Pergamon

Neural Networks, Vol. 7, No 1, pp 195–202, 1994
Copyright © 1994 Elsevier Science Ltd
Printed in the USA. All rights reserved
0893-6080/94 $6.00 + 00

CONTRIBUTED ARTICLE

A Neural Network for Error Correcting Decoding of Binary Linear Codes

ANNA ESPOSITO, SALVATORE RAMPONE, AND ROBERTO TAGLIAFERRI

Universitá di Salerno

(*Received* 8 *February* 1993; *accepted* 11 *May* 1993)

Abstract—*We describe a neural net model that can be used to decode code words belonging to a binary linear code and to correct noisy patterns that it receives from a transmission channel. The net reduces the error probability to zero in the range of the error-correcting capacity of the code We show a net simulation done in DISC, a general purpose parallel programming environment based on the CSP model*

Keywords—Neural network, Threshold tuning, Maximum likelihood decoding, Linear codes, Parallel programming.

1. INTRODUCTION

The problem with information transmission (Gallager, 1968; MacWilliams & Sloane, 1981) is that of reproducing at a destination, either exactly or approximately, a message selected at a source. This problem may be faced in two distinct phases (Figure 1):

1. at the source the data are compressed to eliminate redundancy;
2. in the transmission, the channel encoder receives the input data and adds the systematic redundancy to them.

This controls the received information. The criterion is to correct many errors without introducing too much redundancy. One method to achieve this is to use code words that satisfy the constraints of a linear homogeneous equation system. Each error makes a subset of these equation failures, and, knowing which subset is not verified, we can trace the errors and correct them. Linear codes adopt such a correcting criterion (MacWilliams & Sloane, 1981; Gallager, 1968; Peterson & Weldon, 1972). Unfortunately, in the general case the decoding mechanism of these codes is NP-hard (Garey & Johnson, 1979).

In this work, instead of the standard methods, a neural net model is used for decoding binary linear codes on binary symmetrical channels (Gallager, 1968). The model decodes code words belonging to a binary linear code and performs the maximum likelihood decoding on noisy patterns that are received from the channel. This avoids both the multiplication of the received vector by the parity-check matrix (Gallager, 1968) and the creation of the standard array (Gallager, 1968) to identify the most probable error pattern associated with the obtained syndrome (Gallager, 1968). Moreover, the net has the advantages due to the intrinsic parallelism of the neural model, and it is fast and simple to use.

The net was conceived with the aim of developing a neural chip that was programmable once and for all on a code, guaranteeing that if the code is *e*-correcting and a number $\leq e$ of errors occur in the transmission of a code word, then the neural chip corrects them.

Given the characteristics of the model and to validate experimentally the theoretical results, we simulated the model on a distributed-memory multiprocessor system (Athas & Seitz, 1988). We implemented the net (Esposito et al., 1993) using the DISC (DIStributed C) system (Iannello, Mazzeo, & Ventre, 1989; Iannello et al., 1990). DISC is a general purpose parallel programming environment, developed at the Department of Computer Science of the University of Napoli, based on the CSP (Communicating Sequential Processes) model (Hoare, 1978).

The idea of connecting neural nets and error-correcting codes was developed on Hopfield networks

Acknowledgements: This work was supported in part by C.N.R., Progetto Finalizzato "Sistemi Informatici e Calcolo Parallelo" under Grant 92 01 570.PF69, by MURST 40% unità I.N.F.M. Università di Salerno, and by Contratto Quinquennale C.N.R.-I.I.A.S.S. We wish to thank the Professors Eduardo R. Caianiello, Maria Marinaro, and Ugo Vaccaro for their suggestions, the colleagues and friends Michele Mastroianni and Stefano Russo of the Department of Computer Science of the University of Napoli for their help in prototyping the simulation in the DISC environment, and the staff of the Interdepartmental Computing Center of the University of Salerno for all resources given at our disposal.

Requests for reprints should be sent to Salvatore Rampone, Dipartimento di Fisica Teorica e S.M.S.A., Università di Salerno, I-84081 Baronissi (Sa) Italy.

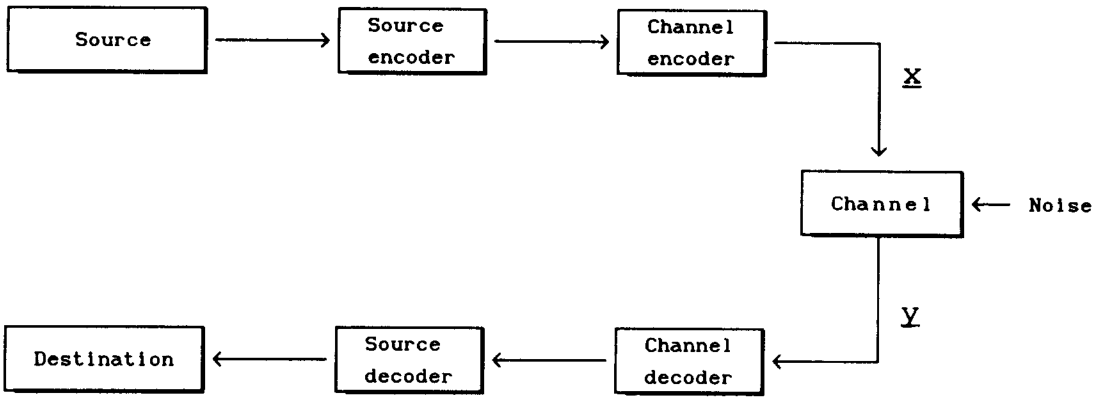

FIGURE 1. Block diagram of communication system. The purpose of the source encoder is to represent the source output by a sequence of binary digits. The purpose of the channel encoder is to allow the binary data to be reliably reproduced at the output of the channel decoder.

