Um Estudo da Viabilidade de Redes Neurais na Solução do Problema de Decodificação Binária

Eduardo Navarra Satuf

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Luís Alfredo Vidal de Carvalho, D.Sc.
(Presidente)

Prof. Lideniro Alegre, Ph.D.

Prof. Luiz Pereira Calôba, Dr.Ing.

RIO DE JANEIRO, RJ - BRASIL

ABRIL DE 1993

SATUF, EDUARDO NAVARRA

Um Estudo da Viabilidade de Redes Neurais na Solução do Problema de Decodificação Binária [Rio de Janeiro] 1993.

VII, 101 p., 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1993)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Redes Neurais 2. Códigos de Controle de Erros

I. COPPE/UFRJ II. Título (série).

Agradecimentos

Meus agradecimentos aos orientadores: ao Remo, que propôs o tema, pela orientação e pela cessão do software; e ao prof. Luís Alfredo, pela orientação, pela atenção e por sua ajuda nos meandros da COPPE.

Fico grato também ao prof. Lideniro Alegre e ao prof. Luiz Calôba, participantes da banca, pelas sugestões e comentários, que me ajudaram no fechamento do texto, ainda que ao final do trabalho.

Meus agradecimentos também aos professores e funcionários da COPPE, especialmente às secretárias da COPPE-SISTEMAS, Ana Paula e Cláudia, pela ajuda com a papelada.

Agradeço à PETROBRÁS, nas pessoas de José Luís Stoller e Roberto Murilo de Souza, pela oportunidade de cursar o mestrado. Também ao Setor de Informação Técnica do CENPES, pela pesquisa bibliográfica.

Muitas outras pessoas me ajudaram na realização desta tese. Mesmo com o inevitável esquecimento de algumas, cito: José Pereira (JP), Marco Aurélio, Flávio Santos, Ênio, Fernando Barreto, Minoru, Glória, Regina, Izabel, Luiz Fernandes e demais colegas, que, num momento ou outro, ajudaram-me neste trabalho.

Last, but not least, agradeço à minha família, por toda a infra-estrutura (em diversos sentidos) que me proporciona.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para obtenção do grau de Mestre em Ciências (M.Sc.).

Um Estudo da Viabilidade de Redes Neurais na Solução do Problema de Decodificação Binária

Eduardo Navarra Satuf

ABRIL, 1993

Orientadores:

Prof. Engº. Luís Alfredo Vidal de Carvalho, D.Sc.

Prof. Engº. Remo Zauli Machado Filho

Redes Neurais Artificiais são estudadas desde a década de 40. Tratam-se de sistemas com vários processadores (neurônios) interligados que se adaptam ao meio, por um processo chamado de aprendizado, e/ou evoluem para estados estáveis a partir de qualquer ponto inicial. Nos últimos anos, tiveram renovado interesse devido às contribuições de Hopfield, Rumelhart, e outros.

O problema de Decodificação Binária se insere na Teoria de Controle de Erros, cujo objetivo é buscar formas de se transmitir mensagens binárias até certo ponto imunes a erros introduzidos pelo canal - codificação -, de forma que o receptor, realizando a decodificação, obtenha a mensagem original.

Nesta tese é feita uma revisão teórica de Códigos de Controle de Erros, e de Redes Neurais. Depois são aplicados diversos paradigmas de redes neurais ao problema de decodificação. Medidas de eficácia são apresentadas de forma a permitir uma comparação.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

A Study of the Viability of Neural Networks in the Solution of the Binary Decoding Problem

Eduardo Navarra Satuf

APRIL, 1993

Thesis supervisors:

Prof. Eng. Luís Alfredo Vidal de Carvalho, D.Sc.

Prof. Eng. Remo Zauli Machado Filho

Artifical Neural Networks are studied since the '40s. They are systems with several interconnected processors (neurons) that can adapt themselves to the environment, by a learning process, and/or can evolve to stable states from any initial point. Recently, interest in Neural Networks is renewed for Hopfield, Rumelhart, and other's contributions.

The Binary Decoding Problem is part of the Error Control Theory, which studies the ways of transmitting binary messages with at least partial imunity to errors induced by the channel - coding -, so that the receptor, through decoding, can get the original message.

A theoretical revision of Error Control Coding, and of Neural Networks is done in this work. Then, some neural networks paradigms are applied to the problem of decoding. Decoding performance results are presented so that a comparison is possible.

ÍNDICE

Capítulo I
Introdução

Capítulo II
Códigos de Controle de Erros

Capítulo III
Redes Neurais Artificiais

Capítulo IV

Códigos e Redes Neurais

Capítulo V

Conclusão

Capítulo I

Introdução

Apresentaremos as origens deste trabalho, o conteúdo do texto e o problema ao qual se aplica o material estudado.

I.1 - *Motivação*

Apesar de terem seus conceitos matematicamente expostos desde a década de '40, Redes Neurais Artificiais vêm-se popularizando particularmente nos últimos 6 a 10 anos, depois de mais de 12 anos fora de evidência. Este ressurgimento segue-se particularmente à contribuição de Hopfield [25] no estudo de equilíbrio de redes, usando conceitos de Física, e ao livro de McClelland e Rumelhart [47] e seu algoritmo backpropagation.

A Teoria de Códigos para controle de erros tem uma história mais regular, desenvolvendo-se desde 1948 com Claude Shannon. Desde então, várias classes de código foram descritas, e métodos de decodificação desenvolvidos, usando computação tradicional.

Esta dissertação tem início no projeto de comunicação hidro-acústica do Setor de Engenharia Submarina do Centro de Pesquisas da PETROBRÁS, onde foi proposto o tema, e com a disciplina de Redes Neurais na COPPE-SISTEMAS. O objetivo é estudar redes neurais aplicadas a um problema real, de decodificação, comparando e confrontando diversos modelos.

I.2 - *Organização do Texto*

Este texto trata da aplicação de diversos paradigmas de Redes Neurais Artificais ao problema da decodificação. Neste capítulo I, é ainda apresentado o problema de decodificação e o contexto da aplicação.

No capítulo II faz-se uma revisão teórica de conceitos de Códigos de Controle de Erros, de forma a se definir códigos BCH e de Golay. Em particular, códigos BCH

possuem uma estrutura algébrica muito bem definida, e um algoritmo de decodificação bastante estudado. São básicos para o entendimento da teoria.

No capítulo III, nos detemos sobre os conceitos básicos de Redes Neurais, e descrevemos os fundamentos dos modelos de rede estudados ao longo do desenvolvimento deste trabalho: topologia, algoritmo de aprendizado e de recuperação, estabilidade, e vantagens e limitações de cada um.

A aplicação propriamente dita está no capítulo IV. Após uma revisão bibliográfica, cada modelo é aplicado na decodificação de um código simples (código de Hamming), e os de melhor resultado são usados em outros códigos de maior capacidade. Resultados quanto à eficácia da combinação rede-código são mostrados.

