - k\Omega_s) \qquad (2)$$

where $©_s = 2À/T$, and *T* is the sampling period in pixels. If $©_N$ is the highest frequency in the image, then for the image to be uniquely determined by its samples, we must have by the Nyquist sampling theorem that

$$\Omega_s = \frac{2\pi}{T} > 2\Omega_N \qquad (3)$$

Two simple spatial averaging functions ¨ are: (1) uniform averaging of the input image so that all pixels in a window of some width are summed and divided by the number of pixels in the window; (2) Gaussian averaging of the input image so that a normalized Gaussian weighted sum of all pixels is taken over a window of some width. Both approaches were investigated by Bradski and Grossberg (1995), who noted that method (1) is a rectangular filter in space, and thus a sinc function in frequency. The side lobes of the sinc function can introduce high-frequency aliasing ("ringing") in the resultant image, which can be reduced by using a function that provides a better low pass filter, such as the Gaussian filter method (2). The Gaussian filter has the further advantage of being an eigenfunction of a Fourier transform, since the Fourier transform of a Gaussian is a Gaussian with reciprocal variance, thereby simplifying calculation.

To optimize Gaussian-based spatial averaging, we must determine how to best set the standard deviation Ã of the Gaussians. Let us define two standard deviations away from the Gaussian midpoint to be essentially zero. The cutoff frequency of such a low pass filter is then À/2Ã. Equation (3) then implies

$$\frac{2\pi}{T} > \frac{2\pi}{2\sigma} \tag{4}$$

which yields, at equality

$$\sigma = \frac{T}{2} \tag{5}$$

By (5), the standard deviation should be set to half the Gaussian center-to-center sampling period so that the zero point of each Gaussian just touches the center of the next Gaussian as in Figure 10c. Figures 10d-f show the results of coarse coding the image in Figure 10b in this way to reduce the original image of 128 × 128 pixels down to 16 × 16, 8 × 8, and 4 × 4 pixels, respectively.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

8. Fuzzy ARTMAP Classification

Fuzzy ARTMAP was used to learn 2-D view categories from the coarse-coded invariant spectrum of the noise-suppressed boundary segmentations. Fuzzy ARTMAP was chosen because it can achieve stable fast incremental learning of categories in response to unlimited amounts of nonstationary input data, can run in both unsupervised and supervised modes, and in its supervised mode can fit the size, number, and shape of its categories to the input statistics. Fuzzy ARTMAP parameters were chosen to enable the network to learn the conditional probability of the true 3-D object given the selected 2-D view category at the map field, which is now defined.

We utilized the simplified version of the Fuzzy ARTMAP network of Carpenter, Grossberg, Markuzon, Reynolds, and Rosen (1992) that was employed in Carpenter, Grossberg, and Iizuka (1992). This circuit consists of a Fuzzy ART module (Carpenter, Grossberg, and Rosen, 1991) ART_a that learns 2-D view categories and a field of 3-D object category output nodes F^b. The 2-D view and 3-D object category nodes are linked together by an associative memory F^{ab} that is called the *Map Field* (Figure 11). In supervised learning mode, Fuzzy ARTMAP receives a sequence of input pairs $(\mathbf{a}_p, \mathbf{b}_p)$ where $\mathbf{b}_p$ is the correct 3-D object class given the analog 2-D view input pattern $\mathbf{a}_p$. The ART_a module classifies analog input vectors $\mathbf{a}_p$ into categories and the Map Field makes associations from the ART_a categories to the outputs $\mathbf{b}_p$ in F^b. This simplified architecture thus does not include an ART_b module for independently clustering the outputs $\mathbf{b}_p$ into their own categories. In the present application, the ART_a categories are 2-D view categories and their associations converge at the map field upon invariant 3-D object categories. In the present application, the ART_a categories are 2-D view categories and their associations converge at the map field upon invariant 3-D object categories.

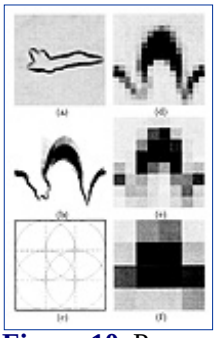


Figure 10 Preprocessing summary: (a) output of CORT-X 2 preprocessing; (b) centered log-polar image; (c) Gaussian coarse coding pattern; (d-f) coarse coding reduction from 128 × 128 pixels down to 16 × 16, 8 × 8, and 4 × 4 pixels. [Reprinted with permission from Bradski and Grossberg, 1995.]

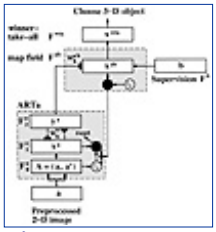


Figure 11 Fuzzy ARTMAP architecture. Each preprocessed 2-D input vector a is fed sequentially to the network as it becomes available. The inputs are complement coded, which transforms the *M*-vector a into the 2*M*-vector **A** = $(\mathbf{a}, \mathbf{f}\ \mathbf{a}_j^c)$ at field F_0^a, which is then fed into the input field F_1^a. A category node *k* is chosen at F_2^a which reads out its prediction to the Map Field via weights $\mathbf{w}_k^{ab}$. If the prediction is disconfirmed, a match tracking process is invoked in ART_a. Match tracking raises the ART_a vigilance $\bar{A}_{a?}$ to just above the match ratio $|\mathbf{x}^a|/|\mathbf{A}|$. This triggers an ART_a search which activates either a different existing category, or a previously uncommitted category node at F_2^a. After the search process concludes, F^{wta} chooses the maximally activated node in F^{ab} as the 3-D object being viewed. [Reprinted with permission from Bradski and Grossberg, 1995.]

Under supervised learning conditions, if $\mathbf{a}_p$ is categorized into an ART_a category that predicts an incorrect output $\mathbf{b}_p$, then the mismatch between actual and predicted $\mathbf{b}_p$ causes a memory search within ART_a via the *match tracking* mechanism. Match tracking raises the ART_a vigilance parameter $\bar{A}_a$ by the minimum amount that will trigger a memory search. In particular, $\bar{A}_a$ grows until it just exceeds the match value between the input vector $\mathbf{a}_p$ and the prototype of the active ART_a category. Since low vigilance leads to learning of large, coarse categories and high vigilance leads to learning of small, fine categories, match tracking sacrifices the minimum amount of category compression that is needed to correct each predictive error. Memory search by match tracking continues until a pre-existing ART_a category that predicts the correct ART_b category is found, or a new ART_a category is chosen.

After one of these conditions is satisfied, learning takes place both within ART_a and from the chosen ART_a category to the Map Field. Match tracking assures that predictive error is minimized while maintaining maximum generalization during fast or slow incremental learning conditions. Between learning trials, vigilance relaxes to its baseline vigilance $\bar{\rho}_a$. In test mode, input vectors $\mathbf{a}_p$ are classified by ART_a and the chosen category reads out its prediction to the Map Field. The index of the maximally activated node in the Map Field is taken to represent the predicted output class.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

The input vector $\mathbf{a}_p$ is *complement coded* before it activates ART_a. This preprocessing step enables the network to code both input features that are critically present and input features that are critically absent. In response to an input vector $\mathbf{a}_p$ complement coding delivers an input vector $\mathbf{A}_p = (\mathbf{a}_p, \mathbf{a}_p^c)$ to ART_a, where $\mathbf{a}_p^c = \mathbf{1} - \mathbf{a}_p$. Complement coding also normalizes the total input $\mathbf{A}_p$ to ART_a such that $\| \mathbf{A}_p \|_1 = 1$. It thereby prevents a category proliferation problem that could otherwise occur (Carpenter, Grossberg, and Rosen, 1991). Complement coding means intuitively that an input vector turns ON the cells corresponding to $\mathbf{a}_p$ as it turns OFF the cells corresponding to $\mathbf{a}_p^c$, much as in the ON and OFF channels of the CORT-X 2 filter. The algorithm is mathematically defined in Appendix B.

Table 1 The parameter set used for CORT-X 2 in the simulations. [Reprinted with permission from Bradski and Grossberg, 1995.]

Parameter	*Description*
$^{o}_{onC} = 7.0$	On-center magnitude
$\pm_{onC} = 1.3$	On-center standard deviation
$^{o}_{offS} = 3.333$	Off-surround magnitude
$\pm_{offS} = 1.875$	Off-surround standard deviation
$\overline{D} = B = 1.0$	Shunting values
$\overline{B} = D = 0.5$	Shunting values
$S = 0.2$	Spontaneous activity level
$A = 134$	Shunting decay
$\pm_1 = \pm_2 = 1.1$	Threshold contrast parameters
$^{2}_{1} = ^{2}_{2} = ^{2} = .003$	Threshold noise parameters
$F = 0.5$	Complex cell scaling constant
$\mu = 0.1$	Hypercomplex cell divisive offset
$¼ = 5.0$	Hypercomplex cell convolution scaling

Ä = 0.004	Hypercomplex cell threshold
´ = 0.001	Long range cooperation threshold
À/8	Oriented kernel orientation spacing
$(a_2, b_2)_{large} = (16, 8)$	Large set, large ellipse axis
$(a_1, b_1)_{large} = (10, 5)$	Large set, small ellipse axis
$(a_2, b_2)_{small} = (10, 5)$	Small set, large ellipse axis
$(a_1, b_1)_{small} = (6, 3)$	Small set, small ellipse axis
$G_1 = 2a_1/3$	Hypercomplex small kernel diameter
$G_2 = 2a_2/3$	Hypercomplex large kernel diameter
$U = 2a_2/5$	Multiple scale interaction kernel diameter
$O = 3a_2/5$	Long-range cooperation kernel length

Fuzzy ARTMAP parameters were chosen to allow for on-line slow learning from ART_a F_2^a to the Map Field nodes. A maximal ART_a vigilance level, $\bar{\rho}_{max}$ was introduced such that an error at the Map Field triggers match tracking only if match tracking leads to a vigilance $\rho_a \leq \bar{\rho}_{max}$. This bound prevents categories from becoming too small. In response to an error that would otherwise cause Á$_a$ to exceed $\bar{\rho}_{max}$, learning takes place instead from the active node in F_2^a to the Map Field. By setting the Map Field learning rate ²$_{ab}$, baseline $(\bar{\rho})$ and maximal $(\bar{\rho}_{max})$ vigilance levels appropriately, weights from F_2^a nodes to the Map Field may begin to approximate the conditional probability of the true class (the 3-D object) given the selected F_2^a category (the 2-D view category). A related approach to slow probability learning is described by Carpenter, Grossberg, and Reynolds (1993).

Table 2 The Fuzzy ARTMAP parameter set used for the simulations. [Reprinted with permission from Bradski and Grossberg, 1995.]

Parameter	**Description**
± = 0.6	Fuzzy ART search order
²$_a$ = 1.0	Fuzzy ART learning rate
²$_{ab}$ = 1.0	Map Field learning rate
$\bar{\rho}_{max} = 1.0$	Baseline Fuzzy ART vigilance Á$_a$
$\bar{\rho}_{max} = 1.0$	Maximum ART_a vigilance
Á$_{ab}$ = 1.0	Map Field vigilance

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

9. Computer Simulations

A computer simulation was run on the airplane database using the CORT-X 2 parameters shown in Table 1. The database was processed twice by CORT-X 2, once with a large pair of large and small oriented filters and once with a smaller pair of large and small filters. The large oriented filter pairs consisted of elliptical receptive fields with axes 16 × 8 and 10 × 5 pixels. The small oriented filter pair consisted of oriented ellipses with axes 10 × 5 and 6 × 3 pixels. This was done so that recognition results could be compared when images were processed at different scales. Coarse coding was done with both simple spatial averaging and Gaussian averaging, reducing the image down to 16 × 16, 8 × 8, and 4 × 4 pixels from an original size of 128 × 128. The window for simple spatial averaging was square with a width twice the sampling period *T*; that is, a window centered at one pixel extended until it just touched its neighboring pixels. The standard deviation for Gaussian averaging was set to *T*/2, as discussed in Section 7. Training and testing sets were assembled as discussed in Section 4 with even-numbered images forming the training set and odd numbered images forming the testing set. Except where explicitly mentioned, the simulations were run with the parameters shown in Tables 1 and 2.

The data were presented to the network in two different ways: (1) 2-D views were presented in the "natural" order in which they would appear if viewing the actual object in motion; (2) 2-D views were presented in random order. These two methods of data presentation were used to test whether presenting views in natural order helps recognition scores. Training in natural order consisted of 160 runs of from 1 to 50 views over each object. Training in random order consisted of a series of 40 runs of 100 training set views over each object. Recognition scores are taken as an average of 15 separate training-testing cycles.

Table 3 Percent of additive white noise surviving processing by CORT-X 2 and coarse coding. [Reprinted with permission from Bradski and Grossberg, 1995.]

% noise surviving CORT-X 2 filtering and Coarse Coding:		
	Large CORT-X 2 filters (16 × 8, 10 × 5)	*Small CORT-X 2 filters (10 × 5, 6 × 3)*
	1.79	2.42
After Gaussian coarse coding from 128 × 128 down to:		
16 × 16	0.33	0.34

8 × 8	0.23	0.29
4 × 4	0.19	0.26
After spatial average coarse coding from 128 × 128 down to:		
16 × 16	0.40	0.40
8 × 8	0.28	0.30
4 × 4	0.21	0.28

9.1 Fast Learning With and Without Noise

No clear advantage results from ordered presentation as compared to unordered presentation using noise-free data ($C = 0$) and fast learning, as shown by the results in Table 4. It can be seen that the smaller CORT-X 2 filter set resulted in better recognition performance overall and did better given more detail (less coarse coding).

The system was also tested with noisy data using additive white noise scaled by $C = 1.0$; that is, each pixel could have noise added to it less than or equal to the value of the maximum activity of pixels (= 1.0) in the original image. Table 3 shows what percent of the additive noise survives processing by CORT-X 2 alone, and by CORT-X 2 and coarse coding together, for two different filter sets and three different coarse codings. The percent noise surviving these transformations was measured by the following formula:

$$\max_{\forall(x,y)} \left[\frac{\Psi(\mathbf{I}+\mathbf{N}) - \Psi(\mathbf{I})}{C} \right] \times 100 \qquad (6)$$

where **I** is the image, **N** is the noise image, ¨ is the CORT-X 2 filter, $C > 0$ is the noise scaling parameter and (x, y) is the pixel index in the images. Table 3 represents the average results from ten measurements using Equation (6).

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

It can be seen that the combination of CORT-X 2 filtering followed by coarse coding is effective in reducing additive noise. Using a fast learning paradigm, the recognition results shown in Table 5 were similar to those for the noise-free case in Table 4, except for some minor falling off of recognition scores at the lowest level of coarse coding (the 16 × 16 case). Less coarse coding has the same effect on noise as raising the cutoff frequency of a low pass filter. Thus, as seen in Table 3, more noise gets through with less coarse coding, yielding slightly lower recognition performance.

Table 4 Recognition results on a noise free database ($C = 0$). CORT-X 2 filter sizes refers to the size of the oriented receptive field filters. CORT-X 2 was run twice using a larger and a smaller set of its large and small elliptical oriented filters. In the table, "Large" refers to the run with the larger set of oriented ellipses with axes 16 × 8 and 10 × 5 pixels; "Small" refers to the run with the smaller set of oriented ellipses with axes 10 × 5 and 6 × 3 pixels. Views were presented either in natural order or in random order. Data was coarse coded from 128 × 128 down to 4 × 4, 8 × 8, or 16 × 16 using simple spatial averaging or Gaussian averaging. Recognition scores refer to the percent of 2-D views correctly associated with a 3-D object. [Reprinted with permission from Bradski and Grossberg, 1995.]

CORT-X 2 filter set	*Data presentation*	*Coarse code using spatial avg / Gaussian avg*		
		4× 4	*8 × 8*	*16 × 16*
Small	Ordered	81.0/83.1	84.4/86.4	86.7/90.5
Small	Unordered	80.3/83.9	84.9/86.5	86.8/89.3
Large	Ordered	76.8/78.7	79.0/81.6	79.1/80.1
Large	Unordered	77.4/79.7	80.5/81.5	77.1/80.5

Table 5 Recognition results on noisy data ($C = 1$) with fast learning ($^2_{ab} = 1.0$). These results differ little from the noise-free results in Table 4 (no noise condition) with the exception of some consistent reduction in scores for the 16 × 16 coarse coding. [Reprinted with permission from Bradski and Grossberg, 1995.]

CORT-X 2 filter set	*Data presentation*	*Coarse code using spatial avg / Gaussian avg*		
		4 x 4	**8 x 8**	**16 x 16**
Small	Ordered	80.1/83.3	84.5/85.9	84.2/89.1
Small	Unordered	79.4/83.2	83.9/86.4	84.3/88.0

Large Large	Ordered Unordered	76.6/79.4 76.0/79.7	79.3/80.8 78.4/80.7	75.8/79.3 75.5/79.0

Table 6 Average number of ART_a categories formed during training for the simulations of Table 4 (no noise) and Table 5 (noise). The format in the table is as follows: [spatial avg.]/[Gaussian avg.] = [No noise, Noise]/[No noise, Noise]. [Reprinted with permission from Bradski and Grossberg, 1995.]

CORT-X 2 filter set	***Data presentation***	***Coarse code using spatial avg / Gaussian avg***		
		4 x 4	**8 x 8**	**16 x 16**
Small Small	Ordered Unordered	[172, 184]/[165, 169] [191, 198]/[175, 179]	[77, 73]/[70, 73] [76, 77]/[73, 76]	[34, 33]/[33, 35] [34, 35]/[35, 36]
Large Large	Ordered Unordered	[168, 179]/[160, 162] [183, 192]/[169, 174]	[71, 68]/[67, 71] [73, 75]/[69, 72]	[31, 33]/[30, 31] [32, 32]/[33, 32]

Table 6 shows the number of nodes created by the network after training for the no noise (left entry) and noise (right entry) results reported above. Noise causes a small increase in the number of categories formed on average as the network attempts to correct a greater number of noise-induced errors during supervised training.

9.2 Slow Learning Simulation With Noise

The network was also run on the noisy data using slow learning to the Map Field ($^2_{ab}$ = 0.2 in Equation (42)). Fast learning was still used within the ART_a module itself however (^{2}a = 1.0 in Equation (41)). In addition, a maximum vigilance level in ART_a $(\bar{\rho}_{max} = 0.95)$ was set so that when match tracking due to error feedback attempts to create an ART_a category smaller than a given size, no new category forms and learning takes place for the current category instead. With noisy inputs, rather than continually making new categories to correct for the noise, ART_a categories below a set prescribed size begin to learn the conditional probability of the true class (the 3-D object) given the selected ART_a category (the categorical 2-D view).

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

For $\bar{\rho}_{max} = 1.0$, the results for slow learning and fast learning to the Map Field are equivalent. They are equivalent because, with Map Field vigilance set to $Á_{ab} = 1.0$ as in Table 2, the slightest mismatch at the Map Field will invoke match tracking and a new category will be created. The main difference between slow and fast learning using $\bar{\rho}_{max} = 1.0$ is that ART_a categories may learn their associations to nodes in the Map Field at different rates. The weights from an F_2^a node in ART_a to the correct node in the Map Field will always have a value of 1.0, however, since any error is corrected by forming a new category. Weights to the other nodes in the Map Field will be less than 1.0 (slow learning) or equal to 0.0 (fast learning). Recognition results on the test set are not hereby affected, since a winner-take-all field chooses the maximum activation in the Map Field as the recognition code via equation (43) in Appendix B.

Table 7 Recognition results on noisy data ($C = 1$) with slow learning to the Map Field $(\beta_{ab} = 0.2, \bar{\rho}_{max} = 0.95)$. Due to the low levels of noise surviving preprocessing, the recognition results here are not substantially different than those found using fast learning in noise in Table 5 except where noise was highest as in the 16 × 16 coarse coding. As noise increases, slow learning becomes more important for maintaining good recognition scores. [Reprinted with permission from Bradski and Grossberg, 1995.]

CORT-X 2 filter set	*Data presentation*	*Coarse code using spatial avg / Gaussian avg*		
		4 x 4	**8 x 8**	**16 x 16**
Small	Ordered	79.9/83.1	84.0/85.6	84.7/89.9
Small	Unordered	78.8/83.3	83.2/85.7	84.9/89.1
Large	Ordered	76.3/78.2	78.5/81.5	77.0/78.8
Large	Unordered	77.4/80.2	79.6/80.41	75.8/79.2

To derive benefit from slow learning, in the case $Á_{ab} = 1.0$, we set $\bar{\rho}_{max} = 0.95$. For $Á_{ab} = 1.0$, we may then compare the results of fast learning to the Map Field using $\bar{\rho}_{max} = 1.0$ with the results of slow learning to the Map Field using $\bar{\rho}_{max} = 0.95$. Table 7 records the results using slow learning in large amplitude noise ($C = 1$). Where noise levels after preprocessing were very small, the results were approximately the same as in the fast learning case shown in Table 5. Slow learning begins to help when the noise level increases, as with the 16 × 16 coarse coding. Table 8 records the average number of categories formed for the noisy data

case using fast learning and slow learning. Slow learning with $\bar{\rho}_{max} = 0.95$, caused approximately 10% fewer categories to be formed than with $\bar{\rho}_{max} = 1.0$, since noise-induced errors do not always cause the formation of a new category in the former case.

For a recognition system that can gather information from successive 2-D views, a key question is: given that an error occurs, how many successive errors will follow on average? For the airplane data set as processed by VIEWNET, it was found that the average overall length of an error sequence was 1.31 2-D views with a standard deviation of 0.57 views. On average then, when an error occurs, collecting two more views will usually be sufficient to correct the error. Thus, as in Seibert and Waxman's system, better 3-D object predictions may be derived by accumulating evidence from 2-D views. This is accomplished in VIEWNET in perhaps the simplest way by using a working memory whose unordered states are updated whenever a new 2-D view category is chosen. The working memory is realized as an integration field (F^{int}) between the Map Field (F^{ab}) that codes 3-D object, categories and the winner-take-all field (F^{wta}) in Figure 11. The equation for the integrator field is updated each time ART_a chooses a new 2-D view category:

$$(x_k^{int})^{new} = \beta_{int} x_k^{ab} + (1 - \beta_{int})(x_k^{int})^{old} \qquad (7)$$

where x_k^{int} is an integrator node for the k^{th} 3-D object, $^2_{int}$ is the integration rate each time the equation is stepped, and x_k^{ab} is the kth Map Field 3-D object category. The integration node with the largest activation represents the prediction of the 3-D object being viewed by the system. This maximum activation in the integration field is chosen by the winner-take-all field (F^{wta}) as the network's prediction of the 3-D object.

Table 8 Average number of nodes formed during training for the simulations of Tables 5 (noise with fast learning) and 7 (noise with slow learning). The format in the table is as follows: [spatial avg.]/[Gaussian avg.] = [fast learning, slow learning]/[fast learning, slow learning]. It can be seen that slow learning reduced the number of nodes formed by approximately 10%. [Reprinted with permission from Bradski and Grossberg, 1995.]

CORT-X 2 filter set	***Data presentation***	***Coarse code using spatial avg / Gaussian avg***		
		4 x 4	**8 x 8**	**16 x 16**
Small	Ordered	[184, 165]/[169, 150]	[73, 67]/[73, 66]	[33, 30]/[35, 32]
Small	Unordered	[198, 180]/[179, 163]	[77, 69]/[76, 70]	[35, 32]/[36, 33]
Large	Ordered	[179, 160]/[162, 147]	[68, 61]/[71, 66]	[33, 30]/[31, 29]
Large	Unordered	[192, 175]/[174, 160]	[75, 69]/[72, 67]	[32, 30]/[32, 30]

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

10. Voting or View Transitions for Evidence Accumulation Voting

Figure 12 shows a simulation of the integrator field where Equation (7) is stepped once each time ART_a chooses a category. Three integrator nodes are shown along the ordinate, one for each of the airplanes. Grey shading in the figure shows which 3-D object is "winning." In this simulation, the 3-D object being viewed is an F-16. The sequence of ART_a F_2^a categories that occurred is shown along the abscissa. The first two categories, 1 and 22, were in error since they were associated with the F-18 node in the Map Field. Categories 21 and 26 code correctly for the F-16 followed by category 48 which codes for the HK-1. Next, category 71 is selected. It is an uncommitted category that has never been activated before. By default, it gives equal activation to all integrator nodes. The remaining categories code correctly for the F-16.

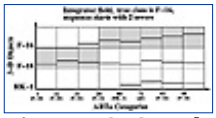


Figure 12 Simulation of the integration field. The ordinate axis contains the integrator nodes representing the 3-D objects: F-16, F-18, and HK-1. The abscissa represents ART_a F_2^a categories chosen by the preprocessed 2-D views being presented to VIEWNET. Black horizontal lines denote activity of the corresponding integrator node through time. Gray shading in the figure indicates the integrator node with maximal activation which represents the VIEWNET decision as to which object is being presented. Categories 1 and 22 erroneously code for the F-18 jet here. Category 48 erroneously codes for the HK-1. Category 71 has not been chosen before and so selects all objects simultaneously. The rest of the categories code correctly for the F-16. The integration step size was set to $^2_{int}$ = 0.2. [Reprinted with permission from Bradski and Grossberg, 1995.]

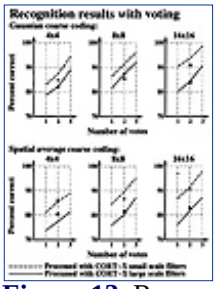


Figure 13 Recognition results for voting with an integration rate of $^{2}_{int}$ = 0.2. The graphs show the recognition results after gathering evidence over one, two, and three 2-D views for data preprocessed using large (solid line) and small (dotted line) scale CORT-X 2 filters. Results from both Gaussian and spatial averaging coarse-coding methods are shown where the images were reduced from 128 × 128 down to 4 × 4, 8 × 8, and 16 × 16 pixels. The circles and squares represent recognition scores resulting from using view transitions as discussed in Section 8. The black circles represent the recognition scores using view transitions for preprocessing with the large scale CORT-X 2 filters, the black squares represent recognition scores using view transitions for preprocessing with small scale CORT-X 2 filters. [Reprinted with permission from Bradski and Grossberg, 1995.]

Implementing evidence accumulation in this way is similar to voting for the 3-D object over a sequence of 2-D view inputs, but with recent views being given more weight. For 1 e $^{2}_{int}$ e 0, the closer $^{2}_{int}$ is to 1, the more weight is given to recent votes. To measure performance on the test set with voting, the integrator field was allowed to collect first two (for the two votes score), or three (for the three votes score) activations before VIEWNET's recognition decision was recorded. The integrator field was then cleared and two or three more activations were again collected before the next decision was made. This process was repeated until 1000 views had been seen in the test set for each object at which time the percent correct recognition score was computed. Figure 13 shows the average recognition scores for voting with ²int = 0.2 over one, two, and three views under CORT-X 2 preprocessing with large and small scale filter sets and coarse coding to 4 × 4, 8 × 8 and 16 × 16 pixels using both Gaussian and spatial averaging. Voting over three frames improves recognition results by an average of 10% with the best results being 98.5% correct for small scale filtered, 16 × 16 Gaussian coarse-coded data. The black dots and squares in the figure show recognition results from using 2-D view transition information. Note that view transitions did not improve performance; see Bradski and Grossberg (1995) for further details.

11. Summary

The cascade of processing stages in VIEWNET 1 converts spatially and temporally variant, noisy, and incomplete data into invariant 3-D object representations and predictions. The architecture thus illustrates that no one computational operation can, in and of itself, fully process noisy and fuzzy data. Qualitatively different, types of algorithms are appropriate to apply at prescribed stages of information processing. Within their own computational domains, however, these distinct types of processing may be realized by the most general processing of that type. Such algorithms may be called *general-purpose solutions of modal problems* in order to emphasize that distinct modalities of intelligence, such as vision, audition, and recognition, require different computational constraints for their solution, but that each modality seems to admit a general-purpose solution. In the VIEWNET architecture, simplified algorithms from theories that are working to develop general-purpose vision architectures, such as FACADE theory (Grossberg, 1987b; 1988; 1994), and recognition architectures, such as Adaptive Resonance Theory (Carpenter and Grossberg, 1991; 1993; 1994; Grossberg, 1987a; 1988; 1995), illustrate this theme. The CORT-X 2 filter is a simplified module inspired by FACADE theory and the Fuzzy ARTMAP algorithm is one module of the larger ART cognitive and neural theory.

Using the smaller scale of CORT-X 2 filters described above, a 3-D object recognition rate of approximately 90% was achieved from single 2-D views alone without recourse to more elaborate methods of generating aspect graph models of the 3-D objects. When evidence integration or voting over an unordered sequence of views was added, recognition rates reached 98.5% within three views. Voting over two views did as well as using view transitions on this database, but without the drawback of needing to learn $O(N^2)$ view transitions given N 2-D views. In addition, it was shown that the above recognition rates can be maintained even in high noise conditions using the preprocessing methods described herein.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

These high recognition rates are achieved by using a different preprocessor and supervised learning to create more optimal category boundaries than in the Seibert and Waxman studies. As reported in Seibert and Waxman (1992), their unsupervised clustering of coarse-coded maximal curvature data created general categories that unambiguously selected for the correct 3-D object only 25% of the time. In so doing, their network created 41 categories during training. In order to overcome the ambiguity of their general ART 2 categories, Seibert and Waxman used explicitly coded 2-D view category transitions to help identify the 3-D objects.

Using this approach, the network must be able to represent possible cross-correlations between every categorical 2-D view in the view transition matrices, one for each object, even if no correlations are eventually found between some of the categories. This is because a view transition matrix represents a definite network structure that explicitly codes the particular sequence of view transitions that ends up coding a prescribed 3-D object. Thus, such an algorithm is committed to represent all possible correlations between each of the 41 2-D view categories. The total number of correlations is then $(41^2 - 41)/2 = 820$, since transitions and their reverse are equivalent and there are no self-transitions, this is done for each object, then a total of $820 \times 3 = 2460$ correlation matrices would be needed. Add to this the 41 ART 2 categories and up to 2501 activations could be needed to recognize the three airplanes. As seen in Table 6, VIEWNET needs only 30-some odd nodes to categorize this database of 4000 examples since, by using unordered voting, it avoids the $O(N^2)$ penalty for using view transition information given N 2-D views.

Appendix A: CORT-X 2 Equations

The equations for the CORT-X 2 filter as described in Section 5. are discussed below. Figure 4a shows the model flow chart and Figure 4b shows the kernels used in the CORT-X 2 filter algorithm. Table 1 summarizes the parameters used in the simulations.

Step 1. Discounting the Illuminant

The 1-D cross-sections of the on- and off-center kernels are shown in Figure 4a.

ON-C and OFF-C Network

The activation x_{ij} at node v_{ij} at position (i, j) obeys the shunting on-center off-surround equation:

$$\frac{d}{dt}x_{ij} = -Ax_{ij} + (B - x_{ij})C_{ij} - (x_{ij} + D)E_{ij} \quad (8)$$

and $\bar{x}_{ij}$ obeys the off-center, on-surround equation:

$$\frac{d}{dt}\bar{x}_{ij} = -A(\bar{x}_{ij} - S) + (\overline{B} - \bar{x}_{ij})\overline{C}_{ij} - (\bar{x}_{ij} + \overline{D})\overline{E}_{ij} \quad (9)$$

where C_{ij}, $\overline{C}_{ij}$, E_{ij}, $\overline{E}_{ij}$ are discrete convolutions of the input with Gaussian kernels of the form:

$$K_{ij} = \sum_{p,q} I_{pq} K_{pqij} \quad (10)$$

with

$$K_{pqij} = \kappa \exp\left\{-\alpha^{-2} \log 2[(p-i)^2 + (q-j)^2]\right\} \quad (11)$$

The on-center kernel of $\bar{x}_{ij}$ is the off-surround kernel of x_{ij}, and the off-surround kernel of $\bar{x}_{ij}$ is the on-center kernel of x_{ij}. Then, $\overline{C}_{ij} = E_{ij}$, $\overline{E}_{ij} = C_{ij}$. Also, in Equations (8) and (9), $\overline{B} = D$ and $\overline{D} = B$. At equilibrium in the ON-C network,

$$x_{ij} = \frac{\sum_{(p,q)}(BC_{pqij} - DE_{pqij})I_{pq}}{A + \sum_{(p,q)}(C_{pqij} + E_{pqij})I_{pq}} \quad (12)$$

and in the OFF-C network,

$$\bar{x}_{ij} = \frac{AS + \sum_{(p,q)}(DE_{pqij} - BC_{pqij})I_{pq}}{A + \sum_{(p,q)}(C_{pqij} + E_{pqij})I_{pq}} \quad (13)$$

Step 2. CORT-X 2 Filter

Oriented receptive fields are elliptical as shown in Figure 4b so that

$$\frac{y^2}{a_s^2} + \frac{x^2}{b_s^2} = 1 \quad (14)$$

where a_s is the major axis and b_s is the minor axis with a_s e b_s. Two sizes of receptive fields were used, indexed by the subscript s with 1 = small and 2 = large scale. Orientations were chosen at angles spaced every À/8 degrees indexed below by the subscript k.

Simple Cells: The output of the pair of simple cells of scale s with activation variable $x = x_{ij}$ and receptive field orientation k is defined by

$$S_{sL}(i,j,k) = \max[L_s(x,k) - \alpha_s R_s(x,k) - \beta_s, 0] \quad (15)$$

and

$$S_{sR}(i,j,k) = \max[R_s(x,k) - \alpha_s L_s(x,k) - \beta_s, 0] \quad (16)$$

where L_s (x, k) and R_s (x, k) are the image inputs to the left- and right-oriented receptive fields:

$$L_s(x,k) = \frac{\sum_{(p,q)\in l_s(i,j,k)} x_{pq} w_{pq}}{\sum_{(p,q)\in l_s(i,j,k)} w_{pq}} \quad (17)$$

and

$$R_s(x,k) = \frac{\sum_{(p,q)\in r_s(i,j,k)} x_{pq} w_{pq}}{\sum_{(p,q)\in r_s(i,j,k)} w_{pq}} \qquad (18)$$

and w_{pq} is a weighting factor proportional to the area of a cell covered by the receptive field. L and R in S_{sL} and S_{sR} indicate that each receptive field is sensitive to the opposite direction-of-contrast from its companion.

The ON and OFF networks have separate sets of simple cells with the ON simple cells denoted by S^+_{sL} and S^+_{sR} and the OFF simple cells denoted by S^-_{sL} and S^-_{sR}.

Complex Cells: The complex cell output C_s (x, k) is defined by

$$C_s(i,j,k) = F[S^+_{sL}(i,j,k) + S^+_{sR}(i,j,k) + S^-_{sL}(i,j,k) + S^-_{sR}(i,j,k)] \qquad (19)$$

These cells are sensitive to the spatial scale s and amount-of-contrast x with orientation k, but are insensitive to direct ion-of-contrast.

Hypercomplex Cells (First Competitive Stage): The hypercomplex cells D_s (i, j, k) receive input from the spatial competition among the complex cells:

$$D_s(i,j,k) = \max\left[\frac{C_s(i,j,k)}{\epsilon + \mu \sum_m \sum_y C_s(p,q,m) G_s(p,q,i,j,k)} - \tau, 0\right] \qquad (20)$$

The circular oriented competition kernel G_s (p, q, i, j, k) shown in Figure 4c is normalized such that

$$\sum_{p,q} G_s(p,q,i,j,k) = 1 \qquad (21)$$

Partial cells at the kernel periphery are weighted in proportion to their area (taken to be one square unit). Grey areas in Figure 4c are inhibitory. Cells with centers within the one unit wide band through the middle of the kernel do not contribute to the inhibition. In our simulations, the small- and large-scale kernels were 2/3 the diameter of the small and large scale major axes of the ellipses shown in Figure 14b, respectively.

Hypercomplex Cells (Second Competitive Stage): Hypercomplex cells $D_s(i, j)$ compute the competition among oriented activities $D_s(i, j, k)$ at each position. This process is simplified as a winner-take-all process:

$$D_s(i,j) = D_2(i,j,K) = \max_k D_s(i,j,k) \qquad (22)$$

where K denotes the orientation of the maximally activated cell.

Multiple Scale Interaction: The interaction between the small and large scales is defined by

$$B_{12}(i,j) = D_1(i,j) \sum_{p,q} D_2(p,q) U(p,q,i,j) \qquad (23)$$

where the unoriented excitatory kernel U (p, q, i, j) is circular as in Figure 5b and is normalized so that

$$\sum_{p,q} U(p,q,i,j) = 1 \qquad (24)$$

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

In our simulations, $U(p, q, i, j)$ had a diameter 2/5 as large as the major axis of the large-scale elliptical filter. Cells covered by the kernel contribute to the excitation to the extent that their area is covered by the kernel. The smaller kernel $D_1(i, j)$ in (23) localizes boundary segments and suppresses noise near the boundary, while the larger kernel $D_2(p, q)$ suppresses noise far from the boundary.

Boundary Completion: The large detectors $D_2(i, j)$, are capable of responding at locations where pixel signal strength has been reduced by noise. Such boundary signals may, however, be poorly localized. To overcome this tradeoff between boundary completion and localization, large-scale cells interact cooperatively as:

$$B_2(i,j) = D_2(i,j)\max\left[\sum_{p,q} D_2(p,q,K)O(p,q,i,j,K) - \delta, 0\right] \qquad (25)$$

Kernel $O(y, x, k)$ is defined by the one-unit-wide white strips in Figure 4e. Cells with centers lying within the one unit wide band contribute to the cooperative process. The kernel is normalized so that:

$$\sum_{(p,q)\ in\ kernel} O(p,q,i,j,k) = 1 \qquad (26)$$

In the simulations, the length of the kernel is 3/5 as long as the major axis of the large-scale ellipse.

CORT-X 2 Output: The final output of the CORT-X 2 filter is the sum of the multiple scale interaction and the cooperative process:

$$B(i,j) = B_{12}(i,j) + B_2(i,j) \qquad (27)$$

Appendix B: Fuzzy ARTMAP Equations

In the following, all architectural references are to Figure 11. Three parameters determine Fuzzy ART dynamics: a choice parameter ± > 0; a learning rate parameter ²$_a$ [0, 1]; and a vigilance parameter Á$_a$ [0,

1].

Input preprocessing: Input **A** into a Fuzzy ART module is normalized by preprocessing the vector *A* as

$$\mathbf{A} = \frac{\mathbf{a}}{|\mathbf{a}|} \tag{28}$$

where the norm operator, | * | is defined as

$$|\mathbf{a}| \equiv \sum_{k=1}^{M} |a_k| \tag{29}$$

Inputs are then **complement coded** by setting $\mathbf{A} = (\boldsymbol{a}, \boldsymbol{a}^c)$, where

$$a_k^c \equiv 1 - a_k \tag{30}$$

Category choice: Category choice is determined by choosing the maximum choice function T_j^a:

$$T_J^a = max\{T_j^a : j = 1 \ldots N\} \tag{31}$$

where *N* is the number of nodes in F_2^a, and T_j^a is defined by

$$T_j^a(\mathbf{A}) = \frac{|\mathbf{A} \wedge \mathbf{w}_j^a|}{\alpha + |\mathbf{w}_j^a|} \tag{32}$$

where $\mathbf{w}_j^a$ is the template belonging to the j^{th} F_2^a node, and the operator $\wedge$ is defined as the fuzzy AND operation:

$$(p \wedge q)_k \equiv \min(p_k, q_k) \tag{33}$$

for *M*-dimensional vectors **p** and **q** (Zadeh, 1965). If more than one T_j^a is maximal, the category *j* with the smallest index is chosen. When category *J* is chosen, the node representing that category has activation $y_J^a = 1$. $y_j^a = 0$ for $j \neq J$. The F_1^a activity vector $\mathbf{x}^a$ obeys

$$\mathbf{x}^a = \begin{cases} \mathbf{A} & \text{if } F_2^a \text{ is inactive} \\ \mathbf{A} \wedge \mathbf{w}_J^a & \text{if the } J^{th} F_2^a \text{ node is chosen} \end{cases} \tag{34}$$

Match resonance and mismatch reset: A match and resonance are said to occur if

$$\frac{|\mathbf{A} \wedge \mathbf{w}_J^a|}{|\mathbf{A}|} \geq \rho_a \tag{35}$$

Otherwise, a mismatch reset occurs. When a reset occurs, the currently active node *J* is set to zero for the duration of the current input presentation and a new maximal node is found until the chosen node *J* satisfies Equation (35).