(Hopfield, 1982) by Bruck and Blaum (1989). Compared to this net, our model obtains better performances using a lower number of neurons.

This work is organized in the following way. In the next two sections we describe the code and the net models and show the net error-correcting capabilities. In Section 4 we extend the class of suitable codes. In Section 5 we introduce the DISC environment. Then we show the net simulation and the experimental results. The simulation program is reported in the Appendix.

2. THE CODING ENVIRONMENT

An (n, k) binary linear code C is a k-dimensional subspace of the n-dimensional vector space $V_n = [(x_1, x_2, \ldots, x_n) | x_i \in \{0, 1\}]$. Each vector $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in C$ is called code word; n is the length of the code words.

We propose to transmit code words on a Binary Symmetric Channel (BSC) (McElice, 1977) (Figure 2). This channel works on binary input and output sequences, where each digit of the input sequence is correctly reproduced at the channel output with some fixed probability $(1 - \xi)$ and is altered by noise into the opposite digit with probability ξ.

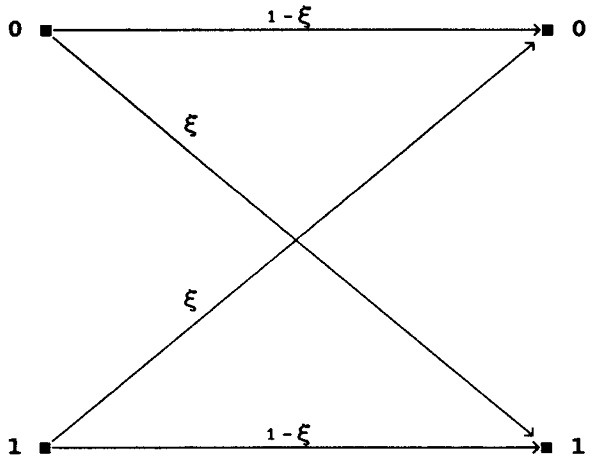

FIGURE 2. Binary symmetric channel (BSC).

If each code word is sent with the same probability and $\xi < \frac{1}{2}$, then the receiver's best strategy for guessing which code word was sent is to perform the maximum likelihood decoding, that is, to pick the code word for which the number of components different from the received vector is smallest. This strategy will be capable of correcting all patterns with at most e errors iff the Hamming distance $d_H(\cdot, \cdot)$ between each pair of code words is greater than or equal to $2e + 1$.

3. THE NEURAL MODEL

The neural net is constituted by $N - 1$ neurons where N is the code word number[1]. The first n neurons of the net take the input from a n-bit buffer. The activation of each neuron i is given by:

$$\mu_i(t + \tau) = 1\left[\sum_{j=1}^{n} a_{ij}\mu_j(t) - \theta_i\right]$$

where

- $1[x]$ is the Heaviside function

$$1[x] = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{if } x <= 0; \end{cases}$$

- $\mu_i(t)$ is the state of the neuron i at time t; $[\mu_i(t) \in \{0, 1\}$ for $1 \leq i \leq N - 1]$;
- θ_i is the threshold of the neuron i, initially fixed to the value $1 - \varepsilon$, where ε is infinitesimal;
- τ is the propagation delay;

[1] In the following we suppose that the number of code words is larger than the length of the code words (n). In the general case N is the maximum between the length of the code words (n) and the code word number.

- a_{ij} is the weight of each edge (j, i), that is, the coupling coefficient from neuron j to neuron i. Each a_{ij} is initially vanishing.

We sign A the coupling coefficient matrix ($A = \{a_{ij}\}$) and in the following we adopt the vectorial form:

$$\mu(t+\tau) = 1[A\mu(t) - \theta] \qquad (1)$$

In this case the function $1[\;]$ is applied to each component of the vector $A\mu(t) - \theta$.

The behaviour of the net is quite simple. The elements set to 1 in the input buffer cause the activation of the corresponding neurons. Then, in the next step, each neuron computes the weighted sum of the input activations and quantizes the outputs to zero or one. The output of each neuron is coupled back to the output buffer (Figure 3).

Now let C be a binary linear code (Gallager, 1968) of N code words

$$\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^{N-1}.$$

Suppose such a code is e-correcting. To learn and decode a code with error-correcting capabilities, the net undergoes two phases:

1. a training phase;
2. a threshold tuning phase.