Finalmente, o capítulo V traz as considerações finais, ressaltando alguns temas em aberto. E o Apêndice traz, a título de ilustração, uma implementação do algoritmo de Berlekamp, feita apenas para fins de estudo.

I.3 - *Apresentação do Problema*

Remo Machado, Manoel F. Tenório, e J. R. M. Silva [42] descrevem um sistema de comunicação hidro-acústica que faz uso de redes neurais. A proposta básica é utilizar uma rede de Hamming, que sofreria o aprendizado em laboratório e, no campo, faria a decodificação das palavras recebidas (transmitidas segundo o código de Hamming (7,4)).

Existe a necessidade de comunicação entre unidades de superfície (navios, plataformas) e sistemas submarinos (por exemplo, a árvore de natal molhada - ANM: conjunto de válvulas de controle, na cabeça do poço submarino), veja Figura I.1. Cabos têm sido usados, mas fatores como a carga a que são submetidos pelas correntes oceânicas, o próprio peso, que causa dificuldades de estabilização, e dificuldades de conexão, além de um custo muito alto, levam à pesquisa por outros meios de comunicação. Uma alternativa é o uso de ondas acústicas.

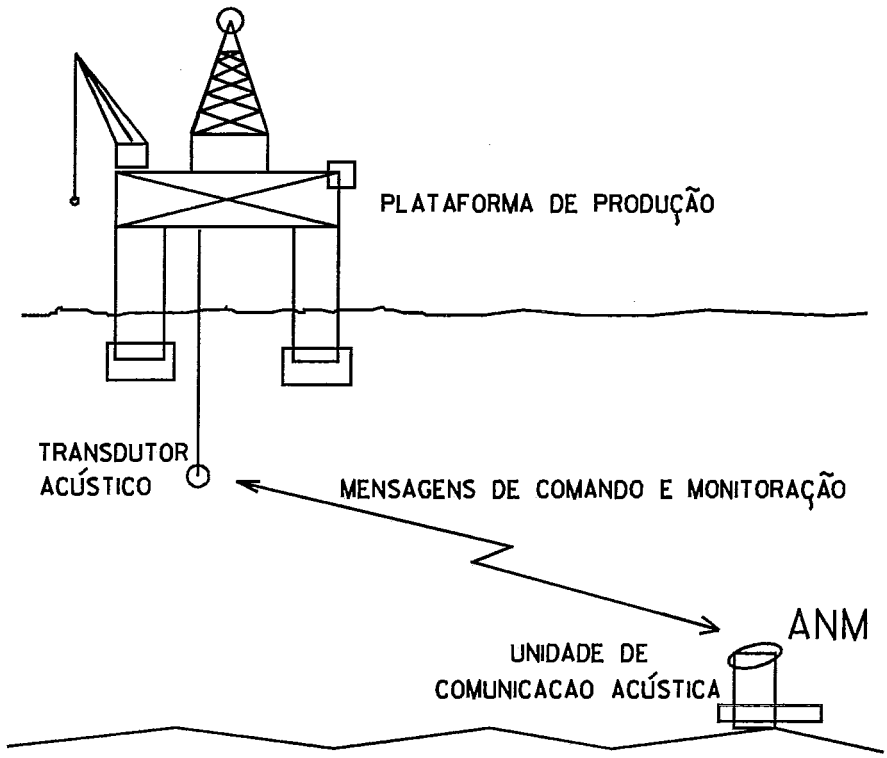


Figura I.1. Comunicação acústica em explotação submarina de petróleo.

Uma das características do canal acústico é a baixa velocidade de propagação (1500 m/s), que implica na demora em se receber uma mensagem qualquer. Jourdain, [33], traz resultados experimentais sobre propagação de ondas acústicas submarinas. Outras características do canal, como a variação desta velocidade com a salinidade, temperatura, pressão ou profundidade, reverberação, *fading* e ecos fazem com que erros possam ser introduzidos nas mensagens transmitidas. Retransmissões tornam-se altamente indesejáveis, pois sofreriam dos mesmos problemas, além do tempo necessário para o processamento completo de uma mensagem triplicar (envio e detecção do erro, solicitação de retransmissão, e retransmissão). A própria unidade de superfície (navio ou plataforma) introduz ruído no canal ([6]). É necessário, então, uma forma de transmissão que resista a estas características do canal.

Esta forma realiza-se na **codificação** das mensagens (comandos para a ANM, por exemplo) e, claro, a **decodificação** no receptor (a ANM), de forma que até um determinado número de erros possa ser corrigido pelo próprio receptor. Um diagram de blocos ([42]) é apresentado na Figura I.2.

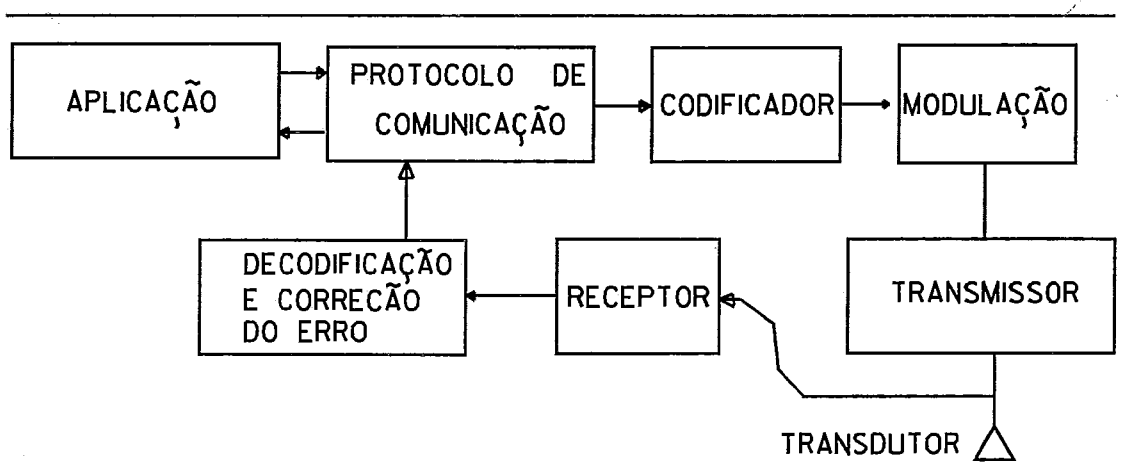


Figura I.2. Diagrama de bloco do sistema hidro-acústico de comunicação.

O decodificador, neste trabalho, é uma rede neural, que recebe os dados após um pré-processamento, e devolve a mensagem original, dado que o número de erros ficou dentro da capacidade do código.

A referência [42] cita a codificação convolucional como vantajosa para comunicação hidro-acústica, mas analisa o código blocado (7,4) (de Hamming), de mais simples estudo e implementação, para o qual descreve a aplicação. É utilizada a rede de Hamming para a decodificação.

Nesta dissertação, estudaremos o mesmo código de Hamming com outras redes, além de outros códigos para determinados modelos de redes neurais.