Map Field: The Map Field F^{ab} becomes active whenever one or both of ART_a or the supervised input *b* is active. Supervised inputs b_k are in 1-to-1 correspondence with nodes in the Map Field. If node *J* of F_2^a is active, then its weights $\mathbf{w}_J^{ab}$ activate F^{ab}. If a supervised input b_k becomes active, it directly activates F^{ab} node x_k^{ab} at an activation level of $b_k = 1$. The F^{ab} activity vector $\mathbf{x}^{ab}$ obeys

$$\mathbf{x}^{ab} = \begin{cases} \mathbf{b} \wedge \mathbf{w}_J^{ab} & \text{if the Jth } F_2^a \text{ node is active and } \mathbf{b} \text{ is active,} \\ \mathbf{w}_J^{ab} & \text{if the Jth } F_2^a \text{ node is active and } \mathbf{b} \text{ is inactive,} \\ \mathbf{b} & \text{if } F_2^a \text{ is inactive and } \mathbf{b} \text{ is active,} \\ 0 & \text{if } F_2^a \text{ and } \mathbf{b} \text{ are inactive.} \end{cases} \tag{36}$$

Match tracking: At the start of an input presentation to ART_a, the vigilance parameter is set to a baseline vigilance $\rho_a = \bar{\rho}$. Parameter $Á_{ab}$ is the Map Field vigilance parameter with $0 \leq Á_{ab} \leq 1$. If

$$|\mathbf{x}^{ab}| < \rho_{ab}|\mathbf{b}| \tag{37}$$

then subject to

$$\rho_a \leq \rho_{max} \tag{38}$$

match tracking causes $Á_a$ to be increased so that

$$\rho_a = \frac{|\mathbf{A} \wedge \mathbf{w}_J^a|}{|\mathbf{A}|} + \epsilon \tag{39}$$

where µ is a small positive constant. If match tracking causes Equation (38) to be violated, then match tracking is inhibited and learning takes place. If after match tracking, Equation (38) is still satisfied, then a new search cycle in ART_a leads to a different F_2^a node J with

$$|\mathbf{A} \wedge \mathbf{w}_J^a| \geq \rho_a|\mathbf{A}| \quad \text{and} \quad |\mathbf{b} \wedge \mathbf{w}_J^{ab}| \geq \rho_{ab}|\mathbf{b}| \tag{40}$$

or, if no such node exists, ART_a is shut down until it next gets an input.

Learning: Once search ends and a winning node y_J^a is chosen in ART_a, the Fuzzy ART weight vector $\mathbf{w}_J^a$ of the winning node is updated according to

$$(\mathbf{w}_J^a)^{new} = \beta_a(\mathbf{A} \wedge (\mathbf{w}_J^a)^{old}) + (1 - \beta_a)(\mathbf{w}_J^a)^{old} \tag{41}$$

where $\mathbf{w}_J^a(0) = 1$. The weight vectors $\mathbf{w}_j^a$, $j \neq J$ of non-winning nodes are not updated.

The weight vectors from the Fuzzy ART modules to the Map Field are updated according to

$$(w_{jk}^{ab})^{new} = \begin{cases} \beta_{ab} h[x_k^{ab}] + (1 - \beta_{ab})(w_{jk}^{ab})^{old} & \text{if } j = J \\ (w_{jk}^{ab})^{old} & \text{if } j \neq J \\ 1 & \text{initially} \end{cases} \tag{42}$$

where $h[x_k^{ab}] = 1$ if $x_k^{ab} > 0$, else $h[x_k^{ab}] = 0$. When $²_{ab} = 1$, Fuzzy ARTMAP is said be in the fast learning mode, when $0 \leq ²_{ab} < 1$, Fuzzy ARTMAP is in the slow learning mode.

Winner-take-all: A choice, or winner-take-all network F^{wta} sits above the Map Field F^{ab}. A winner-take-all field is needed above the map field when slow learning has been in effect from ART_a to the Map Field because it is then possible that a chosen F_2^a node may read out activation to more than one node in the Map Field during test mode.

The winner-take-all field is implemented algorithmically as

$$x_j^{wta} = \begin{cases} 1 & \text{if } j = \arg\max_k [x_k^{ab}], \\ 0 & \text{otherwise.} \end{cases} \tag{43}$$

Since nodes in F^{wta} are in one-to-one correspondence with nodes in the supervised field F^b, the winning node in F^{wta} a represents VIEWNET's choice of the 3-D object given the single 2-D view that the network has experienced. When evidence accumulation is used, a working memory, as in Section 10 interpolates the Map Field and the winner-take-all 3-D object recognition field.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

References

1 G. Bradski and S. Grossberg (1995), Fast learning VIEWNET architectures for recognizing 3-D objects from multiple 2-D views. *Neural Networks*, *8*, 1053-1080. Special issue on automatic target recognition.

2 G.A. Carpenter and S. Grossberg (1987), ART 2, Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26, 4919-4930.

3 G.A. Carpenter and S. Grossberg, (Eds.) (1991), *Pattern Recognition by Self-Organizing Neural Networks*. Cambridge, MA, MIT Press.

4 G.A. Carpenter and S. Grossberg (1993), Normal and amnesic learning, recognition, and memory by a neural model of cortico-hippocampal interactions. *Trends in Neurosciences*, 16, 131-137.

5 G.A. Carpenter and S. Grossberg (1994), Integrating symbolic and neural processing in a self-organizing architecture for pattern recognition and prediction. In H. Honavar and L. Uhr, (Eds.) *Artificial Intelligence and Neural Networks, Steps towards Principled Predictions*, pp. 387-421. San Diego, Academic Press.

6 G.A. Carpenter, S. Grossberg, and K. Iizuka (1992), Comparative performance measures of fuzzy ARTMAP, learned vector quantization, and back propagation for handwritten character recognition. In *Proceedings of the International Joint Conference on Neural Networks*, Volume I, pp. 794-799. Piscataway, NJ, IEEE Service Center.

7 G.A. Carpenter, S. Grossberg, N. Markuzon, J.H. Reynolds, and D.B. Rosen (1992), Fuzzy ARTMAP, A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3, 698-713.

8 G.A. Carpenter, S. Grossberg, and C. Mehanian (1989), Invariant recognition of cluttered scenes by a self-organizing ART architecture, CORT-X boundary segmentation. *Neural Networks*, 2, 169-181.

9 G.A. Carpenter, S. Grossberg, and J.H. Reynolds (1992), Fuzzy ARTMAP, slow learning and probability estimation. *IEEE Transactions on Neural Networks, 3*, 698-713. Reprinted in T. Yamakawa (Ed.), *Fuzzy Neural Systems* (volume 12 of KOUZA-FUZZY), Japan Society for Fuzzy Theory and Systems, 1993.

10 G.A. Carpenter, S. Grossberg, and D.B. Rosen (1991), Fuzzy ART, Fast stable learning and categorization of analog patterns by an adaptive resonance system. *Neural Networks*, 4, 493-504.

11 S. Grossberg (1987), *The Adaptive Brain*, Volumes 1 and 2. Amsterdam, Elsevier/North-Holland.

12 S. Grossberg (1987), Cortical dynamics of three-dimensional form, color, and brightness perception. I, Monocular theory. *Perception and Psychophysics*, 41, 87-116.

13 S. Grossberg (Ed.) (1988), *Neural Networks and Natural Intelligence*. Cambridge, MA, MIT Press.

14 S. Grossberg (1994), 3-D vision and figure-ground separation by visual cortex. *Perception and Psychophysics*, 55, 48-120.

15 S. Grossberg (1995), Are there universal principles of brain computation? *American Scientist*, 83, 79-80.

16 S. Grossberg and E. Mingolla (1985), Neural dynamics of perceptual grouping, Boundary completion, illusory figures, and neon color spreading. *Psychological Review*, 92, 173-211.

17 S. Grossberg, E. Mingolla, and D. Todorović (1989), A neural network architecture for preattentive vision. *IEEE Transactions on Biomedical Engineering*, 36, 79-102.

18 S. Grossberg, E. Mingolla, and J. Williamson (1995), Synthetic aperture radar processing by a multiple scale neural system for boundary and surface representation. *Neural Networks*, pp. 1005-1028.

19 S. Grossberg and D. Todorovic (1988), Neural dynamics of 1-D and 2-D brightness perception, A unified model of classical and recent phenomena. *Perception and Psychophysics*, 43, 241-277.

20 S. Grossberg and L. Wyse (1991), A neural network architecture for figure-ground separation of connected scenic figures. *Neural Networks*, 4, 723-742.

21 S. Grossberg and L. Wyse (1992), A neural network architecture for figure-ground separation of connected scenic figures. In R.B. Pinter and B. Nabet, (Eds.) *Nonlinear Vision, Determination of Neural Receptive Fields, Function, and Networks*, pp. 516-543. Boca Raton, FL, CRC Press, Inc.

22 N.K. Logothetis, J. Pauls, H.H. Buelthoff, and T. Poggio (1994), View-dependent object recognition by monkeys. *Current Biology*, 4, 401.

23 T. Poggio and F. Girosi (1990), Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247, 978-982.

24 E.L. Schwartz (1977), Spatial mapping in primate sensory projection, Analytic structure and relevance to perception. *Biological Cybernetics*, 25, 181-194.

25 M.C. Seibert and A.M. Waxman (1990a), Learning aspect graph representations of 3-D objects in a neural network. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN-90), Washington, D.C.*, volume 2, pp. 233-236.

26 M.C. Seibert and A.M. Waxman (1990b), Learning aspect graph representations from view sequences. In D.S. Touretzky (Ed.) *Advances in Neural Information Processing Systems 2*, pp. 258-265. San Mateo, CA, Morgan Kaufmann Publishing.

27 M.C. Seibert and A.M. Waxman (1991), Learning and recognizing 3-D objects from multiple views in a neural system. In H. Wechsler (Ed.) *Neural Networks for Perception*. New York, NY, Academic Press.

28 M.C. Seibert and A.M. Waxman (1992), Adaptive 3-D-object recognition from multiple views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11, 107-124.

29 L. Zadeh (1965), Fuzzy sets. *Information Control*, 8, 338-353.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Chapter 6
Industrial Applications of Hierarchical Neural Networks: Character Recognition and Fingerprint Classification

Ugur Halici
ugur-halici@metu.edu.tr
Department of Electrical Engineering
Middle East Technical University
06531, Ankara
Turkey

Ali Erol, Guclu Ongun
Halici Software Inc., Halici Group Companies,
METU Technopark, Middle East Technical University
06531, Ankara
Turkey

The hierarchical neural network (FINN) proposed in [7, 11] is a neural network structure that can be used efficiently in pattern recognition problems having a large number of classes, each containing a wide variety of samples. In this chapter, we present two industrial applications of this hierarchical neural network structure, namely, a document processing system called HALdoc and a fingerprint identification system called HALafis, for which prototypes are generated by Halici Software Inc. HNN is implemented for optical character recognition as a part of HALdoc and tested on computer fonts, character samples extracted from real documents, and also on handwritten digits. UHNN, which is the unsupervised version of HNN, is implemented for fingerprint classification as a part of HALafis [12, 22].

The hierarchical architecture of HNN is inspired by the pre-attentive and attentive levels related to attention concept in human cognition [9]. While the pre-attentive level is represented by self-organizing feature map (SOM) which clusters similar samples and makes a rough decision about the outcome; the attentive level is

formed by dedicating to each cluster a recognition module that has the expertise on the discriminating features within the cluster. Each recognition module consists of its own feature extraction module and classifier. Feature extraction is performed using principal component analysis (PCA) implemented by principal component neural networks. Extracted features are classified by multilayer perceptron (MLP). The system has a voting mechanism applied to a selected set of recognition modules. In UHNN the MLP layer is replaced by another SOM layer and voting mechanism is removed.

1. Introduction

The first stage of a simple pattern recognition system is feature extraction, which constitutes a major part of the system. The feature extraction process can be viewed as a transformation from the pattern space to a feature space. The features extracted should give a reliable representation, free from redundant information, of the raw input pattern. From a more formal point of view, the reliability of the representation corresponds to minimization of an error criterion in a reconstruction process. Eliminating redundant information mostly implies a feature space of lower dimension than the pattern space. Reduction in dimension is a must as it provides a way to reduce the number of free parameters of the classifier in the system, which leads to a better generalization capability.

PCA [4, 16] is a statistical feature extraction method that meets the requirements described above. It extracts optimal orthogonal features from an input process in the mean-squared error sense and is equivalent to the Karhunen-Loe’ve transform of a stationary stochastic vector process. As PCA yields a linear transformation, which only depends on the distribution of input patterns, it can be viewed as an unsupervised form of feature extraction. PCA can be implemented directly or by self-organizing neural networks [25] to form unsupervised neural preprocessing modules in pattern recognition problems [1, 21].

The second stage of pattern recognition system is a classifier. Classification is a mapping from feature space to the classes. ANNs (Artificial Neural Networks) provide a general framework with a highly parallel architecture for implementing mapping from several input variables to several output variables and are used as classifiers in many pattern recognition problems. In the statistical approach, classification is implemented by calculation of Bayesian *a posteriori* probabilities but it has been shown that when trained appropriately, some ANN models can directly calculate posterior probabilities [17, 18]. A desired property of a classifier is that its complexity, i.e., number of free parameters in the classifier, should not grow mainly with the dimensions of the input or the size of the training set. ANNs provide such classifiers and are similar to the mixture density estimation models in SPR (Statistical Pattern Recognition) [4].

MLPs [30] form the most common models used in classification problems as they can implement arbitrarily complex mappings. MLPs provide easy-to-use classifiers but have deficiencies regarding the learning speed and convergence to an optimum solution. The back-propagation training algorithm [30] needs a lot of passes over the training set and does not always guarantee the best possible performance. In practical applications, the algorithm is modified for minimizing learning time and maximizing robustness in terms of convergence to an optimum solution [8]. PCA extracts components that are statistically orthogonal to each other; when the components of the input vector of an MLP are decorrelated, training is faster and more robust [19].

Theoretically, a single MLP classifier can be used in arbitrary pattern recognition problems but practical implementations show that the classifier should be of huge size, which yields an infeasible training process. A preprocessing stage is still needed with ANNs. It is usually not easy to design a feature extraction module, and simple modules such as PCA do not suffice to reduce the complexity of the classification network.

An approach to cope with the complexity of the problem is to use a preclassification stage in the beginning of the system to create subsets of the classes and handle each subset by a different recognition module. A way of preclassification is clustering the input space. Clustering can be viewed as an unsupervised preclassification process in which the system itself decides about the elements of the subsets of classes using the distribution of patterns. SOM [15] is a well-known ANN model used for clustering. SOM has a close relationship with the K-Means Clustering algorithm but it also provides a topological ordering of model vectors corresponding to clusters. The model vectors, together with the topology in the output layer of the network, forms a snapshot of the input distribution on a lower dimensional space. Clustering yields convex regions corresponding to the clusters in the input space and higher correlation is expected between the components of the vectors belonging to a single cluster. When PCA is performed over each cluster, a better compression is expected.

A drawback of the clustering technique is that preclassification error is usually high, as one expects a lot of input patterns at the borders of clusters. A logical way, which has been implemented in HNN to cope with the preclassification error, is merging the neighboring clusters to obtain overlapping regions of clusters.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

The last point that affected the structure of HNN is related to the technique of using multiple classifiers for classification to decrease classification error [14]. The recognition modules of overlapping clusters can form a committee of classifiers whose outputs can be processed to make a better decision about the class of the input pattern. In HNN, we have implemented a voting mechanism among a set of activated classifiers.

In this chapter, we present the HNN and UHNN, which is a modified version of HNN, and their utilization in two industrial applications, namely, the document processing system HALdoc and the fingerprint identification system HALafis. HALdoc contains a character recognition system based on HNN, which is tested in both the classical multifont character recognition problem [7] and on handprint digit recognition. HALafis contains a classifier based on UHNN, which is used to classify fingerprints in an unsupervised manner prior to identification [12]. The organization of the rest of the chapter is as follows. In Section 2, a description of architecture of HNN and training algorithms is given. Section 3 describes the character recognition module of HALdoc and summarizes the performance of experimented networks. Section 4 describes the UHNN and its use for fingerprint classification in HALafis. Section 5 concludes the study.

2. The Hierarchical Neural Network

2.1 Network Architecture

The overall system has a hierarchical architecture based on preclassification of input data by a SOM network with *m*-by-*m*neuron output layer. Preclassification network is followed by an m-by-m array of recognition modules (see Figure 1) which are made of a PCA network cascaded with an MLP. Each recognition module is triggered by a unique unit on the output layer of the SOM network. A set of winners in the output layer of the SOM network trigger recognition modules to classify the input data. In the case of multiple winners, a voting mechanism is applied to decide about the pattern class.

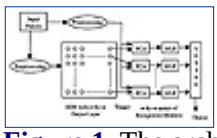


Figure 1 The architecture of HNN.

The flow of data through the network can be summarized as follows (see Figure 1): The input pattern is first preprocessed to obtain vectors that provide the input to the preclassifier and the recognition modules. The

purpose of the preprocessing stage is to get an intermediate representation of the input pattern that can be processed using SOM and PCA. The representation is supposed to be a normalized one, which can provide invariance to distortions. It is possible to handle other forms of feature extraction when PCA does not suffice and it may be better to use different preprocessing modules for the preclassifier and the recognition modules. Having been fed with the preprocessor output, the SOM preclassifier chooses a set of first *K* winners. Each winner triggers the corresponding recognition module and the output of preprocessor corresponding to the recognition modules is fed to the activated networks in which it is compressed to principal components by the PCA and classified by the MLP classifier. The output of a recognition module is the class label for the input and a confidence factor. Each unit on the output layer of the MLP corresponds to a label and the output value of the unit is the confidence factor for the label. The label with maximum confidence and the corresponding confidence factor forms the output of the recognition module. The label assigned by the majority of the recognition modules triggered by the first *K* winners of SOM forms the output of the system. When there is no majority, the label with the maximum number of votes and maximum confidence is chosen as the final output.

2.2 Training Algorithms

The training of HNN starts with training of the SOM network, at the end of which the training data is clustered and each cluster corresponds to a unit in the SOM output layer. Then, for each unit, its corresponding cluster and the clusters corresponding to its neighboring units are merged to obtain a larger cluster for the unit. The neighboring units can be chosen to be the four nearest neighbors or the ones in the 3-by-3 neighborhood. Choosing large neighborhood decreases preclassification error and supports the voting mechanism but increases training overhead. Then, for each cluster obtained, a PCA network is trained; this is followed by the training of a corresponding MLP network. In implementation, standard algorithms reported in the literature are used for training the network modules. The SOM Network is trained using the algorithm given in [27]. PCA is implemented by the self-organizing network and learning algorithm given in [25]. For MLP, the algorithm given in [26] has been used. These algorithms are explained in detail in the following subsections.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

2.2.1 Self-Organizing Feature Maps

SOM is a special neural network that accepts n-dimensional input vectors and maps them to a lower (usually two) dimensional output plane. The topology for a typical SOM network is shown in Figure 2. It has n input nodes and *m*-by-*m* output nodes. Each output node *j* in the SOM network has a connection from each input node *i* and w_{ij} denotes the connection weight between them.

There are two phases of operation in SOM: the training phase and the classification phase. In the training phase, the network finds an output node such that the Euclidean distance between the current input vector and the weight set connecting the input units to this output unit is minimum. This node is called the winner and its weights and the weights of the neighboring output units of the winner are updated so that the new weight set is closer to the current input vector. The effect of update for each unit is proportional to a neighborhood function, which depends on the unit's distance to the winner unit. This procedure is applied repeatedly for all input vectors until weights are stabilized. The choice of the neighborhood function, the learning rate, and the termination criteria are all problem dependent. The classification phase is simple once the training phase is completed successfully. In this phase, after applying the input vector, only the winner unit is determined.

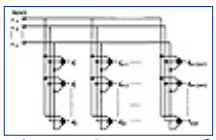


Figure 2 Network topology of the SOM. (From [12], Copyright © IEEE.)

The training steps of SOM are as follows:

1. Assign small random values to weights $\mathbf{w}_j = [w_{1j}, w_{2j}, \ldots w_{nj}]$;
2. Choose a vector **x** from the training set and apply it as input;
3. Find the winning output node d_{win} by the following criterion:

$$d_{win} = \min_j \{\|\mathbf{x} - \mathbf{w}_j\|\} \qquad (1)$$

where $\|\cdot\|$ denotes the Euclidean norm and $\mathbf{w}_j$ is the weight vector connecting input nodes to the output

node *j*;

4. Adjust the weight vectors according to the following update formula:

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)\left[x_i(t) - w_{ij}(t)\right]N(j,t) \quad (2)$$

where w_{ij} is the i^{th} component of the weight vector $\mathbf{w}_j$, $\cdot(t)$ is the learning rate and $N(j,t)$ is the neighborhood function;

5. Repeat Steps 2 through 4 until no significant changes occur in the weights.

The learning rate $\cdot(t)$ is a decaying function of time; it is kept large at the beginning of the training and decreased gradually as learning proceeds. The neighborhood function $N(j,t)$ is a window centered on the winning unit d_{win} found in Step 3, whose radius decreases with time. Neighborhood function determines the degree; an output neuron *j* participates in training. This function is chosen such that the magnitude of weight change decays with increase in distance of the neuron to the winner. This distance is calculated using the topology defined on the output layer of the network. Neighborhood function is usually chosen as rectangular, 2-dimensional Gaussian or Mexican hat windows. In the implementations presented in this chapter, a Gaussian window is used.

2.2.2 Principal Component Analysis Network

The Principal Component Analysis (PCA) is a statistical method that performs a linear mapping to extract optimal features from an input distribution in the mean-squared error sense. The method originated from the Karhunen-Loe've (K-L) transform which was originally developed for continuous time signals. PCA can be implemented by self-organizing neural networks to form unsupervised neural preprocessing modules for classification problems [1, 24].

An important virtue of PCA is that the extracted components are statistically orthogonal to each other. Orthogonality of components of the input vector of a MLP network results in speed-up in training and robust convergence [19].

The K-L transform, which is used in PCA, finds the representation of the input vectors in terms of the eigenvectors of their covariance matrix. It has an excellent energy compaction property; therefore, it is frequently used in statistical pattern recognition. The method is as follows [4, 12, 16].

Given an ensemble of *M* real-valued vectors, $\mathbf{x}^k \in \Re^n$, 1 d*k* d*M*, their covariance matrix $\mathbf{R}_x$, is calculated as

$$\mathbf{R}_x = \frac{1}{M}\sum_{k=1}^{M}(\mathbf{x}^k - \hat{\mathbf{x}})(\mathbf{x}^k - \hat{\mathbf{x}})^T \quad (3)$$

where

$$\hat{\mathbf{x}} = \frac{1}{M}\sum_{k=1}^{M}\mathbf{x}^k \quad (4)$$

The unit length eigenvectors of $\mathbf{R}_x$ are the orthogonal basis for K-L transform and are obtained by solving the following equation:

$$\mathbf{R}_x\Psi = \Psi\Lambda \quad (5)$$

where › is a diagonal matrix having the eigenvalues of $\mathbf{R}_x$ and ¨ is the modal matrix having eigenvectors of $\mathbf{R}_x$ for its columns ordered in decreasing eigenvalues. After determining ¨, the K-L transform of any vector can be found as follows:

$$\nu = \Psi^T\mathbf{x} \quad (6)$$

Reducing ¨ to $¨^m$ and eliminating the last (*n*-*m*) eigenvectors results in an *m*-dimensional subspace spanned by the remaining *m* eigenvectors in $¨^m$. The subspace spanned by these eigenvectors is called the *principal subspace*. The components of the projection of a vector in the principal subspace are called the *principal components*. It results in dimensionality reduction if $¨^m$ is used instead of ¨ in Equation (6). If the m^{th} eigenvalue is considerably small when compared to the first eigenvalue, the vector transformed to the

principal subspace carries approximately the same information as the original vector although the dimensionality is reduced.

PCA can be implemented by the neural network shown in Figure 3. The network has two layers consisting of linear units. There exists a hierarchical lateral connection scheme in the output layer. Each output unit *k* receives input from each input unit and the output units $j < k$.

For a network with n input units and m output units, the activation of an output unit k is given by

$$y_k = \mathbf{w}_k^T \mathbf{x} + \sum_{j<k} u_{j,k} (\mathbf{w}_j^T \mathbf{x}) \qquad (7)$$

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

where y_k denotes the activation of the output unit k, $\mathbf{x} = [x_1, x_2, ..., x_n]^T$ denotes the input vector, $\mathbf{w}_k = [w_{1,k}, w_{2,k}, ..., w_{n,k}]^T$ denotes the weight vector of the output unit k coming from the input layer, and $u_{j,k}$ denotes the weight of connection from the output unit i to the output unit k. The input process is assumed to be a zero mean, wide-sense stationary process.

The weights between the layers are updated by

$$\Delta \mathbf{w}_k = \eta \left(\mathbf{R}_x \mathbf{w}_k + \sum_{j<k} u_{j,k} \mathbf{R}_x \mathbf{w}_j \right) \tag{8}$$

and normalized to have a magnitude of unit length. The lateral weights are updated by

$$\Delta u_{\ell,k} = -\mu \, E[y_\ell \, y_k] \tag{9}$$

¼ and · in (8) and (9) are learning parameters, and they should be kept small enough in order not to violate the assumption of small learning steps. Training ends when lateral weights converge to zero; by that time weights between the layers converge to ¨^{m}, that is, the first m eigenvectors of $\mathbf{R}_x$.

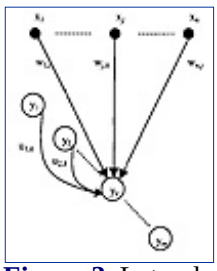


Figure 3 Lateral network for extracting m principal components.

2.2.3 Back-propagation Algorithm

Back-propagation algorithm [30] is a learning algorithm applied on multilayer feed-forward networks consisting of nonlinear units. An L layer network is given in Figure 4.

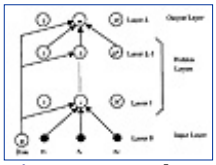


Figure 4 Back-propagation network.

The activation for a unit j in layer ℓ denoted by a_j^ℓ is calculated as

$$a_j^\ell = \sum_i w_{i,j}^\ell y_i^{\ell-1} \qquad \ell \geq 1 \tag{10}$$

where $w_{i,j}^\ell$ represents the weight of the connection from the i^{th} unit in the $(\ell-1)^{th}$ layer to the j^{th} unit in the layer ℓ. $\ell = 0$ corresponds to the input layer in which $y_i^0 = x_i$. Output of the units are determined by a shifted sigmoid function given as

$$y_j^\ell = f(a_j^\ell) = \frac{1}{1+e^{-a_j^\ell}} - 0.5 \qquad \ell \geq 1 \tag{11}$$

The training session consists of presentation of randomly selected samples in the training set and update of weights using the desired output value for the presented sample. An optimum set of weights is obtained after a number of passes over the training set. Weight update equations can be summarized as follows:

$$\Delta w_{j,k}^\ell = \varepsilon \delta_k^\ell y_j^{\ell-1} \qquad \ell \geq 1 \tag{12}$$

where $0 < \mu < 1$ is the learning rate. δ_k^ℓ is called the error derivative and given by

$$\delta_k^\ell = -f'(a_k^\ell) e_k^\ell \qquad \ell \geq 1 \tag{13}$$

where e_k^ℓ denotes the error for the k^{th} neuron at the ℓ^{th} layer. For output layer, error is calculated using

$$e_k^\ell = d_k - y_k^\ell \quad \ell = L \tag{14}$$

where L is the index of output layer and d_k is the k^{th} component of the desired output vector. For hidden layers, we have

$$e_k^\ell = \sum_k \delta_k^{\ell+1} w_{j,k}^{\ell+1} \qquad 1 \leq \ell < L \tag{15}$$

In practical applications, the algorithm needs many passes over the training set and the algorithm is complemented with some techniques to minimize the number of training cycles. When a weight in the network approaches zero, it cannot recover for a long time because the learning steps also approaches zero. This is called the flat spot problem and it causes a decrease in learning speed. In this study, a technique given in [8] is implemented to cope with the flat spot problem.

3. Character Recognition Module of HALdoc

HALdoc is a general-purpose document processing and archiving system. The documents archived in the system are not restricted only to text files. It can also archive images, sounds etc., in addition to text files. The system includes a built-in OCR program and a Scan Control Subsystem to get images of new documents [2, 3, 5, 7, 10, 11].

The block diagram of the text reading system developed for HALdoc is shown in Figure 5. In this system, a document image is taken in PCX format. It is converted into bitmap and binarized using a gray level histogram. Then, a document analysis is performed to separate text areas from graphics and photograph areas. At the same time, the lines in the text areas are found. Then, character segmentation is done with contour following in each text line to extract character images. The characters found are normalized with a scaling invariancy transform; then they are fed to the character recognition module of the system.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

3.1 Multifont Character Recognition

The character recognition module is the HNN given in Figure 1. In the implementation of the recognition module, the preprocessing stages only perform size normalization by scaling the input image to a constant size grid without preserving the aspect ratio. For the recognition modules, the image is scaled to a 10-by-7 16 gray level grid; for the preclassifier, it is scaled to 10-by-7 binary grid.

For a character recognition module, construction of the training set is an important problem. The training set should represent variations of character shapes and distortions. It is best to a use considerably large number of samples from various document images [28]. In this study, a database is constructed using high-resolution computer fonts. For some of the samples, scanner outputs for each character are simulated by adding noise to the screen image of these computer fonts. Added noise consists of clearing or enlarging the edge points on the image with a probability proportional to the slope of the curve that the edge point resides on and producing some blobs on the main body of the character. This simulator is built intuitively after examining scanned images of some documents.

For evaluating the generalization capability of the classification module, two types of test sets have been used. The first type of test sets has been implemented using computer fonts as in the case of the training set. The second type of training set. represents the real environment and has been prepared using randomly selected samples from scanner images of real documents like magazines and books. The documents are processed using the OCR system but images resulting from incorrect segmentation are eliminated during the sampling process.

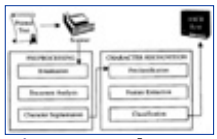


Figure 5 The automatic text recognition system in HALdoc.

A character set consisting of letters in the Turkish and English alphabet, numbers, and most common symbols is chosen for the classifier. Not all of these symbols correspond to different classes. In some of the fonts, a single character image can correspond to multiple characters, e.g., l and I have the same image in all of the sans-serif fonts. Furthermore, as a result of the preprocessing stage, the classifier does not have the ability to differentiate some of the lower case and upper case letters, e.g., the couples (Cc), (Ss). After eliminating such

characters, a set consisting of 73 different classes that corresponds to 90 characters is constructed as shown in Figure 6. The characters that are not separated by a comma are elements of the same class.

Figure 6 Character set of the recognition module.

The character set does not include punctuation marks as the segmentation algorithm is designed to detect them using the structural information available in the segmentation process. The structural information is also used to differentiate upper case and lower case letters for pairs (Cc), (Ss), (Pp), (Oo), etc. in the contextual processing stage of the OCR system that the recognition module resides in.

The training set is built from 8, 10, 12 and 16 point size samples without noise, and 10 and 16 point size samples with noise, from 36 different styles including italic styles of 14 font families. The physical size of characters is calculated for 300-dpi resolution. The training set has 16,152 characters. The system is tested on both computer fonts and samples collected from real documents. The test set of computer fonts is built with 14, 11 and 9 point size samples without noise, and 11 point size samples with noise, from eight font styles of five font families. The test set does not include any font family used in the training set. The test set of real documents is built from randomly chosen samples from 23 different Turkish and English documents scanned at 300 or 200 dpi resolution and contains 6283 samples (see Figure 7). The samples from real documents are extracted using the OCR system that the character recognition module is designed for; but in the sampling phase, erroneous samples resulting from incorrect segmentation are eliminated. As all the test sets prepared for the system are free from segmentation error, the recognition rates reported in this chapter are a measure of the performance of the hierarchical network proposed in Section 2.

The parameters of the experiments concentrated on the compression ratio for PCA modules and the size of the output layer of the SOM. After some experiments, the size of the output layer of SOM is constrained to 6-by-6 or 9-by-9. The size of hidden layers of MLPs in each module is chosen to be 70 units. The PCA networks compressed data to 30 or 15 principal components. It is possible to optimize the size of each PCA and MLP couple but that is not desirable, as it is time consuming.

Some of the networks, together with their recognition rates on the training set, are listed with the parameters of the networks in Table 1. The first column of the table lists the ID of the network. The second column lists the size of the output layer of the SOM networks. The third column refers to the size of the neighborhood used in merging the training sets for the branch networks. The fourth column refers to the number of principal components extracted from the input grid for each classification module. The fifth column gives the number of gray level colors of the image that provides the input to the branch networks. The sixth column gives the recognition rate of the whole network on the training set. The training set recognition rate is measured without using the voting mechanism.

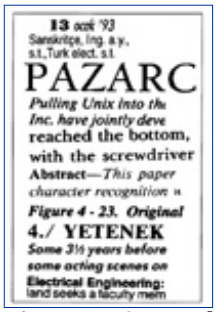

13 ocak '93
Sanskritçe, Ing. a.y.,
s.t.,Turk elect. s.t.
PAZARC
Pulling Unix into the
Inc. have jointly deve
reached the bottom,
with the screwdriver
Abstract—This paper
character recognition s
Figure 4 - 23. Original
4./ YETENEK
Some 3½ years before
some acting scenes on
Electrical Engineering:
land seeks a faculty mem

Figure 7 A sample document used in testing the OCR system.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

High performance on the training set shows that the system can learn easily. To measure the generalization ability of the system, the networks has been tested on the samples collected from real documents.

Table 1 The networks used in the experiments and their recognition rate on the training set.

ID	Size SOM	Merged Neighbors	No. of PCs	Gray Levels	Training Set Rec. Rate %
1	6-by-6	0	30	16	99.26
2	6-by-6	4	30	16	99.54
3	6-by-6	8	30	16	99.80
4	6-by-6	8	30	256	99.73
5	9-by-9	0	15	16	98.71
6	9-by-9	8	15	16	99.85
7	9-by-9	8	15	256	99.75

Table 2 lists the results. Columns 2, 3, 4 and 5 list the recognition rate on the test set constructed from real documents. The parameter K refers to the number of recognition modules that contribute to voting and $K = 1$ means that voting mechanism is not activated.

Table 2 Recognition rate on the test set of real document samples.

	TEST SET RECOGNITION RATE (%)			
ID	**K = 1**	**K = 3**	**K = 5**	**K = 7**
1	90.56	91.71	92.03	90.83
2	93.92	94.89	94.99	95.23
3	95.30	95.89	96.12	96.32
4	94.97	95.70	95.78	96.04
5	89.61	88.37	-	-
6	93.25	95.11	95.00	95.32
7	94.22	95.56	95.75	95.83

It is observed that the voting mechanism increases the performance, but using an excessive number of winners does not always guarantee an increase in recognition rate. Comparison of the performance measures of Networks 1, 2, and 3 shows that assigning MLPs to multiple clusters has an important effect on the voting mechanism. As the number of neighboring clusters that are merged for a single recognition module increases, the winners are guaranteed to know something about the pattern they vote for. Increasing the number of voters causes novice recognition modules to contribute to the voting, which results in a decrease or a negligible increase in the recognition rate. For Network 3, the voting mechanism causes a 1% increase in recognition accuracy but using more winners slows down the speed of recognition. One should choose between speed and accuracy.

An important source of error for a machine-printed OCR system is due to the confusion between the similar characters. The error matrix shows a high failure rate for some of the input classes. For example, the pairs (c,e), (ö,6), (5,s) are among these classes. One reason for this failure is that the resolution of the input grid is very low. Adding more gray levels or using a larger grid can provide a solution to the problem but the recognition speed should be sacrificed. A possible solution is using a postprocessing module containing specialized classifiers that can differentiate the most remarkable couples.

The proposed hierarchical neural network has a very high recognition rate of 99.80% on the training set prepared using computer fonts, which shows that it learns very easily. The generalization capability of the system is measured both on character samples collected from scanned images of real documents and on computer fonts that are not used in the training set. Assuming no segmentation error, the recognition rate in a real environment is between 95.32% and 96.32%, depending on the voting policy.

It is not possible to compare different recognition systems using only the test set performance as they are trained and tested in different environments. It is best to consider test set and training set recognition rates, together with the nature and size of the training and test sets. In [6], a test set recognition rate of 97.2% and a training set recognition rate of 99.5% are reported for a neural network classifier trained and tested in an environment of scanner images of 12 different documents. In this study, a training set of 7102 samples from five different documents, and a test set of 8730 samples from seven different documents are reported. In [26] and [27], test set recognition rates of 97.5% and 96.7% are reported, respectively, where a training set of 110,000 samples from 200 different fonts and 50 documents and a test set of 10,500 samples from five documents is used. The training set recognition rate is given as around 99% in both of the studies. A more recent study [29] gives performance measures for a neural character classification system trained and tested using a NIST database of isolated alphanumeric characters. The recognition rate on the test set is given as 89.40% for lower case characters and 96.44% for upper case characters. All of these studies report recognition rates for neural character classification systems and the data shows that the system implemented in this study is comparable with existing studies.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

The recognition rates given in Table 2 indicate that Network 3 has the highest performance measures on real documents and has a good performance even without voting. This network has been further tested on untrained computer font samples that were explained previously. The results of these tests are given in Table 3.

Table 3 Recognition rate of the Network 3 on test set of computer fonts.

Number of Winners	Recogition Rate (%)
$K = 1$	97.50
$K = 3$	98.17
$K = 5$	98.13
$K = 7$	98.13

These tests on the test set of computer fonts show that the training set used cannot simulate the real environment perfectly. The noise level in poor-quality images is much higher than the noise added to the perfect images of characters and an inevitable kind of distortion, skew distortion, is not simulated at all. Increasing the number of fonts used in training the network and using samples from real documents can provide a better system in the real environment. However, the results in Table 2 gives information about the general behavior of the hierarchical network proposed. Although the test set of computer fonts represents an environment closer to the training environment, the problem is still a complex one as there are 75 different classes, each having a wide variety of samples from 44 different font styles. Reaching a recognition rate of 98.17% for such a complex set is quite important since we observed in our experiments that it was not even possible to make the training session converge when a single MLP classifier was used.

In an experiment with a single module of PCA network and MLP classifier, we trained this module using computer fonts and tested on real document samples. The maximum possible recognition rate was 99.58% on a training set of computer fonts and 88.96% on the test set of real document samples. These test set results are much lower than the results obtained for the hierarchical architecture for which the recognition rate is 96.32% with real document samples and 99.80% with the training set. These results strongly support that the preclassification technique drastically enhances the performance.

3.2 Handwritten Digit Recognition

In the experiments on handwritten digit recognition, a database in a freely distributed CD-ROM of "NIST Form-Based Handprint Recognition System" has been used. The system of NIST (The National Institute of Standards and Technology, USA) provides a part of the NIST Special database 3 (SD3) which contains 313,389 segmented and labeled handwritten character images from 2100 different writers. The database that has been used includes 119,740 segmented and labeled images of handwritten digits from 1000 different writers. In the experiments, the samples available were divided into two disjoint groups for testing and training.

The preprocessing scheme given in "NIST Form-Based Handwritten Recognition System" has been used and it consists of the following stages:

1. Size Normalization: The image of the character is scaled to a 20-by-32 grid without preserving the aspect ratio and placed at the center of a 32-by-32 grid. The scaling algorithm (referred to as second-generation normalization) applies a simple morphological operator to normalize the stroke width within the character image. If the pixel content of an image is significantly high, then the image is eroded (i.e., strokes are thinned). If the pixel content of the image is significantly low, then the image is dilated (i.e., strokes are widened).

2. Slant Normalization: Another great variation observed in handwritten digit images is the slant of the characters. For that reason, the slants of the characters are normalized by a technique that uses horizontal shears in which rows in the image are shifted left or right to straighten the character in the image. Given a segmented character image on a grid, the top and bottom image rows of the grid containing black pixels are located. The left-most black pixel is located in each of the two rows and a shifting function is calculated to shift the rows of the image so that the left-most pixel in the top and bottom rows lines up in the same column. The rows between the top and bottom are shifted by lesser amounts based on the linear shifting function.

A slope factor f, defining the linear shifting function, is calculated as

$$f = \frac{t_l - b_l}{b_r - t_r} \tag{16}$$

where t_r is the vertical position of the top row, b_r is the vertical position of the bottom row, t_l is the horizontal position of the left-most black pixel in the top row, and b_l is the horizontal position of the left-most black pixel in the bottom row. The slope factor is used to compute a shift coefficient as follows:

$$s = (r - m)f \tag{17}$$

where r is the vertical row index in the image and m is the vertical middle of the image. This causes the shifting to be centered about the middle of the image. A positive value of the shift coefficient causes the row to be shifted s pixel positions to the right and a negative value of the shift coefficient causes the row to be shifted s pixel positions to the left.

A major problem with training is that the convergence rate of the PCA network is too slow. The PCA network simply calculates the eigenvectors of the autocorrelation matrix of the input image; so it is replaced by a standard FORTRAN linear algebra routine in EISPACK whose code is provided in the CD-ROM of the "NIST Form Based Handprint Recognition System." The speed gain is enormous, but still the training process is computationally expensive and it takes about 3 to 4 days to train a complete system of 36 subnetworks on a dedicated SPARC workstation.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

The training time of the network forced us to choose the size of the input layer of SOM as small as possible. Using the results of experiments on multifont printed characters, we have intuitively fixed it to 6-by-6 units and 3-by-3 neighborhood is merged in the formulation of training sets of branch networks. The experiments concentrated on the amount of compression handled by PCA and the size of hidden layer units in MLP subnetworks. Table 4 lists the parameters of the networks and the recognition rates on the training set. For the last network (Network 5), the training set size has been increased.

Table 4 Recognition rate on the training set of handwritten digits.

ID	No. of PCs	Hidden Layer Size	Training Set Size	Training Set Rec. Rate (%)
1	30	80	29051 (250 person)	97.43
2	64	140	29051 (250 person)	100
3	50	90	29051 (250 person)	98.80
4	64	90	29051 (250 person)	100
5	64	140	58646 (500 person)	100

Table 5 lists the recognition rates on the training and test sets with different voting parameters.

Table 5 Recognition rate on the test set of handwritten digits.

		TEST SET RECOGNITION RATE (%)			
ID	**Test Set Size**	**K = 1**	**K = 3**	**K = 5**	**K = 7**
1	90689 (750 person)	93.77	96.49	96.75	96.87
2	90689 (750 person)	96.76	96.99	97.14	97.19
3	90689 (750 person)	96.28	96.76	96.95	96.99
4	90689 (750 person)	96.40	96.83	96.98	97.01
5	161094 (500 person)	97.26	97.49	97.58	97.63

Possible sources of error are due to distortions caused by the size and slant normalization process and also due to badly segmented or badly written samples in the database. Some randomly selected examples of digits that

are misclassified are presented in Figure 8.