3.1. Training Phase

We train the net on the code words $\mathbf{x}^1, \ldots, \mathbf{x}^{N-1}$, in this order, with the following learning rule, which fixes the coupling coefficient values once and for all.[2] This rule is based on the Mnemonic Equations (Caianiello, 1961) as modified by Esposito, Rampone, and Tagliaferri (1991).

$$\Delta a_{pk} = \left[\frac{1}{w_H(\mathbf{x}^h)} x_k^h - \beta(1 - x_k^h)\right] * \left(1 - 1\left[\sum_i a_{pi}\right]\right)\left(1\left[\sum_i |a_{p-1,i}|\right]\right). \qquad (2)$$

At each step we train the net on a code word $\mathbf{x}^h$. We see that the coupling coefficients are updated in the following way:

$$a'_{pk} = \begin{cases} \left[\dfrac{1}{w_H(\mathbf{x}^h)} x_k^h - \beta(1 - x_k^h)\right] & \text{if } p = h; \\ a_{pk} & \text{if } p \neq h \end{cases}$$

where

- $w_H(\;)$ is the Hamming weight;
- $\beta = 1/[w_{\max}(C)]$ where $w_{\max}(C) = \max_h\{w_H(\mathbf{x}^h)\}$.

[2] We do not train the net on the code word $\mathbf{x}^0 = \mathbf{0}$. In the following we assume C to be the set $C - \{\mathbf{0}\}$. The code word $\mathbf{0}$ is considered in the decoding phase section

Using a linear code C we find that the following properties for the A matrix, built by eqn (2), are true:

1. $\mathbf{x}^h \in C \Rightarrow \mu = 1[A\mathbf{x}^h - \theta] \neq \mathbf{0}$
2. $\forall \mathbf{x}^h \in C$, only the hth component of μ is different from 0 ($\mu_h = 1$).

In fact:

THEOREM 1. *If* $\mathbf{x}^h \in C$ *then* $\mu = 1[A\mathbf{x}^h - \theta] \neq \mathbf{0}$.

Proof. If $\mathbf{x}^h \in C$, from eqn (2) the net, in the training phase, associates to every bit x_j^h equal to 1 in $\mathbf{x}^h$ a coupling coefficient in the hth row of A, $a_{hj} = 1/w_H(\mathbf{x}^h)$. Therefore, in the hth row (A_h), we have $w_H(\mathbf{x}^h)$ elements $a_{hj} = 1/w_H(\mathbf{x}^h)$. Then $A_h\mathbf{x}^h = 1 > \theta_h = 1 - \varepsilon$, and $\mu_h = 1$. ■

THEOREM 2. $\forall \mathbf{x}^h \in C$, *only the* hth *component of* μ *is different from* 0, *that is,* $\mu_h = 1$ *and* $\mu_r = 0$, $\forall r \neq h$.

Proof. We prove that $\forall r \neq h$ we have $A_r\mathbf{x}^h - \theta_r \leq 0$. We consider three cases.

1. Let us suppose that, in the first case, $w_H(\mathbf{x}^r) > w_H(\mathbf{x}^h)$. In the worst case the $w_H(\mathbf{x}^h)$ bits equal to 1 in $\mathbf{x}^h$ are in the same positions of the bits equal to 1 in $\mathbf{x}^r$. Then we have

$$A_r\mathbf{x}^h - \theta_r = \frac{w_H(\mathbf{x}^h)}{w_H(\mathbf{x}^r)} - 1 + \varepsilon < 0$$

because

$$\frac{w_H(\mathbf{x}^h)}{w_H(\mathbf{x}^r)} < 1.$$

2. We suppose that $w_H(\mathbf{x}^r) < w_H(\mathbf{x}^h)$. In the worst case there are $w_H(\mathbf{x}^r)$ bits equal to 1 in $\mathbf{x}^h$ in the same positions of the bits equal to 1 in $\mathbf{x}^r$. Then we have

$$A_r\mathbf{x}^h - \theta_r = \frac{w_H(\mathbf{x}^r)}{w_H(\mathbf{x}^r)} - \frac{(w_H(\mathbf{x}^h) - w_H(\mathbf{x}^r))}{w_{\max}(C)} - 1 + \varepsilon < 0$$

because

$$w_H(\mathbf{x}^h) - w_H(\mathbf{x}^r) > 0.$$

3. We suppose that $w_H(\mathbf{x}^h) = w_H(\mathbf{x}^r)$. To have $A_r\mathbf{x}^h - \theta_r > 0$ the position of the bits equal to 1 in $\mathbf{x}^h$ should be the same as the bits equal to 1 in $\mathbf{x}^r$. But this is not possible because in the definition of e-correcting binary linear code we have $\forall \mathbf{x}^r, \mathbf{x}^h \in C$, $r \neq h$, $d_H(\mathbf{x}^r\,\mathbf{x}^h) \geq 2e + 1$. ■

By these properties, only one code word corresponds to each row of the A matrix, that is, the association realized by eqn (2) is unique.