Capítulo II

Códigos de Controle de Erros

Códigos de controle de erros são usados para proteger dados de erros que podem ocorrer durante uma transmissão através de um canal de comunicação ([8]). Descreveremos em linhas gerais um sistema de comunicação e estudaremos um método de codificação binária.

II.1 - *Sistemas de Comunicação*

Um sistema de comunicação conecta uma fonte de dados a um usuário de dados (que pode ser uma pessoa ou um programa sendo executado em um equipamento), através de um canal. Um diagrama tradicional é o da Fig. II-1, de Blahut, [8].

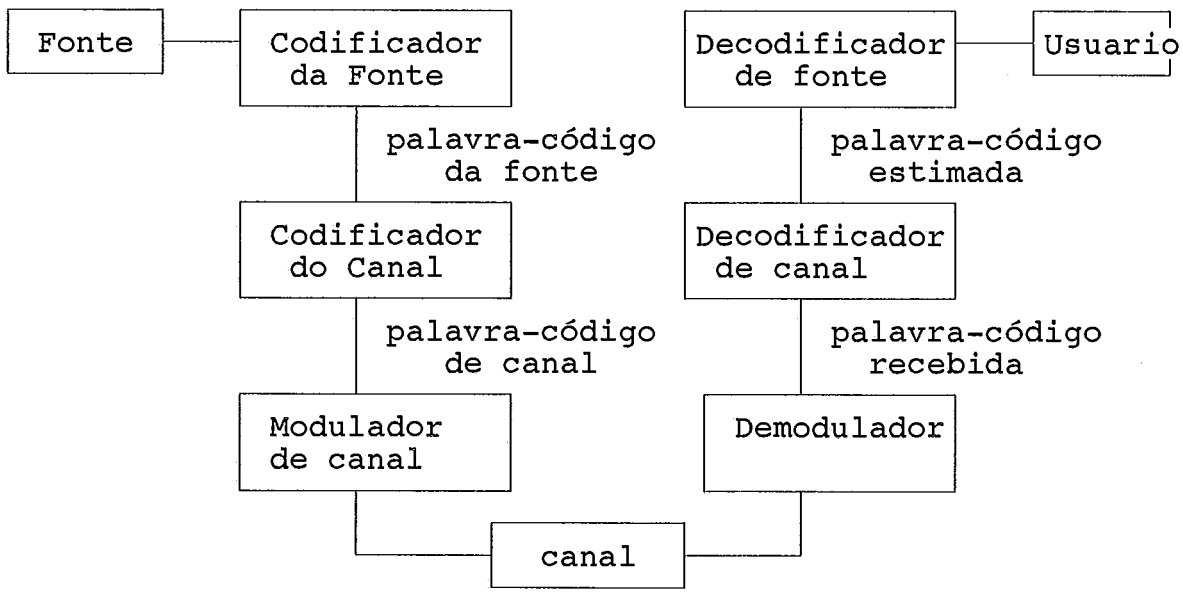


Figura II-1. Sistema de comunicação.

Dados entram no sistema de comunicação, vindos da fonte, através do "codificador de fonte", que os representa de maneira compacta, numa palavra-código fonte. Esta é codificada pelo codificador de canal, de forma a conter redundâncias, resultando em uma palavra-código de canal.

Esta palavra-código é modulada para o canal em questão (por exemplo, canal acústico-marinho) e transmitida. Como o canal é sujeito a ruído, distorção e interferência, sua saída pode não ser igual à entrada. O demodulador converte cada sinal de saída num símbolo de palavra-código de canal, melhor estimada (símbolo estimado) possível, mas também sujeita a erros. Esta estimada é chamada de palavra recebida, que, portanto, pode ser diferente da palavra-código original.

Então, o decodificador de canal usa a redundância que fora colocada na palavra-código de canal para corrigir os erros de transmissão, causados pelo ruído, gerando uma estimada da palavra-código fonte. Se todos os erros forem corrigidos, a palavra-código estimada será igual à palavra-código fonte original. O decodificador de fonte, que faz a operação inversa à do codificador de fonte, entrega a saída ao usuário.

Se nem todos os erros forem corrigidos (mas forem detectados), é necessário pedir retransmissão ou tomar alguma outra providência apropriada. Um problema de maior dimensão é quando ocorrem tantos e tais erros que causam uma decodificação aparentemente correta. Por isso, é necessário conhecer as probabilidades de um dígito estar em erro, e escolher um código que permita precisão suficiente (isto é, detecte e corrija uma quantidade de erros igual ou maior que aquela esperada).

Veremos adiante um método de codificação e decodificação que se traduzem no codificador de canal e no decodificador de canal (ou, simplesmente, no codificador e no decodificador).

II.2 - *Visão Histórica*

Em 1948, Claude Shannon mostrou que existe uma quantidade C de bits que podem ser transmitidos em um segundo por um dado canal. Este número é chamdo de **capacidade do canal** e Shannon mostrou que para qualquer taxa de transmissão R (em bits/s), $R < C$, é possível projetar um sistema de comunicação para aquele canal, usando códigos de controle de erros, cuja probabilidade de erro na saída é tão

pequena quanto se queira. E, portanto, é mais econômico usar um código que construir um canal extremamente bom (mesmo porque, muitas vezes, não temos influência sobre o canal).

Ao longo dos anos seguintes, pesquisou-se **como** obter um código suficientemente bom. A partir dos anos 60, duas linhas se mostram as principais.

A primeira, de forte apelo à Álgebra, baseia-se em códigos blocados (transmissão dos dígitos em blocos), apresentados em 1950 por Hamming [23], cujo exemplo foi um código com capacidade para corrigir um único erro, fraco comparado com o que o teorema de Shannon prometia. Até 1960, muitos códigos foram encontrados, mas sem uma teoria geral. Em 1959, Hocquenghem e, independentemente, em 1960, Bose e Ray-Chaudhuri encontraram uma grande classe de códigos que corrigem múltiplos erros - os códigos BCH.

Em 1960, Reed e Solomon encontraram uma classe de códigos não-binários relacionada aos códigos BCH.

A outra linha de pesquisa é a de decodificação sequencial, baseada em análise probabilística e numa classe de códigos não-blocados que podem ser representados por uma árvore, e são decodificados por uma busca nesta árvore. Um "código-árvore" bastante estruturado é o código convolucional, onde a codificação é feita por uma convolução dos bits de informação. Um algoritmo popular para decodificação é o de Viterbi [51]. (Uma análise do algoritmo de Viterbi é encontrada em [30].)

Embora tenha havido muitos progressos na área, e, dia a dia, surjam códigos mais eficientes, os códigos BCH continuam ocupando uma posição proeminente em virtude do equilíbrio possível entre eficiência, técnicas relativamente simples de codificação e decodificação e de serem básicos para o entendimento de outros tipos de códigos. Por estas razões, é a classe de códigos utilizada neste trabalho.