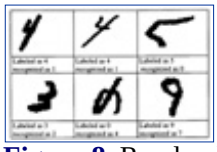


Figure 8 Randomly selected examples of misclassified digits.

4. Fingerprint Classifier in HALafis

With the increasing power of computers, automated fingerprint identification systems (AFIS) are developed to automate the tedious manual classification and matching processes of fingerprints [20]. The Automatic Fingerprint Identification System HALafis has the basic structure given in Figure 9 [12, 22-24]. As shown in the figure, there are two different routes of operation: classification and matching.

4.1 Feature Extraction

One of the most important parts of a classification problem is the generation of a feature space for the input samples that are to be classified. Generally, in AFIS, a fingerprint image is scanned and digitized, then processed to generate a feature vector, and then fed to the classifier. When neural networks are used for classification purposes, the choice of input vectors plays a crucial role in the performance of the net. In the fingerprint classifier of HALafis, a *block directional image* is used as feature vector. The steps implemented in HALafis for feature extraction on a scanned fingerprint are shown in Figure 10 [12, 22, 23].

Processing starts with the segmentation of the fingerprint image so that noisy and corrupted parts that do not carry valid information are deleted. Then the block directional image and its corresponding certainty values are generated. The generated vector is then fed to the classifier and the class is selected. This class will be used as a search space in the matching phase. The feature extraction steps are explained in detail in [12].

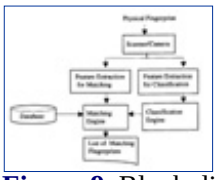


Figure 9 Block diagram of HALafis. (From [12], Copyright © IEEE.)

There are 65,536 pixels, each being assigned a direction, on a 256-by-256 fingerprint image. In order to obtain a reasonable size input vectors, it is preferable to divide the fingerprint image of size 256-by-256 into 16-by-16 pixel grids and take the average of directions for each grid. The resultant vector obtained using average directions on the 16-by-16 grid is called the *block directional image*. Applying the algorithm on a test image yields the block directional image as shown in Figure 11. In addition to reducing the size of the feature vector, averaging has a smoothing effect that increases the validity of directions. Here, the average is not computed by summing up all the directions in a grid and dividing the result by 16-by-16. Instead, directions are considered as vectors of length 1; the vectors are added together and the magnitude is divided by 16-by-16. The resultant directional vector is assigned to the grid. Thus, the grids that do not carry valid direction information are assigned shorter directional vectors, since vectors having different directions cancel each other. These vector magnitudes will constitute the *certainties* of the grids in the classification process, which is based on the SOM algorithm that we modified as depicted in Section 4.2.1.

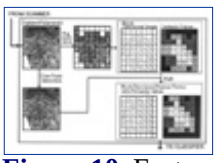


Figure 10 Feature extraction for classification. (From [12], Copyright © IEEE.)

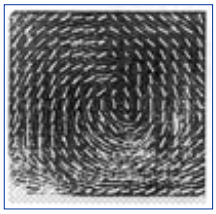


Figure 11 The block directional image overlaid on a sample fingerprint. (From [12], Copyright a IEEE.)

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

4.2 Classification of Fingerprints

The classification of fingerprints in this study is performed by UHNN. However, instead of the original SOM algorithms for training/classification we used the algorithms that we modified by introducing the parameter called *certainty*.

4.2.1 Modified SOM Algorithm

In order to increase the accuracy of fingerprint classification, we have introduced a parameter called certainty for each input node and have modified the SOM algorithm presented in Section 2.2.1 accordingly. The network topology of the modified SOM (MSOM) is shown in Figure 12. The network connections are the same as in the original topology of SOM except for the introduction of the certainty parameter ci on the connections.

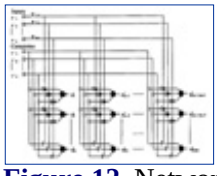


Figure 12 Network topology of MSOM. (From [12], Copyright © IEEE.)

When the network is to be trained, it chooses the output node with the smallest modified distance, which takes certainty values into consideration, between the weights from the input nodes and the current input vector.

The training algorithm of MSOM is as follows:

1. Assign small random values to weights w_{ij}.
2. Choose a vector **x** from the sample space and preprocess it to form $\mathbf{x}^c$, considering certainty vector **c** such that

$$x_i^c = c_i x_i + (1 - c_i)\, x_i^{avg} \qquad (18)$$

where subscript *i* refers to the i[th] component of the corresponding vector, x_i^{avg} is the average value of x_i computed over the sample space and then apply $\mathbf{x}^c$ as input to the network.

3. Find the winning output node d_{win} by the following criterion:

$$d_{win} = \min_j \{ \| \mathbf{x}^c - \mathbf{w}_j \| \} \tag{19}$$

where $\| \cdot \|$ denotes the Euclidean norm and $\mathbf{w}_j$ is the weight vector connecting input nodes to the output node j.

4. Adjust the weight vectors according to the following update formula:

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t) N(j,t)(x_i^c(t) - w_{ij}(t)) c_i \tag{20}$$

where w_{ij} is the i^{th} component of the weight vector $\mathbf{w}_j$, $\cdot(t)$ is the learning rate and $N(j,t)$ is the neighborhood function.

5. Repeat Steps 2 through 4 until no significant changes occur in weights.

One of the differences between this algorithm and the original SOM is that MSOM uses preprocessed input vectors instead of the input vector itself. The input vector is preprocessed using the certainty values, which are between 0 and 1. The basic idea behind using such a means of preprocessing is to stimulate the components of the input vector **x** of which we are certain and inhibit the less certain ones in the SOM competition process. The preprocessing is designed such that if the certainty is high for the i^{th} component, i.e., $c_i = 1$, then $x_i^c = x_i$. Therefore, the network behaves as the original SOM if all the certainty values are high. However, if the certainty is low, i.e., $c_i = 0$, then $x_i^c = x_i^{avg}$, where x_i^{avg} is the average value of the x_i computed over the sample space; therefore it has a minimal contribution to the decision of the winner. For the intermediate values of c_i, the value of x_i^c is determined by interpolation.

The other difference is in the weight update rule, in which certainty values are taken into consideration. If we are certain about the value x_i, i.e. $c_i = 1$, the weight update rule reduces to the one used in the original SOM. However, if we are uncertain about x_i, that is, $c_i = 0$, then the weight w_{ij} connecting the input x_i to the node j is not updated. For the intermediate values of c_i, the amount of weight change is in between these two cases.

After training is completed, classification is a matter of applying Step 3 for the vector to be classified. The winning output node determines the class of the applied input vector. In this way, all fingerprints in the database are assigned a class number. Given an unseen rolling of a fingerprint, if it is desired to search the original rolling of the same fingerprint in a database, then the new rolling is applied as input to the SOM network and the search is conducted beginning first with the class related to the winner of the network. However, if the fingerprint is not found within this class, the class related to the second winner of the SOM is considered and so on for the other winners in the order of activation.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

4.2.2 Fingerprint Classification Based on UHNN

When a single layer of SOM network is used for classification of fingerprints, the classes represented by some of the neurons may become more crowded with fingerprints as compared to other classes, due to the self-organizing characteristics of the SOM network. Increasing the number of classes by enlarging the SOM network solves the problem to some degree; but after a certain network size, the number of non-empty classes remains the same even though the network size is increased since some classes become empty while others remain too crowded. Instead of enlarging the SOM size within the layer, the use of UHNN (two layers SOM-PCA) helps to divide those crowded classes (see Figure 13). In Section 3, we explained the use of the HNN network for character recognition where SOM is used for preclassification, PCA for feature extraction, and then MLP with back-propagation algorithm for classification. However, it is not convenient to use MLP in fingerprint classification since the classes are not known when they are not determined by Henry classification [12].

The network is trained as follows: First of all, the reduced modal matrix (Eqn. 5) made of the principal components for the block directional feature vectors is generated by using all the fingerprints in the training set. The PCA shown in the first layer uses these principal components. Then the feature vectors transformed through this PCA are fed as input to the SOM network of the first layer and the SOM is trained. Another layer of PCA-SOM network is attached for each of crowded classes. Each additional PCA-SOM network in the second layer is trained using the feature vectors extracted by the first layer PCA but only by those samples in the class represented by the related SOM neuron in the previous layer. Therefore, each PCA in the second layer has different characteristics. Each SOM network in the second layer is trained by using the features extracted by the related PCA. If some classes in the second layer are still too crowded, the process can be repeated for several layers in the same manner.

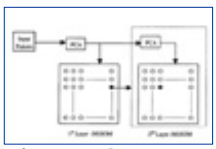


Figure 13 UHNN (2-layer PCA-MSOM) used for fingerprint classification.

Once the system is trained in order to classify a new entry, the block directional feature vector is applied as input to the first layer, then PCA transforms it into the principal subspace that it represents. The transformed

feature vector is fed as input to SOM of the first layer and the winner is determined. If the winner of the first layer is attached to a second layer PCA-SOM network, then it triggers the related network in the second layer while the feature vector extracted by the PCA of the first layer is fed as input to the related PCA in the second layer, which in turn transforms it to its principal subspace. The output of PCA of the second layer is fed as input to the related SOM and the winner is determined.

Each fingerprint in the database is assigned a hierarchical class number indicating the related neuron at each layer. When a new rolling of a fingerprint is to be used to find the original fingerprint in the database, the search is conducted within the class related to the winner of the SOM network at the last layer. However, if the fingerprint is not found within this class, the class related to the second winner of the SOM in the same layer is considered, and so on for the other winners in the order of activation. If the fingerprint is not found within any of the classes related to the last layer nodes, the search continues with the second winner of the previous layer, in the same manner as was conducted with the first winner, and so on.

4.3 Experimental Results

In this section, we present the experimental results obtained by using the algorithms discussed in section 4.2. Here, the modified SOM algorithms (MSOM) are used for training and classification instead of the original SOM algorithms.

The fingerprint database used in the experiments was obtained from The American National Institute of Technology (NIST). In the test database there are 1000 fingerprints of which the second 500 are different rollings of the first 500.

In Table 6, we present the worst-case search performance for various sizes of single-layer and two-layers SOM, MSOM, and UHNN networks. In this table, the column headings show the size of the networks used. The row headings show the percentage of the fingerprint database searched in order to find the original fingerprint.

Each entry of the table denotes the worst-case percentage of the database that is correctly identified by using the search space percentage given in the row heading. The worst-case is the one in which all elements in each class visited are used in comparison and thus contribute to the search space percentage mentioned in the row heading.

The values in the table are obtained as follows: The network is first trained using only the first rolling of the fingerprints and then it is tested through the second rolling of the same fingerprints. For each of the 500 fingerprints in the test set, the sizes of the classes related to the winners are summed up as these classes are visited to find the original rolling of the fingerprint in the manner explained in Section 4.2.2. The summation represents the worst-case search size; here all the classes visited, correctly or incorrectly, contribute in full size to the summation, although the related fingerprint can be reached when only half of the last class is searched on the average. The summation would constitute the average search size if half of the size for the last visited class were added to the full sizes of incorrectly visited classes since the original rolling would be reached on the average when the half of the class that contain the original rolling is examined through matching process. Then a histogram is prepared showing the distribution of cases on the mentioned summations. The values in the histogram are converted to percentages of the database size. The integral of the histogram up to the search percentage specified in the related row heading gives the values presented in the table, where only selected rows are included due to space limitations.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

It is evident from Table 6 that introducing a second layer to the network drastically reduces the search space percentage necessary to find the fingerprints, independently of whether SOM or MSOM is used. From the same table, we also observe that MSOM performs better than the original SOM for all cases. These results are further improved when UHNN is used, although there are some exceptional cases where two layers MSOM performs better than UHNN.

Table 6 Worst-case performances for original SOM, modified SOM, and UHNN networks. (Adapted from [12], Copyright © IEEE.)

		SOM		MSOM		UHNN
	SEARCH %	**Single layer**	**Two layers**	**Single Layer**	**Two layers**	
3-by-3	10	21.2	28.2	25.8	39.8	45.0
	20	63.4	68.0	69.0	72.0	72.8
	30	77.4	79.4	76.4	82.0	84.2
	40	81.6	88.6	87.0	94.0	92.4
	50	88.0	93.2	90.6	96.8	98.6
	60	92.4	95.0	95.4	100.0	100.0
	70	92.4	99.0	98.4		
	80	94.6	100.0	99.0		
	90	97.0		100.0		
	100	100.0				
5-by-5	10	49.0	55.0	56.8	62.8	65.0
	20	66.0	74.2	71.2	80.0	90.2
	30	76.0	86.0	84.4	94.2	100.0
	40	84.4	90.8	91.2	96.2	
	50	90.8	94.6	96.0	100.0	
	60	96.0	97.0	99.4		
	70	98.0	100.0	100.0		

	80	100.0				
	90					
	100					
8-by-8	10	51.2	55.0	57.8	62.0	70.4
	20	77.0	77.4	80.6	82.0	84.2
	30	85.0	88.0	90.8	94.3	100.0
	40	92.6	91.0	96.0	100.0	
	50	96.8	96.8	100.0		
	60	96.8	100.0			
	70	100.0				
	80					
	90					
	100					
10-by-10	10	55.6	69.6	60.2	75.4	69.4
	20	78.0	82.2	82.4	87.2	86.2
	30	88.0	92.8	94.2	96.8	100.0
	40	92.2	97.2	99.6	100.0	
	50	96.0	98.8	100.0		
	60	100.0	100.0			
	70					
	80					
	90					
	100					

For SOM and MSOM, the results improve with increasing network size. This is the general trend for the case of UHNN as well, however the best performance is obtained by the UHNN of size of 5-by-5. With this network size 90.2% of the original fingerprints are reached in the worst-case, when only 20% of the database is searched.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

5. Conclusion

In this chapter, we presented two industrial applications HALdoc and HALafis, based on the hierarchical neural network architecture called HNN. HALdoc contains a character recognition system based on HNN itself and HALafis contains a fingerprint classifier based on UHNN, the unsupervised version of HNN.

HNN has a hybrid architecture in which different ANN models have been used for different purposes. Every component of the network is an ANN model specialized for the task that the component handles. The combination resulted in a more flexible and robust system.

The proposed hierarchical architecture has an analogy to the *pre-attentive* and *attentive levels* related to attention concept in human cognition where pre-attentive level examines global features and the attentive level takes discriminating features between similar patterns into consideration. The attentive level is responsible for finding which parts of the pattern convey most discriminating information while *shadowing* unrelated features [9]. In our hierarchical neural network structure the SOM layer, which corresponds to the pre-attentive level, makes a rough decision for possible candidates. The PCA-MLP combination corresponds to the attentive level. The PCA extracts the feature components with most discriminating information for a decision among the candidates and MLP (or a second layer of SOM for unsupervised case) makes its decision paying *focused attention* to these discriminating details. Voting mechanism helps to decide in the case of alternative possibilities. HNN and UHNN performed successfully as the Character Recognition Module of HALdoc and Fingerprint Classifier module of HALafis, respectively.

Acknowledgments

HALdoc and HALafis projects are under development in Halici Group of Companies in cooperation with Middle East Technical University. HALdoc is partially supported by the grant EEEAG-55/92 of the Turkish Scientific and Technical Research Council. HALafis is partially supported by the grants TTGV-55/92 of the Turkish Technology Development Foundation and TIDEB0094-OPAT01 of the Turkish Scientific and Technical Research Council.

References

1 Almedia L. B. (1993), Neural Pre-processing Methods, *Proceedings of the NATO Advanced Studies*

Institute on Statistics and Neural Networks, Theory and Pattern Recognition Applications, Les Arcs, Bourg Saint Maurice, France, pp 213-225.

2 Bircan T. (1993), A Text Reading System Based on Artificial Neural Network, MSc Thesis, Department of Electrical and Electronics Engineering, Middle East Technical University, Ankara, Turkey.

3 Bircan T., Halici U. (1993), A Neural network Using Kirsh Masks for OCR, *Proceedings of the 7th International Symposium on Computer and Information sciences*, ISCIS 93, Istanbul.

4 Bishop C. M. (1995), *Neural Networks for Pattern Recognition*, Clarendon Press, Oxford.

5 Buyukkokten F., Isikli O., Koksal M., Halici U., Aybay I. (1994), Document Management in Haldoc, *Proc. of IEEE Melecon 94*, Antalya, Turkey.

6 Dolgun M. (1993), Text Recognition with Neural Networks, MSc Thesis, Department of Electrical and Electronics Engineering, Middle East Technical University, Ankara, Turkey.

7 Erol A. (1995), A Neural Network with Unsupervised Pre-classification and PCA Feature Extraction for Character Recognition, MSc Thesis, Department of Electrical and Electronics Engineering, Middle East Technical University, Ankara, Turkey.

8 Fahlman S. E. (1988), An Empirical Study of Learning Speed in Back-Propagation Networks, Carnegie Mellon University Computer Science Department, CMU-CS-88-162.

9 Glass A. L., Halyoak K. J., *Cognition*, McGraw Hill, 1986.

10 Halici U., Bircan T. Aybay I. Köksal M., Erol A. (1994), An Automatic Text Reading System for HALdoc, *Proc. of Bilisim 94*, Istanbul, pp 196-201.

11 Halici U., Erol A. (1995), A Hierarchical Neural Network for Optical Character Recognition, *Proc. of ICANN 95*, Paris, pp 251-256.

12 Halici U., Ongun G. (1996), Fingerprint Classification through Self Orginizing Feature Maps modified to handle uncertainties, *The Proceedings of the IEEE*, Vol. 84 No 10, pp. 1497-1512.

13 Jain A. K. (1989), *Fundamentals of Digital Image Processing*, Prentice Hall, Englewood Cliffs, NJ.

14 Kaynak C., Alpaydin E. (1996), Tecchniques for merging multiclassifiers and Handprint Recognition Application, *Proceedings of Fuzzy Logic and Neural Networks* (Eds: Ciftcibasi T, Halici U.) METU, Ankara, Turkey, pp. 88-75.

15 Kohonen T. (1998), *Self Organization and Associative Memory*, Springer Verlag.

16 Kung S. Y. (1993), *Digital Neural Networks*, PTR Prentice Hall, Englewood Cliffs, NJ.

17 Lippmann R. P. (1993), Neural Networks, Bayesian a posteriori Probabilities, and Pattern Classification, *Proceedings of the NATO Advanced Studies Institute on Statistics and Neural Networks, Theory and Pattern Recognition Applications*, Les Arcs, Bourg Saint Maurice, France, pp. 83-104.

18 Lippmann R. P. (1992), Neural Network Classifiers Estimate Bayesian a Posteriori Probabilities, *Neural Computation*, Vol. 3, pp. 461-483.

19 Malki H. A., Moghaddamjoo A. (1991), Using the Karhunen-Loe’ve Transformation in the Back-propagation Training Algorithm, IEEE *Trans. on Neural Networks*, Vol. 2, No. 1.

20 Miller B. (1994), Vital signs of identity, IEEE *Spectrum*, vol. 31, no. 2, pp. 22-30.

21 Oja E. (1991), Data Compression, Feature Extraction, and Autoassociation in Feedforward Neural Networks, *Proceedings of IJCANN-91*.

22 Ongun G. (1995), An Automated Fingerprint Identification System Based on Self Organizing Feature Maps Classifier, MSc Thesis, Middle East Technical University.

23 Ongun G., Halici U., and Ozkalayci E. (1995), A neural network based fingerprint Identification System, in *Proceedings of the Tenth International Symposium on Computer and Information Sciences*, Izmir.

24 Ongun G., Halici U., and Ozkalayci E. (1995), HALafis: An automated fingerprint identification system, in *Proceedings of Bilisim 95*, Istanbul.

25 Rubner J., Schulten K. (1990), Development of Feature Detectors by Self Organization, *Biol. Cybern*. 62, pp. 193-199.

26 Sabourin M., Mitiche A. (1992), Optical Character Recognition by a Neural Network, *Neural Networks*, Vol. 5, pp. 843-852.

27 Sabourin M., Mitiche A.(1993), Modeling and Classification of Shape Using a Kohonen Associative Memory With Selective Multiresolution, *Neural Networks*, Vol. 6, pp. 275-283.

28 Shunji M., Suen C.Y. and Yamamoto K. (1992), Historical Review of OCR Research and

Development, *Proceedings of the IEEE*, Special Issue on Character Recognition, Vol. 80, No.7.

29 Shustorovich A. (1994), A Subspace Projection Approach to Feature Extraction: The Two-Dimensional Gabor Transform for Character Recognition, *Neural Networks*, Vol. 7, No. 8, pp 1295-1301.

30 Werbos P. J. (1974) Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, Ph.D. Thesis, Harvard University, Cambridge, MA.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Chapter 7
Neural Networks for Performance Optimization in Flexible Manufacturing Systems

S. Cavalieri
University of Catania
Faculty of Engineering
Institute of Informatic and Telecommunications
Viale A. Doria, 6
95125 Catania
Italy

A Flexible Manufacturing System (FMS) is an interconnected system of resources (machine tools, robots, etc.) capable of automatically processing a wide variety of products simultaneously and under computer control [1]. Each product (also called part type) is the result of a production cycle, i.e., a sequence of processes performed by the resources available according to its technological requirements. A production cycle is generally characterized by a technological routing, i.e., a sequence of resources (e.g., machines). In an FMS, several jobs may be done simultaneously on the same part type (i.e., the same kind of product) in each production cycle. In this case, the number of jobs is called Work-in-Process (WIP). Generally, jobs are carried on facilities such as pallets or carts. Due to the very high costs incurred by the implementation of these transportation facilities and to avoid any congestion problems, it is generally desirable to minimize the WIP.

Maximization of the number of jobs done in a time unit (i.e., throughput) is a key point in FMS performance optimization. It is clear that the maximization of throughput is strongly linked to the WIP. A lesser number of jobs processed underexploits the available resources and reduces the throughput, while a higher number causes conflicts between resources, which slow down the productivity of the system. The aim is a trade-off between WIP and throughput; on account of what was said previously, it is necessary to determine a minimum WIP such that the productivity of the FMS as a whole is optimized.

One of the best-known theoretical solutions to this problem is presented in [2]. It is based on the use of Event Graphs, which are a particular category of Petri Nets [3, 4], and is characterized by the solution of an integer

linear programming problem. Two algorithms which are able to solve the FMS performance optimization problem according to the theoretical approach presented in [2], were proposed in [5]. They are essentially based on a heuristic search which aims to identify an optimal or quasioptimal solution to the problem.

The aim of this chapter is to present an alternative to these heuristic algorithms. The FMS performance optimization approach proposed here is still based on the theoretical results presented in [2], but applies a neural strategy to find an optimal or quasi-optimal solution to the problem. This is done according to two different aims. The first is to present an example of application of neural networks in real engineering applications. This example may be added to the large number of neural network applications present in the literature. The second and more important aim is to propose a novel neural model, obtained by making significant changes to the well-known Hopfield network [6][7].

The chapter will present the novel neural model and its use in FMS performance optimization. Then, in order to highlight the capacity of the neural approach to solve the problem, some examples of FMS performance optimization are shown. Finally, its computational capability will be discussed.

1. A Brief Overview of Event Graphs

An Event Graph is a Petri Net [4] in which each place (graphically represented by a circle) is connected with only one output transition (graphically represented by a thin bar) by means of an arc going from the place to the transition (in the following, the place is defined as an input place for the transition) and with only one input transition by means of an arc going from the transition to the place (this is defined as an output place for the transition) [3]. A cost (known as multiplicity) may be associated to each arc. In the following it is assumed that each arc has a multiplicity of 1. Like Petri Nets, in an Event Graph each transition may have an associated firing time, in which case the Event Graph is called timed, and each place can contain one or more tokens (graphically represented by black dots).

A transition is enabled to fire if each input place contains at least one token, and if the transition itself is not already in a firing state. If the transition is immediate, once it is enabled to fire, one token is removed from each input place and is immediately transferred to each output place. If the transition is timed, the token remains in each input place of the transition until the firing process ends (i.e., the time associated to the transition elapses). One token then disappears from each input place and a new token appears in each output place.

Figure 1 shows an example of an Event Graph. As can be seen, it is made up of 7 places and 6 timed transitions. For each transition, the firing time is represented inside the parentheses.

An Event Graph is said to be strongly connected if there is a sequence of transitions connecting any pair of places in the graph. In a strongly connected Event Graph, we can identify elementary circuits, each of which is a sequence of places and transitions that goes from one place (or from a transition) back to the same place (or to the same transition), while no other place (or transition) is repeated. In the following, an elementary circuit will be indicated with ³. For example, in Figure 1, the sequence of places and transitions $^{3}_{1}$ = {P1, t0, P0, t1, P2, t2, P3, t3, P1} is an elementary circuit, connecting place P1 with itself. The sequence $^{3}_{2}$ = {P1, t0, P0, t1, P2, t2, P3, t3, P6, t5, P5, t4, P4, t1, P2, t2, P3, t3, P1} also connects P1 with itself, but it is not an elementary circuit, as places P2, P3 and transitions t1, t2, and t3 are repeated twice.

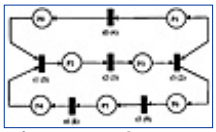


Figure 1 A strongly connected timed event graph.

If ¼(³) is the sum of firing times related to the transitions belonging to the elementary circuit ³, and M(³) the number of tokens circulating in ³, the following ratio is called the cycle time of the circuit:

$$C(\gamma)=\mu(\gamma) / M(\gamma) \qquad (1)$$

Let “ represent the set of elementary circuits in a strongly connected Event Graph, and, $C' = \max_{\gamma \in \Gamma} C(\gamma)$. Any ³ “ such that C(³) = C’ is called a critical circuit.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

1.1 Performance Optimization of Strongly Connected Event Graphs

The performance optimization theory based on Event Graphs is mainly due to the following two results. The first is due to Commoner [8].

Result 1. The total number of tokens in any elementary circuit ^{3} is invariant by transition firing.

In other words, this result states that C(^{3}) is a constant in each elementary circuit.

The other result was given by Chretienne [9].

Result 2. Under an operational mode where transitions fire as soon as they are enabled, the firing rate of each transition in the steady state is given by:

$$\rho = 1/C'$$

From Result 2, it is obvious that if we want to increase the speed (i.e., the throughput) of the system modeled by the Event Graph, we have to reduce the quantity C'. According to (1), C' may be reduced by acting on M(^{3}), adding tokens in each critical circuit.

In general, an increase in the number of tokens will lead to several disadvantages. Let us consider, for example, an Event Graph modeling a distributed system, where several resources (e.g., computers, machines) are shared among several users. In this scenario, the users are represented by the number of tokens in each elementary circuit. Although an increase in tokens leads to a better exploitation of the available resources, a decrease in the performance of the system may occur on account of the excessive conflicts for use of the resources. With particular reference to the FMS, an increase in the number of tokens in each elementary circuit may correspond to an increase in the number of jobs processed (i.e., WIP), leading to further disadvantages. In FMS, jobs are carried on such facilities as pallets or carts. Due to the very high costs incurred by the implementation of these transportation facilities and to avoid any congestion problems, it is generally desirable to minimize the in-process inventories.

A reduction in the number of tokens in each elementary circuit should not be at the cost of reducing the overall productivity of the system, modeled by the Event Graph. What is necessary to reach is an optimal trade-off between the production rate and the in-process inventories. This can be achieved by minimizing a linear function of place markings under the constraint of reaching a given level of performance.

If ± is the required performance index (i.e., the desired throughput), the performance optimization can be set as follows:

$$\text{minimize}\left(\sum_{i=1}^{n} u_i \cdot x_i\right) \qquad (2)$$

where u_i are integer numbers, called P-Invariants [3][4], x_i represents the number of tokens in place Pi, and n is the number of places in the Event Graph. Condition (2) must be constrained by

$$\rho = 1/C' \geq \alpha \qquad (3)$$

Taking into account (1), condition (3) can be rewritten as

$$M(\gamma) \geq \alpha \cdot \mu(\gamma) \quad \forall \gamma \in \Gamma \qquad (4)$$

As can be seen, condition (2) minimizes the overall number of tokens in the Event Graph, without leading to a decrease in throughput, which is guaranteed by condition (3) or (4).

As the throughput of the system is limited by the slowest transition (i.e., the transition featuring the highest firing time), ± must be chosen less than or equal to $\min_{i=1,\ldots,n} \frac{1}{\theta_i}$ where ¸$_i$, is the firing time of transition t_i.

The linear function to be minimized (2) should not depend on the state of the system. In other words, this function should remain unaltered by any transition firing, otherwise, its minimization would be meaningless. For this reason, the use of the P-Invariants in (2) is needed (see [3] for a better explanation of P-Invariants).

Consider the Event Graph presented in Figure 1. It can be shown that a possible P-Invariant of this net is the vector U = (1, 1,2,2,1,1,1), where the i-th component, u_i, (i = 1,.., 7) refers to the i-th place in the Event Graph considered. In this case, the performance optimization problem to solve is as follows:

$$\text{minimize}(x_1 + x_2 + 2x_3 + 2x_4 + x_5 + x_6 + x_7)$$

under the constraints

$$x_1 + x_2 + x_3 + x_4 \geq 13 \cdot 0.1 = 1.3$$
$$x_3 + x_4 + x_5 + x_6 + x_7 \geq 26 \cdot 0.1 = 2.6$$

having chosen ± = 0.1 as the highest firing time is 9. An optimal solution to this problem is

$$x_1 = 2,\ x_2 = x_3 = x_4 = 0,\ x_5 = 3,\ x_6 = x_7 = 0$$

2. FMS Performance Optimization by Event Graphs

The results presented in the previous section allow us to define a methodology to optimize the performance of an FMS in terms of throughput and WIP. The methodology is based on the realistic assumption of the cyclic manufacturing of jobs in the FMS. On this basis, when a job is completed and leaves the production cycle, a new job is inserted in the same production cycle.

The FMS performance optimization methodology is made up of two phases: modeling the FMS, and definition and solution of an integer linear programming problem.

2.1 Modeling an FMS by Event Graphs

Modeling an FMS by Event Graphs is very easy and consists of representing each FMS resource (buffer, machine, robot, etc.) by a place. Moreover, each activity a resource performs is represented by a transition, to which the time required to complete the activity is associated [2, 3, 5]. The cyclic manufacturing is modeled connecting the transition concerning the last process of each production cycle with the transition corresponding to the first process of the same production cycle by means of a place. In this way, when a token (i.e., a job) leaves the production cycle, a new one is inserted in the same production cycle.

The Event Graph in Figure 2 is a significant example of FMS modeling. It refers to a Kanban system [10]

composed of three machines, M1, M2 and M3, which manufacture two different part types denoted as R1 and R2.

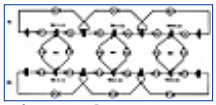


Figure 2 Event graph model of a Kanban system with three machines and two part types.

Each place in the Event Graph represents a resource: P18, P19, P20, P21, P22 and P23 model possible states of the machines (the machine Ml, for example, can be in the state P18 or P19 according to whether it is processing R1 or R2 part type), while the remaining places are input/output buffers in which the jobs belonging to the two part types wait to be processed by the relative machines. As far as the transitions are concerned, only TR12, TR22, TR32, TR11, TR21, and TR31 are timed (the figure shows the time associated with each transition), while the remaining ones are immediate, i.e., the time associated with them is zero. More specifically, transitions TR12, TR22, and TR32 model the activities performed by the machines M1, M2, and M3 to process part type R2, while transitions TR11, TR21, and TR31 model the activities performed by the machines M1, M2, and M3 for part type R1. The presence of places PO, P3, P6, P9, P12, and P15 guarantees the cyclic manufacturing assumption.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

The Event Graph in Figure 3 shows another example of FMS modeling by an Event Graph. It models a job-shop composed of four machines, M1, M2, M3, and M4, which manufacture three part types denoted by R1, R2, and R3 [5]. As can be seen, the job-shop features a production cycle related to R1, another production cycle related to R2, and two production cycles both related to the part type R3. For each production cycle, the Event Graph models the sequence of activities performed by the four machines to process the specific part type. In particular, the sequence of activities to process the part type R1 is {M1, M2, M3, M4}; in the R2 production cycle, the sequence is {M1, M4, M3}; and in R3, the sequence of activities is {M1, M2, M4}. The processing times are shown in Figure 3 for each transition. Finally, the presence of the places P3, P6, P9, and P12 again refers to the cyclic manufacturing assumption.

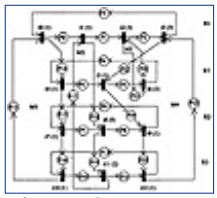


Figure 3 Event graph model of a job-shop.

2.2 Integer Linear Programming Problem

In order to apply the theory shown in Section 1 to FMS performance optimization, some modifications must be introduced. As said in the introduction, FMSs feature the presence of sequence of activities performed by a particular resource (e.g., a machine). As seen in Section 2.1, the Kanban system represented in Figure 2 features the presence of three machines, each processing part types R1 and R2. The elementary circuits $^{3}_{1}$ (made up of the sequence of places and transitions P18, TR11, P19, TR12), $^{3}_{2}$ (made up of the places and transitions P20, TR21, P21, TR22), and $^{3}_{3}$ (made up of the sequence of places and transitions P22, TR31, P23, TR32), represents the sequence of activities performed by machine M1, M2, and M3, respectively. The same considerations can be applied to the Event Graph shown in Figure 3. It features the presence of four machines, each performing a sequence of activities modeled by the elementary circuits $^{3}_{1}$ (made up of P13, t0, P14, t4, P15, t7, P16, t10), $^{3}_{2}$ (made up of P17, t1, P18, t8, P19, t11), $^{3}_{3}$ (made up of P20, t2, P21, t6), and $^{3}_{4}$ (made up of P22, t3, P23, t5, P24, t9, P25, t12).

In the following, each elementary circuit which specifies the operations sequentially performed by a single

resource is called a command circuit. A generic command circuit will be indicated with ³c, and “c will represent the set of command circuits. If “* = “ - “c, ³* will indicate the generic elementary circuit belonging to “*.

Two considerations must be made concerning the command circuits. The first is about the number of tokens present in each of them. It must be exactly equal to 1. A higher number of tokens would be meaningless, as a machine can only process one job at a time. The second consideration concerns the throughput of the system which is now limited by the bottleneck machine, i.e., by the machine featuring the highest processing time.

According to these considerations, the FMS optimization problem, for a strongly connected Event Graph and in the presence of command circuits, can be formulated through the following integer linear programming problem:

$$\begin{cases} \text{minimize}\left(\sum_{i=1}^{n} u_i \cdot x_i\right) & (5a) \\ M(\gamma^c) = 1 \quad \forall \gamma^c \in \Gamma^c & (5b) \\ M(\gamma^*) \geq \alpha \cdot \mu(\gamma^*) \forall \gamma^* \in \Gamma^* & (5c) \end{cases} \qquad (5)$$

where ± is the FMS performance index required [2, 5]. As can be seen, condition (5.b) fixes the number of tokens in each command circuit equal to 1. Condition (5.c) fixes a lower bound for the number of tokens in each elementary circuit which is not a command circuit. The desired throughput, or, must be fixed taking into account the bottleneck machine. In particular, ± must be chosen less than or equal to the minimum value of $\frac{1}{\mu(\gamma^c)}$, ³c “c

3. A Novel Neural Model

The aim of this section is to describe the novel neural model used to solve the FMS performance optimization problem. In order to gain a better understanding of the need for a new neural model, the following subsection presents the original Hopfield model [6, 7] on which the new model is based. Then, the limits of the original Hopfield model and the capacity of the new model to overcome them will be pointed out.

3.1 The Hopfield Neural Network

The Hopfield neural model is very suitable for solving optimization problems. This network was first used to solve the well-known Traveling Salesman Problem (TSP) [11], and then its use was extended to a large number of optimization problems (see [12 - 16]). The Hopfield-type model is based on a single-layer architecture of neurons, the outputs of which are fed back toward the inputs.

Figure 4 shows a Hopfield network with n neurons. As can be seen, the i-th neuron (which is drawn by a circle), i = 1,..,n, receives the outputs of the other neurons and an external fixed bias current I_i, and produces an output O_i. Each feedback between the output of the j-th neuron and the input of the i-th neuron has an associated weight, w_{ij}, which determines the influence of the j-th neuron on the i-th neuron. In the i-th neuron, the weighted sum of the outputs, O_i, and the external bias current, I_i, produces a signal, U_i, given by:

$$\frac{dU_i}{dt} = -\frac{U_i}{\tau} + \sum_j w_{ij} \cdot O_j + I_i \qquad (6)$$

where Ä is a user-selected decay constant. As can be seen, (6) describes the dynamics of the Hopfield model. The output of the i-th neuron, O_i is linked to U_i by a monotonic increasing function. In this chapter, the following function will be considered:

$$O_i = g_i(U_i) = \frac{1 + \tanh\left(\frac{U_i}{u_0}\right)}{2} \qquad (7)$$

where the parameter u_0 controls the effective steepness of the function: the lower it is, the steeper the function. As can be seen, the function g_i, gives a positive value O_i ranging between 0 and 1.

Hopfield [7] showed that if the weights matrix $W=[w_{ij}]$ is symmetrical, the dynamics of the neurons described by (6) follow a gradient descendent of the quadratic energy function, also known as the Lyapunov function:

$$E = -\frac{1}{2} \cdot \sum_i \sum_j w_{ij} \cdot O_i \cdot O_j + \frac{1}{\tau} \sum_i \int_0^{O_i} g^{-1}{}_i(O) dO - \sum_i I_i \cdot O_i \qquad (8)$$

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Its derivative with the respect to time is

$$\frac{dE}{dt} = -\sum_i \frac{dO_i}{dt} \cdot \left(\sum_j w_{ij} \cdot O_j - \frac{U_i}{\tau} + I_i\right) \qquad (9)$$

in which the term in brackets coincides with dU_i/dt as defined in (6). According to the definition of the function g_i given by (7), it follows that if dU_i/dt d0, then dO_i/dt d0 as well and, vice versa, if dU_i/dt e0 then dO_i/dt e0 as well. Equation (9) therefore establishes that dE/dtd0, i.e., the energy function is always decreasing during evolution of the network. In addition, the achievement of a steady state, $dO_i/dt = 0$, corresponds to a minimum of the energy function (i.e., $dE/dt = 0$).

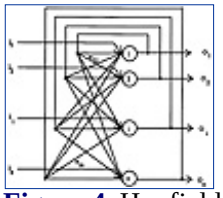


Figure 4 Hopfield neural model.

Again in [7], Hopfield showed that if the function g_i is a steep-like curve (i.e., u_o ' 0), then (6) coincides with:

$$E = -\frac{1}{2} \cdot \sum_i \sum_j w_{ij} \cdot O_i \cdot O_j - \sum_i I_i \cdot O_i \qquad (10)$$

In this case, it can be shown that the minima of the energy function (10) coincide at the corners of the hypercube defined by O_i {0,1}. This means that the Hopfield model iterates until a stable solution is reached. This stable solution coincides with a minimum of the energy function (10) and is coded by output values equal to $O_i = 0$ or $O_i = 1$.

These theoretical results allow a solution to a particular optimization problem to be obtained from the stabilized outputs of the Hopfield network, by the following method. First, the surrounding conditions of the optimization problem are identified and expressed in the form of the energy function given by (10). Then,

each term of the energy function relating to a surrounding condition is multiplied by a real coefficient which weights the influence of the condition itself on the solution to the problem. By comparison of each term linked to a surrounding condition with function (10), the weights and bias currents values are finally obtained as a function of the real coefficients. If the weights matrix is symmetrical and the function g_i is steep-like, the stable output of the Hopfield model corresponds to a minimum of (10) and thus to a solution to the problem.

3.2 Description of the Novel Neural Model

As pointed out above, the original Hopfield model features the presence of a constant bias current throughout the neural evolution. The bias current plays an essential role in the dynamics of the neural model since, as can be seen in (6), it determines the value of each neuron. A bias current value close to zero makes the output, O_i, of the i-th neuron assume a value of 1 or 0, depending exclusively on the weighted sum $\sum_j w_{ij} \cdot O_j$. A very positive bias current value determines an output value close to 1.

By virtue of its role, the bias current is always tied to one or more surrounding conditions of the optimization problem to be solved, so as to make the output of each neuron respect the surrounding condition/s in question. In particular, the neural model proposed in [6, 7] is applied to solving optimization problems in which one or more surrounding conditions impose rigid constraints on the solution, in which case an appropriate fixed bias current value can ensure the validity of the solution. For example, many optimization problems, like the TSP [11], feature at least one surrounding condition which imposes a fixed number of activated neurons. In such cases, a bias current value proportional to the number of neurons that must be activated is generally fixed.

There are, however, many optimization problems (such the FMS optimization problem dealt with in this chapter) whose surrounding conditions do not impose rigid constraints on the solution, but allow it to vary within an admissible range of values. An example would be a problem such that the validity of its solution is guaranteed by the presence in the binary coding of the solution of a number of activated neurons that can vary within an interval known a priori. Application of the Hopfield neural model presented in [6, 7] to problems of this kind may not give valid solutions, as it would rigidly fix the number of activated neurons.

A strategy which can solve this kind of problem features the possibility of dynamically modifying the bias current values during the neural iterations in order to meet all the surrounding conditions. The neural model which is proposed to achieve this is shown in Figure 5 [17 - 20]. Comparison between Figures 4 and 5 highlights the modifications made to the model originally proposed in [6, 7].