3.2. Threshold Tuning Phase

In the threshold tuning phase, the threshold, θ_h, of each neuron h is decreased by a quantity $e\Delta$, $\Delta = 1/(2e + 1)$. In this way, if in the transmission of a code word a number $\leq e$ of errors was made, we input to the net the

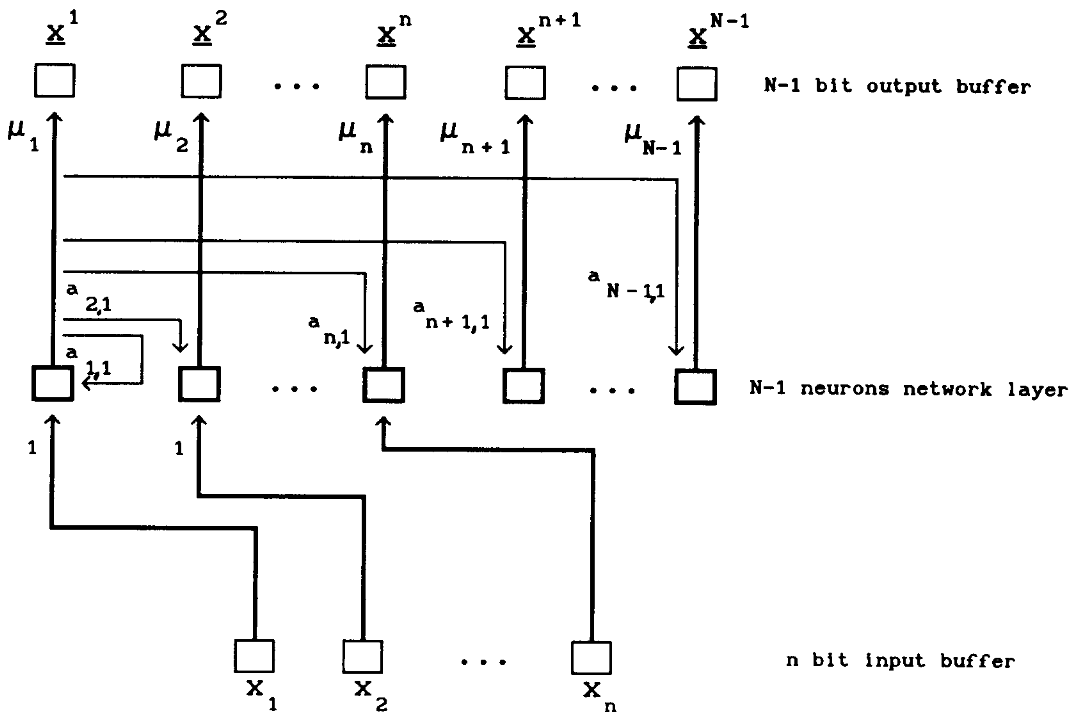

FIGURE 3. Neural network schema.

noisy pattern y and the neuron associated to the sent code word fires. In fact:

THEOREM 3. *Let* $\mathbf{x}^h$ *and* $\mathbf{y}$ *be the transmitted and the received vectors, respectively, with* $\mathbf{y} \neq \mathbf{x}^h$. *Decreasing the threshold of each neuron of* $e\Delta$, $\Delta = 1/(2e+1)$, *we have the firing of only one neuron, exactly the h one*

Proof First we determine the value Δ should assume so that neuron h fires. To achieve this we consider the value $A_h\mathbf{y} - \theta_h$ in the three worst cases:

1. If in $\mathbf{x}^h$ we have e bits changed by $1 \rightarrow 0$, then

$$A_h\mathbf{y} - \theta_h = \frac{w_H(\mathbf{x}^h) - e}{w_H(\mathbf{x}^h)} - 1 + e\Delta + \varepsilon$$

$$= -\frac{e}{w_H(\mathbf{x}^h)} + e\Delta + \varepsilon > 0 \Leftrightarrow \Delta \geq \frac{1}{w_H(\mathbf{x}^h)}.$$

2. If in $\mathbf{x}^h$ e bits have been altered by $0 \rightarrow 1$, then

$$A_h\mathbf{y} - \theta_h = \frac{w_H(\mathbf{x}^h)}{w_H(\mathbf{x}^h)} - \frac{e}{w_{\max}(C)} - 1 + e\Delta + \varepsilon$$

$$= 1 - \frac{e}{w_{\max}(C)} - 1 + e\Delta + \varepsilon > 0 \Leftrightarrow \Delta \geq \frac{1}{w_{\max}(C)}.$$

3. If in $\mathbf{x}^h$ k bits have been altered by $1 \rightarrow 0$ and $(e - k)$ bits by $0 \rightarrow 1$, then

$$A_h\mathbf{y} - \theta_h = \frac{w_H(\mathbf{x}^h) - k}{w_H(\mathbf{x}^h)} - \frac{e-k}{w_{\max}(C)} - 1 + e\Delta + \varepsilon$$

$$= \frac{w_H(\mathbf{x}^h)}{w_H(\mathbf{x}^h)} - \frac{k}{w_H(\mathbf{x}^h)} - \frac{e-k}{w_{\max}(C)} - 1 + e\Delta + \varepsilon$$

$$\geq -\frac{k}{w_H(\mathbf{x}^h)} - \frac{e-k}{w_H(\mathbf{x}^h)} + e\Delta + \varepsilon$$

$$> 0 \Leftrightarrow \Delta \geq \frac{1}{w_H(\mathbf{x}^h)}.$$

Therefore the neuron h fires $\Leftrightarrow \Delta \geq 1/w_H(\mathbf{x}^h)$. Because $1/(2e+1) \geq 1/w_H(\mathbf{x}^h)$, $\forall h$ we must choose

$$\Delta \geq \frac{1}{2e+1}. \tag{3}$$

Now we have to find the value Δ should assume so that the neuron h is the only one that fires. $\forall r \neq h$, in the worst case, $d_H(\mathbf{x}^r, \mathbf{y}) \geq e + 1$. Then we have