II.3 - *Conceitos Elementares*

Um **código blocado binário** de tamanho M e comprimento de bloco n é um conjunto de M palavras binárias de comprimento n, chamadas palavras-código. Em geral, $M = 2^k$

para um inteiro k, $k < n$, e o código é dito código binário (n,k).

Pode parecer que seria suficiente definir os requisitos de um código (os parâmetros n e k) e deixar que um computador procure as 2^k palavras "mais diferentes" entre si, cada uma com n símbolos. Há um total de $2^{n \cdot s}$, $s = 2^k$, maneiras de combinar estes símbolos, que é o número de diferentes códigos (n,k).

Exemplo: Para um código (7,4) - código de Hamming, capaz de corrigir um erro - haveria $5{,}19 \times 10^{33}$ ($= 128^{16} = 2^{7x16=112}$) possibilidades diferentes, isto é, conjuntos candidatos a código, de 16 palavras cada um, quantidade impossível de ser processada pelos computadores atuais num tempo hábil.

Assim, é necessária uma teoria para que se possa construir, de uma maneira prática, códigos de comunicação, onde se represente k símbolos de informação (bits) por n bits da palavra-código - um mapeamento de $\{0,1\}^k$ em $\{0,1\}^n$, de 1 para 1. A taxa R de eficiência do código (não confundir com a taxa de canal, medida em bits/s) é

$$R = \frac{k}{n}$$

onde R é um número adimensional, ou medido em símbolos/símbolo, ou ainda, em bits/bit.

O Teorema de Shannon diz que, para qualquer taxa de transmissão $R < C$, e tamanho n, existe um código tal que a probabilidade de decodificação errada é (veja Lin, [41]):

$$p(e) \leq e^{-n.E(R)}$$

onde $E(R)$ é uma função positiva de R, especificada pelas probabilidades de transição (alteração de símbolo) do canal. Assim, para um maior n, pode-se diminuir a probabilidade de erro na decodificação (mantendo $R < C$). A palavra-código $\mathbf{c}_t$ associada à palavra-recebida $\mathbf{r}$ é aquela para a qual $p(\mathbf{r}|\mathbf{c}_t)$ é máxima.

Para efeito de modelagem, o canal é tratado como sendo simétrico e binário (BSC, *Binary Symmetric Channel*,

veja Fig.II-2). Num BSC, a probabilidade de um símbolo ('0' ou '1') inverter-se por causa do ruído (tornando-se '1' ou '0', respectivamente) é $p_0 < 0.5$; e $q_0 = 1 - p_0$ é a probabilidade de não haver erro.

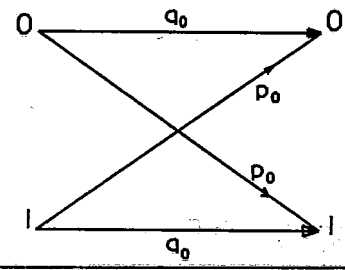


Figura II-2. Canal simétrico binário (BSC).

Neste caso, a probabilidade condicional é dada por:

$$p(\mathbf{r}|\mathbf{c}_l) = \prod_{i=0}^{n-1} p(r_i|c_{l_i})$$

Onde: $p(r_i|c_{l_i}) = q_0$ para $r_i = c_{l_i}$, e
$p(r_i|c_{l_i}) = p_0$ para $r_i \neq c_{l_i}$.

Se existem d_l posições de diferença:

$$p(\mathbf{r}|\mathbf{c}_l) = q_0^{(n-d_l)} \cdot p_0^{d_l} .$$

Como $q_0 > p_0$, $p(\mathbf{r}|\mathbf{c}_l)$ decresce quando d_l cresce. Assim, achar $\mathbf{c}_t$ tal que $p(\mathbf{r}|\mathbf{c}_t)$ é máxima equivale a achar a palavra-código $\mathbf{c}_t$ que difira de $\mathbf{r}$ no menor número de posições.

Sempre que possível, é preferível usar códigos de maior tamanho de bloco. Em geral os erros de transmissão aparecem agrupados (em "clusters", ou rajada), e blocos maiores compensarão o fato de haver trechos com poucos erros e outros trechos com mais erros.

Exemplo: Um código (fictício) de n = 2000 e k = 1000 que corrija 100 erros comparado com um outro (também fictício) com n = 200, k = 100 e que corrija 10 erros. Este segundo corrige 100 erros se estiverem bem distribuídos; mas se um bloco contiver 11 erros e outro bloco contiver zero

erros, o bloco com erro não será corretamente decodificado (e no total, há somente 11 erros << 100 erros).

O custo de se aumentar n é a complexidade do codificador e do decodificador.

Códigos blocados são avaliados por 3 parâmetros: o comprimento do bloco n, o tamanho da informação k, e a distância mínima d^*. Esta última mede a dessemelhança (diferença) entre as duas palavras-códigos mais parecidas, baseada nas duas definições seguintes:

Def.II-1: A **distância de Hamming** $d(\mathbf{x},\mathbf{y})$ entre duas sequências **x** e **y** de comprimento n é o número de posições em que as sequências diferem.

Por exemplo, sejam **x** = 10101, e **y** = 01100. Então, $d(\mathbf{x},\mathbf{y}) = 3$.

Def.II-2: Seja $C = \{ c_i, i = 0...(M-1) \}$ um código. Então a **distância mínima** d^* do código C é a distância de Hamming mínima obtida tomando-se todas as palavras-códigos duas a duas:

$$d^* = \text{mín } d(\mathbf{c}_i , \mathbf{c}_j),\ \mathbf{c}_i, \mathbf{c}_j \in C,\ i \neq j$$

Refere-se a um código blocado (n,k) com distância mínima d^* como (n,k,d^*).

Se t erros ocorrem durante uma transmissão, e se a distância de Hamming entre a palavra recebida **r** e a palavra-código $\mathbf{c}_i$ é maior que t, *exceto* por $\mathbf{c}_0$, para quem $d(\mathbf{r},\mathbf{c}_0) \leq t$, então presume-se que a palavra transmitida foi $\mathbf{c}_0$, recebida com t erros. Esta situação ocorre sempre se $d^* \geq 2.t + 1$. Ou seja, a distância mínima deve ser maior que o dobro da quantidade de erros presumida para haver decodificação.

Eventualmente, para determinados padrões de erro, é possível haver correção sem que a desigualdade seja satisfeita, mas não há garantia para todos os padrões de erro que não satisfaçam a desigualdade. A Fig.II-3 ilustra a situação geometricamente.

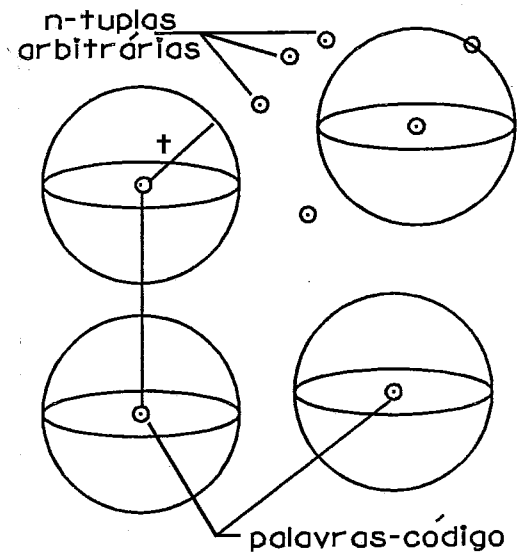


Figura II-3. Esferas de decodificação.