Two different levels can be seen in the novel neural model: the Bias Current Level and the Neuron Level. Both levels receive the fed-back neural outputs. The Bias Current level is made up of processing units, represented in Figure 5 by squares and labeled with I_i (i = 1,.., n). Each processing unit I_i is connected with the corresponding neuron with index i, to which it supplies a bias current. At each neural iteration, the processing units I_i receive the output values of all the neurons. If the neural solution meets all the surrounding conditions at that iteration, the bias current each neuron is supplied with is left unaltered. If, on the other hand, one or more conditions are not met, the bias current for each neuron which does not meet the condition(s) is modified. Obviously, this modification has to be made in such a way as to force the neural solution to meet the surrounding condition(s). The Neuron Level is the same as the one in the Hopfield model presented in [6, 7]. As can be seen in Figure 5, this level is made up of neurons (represented by circles), each of which receives its own output, the outputs of the remaining neurons and the bias current supplied by the corresponding processing unit in the Bias Current Level.

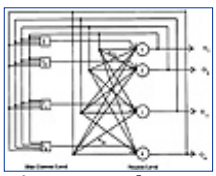


Figure 5 The novel neural model proposed.

Processing at the two levels is in cascade as it is first necessary to modify the bias currents for each neuron (at the Bias Current Level) and then calculate the output (at the Neuron Level).

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

3.2.1 Analytical Description of the Novel Neural Model

From the description of the novel neural model given previously, the dynamics of the model are quite straightforward to deduce. They are described by

$$\frac{dU_i}{dt} = -\frac{U_i}{\tau} + \sum_j w_{ij} \cdot O_j + I_i(t)$$

which is identical to (6) except for the term, $I_i(t)$, relating to the bias current, which now varies in time. If the weights matrix $W = [w_{ij}]$ is symmetrical, (8) continues to hold and its derivative with the respect to time now becomes

$$\frac{dE}{dt} = -\sum_i \frac{dO_i}{dt} \cdot \left(\sum_j w_{ij} \cdot O_j - \frac{U_i}{\tau} + I_i\right) - \sum_i \frac{dI_i}{dt} \cdot O_i \qquad (11)$$

From (11) it can be seen that, unlike the original Hopfield model, the evolution in time of the energy function also depends on the term $\sum_i \frac{dI_i}{dt} \cdot O_i$. For the energy function to continue decreasing, during the neural iterations the following must always hold:

$$\sum_i \frac{dI_i}{dt} \cdot O_i \geq 0 \qquad (12)$$

That is, it is necessary for the bias currents to be modified by means of increments (since the outputs O_i are always positive or equal to zero). This ensures convergence toward a minimum of the function during evolution of the neural network.

If (12) holds, the modifications made to the original Hopfield model do not jeopardise stability, as convergence to a (local or global) minimum of the Lyapunov function is always guaranteed. This is confirmed by experimental tests performed on a large number of different optimization problems. In all the tests, it was seen that the modification made to the Hopfield model does not alter the stability of the network as long as

condition (12) holds and the symmetry of the weights matrix is maintained.

4. FMS Performance Optimization Using the Novel Neural Model

The solution of the FMS performance optimization problem by the novel neural model may be achieved by the following four steps.

4.1 Modeling the FMS by an Event Graph

The FMS is first modeled by means of an Event Graph. As stated previously, modeling an FMS by an Event Graph is very easy, and Section 2.1 has shown two examples.

4.2 Mapping the Event Graph onto the Neural Model

Once the FMS has been modeled by the Event Graph, it is transformed into a neural model of the type presented in Section 3.2. This is achieved by making each place, Pi, in the Event Graph correspond to a neuron, O_i, in the neural network. The value (1 or 0) of the output of each neuron O_i models the presence or absence of a token in the place Pi modeled: if the output O_i is 1 the place Pi corresponding to the neuron contains a token, if O_i is 0 the place Pi contains no tokens. It is clear that the proposed coding of the neural output is based on the necessary assumption that each place in the Event Graph contains at most one token. This does not represent a limit, as it is demonstrated in [5] that it is always possible to modify a generic Event Graph into an equivalent one in which each place possesses at most one token. The mapping between the Event Graph modeling the FMS and the novel neural model allows the number of neurons it contains to be defined. For example, the Event Graph shown in Figure 2 is transformed into a neural network of the kind shown in Figure 5 with 24 neurons.

4.3 Determination of Weights and Bias Currents for the Novel Neural Model

Once the number of neurons and their relationship with the Event Graph modeling the FMS has been fixed, the next step is to obtain the weights and bias current values for the neural model. This must be done by expressing each surrounding condition of the FMS performance optimization problem in terms of the Lyapunov Energy Function (10), and then by comparing each of these terms with (10). In the following, the determination of weights and bias currents will be shown for each of the three surrounding conditions of the FMS performance optimization problem. As said before, the first surrounding condition (5a) refers to the quality of solution, while the other two surrounding conditions (5b) and (5c) refer to the validity of the solution.

4.3.1 Quality of the Solution

The energy function term relating to the quality of the solution can be obtained by considering that condition (5a) can also be expressed in the following form:

$$\text{minimize } \left(\sum_{i=1}^{n} u_i \cdot x_i\right) = \text{minimize } \left(\sum_{i=1}^{n} u_i \cdot x_i\right)^2 \qquad (13)$$

where n, as said, is the number of places in the Event Graph, i.e., the number of neurons in the novel neural model. Condition (13) can be expressed by

$$\text{minimize } \left(\sum_{i=1}^{n}\sum_{j=1}^{n} u_i \cdot x_i \cdot u_j \cdot x_j\right)^2 \qquad (14)$$

This condition can be formalized by the following term of the Lyapunov energy function:

$$\frac{A}{2} \cdot \left(\sum_{i=1}^{n}\sum_{j=1}^{n} u_i \cdot O_i \cdot u_j \cdot O_j\right) = \frac{A}{2} \cdot \left(\sum_{i=1}^{n}\sum_{j=1}^{n} u_i \cdot u_j \cdot O_i \cdot O_j\right) \qquad (15)$$

based on the consideration that each x_i corresponds to O_i, according to the mapping between the Event Graph and the neural model, as outlined previously. The aim of the real coefficient A is to weight the influence of condition (15) on the solution to the problem. By comparing (15) with (10), we obtain the contribution to the bias currents and weights of this term:

$$w_{ij} = -A \cdot u_i \cdot u_j \quad \forall\, i,j \in [1,..,n] \qquad (16a)$$

$$I_i = 0 \quad \forall\, i \in [1,..,n] \qquad (16b)$$

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

4.3.2 Validity of the Solution

The energy function term relating to the validity of the solution has to impose a certain number of tokens in each elementary circuit ^{3} in the Event Graph, according to (5b) and (5c). The corresponding term of the Lyapunov energy function has to be of the following kind:

$$\frac{\text{Constant}}{2}\cdot\sum_{\gamma}\left(\sum_{i:Pi\in\gamma}O_i - M_t(\gamma)\right)^2 \qquad (17)$$

where the first sum is extended to every elementary circuit ^{3}, the second sum is extended to i [1,.., n] such that the place Pi belongs to the elementary circuit ^{3}, and $M_t(^3)$ represents the number of tokens desired in each circuit ^{3}. The term (17) is minimized when the number of activated neurons in each circuit ^{3} is equal to $M_t(^3)$. The value of $M_t(^3)$ is given by conditions (5b) and (5c).

- **Validity of the solution according to (5b).** Condition (5b) imposes a single token in each of the command circuits ^{3c}. It corresponds to the Lyapunov function term:

$$\frac{B}{2}\cdot\sum_{\gamma^c}\left(\sum_{i:Pi\in\gamma^c}O_i - 1\right)^2 \qquad (18)$$

where B, like A, weights the influence of condition (18) on the solution to the problem. As can be seen, term (18) is minimized when a single token is present in each command circuit ^{3c}. The contribution to the bias currents and weights of (18) becomes

$$w_{ij} = -B\cdot n_{\gamma^c_{i,j}} \quad \forall\, i,j\in[1,..,n] \qquad (19a)$$

$$I_i = +B\cdot n_{\gamma^c_i} \quad \forall\, i\in[1,..,n] \qquad (19b)$$

where $n_{\gamma^c_{i,j}}$ represents the number of circuits in the set "^{c} to which the places Pi and Pj Simultaneously

belong, and $n_{\gamma_i^c}$ represents the number of circuits in the set “ to which the place Pi belongs. It is important to point out that the bias currents given by (19b) exactly fix one token in each command circuit ^{3c}.

- **Validity of the solution according to (5c).** Condition (5c) establishes that $M_t(^3)$ has to be greater or at least equal to $\lceil \alpha \cdot \mu(\gamma^*) \rceil$ in all the circuits ^{3*} “*. Let us consider the subcondition:

$$M_t(\gamma) = \lceil \alpha \cdot \mu(\gamma^*) \rceil \ \forall \gamma^* \in \Gamma^* \qquad (20)$$

It can be formalized by the following Lyapunov function term:

$$\frac{C}{2} \cdot \sum_{\gamma^*} \left(\sum_{i:Pi \in \gamma^*} O_i - \lceil \alpha \cdot \mu(\gamma^*) \rceil \right)^2 \qquad (21)$$

where C performs the same role as A and B. Comparison with (10) gives the following weights and bias current values:

$$w_{ij} = -C \cdot n_{\gamma_{i,j}^*} \quad \forall i,j \in [1,.., n] \qquad (22a)$$

$$I_i = +C \cdot \sum_{\gamma^*:Pi \in \gamma^*} \lceil \alpha \cdot \mu(\gamma^*) \rceil \quad \forall i \in [1,.., n] \qquad (22b)$$

where $n_{\gamma_{i,j}^*}$ represents the number of circuits in the set “* to which the places Pi and Pj simultaneously belong. As can be seen, condition (22a) determines a constant bias current value, fixing the number of tokens in each non-command elementary circuit $\lceil \alpha \cdot \mu(\gamma^*) \rceil$. In this way, condition (5c) may not be respected, because a valid solution may feature the presence of some circuits in which the number of tokens is strictly greater than $\lceil \alpha \cdot \mu(\gamma^*) \rceil$. These considerations show that the solution of the optimization problem being dealt with requires dynamic modification of the bias current values during the neural iteration, in order to cause the number of tokens in each circuit ^{3*}, to be no less than $\lceil \alpha \cdot \mu(\gamma^*) \rceil$. This can be obtained by using the processing units in the Bias Current Level of the novel neural model presented in Section 3.2. The dynamic modification of the bias currents will be explained in greater detail in the following subsection.

4.4 FMS Performance Optimization by the Novel Neural Model

Once the weights and bias currents have been determined, the novel neural model can provide a solution constrained to satisfy the surrounding conditions (5a), (5b) and (5c) of the FMS performance optimization problem. The respect of conditions (5a) and (5b) is guaranteed by imposing the weights and bias current values given by (16a), (16b), (19a), and (19b). The respect of condition (5c) is guaranteed by imposing the weight values given by (22a) and providing for modification of the bias currents during the neural iteration. This modification is performed by the processing units in the Bias Current Level, and its aim is to force the number of tokens in each circuit ^{3*} “*, to be no less than $\lceil \alpha \cdot \mu(\gamma^*) \rceil$. The bias current modification procedure will be described in greater detail below.

During a preliminary phase, the values for the weights of the novel neural model are fixed according to conditions (16a), (19a) and (22a). They are equal to:

$$w_{ij} = -A \cdot u_i \cdot u_j - B \cdot n_{\gamma_{i,j}^c} - C \cdot n_{\gamma_{i,j}^*} \ \forall i,j \in [1,.., n] \qquad (23)$$

The bias currents, on the other hand, are fixed on the basis of the contribution made by (16b) and (19b). The contribution of (22b) is not initially considered because, as said above, it might determine a non-valid solution. The initial value of the bias currents is thus:

$$I_i = B \cdot n_{\gamma_i^c} \quad \forall i \in [1,..., n] \qquad (24)$$

Once these initial values have been fixed, the network starts to iterate. At each neural iteration, the generic processing unit I_i in the Bias Current Level checks whether the output of the corresponding neuron, O_i, is close to 0. If it is, all the elementary circuits ^{3*} to which the corresponding place Pi belongs are considered.

The number of tokens M(^{3*}) for each of these circuits is counted. If it is less than $\lceil \alpha \cdot \mu(\gamma^*) \rceil$, i.e., condition (5c) is not met for that circuit, the bias current of the i-th neuron being considered is increased by:

$$\delta C \cdot \sum_{\gamma^*: P_j \in \gamma^*} \lceil \alpha \cdot \mu(\gamma^*) \rceil \qquad (25)$$

where the value ´C represents a user-selected fraction of C. If the processing unit I_i detects activation of the corresponding neuron, the bias current value is left unaltered. In this way the increase in current provided by (25) is a fraction of the contribution of condition (5c) expressed by (22b). In addition, the bias current is always increased, respecting the condition requiring convergence toward a minimum of the energy function expressed by (12). The modification of the bias currents in the novel neural model can be better explained by the Pascal-like algorithm shown in Figure 6.

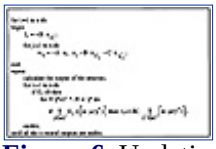


Figure 6 Updating the bias currents during neural iterations.

5. Examples of FMS Performance Optimization by the Neural Approach

The aim of this section is to give some FMS performance optimization results obtained using the neural approach presented above. Two examples will be considered which are well known in literature and refer to a deterministic Kanban system [10] and to a job shop system [5].

As noted before, the Event Graph in Figure 2 shows a Kanban system composed of three machines, M1, M2, and M3, which manufacture two part types denoted as R1 and R2. It can easily be verified that the Event Graph in Figure 2 is strongly connected. So the theory presented before (see equations (???)) can be applied, aimed at determining the optimal number of jobs (i.e., the number of tokens) simultaneously present in the production cycles R1 and R2.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Table 1 refers to the solution of this Kanban system performance optimization problem provided by the neural strategy proposed in the previous sections, considering a value of ± = 1/3 and values of u_i = I i [0, 17] and u_i = 0 i [18, 23]. The value of ± = 1/3 is due to the processing times in each of the three command circuits. As can be seen in Figure 2, the sum of the firing times in these circuits is always 3.

For each place in the Event Graph, the table shows the distribution of the tokens determined by the neural network. Only the places containing one token are shown. The places not shown in Table 1 contain no token. As can be verified, this solution produces the best performance for the whole system.

Table 1 Solution by the Neural Approach of the Kanban System Performance Optimization Problem.

P2	P3	P7	P11	P14	P17	P18	P20	P22
1	1	1	1	1	1	1	1	1

The Event Graph shown in Figure 3 models a job-shop composed of four machines M1, M2, M3, and M4, which manufacture three part-types denoted by R1, R2 and R3. Here again, the theory seen before (equations (5)) can be applied as the Event Graph in Figure 3 is strongly connected.

Table 2 refers to the solution of the job-shop performance optimization problem provided by the neural strategy proposed in the previous sections, considering a value of ± = 1/6, imposed by the bottleneck machine M4, and the following values of u_i in (1): U_i = 1 i [0, 12] and u_i = 0 i [13, 25]. For each place in the Event Graph, the table shows the distribution of the tokens determined by the neural network. As before, only the places containing one token are shown. Again, it can be verified that this solution produces the best performance for the whole system.

Table 2 Solution by the Neural Approach of the Job-Shop System Performance Optimization Problem

P2	P3	P4	P5	P8	P11	P13	P17	P20	P23
1	1	1	1	1	1	1	1	1	1

6. Some Considerations on the Neural Computation

The aim of this section is to give some indication about the time required for the neural solution proposed here to reach a solution of the FMS optimization problem.

The time required to compute a solution depends on the complexity of the following types of operations that have to be performed in sequence:

1. Before the neural network starts to iterate, it is necessary to determine all the network weights. In the FMS performance optimization problem, the initialization of the weights and bias currents is represented by (23) and (24). As can easily be seen, the time needed to initialize the weights and bias currents is proportional to the size of the weights matrix, i.e., to the number of neurons squared. As each neuron corresponds to a place in the Event Graph modeling the FMS, this time is proportional to the number of places in the Event Graph squared. This computational overhead is generally known as programming complexity, which, on account of what said, is $O(n^2)$.

2. At each iteration, the bias current has to be determined for each node. This operation is performed by the processing units in the Bias Current Level, indicated in Figure 5 as I_i. Figure 6 gives the algorithm used to calculate these currents, which will be repeated in the following for the sake of clarity:

a. for i: = 1 to n do
  if $O_i = 0$ then
b. for $\forall \gamma^* \in \Gamma^*$: $Pi \in \gamma^*$ do
c. if $\sum_{j:Pj \in \gamma^*} O_j < \lceil \alpha \cdot \mu(\gamma^*) \rceil$ then $I_i += \delta C \cdot \sum_{\gamma^*:Pj \in \gamma^*} \lceil \alpha \cdot \mu(\gamma^*) \rceil$

It comprises two for cycles (lines (a) and (b)). The first (line (a)) features the index i: = 1 to n, i.e., it executes n operations. It is, in fact, necessary to determine the bias currents, $I_i(t)$, for all the neurons in the network. For each index i, the second cycle (line (b)) features as many iterations as there are elementary circuits ³* "* to which the place Pi (i = 1,.., n) belongs. Let $nmax_3$ be the maximum number of elementary circuits each place can belong to. According to this definition, for each index i, $nmax_3$ iterations are done in the worst case. The number of operations performed in each iteration of the *for* cycle in line (b) is equal to the number of places belonging to each of the elementary circuits ³* considered. In fact, the condition controlled by the if in line (c) relates to the sum of the network outputs for all the places belonging to the elementary circuit ³* being considered by the *for* cycle in line (b). Let $nmax_i$ be the maximum number of places belonging to an elementary circuit; it is obvious that the number of operations performed in each iteration of the for cycle in line (b) is equal to $nmax_i$ in the worst case. On account of what was said, it is clear that the computational complexity of calculating the bias currents is $O(n \cdot nmax_3\ nmax_i)$.

3. At each iteration, the neural network output is obtained by calculating the weighted sum of all the n inputs. It can be easily shown that the complexity of this operation is $O(n^2)$, as the computational complexity to determine the weighted sum of all the n inputs of each neuron is O(n).

4. The neural network must perform a certain number of iterations until it converges toward a stable state. In [21], it was shown that the number of iterations is always very limited (a few hundred iterations) and is practically independent of the complexity of the problem (i.e., the number of neurons in the network).

Therefore, it is clear that the time needed for the neural network to converge and to provide for a solution to the FMS Performance Optimization problem is therefore equal to the sum of the complexities of operations (1), (2), and (3), i.e., $O(n^2 + n \cdot nmax_3\ nmax_i)$.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

7. Conclusion

This chapter has presented an original approach to FMS performance optimization, featuring the application of a neural strategy to find an optimal or quasi-optimal solution to the problem.

A novel neural model has been proposed and described. It derives from the original Hopfield model, to which some modifications were applied. The need for this model has been clearly pointed out in the chapter. Essentially, the original Hopfield model is able to solve optimization problems in which one or more surrounding conditions impose rigid constraints on the solution. There are many optimization problems whose surrounding conditions do not impose rigid constraints on the solution, but allow it to vary within an admissible range of values. Application of the original Hopfield neural model to problems of this kind may not give valid solutions. The proposed neural model was conceived to overcome this limit, allowing the bias current to vary during the evolution of the neural network.

The FMS performance optimization is based on the modeling of an FMS by the proposed novel neural model. The solution the neural network provides determines the FMS configuration which will maximize performance.

From tests carried out on a large number of examples, it was found that the quality of the neural solution is always high, as the solution reached is always optimal or close to optimal. Furthermore it was experimentally found that the computational time the neural approach provides for is polynomial with the dimension of the FMS (i.e., number of resources). This was confirmed by the computational analysis presented in the chapter.

References

1 J.R. Pimentel (1990), *Communication Networks for Manufacturing*, Prentice-Hall International Edition.

2 H.P. Hillion and J.M.Proth (1989), Performance Evaluation of Job-Shop Systems using Timed Event Graphs, *IEEE Transaction on Automatic Control*, vol.34, no. 1, pp.3–9.

3 R. Zurawski (1994), Petri Nets and Industrial Application: A Tutorial, *IEEE Transaction on Industrial Electronics*, vol.41, no.6, pp.567–583.

4 M. Ajmone Marsan, G. Conte and G. Balbo (1986), *Performances Models of Multiprocessor*

Systems, Computer System Series.

5 S. Laftit, J.M. Proth and X.L. Xie (1992), Optimization of Invariant Criteria for Event Graphs, *IEEE Transaction on Automatic Control*, vol.37, no.5, pp.547–555.

6 J.J. Hopfield (1992), Neural Networks and Physical Systems with Emergent Collective Computational Abilities, *Proceedings National Academy of Sciences*, vol. 79, pp.2554–2558.

7 J.J. Hopfield (1994), Neurons with Graded Response Have Collective Computational Properties Like those of two-state Neurons, *Proceedings National Academy of Sciences*, 81:3088–3092.

8 F. Commoner, A. Holt, S. Even and A. Pnueli (1971), Marked Directed Graphs, *Journal of Computer and System Science*, vol.5, no.5.

9 P. Chretienne (1983), Les Reseaux de Petri Temporises, *These d'Etat*, Universite' de Paris VI, Paris France.

10 M. Di Mascolo, Y. Frein, Y. Dallery and R. David (1989), A Unified Modelling of Kanban Systems using Petri Nets, *Technical Report no.89-06*, LAG.Grenoble, France.

11 J.J. Hopfield and D.W. Tank (1985), Neural Computations of Decisions in Optimization Problems, *Biol. Cybern.*, vol.52, pp. 1–25.

12 D.W. Tank and J.J. Hopfield (1986), Simple 'Neural' Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit, *IEEE Trans. Circuits. Syst.*, vol.CAS-33, no.5, pp.533–541.

13 M.K. Mehmet Ali and F. Kamoun (1993), Neural Networks for Shortest Path Computation and Routing in Computer Networks, *IEEE Transaction on Neural Networks*, vol.4, no.6, pp.941–954.

14 S. Cavalieri, A. Di Stefano and O. Mirabella (1994), Optimal Path Determination in a Graph by Hopfield Neural Network, *Neural Networks*, vol.7, no.2, pp.397–404.

15 S. Cavalieri and O. Mirabella (1996), Neural Networks for Process Scheduling in Real-Time Communication Systems, *IEEE Transactions on Neural Networks*, vol.7, no. 5, pp. 1272–1285.

16 M.M. Ali and H. Tri Nguyen (1989), Neural Network Implementation of an Input Access Scheme in a High Speed Packet Switch, *Proceedings Globecom 89*, Dallas, TX, pp. 1192–1197.

17 S. Cavalieri (1996), Performance Optimization of Flexible Manufacturing Systems using Artificial Neural Networks, *Proceedings The Ninth International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*, Fukuoka, Japan, pp.479–486.

18 S. Cavalieri (1996), Applications of Neural Networks to Factory Automation Performance Optimization, *Proceedings Computational Engineering in Systems Applications*, Lille, France, pp.240–245.

19 S. Cavalieri (1996), A Novel Neural Network Model for the Performance Evaluation of Flexible Manufacturing Systems, *Proceedings IEEE 22nd Annual International Conference on Industrial Applications*, Taipei, Taiwan.

20 S. Cavalieri (1996), Enhancing Hopfield Neural Net Capabilities in Solving Optimization Problems, *Proceedings World Congress on Neural Networks*, Town & Country Hotel, San Diego, CA, USA.

21 S.V.B. Aiyer, M. Niranjan and F. Fallside (1990), A Theoretical Investigation into the Performance of the Hopfield Model, *IEEE Transaction on Neural Networks*, vol. 1, no.2, pp.204–215.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Previous | Table of Contents | Next

Chapter 8
Channel Assignment in Mobile Communication Networks - A Computational Intelligence Approach

Gangsheng Wang
Department of Multimedia Communications
Sharp Laboratories of America, Inc.
5750 NW Pacific Rim Blvd.
Camas, WA 98607
U.S.A.

Nirwan Ansari
Center for Communications and Signal Processing
Department of Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, NJ 07102
U.S.A.

The channel assignment problem arises when the scarce and expensive spectral resources must be shared and utilized efficiently. In this chapter, a computational intelligence approach is introduced to allocate channels in a mobile communication network. Numerical simulations have demonstrated the feasibility of the proposed approach.

1. Introduction

The recent telecommunications revolution poses a greater demand and sophistication on optimization problems encountered in today's telecommunications systems. As a result, conventional optimization techniques can no longer meet the new challenge. New approaches such as neural networks, simulated annealing, stochastic machines, mean field annealing, and genetic algorithms, have been proven to be effective in solving complex optimization problems in recent years. Readers are referred to the recently published text [1] for a solid foundation on computational intelligence for optimization.

In this chapter, a computational intelligence approach based on Mean Field Annealing (MFA) is presented to solve the channel assignment problem in mobile communication networks. A communication network is expected to serve an increasing number of users for data transmissions, information exchanges, and various multimedia applications. Bandwidth is often a primary and invaluable resource. In order to support concurrent access by numerous users in a network, this finite and expensive resource must be shared among many independent contending users. Multiaccess protocols control the access of resources among users to achieve their efficient utilization, to satisfy connectivity requirements, and to resolve any conflict among the contending users. Frequency-division multiple access (FDMA) is a widely used protocol in current mobile telephone communication systems, in which the available bandwidth is subdivided into a number of subbands called *channels*. Three interferences, namely *co-channel*, *adjacent*, and *co-site*, must be avoided in channel assignment while maximizing the number of communication connections. Such a channel assignment problem has proved to be NP-complete, for which efficient algorithms rarely exist. As the problem size gets large, there may exist a large number of local optima, and searching for the global optimal solution by using conventional heuristic algorithms may become intractable. On the other hand, Simulated Annealing (SA) provides a means to search for global solutions. However, SA is often time consuming. MFA replaces the stochastic process in SA by using a set of deterministic equations, which can usually find suboptimal solutions much faster than its counterpart. In Section 2, the channel assignment problem is described. The energy function required by MFA is formulated in Section 3. In Section 4, the determination of the frequency span and convergence of the algorithm are presented. In Section 5, remarks are drawn based on numerical results of three test instances. In Appendix A, the MFA theory and its convergence are discussed. The NP-completeness of the channel assignment problem is proved in Appendix B.

2. The Channel Assignment Problem

In the FDMA protocol, each user in the network is confined to access an allocated channel only. Although FDMA consumes a fraction of the bandwidth to achieve adequate frequency separation, it is relatively easy to implement and requires no real-time coordination. One of the FDMA examples is cellular communication systems, in which the frequency band is allocated by the Federal Communications Commission (FCC) to be on 824-849 MHz for uplink transmissions (from a mobile to a base station) and on 869-894 MHz for downlink transmissions (from a base station to a mobile). The frequency band is subdivided into a certain number of narrowband channels, each capable of supporting one phone circuit that can be accessed by any user. The channels are indexed by a sequence of numbers $\{1, 2,..., N\}$. Channel i and Channel $i+1$ are called *adjacent*. The channel bandwidth is 30 kHz. Therefore, this frequency band can accommodate 832 duplex channels [2, 3]. Furthermore, the 832 duplex channels are equally divided into Bands *A* and *B*. Voice and control channels are assigned at each base station from the allocated spectrum, either *A* or *B* band. Therefore, there are 416 channels for each band, including 21 control channels for call setup. Thus, each band has 395 available traffic channels for voice transmissions. Figure 1 shows a cellular communication system in which a geographical area is divided into hexagonal *cells*. The number of cells K in which the same channel cannot be used is called the *frequency reuse factor*. Using different channels within K cells prevents adjacent cells from interfering with one another. Starting from the center of a cell, say Cell *a*, one can reach another cell's center by traversing through i cells, rotating 45°, then traversing j cells, and finally reaching the other cell *a* that can reuse the same channel, as shown in Figure 1. A frequency pattern, or cluster, composed of K cells is determined by the equation $K = i^2 ij + j^2$, where $i, j \in I^+$, and I^+ is the set of positive integers. The cluster shown by the shaded cells in 1 corresponds to $i = 1$ and $j = 2$, or $K = 7$.

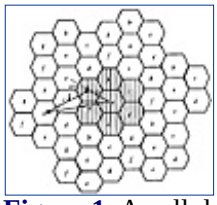


Figure 1 A cellular network with $K = 7$.

To minimize interference, each cell can only use $395/K$ channels for each band. For example, for $K = 7$ as shown in Figure 1, each cell can support 56 traffic channels. As the demand for communications increases, hundreds of channels may be required to serve thousands of concurrent users. To meet this requirement, channels must be reused in an efficient manner. In a cellular mobile-telephone system, the same channel used in one cell might be reused in another cell provided that the two cells are separated by a certain distance in space. Interference may occur when the same cell or different cells use certain pairs of channels. In order to avoid any interference, three types of interference constraints, namely *co-channel*, *adjacent-channel*, and

co-site, must be satisfied. The constraints will be addressed in the next section. The channel assignment task, given a group of available channels, is to find an assignment that meets various constraints [4].

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

3. Problem Formulation

For an *n*-cell cellular radio network, the cell system is denoted by $X = \{x_1, x_2, ..., x_n\}$, where x_i, i, represents a cell. The requirement on X is an *n*-vector $R = (r_i)$ where r_i is the number of required channels by cell x_i. The interference constraints can be specified by an $n \times n$ non-negative matrix $C = [c_{ij}]$. C is called a *compatibility matrix* on X. For a given problem, C is prespecified by the designer. For any channel f assigned to Cell x_i and f 2 assigned to Cell x_j, it must satisfy the condition $|f - f2|$ e c_{ij}. Based on the value of c_{ij}, three types of constraints are defined:

1. *Co-channel constraint*: If $c_{ij} = 1$, then $|f - f2|$ e 1 must hold, implying that Cells x_i and x_j cannot use the same channel.
2. *Adjacent-channel constraint*: Two cells x_i and x_j are prohibited from using adjacent channels; i.e., any two channels assigned to Cells x_i and x_j must be separated by at least two channels. It is reflected by the inequality c_{ij} e 2.
3. *Co-site channel constraint*: Two channels assigned to a cell x_i must be separated by a certain number of channels, i.e., c_{ij} e l where l e 2.

Define an *n*-vector $F = (F_i)$, where F_i is a bundle of channels assigned to cell x_i and the cardinality of F_i is the number of channels assigned to cell x_i. A triple $_c = (X, R, C)$ is called an instance of the channel assignment problem. F will be called a *feasible channel assignment* for $_c$ if the following conditions are satisfied.

1. The cardinality of F_i equals r_i, i
2. $|f - f2|$ e c_{ij}, for all x_i, x_j X, f F_i, $f2$ F_j

If *m* successive channels are assigned to cells in an assignment without causing interference, *m* is called the span of the assignment. For a given channel assignment problem, the objective is to find a feasible channel assignment F that has the minimum span while satisfying the interference constraints and user requests.

In the simplest form of the channel assignment problem where only co-channel constraint is considered, it is shown in Appendix B that the problem is equivalent to a graph coloring problem, and therefore it is an NP-complete optimization problem. The extension of the constraints will show that the channel assignment is NP-complete. The proof of its NP-completeness is given in Appendix B.

To map the problem onto a Hopfield-type neural network [5], define

$$s_{ij} = \begin{cases} 1 & \text{if channel } j \text{ is assigned to cell } xi \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

As shown in Appendix A, an energy function must be formulated in order to solve an optimization problem by MFA. The energy function should reflect the parameters to be optimized and various constraints. Then, the MFA procedure is simply to find the optimal solutions which yield the minimum energy.

For a channel assignment problem $_c = (X, R, C)$ with n cells and m channels (to be decided), $S = [s_{ij}]$ is an $n \times m$ binary matrix representing a neural network. The steady state of the neural network S corresponds to an assignment. The energy function is composed of two parts, E_1 and E_2. E_1 reflects the interference constraints as given above:

$$E_1 = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{p=1}^{m} \sum_{q:\ |q-p|< c_{ij}} c_{ij} s_{ip} s_{jq}$$

When a channel q is assigned to Cell i, any channel p assigned to Cell j must satisfy $|q-p| \geq c_{ij}$ to be considered as a feasible assignment, which is equivalent to

$$\sum_{q:\ |q-p|< c_{ij}} c_{ij} s_{ip} s_{jq} = 0$$

If all channels and cells are taken into account, a feasible assignment will result in

$$E_1 = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{p=1}^{m} \sum_{q:\ |q-p|< c_{ij}} c_{ij} s_{ip} s_{jq} = 0$$

In other words, any invalid channel assignment will yield a non-zero energy value.

The second part of the energy function is E_2, which reflects the channel requirement constraint

$$E_2 = \sum_{i=1}^{n}\left(\sum_{j=1}^{m} s_{ij} - r_i\right)^2$$

For a channel assignment, E_2 is minimized only when the number of channels assigned to any cell x_i i is equal to the number of required channels r_i, e.g., $\left(\sum_{j=1}^{n} s_{ij} - r_i\right) = 0$.

The energy function E can be expressed as

$$E(S) = \frac{w_1}{2} E_1 + \frac{w_2}{2} E_2 \quad (2)$$

where $w_1, w_2 > 0$ are the weights.

Combining E_1 and E_2, it can be seen that for a feasible channel assignment, Equation (2) yields the minimum energy value, zero. Any channel assignment that violates the interference constraints and channel requirement will yield higher energy. The MFA procedure is to find a feasible channel assignment which provides the minimum energy. If a function $g(x)$ is defined as

$$g(x)=\begin{cases}1 & \text{if } x<1\\ x & \text{if } 1\le x\le m\\ m & \text{if } x>m\end{cases} \tag{3}$$

then the energy function can be written as

$$E(S)=E_1+E_2=\frac{w_1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{p=1}^{m}\sum_{q=g(p-c_{ij}+1)}^{g(p+c_{ij}-1)}c_{ij}s_{ip}s_{jq}+\frac{w_2}{2}\sum_{i=1}^{n}\left(\sum_{j=1}^{m}s_{ij}-r_i\right)^2 \tag{4}$$

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

4. Convergence and Determination of the Frequency Span

For a given compatibility matrix C and the channel requirement vector R, one needs to determine the minimum frequency span (the number of required successive channels), denoted by m. However, as shown in Appendix B, the determination of the minimum span is polynomially related to the graph coloring problem which is an NP-complete problem. Therefore, the problem itself is NP-complete. In order to apply the MFA scheme to solve the channel problem, the minimum span is approximated for the following two cases. Then, the MFA algorithm is used to search for the feasible channel assignment. If no satisfactory assignments can be found, the span m is incremented by one or more, depending on how far the assignments are from the channel requirement R.

4.1 Case 1

Usually, c_{ii}, the minimum frequency separation for any channels assigned to Cell i (i), is larger than c_{ij}, the frequency separation for channels assigned to any two different cells i, j. If $c_{ii} >> c_{ij}$, m can be obtained as follows.

$$m = c_{ii} \cdot (r_i - 1) + 1,$$

$$\text{where} \quad i = \{k: r_k = \max_{\forall j \neq k} r_j\}$$

$$+ \frac{w_2}{2} \sum_{i=1}^{n} \left[\sum_{p=1}^{n} s_{ip}^2 + 2 \sum_{p=1}^{m-1} \sum_{q=p+1}^{m} s_{ip} s_{iq} - 2 \sum_{p=1}^{m} s_{ip} r_i + r_i^2 \right]$$

$$= \frac{w_1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{p=1}^{m} \sum_{q=1}^{m} c_{ij} s_{ip} s_{jq} \left(\prod_{l=\overline{Z}_{-}[g(p+c_{ij}-1),\, g(p+c_{ij}-1)]} (1 - \delta_{ql}) \right) +$$

$$+\frac{w_2}{2}\sum_{i=1}^{n}\left[\sum_{j=1}^{n}\sum_{p=1}^{m}\sum_{q=1}^{m}s_{ip}s_{jq}\delta_{ji}\delta_{qp}-2\sum_{p=1}^{m}s_{ip}\cdot r_i+r_i^2\right]+$$

$$+w_2\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{p=1}^{m}\sum_{q=1}^{m}s_{ip}s_{jq}\delta_{ji}\ (1-\delta_{pm})\prod_{l\in Z_m(1,p)}(1-\delta_{ql})$$

$$=\frac{w_1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{p=1}^{m}\sum_{q=1}^{m}c_{ij}s_{ip}s_{jq}\left(\prod_{l=\overline{Z}_m[g(p+c_{ij}-1),\ g(p+c_{ij}-1)]}(1-\delta_{ql})\right)+$$

$$+\frac{w_2}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{p=1}^{m}\sum_{q=1}^{m}\left\{s_{ip}s_{jq}\delta_{ji}\delta_{qp}+2s_{ip}s_{jq}\delta_{ji}\ (1-\delta_{pm})\prod_{l\in Z_m(1,p)}(1-\delta_{ql})\right\}$$

$$-w_2\sum_{i=1}^{n}\sum_{p=1}^{m}s_{ip}\cdot r_i\ +\ \frac{w_2}{2}\sum_{i=1}^{n}r_i^2$$

$$=-\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{p=1}^{m}\sum_{q=1}^{m}T_{ip,jq}\,s_{ip}s_{jq}-\sum_{i=1}^{n}\sum_{p=1}^{m}I_{ip}\cdot s_{ip}+\frac{w_2}{2}\|R\|^2$$

where

$$T_{ip,jq}=-\frac{w_1}{2}c_{ij}\left(\prod_{l=\overline{Z}_m[g(p-c[j+1]),g(p+c[j-1])]}(1-\delta_{ql})\right)-\frac{w_2}{2}\delta_{ji}\delta_{qp}-w_2\delta_{ji}(1-\delta_{pm})\left(\prod_{l=Z_m(1,p)}(1-\delta_{ql})\right)$$

$$I_{ip}=w_2 r_i$$

$$\|R\|^2=\sum_{i=1}^{n}r_i^2$$

4.2 Case 2

If c_{ii} is close to c_{ij} for all *i, j*, the procedure described for Case 1 may not be applicable. The following heuristic algorithm is used to determine the minimum span *m*. It can provide a tighter span only when $c_{ii} = 2$ and $c_{ij} = 0$ or 1. Let $X = (x_i)$ be the set of cells. Denote A as the set of assigned cells which have been incorporated in evaluating *m*, and U as the set of the unassigned cells. The notation $A \leftarrow x_p$ indicates that cell x_p is assigned to Set A.

1. $m = c_{pp}\,(r_p\text{-}1)+1$, $A \leftarrow x_p\ U \leftarrow X - \{x_p\}$ where *p* satisfies the inequality $c_{pp}(r_p\text{-}1)$ e $C_{ii}\,(r_i\text{-}1)$ *i*; *y* • 0 (if *y = 0, m* remains unchanged in the next step; if *y = 1, m* will be incremented by 1).

2. Select $x_q \in U$ such that $c_{qq}(r_q\text{-}1) \geq c_{ii}(r_i\text{-}1)\ \forall\ x_i \in U$.

If $c_{qi} = 1\ \forall x_i \in A$ and *y* = 1, then $m = m + c_{qq}\,(r_q - 1)$, *y*•*0*.

If $c_{qi} = 1\ \forall\ x_i \in A$ and *y* = 0, then *y*• 1.

3. $A \leftarrow A + \{x_q\}, U \leftarrow U - \{x_q\}$, repeat Step 2 until U is empty.

As demonstrated by simulations, the above approximation procedures provide satisfactory results. The determination of the exact required frequency span is still an open question.

4.3 Convergence of the MFA Channel Assignment Algorithm

Define

$$Z_m(a,b) = \{l:\ l,a,b \in Z^+,\ a \le l \le b\}$$

$$\overline{Z}_m(a,b) = \{l:\ l \in Z_m(1,m),\ \text{and}\ l \notin Z_m(a,b)\} \qquad (5)$$

Then,

$$E(S) = \frac{w_1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{p=1}^{m}\sum_{q=g(p-c_{ij}+1)}^{q=g(p+c_{ij}-1)} c_{ij}\,s_{ip}\,s_{jq} + \frac{w_2}{2}\sum_{i=1}^{n}\left(\sum_{p=1}^{m} s_{ip} - r_i\right)^2$$

$$= \frac{w_1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{p=1}^{m}\sum_{q=g(p-c_{ij}+1)}^{q=g(p+c_{ij}-1)} c_{ij}\,s_{ip}\,s_{jq} + \frac{w_2}{2}\sum_{i=1}^{n}\left[\left(\sum_{p=1}^{m} s_{ip}\right)^2 - 2\sum_{p=1}^{m} s_{ip}\,r_i + r_i^2\right]$$

$$= \frac{w_1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{p=1}^{m}\sum_{q=1}^{m} c_{ij}\,s_{ip}\,s_{jq}\left(\prod_{l \in \overline{Z}_m[g(p-c_{ij}+1),\,g(p+c_{ij}-1)]}(1-\delta_{ql})\right) +$$

Since $\frac{w_2}{2}\|R\|^2$ is a constant for a given w_2 and does not affect the iteration procedure, this term can be ignored, and the energy function can be written in the form of

$$E(S) = -\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{p=1}^{m}\sum_{q=1}^{m} T_{ip,jq}\,s_{ip}\,s_{jq} - \sum_{i=1}^{n}\sum_{p=1}^{m} I_{ip}\,s_{ip} \qquad (6)$$

Note that

$$T_{ip,jq} = T_{jq,ip} \qquad (7)$$

In Equation A.4, it is shown that, if the energy function can be written in a form of the Hopfield energy function, the MFA iterations will guarantee stability, and therefore convergence at each temperature. Since Equation (6) is expressed in this form and the matrix $[T_{ip,jq}]$ is symmetric, the MFA approach to solve the channel assignment problem will converge to the local minima at each temperature.