$$A_r\mathbf{y} \leq 1 - \frac{e+1}{w_{\max}(C)} + \varepsilon$$

and

$$A_r\mathbf{y} - \theta_r \leq 1 - \frac{e+1}{w_{\max}(C)} - 1 + e\Delta + \varepsilon$$

$$= -\frac{e+1}{w_{\max}(C)} + e\Delta + \varepsilon$$

$$\leq 0 \Leftrightarrow \Delta < \frac{e+1}{w_{\max}(C)e}. \tag{4}$$

To satisfy eqns (3) and (4) it must be

$$\frac{1}{2e+1} < \frac{e+1}{w_{\max}(C)e} \Rightarrow w_{\max}(C) < 2e + 3 + 1/e. \quad \blacksquare$$

3.3. Decoding Phase

After training and tuning, the net can be used as an error-correcting decoder of patterns received from binary symmetric channels (Gallager, 1968; MacWilliams & Sloane, 1981; Peterson and Weldon, 1972).

A noisy pattern $\mathbf{y}$ from the channel was input to the net. After the propagation delay, if a number $\le e$ of errors was made in the transmission, only the neuron associated to the right code word is activated by eqn (1).[3]

4. EXTENSION TO COMPLEMENTARY CODES

Because $w_{\max}(C)$ must be less than $2e + 3 + 1/e$, it is hard to apply the net to the complementary codes (Gallager, 1968). But, if we decrease the threshold of the neuron associated to vector $\mathbf{1}$ (where $\mathbf{1}$ is a code word) of a quantity $e\Delta^1$, where $\Delta^1 \neq \Delta$, this becomes possible. We train the net as before, fixing the parameter $\beta = 1/[w_{\max}(C - \{\mathbf{1}\})]$.

In the threshold tuning phase we tune the threshold of the neuron associated to code word $\mathbf{1}$ with a quantity $e\Delta^1$, where $\Delta^1 = 1/n$. We have:

THEOREM 4. *If we decrease the threshold of the neuron associated to vector* $\mathbf{1}$ *(code word) of a quantity* $e\Delta^1$, *where* $\Delta^1 = 1/n$, *the decoding algorithm can also be applied to complementary codes with* $w_{\max}(C - \{\mathbf{1}\}) \le 2e + 3$.

Proof. We suppose $\beta = 1/[w_{\max}\{C - \{\mathbf{1}\}\}]$. We call A_{h^1} the row of the A matrix that is associated to vector $\mathbf{1}$ and $\mathbf{y}^1$ a noisy version of the code word $\mathbf{1}$ with, at most, e bits altered. We have

$$A_{h^1}\mathbf{y}^1 - \theta_{h^1} \ge \frac{n-e}{n} - 1 + e\Delta^1 + \varepsilon > 0 \Leftrightarrow \Delta^1 \ge 1/n.$$

This assures us that the neuron h^1 fires for any noisy version of the code word $\mathbf{1}$ iff $\Delta^1 \ge 1/n$.

For every vector $\mathbf{y}$, noisy or not version of any other code word, we have that the neuron h^1 will not fire because

$$\begin{aligned} A_{h^1}\mathbf{y} - \theta_h^1 &\le \frac{n-(e+1)}{n} - 1 + e\Delta^1 + \varepsilon \\ &= 1 - \frac{e+1}{n} - 1 + e\Delta^1 + \varepsilon \\ &\le 0 \Leftrightarrow \Delta^1 \le 1/n + 1/en + \varepsilon. \end{aligned}$$

Moreover, $\forall r \neq h^1$ we have $A_r\mathbf{y}^1 - \theta_r \le 0$. In fact:

$$A_r\mathbf{y}^1 - \theta_r \le 1 - \frac{e+1}{w_{\max}(C - \{\mathbf{1}\})} - 1 + \frac{e}{2e+1} + \varepsilon \le 0.$$

Therefore, $\forall r \neq h^1$ the neuron r does not fire. $\blacksquare$

4.1. Neuron Number Reduction

By the property of the complementary codes

$$\text{if} \quad \mathbf{x}^h \in C \quad \text{then} \quad (1, 1, \ldots, 1) - \mathbf{x}^h \in C$$

we can use a net of $N/2$ neurons, training it on only $N/2$ code words. Then, in the operative phase, we use the following procedure:

1. input the received pattern $\mathbf{y}$ to the net;
2. if the net answer is nonzero then exit;
3. complement $\mathbf{y}$; let $\mathbf{y}'$ be the new pattern;
4. input the pattern $\mathbf{y}'$ to the net.

5. PARALLEL IMPLEMENTATION

Given the characteristics of the model and to experimentally validate the theoretical results, we simulate the model on a distributed-memory multiprocessor system (Athas & Seitz, 1988). We use the DISC system, made up by a parallel programming language and a programming environment, based on the CSP model of computation (Hoare, 1978). It was conceived for distributed memory MIMD (Multiple Instruction, Multiple Data) parallel architectures. Currently, it runs on two different platforms: a local area network of UNIX workstations, and a network of Transputer processors.