No espaço de todas as n-tuplas, algumas são escolhidas como palavras-códigos ($\mathbf{c}_i$, $\mathbf{c}_j$, e $\mathbf{c}_k$). Se d^* é a distância mínima deste código e t é o maior inteiro satisfazendo: $d^* \geq 2.t + 1$, então esferas dijuntas de raio t podem ser construídas em torno de cada palavra-código. Uma palavra recebida que pertença a uma determinada esfera é decodificada como a palavra-código centro desta esfera, o que corrige t ou menos erros.

Se mais de t erros ocorreram, a palavra recebida pode pertencer a outra esfera, e será decodificada incorretamente. Ou pode estar entre as esferas, sem pertencer a nenhuma, sendo tratada por uma de duas maneiras.

Se se trata de um decodificador incompleto (caso mais comum na prática), a palavra recebida não é decodificada, tratando-se de uma padrão de erro não-corrigível.

Se se trata de um codificador completo, a palavra recebida será decodificada como a palavra-código centro da esfera mais próxima, ou uma das mais (igualmente) próximas, arbitrariamente escolhida.

Além do erro, pode ocorrer um **apagamento**, isto é, não é possível ao receptor decidir se um símbolo é "0" ou "1" (no caso binário), mas sabe-se que existe um símbolo na posição em questão. Não confundir com **remoção**, quando uma sequência ("abcde") perde um de seus elementos ("abce"). Se

o código tem distância mínima d^*, então qualquer padrão com ρ apagamentos pode ser preenchido se $d^* \geq \rho + 1$.

Se $d^* \geq 2.t + 1 + \rho$, padrões com t erros e ρ apagamentos podem ser decodificados pois, se removermos as ρ posições apagadas, obtemos um novo código com distância mínima $d'^* \geq d^* - \rho$, que corrige t erros ($d^* - \rho \geq 2.t + 1$) e recuperamos a palavra reduzida (com $n' = n - \rho$). Agora, ao re-inserirmos os ρ apagamentos temos a distância mínima $d^* \geq \rho + 1$, suficiente para corrigí-los.

A teoria de Códigos de Controle de Erros baseia-se grandemente na Álgebra Moderna. Veremos alguns conceitos que nos permitirão formalizar a definição de um código.

II.4 - *Aritmética no Campo de Galois*

Se temos uma quantidade q^m de símbolos que seja um número primo q a uma potência m, então é possível definir uma adição e uma multiplicação neste conjunto de símbolos para as quais a maioria das regras aritméticas se aplicam. A esta estrutura dá-se o nome de **campo** (pois a subtração e a divisão também são possíveis).

Para um conjunto binário ($q = 2$, $m = 1$), define-se a soma módulo-2 ("+") e a multiplicação (".") da seguinte maneira:

0 + 0 = 0	0 . 0 = 0
0 + 1 = 1	0 . 1 = 0
1 + 0 = 1	1 . 0 = 0
1 + 1 = 0	1 . 1 = 1

E temos um **campo de Galois**, ou **GF(2)** neste caso.

Podemos construir vetores e matrizes, e calcular seus determinantes em GF(2). Podemos também usar polinômios, onde os coeficientes são 0 ou 1 somente.

Exemplo: Polinômio em GF(2): $f(X) = X^4 + X^3 + X^2 + 1$, $f(1) = 0$, isto é, 1 é raiz de f(X), e f(X) é divisível por $(X-1) = (X+1)$.

Um polinômio p(X) de grau m é irredutível em GF(2) se não é divisível por nenhum polinômio de grau de m', $0 < m' < m$.

Campos com 2^m símbolos são denotados por GF(2^m) e são usados no estudo de códigos. Em particular, são usados na decodificação de códigos BCH.

Uma aritmética com 2^m símbolos é derivada a partir da aritmética de 2 símbolos e um polinômio p(X), de grau *m*. Define-se então que para o símbolo α, $p(\alpha) = 0$ (como 2 = 0 na aritmética binária). Se escolhemos p(X) apropriadamente, as potências de α, α^i, para i = 1,2,...,(2^m-2), serão diferentes, $\alpha^{2^{(m-1)}} = 1$ e { 0, 1, α, α^2, ..., $\alpha^{2^{(m-2)}}$ } será o conjunto de 2^m símbolos do campo; e cada elemento poderá ser expresso como soma dos elementos 1, α, α^2, ..., α^{m-1}.

Exemplo: Para m=4, $p(X) = X^4 + X + 1$, $p(\alpha)=0$, e $0 = \alpha^4+\alpha+1 \Rightarrow \alpha^4 = \alpha+1$, resultando:

0
1
α
α^2
α^3
$\alpha^4=\alpha+1$
$\alpha^5=\alpha^2+\alpha$
$\alpha^6=\alpha^3+\alpha^2$
$\alpha^7=\alpha^3+\alpha+1$
$\alpha^8=\alpha^7+1$
$\alpha^9=\alpha^3+\alpha$
$\alpha^{10}=\alpha^2+\alpha+1$
$\alpha^{11}=\alpha^3+\alpha^2+\alpha$
$\alpha^{12}=\alpha^3+\alpha^2+\alpha+1$
$\alpha^{13}=\alpha^3+\alpha+1$
$\alpha^{14}=\alpha^3+1$
$\alpha^{15}=\alpha^4+\alpha=1$

Tabela II-1

Um elemento cujas potências resultem nos elementos da tabela (exceto pelo zero) é chamado de **elemento primitivo.** α e α^4 são elementos primitivos. α^3 não é elemento primitivo.

Um polinômio p(X) de grau *m* que gere a tabela completa com 2^m símbolos incluindo 0 e 1 é um **polinômio primitivo.** Ou, dito de outra forma, um polinômio irredutível de grau *m* é primitivo se $p(\beta)=0$, β elemento primitivo de GF(2^m). Para cada *m* inteiro positivo, existe ao menos um polinômio primitivo de grau *m*. Não é fácil reconhecer um

polinômio primitivo, existem tabelas para tanto, com em [41], e [43].

Para multiplicar símbolos, somamos os expoentes; e para a divisão, subtraímos, lembrando que $\alpha^{15}=1$ (ou em geral, $\alpha^{2^{(m-1)}} = 1$), temos, por exemplo:

$$\alpha^{12}.\alpha^{7} = \alpha^{19} = \alpha^{4}$$

$$\alpha^{12}/\alpha^{5} = \alpha^{7}$$

$$\alpha^{4}/\alpha^{12} = \alpha^{19}/\alpha^{12} = \alpha^{7}$$

Para a soma, recorremos à tabela. (Notar que como -1 = 1, subtrair é o mesmo que somar):

$$\alpha^{5}+\alpha^{7} = (\alpha^{2}+\alpha)+(\alpha^{3}+\alpha+1) = \alpha^{3}+\alpha^{2}+1 = \alpha^{13}$$

Podemos, como em GF(2), usar vetores e matrizes e determinantes sobre $GF(2^4)$.