By taking the derivative of the energy,

$$\frac{\partial E(S)}{\partial s_{ip}} = w_1\sum_{j=1}^{n}\sum_{q=g(p-c_{ij}+1)}^{g(p+c_{ij}-1)} c_{ij}\,s_{jq} + w_2\left(\sum_{q=1}^{m} s_{iq} - r_i\right) \qquad (8)$$

Therefore, the MFA iterations can be performed according to

$$v_{ip}^{(k)} = \frac{1}{2} + \frac{1}{2}\tanh\left\{-\frac{1}{2T}\left[w_1\sum_{j=1}^{n}\sum_{q=g(p-c_{ij}+1)}^{q=g(p+c_{ij}-1)} c_{ij}\,v_{jq}^{(k-1)} + w_2\sum_{q=1}^{m}\left(v_{iq}^{(k-1)} - r_i\right)\right]\right\} \qquad (9)$$

5. Numerical Examples and Conclusions

Three instances with 5, 10 and 25 cells have been tested by the proposed MFA channel assignment algorithm. Figure 2 lists the compatibility matrix and the requirement vectors for each of the instances. The assignments obtained by the proposed algorithm are shown in Figure 3. In Figures 3a and 3b, the frequency spans *m* are estimated using the procedure outlined for Case 1. They are 67 and 204, respectively, and are equal to the spans of the actual assignments. In Figure 3c, *m* estimated by the procedure outlined for Case 2 is 69, which is rather close to the actual required span of the assignment, 73. Although the determination of *m* is NP-complete, the proposed procedures provide satisfactory results in our simulations.

As pointed out in Section 3, each binary neuron s_{ij} represents an assignment, e.g., if $s_{ij} = 1$, then channel *j* is

assigned to cell *i*. The MFA procedure is iterated according to Equation (9), starting at temperature $T_0 = 5$. The weights w_1, w_2 are set to 15 and 20, respectively. The annealing schedule, in which the temperature is lowered, $T_k = 0.98T_{k-1}$, is used, where k is the k-th iteration. The annealing process terminates and the decoding $\boldsymbol{v} = [v_{ij}]$ provides the channel assignment solutions when the following condition is met:

$$\frac{1}{nm}\left(\sum_i\sum_j v_{ij}^{(k)} - \sum_i\sum_j v_{ij}^{(k-1)}\right)^2 \le 0.01 \qquad (10)$$

where $v_{ij}^{(k)}$ is the value of v_{ij} at equilibrium at temperature T_k, n is the number of cells, and m is the frequency span.

In summary, the channel assignment problem arises when scarce and expensive spectral resources must be fully utilized. It is proved that the assignment problem is NP-complete. In this chapter, an MFA-based algorithm is proposed to solve the difficult optimization problem. Three interference constraints, namely co-cell, adjacent, and co-site, are considered. The energy function can be written in the form of Hopfield net, and subsequently the convergence of the proposed MFA iteration is guaranteed. Two procedures are presented to estimate the minimum span, and they provide satisfactory results. Three instances are tested by the algorithm, and feasible solutions are obtained for each of the instances.

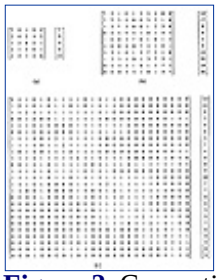


Figure 2 Compatibility matrix and requirement matrix: (a) C1 and R1; (b) C2 and R2; (c) C3 and R3.

Figure 3 Channel assignments: (a) F1; (b) F2; (c) F3.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Appendix A: Mean Field Annealing

In a combinatorial optimization problem, cost is defined as a function of discrete variables representing configurations. A combinatorial optimization problem is defined by $\Pi = (f, S)$ where $S = \{s\}$ is a finite set of configurations and f is the cost function, $s \in I^n$ and $f: s \rightarrow R^+$; here I^n represents the n-dimensional integer space and R^+ the positive real value space. The objective is to find an optimal configuration s_{opt} which provides the minimum cost, i.e.,

$$f_{opt} = f(s_{opt}) = \min_{s \in S} f(s) \qquad \text{(A1)}$$

A.1 Statistical Mechanics

There exists a significant analogy between statistical mechanics and the process in solving complicated combinatorial optimization problems. Statistical mechanics examines the properties of ensemble of particles. Since the number of particles is quite large per cubic centimeter, only the behavior of the system in thermal equilibrium at a given temperature is observable. Different position placements of particles in a liquid or solid matter will yield different energies. At each temperature, all particles randomly move around until thermal equilibrium is reached. If a state is defined by the set of particle positions, then, at thermal equilibrium, the probability of the system being in state i is represented by the *Gibbs distribution* [6, 7]:

$$\pi_i = \Pr(s = i) = \frac{\exp\left(-\frac{E(i)}{k_b T}\right)}{Z} \qquad \text{(A2)}$$

where $Z = \sum_{j \in S} \exp\left(-\frac{E(j)}{k_b T}\right)$ is called the partition function, k_b is the Boltzmann constant, T the absolute temperature, $E(i)$ the energy of state i, S the state space (corresponding to the

configuration space in an optimization problem), where k_b, T, $E(i) \in R^+$. It is easy to show that [6]

$$\lim_{T\to\infty} \pi_i = \lim_{T\to\infty} \frac{\exp\left(-\frac{E(i)}{k_b T}\right)}{\sum_{j\in S} \exp\left(-\frac{E(j)}{k_b T}\right)} = \frac{1}{|S|} \qquad \text{(A3)}$$

implying that, at a very high temperature, all states are equally probable.

On the other hand,

$$\lim_{T\to 0} \pi_i = \lim_{T\to 0} \frac{\exp\left(-\frac{E(i)-E_{\min}}{k_b T}\right)}{\sum_{j\in S} \exp\left(-\frac{E(j)-E_{\min}}{k_b T}\right)}$$

$$= \lim_{T\to 0} \frac{\exp\left(-\frac{E(i)-E_{\min}}{k_b T}\right)}{\sum_{j\in S_{\min}} \exp\left(-\frac{E(j)-E_{\min}}{k_b T}\right) + \sum_{j\notin S_{\min}} \exp\left(-\frac{E(j)-E_{\min}}{k_b T}\right)}$$

$$= \begin{cases} \frac{1}{|S_{\min}|} & \text{if } i \in S_{\min} \\ 0 & \text{otherwise} \end{cases} \qquad \text{(A4)}$$

where $S_{\min} = \{j: E(j) = E_{\min}\}$ and $E_{\min} = \min_{j\in S} E(j)$.

As can be seen from this equation, the system will likely settle in those states with minimum local energy as the temperature approaches zero.

The crystalline lattice structure in a solid usually yields lower energy. A physical process called *annealing* is often performed in order to form a crystal. In the annealing process, a solid is heated up in a *heat bath* by increasing the temperature of the heat bath until it has melted into liquid, and then the temperature is slowly lowered. At each temperature, all particles randomly arrange themselves until thermal equilibrium is reached. If the annealing is slow enough to allow the solid to reach thermal equilibrium at each temperature, a low-energy crystalline solid would be formed when the system is frozen *(T '0)*. If the annealing is too fast, the solid may become glass-like with noncrystalline structure or consist of defective crystals with metastable amorphous structures.

A.2 Simulated Annealing

Based on the annealing process in statistical mechanics, Kirkpatrick et al. [8] proposed an algorithm, namely, *simulated annealing* (SA), for solving complicated combinatorial optimization problems. The cost function and configuration in optimization correspond, respectively, to the energy function and state in statistical physics. Therefore, state and configuration are used interchangeably. The temperature is introduced as a control parameter.

Suppose that a cost function $f: S \to R^+$, $S \in S$,is defined on some finite configuration set S. For each configuration $S \in S$, there is a neighboring set $N(s) \subseteq S$, which is generated by a small perturbation of *s*.

Given the current state *s(k)*, a neighboring state *s'(k)* is randomly selected from $N(s)$, where *k* is the *k*-th trial. The transition probability from state *s(k)* to *s'(k)* is given by *Metropolis*-[9]:

$$P[s(k), s'(k)] = \Pr\{s(k) \to s'(k)\} = \exp\left[-\frac{[f(s'(k)) - f(s(k))]^+}{T}\right] \qquad \text{(A5)}$$

where

$$[x]^+ = \max\{0, x\} \tag{A6}$$

Equation (A5) can be written in another form:

$$\Pr\{s(k+1) = s'(k)\} = \begin{cases} 1 & \text{if } f(s'(k)) < f(s(k)), \\ \exp\left(-\dfrac{f(s'(k)) - f(s(k))}{T}\right) & \text{otherwise.} \end{cases} \tag{A7}$$

Equation (A7) allows occasional transitions from a lower cost configuration to a higher cost configuration with certain probability, thus preventing the system from getting stuck in local minima. The random process $X = (s(k): k \geq 0)$ produced thus can be characterized by a discrete time homogeneous-Markov chain [6]. The one-step transition matrix is

$$\Pr(x, y) = \Pr[s(k+1) = y \mid s(k) = x]$$

$$= \begin{cases} 0 & \text{if } y \notin N(x) \text{ and } y \neq x \\ G(x,y) \min\left\{1, \exp\left(-\dfrac{[f(y) - f(x)]}{T}\right)\right\} & \text{if } y \in N(x) \text{ and } y \neq x \\ 1 - \sum_{x' \neq x} G(x,x') \min\left\{1, \exp\left(-\dfrac{[f(x') - f(x)]}{T}\right)\right\} & \text{if } y = x \end{cases}$$

where $G(x,y)$ is the probability of generating configuration y from x.

If the generation probability of any configuration x is uniformly distributed its neighboring configuration set $N(x)$ and the configuration transition is based on the above equation, the corresponding Markov chains are irreducible, aperiodic, and recurrent [6]. Under these conditions, the stationary equilibrium distribution $\grave{A}_i$ for configuration i is reached after an infinite number of transitions:

$$\begin{aligned} \pi_i(T) &= \lim_{k \to \infty} \Pr\{s(k) = i \mid T\} \\ &= \lim_{k \to \infty} \Pr\{s(k) = i \mid s(0) = s_0, T\} \\ &= \frac{\exp\left(-\dfrac{f(i)}{T}\right)}{\sum_{j \in S} \exp\left(-\dfrac{f(j)}{T}\right)} \end{aligned} \tag{A8}$$

From Equation (A4),

$$\pi_i^* = \lim_{T \to 0} \pi_i(T) = \begin{cases} \dfrac{1}{|S_{\min}|} & \text{if } i \in S_{\min} \\ 0 & \text{otherwise} \end{cases} \tag{A9}$$

Therefore,

$$\lim_{T \to 0}[\lim_{k \to \infty} P(s(k) \in S_{\min})] = \lim_{T \to 0} \sum_{i \in S} \pi_i(T) = \sum_{i \in S_{\min}} \pi_i^* = 1 \tag{A10}$$

Equation (A10) states that the SA algorithm asymptotically converges to the configurations with the minimum cost, i.e., if the temperature is slowly lowered and the system performs a sufficient number of

transitions at each temperature, the configurations (solutions) with the global minimum cost will be found with probability one.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

A.3 Mean Field Annealing

Even though SA is proven to be able to reach the global optima asymptotically, it is often time-consuming to reach thermal equilibrium at each temperature. A finite number of transitions at each temperature cannot guarantee convergence to the global optima. In statistical physics, *mean field* approximation is often used. Mean field annealing (MFA) uses a set of deterministic equations to replace the stochastic process in SA. It uses saddle point approximation in the calculation of the stationary probability distribution at equilibrium, and reaches equilibrium at each temperature much faster than SA. Even though this approximation method may not be guaranteed to converge to global minima, it does provide a good approximation in finding near-optimal solutions with much less computing effort.

As shown in the previous section, the stationary probability distribution at equilibrium for configuration ***s'*** is given by

$$\pi_{s'} = \frac{\exp\left(-\frac{f(s')}{T}\right)}{Z}$$

$$Z = \sum_{s} \exp\left(-\frac{f(s)}{T}\right)$$

where s, $s' \in I^n$ are configurations and I^n is the n-dimensional integer space. For a large optimization problem, exact calculation of the partition function Z is prohibitive. The saddle point approximation [10] is used. Note that the Dirac delta function, ´(.), can be expressed as

$$\delta(x) = \frac{1}{2\pi i}\int_I e^{xy}dy \qquad \text{(A11)}$$

where the integral is taken along the imaginary axis. Hence,

$$Z = \sum_s exp\left(-\frac{f(s)}{T}\right) = C_0 \int_R dv \int_I e^{-f_e(u,v)}\, du \qquad \text{(A12)}$$

where

$$f_e(u,v) = \frac{f(v)}{T} + uv - \ln \sum_s e^{us} \qquad \text{(A13)}$$

C_0 is a complex constant, and f_e is called the *effective energy* in statistical mechanics. At saddle points,

$$\frac{\partial f_e}{\partial u} = v - \frac{\sum_s s \cdot e^{u \cdot s}}{\sum_s e^{u \cdot s}} = 0$$

and

$$\frac{\partial f_e}{\partial v} = \frac{1}{T}\frac{\partial f(v)}{\partial v} + u = 0 \qquad \text{(A14)}$$

Therefore,

$$v = \bar{s}_T = \frac{\sum s \cdot e^{us}}{\sum e^{us}} \qquad \text{(A15)}$$

$$u = -\frac{1}{T}\frac{\partial f(v)}{\partial v}$$

where $\bar{s}_T$ is the thermal average of $\boldsymbol{s}$ at temperature T.

In statistical physics, $h = -\frac{\partial f(v)}{\partial v}$ is called the *mean field*. If a configuration $\boldsymbol{s}=[s_1, s_2, ..., s_n]^T$ is represented by a sequence of binary values, i.e., $\boldsymbol{s} \in \{0, 1\}^n$, then $\boldsymbol{v}=[v_1, v_2, ..., v_n]^T$ and

$$v_i = \frac{\sum_{s_i=0}^{1} s_i \cdot e^{u_i s_i}}{\sum_{s_i=0}^{1} e^{u_i s_i}} = \frac{e^{u_i}}{1+e^{u_i}} = \frac{1}{2}\left[1 + \tanh\left(\frac{u_i}{2}\right)\right] \qquad \text{(A16)}$$

where $\boldsymbol{u} = [u, u_2, ..., u_n]^T$ and $h = -\frac{\partial f(v)}{\partial v_i}$

For the binary system,

$$v_i = \frac{1}{2}\left[1 + \tanh\left(\frac{h_i}{2T}\right)\right] \qquad \text{(A17)}$$

$$h_i = -\frac{\partial f(v)}{\partial v_i} \qquad \text{(A18)}$$

In 1982, Hopfield [5] defined the following energy function of the Hopfield net for optimization:

$$f_h(s) = -\frac{1}{2}\sum_i \sum_j T_{ij} s_i s_j - \sum_i s_i I_i \qquad \text{(A19)}$$

where s_i {*0, 1*}. In the Hopfield model, the system is represented by a network composed of *n* neurons. Each neuron *i* can be represented by an operational amplifier, s_i is the output of neuron *i*, and T_{ij}, which is symmetric ($T_{ij}=T_{ji}$ and $T_{ii}=0$), represents the synaptic connection between neuron *i* and *j*. I_i is the input current to amplifier *i*. The stable states of the network correspond to the 2^n corners of the hypercube $\{0, 1\}^n$, the local minima of the energy function defined in Equation (A19). For the MFA approximation, if the energy is formulated as in Equation (A19), the mean field h_i and the thermal average v_i become

$$h_i = -\frac{\partial f_h(v)}{\partial v_i} = \sum_j T_{ij} v_j + I_i \qquad \text{(A20)}$$

$$v_i = \bar{s}_i = \frac{1}{2}\left[1 + \tanh\left(\frac{h_i}{2T}\right)\right] \qquad \text{(A21)}$$

The iterative procedure to reach thermal equilibrium at each temperature is called *relaxation*, in which the mean field is updated by

$$h_i(t+\Delta t) = h_i(t) + \Delta t\left[-\frac{\partial f_h(v)}{\partial v_i} - h_i(t)\right]$$

Taking the limit,

$$\frac{dh_i}{dt} = \lim_{\Delta t \to 0} \frac{h_i(t+\Delta t) - h_i(t)}{\Delta t} \qquad \text{(A22)}$$

or

$$\frac{dh_i}{dt} = -\frac{\partial f_h(v)}{\partial v_i} - h_i(t) = \sum_j T_{ij} v_j + I_i - h_i \qquad \text{(A23)}$$

The MFA relaxation operation at each temperature should lead the system to stable equilibrium. The stability and convergence of MFA will be analyzed in the next section. The MFA procedure can be summarized in the flow chart shown in Figure Al. In this figure, a *sweep* represents a complete update of all neurons, and an *iteration* consists of L_k sweeps at temperature T_k.

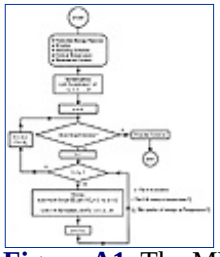


Figure A1 The MFA iteration procedure.

A.4 Convergence of MFA

To prove convergence of MFA requires some knowledge of stability theory [11]. Consider the following differential equation:

$$\frac{dx}{dt} = f(x) \qquad \text{(A24)}$$

where $x \in R^n$, $f: x \to R^{+n}$. R^{+n} represents the *n*-dimensional positive real value space.

Definition A1 (*Equilibrium Point): x* is called an equilibrium point of Equation (A24) if f(x*) = 0. That is, at*

the equilibrium point, the system will no longer change with time.

Definition A2 *(Stability): Let x*(t) be a solution to* $\dot{x} = f(x)$*. An equilibrium point x* is stable if, for every neighborhood set* N *of x*, there is a neighborhood* N_1 *of x* such that every solution x(t) with initial point*(t_0) *in* N_1 *defined in both* N_1 *and* N *for all* $t>t_0$*. That is, for any* $x_0 = x(t_0)$ *and any given value* $\mu > 0$*, there exists an arbitrarily small value* $\acute{} > 0$ *so that if* $\| x(t_0) - x^* \| < \delta$*, then* $\| x(t) - x^* \| < \varepsilon$*, where* $\|x\|$ *is the Euclidean norm, i.e.,*

$$\|x_i - x_j\| = \left[\sum_k (x_{ik} - x_{jk})^2\right]^{\frac{1}{2}}. \qquad \text{(A25)}$$

Definition A3 (*Asymptotically Stable): If x* is stable and* $\lim_{t'}$ *x(t) = x*, then x* is asymptotically stable.*

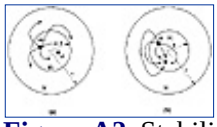


Figure A2 Stability: (a) stable; (b) asymptotically stable.

Figure A2 illustrates the stable and asymptotically stable points.

Theorem A1 (*Liapunov's Stability Theorem) [11]: Let x* be an equilibrium point for Equation (A24). Let E:* $N \rightarrow R$ *be a continuous and differentiable function defined on a neighborhood* N *of x* such that if:*

a) $E(x^*) = E_{min}$ *and* $E(x) > E(x^*)$ *if* $x \neq x^*$,

b) $\dot{E} \leq 0 \quad \forall x \in N$*, then x* is stable. Furthermore, if also,*

c) $\dot{E} < 0 \ \forall \ x \in N$*, x* is asymptotically stable, where:*

$$\dot{E}(x) = \frac{dE}{dt} = \sum_j \frac{\partial E}{\partial x_j} \cdot \frac{dx_j}{dt} = \nabla E^T \cdot \dot{x} \qquad \text{(A26)}$$

$$\nabla E(x) = \left[\frac{\partial E}{\partial x_1}, \frac{\partial E}{\partial x_2}, \cdots, \frac{\partial E}{\partial x_n}\right]^T \qquad \text{(A27)}$$

A function *E(x)* satisfying (a) and (b) is called a *Liapunov function*. If (c) also holds, *E(x)* is known as a strict Liapunov function.

For MFA, if we construct

$$E(v) = f_h + \sum_{i=1}^{n} \int_0^{v_i} h_i(y)dy \qquad \text{(A28)}$$

then

$$\frac{\partial E(\boldsymbol{v})}{\partial v_i} = \frac{\partial f_h(v)}{\partial v_i} + h_i = -\sum_j T_{ij} v_j - I_i + h_i \qquad \text{(A29)}$$

or

$$\nabla E(\boldsymbol{v}) = -\boldsymbol{T}\boldsymbol{v} - \boldsymbol{I} + \boldsymbol{h} \qquad \text{(A30)}$$

where $T = \{T_{ij}: \ _{ij}\}$ and $\boldsymbol{v}, \boldsymbol{I}, \boldsymbol{h} \in R^n$.

Let

$$\Psi(v) = E(v) - E(v^*) \qquad \text{(A31)}$$

and assume that v* is an equilibrium point and a local minimum; then,

a) ¨(**v***) = *E*(***v****) - *E*(***v****) = 0, and $\Psi(v) = E(v) - E(v^*) > 0, \quad \forall v \in N(v^*)$

b) From Equation (A17), we have

$$v_i = \frac{1}{2}\left[1 + \tanh\left(\frac{h_i}{2T}\right)\right]$$

Therefore,

$$\frac{\partial v_i}{\partial h_i} = \frac{\partial}{\partial h_i}\left(\frac{1}{1+e^{-h_i/T}}\right) = \frac{e^{h_i/T}}{T\left(1+e^{-h_i/T}\right)} > 0$$

$$\dot{\Psi}(v) = \sum_i \frac{\partial \Psi}{\partial v_i} \cdot \frac{dv_i}{dt}$$

$$= \sum_i \frac{\partial \Psi}{\partial v_i} \cdot \frac{\partial v_i}{\partial h_i} \cdot \frac{dh_i}{dt}$$

$$= \sum_i \left(\frac{\partial f_h}{\partial v_i} + h_i\right) \cdot \frac{\partial v_i}{\partial h_i} \cdot \left(-\frac{\partial f_h}{\partial v_i} - h_i\right)$$

$$= -\sum_i \frac{\partial v_i}{\partial h_i} \cdot \left(\frac{\partial f_h}{\partial v_i} + h_i\right)^2$$

$$= -\frac{e^{h_i/T}}{T\left(1+e^{-h_i/T}\right)^2}\left(\frac{\partial f_h}{\partial v_i} + h_i\right)^2$$

$$= -\nabla E^T \cdot W \cdot \nabla E < 0 \qquad \text{(A32)}$$

where $W = \text{diag}\left(\frac{\partial v_1}{\partial h_1}, \frac{\partial v_2}{\partial h_2}, \cdots, \frac{\partial v_n}{\partial h_n}\right)$. It is shown that if an equilibrium point ***v**** is a local minimum, it will be asymptotically stable. ¨(***v***); therefore *E*(***v***) is a strict Liaponov function; i.e., at each temperature, the evolution of Equation (A23) will lead the system to converging to a local minimum.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Appendix B: NP-Completeness of the Channel Assignment Problem

The channel assignment problem, $_C$, is to assign channels to cell sites while satisfying channel constraints described by the compatibility matrix $C = [c_{ij}]$. In the FDMA cellular communication systems, the frequency band is subdivided into a certain number of narrowband channels, each capable of supporting one phone circuit that can be accessed by any user. The channels are labeled as a sequence of numbers {*1, 2,..., N*}. Channel *i* and Channel *i+1* are called *adjacent*. The *frequency distance* between channel *f* and *f*2 is |*f* - *f*2|. c_{ij} represents the channel constraint that the frequency distance between the channels assigned to cell *i* and cell *j* must be greater than or equal to c_{ij}. Each diagonal element c_{ii} in *C* represents the minimum separation distance between any two channels assigned to cell *i*. Let vector $X = (x_1, x_2, ..., x_n)$ represent the *n* cell sites for a given cellular communication system. $R = (r_1, r_2,..., r_n)$ is the requirement vector, where r_i indicates that cell x_i is requesting for r_i, channels. $F = (F_1, F_2,..., F_n)$ is a feasible channel assignment, where $F_i = (f_{i1}, f_{i2},..., f_{i,ri})$, and f_{ik} *k* is the channel number assigned to cell x_i. For example, if three channels numbered 2, 5, 7 are assigned to cell x_i, and $r_i = 3$, then $F_i = (2, 5, 7)$. Thus, the problem $_c$ can be defined as a decision problem:

INSTANCE: $\Pi_C = [X, R, C, K]$ where K is a positive integer.

QUESTION: Is there a feasible channel assignment vector *F* such that the frequency span is K or less? Here, the *frequency span* is defined as the frequency distance between the largest channel number and the smallest channel number assigned to cells.

$_C$ is classified into two subcategories: *co-channel* and *co-site*. Those where only co-channel constraints occur are called co-channel cases. Those where all channel constraints - co-channel, adjacent channel and co-site occur are called co-site cases. The channel assignment problems for co-channel and co-site cases are denoted as $_{C1}$ and $_{C2}$, respectively.

Theorem B1 $_{C1}$ is NP-complete.

Proof. To prove $_{C1}$ is NP-complete, one first needs to show $_{C1}$ *NP*, and then show that $_{C1}$ is NP-hard by finding a polynomial transformation from a known NP-complete problem.

1. $_{C1}$ *NP*

For a given instance $_{C1}$ = [*X*, *R*, *C*], checking whether a given assignment is feasible can be completed in polynomial time. The checking procedure is outlined below:

For the assignment *F*, check whether the frequency distance | f_{ik} - f_{jl}| e c_{ij} for *i*, *j* = 1,2,..., *n*, *k* = *1,2*,..., r_i, and *l* = *1, 2*,..., r_j. Then check if the frequency span is K or less Obviously, the checking operations can be completed in polynomial time. Therefore, $_{C1}$ *NP*.

Let $_{I}$ denote the independent set problem. The independent set problem is, for a given graph, to find the maximum independent set for the graph. $_{I}$, is proved NP-complete in [12]. $_{C1}$ can be proven NP-hard by showing that $_{I\ p\ C1}$, where *a* $_{p}$ *b* stands for a polynomial transformation from *a* to *b*.

2. $_{I\ p\ C1}$

In the problem $_{C1}$, only co-channel constraints are considered. Therefore, c_{ii} = 1, c_{ij} {0,1} for all *i*, *j* (*i* ` *j*). A graph *G* = (*V*, *E*) can be used to represent $_{C1}$. Vertex *v* *V* represents a *requirement* and a link (*u*, *v*) connects pairs of requirements *u*, *v* that cannot be assigned the same channel; i.e., (*u*, *v*) *E* if c_{uv} = 1 and (*u*,*v*) E if c_{uv} = 0. For example, a given instance of $_{C1}$ is

$$X = \{x_1, x_2, x_3, x_4, x_5\}$$

$$C = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \qquad R = \begin{bmatrix} 2 \\ 1 \\ 1 \\ 2 \\ 1 \end{bmatrix}$$

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

The derived graph is shown in Figure B1, where i^j stands for the j-th requirement by cell x_i. In the derived graph G, every connected pair of vertices cannot be assigned the same channel. The channel assignment problem $_{C1}$ is to find the minimum number of channels which satisfy the co-channel constraints and the cells' requirements. By observation, $_{C1}$ is equivalent to the graph coloring problem. In a graph coloring problem, interconnected vertices may not have the same color. The objective is to find the minimum number of colors required to color all vertices. In $_{C1}$, the vertices represent requirements and the colors represent channels. Obviously, if one color is used to cover as many vertices as possible, the number of required colors will be minimum. Since the unconnected vertices can use the same color, the graph coloring problem is transformed into the independent set problem $_{I}$; i.e., finding the minimum number of colors is equivalent to the problem $_{I}$ by simply finding the maximum independent set for the graph. Then the independent set V´ and the links attached to the vertices in V´ are removed. For the reduced graph, repeat the search procedure for a new maximum independent set until the graph is reduced to *empty*. Due to the equivalent relationship between $_{C1}$ and $_{I}$, it is not difficult to find a polynomial transformation to map $_{I}$ to $_{C1}$, e.g., $_{I\ p\ C1}$.

Since $_{C1}$ NP and $_{I\ p\ C1}$ is NP-complete.

In the example shown in Figure B1, the graph can be decomposed into a bunch of subgraphs. In each sub-graph, the maximum independent set is found. The decomposition and coloring procedure are illustrated in Figure B2 (a)-(c). The shaded circles (cells) will be assigned to use the same channel. The feasible channel assignment is shown in Figure B2(d), where the shaded box represents the assigned cell.

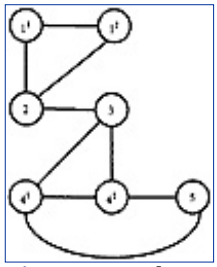


Figure B1 The construction of a graph.

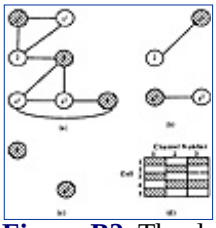


Figure B2 The decomposition of the graph (a)-(c), and (d) the channel assignment.

Theorem B2 $_{C2}$ is NP-complete.

Proof:

1. $_{C2}$ *NP*.

It can be shown that $_{C2}$ NP by performing the same checking procedure as in the proof of $_{C1}$ NP. Therefore, $_{C2}$ NP.

2. $_{C1}$ $_{p}$ $_{C2}$

Once $_{C1}$ is proven NP-complete, the NP-complete proof of $_{C2}$ is relatively easy. Since it has been shown $_{C2}$ NP, it is only necessary to prove that there is a transformation $_{C1}$ $_{p}$ $_{C2}$.

All of the constraints in $_{C2}$ can be described by the compatibility matrix $C = [c_{ij}]$ with c_{ij} $\boldsymbol{Z}^{+}$. $_{C1}$ is just the subset of $_{C2}$ with c_{ij} $\{0,1\}$. Therefore, every instance of $_{C1}$ can be directly mapped to an instance of $_{C2}$, i.e., if each instance I $_{C1}$, then I $_{C2}$. Therefore, $_{C1}$ $_{p}$ $_{C2}$.

Since $_{C1}$ is NP-complete, and $_{C2}$ NP and $_{C1}$ $_{p}$ $_{C2}$, it can be concluded that $_{C2}$ is NP-complete.

References

1 Ansari N. and Hou E. (1997), *Computational Intelligence for Optimization*, Kluwer Academic Publishers, Boston.

2 Donald H. V. (1979), The Cellular Concept, *Bell Syst. Tech. J*., Vol. 58, No. 1, pp. 15-41.

3 Lee W. C. Y. (1989), *Mobile Cellular Telecommunications Systems*, McGraw-Hill, New York.

4 Hale K. (1980), Frequency assignment: Theory and applications, *Proc. IEEE*, Vol. 68, pp. 1497-1514.

5 Hopfield J. and Tank D. W. (1982), Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Science*, Vol. 79, pp. 2541-2554.

6 Aarts E. and Korst J. (1989), *Simulated Annealing and Boltzmann Machine* - a *Stochastic Approach to Combinatorial Optimization and Neural Computing*, John Wiley & Sons, New York.

7 Wannier Ho. (1966), *Statistical Physics*, Ch. 4: John Wiley & Sons, New York.

8 Kirkpatrick (1983), Optimization by simulated annealing, *Science*, Vol. 220, pp. 671-680.

9 Metropolis, Rosenbluth A., Rosenbluth M., Teller A. and Teller E. (1953), Equation of state calculations by fast computing machines, *J. of Chem. Physics*, Vol. 21, pp. 1087-1092.

10 Peterson and Soderberg B. (1989), A new method for mapping optimization problems onto neural network, *International Journal of Neural Systems*, Vol. 1, No. 1, pp. 3-22.

11 Morris H. and Smale S. (1974), *Differential Equations, Dynamic Systems, and Linear Algebra*, Academic Press, Orlando, Florida.

12 Garey R. and Johnson D. S. (1979), *Computers and Intractability*: *A Guide to the Theory of NP-Completeness*, W.H. Freeman, New York.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Previous | Table of Contents | Next

Chapter 9
Application of Cellular Compact Neural Networks in Digital Communication

Bing J. Sheu, Michelle Y. Wang, Wayne C. Young
Dept. of Electrical Engineering
and Integrated Media Systems Center
University of Southern California
Los Angeles, CA 90089-0271
U.S.A.

In the digital communication field, neural networks can be applied as baseband maximum likelihood sequence detectors with high speed, low power and competitive detection bit error rate. Thus, it can replace some kinds of digital communication receivers. Two neural networks based detectors for hard-disk drive recording channel and code-division multiple access (CDMA) multi-user detection are presented. Performance comparison between neural network based detectors and digital counterparts are also proposed.

1. Introduction

With the rapid progress of multimedia technology, there is strong necessity for high speed, high capacity and high quality communication systems. In order to process the large amount of data through different channels in real-time and with high accuracy, the ability to overcome channel imperfectness is essential.

Neural networks is a very promising computational technology due to its capabilities in modeling and solving complex problems hardly approachable with traditional methods. In the communication field, neural networks also show a good potential for replacing certain kinds of digital communication receivers. The advantages of neural networks are the following:

- Competitive bit error rate (BER) of data detection
- Parallel computing architecture
- Low power consumption

- No need for analog-to-digital conversion (A/D) before detection as in conventional digital systems

The motivation of applying artificial neural networks or biologically inspired neural networks to the communication field also comes from the rapid improvement of the research on human brains. The research project, Chip in the Brain, which is working on replacing the impaired brain cells or a part of the brain with artificial neural network chips, is going on very well here at the Center of Neural Engineering, University of Southern California. We tried to implement the biologically inspired neural networks not only for the chip-to-brain communication purpose, but also for the real-world machine-to-machine communication applications. In this chapter, the basic theory of cellular/compact neural networks is first described. Then, the generic 1-D architecture for communication is presented. In Section 4, the implementations of 1-D compact neural network for detection of partial response (PR) signals and code division multiple access (CDMA) signals are described, and the performance comparisons between the neural network approaches and the conventional approaches are given.

2. Cellular/Compact Neural Networks

Many complex scientific problems can be formulated with a regular 1-D, 2-D, or 3-D grid. Direct interaction between the signals on various grid points is limited within a finite local neighborhood, which is sometimes called the receptive field. The original cellular neural network (CNN) paradigm was first proposed by Chua and Yang [1, 2] in 1988. Later, it was modified to be the compact neural network by Sheu and Choi [3] for the communication purpose in 1-D form. The two most fundamental ingredients of the CNN paradigm are the use of analog processing cells with continuous signal values, and local interaction within a finite radius. Many results on the algorithm development, VLSI implementations of the CNN systems are reported in the first three *IEEE International Workshops on Cellular Neural Networks and Their Applications* (Budapest, Hungary, 1990; Munich, Germany, 1992; Rome, Italy, 1994), the book entitled *Cellular Neural Networks*, which was edited by T. Roska and J. Vandewalle, and papers published in *IEEE Transactions on Circuits and Systems*, and other IEEE journals and conference proceedings.

Due to its regular structure and parallelism, a 10 times 10 mm^2 CNN microchip in a 0.5 ¼m CMOS technology can achieve the equivalence of more than 1 tera operations per second. The CNN architecture could be the paradigm that biologists have been seeking for many years [4]. It provides a unified model of many complex neural network architectures, especially for various forms of sensory modality.

The cellular neural networks can be viewed as *cellular nonlinear networks*. A CNN has many important features. It is a 2-, 3-, or n-dimensional *array of* mainly identical *dynamical cells*, which satisfies two properties:

- Most *interactions are local* within a finite radius *r*
- All state variables are continuous valued signals

A cloning template specifies the interaction between each cell and its neighboring cells in terms of their input, state, and output variables.

Each cell is identified by 2, 3, or *n* integers, (*i, j, k,..., n*). The time variable *t* may be continuous or discrete. The cloning template may be a linear or a nonlinear function of the state, input, and output variables of each cell. It could contain time-delay or time-varying coefficients. The dynamic systems may be perturbed by some noise sources of known statistics.

The heat equation, which is a typical partial differential equation, can be mapped onto a CNN as reported in [1]. If a capacitor is added at the output node, wave-type equations can also be processed by a CNN. At equilibrium, the Laplace equation can be effectively handled [3]. Hence, the CNN can be used to solve all three basic types of PDEs (partial differential equations): *the diffusion equation, the Laplace equation*, and *the wave equation*.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

2.1 Basic Theory and Computation Paradigm

A cellular/compact neural network is a continuous- or discrete-time artificial neural network that features a multidimensional array of neuron cells and local interconnections among the cells. The basic cellular neural network proposed by Chua and Yang [1, 2] in 1988 is a continuous-time network in the form of an *n*-by-*m* rectangular-grid array where n and m are the numbers of rows and columns, respectively. Each cell in a cellular neural network corresponds to an element of the array. However, the geometry of the array need not be rectangular and can be such shapes as triangles or hexagons [5]. A multiple of arrays can be cascaded with an appropriate interconnect structure to construct a multilayered cellular neural network. The *r*-th neighborhood $N_r(i,j)$ of a cell $C(i,j)$, $1 \le i \le n$, $1 \le j \le m$, is defined as a set of cells $C(k,l)$, $1 \le k \le n$, $1 \le l \le m$, for which $|k - i| \le r$ and $|l - j| \le r$. The cell $C(i,j)$ has direct interconnections with cells in $N_r(i,j)$ through two kinds of weights, the feedback weights $A(k,l;i,j)$ & $A(i,j;k,l)$ and the feedforward weights $B(k,l;i,j)$ & $B(i,j;k,l)$, where the index pair $(k,l;i,j)$ represents the direction of signal from $C(i,j)$ to $C(k,l)$. The cell $C(i,j)$ communicates directly with its neighborhood cells $C(k,l)$ $N_r(i,j)$. Since every $C(k,l)$ has its neighborhood cells, cell $C(i,j)$ can also communicate with all other cells indirectly. Figure 1(a) shows an *n*-by-*m* CNN with $r = 1$. The shaded cells represent the neighborhood $N_r(i,j)$ of $C(i,j)$, including $C(i,j)$ itself.

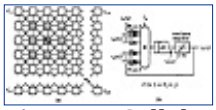


Figure 1 Cellular neural network (CNN). (a) An n-by-m cellular neural network on rectangular grid (shaded boxed are the neighborhood cells of C(*i*,*j*)). (b) Functional block diagram of neuron cell.

The block diagram of the cell $C(i,j)$ is shown in Figure 1(b). The external input to the cell is denoted by $v_{vuij}(t)$, and typically assumed to be constant, i.e., $v_{vuij}(t) = v_u(i,j)$ over an operation interval $0 \le t < T$. The input is connected to $N_r(i,j)$ through the feedforward weights $B(i,j;k,l)$s. The output of the cell, denoted by $v_y(i,j)$, is coupled to the neighborhood cells $C(k,l)$ $N_r(i,j)$ through the feedforward weights $A(i,j;k,l)$s. Therefore, the input signals is the weighted sum of feedforward inputs and weighted sum of feedback inputs. In addition, a constant bias term is added to the cell. If the weights represent the transconductances, the total input current i_{xij} to the cell is given by

$$i_{xij}(t) = \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l)v_{ykl}(t) + \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l)v_{ukl}(t) + I_b \qquad (1)$$

where I_b is the bias current. The equivalent circuit diagram of a cell is shown in Figure 2, where R_x and C are the equivalent resistance and capacitance of the cell, respectively. For simplicity, I_b, R_x, and C_x are assumed to be the same for all cells throughout the network. All inputs are represented by dependent current sources and summed together. Due to the capacitance C_x and resistance R_x, the state voltage $v_x(ij)$ is established at the summing node and satisfies a set of differential equations:

$$\begin{aligned} C_x \frac{dv_{xij}(t)}{dt} &= -\frac{1}{R_x} v_{xij}(t) + i_{xij}(t) \\ &= -\frac{1}{R_x} v_{xij}(t) + \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l)v_{ykl}(t) \\ &\quad + \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l)v_{xkl}(t) + I_b \\ &\qquad 1 \le i \le n, \quad 1 \le j \le m. \end{aligned} \qquad (2)$$

The cell contains a nonlinearity between the state node and the output, and its input-output relationship is represented by $V_{yij}(t) = f(v_{xij}(t))$. The nonlinear function used in a CNN can be any differentiable, non-decreasing function $y = f(x)$, provided that $f(0) = 0$, $df(x)/dx$ e 0, $f(+\,)$ ' +1 and $f(-\,)$ ' -1. Two widely used nonlinearities are the piecewise-linear and sigmoid functions as given by

$$y = f(x) = \begin{cases} \frac{1}{2}\left(|x+1| - |x-1|\right) & \text{piecewise-linear} \\ \frac{1-e^{-\lambda x}}{1+e^{-\lambda x}} & \text{sigmoid} \end{cases} \qquad (3)$$

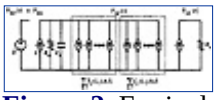


Figure 2 Equivalent circuit diagram of one cell.

Here, the parameter » is proportional to the gain of the sigmoid function. For a unity neuron gain at $x = 0$, » = 2 may be used for the sigmoid function. The gain of neurons in a Hopfield neural network is very large so that the steady-state outputs are all binary-valued. However, if the positive feedback in the CNN cell is so strong that the feedback factor is greater than one, the gain of the cell need not be large to get a guaranteed binary output in the steady state. Typically, a unity gain $df(x)/dxl|_{x=0} = 1$ is used in CNNs. The transfer characteristics of the piecewise-linear function are shown in Figure 3.

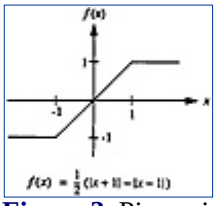


Figure 3 Piecewise-linear function.