The DISC concurrent language (Iannello et al., 1989) extends the C sequential language with three basic concurrent constructs: a *parallel* command (*par*), to activate several user processes in parallel, independently of their location in the underlying multicomputer system; commands for message exchange (input/output) between processes over many-to-one (*channel*) abstractions; an *alternative* command (*alt*) for nondeterministic selection of one among several potential partners in the communication over a channel.

DISC programs are made up by a set of sequential processes, structured in a hierarchical way by means of the parallel command (there is always a *root* process), and interacting by means of explicit message passing statements.

The DISC system presents some features that make it suitable for fast prototyping of neural networks parallel simulations.

[3] This is not true if we only received the code word $\mathbf{0}$, or a noisy version of it. In this case, because its weight is $\le e$, no neuron is activated by eqn (1) In this way we associate the vanishing value of all the neuron activations to the reception of the code word $\mathbf{0}$ or a noisy version of it.

At the language level, the user has no view of the number of processing elements in the target machine, and of the topology of interconnection among them. Of course, the number of processors, the topology, and the algorithm used to map processes onto processors have a strong influence on the performances of the program. However, with respect to other concurrent languages like OCCAM, DISC frees the programmer, while designing and testing a parallel implementation for a neural network, from being concerned with many concurrent software engineering problems, like writing code for routing of messages in the multicomputer system, writing nonscalable programs, and so on. Moreover, the capability to develop the DISC neural application in a friendly programming environment on a UNIX workstation, before running it on a Transputer-based machine, can considerably shorten the time required to implement and test a neural model.

These properties appear to be particularly interesting in a research environment, where people interested in speeding up their simulations must face the time and effort needed to implement a simulation program on a Transputer system.

5.1. Simulation Program

The structure of a DISC program to simulate a given neural network consists basically of definitions of processes, which correspond to neuron types, and of a root process, which creates actual neurons and sets up connections among them. Actual neurons are activated as instances of the user-defined neuron types.

Our parallel implementation of the neural error-correcting decoder consists of as many concurrent instances of a neuron process as there are neurons in the model, and a multiplexer process, that carries out the task of interfacing the user and distributing the code words to the neuron processes. Neurons and multiplexer are interconnected through many-to-one communication channels. The program code is reported in the Appendix.

In the training phase, the neuron issues an input command over the channel *DataChan* to receive the code word from the multiplexer. In the same way, during recognition, each neuron instance waits on the channel for a pattern to decode; according to the properties of the net, only one neuron will recognize the code word, issuing the print statement.

The multiplexer process gets code words from the user, or from a database, and distributes them to the neurons. Then it loops getting patterns to be decoded and presenting them to the whole net. The program exploits the *automatic termination* mechanism of DISC. When there are no more input patterns to be decoded, the multiplexer exits from the loop and terminates; because the neuron processes will never be able to communicate with the multiplexer, the system kernel is responsible for their automatic termination.

The root process simply activates the neurons and the multiplexer by means of a parallel command. The program stops when all child processes terminate, and the parallel command will terminate too, returning control to the root process.

6. EXPERIMENTS

We run our simulation program on a set of four Sun workstations SPARCstation IPX running SunOs. We use a particular linear code, the Reed–Muller code RM(1, 3) (Peterson & Weldon, 1972) (Figure 4) with 16 code words, each of them 8 bits long. Given the properties of this code, each neuron is able to recognize two complementary code words. Thus, the number of neurons required can be set to one-half the number of code words (Figures 5, 6).

We distribute neuron processes evenly among processors. The training time for the network made up of 8 neurons is about 200 ms. This is also the time required to recognize a code word, because the amount of data to be transmitted in the network is the same in both the learning and decoding phases, and the local computation time for each neuron can be neglected.

7. CONCLUDING REMARKS

This work is devoted to an application of neural networks for error-correcting decoding of binary linear codes.

The idea of connecting neural nets and error-correcting codes was theoretically developed by Bruck and Blaum (1989), where it was established that it is pos-

$\underline{x}^0$	0	0	0	0	0	0	0	0
$\underline{x}^1$	0	0	0	0	1	1	1	1
$\underline{x}^2$	0	0	1	1	0	0	1	1
$\underline{x}^3$	0	1	0	1	0	1	0	1
$\underline{x}^4$	0	0	1	1	1	1	0	0
$\underline{x}^5$	0	1	0	1	1	0	1	0
$\underline{x}^6$	0	1	1	0	0	1	1	0
$\underline{x}^7$	0	1	1	0	1	0	0	1
$\underline{x}^8$	1	1	1	1	1	1	1	1
$\underline{x}^9$	1	1	1	1	0	0	0	0
$\underline{x}^{10}$	1	1	0	0	1	1	0	0
$\underline{x}^{11}$	1	0	1	0	1	0	1	0
$\underline{x}^{12}$	1	1	0	0	0	0	1	1
$\underline{x}^{13}$	1	0	1	0	0	1	0	1
$\underline{x}^{14}$	1	0	0	1	1	0	0	1
$\underline{x}^{15}$	1	0	0	1	0	1	1	0

FIGURE 4. The Reed–Muller code RM(1, 3).

sible to construct a Hopfield network (Hopfield, 1982) in which every local maximum is a code word and vice versa and that the maximum likelihood decoding of error-correcting codes is equivalent to find the maximum of the energy function of the net. Because the maximum likelihood decoding problem is NP-hard, we cannot expect to have the Hopfield network converge to the optimal solution unless the neuron number increases exponentially. In this case the error probability is high, because the energy function of the Hopfield net can have more than one maximum.