Exemplo: para resolver a equação:

$$f(X) = X^{2}+\alpha^{7}X+\alpha = 0$$

não poderemos usar a fórmula comum, pois não poderemos dividir por 2 (2=0). É necessário resolver por tentativas, e verificar que:

$f(\alpha^{6}) = 0$, $f(\alpha^{10}) = 0$, e $f(X) = X^{2}+\alpha^{7}X+\alpha = (X+\alpha^{6}).(X+\alpha^{10})$.

Seja $f_i \in \{0, 1\}$ e $f(X)$ o polinômio
$f(X) = f_k.X^k + f_{k-1}.X^{k-1} + \ldots + f_1.X + f_0$.

Então, lembrando que $(b+c)^2 = b^2 + 2.b.c + c^2$:

$$f^2(X) = (f_k.X^k + f_{k-1}.X^{k-1} + \ldots + f_1.X + f_0)^2 =$$

$$= (f_k.X^k)^2 + 2.(f_k.X^k)(f_{k-1}.X^{k-1} + \ldots + f_1.X + f_0) +$$

$$+ (f_{k-1}.X^{k-1} + \ldots + f_1.X + f_0)^2$$

Como $1+1=2=0$ e $1.1=1^2=1$, temos:

$$f^2(X) = f_k.X^{2.k} + (f_{k-1}.X^{k-1} + \ldots + f_1.X + f_0)^2$$

que, expandida, resulta em:

$$f^2(X) = f_k.X^{2.k} + f_{k-1}.X^{2.(k-1)} + \ldots + f_1.X^2 + f_0 = f(X^2)$$

Segue-se que para qualquer inteiro positivo l,

$$[f(X)]^{2^l} = f(X^{2^l})$$

Seja β um elemento qualquer de $GF(2^m)$. O polinômio m(X) de menor grau com coeficientes binários tal que $m(\beta)=0$ é chamado **polinômio mínimo** de β, e é irredutível. Pelo resultado anterior, $[m(\beta)]^{2^l} = m(\beta^{2^l}) = 0$, e β^{2^l} é também raíz de m(X). Isto é, β, β^2, β^{2^2}, ..., β^{2^l}, ... são todos raízes de m(X). Como m(X) tem grau finito, tem um número finito de raízes e deve haver repetição na sequência anterior. Se e é o grau de m(X), as diferentes raízes são

$$\beta,\ \beta^2,\ \beta^{2^2},\ \ldots,\ \beta^{2^{(e-1)}}.$$

Para achar o polinômio mínimo de β, formamos a sequência das potências, encontramos as raízes distintas e formamos o produto dos fatores. Por exemplo:

$$\beta=\alpha^3,\ \beta^2=\alpha^6,\ \beta^{2^2}=\alpha^{12},\ \beta^{2^3}=\alpha^{24}=\alpha^9,\ \beta^{2^4}=\alpha^{48}=\alpha^3,\ \beta^{2^5}=\alpha^{96}=\alpha^6$$

as raízes são

$$\alpha^3,\ \alpha^6,\ \alpha^{12},\ \alpha^9 \quad e$$

$$m(X) = (X+\alpha^3).(X+\alpha^6).(X+\alpha^{12}).(X+\alpha^9) = X^4+X^3+X^2+X+1$$

II.5 - *Códigos Blocados Lineares*

Vimos que um código blocado (n,k) é um conjunto de palavras binárias de comprimento de n símbolos. Destes n símbolos, k são símbolos de informação, obtidos segmentando-se a mensagem, e o restante são símbolos (ou bits) de redundância. Como uma palavra-código é uma n-tupla de um espaço vetorial V de dimensão n, também é chamada de **vetor-código.**

Um **código linear** C de 2^k vetores de dimensão n é um conjunto de vetores que formam um sub-espaço vetorial do

espaço vetorial V de todos os vetores de dimensão n (todas as n-tuplas). Isto é, se $\mathbf{a},\mathbf{b} \in C \Rightarrow (k.\mathbf{a}+\mathbf{b}) \in C$.

Códigos lineares não são os melhores, mas ainda não existe uma teoria que sistematize a busca de códigos não lineares. Por isso são os estudados, e são conhecidos bons códigos lineares.

Podemos descrever um código linear de 2^k vetores por um conjunto de k vetores-código LI (base), formando a **matriz geradora** (com vetores-linha):

$$G_1 = \begin{bmatrix} V_1 \\ \dots \\ V_k \end{bmatrix} = \begin{bmatrix} V_{11} & V_{12} & \dots & V_{1n} \\ \dots & \dots & \dots & \dots \\ V_{k1} & V_{k2} & \dots & V_{kn} \end{bmatrix}$$

E o vetor-código $\mathbf{v}$ correspondente à mensagem $\mathbf{m}$ é dado por: $\mathbf{v} = \mathbf{m}.G_1 = m_1.\mathbf{v}_1+m_2.\mathbf{v}_2+\dots+m_k.\mathbf{v}_k$.

Nesta forma, não garantimos separação entre os bits de informação e os de redundância. Um **código sistemático** tem o vetor-código na seguinte forma (m_0, m_1, ..., m_k, p_0, p_1, ..., p_{n-k}) onde m_i é bit de informação e p_i é bit de redundância.

A matriz geradora para um código sistemático tem a seguinte forma:

$$G = \begin{bmatrix} 100\dots0 & p_{11}p_{12} & \dots & p_{1(n-k)} \\ 010\dots0 & p_{21}p_{22} & \dots & p_{2(n-k)} \\ \dots & \dots & \dots & \dots \\ 000\dots1 & p_{k1}p_{k2} & \dots & p_{k(n-k)} \end{bmatrix} = [\ I_k\ P\ ]$$

Onde I_k é a matriz identidade k x k, $p_{ij} \in \{0,1\}$ e P é a matriz formada pelos p_{ij}.

Pode-se verificar que, sendo $\mathbf{v}=\mathbf{m}.G$, $v_i=m_i$ para i=1...k, e $v_{k+j}=p_{1j}.m_1+p_{2j}.m_2+\dots+p_{kj}.m_k$, para j=1...(n-k).

Observe que, antes, o decodificador deveria armazenar todas as 2^k palavras-códigos, ao passo que agora é suficiente armazenar a matriz P, de k x (n-k) elementos.