The piecewise-linear function provides a mathematical tractability in the analysis, while the sigmoid-like nonlinearity can be easily obtained as a by-product of electronic circuits such as operational amplifier. The shift-invariant CNNs have the interconnections that do not depend on the position of cells in the array except at the edges. The shift-invariant property of a CNN is the most desirable feature when implementing a large-size electronic network such as a VLSI chip. The weights of a shift-invariant CNN can be represented by the $(2r+1)\times(2r+1)$ feedforward and feedback cloning templates:

$$\begin{aligned} \mathbf{T_A} &= [a_{p,q}, -r \le p,q \le r], \\ \mathbf{T_B} &= [b_{p,q}, -r \le p,q \le r]. \end{aligned} \qquad (4)$$

Let $N = n \times m$ be the number of cells in a CNN. By using the vector and matrix notations, (2) can be rewritten as

$$C_x \frac{\mathbf{dx}}{\mathbf{dt}} = -\frac{1}{R_x}\mathbf{x} + \mathbf{Ay} + \mathbf{Bu} + I_b\mathbf{w} \tag{5}$$

where

$$\mathbf{x} = [x_1\, x_2 \cdots x_N] = \left[v_{x1}(t) \mid v_{x2}(t) \mid \cdots \mid v_{xn}(t)\right]^T \quad N \times 1$$

$$\mathbf{y} = [y_1\, y_2 \cdots y_N] = \left[v_{y1}(t) \mid v_{y2}(t) \mid \cdots \mid v_{yn}(t)\right]^T \quad N \times 1$$

$$\mathbf{u} = [u_1\, u_2 \cdots u_N] = \left[v_{ux1}(t) \mid v_{ux2}(t) \mid \cdots \mid v_{un}(t)\right]^T \quad N \times 1$$

$$\mathbf{A} = toeplitz\left(\left(\mathbf{A}_0 \mid \mathbf{A}_1 \mid \cdots \mid \mathbf{A}_r \mid \mathbf{0} \mid \cdots\right), \left(\mathbf{A}_0 \mid \mathbf{A}_{-1} \mid \cdots \mid \mathbf{A}_{-r} \mid \mathbf{0} \mid \cdots\right)\right) \quad N \times N$$

$$\mathbf{B} = toeplitz\left(\left(\mathbf{B}_0 \mid \mathbf{B}_1 \mid \cdots \mid \mathbf{B}_r \mid \mathbf{0} \mid \cdots\right), \left(\mathbf{B}_0 \mid \mathbf{B}_{-1} \mid \cdots \mid \mathbf{B}_{-r} \mid \mathbf{0} \mid \cdots\right)\right) \quad N \times N$$

$$\mathbf{w} = [1 \quad 1 \quad \cdots \quad 1]^T \quad N \times 1$$

Here,

$$\mathbf{v}_{\mathbf{xk}} = \left[v_{xk1}(t)\, v_{xk2}(t) \cdots v_{xkm}(t)\right] \quad 1 \times m$$

$$\mathbf{v}_{\mathbf{yk}} = \left[v_{yk1}(t)\, v_{yk2}(t) \cdots v_{ykm}(t)\right] \quad 1 \times m$$

$$\mathbf{v}_{\mathbf{uk}} = \left[v_{uxk1}\; v_{uk2} \cdots v_{ukm}\right] \quad 1 \times m$$

$$\mathbf{A}_k = toeplitz\left(\left(a_{k,0} a_{k,1} \cdots a_{k,r}\, 0 \cdots\right), \left(a_{k,0} a_{k,-1} \cdots a_{k,-r}\, 0 \cdots\right)\right) \quad m \times m$$

$$\mathbf{B}_k = toeplitz\left(\left(b_{k,0} b_{k,1} \cdots b_{k,r}\, 0 \cdots\right), \left(b_{k,0} b_{k,-1} \cdots b_{k,-r}\, 0 \cdots\right)\right) \quad m \times m$$

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

and toeplitz(**a,b**) is defined as the Toeplitz matrix with **a** in the first row and **b** in the first column. Note that the submatrices $\mathbf{A_k}$ and $\mathbf{B_k}$ are toeplitz, but **A** and **B** are not. The elements of $\mathbf{T_A}$ and $\mathbf{T_B}$ are often normalized to the scale of T_x , e.g., 10^{-3}. The notations of voltages $v_x(t)$, $v_y(t)$ and the state variables **x**, **y** will be used interchangeably hereafter. Because -1 d y_k d + 1, k, the output variable **y** is confined within the N-dimensional hypercube so that **y** $\mathbf{D^N}$= {**y** $\mathbf{R^N}$: -1 d y_k d 1; k = 1, 2, ..., N}. The cloning templates are called symmetric if $A(i,j;k,l) = A(k,l;i,j)$ and $B(i,j;k,l) = B(k,l;i,j)$. In this case, **A** and **B** are symmetric matrices and the stability of the network is guaranteed. In fact, the symmetry of **A** is a sufficient condition for stability. Under the constraint conditions $|v_{xij}(0)|$ d 1 and $|v_{uij}|$ d 1, i, j, theshift-invariant CNN always produces a stable output in the steady state. Moreover, if $A(i,j;i,j) > 1/R_x$, then the saturated binary outputs are guaranteed.

In any CNNs, all states $v_{xij}(t)$, t e 0, are bounded and the bound $v_{x,max}$ can be determined by [1]:

$$v_{x,\max} = 1 + R_x|I_b| + R_x \max_{\substack{1\le i\le n \\ 1\le j\le m}}\left(\sum_{C(k,l)\in N_r(i,j)} \left(|A(i,j;k,l)| + |B(i,j;k,l)| \right) \right) \qquad (6)$$

The terms in (6) account for the initial value, bias, feedback, and feedforward interactions, respectively. Therefore, the operating range of the circuits for summation and integration in Figure 1(b) must be at least $-v_{x,max}$ d $v_{xij}(t)$ d $v_{x,max}$

2.2 Stability

The energy function of a cellular/compact neural network with the piecewise-linear function [1] which is stable for signal processing is described by the Lyapunov or generalized energy function [6]

$$E(t) = -\frac{1}{2}\sum_{i,j}\sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l)v_{yij}(t)v_{ykl}(t) + \frac{1}{2R_x}\sum_{i,j}\left(v_{yij}(t)\right)^2 - \sum_{i,j}\sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l)v_{yij}(t)v_{ukl} - \sum_{i,j} I_b v_{yij}(t) \qquad (7)$$

For the networks with sigmoid function, the second item of (7) is replaced by

$$\frac{1}{R_x}\sum_{i,j}\int_0^{v_{yij}(t)} f_{i,j}^{-1}(v)dv \qquad (8)$$

The expression (8) can be used for any suitable nonlinearity $y = f(x)$ if its inverse function $x = f^{-1}(y)$ can be well defined over the range of x. It can be interpreted as the area of the function $x = f^{-1}(y)$ when integrated from $y = 0$ to $y = v_{yij}$ d 1. The piecewise-linear function used in (7) is a special case of this general expression (8). For the piecewise-linear function, $x = f^{-1}(y) = y$, -1 d y d 1,

$$\int_0^{v_{yij}(t)} f^{-1}(y)dy = \int_0^{v_{yij}(t)} ydy = \frac{1}{2}\left(v_{yij}(t)\right)^2 \qquad (9)$$

which is consistent with the one in (7). In the vector form, (7) is a scalar-valued quadratic function of output vector **y**,

$$\begin{aligned} E &= -\frac{1}{2}\mathbf{y}^T\mathbf{A}\mathbf{y} + \frac{1}{2R_x}\mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{B}\mathbf{u} - I\mathbf{y}^T\mathbf{w} \\ &= -\frac{1}{2}\mathbf{y}^T\mathbf{M}\mathbf{y} - \mathbf{y}^T\mathbf{b} \end{aligned} \qquad (10)$$

where $\mathbf{M} = \mathbf{A} - (1/R_x)\mathbf{I}$ and $\mathbf{b} = \mathbf{B}\mathbf{u} + I_b\mathbf{w}$. The stability of the network can be tested by checking the behavior of the energy function after the network is activated at time $t = t_0$.

If **A** is symmetric, so is **M** and $\mathbf{M} = \mathbf{M}^T$. From [3], we have

$$\nabla_y E = -\mathbf{A}\mathbf{y} + \frac{1}{R_x}f^{-1}(\mathbf{y}) - \mathbf{b} = -\mathbf{A}\mathbf{y} + \frac{1}{R_x}\mathbf{x} - \mathbf{b} = -C_x\frac{dx}{dt} \qquad (11)$$

and

$$\frac{dE}{dt} = \nabla_y E\frac{dy}{dt} = -C_x\frac{dx}{dt}\frac{dy}{dt} = -C_x\sum_{k=1}^{N}\frac{\partial f}{\partial x_k}\left(\frac{dx_k}{dt}\right)^2 \le 0 \qquad (12)$$

So, the network is stable.

Note that in any case, the initialization of the state voltage $v_{xij}(t)$ is required at the beginning of each operation, such that $|v_{xij}(0)|$ d 1, i,j. Otherwise, the undesirable situation $E(t = 0) < E(t = +\ )$ may occur.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

3. 1-D Compact Neural Networks for Wireless Communication

The performance of digital communication systems is largely affected by an ability to overcome the channel impairment introduced during signal transmission. Information-carrying signals in wireless communication are often distorted severely due to noise, limited bandwidth, and multipath propagation through surrounding objects. An optimum method of detecting digital data symbols transmitted over time-dispersive, time-varying channels in the presence of additive white Gaussian noise has been known as the maximum-likelihood sequence estimation (MLSE). As a class of nonlinear receivers, the MLSE detectors exhibit optimum error rate performance compared to their linear counterparts. However, it is often impractical to construct MLSE receivers due to the computation-intensive complexity required for the signal processing functions. A more efficient computational method of implementing MLSE is the Viterbi algorithm [7, 8] in which redundant computation involved in the MLSE is avoided. Research toward more efficient VLSI implementations of the algorithm continues in both academia and industry [9, 10].

From an optimization point of view, the MLSE is a combinatorial maximization or minimization of the cost function over all possible sequences of a certain length. The signaling alphabet, $\pm = \{\pm_k\}$, $k = 1, 2,..., M$, and sequence $s_n = \{s_i\}$, $i = 0, 1,..., n - 1$, correspond to a finite set of numbers and the degree of freedom, respectively. There are possible combinations over which the MLSE computes the cost function.

Artificial neural networks have shown great promise in solving many complex signal processing and optimization problems that cannot be addressed satisfactorily with conventional approaches. The neural network approaches in communications have been motivated by the adaptive learning capability and the collective computational properties to process real-world signals. Well-designed neural networks have the ability to perform the error correction of error-control codes [11, 12], equalization of transmission channels, crossbar switch control, and wireless/networking control. Performing maximum likelihood decoding of linear block error-correcting codes is shown to be equivalent to finding a global minimum of the energy function associated with a neural network [12]. Given a code, a neural network can be constructed in such a way that there exists a one-to-one correspondence between every codeword and every local minimum of the energy function. Decoding techniques using neural networks can be a useful tool for solving problems of maximization of polynomials over the multi-dimensional space.

In this section, a generic architecture of compact neural network [3] as a parallel computational framework of the MLSE is presented. The compact neural network has collective computational properties similar to those

of Hopfield neural networks [14, 15] and can be used to solve difficult optimization problems. The cost function to be minimized in the MLSE has the same quadratic form as the Lyapunov function associated with a compact neural network. If the cost function is mapped onto the network, then the desired estimate is obtained at the output. However, for a combinational optimization it suffers from the local minimum problem as in Hopfield neural networks [16, 17]. Optimum or near optimum solutions can be obtained by applying the parallel hardware annealing [17-19] which is a deterministic relaxation process for searching a global minimum energy state in a short period of time. As an efficient implementation of MLSE function, a hardware-annealed compact neural network is proposed and examined as a real-time machine for combinatorial optimization problems.

3.1 Digital Communication and Neural Networks

Figure 4 shows a block diagram of the baseband model of a digital communication system over the inter-symbol interference (ISI) and additive Gaussian noise channel. The actual ISI channel together with baseband Nyquist filters in the transmitter and receiver can be modeled as a finite impulse response (FIR) filter of length L+1, whose impulse response is given by $h(k) = h_k$ with the corresponding Z-transform $H(z)$. Here, L is the number of symbol intervals over which the ISI spans and hence $h(k) = 0$ for $k < 0$ and $k > L$.

The received signal *r(t)* is produced by the convolution of $u(k) = \sum_i u_i \delta(k-i)$ with $h(k)$ where ´ (k) is the Kronecker delta function, plus white Gaussian noise *n(k)* of zero-mean and finite variance Ã²:

$$r(k) = \sum u_i h(k-i) + n(k) \qquad (13)$$

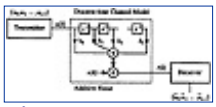


Figure 4 Discrete-time model of baseband communication system.

The maximum-likelihood sequence estimator selects a sequence as a best estimate of the transmitted sequence, which maximizes the conditional *a posterior* probabilities $p(\mathbf{r}_n \mid \mathbf{u}_n)$, where $\mathbf{r_n} = \{r_0, r_1, ..., r_{n-1}\}$ and $\mathbf{u}_n = \{u_0, u_1, ..., u_{n-1}\}$ are the received and transmitted sequences of length n, respectively. For a sufficiently large n, the MLSE algorithm is to choose a sequence that maximizes a scalar cost function

$$J = -\sum_{k=0}^{n-1}\left[r_k - \sum_{i=0}^{L} h_i^* u_{k-i}\right]^2 = -\sum_{k=0}^{n-1}\left[r_k - \sum_{i=0}^{L} h_{k-i}^* u_i\right]^2 \qquad (14)$$

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

for all possible combinations of the sequences of length n. Without the minus sign, this cost function is simply the sum of squared errors between received samples and the outputs of a noise-free channel for the input $\mathbf{u}_n$. The evaluation of values given by (14) must be performed over all possible sequences of $\mathbf{u}_n = \{u_0, u_1,..., u_{n-1}\}$. Therefore the algorithm complexity is proportional to M^n, where M is the number of signaling alphabets, i.e., $u_k \quad \{\pm_1, \pm_2,..., \pm_M\}$, k and n is the length of the sequence to be estimated. In typical data communication, in which the length of a sequence is not given explicitly, the number n can be arbitrarily large and in principle could be infinity. When (14) is expanded, the first term can be discarded because it is a constant for the given input r_n. Then, by changing the sign of the resulting cost function and dividing it by 2, the MLSE is equivalent to minimizing

$$\begin{aligned}\hat{J}_n &= -\mathrm{Re}\left\{\sum_{i=0}^{n-1}\left[\sum_{k=0}^{n-1} r_k h_{k-i}^*\right]u_i\right\} + \frac{1}{2}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\left[\sum_{k=0}^{n-1} h_{k-i}^* h_{k-j}\right]u_i u_j^* \\ &= \frac{1}{2}\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} x_{i-j} u_i u_j^* - \mathrm{Re}\left\{\sum_{i=0}^{n-1} y_i u_i\right\}\end{aligned} \tag{15}$$

where

$$y_i \equiv \sum_{k=0}^{n-1} r_k h_{k-i}^* ,$$

$$x_l \equiv \sum_{k=0}^{n-1} h_k^* h_{k+l} = \sum_{k=0}^{L} h_k^* h_{k+l}$$

The sample y_i is the cross-correlation between the received signal and $h(k)$, while x_l is the auto-correlation of $h(k)$. Here, the channel is assumed to be time-invariant during at least n symbol intervals so that $x_{-k} = x^*_k$, $k = 1, 2,...,L$. In vector and matrix forms, (15) can be written as

$$\hat{J}_n = \frac{1}{2}\mathbf{u}^H\mathbf{X}\mathbf{u} - \mathrm{Re}\{\mathbf{u}^H\mathbf{y}\} \qquad (16)$$

where

$$\mathbf{u} = [u_0\ u_1 \cdots u_{n-1}] = \mathbf{u}_I + j\mathbf{u}_Q, \qquad \mathbf{u} \in \{\alpha_1, \alpha_2, \cdots, \alpha_M\}^n, \alpha_i \in C$$

$$\mathbf{y} = [y_0\ y_1 \cdots y_{n-1}] = \mathbf{y}_I + j\mathbf{y}_Q, \qquad \mathbf{y} \in C^n$$

$$\mathbf{X} = \begin{bmatrix} x_0 & x_{-1} & \cdots & x_{-n+2} & x_{-n+1} \\ x_1 & x_0 & \cdots & x_{-n+3} & x_{-n+2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{n-2} & x_{n-3} & \cdots & x_0 & x_{-1} \\ x_{n-1} & x_{n-2} & \cdots & x_1 & x_0 \end{bmatrix} = \mathbf{X}_I + j\mathbf{X}_Q, \qquad \mathbf{X} \in C^{n \times n}$$

In general, a data communication system transmits and receives a sequence of complex-valued data symbols $\{u_k\}$, where $u_k = u_{I,k} + ju_{Q,k}$, $u_{I,k}$ $\pm$, $u_{Q,k}$ $\pm_Q$. In this case, the correlation matrix X is Hermitian, and positive semi-definite, so that it can guarantee stable operation of the neural network [3].

3.2 System Model

The signaling alphabet $\pm = \{\pm_1, \pm_2, ..., \pm_M\}$ depends on the modulation techniques employed. For simplicity, the binary and QPSK modulations are considered. In this case, the binary antipodal neurons of the form in (3) can be directly incorporated with the signaling formats. However, a neural network with multilevel neurons can be used for more complicated signal constellations, e.g., 4-level bipolar neurons for 16-ary QAM.

Figure 5 shows the block diagram of the neural network MLSE receiver. The received signal $r(t)$ is first separated into two baseband signals, i.e., in-phase signal $r_I(t)$ and quadra-phase signal $r_Q(t)$. The signals are then sampled at $t = kT$ where T is the duration of a symbol, and the resulting discrete-time signals $r_I(t)$ and $r_Q(t)$ are correlated with the channel impulse response $h(k)$. The correlation filter matched to channel impulse response $h(k)$ is approximated by an FIR filter, whose tab coefficients are updated sequence by sequence. In general, because the channel characteristics are not known, the channel's impulse response is also estimated by using the received reference signals. Thus, the estimate $\hat{h}(t)$ is used instead. Note that the channel estimation is equivalent to finding a set of filter coefficients that minimizes the cost function or the mean-squared errors between received and desired signals. Therefore, it corresponds to a convex optimization problem that can be efficiently solved by a neural network [15, 20]. However, such an approach is not considered and the structure of the channel estimator in Figure 5 is left undecided here.

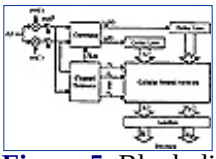


Figure 5 Block diagram of neural network MLSE receiver.

A compact neural network can be used as the core of nonlinear signal processor for the MLSE as shown in Figure 5. From [3], the desired estimate $\hat{\mathbf{u}}_n$ can be obtained at the output of a compact neural network if

$$\mathbf{M} = -\overline{\mathbf{X}} = -\begin{bmatrix} \mathbf{X}_I & \mathbf{X}_Q^T \\ \mathbf{X}_Q & \mathbf{X}_I \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \overline{\mathbf{y}} = \begin{bmatrix} \mathbf{y}_I \\ \mathbf{y}_Q \end{bmatrix} \qquad (17)$$

In other words, the cost function $\hat{J}_n$ is mapped onto a neural network constructed by the transconductance matrix $\overline{\mathbf{A}} = -\overline{\mathbf{X}} + T_x\mathbf{I}$ and input vector $\mathbf{b} = \overline{\mathbf{y}}$. Here, the constant term $T_x\mathbf{I}$ represents a positive unity feedback in each cell. If the neural network produces saturated binary or multilevel values in the steady state, the output represents the MLSE of received sequence, i.e., $\hat{\mathbf{u}}_n = \{\hat{u}_0, \hat{u}_1, \cdots, \hat{u}_{n-1}\}$. After n symbols are shifted into the delay lines, the network performs the MLSE of an n-symbol sequence through an autonomous evolution of its internal state for 0 d $t < T_c$ where T_c is the convergence speed of the network. If the shift

operations of delay lines are pipelined, the network can estimate n/T_c symbols per second. For example, if T_c = 1 ¼ sec and n = 100, then a real-time operation of symbol rate up to 1×10^8 symbols/sec is readily achievable.

The Lyapunov function of a Hopfield neural network is of the form described in (10) [13, 14] and has been utilized to solve many optimization problems [14, 15, 20]. However, for the cost functions in which the diagonal elements are non-zero, the neuron gain must be finite and large for guaranteed stable operation [13, 14]. This in turn produces high-order nonlinear terms in the energy function, which can cause errors in mapping a cost onto the energy function. On the other hand, the expression (10) of a CNN with the piecewise-linear neurons is an exact quadratic function that can be easily incorporated with a large class of optimization problems.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

3.2.1 Combinatorial Optimization

When the MLSE cost function is mapped onto a compact neural network by (17), one difficulty may arise. For the autocorrelation function, x_0 e 0 and all the diagonal elements of M are negative $A(i, j; i, j)$-T_x = -x_0 d 0 and the amount of positive self-feedback in each cell is less than 1. In other words, the matrix $-\mathbf{M} = \overline{\mathbf{X}}$ is positive semi-definite and E is a convex function of output $\mathbf{v}_y$. Correspondingly, the saturated binary output is not guaranteed and continuous-valued steady-state output $\mathbf{v}_y$ $\mathbf{D}^{2n}$ may occur. Although it is possible to obtain the desired estimate by using additional limiting devices at the output, a network with combinatorial solutions $\mathbf{v}_y = \bar{\mathbf{u}} \in \{-1, +1\}^{2n}$ is highly desirable for reducing the circuit complexity and the effect of noise. To transform the convex optimization into a concave equivalence, we add the constraint energy:

$$E_C = \mu \sum_{k=1}^{n} \left(v_{yk}^2 - 1\right) = \mu \left(\mathbf{v}_y + \mathbf{w}\right)^T \left(\mathbf{v}_y - \mathbf{w}\right) \qquad (18)$$

where $\mathbf{w}$ is now a 2n-by-1 unity vector and a constant ¼ is chosen such that $\overline{\mathbf{M}} = -\overline{\mathbf{X}} - 2\mu\mathbf{I}$ is positive definite. The constraint energy corresponds to an additional cost for constraint satisfaction in mathematical optimization [21] and must satisfy E_ce0, $\mathbf{v}_y$ $\mathbf{D}^{2n}$, where the equality holds only if $\mathbf{v}_y = \bar{\mathbf{u}} \in \{-1, +1\}^{2n}$. If we neglect the constant term ¼ $\mathbf{W}^T\mathbf{W}$ = ¼(2N), the cost function is mapped onto a neural network with a modified energy function:

$$\begin{aligned} \hat{E} &= E\big|_{\mathbf{M}=-\overline{\mathbf{X}},\mathbf{b}=\overline{\mathbf{y}}} + \mu\, \mathbf{v}_y^T\mathbf{v}_y = -\frac{1}{2}\mathbf{v}_y^T\left(-\overline{\mathbf{X}} - 2\mu\mathbf{I}\right)\mathbf{v}_y - \mathbf{v}_y^T\overline{\mathbf{y}} \\ &\equiv -\frac{1}{2}\mathbf{v}_y^T\overline{\mathbf{M}}\mathbf{v}_y - \mathbf{v}_y^T\overline{\mathbf{y}} = -\frac{1}{2}\mathbf{v}_y^T\left(\overline{\mathbf{A}} - T_x\mathbf{I}\right)\mathbf{v}_y - \mathbf{v}_y^T\overline{\mathbf{y}} \end{aligned} \qquad (19)$$

where I is a 2n-by-2n unity matrix and, $\overline{\mathbf{A}} = -\overline{\mathbf{X}} + (T_x - 2\mu)\mathbf{I}$. The parameter ¼ controls the shape of energy landscape. If ¼ < -»$_{max}$/2 where »$_{max}$ is the maximum eigenvalue of $\overline{\mathbf{X}}$, then $\hat{E}$ is a concave function of $\mathbf{v}_y$ by

the negative definiteness of $-\overline{\mathbf{M}} = \overline{\mathbf{X}} + 2\mu\mathbf{I}$, and the saturated binary output in the steady state is guaranteed such that $\mathbf{v}_y \in \{-1,+1\}^{2n}$. The maximum eigenvalue λ_{max}, on the other hand, is difficult to determine and may vary sequence by sequence for channels with slowly time-varying characteristics. The eigenvalues of $\overline{\mathbf{X}}$ are real and upper-bounded by

$$\lambda \le \lambda_{\max}^{U} = x_0 + 2\sum_{k=1}^{L}\left(|x_{ik}| + |x_{qk}|\right)$$

Therefore, the parameter ¼ can be chosen such that ¼ <- λ_{max}^{U}/2. The condition on the parameter ¼ can be somewhat alleviated by allowing an indefinite $\overline{\mathbf{M}}$ with all positive elements on the main diagonal, i.e., $\overline{A}(i,j;i,j) - T_x = -x_0 - 2\mu > 0$.

3.2.2 Cloning Templates

From (17), the compact neural network has two rows and the feedback synaptic weight:

$$\overline{\mathbf{A}} = \left[\begin{array}{c|c} \mathbf{A}_0 & \mathbf{A}_1^T \\ \hline \mathbf{A}_1 & \mathbf{A}_0 \end{array}\right] = \begin{bmatrix} -\mathbf{X}_I + (T_x - 2\mu)\mathbf{I} & -\mathbf{X}_Q^T \\ -\mathbf{X}_Q & -\mathbf{X}_I + (T_x - 2\mu)\mathbf{I} \end{bmatrix} \qquad (21)$$

where $\mathbf{A}_0^T = \mathbf{A}_0$ and $\mathbf{A}_1^T = -\mathbf{A}_1$. The corresponding cloning templates are given as

$$\mathbf{T}_A = \begin{bmatrix} -x_{iL} & \cdots & -x_{i2} & -x_{i1} & 0 & -x_{i1} & -x_{i2} & \cdots & -x_{iL} \\ -x_{rL} & \cdots & -x_{rL} & -x_{r1} & -x_0 + T_x + 2\mu & -x_{r1} & -x_{r2} & \cdots & -x_{rL} \\ x_{iL} & \cdots & x_{i2} & -x_{i1} & 0 & x_{i1} & x_{i2} & \cdots & x_{iL} \end{bmatrix}$$

$$\mathbf{T}_B = [1.0] \qquad (22)$$

For the binary case, the feedback operator (22) is reduced to

$$\mathbf{T}_A = \begin{bmatrix} -x_L & \cdots & -x_2 & -x_1 & -x_0 + T_x + 2\mu & -x_1 & -x_2 & \cdots & -x_L \end{bmatrix} \qquad (23)$$

The network diagram of (22) is shown in Figure 6. Note that the elements of $\mathbf{T}_A$ may be updated dynamically as the characteristics of transmission medium changes. For the cloning templates of the form described in (22), the feedback operator is symmetric in a row, but has an opposite sign in the adjacent two rows. However, the matrix M is symmetric and the compact neural network always finds a stable equilibrium.

3.3 Hardware Annealing

Even with a correct mapping of the MLSE function onto a neural network, the desired optimal or near-optimal solutions are not guaranteed because a combinatorial optimization problem always involves a large number of local minima [21]. Therefore, in addition to the basic structure of the network, the annealing capability is provided to obtain the global minimum of the cost function (15) over all possible combinations of sequence. The hardware annealing [17-19] is a dynamic relaxation process for finding the optimum solutions in the recurrent associative neural networks such as Hopfield networks and compact neural networks. Near-optimal solutions can be obtained by applying the hardware annealing technique for avoiding local minima problems, which are inherent in combinational optimizations.

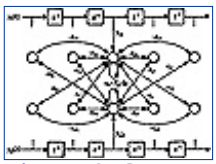


Figure 6 Compact neural network for QPSK modulation $L = 2$.

The hardware annealing is performed by controlling the gain of the neuron, which is assumed to be the same for all neurons throughout the network. After the state is initialized ½$_x$ = ½$_x$(0), the initial gain at time $t = 0$ can be set to an arbitrarily small, positive value such that 0 d $g(0)$ << 1. It then increases continuously for 0 < t dT_A to the nominal gain of 1. The maximum gain $g_{max} = 1$ is maintained $T_A < t$ d T_c, during which the network

is stabilized. When the hardware annealing is applied to a compact neural network by increasing the neuron gain *g(t)*, the transfer function can be described by

$$v_{xij}(t) = f\left(g(t)v_{xij}(t)\right) = \begin{cases} +1; & g(t)v_{xij}(t) \geq 1 \\ g(t)v_{xij}(t); & -1 < g(t)v_{xij}(t) < 1 \\ -1 & g(t)v_{xij}(t) \leq -1 \end{cases} \qquad (24)$$

or simply $y = f(gx)$. Note that the saturation level is still $y = +1$ or -1 and only the slope of $f(x)$ around $x = 0$ varies. By using the normalized variables in vector and matrix notations, (10) can be rewritten as

$$\begin{aligned} E &= -\frac{1}{2}\mathbf{v}_y^T\mathbf{A}\mathbf{v}_y + \frac{T_x}{g}\mathbf{v}_y^T\mathbf{v}_y - \mathbf{v}_y^T\mathbf{b} \\ &= -\frac{1}{2}\mathbf{v}_y^T\mathbf{M}_g\mathbf{v}_y - \mathbf{v}_y^T\mathbf{b} \end{aligned} \qquad (25)$$

where $\mathbf{v}_y = f(g\mathbf{v}_x)$ and $\mathbf{M}_g = \mathbf{A} - (T_x/g)\mathbf{I}$. The process of finding the optimal solutions takes place during the change of $\mathbf{M}_g$ from a negative definite matrix to indefinite or positive matrix, as the annealing gain g increases. Figure 7 shows the diagrams of a possible realization of variable-gain piecewise-linear function (24) using two-quadrant analog multiplier. Notice that although the variable-gain can be readily achieved by controlling V_{cx} , it also changes the value of R_x, which must be kept constant as closely as possible for proper annealing operation. Thus, a separate gain control circuit is required in each neuron.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

3.4 Simulation Results

The simulations of a simple binary communication system with several ISI channels are performed by solving the differential equations (2). Random data sequence $\mathbf{u_n} = \{u_k\}$, $k = 0, 1,..., n\text{-}1$, is generated and convoluted with a channel response h(k) which is assumed to be known exactly. The simulation is conducted on a binary communication system with the ISI channel given by

$$H_m(z) = \frac{1}{\sqrt{1.25}}(1.0 + 0.5z^{-1}) \qquad (26)$$

In this case, $x_0 = 1.0$, $x_1 = x_{-1} = 0.4$, $x_k = 0$ for $|k|$d2, and the correlation matrix is given by

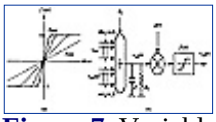


Figure 7 Variable-gain piecewise-linear neuron cell. (a) Transfer curves for several gain values. (b) Block diagram of variable-gain cell with two-quadrant analog multiplier.

For $n = 100$, the minimum and maximum eigenvalues of $\mathbf{X}$ are approximately $»_{min} = 0.2$ and $»_{max} = 1.8$, respectively. Therefore, the coefficient ¼ can be chosen such that the additional value of synaptic weights in the main diagonal satisfies T_x - 2¼ > T_x + 1.8 = 2.8 for T_x =1.

In Figure 8, error rates are plotted for the MLSE of unannealed and annealed NNs. In addition, two different neuron models, i.e., the sigmoid and piecewise-linear functions, are used to compare their effects on the combinatorial optimization problem. Here, the channel model is the same as (26) and 100 simulation runs were performed independently on the sequences of length 100 ($n = 100$) for each signal-to-noise-ratio (SNR) value. From the figure, it can be seen that the neural network with

- Piecewise-linear function, and
- Annealed operation

have 1 ~ 1.5*dB* better performance over the others. The sigmoid nonlinearity not only reduces the number of local minima but also provides chances of smoothing down the desired global minimum.

For the case when no inter-symbol interference is present, the error rate performances of the proposed method and Viterbi algorithm (VA) are shown in Figure 9. All compact neural network models and VA have the same error rates in this case. This might be the consequence of independence between received samples. However, since the correlator in Figure 5 just bypasses the input $(\bar{h}(k) = \delta(k))$, the performances are somewhat poorer than theoretical values $P_e = 0.5\,\mathrm{erfc}(\sqrt{\gamma_b})$, obtained through an ideal continuous-time, matched-filtering of received signal prior to the sampling. Here, erfc(·) is the complementary error function and $^{3}_{b}$ is the received SNR per information bit.

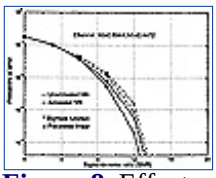


Figure 8 Effects of neuron models and hardware annealing on MLSE performance (L = 1).

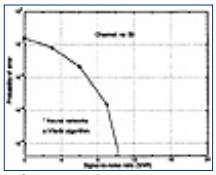


Figure 9 Error rate performances of neural network MLSE and Viterbi algorithm (L = 0).

Next, the error rates of MLSE by unannealed and annealed NNs are shown in Figure 10(a) for a two-ray minimum-phase channel (26) and in Figure 10(b) for a two-ray nonminimum-phase channel $H_n(z) = (0.5 + z^{-1})/\sqrt{1.25}$, respectively. For comparison, the results of VA are also shown in the figures. In the simulation of the second channel $H_n(z)$, it is assumed that the decisions are made in reference to direct received samples, which are half the magnitude of delayed versions. As expected, an annealed NN has a better performance than an unannealed NN in both cases. It might be worthwhile mentioning that the NN-MLSE for a minimum-phase channel $H_n(z)$ is less efficient than the VA at the moderate values of SNR, but does not suffer much from the nonminimum-phase characteristics of the channel.

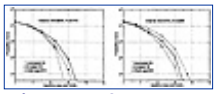


Figure 10 Error rate performances of neural network MLSE and Viterbi algorithm (L = 1, n = 100, 10,000 binary symbols). (a) Two-ray minimum-phase channel. (b) Two-ray nonminimum-phase channel.

4. 1-D Compact Neural Network for Communication: Applications

4.1 Partial Response Maximum Likelihood (PRML) Sequence Detector

Partial response signaling representation is a class of transmission coding schemes with minimized degradation for many high-speed communication systems. For example, the quaternary partial-response class-IV (QPRIV) scheme was used for fiber distributed data interface (FDDI) [22] in twisted pair communication cables. PRIV and extra extended PRIV (EEPR4) schemes are used in the hard-disk drive (HDD) recording channels. Many industrial implementations of digital Viterbi detector at the data rate around 150 Mb/s or higher were recently reported for the HDD recording channels [23-25]. In digital Viterbi detection implementation, the computing operation has an exponential relationship with the intersymbol interference level. The operation has to be executed either serially at lower bitrate or in parallel with significant increase in hardware resource. In addition, an analog/digital (A/D) converter of high resolution is required. The A/D converter usually occupies quite a large silicon area. Analog circuit approaches [26] have been considered to provide better solutions to many barriers in digital implementation from both the algorithmic and hardware design aspects.

A compact neural network-based PRIV signaling scheme detection algorithm was proposed by Choi and Sheu in [3]. The algorithm is an outstanding candidate to resolve the complexity issue in digital Viterbi algorithms and enables analog implementation to achieve high computing efficiency. The salient features of this

algorithm are:

- Reduced hardware complexity, i.e., no A/D converter
- Easy to scale
- Compact VLSI implementation especially in mixed-mode IC design
- Low power consumption

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

4.1.1 State-Constrained Neuron Model

State-Constrained Neuron (SCN) model is the neuron model suitable for the proposed PRML detection algorithms. It is a mathematical dynamic system model which can be implemented in voltage-mode or current-mode VLSI design. It is applicable for optimization applications because of its high linearity, accurate frequency response, and guaranteed stability at the grid points in the solution space of a symmetric network. The structure of an SCN neuron is depicted in Figure 11. C_i is the state-constrained neuron cell, where i is the index number of the cell. The neuron cell is to perform the integration on the state variable x_i. The synaptic connections between Ci and its neighborhood cells are a_{ij}'s, where j is the index number of the neighborhood cells. The external input coefficients are b_{ik}'s, where k is the index of the neighborhood cells. The dynamics of the state variable x_i can be formulated as

$$\tau \frac{dx_i}{dt} = \begin{cases} \sum_{j \in N(C_i)} a_{ij} y_i + \sum_{k \in I(C_i)} b_{ik} + Z_i & -1 \le x_i \le 1 \\ 0 & x_i \ge 1 \| x_i \le -1 \end{cases} \qquad (28)$$

where $y_i = f(x_i)$ is output of the activation function, $N(C_i)$ is the set of neighboring cells which has connections with C_i, and $I(C_i)$ is the set of input nodes to determine external input coefficients b_{ik}'s into the state variable x_i. And in this algorithm, a piecewise linear function is used as the activation function.

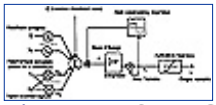


Figure 11 State-Constrained Neuronal (SCN) model for neuron cell C_i.

(28) can be rewritten in matrix format as

$$\tau \frac{d\mathbf{X}}{dt} = \mathbf{AY} + \mathbf{B} + \mathbf{Z} = \mathbf{AX} + \mathbf{B} + \mathbf{Z} \qquad (29)$$

where $\mathbf{X} = \{x_i\}$, $i = 1, 2, ..., n$, $\mathbf{Y} = f(\mathbf{X})$ and -1 d x_i d 1, for all i. **A** is the n-by-n connection matrix for the network, **B** is an external input coefficient vector, and **Z** is a constant threshold vector.

For a network without the threshold component, i.e., Z = 0, the energy function for the network is

$$E = -\frac{1}{2}\mathbf{X}^{\mathrm{T}}\mathbf{A}\mathbf{X} - \mathbf{B}\mathbf{X} \tag{30}$$

This energy function has a gradient of

$$\nabla_x E = -\mathbf{A}\mathbf{X} - \mathbf{B} = -(\mathbf{A}\mathbf{X} + \mathbf{B}) = -(\mathbf{A}f(\mathbf{X}) + \mathbf{B}) = -\tau\left(\frac{d\mathbf{X}}{dt}\right)^2 \le 0$$
$$-1 \le x_i \le 1, \quad \text{for every } i. \tag{31}$$

4.1.2 Maximum Likelihood Sequence Detection (MLSD) Algorithm

Partial response signaling uses the spectrum reshaping idea to produce a signal representation less vulnerable to noise for data communication system given a rectangular binary symbol at the input [30]. Partial response signaling deliberately generates ISI and a degree of correlation between transmitted symbols. Thus, it can be referred to as correlative coding. It is used in baseband systems and various carrier formats including Frequency Modulation (FM) and Spread-Spectrum Broadcasting (SSB). In addition to data communication systems, it is also popular to apply partial response maximum-likelihood (PRML) detection scheme for the data receiver design for reading channel in magnetic recording systems, and recently Digital Video Disc (DVD).

The coder characteristic functions for different classes of partial response signaling are listed in Table 1 [29]. If $d(k)$ is the bipolar input sequence of coder, $f(k)$ is the impulse response of the partial response signaling coder, which could be a delay-impulse response filter or a magnetic reading channel, and $n(k)$ is the white Gaussian noise of the channel, as in Section 3.1, the received data sequence $r(k)$ after channel equalization takes the form of

$$r(k) = \sum d(i)f(k-i) + n(k) \tag{32}$$

Table 1 System characteristic functions for partial response (PR) signaling systems.

PR coder characteristic function *F(Z)*	PR classification	Output levels
$1 + Z^{-1}$	PR1 (duobinary)	3
$1 + 2Z^{-1} + Z^{-2}$	PR2	5
$2 + Z^{-1} - Z^{-2}$	PR3	5
$1 - Z^{-2}$	PR4 (modified duobinary)	3
$1 + Z^{-1} - Z^{-2} - Z^{-3}$	EPR4 (extended PR4)	5
$1 - 2Z^{-2} + Z^{-4}$	PR5	5

And the cost function E can be defined as

$$\begin{aligned} E &= \frac{1}{2}\overline{M(u_n)} - \sum_{k=0}^{n-1} r(k)^2 \\ &= \frac{1}{2}\sum_{k=0}^{n-1}\sum_{j=0}^{n-1}\sum_{m=0}^{n-1} u(j)u(m)f(k-j)f(k-m) - \sum_{k=0}^{n-1}\sum_{i=0}^{n-1} r(k)f(k-i)u(i) \\ &= \frac{1}{2}\mathbf{u}^{\mathrm{T}}\mathbf{D}\mathbf{u} - \mathbf{B}\mathbf{u} \end{aligned} \tag{33}$$

where

$$\mathbf{D} = \left\{d_{j,m}\right\}_{n\times n}, \quad d_{j,m} = \sum_{k=0}^{n-1} f(k-j)f(k-m) \tag{34}$$

$$\mathbf{B} = \left[b_0, b_1, \cdots, b_{n-1}\right]^T, \qquad b_i = \sum_{k=0}^{n-1} r(k)f(k-i) \tag{35}$$

To map the maximum likelihood detection algorithm onto the compact neural network, we define the connection matrix **A** as **-D**. Then the dynamic equation for the compact neural network is

$$\tau \frac{d\mathbf{X}}{dt} = \mathbf{AX} + \mathbf{B} \tag{36}$$

and the energy function becomes

$$E = -\frac{1}{2}\mathbf{X}^{\mathrm{T}}\mathbf{AX} - \mathbf{BX} \tag{37}$$

The state variable vector **X** has a direct mapping to the estimator vector **u** of the data sequence. The connection matrix **A** for each class of coders is listed in column 3 of Table 2. For the connection matrix and network topology, a 4-neuron network is used as an example. The connection matrix can be linearly scaled for larger networks. All the connection matrices are symmetric. A partial response maximum likelihood sequence detecting problem can be solved by a compact neural network with direct mapping using the connection matrix and correlators.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

4.1.3 Performance of Algorithm

The quadratic likelihood metric function of a compact neural network is similar to the likelihood metric function used in the Viterbi detectors [26]. The advantages of using compact neural networks are:

(1) No need of trellis tree expansion for full searching in solution space

(2) Distributed parallel optimization

(3) No need of temporary memory

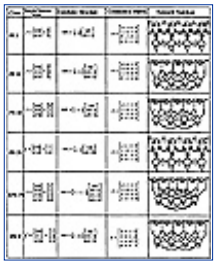
(4) No hardware duplication redundancy

A 4-neuron compact neural network for the maximum likelihood sequence detection of the class IV partial response signaling scheme was simulated with *Matlab* tools on a SPARC-20 workstation. The simulation result indicates that the network can reach the stable state within 2 time constants. The worst cases found in the simulation under different noise levels are all bounded within 6 time constants. The worst-case stable time for the network of different sizes at the SNR ratio of 12dB is shown in Figure 2. An almost linear dependence on the frame size can be observed. This indicates that the performance of a detector does not scale with the frame size. And, the time constant t is an important performance indicator which is decided by the VLSI circuit implementation. A time constant of 10 ns is feasible for analog VLSI implementation.