Our neural net, after learning the code, reduces the error decoding probability to zero for a noise version of code words with a number of errors in the range of the error-correcting capacity of the code, with the further advantages of a lower number of neurons and of a fast and simple traming.

We proved that the class of binary linear codes that can be given in input to the net is limited, in the general case, by the relation $w_{\max}(C) \leq 2e + 3$, and, for the complementary codes, by $w_{\max}(C - (1, 1, \ldots, 1)) \leq 2e + 3$, where $w_{\max}(C)$ is the maximum Hamming weight of a code word in C. To extend the net to larger classes is a direction of future work.

Regarding the parallel simulation, we should point out that the small grain size of each neuron prevents obtaining good speed-up with simulation programs using an increasing number of processors, at least as far as networks with up to a few tens of neurons are simulated. This is due to the fact that the advantage of distributing the load among several processors can be outperformed by the greater cost of nonlocal communications in a network where the activity of the neurons is negligible. Actually, the neural error-correcting decoder is better suited for a specifically designed hardware implementation.

$$\begin{array}{ccccccccc}
\mathbf{A_1} & -\beta & -\beta & -\beta & -\beta & \alpha & \alpha & \alpha & \alpha \\
\mathbf{A_2} & -\beta & -\beta & \alpha & \alpha & -\beta & -\beta & \alpha & \alpha \\
\mathbf{A_3} & -\beta & \alpha & -\beta & \alpha & -\beta & \alpha & -\beta & \alpha \\
\mathbf{A_4} & -\beta & -\beta & \alpha & \alpha & \alpha & \alpha & -\beta & -\beta \\
\mathbf{A_5} & -\beta & \alpha & -\beta & \alpha & \alpha & -\beta & \alpha & -\beta \\
\mathbf{A_6} & -\beta & \alpha & \alpha & -\beta & -\beta & \alpha & \alpha & -\beta \\
\mathbf{A_7} & -\beta & \alpha & \alpha & -\beta & \alpha & -\beta & -\beta & \alpha \\
\mathbf{A_8} & \gamma & \gamma & \gamma & \gamma & \gamma & \gamma & \gamma & \gamma \\
\mathbf{A_9} & \alpha & \alpha & \alpha & \alpha & -\beta & -\beta & -\beta & -\beta \\
\mathbf{A_{10}} & \alpha & \alpha & -\beta & -\beta & \alpha & \alpha & -\beta & -\beta \\
\mathbf{A_{11}} & \alpha & -\beta & \alpha & -\beta & \alpha & -\beta & \alpha & -\beta \\
\mathbf{A_{12}} & \alpha & \alpha & -\beta & -\beta & -\beta & -\beta & \alpha & \alpha \\
\mathbf{A_{13}} & \alpha & -\beta & \alpha & -\beta & -\beta & \alpha & -\beta & \alpha \\
\mathbf{A_{14}} & \alpha & -\beta & -\beta & \alpha & \alpha & -\beta & -\beta & \alpha \\
\mathbf{A_{15}} & \alpha & -\beta & -\beta & \alpha & -\beta & \alpha & \alpha & -\beta
\end{array}$$

FIGURE 5. The coupling coefficient matrix for the Reed–Muller code RM(1, 3). $\alpha = \frac{1}{4}, \beta = \frac{1}{4}, \gamma = \frac{1}{8}$.

$$\begin{array}{ccccccccc}
\mathbf{A_1} & -\beta & -\beta & -\beta & -\beta & \alpha & \alpha & \alpha & \alpha \\
\mathbf{A_2} & -\beta & -\beta & \alpha & \alpha & -\beta & -\beta & \alpha & \alpha \\
\mathbf{A_3} & -\beta & \alpha & -\beta & \alpha & -\beta & \alpha & -\beta & \alpha \\
\mathbf{A_4} & -\beta & -\beta & \alpha & \alpha & \alpha & \alpha & -\beta & -\beta \\
\mathbf{A_5} & -\beta & \alpha & -\beta & \alpha & \alpha & -\beta & \alpha & -\beta \\
\mathbf{A_6} & -\beta & \alpha & \alpha & -\beta & -\beta & \alpha & \alpha & -\beta \\
\mathbf{A_7} & -\beta & \alpha & \alpha & -\beta & \alpha & -\beta & -\beta & \alpha \\
\mathbf{A_8} & \gamma & \gamma & \gamma & \gamma & \gamma & \gamma & \gamma & \gamma
\end{array}$$

FIGURE 6. The coupling coefficient matrix for the Reed–Muller code RM(1, 3) using the neuron number reduction. $\alpha = \frac{1}{4}, \beta = \frac{1}{4}, \gamma = \frac{1}{8}$.

REFERENCES

Athas, W. C , & Seitz, C. L. (1988). Multicomputers: Message-passing concurrent computers. *IEEE Computer,* **Jan.,** 9–24.

Bruck, J., & Blaum, M. (1989) Neural networks, error correcting codes and polynomials over the binary *n* cube *IEEE Transactions on Information Theory,* **35,** 976–987

Caianiello, E R. (1961). Outline of a theory of thought processes and thinking machines. *Journal of Theoretical Biology,* **2,** 204–235

Esposito, A., Mastroianni, M., Rampone, S , & Russo, S. (1993). Parallel simulation of a neural error correcting decoder In E. R. Caianiello (Ed.), *Neural nets WIRN Vietri*-92 Singapore: World Scientific.