Associada à matriz G de um código sistemático, existe a **matriz de verificação de paridade** H:

$$H = \begin{bmatrix} p_{11} & p_{21} & \cdots & p_{k1} & 100 & \cdots & 0 \\ p_{12} & p_{22} & \cdots & p_{k2} & 010 & \cdots & 0 \\ & & \cdots & \cdots & \cdots & & \\ p_{1(n-k)} & p_{2(n-k)} & \cdots & p_{k(n-k)} & 000 & \cdots & 1 \end{bmatrix} = [\ P^t\ I_{n-k}\ ]$$

onde P^t é a matriz P transposta, e $I_{(n-k)}$ é a matriz identidade (n-k). A matriz H é ortogonal ao espaço-linha (gerado pelos vetores linha) de G. Assim, se **v** está no espaço gerado por G (é vetor-código), então

$$\mathbf{v} \cdot H^t = \mathbf{0}$$

E o código pode ser descrito, alternativamente por H (ou por P^t).

Vimos a definição de distância de Hamming. O **peso de Hamming** de um vetor **v**, $w(\mathbf{v})$, é o número de componentes diferentes de zero. Pela definição da soma módulo-2, $d(\mathbf{u},\mathbf{v}) = w(\mathbf{u}+\mathbf{v})$.

Se **v** e **u** são vetores de um código linear C, então também $\mathbf{u}+\mathbf{v} \in C$ e $w(\mathbf{u}+\mathbf{v})$ é o seu peso. Então, a distância mínima, d^*, do código é o peso do vetor $\mathbf{v}\neq\mathbf{0}$, de menor peso.

Seja C um código linear (n,k) e $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{2^k}$ os vetores-códigos. Seja **r** o vetor recebido pelo decodificador. **r** pode não estar no espaço V_k, mas está no espaço V_n. A decodificação consiste em particionar os 2^n vetores (de V_n) em 2^k sub-conjuntos dijuntos $D_1, D_2, \ldots, D_{2^k}$, tais que cada D_i contenha $\mathbf{v}_i$, e **r** esteja contido no mesmo sub-conjunto que o vetor transmitido originalmente. Estes conjuntos são as esferas de decodificação. Se a união de todos eles, e somente eles, resultar em V_n, o código é dito **perfeito.**

Se, além de linear, um código (n,k) é tal que ao fazermos um deslocamento a direita sobre um qualquer vetor-código $\mathbf{v}=(v_0, v_1, \ldots, v_{n-1})$ obtemos um vetor $\mathbf{v}'=(v_{n-1}, v_0, v_1, \ldots, v_{n-2})$, também vetor-código, então o código é dito **cíclico.** Códigos cíclicos contêm uma inerente estrutura algébrica que os torna bastante interessantes.

A cada vetor-código podemos associar um **polinômio-código** da seguinte maneira:

$$\mathbf{v} = (v_0, v_1, \ldots, v_{n-1}) \quad <===> \quad v(X) = v_0+v_1.X+\ldots+v_{n-1}.X^{n-1}$$

Para o vetor $\mathbf{v}^{(i)}$, obtido por *i* deslocamentos sobre **v**:

$$\mathbf{v}^{(i)} = v_{n-i}+v_{n-i+1}.X+\ldots+v_0.X^i+\ldots+v_{n-i-1}.X^{n-1}$$

Pode ser demonstrado que $v^{(i)}(X)$ é o resto da divisão de $X^i.v(X)$ por (X^n+1):

$$X^i.v(X) = q(X).(X^n+1) + v^{(i)}(X)$$

Se o grau de $X^i.v(X)$ for (n-1) ou menor, então $v^{(i)}(X) = X^i.v(X)$.

Enunciaremos a seguir uma série de teoremas úteis na teoria de Códigos. Para as respectivas demonstrações, veja a referência [41].

Teorema II-1: Em um código cíclico (n,k), existe um e somente um polinômio-código g(X) de grau (n-k),

$$g(X) = 1+g_1.X+g_2.X^2+ \ldots + g_{n-k-1}.X^{n-k} .$$

Todo polinômio-código v(X) é múltiplo de g(X) e todo polinômio de grau (n-1) ou menor que seja múltiplo de g(X) é necessariamente um polinômio-código.

Segue-se que todo polinômio-código v(X) de um código cíclico (n,k) pode ser obtido pela multiplicação:

$$v(X) = m(X) \ . \ g(X)$$

E g(X) é chamado de **polinômio gerador** do código. Seu grau é (n-k), o número de bits de redundância.

Teorema II-2: O polinômio gerador g(X) de um código cíclico (n,k) é um fator de (X^n+1), ou seja:

$$X^n + 1 = g(X) \ . \ h(X) \ .$$

Teorema II-3: Se g(X) é um polinômio de grau (n-k) e é um fator de (X^n+1), então g(X) gera um código (n,k) cíclico.

Seja agora r(X) o vetor (ou polinômio) recebido:

$$r(X) = r_0+r_1.X+ \ \ldots \ +r_{n-1}.X^{n-1}$$

r_0, r_1, ... , r_{n-k-1} são os bits de informação, e os restantes r_{n-k}, ... , r_{n-1} são os de redundância. O primeiro passo na decodificação é calcular a **síndrome** s(X) de r(X), para verificar se r(x) é vetor-código ou não. Usa-se a seguinte expressão:

$$r(X) = p(X).g(X) + s(X)$$

s(X) é o resto da divisão de r(X) por g(X), e tem grau (n-k-1) ou menor. Isto é, **s** é uma $(n-k)$-tupla. Se $s(X) \neq \mathbf{0}$, então r(X) não é vetor-código, e r(X)=v(X)+e(X), onde e(X) é o padrão de erro.

Como v(X)=m(X).g(X), e(X)=[p(X)+m(X)].g(X)+s(X), e s(X) é o resto da divisão do padrão de erro pelo polinômio gerador.

Veremos agora dois exemplos de códigos perfeitos.

II.6 - *Códigos de Hamming e de Golay*

Seja p(X) um polinômio de grau m. O menor inteiro n tal que (X^n+1) é divisível por p(X) é (2^m-1). Um **código de Hamming** é tal que o polinômio gerador é um polinômio primitivo p(X) de grau m. O comprimento será $n=2^m-1$, e a quantidade de bits de informação $k=2^m-1-m$. Pode-se provar que a capacidade de correção é de no máximo t=1 erro, e que o código de Hamming é perfeito. Uma descrição recursiva do código de Hamming pode ser vista na referência [45].

O exemplo mais simples, mas não trivial, ocorre para m=3: $n=2^3-1=7$ e $k=2^3-1-3=4$. Trata-se do código (7,4), cujo polinômio gerador é $g(X) = p(X) = 1 + X + X^3$.

Observe que

$$2^4 . [C_7^0 + C_7^1] = 2^4 . [1 + 7] = 16 . 8 = 128 = 2^7$$

Isto é, somando as ocorrências de todos os erros possíveis com nenhum erro sobre os 2^4 vetores-códigos encontramos os 2^7 vetores de V_7, condição necessária (mas não suficiente) para que um código seja perfeito. De maneira geral, é necessário que:

$$\sum_{i=0}^{t} C_n^i = 2^{n-k}$$

para que um código (n,k), corrigindo até t erros seja perfeito.