To compare the compact neural network detector with a digital Viterbi maximum likelihood detector, simulations of the BER under various SNRs are performed. The result is shown in Figure 13. In this experiment, 1000 random data sequence runs were used as the input to a compact neural network in a 20-neuron processor array. The simulation result indicates that the difference of the maximum likelihood detection result between the 20-neuron compact neural network and the digital Viterbi detector is within 5 - 15% for different SNR values.

Table 2 Computing structure of compact neural

networks for PR signaling schemes. (A 4-neuron network example)



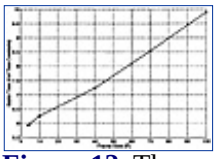


Figure 12 The worst-case stable time for the network of different sizes at SNR of 12 dB.

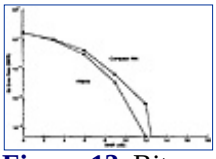


Figure 13 Bit error rate performance for a 20-neuron processor array.

4.2 A CDMA Communication Detector with Robust Near-Far Resistance

With rapid progress of wireless communication and the strong necessity for high capacity and high quality communication systems, digital communication techniques in mobile systems, CT-2, satellites, wireless asynchronous transfer modes (ATM), mobile personal computers, etc., are widely used. Among these techniques, code division multiple access (CDMA) is receiving significant attention due to its high spectral efficiency and robust resistance of multipath and jamming effects [3]. Generally, according to recent statistics, 11 subscribers can be accommodated per Megahertz per base station for Advanced Mobile Phone Service (AMPS); 85 for Global System for Mobile (GSM); and 283 for CDMA. Besides, the CDMA standards TIA/EIA IS-95A (cellular) and ANSI J-STD-008 (PCS) have been built for commercial use. Part of the uplink block diagram for IS-95 is shown in Figure 14 [32]. CDMA is a technique of spread spectrum communication. Each user is assigned a distinct pseudo-noise (PN) code. If every user's code is uncorrelated with those of other users, all the traffic channels within one cell can share the same bandwidth simultaneously.

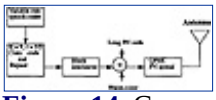


Figure 14 Conceptual block diagram of uplink in IS-95.

To maintain high quality and high spectral efficiency in CDMA systems, controlling signal power of users is very important. In satellite communication, the high-power and low-power transmitters co-exist. In ground communication, some users may be near the base station and some may be far away. A 60 dB or greater signal power difference at the base station for two mobiles is quite possible. When an unwanted user's signal received is much larger than the received signal power contributed by the desired user, the performance of CDMA is seriously impaired in the radio environment. This is called the near-far problem and is a major technical problem in CDMA. In 1986, Verdu [33] showed that the optimal near-far resistant detector could be achieved by minimizing an integer quadratic object function. It means multi-user detection in CDMA can be converted into an optimization problem.

In this section, a compact neural network to implement the optimal multi-user detectors (OMDs) is presented.

By mapping the quadratic object function in OMD onto the Lyapunov function associated with compact neural network, the desired estimate can be obtained. Optimum or near-optimum solutions can be obtained by applying the paralleled hardware annealing. The annealed compact neural network multi-user detector (ACNNMD) does not require specific initial states and can provide results very close to that of the OMD.

4.2.1 Conventional Detector and Optimized Decision Rule

For illustrative purposes, K active users for the same synchronous Gaussian channel in a direct-sequence CDMA (DS-CDMA) system at a given time t are assumed. Furthermore, all the carrier phases are assumed to be zero. There are K different signature waveforms $\{s_k(i), k = 1, 2,..., K\}$. Each waveform s_k is composed of a string of bits $\{b_k(i) \quad \{-1, +1\}\}$. In a DS-CDMA system, the received signal is

$$r(t) = \sum_{k=1}^{K} b_k(i) s_k(t - iT) + n(t) \qquad (38)$$

where $n(t)$ is Gaussian noise. If focus is centered on one symbol interval in (38), the function of a detector is to recognize every active user's symbol at the specified interval. With consideration to custom VLSI chip architecture, conventional detector and optimal multi-user detector (OMD) are discussed.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

(A) Conventional Detector

A conventional detector consists of K filters matched to the signature waveforms of K subscribers. Each signature waveform (or PN code) S_k is reconstructed and correlated with the received signal $r(t)$. After passing through the matched filter bank, $r(t)$ becomes K parallel outputs. The K outputs are sampled at the bit time. Simple decision devices following the matched filters provide every user's symbol estimates based upon the signs of the output of the matched filters. The block diagram is shown in Figure 15,

$$y_k^{(i)} = \int_{iT}^{(i+1)T} r(t)s_k(t-iT)dt$$
$$\mathbf{b}_{CD}^{(i)} = sign\left(\mathbf{y}^{(i)}\right) \qquad (39)$$

where $\mathbf{b}_{CD}^{(i)} = \left[b_1^{(i)} b_2^{(i)} \cdots b_K^{(i)}\right]^T$ and $\mathbf{y}^{(i)} = \left[y_1^{(i)} y_2^{(i)} \cdots y_K^{(i)}\right]^T$. Multiple access interference (MAI) has significant effect on the performance and capacity of conventional detectors. In addition, the near-far problem cannot be solved in a conventional detector. To achieve better performance, the optimal multi-user detector (OMD) was developed. OMD not only improves the performance but also reduces the precision requirements for power control. Besides, reducing interference in uplink also means the same transmitted power can be used for a larger coverage of a cell.

(B) Optimal Multi-user Detector

By minimizing the noise energy, an OMD [35] can be derived. This minimization is equivalent to maximizing the logarithm of the likelihood function. The matrix representation of $\mathbf{y}$ is

$$\mathbf{y} = \mathbf{H}\mathbf{b} + \mathbf{n} \qquad (40)$$

where $\mathbf{n} = \left[n_1^{(i)} n_2^{(i)} \cdots n_K^{(i)}\right]^T$ and $\mathbf{H}$ $R^{K\times K}$ is a cross correlation matrix of the signature waveforms:

$$h_{ij} = \int_0^T s_i(t) \cdot s_j(t)dt \qquad (41)$$

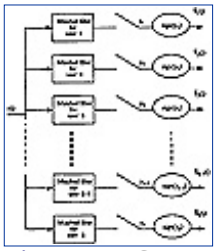


Figure 15 Conventional detector.

The matrix **H** is nonnegative definite, Given the observation *r(t)*, the OMD is to generate an estimate $\hat{\mathbf{b}} = [\hat{b}_1, \hat{b}_2, \cdots, \hat{b}_K]^T$ to minimize the cost function [33]:

$$\hat{\mathbf{b}}_{OMD}^{(i)} = \arg \min_{b \in \{-1,+1\}^K} \int_0^T \left[r(t) - \sum_{k=1}^{K} b_k s_k(t) \right]^2 dt \qquad (42)$$

Note that (43) can be written in a matrix form:

$$\hat{\mathbf{b}}_{OMD}^{(i)} = \arg \min_{b \in \{-1,+1\}^K} \frac{1}{2} \mathbf{b}^T \mathbf{H} \mathbf{b} - \mathbf{b}^T \mathbf{y} \qquad (43)$$

H is Hermitian symmetric and positive definite. In (43), the optimized multi-user detection problem becomes a quadratic optimization problem. Because of the high throughput rate, shift-invariant property and collectively computational properties, the compact neural network is chosen for the architecture realization of OMD. Figure 16 shows the functional block diagram of a compact neural network-based CDMA detector.

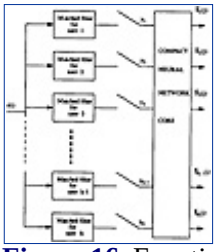


Figure 16 Functional block diagram of compact NN based CDMA detector.

4.2.2 Compact Neural Network CDMA Detector

Compact neural network is an effective architecture for one-dimensional signal processing in communication. Due to the similarity between (43) and (10), a linear mapping technique is applied. The output of a compact neural network will be the desired estimate $\hat{\mathbf{b}}$ if

$$\mathbf{M} = -\mathbf{H} \text{ and } \quad \mathbf{d} = \mathbf{y} \qquad (44)$$

Figure 17 shows the architecture of the neuron core in CDMA detector for five users. Due to the MAI, this network is fully connected. The synapse values are determined by the correlations of these users' PN codes.

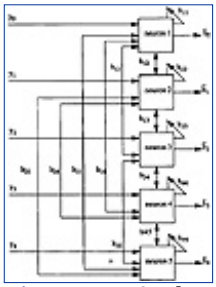


Figure 17 Architecture of compact NN core in CDMA detector (K = 5).

4.2.3 Simulation Results

In this section, the performance of the conventional detector and that of the biologically inspired compact neural network-based CDMA detector are compared. First, consider a K = 2 synchronous and noiseless case.

The original transmitted signal is $b_{sent} = [-1, -1]^T$ at a given time. The near-far ratio r_{nf} is defined as

$$r_{nf} = \frac{\int s_i^2 dt}{\int s_j^2 dt} \tag{45}$$

and the normalized cross-correlation of signature waveforms h is defined as

$$h = \frac{\int s_i s_j dt}{\sqrt{\int s_i^2 dt \int s_j^2 dt}} \tag{46}$$

Therefore, the synapse matrix **H** can be written as

$$\mathbf{H} = \begin{bmatrix} r_{nf}^{1/2} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & h \\ h & 1 \end{bmatrix} \begin{bmatrix} r_{nf}^{1/2} & 0 \\ 0 & 1 \end{bmatrix} \tag{47}$$

The normalized cross-correlation of the users' signature waves is h. The outputs of the matched filters **y** were sent to the neuron core for detection. The failure points (FPs) were recorded. Figure 18 show the distribution of failure points. Each failure point is determined if $b_{detected}$ ` b_{sent}. The result of OMD is obtained by full search as the reference method. The simulation range for the near-far ratio r_{nf} is [1, $10^{0.01}$, $10^{0.02}$,..., 10] and the range for the correlation function h is [-0.9, - 0.89, - 0.88,..., 0].

Simulation results which include the constraint energy function are shown in Figure 18. There were no failure points from the compact neural network-based CDMA detector with piecewise linear function. In Figure 19, signals corrupted by neighboring user's interference and Gaussian noise were considered. The signal-to-noise ratio for user 1 is fixed at 10 dB. The compact neural network-based CDMA detector with piecewise linear function was employed. Note that the neural network detector could have better performance than the optimal multi-user detector in the noise-corrupted cases. Data were obtained from 10,000 cases. Figure 20 shows results from a case of a three synchronous users transmitting their signals spreading by the Gold codes, which have the well-controlled cross-correlation values. The power of the first user is 2 dB stronger than the other two users. Errors were cumulated and divided by 10,000 to determine the probability of error. The biologically inspired compact neural network detector performs almost as well as the optimal multi-user detector.

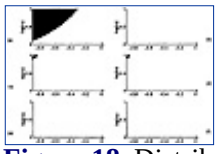


Figure 18 Distribution of failure points. Constraint energy function is added. The logarithmic function is with base 10. (a) Conventional detector. (b) Optimal multi-user detector. (c) Compact NN-based CDMA detector with sigmoid function. (d) Compact NN-based CDMA detector with sigmoid function and hardware annealing. (e) Compact NN-based CDMA detector with piecewise linear function. (f) Compact NN-based CDMA detector with piecewise linear function and hardware annealing.

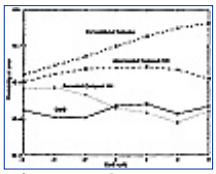


Figure 19 Error probability of conventional detector, compact NN-based CDMA detector with or without hardware annealing function, and optimal multi-user detector. Signal-to-noise ratio for user 1 is fixed at 10 dB.

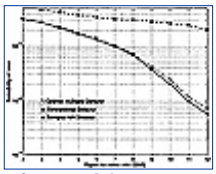


Figure 20 Error probability of compact NN-based CDMA detector with hardware annealing function,

optimal multi-user detector, and conventional detector. Maximum near-far ratio is 2 dB.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

4.3 Conclusion

The collective computational behavior of a compact neural network is used to solve the maximum-likelihood estimation of signals in the presence of intersymbol interference and white Gaussian noise. It is demonstrated that artificial neural network is an efficient way of realizing the PRML receiver and the OMD for CDMA communication. In addition, the performance of the NN-MLSE and NN-OMD can be improved by a paralleled hardware annealing technique which has been developed for high-speed operation of neural networks.

The hardware architecture of the proposed method has the following important properties:

- Massively paralleled operations
- Circuit complexity proportional to length L
- Space-independent local interconnections
- Simple modularity

The NN-MLSE or NN-OMD of a sequence is done in a single evolution interval of the neural network, which can be a few microseconds or just fractions of a microsecond. Moreover, the operating speed of recurrent associative neural networks is almost independent of network size. The throughput rate is typically determined by supporting circuitry which runs at a symbol rate.

Acknowledgments

This work was partially sponsored by ONR under grant N00014-96-1-0331 and by Integrated Media Systems Center (IMSC), which is an Engineering Research Center sponsored by NSF. Valuable interactions with Engineering Dean/Prof. Len Silverman, IMSC Director/Prof. Max Nikias, Associate Director/Prof. Jerry Mendel, and Center for Neural Engineering Director/Prof. Theodore Berger; and Prof. Leon Chua of University of California, Berkeley, Prof. Tamas Roska of Hungarian Academy of Science, are highly appreciated. We would like to thank prior Ph.D. students who graduated from our laboratory, including Dr. Sa H. Bang (currently President of Standard Telecom Inc. in Santa Clara, CA), Dr. Eric Chou (currently research scientist of HP Labs in Palo Alto, CA), and Dr. Cheng-Hsiung Chen for their valuable contributions to this field.

References

1 Chua L.O. and Yang L. (1988), Cellular neural networks: Theory, *IEEE Trans. on Circuits and Systems*, vol. 35, pp. 1257–1272.

2 [VMLI]Chua L.O. and Yang L. (1988), Cellular neural networks: Applications, *IEEE Trans. on Circuits and Systems*, vol. 35, pp. 1273–1290.

3 Sheu B.J. and Choi J. (1995), *Neural Information Processing and VLSI*, Kluwer Academic, Boston, MA.

4 Chua L.O. and Roska T. (1993), The CNN paradigm, *IEEE Trans. on Circuits and Systems*, vol. 40, pp. 147–156.

5 Roska T. and Chua L.O. (1992), Cellular neural networks with non-linear and delay-type template elements and non-uniform grids, *Int J. Circuit Theory and Applications*, vol. 20, pp. 469–481.

6 Kalman R.E. and Bertram J.E. (1960), Control system analysis and design via the "second method" of Lyapunov Part I: Continuous-time systems, *Trans. ASME*, pp. 371–393.

7 Forney G.D., Jr. (1972), Maximum-likelihood sequence estimation of digital sequences in the presence of intersymbol interference, *IEEE Trans. Inform. Theory*, vol. 18, pp. 363–378.

8 Proakis J.G. (1983), *Digital Communications*, McGraw Hill, New York.

9 Gulak P.G. and Shewdyk E. (1986), VLSI structures for Viterbi receivers. Part I: General theory and applications, *IEEE J. Selected Areas in Communications*, vol. 4, pp. 142–154.

10 Fettweis G. and Meyr H. (1990), High-rate Viterbi processor: A systolic array solution, *IEEE J. Selected Areas in Communications*, vol. 8, pp. 1520–1534.

11 Zeng G., Hush D. and Ahmed N. (1989), An application of neural net in decoding error-correcting codes, *Proc. IEEE Int. Symp. Circuits Systems*, pp. 782–785.

12 Bruck J. and Blaum M. (1989), Neural networks, error-correcting codes, and polynomials over the binary n-cube, *IEEE Trans. Inform. Theory*, vol. 35, no. 5, pp. 976–987.

13 Hopfield J.J. (1984), Neurons with graded response have collective computational properties like those of two-state neurons, *Proc. Natl. Acad. Sci. U.S.A.*, vol. 81, pp. 3088–3092.

14 Hopfield J.J. and Tank D.W. (1985), "Neural" computation of decisions in optimization problems, *Biol. Cybern.*, vol. 52, pp. 141–152.

15 Tank D.W. and Hopfield J.J. (1986), Simple "neural" optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit, *IEEE Trans. Circuits Systems*, vol. 33, no. 5, pp. 533–541.

16 Kunz D. (1991), Suboptimum solutions obtained by the Hopfield-Tank neural network algorithm, *Biol. Cybern.*, vol. 65, pp. 129–133.

17 Lee B.W. and Sheu B.J. (1991), Modified Hopfield neural networks for retrieving the optimal solution, *IEEE Trans. Neural Networks*, vol. 2, pp. 137–142.

18 Lee B.W. and Sheu B.J. (1993), Parallel hardware annealing for optimal solutions on electronic neural networks, *IEEE Trans. Neural Networks*, vol. 4, pp. 588–598.

19 Bang S.H. and Sheu B.J. (1996), A hardware annealing method for optimal solutions on cellular neural networks, *IEEE Trans. Circuits Systems-II*, vol. 43, pp. 409–421.

20 Tagliarini G.A., Christ J.F. and Page E.W. (1991), Optimization using neural networks, *IEEE Trans. Comp.*, vol. 40, pp. 1347–1358.

21 Pardalos P.M. and Rosen J.B. (1987), Constrained Global Optimization: Algorithms and Applications, Springer-Verlag, Berlin.

22 Cherubini G., Olcer S. and Ungerboek G. (1995), A quanternary partial-response class-IV transceiver for 125 Mbits/s data transmission over unshielded twisted pair cables: Principles of operation and VLSI realization, *IEEE Journal on Selected Areas in Communications*, vol. 13, no.9, pp. 1656–1669.

23 Mita S. and Ouchi Y. (1996), A 150Mb/s PRML chip for magnetic disk drives, *IEEE Inter. Solid-State Circuits Conference*, pp. 62–63, San Francisco, CA.

24 Tuttle G.T. and Visshakhadatta G. D. (1996), A 130 Mb/s PRML read/write channel with digital-servo detection, *IEEE Inter. Solid-State Circuits Conference*, pp. 64–65, San Francisco, CA.

25 Parsi K. and Rao N. (1996), A 200 Mb/s PRML read/write channel IC, *IEEE Inter. Solid-State Circuits Conference*, pp. 66–67, San Francisco, CA.

26 Spencer R.R. (1992), Simulated performance of analog Viterbi detectors, *IEEE Journal on Selected Areas in Communications*, vol. 10, no. 1, pp. 277–288.

27 Rodriguez A., Espejo S., Dominguez-Castro R., Huertas J. and Sanchez-Sinencio E. (1993), Current-mode techniques for implementation of continuous- and discrete-time cellular neural networks, *IEEE Trans. on Circuits and Systems-II*, vol. 40, no. 3, pp. 132–146.

28 Chou E.Y., Sheu B.J. and Tsai R.H. (1997), A state-constrained model for cellular nonlinear network optimization, submitted to *IEEE Trans. on Circuits and Systems I*, vol. 44, no. 5, pp. 445–449.

29 Wade G. (1994), *Signal Coding and Processing*, Cambridge University Press, New York.

30 Lin H. and Messerschmitt D. (1993), Parallel Viterbi decoding methods for uncontrollable and controllable sources, *IEEE Trans. on Communications*, vol. 41, pp. 62–69.

31 Viterbi A.J. (1995), *CDMA: Principles of Spread Spectrum Communication*, Addison Wesley, Reading, MA.

32 Peterson R.L., Ziemer R.E. and Borth D.E. (1995), *Introduction to Spread Spectrum Communication*, Prentice Hall, Englewood Cliffs, NJ.

33 Verdu S. (1986), Minimum probability of error for asynchronous Gaussian multiple-access channels, *IEEE Trans. on Information Theory*, vol. IT-32, no. 1, pp. 85–96.

34 Moshavi S. (1996), Multi-user detection for DS-CDMA communications, *IEEE Communications Magazine*, vol. 34, no. 10, pp. 124–136.

35 Lupas R. and Verdu S. (1989), Linear multi-user detectors for synchronous code-division multiple-access channels, *IEEE Trans. Info. Theory*, vol. 35, no. 1, pp. 123–136.

36 Xie Z., Short R.T. and Rushforth C.K. (1990), A family of suboptimum detector for coherent multi-user communications, *IEEE JSAC*, vol. 8, no. 4, pp. 683–90.

37 Hong M., Madhow U. and Verdu S. (1994), Blind adaptive multi-user detection, *IEEE Trans. on Commun.*, vol. 42, no. 12, pp. 3178–3188.

38 Bang S.H. and Sheu B.J. (1996), A neural network for detection of signals in communication, *IEEE Trans. on Circuit and Systems, Part I*, vol. 43, pp. 644–655.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Previous | Table of Contents | Next

Chapter 10
Neural Networks for Process Scheduling in Communication Systems

S. Cavalieri, O. Mirabella
University of Catania
Faculty of Engineering
Institute of Informatic and Telecommunications
Italy

Scheduling problems occur in a number of application areas whenever different activities have to be sequentially ordered to perform a pre-established task. A scheduling problem, which frequently occurs, is to assign a single resource to several users, and in this case the main aims are maximum exploitation of the resource and satisfying the requirements of the various users.

An example of this kind of scheduling is access to the physical channel by the various processing nodes in a common-bus computer network. In this case, the aim of scheduling is essentially the respect of the time constraints of the users in each node [1, 2]. These time constraints may refer to throughput (i.e., the amount of information transmitted on the physical channel per time unit) or to the time required for the information transmitted to be delivered to the destination processing node.

Scheduling access to the physical channel in a computer network is an extremely important problem in applications such as process control, which are often critical and delays in delivery may lead to errors in the synchronization between cooperating processes, with serious consequences for their dynamics. In this area, common-bus communication systems, called FieldBuses, are widely used nowadays to connect sensors and actuators with regulation and control devices to create regulation loops [3, 4]. The best-known FieldBuses, which use scheduling, are the FIP network [5] and the IEC/ISA FieldBus standard [6]. They are characterized by centralized access control: a particular Master station manages access to the physical channel by sending the various stations authorization for the transmission of information.

Generally, the Master station uses a scheduling table containing the transmission instants of the information

produced by the processes connected to the communication system. The transmission instants are determined in such a way that the time constraints of the various control processes are satisfied. On the basis of the scheduling table, the Master authorizes the various transmissions contained in it.

Sometimes, the size of the scheduling table may be extremely large, thus creating great memorization problems. These problems are particularly important in control systems in which simple devices, i.e., ones with little memory, are used. For this reason, the authors have developed an innovative method which allows the size of the table to be reduced greatly, but complicates the problem of scheduling as it requires the use of a computationally complex algorithm to determine the compressed scheduling table [7].

Certain classes of neural networks (the Hopfield-type neural model, for example) are able to solve exhaustive NP-hard constrained optimization problems, reducing the computation time as compared with algorithmic solutions. For this reason, they seem to be suitable to solve the process scheduling problem [8, 9].

In this chapter the authors propose a process scheduling methodology based on a Hopfield-type neural model. The general principles involved in the design of the proposed neural model are discussed in detail. Then, some guidelines for the tuning of the network are given. Finally, the computational complexity of the proposed neural model is evaluated and compared with that of the algorithmic scheduling solution. This comparison highlights how the neural solution overcomes the computational complexity of the exhaustive algorithmic solution, guaranteeing computation times, which are polynomial with respect to the number of processes to be scheduled. This feature allows real-time control process scheduling, which is of fundamental importance in process control systems where the occurrence of new transmission requirements on the part of new or existing control processes has to be promptly followed by a modification in the scheduling table. Any delay in the scheduling of new processes would cause a loss of information which is at times critical for the whole control system.

1. Characteristics of the Processes in a Communication System

This section deals with the exchange of information between processes in a communication system featuring centralized access control. As mentioned in the introduction, in such a system, a Master connected to the common bus has the task of distributing the available bandwidth among processes. The exchange of information between the latter is typically organized on the basis of a producer/consumer relationship.

A process (publisher) produces information and distributes it to one or more consumer processes (subscribers). Production can be either periodic or asynchronous.

In this chapter, only the periodic exchange of information will be dealt with, as it is more time-critical than asynchronous transmission.

A periodic producer process, P_{px}, is characterized by a period, T_{px}, of production and by an instant, r_{px}, at which the first production starts, both based on a time reference which is the same for the whole communication system. The information produced by the publisher is stored in a transmission buffer until the Master station authorizes its transmission on the bus, by means of a token. In this way, the communication system delivers the information produced by the publisher to the consumer process/es requiring it. The information is usually delivered by memorizing it in a reception buffer present in each consumer process. Figure I shows the transmission mechanism, based on the use of buffers (transmission and reception buffers are indicated as Tr and Rx Buffers).

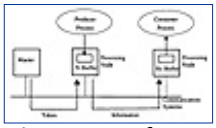


Figure 1 Information delivery to consumer process.

Consumption of the information, which corresponds to reading the contents of the reception buffer on the part of the subscriber process, may have the same period as production or a different one. In the following sections, it will be assumed that the subscriber consumption periods coincide with those of the producer process from which they receive information. On the basis of this hypothesis, a consumer process, P_{cx}, is characterized by the tuple (T_{px}, r_{px}, r_{cx}) where (T_{px}, r_{px}) refer to the time characteristics of the producer process, P_{px}, from which it receives information and r_{cx} is the distance in time between the instants of production and consumption.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Figure 2 shows the production and consumption instants for a process, P_{px}, featuring (4,2) and a consumer process, P_{cx}, featuring (4,2,3). In the figure, production (consumption) instants are drawn by arrows above (below) the time axis. It should be noted that these instants occur in discrete time, synchronized with a reference clock which is common to all the processes in the distributed control system.

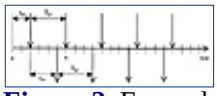


Figure 2 Example of producer and consumer processes.

For each consumer process, P_{cx}, to receive all the information produced by the producer process P_{px}, it is sufficient for the information to be transmitted in each time interval between the production instant and the subsequent consumption instant. This is an open interval in the sense that the information cannot be transmitted at instants which coincide with either the production or consumption instants. In the former case, as production (i.e., writing the information produced in the transmission buffer) cannot be instantaneous, the previous contents of the buffer (i.e., the information produced previously) would be transmitted. In the latter case, the consumer would read the contents of the buffer before it is updated with the new information produced. For this reason, we will henceforward refer to the transmission interval that can be used as an open production/consumption interval, the duration of which is $r_{cx}-1$. With reference to Figure 2, the interval]2,5[is an example of an open production/consumption interval, the duration of which is 1.

Figure 3 shows a correct transmission sequence relating to the pair of processes P_{px}/P_{cx} shown in Figure 2. The black bars in the figure represent the transmissions.

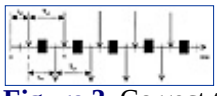


Figure 3 Correct transmission sequence.

As can be seen, we have assumed that each transmission is completed within a time unit. Although, for the sake of simplicity, this assumption will be maintained throughout the chapter, in Section 4.1, the authors present theoretical scheduling results, including transmissions exceeding the time unit.

2. Process Scheduling in a Communication System

From the considerations made so far it follows that a condition for correct scheduling of the processes in a centralized access communication system consists of guaranteeing that for each pair of producer/consumer processes, P_{px}/P_{cx}, at least one transmission is inserted into each open production/consumption interval.

One of the best-known scheduling methods in the literature is the off-line or pre-runtime strategy [10]. In this scheduling method, the transmission instants for each open production/consumption interval are fixed before the network starts operating or before a producer process starts its activity. In order to simplify a priori calculation of all the transmission instants for each pair of producer/consumer processes, this calculation is confined to a restricted time interval known as the macrocycle or scheduling interval, beyond which transmission sequences are periodically repeated.

Let Ts be the duration of this interval, and S_t, the set of transmission instants it contains. This set can be decomposed into $S_t = \bigcup_{x=1}^{p} S_{tx}$, where p is the number of different pairs of producer/consumer processes P_{px}/P_{cx}, and S_{tx}, = $\{t_{x1}, . ., t_{xm}\}$ 4 S_t is the generic subset of transmission instants for the xth pair P_{px}/P_{cx}, such that $\bigcap_{x=1}^{p} S_{tx} = \varnothing$.

Each element t_{xj} S_{tx} (j = 1..m) is determined as follows. Consider the time interval [0,Ts - 1], and let m be the number of consecutive open production/consumption intervals for the pair P_{px}/P_{cx} contained in it. In each one, a transmission is fixed at the instant t_{xj}.

On the basis of the set $S_{tx} = \{t_{x1}, . ., t_{xm}\}$ determined as described above, all the transmission instants for the pair P_{px}/P_{cx} in the whole time span [Ts,+ [are determined by the following relation:

$$t_{xj} + \alpha \cdot Ts, \text{ where } \alpha \in N, \forall t_{xj} \in S_x \ (j = 1..m) \qquad (1)$$

It can be shown that a sufficient condition for any open production/consumption interval for the pair P_{px}/P_{cx} in the time span [Ts, + [to contain at least one of the instants given by (1) is that Ts be a multiple of Tpx [10-12].

If this consideration is extended to the more general case in which there are p producer/consumer process pairs, a sufficient condition for correct scheduling is that the duration Ts be the Lowest Common Multiple (LCM) of all the periods T_{px} (x = 1..p).

2.1 Determination of the Scheduling Sequence Stx

Assuming Ts to be a multiple of T_{px}, the integer quantity

$$n_{px} = \frac{Ts}{T_{px}} \qquad (2)$$

represents the number of open production/consumption intervals of the pair P_{px}/P_{cx} contained in each macrocycle. The scheduling sequence S_{tx} is determined when we fix at least one transmission in each open production/consumption interval. The number of transmissions, n_{tx}, to be fixed in S_{tx} must be

$$n_{tx} = n_{px} = \frac{Ts}{T_{px}} \qquad (3)$$

Determination of the scheduling sequence S_{tx}, for each pair P_{px}/P_{cx}, can be achieved by taking into account a bidimensional matrix of a size $n_{px} \cdot Ts$, with binary values, which we will henceforward call Tbl_x. The j-th row (j = 1..n_{px}) of Tbl_x represents the open interval between a production and the subsequent consumption, into which a transmission has to be inserted.

On the basis of the table, determination of S_{tx} corresponds to identifying a set of transmissions such as to cover all the n_{px} open production/consumption intervals, each represented in a row in the table. Let interval_set be the set of n_{px} open production/consumption intervals. Determination of each transmission instant means the elimination from interval_set of one or more open production/consumption intervals

containing that transmission. Determination of S_{tx} thus ends when the interval_set remains empty.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Table 1 shows an example of a Tbl_x structure considering the pair of processes P_{px}/P_{cx} represented in Figure 2 and Ts = 12. In this case, $n_{tx} = 3$, $n_{px} = 3$ and the interval_set = {1, 2, 3}. Insertion of a transmission at the instant t = 3, for example, means reducing the interval_set to {2, 3}. The choice of S_{tx} = {3, 7, 11} makes this set empty.

Table 1 Example of a Tbl_x structure when Ts = 12, $T_{px} = 4$, $r_{px} = 0$, $r_{cx} = 3$, $n_{tx} = 3$, $n_{px} = 3$

3. Hopfield Neural Network for Process Scheduling

The aim of this section is to illustrate the Hopfield-based strategy used to schedule p producer/consumer processes P_{px}/P_{cx}, considering a macrocycle, featuring a length, Ts, equal to the LCM of the p periods T_{px}. As the Hopfield network is well known, a description is not given here; it can be found in [13 -16].

The neural solution to the scheduling problem was achieved through the following steps: definition of the neural architecture, specification of the surrounding conditions of the scheduling problem to be solved, and calculation of the bias currents and weights for the neural architecture defined. Each of these items will be dealt with in the following subsections, with reference to process scheduling.

3.1 Definition of the Neural Architecture

The definition of the neural architecture (i.e., the number of nodes) strictly depends on the coding chosen for the solution of the problem. In relation to the scheduling problem being dealt with here, it is assumed that the output of each neuron corresponds to a transmission instant in the macrocycle. More specifically, each pair of processes P_{px}/P_{cx}, is associated with Ts neurons relating to the instants of the macrocycle. For each pair of processes, the generic neuron takes a value of 1 if a transmission relating to the pair occurs in the instant it represents. A value of 0, on the other hand, means that no transmission was scheduled for the pair P_{px}/P_{cx} in the instant represented by the neuron. The total number of neurons in the neural model is thus equal to the product of the number of pairs of producer/consumer processes and the length of the macrocycle, i.e., Ts·p. The neurons of the neural model are logically divided into p groups of Ts neurons each. According to this assumption, each group refers to a pair of producer/consumer processes P_{px}/P_{cx} with x [1..p] and contains Ts

neurons, each relating to an instant in the scheduling interval of length Ts.

The matrix in Figure 4 is an example of a possible neural solution to the scheduling of three pairs of producer/consumer processes characterized by (T_{p1} = 3, r_{p1} = 0, r_{c1} = 3), (T_{p2} = 4, r_{p2} = 0, r_{c2} = 4), (T_{p3} = 6, r_{p3} = 0, r_{c3} = 6)) and a scheduling interval of Ts = 12.

The rows of the matrix refer to the pairs of processes, while the columns refer to the instants of the macrocycle. As can be seen, the neural model determines the sets of transmission instants: S_{t1} = {1, 4, 7, 10}, S_{12} = {2, 5, 8}, S_{t3} = {3, 9}.

Figure 4 Example of a neural solution.

In the following, each neuron is specified by a double index, xi or yj, where x,y = 1..p identifies the group to which the neuron belongs, while ij = 0..Ts-1 identifies an instant in the macrocycle. Table 2 summarizes the symbols used to represent the Hopfield model, based on the double index notation.

Table 2 Hopfield-type neural model formalization

3.2 Definition of the Surrounding Conditions

The neural network has to find a solution for the scheduling of p pairs of producer/consumer processes in such a way that all the surrounding conditions are met. These conditions are outlined below.

3.2.1 Number of Transmissions

It is necessary for the solution provided by the neural model to have n_{tx} 1s in the x-th group of neurons, where x [1..p]. As noted in Section 2. 1, n_{tx} represents the number of transmissions that must be fixed in the macrocycle for the pair P_{px}/P_{cx} in order to guarantee its schedulability.

3.2.2 Correctness of the Scheduling of a Single Process

The neural model has to choose the transmission instants contained in each tuple with n_{tx} elements in such a way that correct scheduling of the pair P_{px}/P_{cx} is guaranteed, i.e. that the n_{tx} transmissions will cover all the n_{px} open production/consumption intervals. Let us consider a tuple of $n_{tx} > 1$ transmissions and any two transmission instants, t_{xi} and t_{xj}, belonging to this tuple (i ` j). Let us assume that the transmission instants t_{xi} and t_{xj} are such as not to cover n_0 open production/consumption intervals (i.e., n_0 rows in the matrix Tbl_X). As the number, n, of open production/consumption intervals that can be covered by a single transmission is 1, it follows that, if

$$n_0 > n_{tx} - 2 \qquad (5)$$

holds, choice of the transmission instants t_{xi} and t_{xj} has to be inhibited, as it involves the presence of a number, n_0, of open production/consumption intervals that will never be able to be covered by the remaining (n_{tx} - 2) transmissions.

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

The scheduling solution provided by the neural network thus must not satisfy condition (5). We will now illustrate the strategy we have adopted to enable the neural model to find a scheduling solution taking condition (5) into account. Let d_{xi} indicate the i-th column in the matrix Tbl_x. It is therefore a vector of n_{px} rows and 1 column ($n_{px} \times 1$), in which the 1s identify the open production/consumption intervals for the pair P_{px}/P_{cx} covered by the transmission placed at the i-th instant. Then let us consider the product $d_{xi} \cdot d_{xj}^T$ (where i,j = 1..n_{tx}, i ` j). The integer value provided by this product is zero if the open production/consumption intervals identified by d_{xi} are different from those identified by d_{xj}.

This consideration suggests that the introduction of the term $d_{xi} \cdot d_{xj}^T$ (,ij [1..n_{tx}], i ` j) in the expression of the energy function will privilege determination of tuples with n_{tx} elements such that the sets of open production/consumption intervals covered by each transmission will have a null intersection.

Consider the pair of processes P_{px}/P_{cx} featuring $T_{px} = 6$, $r_{px} = 0$, $r_{cx} = 6$ and a macrocycle characterized by a length of Ts = 12. It follows that $n_{tx} = 2$ and $n_{px} = 2$. The relative Tbl_x is shown in Table 3.

Table 3 Example of a Tbl_x, structure when Ts = 12, $T_{px} = 6$, $r_{px} = 0$, $r_{cx} = 6$, $n_{tx} = 2$, $n_{px} = 2$

The tuples (1,7) (1,8) (1,9), (1,10), (1,11) corresponding to some of the valid scheduling solutions, such that the products $d_{xi} \cdot d_{xj}^T$ between the elements of each tuple are equal to zero. Now let us consider the tuples (1,7) (0,7). The first corresponds to a valid solution, while the second corresponds to a solution that does not guarantee correct scheduling as it does not cover the open production/consumption interval 1. This could have been shown by analyzing condition (5): the value n_0 relating to the tuple (0,7) is equal to 1 and n = 1, thus satisfying condition (5).

Let us now consider the products $d_{xi} \cdot d_{xj}^T$ for the two tuples, which is 0, i.e., it is not possible to identify the valid scheduling solution (1,7). This shows that the criterion proposed for correct scheduling of the x-th pair of processes is not efficient in that it does not allow the neural network to take condition (5) into account in finding a valid scheduling solution.

To overcome this problem, the matrix Tbl_x was modified, considering integer values of D > 1 instead of 0s. In

such a way, again with reference to the tuples (1,7) and (0,7), the products $d_{xi} \cdot d_{xj}^T$ will be $(2 \cdot D)$ for the first tuple and $(D + D^2)$ for the second. Having to minimize the energy function, the neural model can now identify the non-valid tuple (i.e., the one satisfying condition (5)) and discard it.

For the neural network to be able to verify the validity of a scheduling solution, we considered the product given by $v_{xij} \cdot d_{xi} \cdot d_{xj}^T$, where the vector with 1 row and n_{px} columns $(1 \times n_{px})$, v_{xij}, is defined as follows

$$v_{xij}(k) = \begin{cases} D^3 \text{ if } d_{xi}(k) = 1 \text{ and } d_{xj}(k) = 1 & (6) \\ D^2 \text{ if } (d_{xi}(k) = D \text{ or } d_{xj}(k) = D) \text{ and } (n_0 > n_{tx} - 2) & (7) \\ D \text{ if } (d_{xi}(k) = D \text{ and } d_{xj}(k) = D) \text{ and } (n_0 > n_{tx} - 2) & (8) \\ 1 \text{ otherwise} \end{cases}$$

Conditions (6), (7), and (8) aim to inhibit the choice of tuples such that some open production/consumption intervals are not covered.

3.2.3 Correctness of the Scheduling of Several Processes

The conditions illustrated so far constrain the neural network to determine a valid scheduling solution for each pair of processes P_{px}/P_{cx}. It is, however, necessary to introduce a further constraint for the global scheduling of p pairs of processes. The neural network may, in fact, schedule transmissions correctly for all the pairs of processes but two or more pairs may share the same transmission instant. It is therefore necessary to establish that choice of a transmission at instant i for group x will inhibit the choice of a transmission at the same instant for the remaining groups.

3.3 Formalization of the Surrounding Conditions

Hopfield [16] showed that if the weights matrix $W = [w_{xi,yj}]$ is symmetrical and if the function g_{xi} is a steep-like curve (i.e., u_0 ' 0), the dynamics of the neurons described by (4) follow a gradient descendent of the quadratic energy function, also known as the Lyapunov function:

$$E = \sum_x \sum_i \sum_y \sum_j w_{xi,yj} \cdot O_{xi} \cdot O_{yj} + \sum_x \sum_i I_{xi} \cdot O_{xi} \qquad (9)$$

Under the same hypothesis, Hopfield [16] showed that the minima of the energy function (9) coincide at the corners of the hypercube defined by O_{xi} {0,1}.