Esposito, A., Rampone, S., & Tagliaferri, R. (1991). Addition and subtraction in neural nets as results of a learning process. In T Kohonen, K Makisara, O. Simula, & J. Kangas (Eds.), *Artificial neural networks* Amsterdam· North-Holland.

Gallager, R. G (1968). *Information theory and reliable communication* New York· J. Wiley and Sons

Garey, M. R., & Johnson, D. S (1979) *Computers and intractability A guide to the theory of NP completeness* San Francisco: W. H. Freeman

Hoare, C. A R (1978) Communicating sequential processes. *Communication of the ACM,* **21,** 666–677

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA,* **79,** 2554–2558

Iannello, G , Mazzeo, A , & Ventre, G (1989) Definition of the DISC concurrent language *SIGPLAN Notices,* **24,** 59–68.

Iannello, G , Mazzeo, A , Savy, C., & Ventre, G. (1990). Parallel software development in the DISC programming environment. *Future Generation Computer Systems,* **5**(N.4), 365–372.

MacWilliams, F. J., & Sloane, N. J. A. (1981). *The theory of error correcting codes* New York. North Holland.

McElice, R. J (1977). The theory of information and coding. In G -C. Rota (Ed.), *Encyclopedia of mathematics and its applications* (vol 3). London: Addison-Wesley

Peterson, W W, & Weldon, W. J. (1972) *Error correcting codes* Cambridge, MA MIT Press.

APPENDIX: SIMULATION PROGRAM

The following is the DISC code of the simulation program. In the code, N stands for the length of the codewords and NPC for the number of codewords, that is, the number of neurons.

A.1. Definition File (defs.h).

```
#define N 8 /* codeword length */
#define NPC 8 /* neuron number */
#define WMAX 4.0

/* error correcting parameters */
#define E 1.0
#define DELTA (1.0/(2.0* E + 1.0))
#define DELTA1 (DELTA - (1.0/(float) N))
#define S (1.0 - E * DELTA/(2.0* E + 1.0))
```

A.2. Neuron Process (neuron.d).

```
#include "defs.h"
process neuron(i, DataChan, Ack)
in int i,
used int DataChan, Ack,

{
int j, CodeSum,
int CodeWord[N], Pattern[N];
float wh, val, threshold, weight[N],

/* training phase */
DataChan ?? CodeWord; /* get codeword from mux */

/* compute weights */
for(j=0, wh=0.0; j < N, j++) wh = wh + (float) (CodeWord[j]);
for (j = 0, j < N, j++)
   if(CodeWord[j]) weight[j] = 1.0/wh,
   else weight[j] = - 1.0/WMAX;

/* threshold tuning */
for (j = 0, CodeSum=0; j < N, j++) CodeSum = CodeSum
       + CodeWord[j];
if(CodeSum < N) threshold = (float) S;
   else threshold = (float) S1,

/* end learning */
Ack !! sync,

/* input and decoding phase */
while (1) {
   DataChan ?? Pattern, /* input pattern from mux */
   for (j = 0, val = 0.0; j<N; j++) val = val + ((float)(Pattern[j])
         * weight[j]),
   if ((val - threshold) >0) {
      for (j=0, j<N, j++) printf("%d", Codeword[j]), }
/* codeword decoded! */

/* try for complementary codeword */
   for (j = 0, j<N, j++) (pattern[j] + 1) %2
   for (j = 0, val = 0.0; j<N, j++) val = val + ((float)(Pattern[j])
         * weight[j]),
   if ((val - threshold) >0) {
      for (j=0; j<N; j++) printf("%d", (Codeword[j] + 1) %2), }
/* codeword decoded! */
Ack !! sync;
} /* endwhile */
}
endprocess
```

A.3. Multiplexer Process (multiplexer.d).

```
process multiplexer(DataChan, Ack)
owned into DataChan, Ack,

{
char str[N],
int i, j,
int dati[NPC][N],
FILE *fp;

fp = fopen("rm13.code","r")
if(fp == NULL) printf("Error opening file"),

/* Get codewords */
for (i=0, i < NPC, i++) {
   if(fscanf(fp, "%s", str) != EOF {
                  for (j=0; j < N, j++) dati[i][j] = str[j] - '0',
   }
}

/* training phase */
for (i=0; i < NPC, i++) DataChan !! dati[i],

/* wait for end learning */
for (i=0, i < NPC, i++) Ack ?? sync,

/* decoding phase */
while(scanf("%s",str) != EOF) {
   for (j = 0, j < N, j++) dati[0][j] = str[j] - '0',
   fflush(stdin),
   for (i = 0, i < NPC, i++) DataChan !! dati[0],
   for (i = 0, i < NPC, i++) Ack ?? sync,
}
endprocess
```

A.4. Root Process (root.d).

```
process root()

{
int i,
par {
   local ch1, ch2,
   process multiplexer(chan ch1,ch2),
   for(i = 0, i < NPC; i++) process neuron(in i, chan ch1,ch2),
   }
}
endprocess
```