These theoretical results allow a solution to a particular optimization problem to be obtained from the stabilized outputs of the Hopfield network, by the following method.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

First the surrounding conditions of the optimization problem are expressed in the form of the energy function given by (9). Each term of the energy function referring to a specific surrounding condition is multiplied by a real coefficient weighting the influence of the condition itself on the neural solution. By comparison of the energy function obtained with the function (9), the weights and bias currents are expressed in terms of the above-mentioned real coefficients. In this way, the weights and bias currents are linked to the surrounding conditions of the problem to be solved. If the weights matrix obtained is symmetrical and the function g_{xi} is steep-like, the stable output of the Hopfield model obtained by imposing the weights and biases obtained previously corresponds to a minimum of (9) and thus to a solution to the problem.

In this section, we will formalize the conditions surrounding the scheduling problem described previously, expressing them in the form given by (9). Each surrounding condition term will be linked to a real coefficient (±, ², and ³).

3.3.1 Number of Transmissions

The first condition concerns the presence in each group of neurons x of exactly n_{tx} activated neurons (i.e., set to 1). This constrains the neural network to choose tuples with n_{tx} elements for each pair P_{px}/P_{cx}. The expression of this surrounding condition is

$$\frac{\alpha}{2} \cdot \sum_{x} \left(\sum_{i} O_{xi} - n_{tx} \right)^2 \qquad (10)$$

3.3.2 Correctness of the Scheduling of a Single Process

Another surrounding condition refers to the strategy for choosing the n_{tx} transmissions for each pair P_{px}/P_{cx}. As mentioned, the aim is to choose transmissions which will minimize overlapping between the open production/consumption intervals identified by each transmission, and will not satisfy condition (5), as outlined above. The expression of this surrounding condition is

$$\frac{\beta}{2} \cdot \sum_{x} \sum_{i} \sum_{j \neq i} v_{xij} \cdot d_{xi} \cdot d^{T}_{xj} \cdot O_{xi} \cdot O_{xj} \qquad (11)$$

in which the term $v_{xij} \cdot d_{xi} \cdot d^{T}_{xj}$, as described in the previous section, has been introduced.

3.3.3 Correctness of the Scheduling of Several Processes

Finally, it is necessary to establish that activation of the i-th neuron belonging to group x will inhibit activation of the i-th neuron belonging to any other group y (y [1, p], y ` x). If this were not so, the transmission instants referring to different pairs of processes might coincide. This condition is formulated by the term:

$$\frac{\gamma}{2} \cdot \sum_{x} \sum_{i} \sum_{y \neq x} O_{xi} \cdot O_{yi} \qquad (12)$$

3.4 Determination of Weights and Biases

Let us consider equation (4) concerning the dynamics of the neural model. Taking into account expression (9) of the Lyapunov function, (4) can be rewritten as

$$\frac{dU_{xi}}{dt} = -\frac{U_{xi}}{\tau} - \frac{\partial E}{\partial O_{xi}} \qquad (13)$$

Replacing the expression of the energy function which takes into account the surrounding conditions, obtained by summing the terms described in Section 3.3:

$$E = \frac{\alpha}{2} \cdot \sum_{x} \left(\sum_{i} O_{xi} - n_{tx}\right)^2 + \frac{\beta}{2} \cdot \sum_{x} \sum_{i} \sum_{j \neq i} v_{xij} \cdot d_{xi} \cdot d^{T}_{xj} \cdot O_{xi} \cdot O_{xj} + \frac{\gamma}{2} \cdot \sum_{x} \sum_{i} \sum_{y \neq x} O_{xi} \cdot O_{yi} \qquad (14)$$

(13) becomes

$$\frac{dU_{xi}}{dt} = -\frac{U_{xi}}{\tau} + \alpha \cdot n_{tx} - \alpha \cdot \sum_{j} O_{xj} - \beta \cdot v_{xij} \cdot d_{xi} \cdot d^{T}_{xj} \cdot \sum_{j \neq i} O_{xj} - \gamma \cdot \sum_{y \neq x} O_{yi} \qquad (15)$$

From a comparison of (4) and (15) the following weights and bias currents are obtained:

$$I_{xi} = \alpha \cdot n_{tx} \qquad (16a)$$

$$w_{xi,yj} = -\alpha \cdot \delta_{xy} - \beta \cdot v_{xij} \cdot d_{xi} \cdot d^{T}_{xj} \cdot \delta_{xy} \cdot (1 - \delta_{ij}) - \gamma \cdot \delta_{ij} \cdot (1 - \delta_{xy}) \qquad (16b)$$

As can be seen from (16b), the expression of the weights, $w_{xi,yj}$, depends on the term $v_{xij} \cdot d_{xi} \cdot d^{T}_{xj}$. On the basis of what was said above, the values this term takes vary from one process to another according to the structure of the Tbl_x. This may cause asymmetry in the distribution of the values of weights among the various processes. It may, in fact, happen that the contribution of the term $v_{xij} \cdot d_{xi} \cdot d^{T}_{xj}$ is greater for one process than for another. In such cases, determination of the scheduling solution for the process featuring higher $v_{xij} \cdot d_{xi} \cdot d^{T}_{xj}$ values would be heavily penalized (i.e., the network would tend to place 1s in relation to neurons with smaller weights). For this reason, it was decided to normalize this term between two values, MIN and MAX. More specifically, prior to weights calculation the minimum and maximum value the term $v_{xij} \cdot d_{xi} \cdot d^{T}_{xj}$, i,j, can assume was calculated for each pair of processes P_{px}/P_{cx} (henceforward these values will be indicated as min_x and max_x respectively). The actual value of the term $v_{xij} \cdot d_{xi} \cdot d^{T}_{xj}$ for each pair P_{px}/P_{cx} is then calculated by the relation:

$$v_{xij} \cdot d_{xi} \cdot d^{T}_{xj} = MIN + (MAX - MIN) \cdot \frac{v_{xij} \cdot d_{xi} \cdot d^{T}_{xj} - min_x}{max_x - min_x} \qquad (17)$$

On the basis of the value given by (17), the weights $w_{xi,yj}$ are calculated in accordance with (16b).

4. Reduction in Schedule Length

The considerations made in the previous sections fix the duration, Ts, of the macrocycle in such a way as to guarantee correct transmission of the whole flow of information produced by the periodic processes P_{px} (i = 1..p) connected with the communication system. The choice of Ts as the lowest common multiple of all the

periods T_{px} may lead to a very high Ts value. This certainly happens when the periods T_{px} are prime numbers. In a real application scenario, the co-existence of several control systems means there will be processes with different time characteristics. A number of them usually have production times which share prime factors. They would therefore not require high Ts values. However, the presence of a few prime periods causes the duration Ts to increase drastically.

In a control system having 4 producer processes with the following characteristics, for instance: $T_{p1} = 4$, $T_{p2} = 8$, $T_{p3} = 12$, $T_{p4} = 17$, the first three periods would entail Ts = 24, whereas the period T_{p4} causes an increase in the Ts, which goes up to 408. The high Ts value requires an adequate area of memory as scheduling is managed by consulting a look-up table containing all the times of the scheduling sequence, S_t [17,18].

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

There are not many solutions in the literature to the problem of reducing schedule length. Most that do exist refer to minimizing the maximum completion time of the processes and cannot be applied to our case [19-22]. The absence of such a solution is also accounted for by the fact that, in practice, it is often possible to adjust the periods of tasks so that the LCM of all the periods is reasonable long [12]. In process control applications, however, the periods of the producer and consumer processes correspond to the time characteristics of real processes and so the possibility of varying these values is rather low.

In [7], the authors developed a first solution to this problem. This solution is based on theoretical conditions which guarantee correct transmission, allowing a drastic reduction in the scheduling length Ts. The algorithmic implementation presented in [7] features a high computational complexity, which makes its use difficult.

In order to make the use of the solution presented in [7] easier, the authors developed a neural-based implementation, which will be presented in Section 5.

In the following, the fundamentals of the solution presented in [7] will be explained.

4.1 Theoretical Results

Let us consider a generic producer process, P_{px}, characterized by the period T_{px}. If the duration, Ts, of the scheduling interval is not a multiple of the period, in each interval $I_{\pm} = [\pm \cdot Ts, (\pm + 1) \cdot Ts - 1]$, where $\pm \quad N$, there will be a shift of Tsc_x between the instants of production or consumption and the beginning of the interval $I_{\pm}$, given by

$$\begin{cases} Tsc_x = Ts \bmod T_{px} & \text{if } \left(\frac{Ts}{T_{px}} - \left\lfloor \frac{Ts}{T_{px}} \right\rfloor\right) \le 0.5 \\ Tsc_x = T_{px} - Ts \bmod T_{px} & \text{otherwise} \end{cases} \qquad (18)$$

Having then defined

$$k_x = \frac{LCM(Ts, T_{px})}{Ts} \qquad (19)$$

where LCM(Ts,T_{px}) is the lowest common multiple of Ts and T_{px} the instants of production or consumption and the beginning of the interval $I_{\pm}$ transmission will resynchronize after k_x intervals $I_{\pm}$.

Figure 5 shows an example of this. In this case the macrocycle has a duration of Ts = 8 and the processes P_{px}, P_{cx} are characterized by the pairs T_{px} = 3, r_{px} = 0, and T_{px} = 3, r_{px} = 0, r_{cx} = 3, respectively. We therefore have Tsc_x = 1 and k_x = 3. As can be seen, each interval $I_{\pm}$ (with ± e 0) contains 3 production and 3 consumption instants. In the interval [0,Ts - 1], the production at instant 0 coincides with the beginning of I_0. In each of the subsequent intervals $I_{\pm}$ (with ± > 0), the shift between the corresponding production instant and the beginning of the same intervals $I_{\pm}$ is equal to Tsc_x, until they resynchronize after k_x intervals $I_{\pm}$. The same happens for the consumption intervals.

Let us fix a transmission in each of the open production/consumption intervals in the range [0,Ts - 1]. Figure 6 shows the transmissions at instants 1,4,7. In the same figure, transmissions in the interval [Ts, + [obtained by the application of (1) are represented. The black bars represent the transmissions in the macrocycle, while the white bars are those obtained by (1).

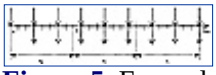


Figure 5 Example of resynchronization after K_x intervals $I_{\pm}$.

On account of the shift Tsc_x, some transmissions given by (1) may produce an invalid schedule. For example, in Figure 6, some transmissions in the range [Ts, + [coincide with the production or consumption instants.

From this example, it is clear that correct scheduling is guaranteed if there is at least one transmission in each open production/consumption interval in the space of k_x intervals I±. Let us now formalize this condition.

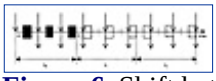


Figure 6 Shift between interval I± and transmissions.

Let n_{px} be redefined as the total number of productions by the process P_{px} in k_x subsequent intervals $I_{\pm}$:

$$n_{px} = \frac{k_x \cdot Ts}{T_{px}} \qquad (20)$$

Note that n_{px} given by (20) coincides with that given by (2) when Ts is a multiple of T_{px}. In this case, in fact, k_x is equal to 1.

If t_{pxj} and t_{cxj} indicate the generic instants of production and subsequent consumption relating to the producer process P_{px} and the consumer process P_{cx}, correct scheduling of all the transmissions relating to the pair of producer/consumer processes P_{px}/P_{cx} implies determination of the subset $S_{tx} = \{t_{x1}, . ., t_{xm}\}$ 4 S_t such that

$$\forall j \in [1,n_{px}] \; \exists \, t_{xi} \in S_{tx}, \; \exists \, m \in [0, k_x - 1] : t_{xi} + m \cdot Ts \in \;] t_{pxj}, t_{cxj}[$$

Let us redefine

$$n_{tx} = \left\lceil \frac{Ts}{T_{px}} \right\rceil \qquad (21)$$

which represents the maximum number of productions by the process P_{px} in each interval $I_{\pm}$. It must be pointed out that the definition of n_{tx} coincides with (3) when Ts is a multiple of T_{px}.

Theoretically, the number of transmissions to be inserted in the subset S_x should also be equal to n_{tx}, for each pair of processes P_{px}/P_{cx}. In many cases, in order to guarantee correct scheduling, it may be necessary to insert a number of transmissions exceeding n_{tx}. In Figure 7, we consider the pair P_{px}/P_{cx} featuring by (9,0) and (9,0,4), respectively. In order to guarantee correct scheduling, we fixed transmissions at the instants 1, 5, 7, 10, even if n_{tx} = 2.

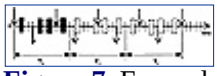


Figure 7 Example of scheduling with more than n_{tx} transmissions.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Below, we will give a theorem which provides a sufficient condition for correct scheduling with only n_{tx} transmissions. It will first be formulated considering unitary transmissions. Then the theorem will be extended to transmissions exceeding the time unit. Both theorems are demonstrated in [7].

Theorem 1: A sufficient condition for correct scheduling of the transmission with a unit value of a pair of producer/consumer processes, P_{px}/P_{cx}, through only n_{tx} transmissions in each interval $I_{\pm}$ is that

Figure 8 gives an example of what is set out in the theorem. It refers again to a scheduling interval of length Ts = 12, and a producer process, P_{px}, characterized by the pair (9,0), but considers a consumer process characterized by (9,0,7). As seen before, $k_x = 3$, $Tsc_x = 3$, $n_{tx} = 2$, $n_{px} = 4$.

In this case, condition (22) is satisfied, and the number of transmissions in the scheduling sequence S_{tx} can be fixed equal to 2. As can be seen, they are placed at instants 5 and 9.

The previous theorem referred to the hypothesis of unit transmissions. We will now formulate the sufficient condition for scheduling with a transmission width, c_x, greater than or equal to 1.

Figure 8 Example of scheduling with ntx transmissions.

Theorem 2: A sufficient condition for correct scheduling of the transmission with a duration c_x (e1) by a pair of producer/consumer processes, P_{px}/P_{cx}, through only n_{tx} transmissions in each interval $I_{\pm}$ is that

Theorems 1 and 2 establish sufficient conditions referring to a generic pair of, producer/consumer processes and allow a suitable Ts value to be fixed for them. The minimum Ts value guaranteeing correct scheduling for all the pairs of processes is one which satisfies these sufficient conditions simultaneously.

4.2 Determination of the Scheduling Sequence S_{tx}

Determination of the scheduling sequence S_{tx}, for each pair P_{px}/P_{cx}, can be achieved by taking account of the intersections of the projections on [0,Ts - 1] of the open production/consumption intervals identified in k_x intervals $I_{\pm}$. In order to take these projections into account, we considered the Tbl_x bidimensional matrix of a size $n_{px} \cdot Ts$, introduced in Section 2.1.

The j-th row ($j = 1..n_{px}$) of Thl_x represents the open interval between a production and the subsequent consumption, into which a transmission has to be inserted:

where c [0, $k_x \cdot (Ts - 1)$].

As noted in Section 2.1, determination of S_{tx} corresponds to identifying a set of transmissions such as to cover all the n_{px} open production/consumption intervals, each represented in a row in the table.

Table 4 shows an example of a Tbl_x structure when Ts = 21, $T_{px} = 9$, $r_{px} = 0$, $r_{cx} = 8$, $n_{tx} = 3$, $n_{px} = 7$. In this case, the interval_set = {1, 2, 3, 4, 5, 6, 7}. Insertion of a transmission at the instant t = 7, for example, means reducing the interval_set to {2,3,5,7}. The choice of S_{tx} = {7,16,19} makes this set empty.

Table 4 Tbl_x structure when Ts = 21, $T_{px} = 9$, $r_{px} = 0$, $r_{cx} = 8$, $n_{tx} = 3$, $n_{px} = 7$

5. Hopfield Neural Network for Process Scheduling with Reduction in Schedule Length

The aim of this section is to illustrate the Hopfield-based strategy used to schedule p producer/consumer processes P_{px}/P_{cx}, considering a macrocycle with a length, Ts less than the LCM of the p periods T_{px}. The neural approach is quite similar to that introduced in Section 3. For this reason, only the modifications introduced to schedule p processes considering a reduced schedule length will be highlighted in the following subsections. These modifications only refer to the definition of the surrounding conditions and concern the correctness of the scheduling of a single process. The other surrounding conditions are the same as those seen in Section 3.

5.1 Correctness of the Scheduling of a Single Process

If the length Ts of the macrocycle is chosen in such a way that condition (22) is satisfied for each pair of processes P_{px}/P_{cx}, the correct scheduling for the generic pair P_{px}/P_{cx} is given by a tuple of n_{tx} transmission instants. So it is necessary for the solution provided by the neural model to have n_{tx} 1s in the x-th group of neurons, where x [1..p].

The neural model has to choose the transmission instants contained in each tuple with n_{tx} elements in such a way that correct scheduling of the pair P_{px}/P_{cx} is guaranteed, i.e., that the n_{tx} transmissions will cover all the n_{px} open production/consumption intervals.

Let us consider a tuple of $n_{tx} > 1$ transmissions and any two transmission instants, t_{xi} and t_{xj}, belonging to this tuple (i ` j). Let us assume that the transmission instants t_{xi} and t_{xj} are such as not to cover n_0 consecutive open production/consumption intervals (i.e. n_0 consecutive rows in the matrix Tbl_x). It can be shown [7] that the number n of consecutive open production/consumption intervals that can be covered by a single transmission is:

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

This means that a very high ± value forces the network to set n_{tx} neurons for each group x to 1 from the start of the neural iteration. This has its drawbacks. Although the neurons activated from the start of the neural iteration correspond to a valid scheduling solution for the pair P_{px}/P_{cx}, overall scheduling of the p pairs of processes may not be achieved, because two or more processes may share the same transmission instant. In this case, the network has to succeed in modifying the scheduling solution for one or both of these pairs of processes so that the correct scheduling conditions for each pair of processes are satisfied without transmissions overlapping. Equation (32) establishes (and it has also been experimentally demonstrated) that a very high ± value does not allow the network to modify the positions of neurons activated from the start of the neural iteration. The term ± should be relatively small so as to allow a certain flexibility in positioning the n_{tx} neurons for each process, thus completely avoiding overlapping of the transmission instants for the pair P_{px}/P_{cx}.

As far as choice of the parameters ² and ³ is concerned, the tests performed showed that an increase in ² has a limit linked to the presence of the term ³. An infinite increase in ² means an increasing reduction in the effect of ³, i.e. non-valid solutions may be determined by the neural network. For the p pairs of processes P_{px}/P_{cx} not to share the same transmission instants (i.e. for the condition linked to ³ to be respected), the relation $\gamma > \beta \cdot \max_{x,i,j}(v_{xij} \cdot d_{xi} \cdot d^{T}_{xj})$ must always hold. The upper limit for the term ² is therefore $\beta < \frac{\gamma}{\max_{x,i,j}(v_{xij} \cdot d_{xi} \cdot d^{T}_{xj})}$. On the basis of the normalization performed by (17), the upper, limit for the term ² can be rewritten as $\beta < \frac{\gamma}{MAX}$.

6.1 Example of Neural Scheduling

In the tests performed, the values of the terms ±, ² and ³ were fixed according to the guidelines mentioned previously, as follows: ± = 100, ³ = 500, ² = 4, and considering a value of 5 for D in each Tbl_x. The values of MIN and MAX were fixed to 1 and 50, respectively. In all the tests performed the network converged to states corresponding to correct scheduling solutions within 100 to 300 iterations.

Figure 9 shows the neural solution to the scheduling problem in which p = 5. Table 6 summarizes the time characteristics of the producer/consumer processes considered. On the basis of these characteristics, condition (22) fixes the value of Ts = 12.

Figure 9 Neural solution to the scheduling problem in which p = 5 and Ts = 12

Table 6 Time characteristics

x	T_{px}	r_{px}	r_{cx}	n_{tx}	n_{px}
1	5	0	5	3	12
2	8	0	8	2	3
3	8	1	7	2	3
4	9	0	9	2	4
5	9	2	8	2	4

Table 7 summarizes the open production/consumption intervals covered by each set S_{tx} with n_{tx} elements, determined by the neural network as shown in Figure 9. As can be seen, for each pair of processes, all the open production/consumption intervals are covered. It is important to point out that the neural network chooses each tuple of transmissions in such a way as to make the intersection between the open production/consumption intervals covered by each transmission of the tuple for each pair of processes P_{px}/P_{cx} null.

Table 7 Open production/consumption intervals covered in the scheduling solution.

x	n_{tx}	S_{tx}	Open production/consumption intervals covered by each element of S_{tx}
1	3	1	1-3-8-10
		5	4-6-9-11
		9	2-5-7-12
2	2	3	1-2
		8	3
3	2	4	1
		11	2-3
4	2	0	2-3
		6	1-4
5	2	2	2-3
		7	1-4

Although the number of pairs of processes is low, the example proposed in Figure 9 stresses the available bandwidth considerably as the total number of transmissions to be inserted into the macrocycle with a length of 12 is 11. Another consideration concerns the length of the macrocycle, which is drastically shorter than the LCM of all the five processes considered. The theoretical condition shown in Section 4 made it possible to reduce the value of Ts from 360 to 12 time units.

7. Neural versus Classical Scheduling

Verification of the capacity of the neural model proposed to schedule several processes is not enough to demonstrate the power of the neural solution. It is necessary to compare its performance with that of an algorithmic scheduling implementation. Such a comparison cannot be made on the basis of the quality of the scheduling solutions provided by the two approaches, as they both ensure correct scheduling. It will therefore be made by assessing the order of magnitude of the time required to obtain a solution in the two approaches, with reference in both cases to the most onerous computational conditions.

Evaluation will refer to the programming complexity of the neural model and the computational complexity of a classical scheduling algorithm. The comparison between these two complexities seems to be the only significant one, as was done, for example, in [23, 24].

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

7.1 Programming Complexity

The computation time needed by the neural model to obtain a valid scheduling solution is expected to be very short, if we refer to a parallel hardware implementation of the model. There are several well-known works in the literature addressing the problem of the hardware implementation of neural networks [25-29]. In [29], for instance, the (analog or digital) implementation of the Hopfield network is discussed, showing that analog implementation guarantees low processing times and allows the greatest integration (with current analog technology it is possible to integrate several hundreds of fully interconnected neurons, although a great deal of work remains to be done on the integration of larger neural networks). The use of a hardware implementation of a Hopfield network drastically reduces the time required to compute the neural output, given the high calculation power obtainable (about 10^{11} connections processed per second [25]). The time required for the network to converge, and thus to reach a solution to the optimization problem, may be longer. It essentially depends on the number of iterations needed to reach a minimum of the energy function. In [30], it was shown that the number of iterations is always very limited (a few hundred iterations) and is practically independent of the complexity of the problem (i.e., the number of neurons in the network). These results were confirmed during the tests carried out here: as stated before, valid solutions were always obtained within 300 iterations at most. The low number of iterations needed to reach a neural solution allows us to neglect the time required to calculate the solution as opposed to that relating to programming complexity.

Programming complexity refers to calculation of the matrix of weights and bias currents linked to the optimization problem to be solved. It should be borne in mind that the neural solution requires an initial phase in which the weights $w_{xi,yj}$ and the bias currents I_{xi} are calculated. It is reasonable to assume that these are calculated off-line by an algorithm. In that case, the computation time required for the neural model to find a solution practically coincides with the time needed to determine the weights matrix $W = [w_{xi,yj}]$ and the vector of bias currents $I = [I_{xi}]$.

This calculation is made before the neural iteration and involves the following operations:

- Calculation of Tbl. This matrix is defined as $Tbl = [Tbl_x]$ (x = 1..p). It is calculated by determining each matrix Tbl_x for each process. This consists of identifying the open production/consumption intervals (featuring the presence of 1s) for each of the n_{px} rows of the matrix. The complexity of this calculation is therefore proportional to both the number of rows (n_{px}) and the number of columns (Ts), i.e., to $Ts \cdot n_{px}$. Calculation of the whole table Tbl is also proportional to the number of processes (p).

The programming complexity is therefore proportional to ($Ts \cdot n_{px} \cdot p$).

- Calculation of the minimum and maximum of $v_{xij} \cdot d_{xi} \cdot d^{T}_{xj}$. This is done for each Tbl_x and its aim is to normalize the term $v_{xij} \cdot d_{xi} \cdot d^{T}_{xj}$ present in (16b). In Section 3.4, the formula used for normalization was given (see (17)), and the bounds (MIN and MAX) of the normalization interval adopted were indicated. This calculation has a complexity proportional to Ts^2 for each process, and a complexity proportional to ($p \cdot Ts^2$) for the total number of processes.
- Calculation of the bias currents. The programming complexity is proportional to the number of neurons, i.e., ($p \cdot Ts$).
- Calculation of the weights. This can be done on the basis of the Tbl calculated previously and has, a, programming complexity which is proportional to the size of the weights matrix W, i.e., $(p \cdot Ts)^2$.

From the above discussion, it can be seen that the programming complexity is heavily affected by calculation of the weights matrix, that is, it is proportional to

$$(p \cdot Ts)^2 \qquad (33)$$

7.2 Computational Complexity

Computational complexity refers to the computation time needed to obtain a solution through an algorithm. In the following, computational complexity will be analyzed considering process scheduling in a macrocycle featuring a length equal to the LCM of the periods and in a macrocycle whose length is less than the LCM of the periods.

7.2.1 Process Scheduling When the Length of the Macrocycle is Equal to the LCM of the Periods

Process scheduling in centralized access communication systems can be seen as part of the more general problem of the scheduling of periodic tasks in operating systems: the aim of scheduling is to determine the order in which one or more resources are assigned to particular periodic tasks in such a way as to meet their time constraints. In the literature, the timing constraints of a periodic task, P_x, are known as:

- T_x, which represents the task period.
- R_x, which represents the release time, i.e., the instant starting from which the task can be assigned to the resource.
- D_x, which is the deadline, i.e., the instant by which the task has to be assigned to the resource.
- C_x, which is the runtime, i.e., the time interval during which the task is assigned to the resource.

The aim of scheduling is to assign each task, P_x, to the resource no earlier than its release time and to complete it no later than its deadline. This condition has to be met in each task repetition period.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Scheduling in centralized access communication systems involves the same problems as task scheduling. Given a producer process P_{px} characterized by (T_{px}, r_{px}) and a corresponding consumer process P_{cx}, characterized by (T_{px}, r_{px}, r_{cx}), scheduling of their respective transmissions corresponds to a task scheduling with $T_x = T_{px}$, $R_x = r_{px}$, $D_x = r_{cx}$. The parameter C_x corresponds to the time units, c_x, required to transmit the information.

Classical task scheduling solutions vary according to the conditions surrounding the problem (the reader is referred to [31] for details of the best-known classical solutions to the task-scheduling problem). We will now discuss the conditions surrounding the problem of transmission scheduling in communication systems with centralized access control.

In such systems, the time characteristics of the periodic processes involved are known a priori. The transmission instants of all the periodic processes can therefore be determined off-line, before the communication network starts functioning. This situation corresponds, in the field of task scheduling, to static scheduling or pre-runtime scheduling [12], as opposed to dynamic scheduling (also called runtime), in which a scheduler process chooses the instant at which to transmit a value each time it is produced. Runtime scheduling takes into account the time characteristics of the producer and consumer processes, generally assigning each process a priority when allocating resources. This does not always ensure that all the time constraints are respected. There is, in fact, always the possibility that a process will miss its deadline or make others miss it [12]. Runtime scheduling is, however, flexible as regards variations in the number and time characteristics of the producer and consumer processes. This makes it suitable for application scenarios where the time characteristics cannot be known a priori [18].

Transmission is obviously not pre-emptive, in the sense that once it has begun, it cannot be interrupted. This condition corresponds to non-pre-emptive task-scheduling solutions, as opposed to pre-emptive ones in which assignation of a task to a resource can be interrupted and resumed afterward.

A communication system with centralized access control can be seen as a resource shared by several producer and consumer processes. This corresponds to task-scheduling scenarios in which there is only one resource to be assigned to several tasks.

Last, in centralized access control communication systems, no assumptions can be made a priori about the values of T_{px}, r_{px}, r_{cx}. This corresponds to task-scheduling solutions with arbitrary release times and deadlines.

To summarize, the surrounding conditions of task scheduling which coincide with transmission scheduling in centralized access control communication systems are: pre-run, non-pre-emptive, uni-resource, and arbitrary release and deadline times.

In the literature, there are well-known methods such as the cyclic executive method [10,12], by which pre-runtime scheduling can be done manually if the number of processes is very low and if most processes have uniform characteristics. As pointed out in [12], however, these methods cannot be applied to a scheduling problem like the one being dealt with here, with a relatively large number of processes that have strong time constraints and feature different time characteristics and interdependencies.

There is one solution in the literature for such time-critical task scheduling, which meets all the surrounding conditions pointed out. Proposed by Garey et al. in [32], it consists of an algorithm which, given n tasks, determines a feasible schedule. In the case of periodic tasks, this algorithm can fix a schedule for the instances of a given set of tasks within a time interval ranging between 0 and the LCM of the task period. When applied to a centralized access control communication system, this algorithm gives a set, S_t, of transmissions to be scheduled in the interval with a length of Ts, equal to the LCM of all the periods being considered.

As regards the computational complexity of this algorithm, in [32] it is shown to be proportional to (p^2). Again in [32], it is shown that this computational complexity can be improved through, for example, careful use of appropriate data structures. In this case, it is proportional to

$$(p \cdot \log p) \qquad (34)$$

7.2.2 Process Scheduling When the Length of the Macrocycle is Less Than the LCM of the Periods

The algorithmic solution to the scheduling problem when the length of the macrocycle is less than the LCM of the periods is based on an exhaustive search for a valid solution. This search consists of considering a tuple of n_{tx} transmission instants for each pair of processes P_{px}/P_{cx} that will satisfy the condition for correct scheduling. The p tuples obtained have to be compared in order to see whether they have a null intersection, i.e., that the p tuples do not contain the same transmission instants. If they do, a new set of p tuples has to be considered. In the worst case, the computational complexity of the algorithm relates to calculation of all the permutations of the p tuples with n_{tx} elements for the pair P_{px}/P_{cx}. As the number of possible tuples with n_{tx} elements that can be calculated for each pair is given by the combination of n_{tx} classes:

$$C_{Ts,n_{tx}} = \binom{Ts}{n_{tx}} = \frac{Ts!}{(Ts - n_{tx})! \, n_{tx}!} \qquad (35)$$

the total number of possible permutations is

$$\prod_{x=1}^{p} C_{Ts,nt_x} = \prod_{x=1}^{p} \binom{Ts}{n_{tx}} = \prod_{x=1}^{p} \frac{\prod_{i=0}^{n_{tx}-1} (Ts - i)}{n_{tx}!} \qquad (36)$$

In order to assess this computational complexity, we will assume that $n_{tx} = n_t \times [1..p]$. On the basis of this assumption, the computational complexity is proportional to

$$\left(\frac{\prod_{i=0}^{n_t-1} (Ts - i)}{n_t!} \right)^p \qquad (37)$$

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

7.3 A Comparison Between Computational and Programming Complexity

The comparison between (33) and (34) highlights that the performance of the neural approach is comparable with that of a classical algorithmic solution, when the length of the macrocycle is equal to the LCM of the periods. In other words, from a computational point of view, neural scheduling does not show remarkable advantages, but it only represents a valid alternative for process scheduling in centralized access communication systems.

A comparison between the two complexities given by (33) and (37) points out that the complexity of the neural approach is decidedly below that of the classical algorithm. So, in this case the use of the neural solution shows a remarkable advantage in terms of computational time.

8. Real-Time Neural Scheduling Strategy

The results given in the previous section allow us to work out a real-time scheduling strategy based on the use of the Hopfield neural model. The strategy assumes that hardware implementation of the Hopfield-type neural model presented here is used. This assumption is realistic since, as was said previously, there are a number of works in the literature which demonstrate the hardware implementability of the Hopfield model. To make scheduling flexible to modifications in the number of processes and in the size of the macrocycle, it is assumed that the hardware implementation of the neural model is reasonably oversized with respect to the number of processes, p, and the length of the scheduling table, Ts. In [9], the oversizing of the model is dealt with in detail.

The neural model proposed in this chapter is used to determine the scheduling table off-line, before the process control system starts functioning. During the evolution of the system, a new process may be added or an existing one may change its time characteristics. In the latter case, the transmissions referring to a process which has undergone a modification in its time characteristics are eliminated from the scheduling table and the modified process is considered as a new process to be scheduled. In both cases, therefore, the scheduling table has to be modified so as to include the new transmission requirements. This can be achieved in two different ways. The simplest solution is to maintain the old scheduling table and only add the new transmissions. In [9] a algorithmic solution to reach this aim is presented, and it is shown that the maximum amount of time the algorithm takes to update the scheduling table on-line is proportional to TS^3. The algorithm only fails if the bandwidth is not sufficient for the insertion of new transmissions. It should be

pointed out that failure on the part of the algorithm can be foreseen before it is executed by appropriate analysis of the amount of bandwidth available. The schedulability of the new process can thus be assessed without useless on-line execution of the algorithm. If it is impossible to include the scheduling of a new process (i.e. the relative transmissions) in the existing macrocycle it is necessary to reschedule all the control system processes. In this case, condition (22) or (23) determines the new Ts value for scheduling all the processes. The new schedule cannot not be achieved using the exhaustive algorithmic solution on account of its NP-hard complexity, but the neural model can be used. As mentioned above, the maximum computation time with hardware implementation of the Hopfield network coincides with the programming complexity of calculation of the weights and bias currents. This complexity is given by $(p \cdot Ts)^2$.

Hence, it follows that in both cases (on-line algorithm or neural model) the time required for the proposed scheduling strategy to adapt to changes in the control system is always quite short. The authors thus consider that the neural scheduling solution can be implemented in real-time, providing adaptability to changes in manufacturing systems.

9. Conclusion

The chapter has discussed the advantages of using neural techniques in process control scheduling. The particular scheduling problem dealt with concerns process scheduling within a time interval, known as a macrocycle, whose length is less than the Least Common Multiple of all the periods. It has been shown that it can be solved by an exhaustive strategy which explores all the possible solutions until it finds the optimal one. For this reason, it is difficult to solve using traditional programming, especially when the number of processes to be scheduled is very high.

The Hopfield-type model devised succeeds in providing correct scheduling solutions in times which are proportional to the size of the problem itself. This represents a considerable advantage in real use of the model, as its computation times are always bounded, even when there are a large number of processes. This allows the scheduling table to be modified in real time when new periodic processes are added to the process control system or when existing processes modify their time requirements. In this way, dynamic process scheduling can be achieved, making the scheduling adaptive to process control features. It is important to point out that scheduling table updating is not a frequent event in a process control system. However, fast adaptation is required in order to allow reconfiguration after control system faults or after a sudden change in the dynamics of some critical industrial processes.

Previous | Table of Contents | Next

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

References

1 N. Malcom, W. Zhao, (1992) Advances in Hard Real-Time Communications with Local Area Networks, *Proceedings of 17th Local Computer Network*, Minneapolis, USA, pp. 548–557.

2 J.W. Park, H.G. Park, W.H. Kwon, (1994) A Real-Time Communication Protocol with Contention-resolving Algorithm for Programmable Controllers, *Proceedings IECON'94*, Bologna, Italy, pp. 1159–1164.

3 P. Pleinevanx, J.D. Decognie, (1988) Time Critical Communication Networks: Field Buses, *IEEE Network*, vol.2, no.3.

4 J.R. Pimentel, (1990) Communication Networks for Manufacturing, Prentice-Hall International.

5 French Association for Standardization, (1990) Bus FIP for Exchange of Information between Transmitters, Actuators and Programmable Controllers, *NF C46*, documents 601–607.

6 IEC 65C WG6, Digital Data Communications for Measurement and Control Working Group 6: FieldBus Standard for Use in Industrial Control Systems, Part 1: Introductory Guide.

7 S. Cavalieri, A. Di Stefano, O. Mirabella, (1995) Pre-Run-Time Scheduling to Reduce Schedule Length in the FieldBus Environment, *IEEE Transaction on Software Engineering*, vol.21, no. 11, pp. 865–887.

8 S. Cavalieri, O. Mirabella, (1995) Using Neural Networks to Reduce Schedules in Time-Critical Communication Systems, *Proceedings World Scientific 7-th Italian Workshop on Neural Nets*, Vietri sul Mare, Salerno, Italy, pp. 177–182.

9 S. Cavalieri, O. Mirabella, (1996) Neural Networks for Process Scheduling in Real-Time Communication Systems, *IEEE Transactions on Neural Networks*, vol.7, no.5, pp. 1272–1285.

10 S. Cheng, J.A. Stankovic, (1988) Scheduling Algorithms for Hard Real-Time Systems A Brief Survey, *Hard Real-Time Systems*, IEEE Press, pp. 88–173.

11 P. Raja, G. Noubir, (1993) Static and Dynamic Polling Mechanisms for Fieldbus Networks, *Operating Systems Review*, ACM Press, vol. 27, no. 3, pp. 34–45.

12 J. Xu, D.L. Parnas, (1993) On Satisfying Timing Constraints in Hard-Real-Time Systems, *IEEE Transactions on Software Engineering*, vol. 19, no. 1.

13 J.J. Hopfield, (1992) Neural Networks and Physical Systems with Emergent Collective

Computational Abilities, *Proceedings National Academy of Sciences*, vol. 79, pp. 2554–2558.

14 J.J. Hopfield, D.W. Tank, (1986) Neural Computations of Decisions in Optimization Problems, *Biol. Cybern.*, vol. 52, pp. 141–152.

15 D.W. Tank and J.J. Hopfield, (1986) Simple 'Neural' Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit, *IEEE Transactions on Circuits.Syst.*, vol. CAS-33, no. 5, pp. 533–541.

16 J.J. Hopfield, (1984) Neurons with Graded Response Have Collective Computational Properties Like those of two-state Neurons, *Proceedings National Academy of Sciences*, vol. 81, pp. 3088–3092.

17 T. Laine, (1991) Modelisation d' Applications Reparties pour la Configuration Automatique d'un Bus de Terrain, Ph.D. Thesis, Intitut National Polytechnique de Loraine, CRAN.

18 S. Kumaran, J.D. Decotignie, (1989) Multicycle Operations in a Fieldbus: Application Layer Implications, *Proceedings of IECON'89*.

19 E.G. Coffman, Jr., (1976) Computer and Jobshop Scheduling Theory, New York, Wiley-Interscience.

20 M.J. Gonzalez, Jr., (1977) Deterministic Processor Scheduling, *Computing Surveys*, vol. 9, pp. 173–204.

21 E.L. Lawler et al., (1992) Recent Developments in Deterministic Sequencing and Scheduling: a Survey, Proceedings NATO Advanced Study and Research Institute on Theoretical Approaches to Scheduling Problems, Durham, England, July 1981, in Deterministic and Stochastic Scheduling, M.A.H. Dempster et al., Eds, Dordrecht, The Netherlands, D. Reidal Publishing, pp. 35–73.

22 J. Blazerwicz, M. Drabowski, J. Weglarz, (1986) Scheduling Multiprocessor Tasks to Minimize Schedule Length, *IEEE Transactions on Computer*, vol. C-35, no. 5.

23 M.K. Mehmet Ali, F. Kamoun, (1993) Neural Networks for Shortest Path Computation and Routing in Computer Networks, *IEEE Transaction on Neural Networks*, vol. 4, no. 6, pp. 941–954.

24 M. Takeda, J.W. Goodman, (1986), Neural Networks for Computation: Number Representations and Programming Complexity, *Appl.Opt.*, vol. 25, no. 18, pp. 3033–3046.

25 H.P. Graf, L.D. Jackel, (1989) Analog Electronic Neural Networks Circuits, *IEEE Circuit and Devices Magazine*, pp. 44–50.

26 M.A.C. Maher, S.P. Deweerth, M.A. Mahawald, C.A. Mead, (1989) Implementing Neural Architectures Using Analog VLSI Circuits, *IEEE Transaction on Circuits and Systems*, vol. 36, no. 5, pp. 643–651.

27 S.P. Eberhardt, R. Tawel, T.X. Brown, T. Daud, A. Thakoor, (1992) Analog VLSI Neural Networks: Implementation Issues and Examples in Optimization and Supervised Learning, *IEEE Transaction On Industrial Electronics*, vol. 39, no. 6, pp. 552–564.

28 J. Wang, (1995) Analysis and Design of an Analog Sorting Network, *IEEE Transactions on Neural Networks*, vol. 6, no. 4.

29 M. Verleysen, P.G.A. Jespers, (1989) An Analog VLSI Implementation of Hopfield's Neural Network, *IEEE Micro*, pp. 46–55.

30 S.V.B. Aiyer, M. Niranjan, F. Fallside, (1990) A Theoretical Investigation into the Performance of the Hopfield Model, *IEEE Transaction on Neural Networks*, vol. 1, no. 2, pp. 204–215.

31 S. Cheng, J.A. Stankovic, (1988) Scheduling Algorithms for Hard Real-Time, Systems - A Brief Survey, *Hard Real-Time Systems*, IEEE Press, vol. 88, pp. 88–173.

32 M.R. Garey, D.S. Johnson, B.B. Simons and R.E. Tarjan, (1981) Scheduling Unit-Time Tasks with Arbitrary Release Times and Deadlines, *SIAM J. Computing*, vol. 10, pp. 256–269.

Previous | Table of Contents | Next

HOME | SUBSCRIBE | SEARCH | FAQ | SITEMAP | CONTACT US

Industrial Applications of Neural Networks
by Lakhmi C. Jain; V. Rao Vemuri
CRC Press, CRC Press LLC
ISBN: 0849398029 **Pub Date:** 10/01/98

Index

- A -

- B -

- C -

- D -

- E -

Table of Contents

HOME SUBSCRIBE SEARCH FAQ SITEMAP CONTACT US

Products | Contact Us | About Us | Privacy | Ad Info | Home