Golay, em 1954, observou que

$$C_{23}^{0} + C_{23}^{1} + C_{23}^{2} + C_{23}^{3} = 2^{11}$$

sugerindo a existência de um código perfeito (23,12), corrigindo até 3 erros. De fato, existem 2 códigos, equivalentes, de Golay (23,12), perfeitos, com polinômios geradores:

$$g'(X) = X^{11} + X^{10} + X^{6} + X^{5} + X^{4} + X^{2} + 1, \text{ e}$$
$$g''(X) = X^{11} + X^{9} + X^{7} + X^{6} + X^{5} + X + 1 .$$

E verifica-se que, sobre GF(2):

$$(X + 1) \cdot g'(X) \cdot g''(X) = X^{23} - 1.$$

Os códigos de Golay e de Hamming são os únicos códigos binários perfeitos, além do código formado pela mera repetição de um símbolo (2.m+1) vezes (que corrige até *m* erros). Códigos de Hamming são códigos BCH (polinômio gerador $g(X) = p(X)$ e $n = 2^m-1$ - veja seção II.7), decodificados pelo algoritmo de Berlekamp, portanto. Códigos de Golay também podem ser decodificados por métodos algébricos, conforme Elia [20].

II.7 - *Códigos de Bose-Chaudhuri-Hocquenghem*

Os códigos BCH, foram inicialmente definidos como binários e depois generalizados para p^m símbolos, *p* primo, *m* inteiro positivo. São códigos cíclicos, e no caso binário, tem como parâmetros ([41]):

$$n = 2^m - 1, \quad n - k \leq m.t, \quad \text{e} \quad d \geq 2.t + 1 .$$

Seja α elemento primitivo de $GF(2^m)$, e a sequência:

$$\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2.t},$$

e $m_i(X)$ o polinômio mínimo de α^i. O polinômio gerador do código BCH corrigindo *t* erros é o mínimo múltiplo comum (LCM - Least Common Multiple):

$$g(X) = LCM(m_1(X), m_2(X), \ldots, m_{2.t}(X))$$

E a sequência dada é das raízes de $g(X)$: $g(\alpha^i) = 0$, $i=1,2,\ldots,2.t$.

Se *i* é par, $i=i'.2^l$, onde *i'* é ímpar e *l* é inteiro. Da Seção II-4, α^i e $\alpha^{i'}$ têm o mesmo polinômio mínimo: $m_i(X) = m_{i'}(X)$. Isto reduz o polinômio gerador a:

$$g(X) = \text{LCM}(m_1(X), m_3(X), \ldots, m_{2.t-1}(X)) \qquad \text{(II-1)}$$

$$\text{grau}(m_i(X)) \leq m \Rightarrow \text{grau}(g(X)) \leq m.t$$

Exemplo: Seja α elemento primitivo de $GF(2^4)$, $p(\alpha) = \alpha^4+\alpha+1$ (ou $\alpha^4 = \alpha+1$), $m_1(X)$, $m_3(X)$, e $m_5(X)$ polinômios mínimos de α, α^3, α^5, respectivamente. Para encontrar $m_1(X)$, formamos a sequência:

$$\alpha,\ \alpha^2,\ \alpha^{2^2}=\alpha^4,\ \alpha^{2^3}=\alpha^8,\ \alpha^{2^4}=\alpha^{16},\ \alpha^{2^5}=\alpha^{32}=\alpha^2$$

E α, α^2, α^4, e α^8 são as raízes de $m_1(X)$ (veja Tabela II-1):

$$m_1(X) = (X+\alpha).(X+\alpha^2).(X+\alpha^4).(X+\alpha^8) = 1 + X + X^4 .$$

Da mesma forma:

$$m_3(X) = 1 + X + X^2 + X^3 + X^4, \quad \text{e} \quad m_5(X) = 1 + X + X^2 .$$

De acordo com a equação (II-1), o código BCH para corrigir 2 erros, de comprimento $n=2^4-1=15$ é gerado por:

$$g(X) = \text{LCM}(m_1(X), m_3(X)) = m_1(X).m_3(X)$$

por serem ambos os polinômios diferentes e irredutíveis. Ou seja,

$$g(X) = 1 + X^4 + X^6 + X^7 + X^8$$

E o código é (15,7), cíclico com $d \geq 5$.

O código de Hamming apresentado anteriormente é uma sub-classe dos códigos BCH. No caso, o polinômio gerador é $m_1(X)$. Um possível uso de parte (subconjunto das palavras) de um código BCH é descrito em [46], de forma a obter melhor desempenho na decodificação.

A decodificação no código BCH é baseada no seguinte:

Sejam $v(X) = v_0 + v_1X + \ldots + v_{n-1}X^{n-1}$ o vetor transmitido,

$r(X) = r_0 + r_1X + \ldots + r_{n-1}X^{n-1}$ o vetor recebido, e

$e(X) = r(X) + v(X)$ o padrão de erro. (II-2)

Calcula-se o vetor síndrome **S** com $2.t$ componentes:

$$S_i = r(\alpha^i) = r_0 + r_1(\alpha^i) + \ldots + r_{n-1}(\alpha^i)^{n-1} \qquad \text{(II-3)}$$

para $i=1,2,\ldots,2.t$.

Das equações (II-2) e (II-3):

$$S_i = v(\alpha^i) + e(\alpha^i) = e(\alpha^i),\ i=1,2,\ldots,2.t,$$

pois α, α^2, ..., $\alpha^{2.t}$ são as raízes de $v(X)$.

Se $e(X)$ contem os erros:

$$e(X) = X^{j_1} + \ldots + X^{j_v} \qquad \text{(II-4)}$$

Então:

$$\begin{aligned} S_1 &= (\alpha^{j_1}) + \ldots + (\alpha^{j_v}) \\ S_2 &= (\alpha^{j_1})^2 + \ldots + (\alpha^{j_v})^2 \\ &\ldots \\ S_{2.t} &= (\alpha^{j_1})^{2.t} + \ldots + (\alpha^{j_v})^{2.t} \end{aligned} \qquad \text{(II-5)}$$

O procedimento de correção de erro será resolver as equações acima. Uma vez encontradas as potências de α^q, $q=j_1\ldots j_v$, cada q dirá a posição de erro em $e(X)$, como na eq. (II-4). Poderá haver mais de uma solução, sendo considerada a correta aquela em que o número de erros $v \leq t$. Para um valor grande de t, este método é pouco efetivo.

Vamos chamar de **número de localização de erro** a:

$$\beta_l = \alpha^{j_l},\ 1 \leq l \leq v$$

A equação (II-5) pode ser re-escrita como:

$$\begin{aligned} S_1 &= \beta_1 + \beta_2 + \ldots + \beta_v \\ S_2 &= \beta_1^2 + \beta_2^2 + \ldots + \beta_v^2 \\ &\ldots \\ S_{2.t} &= \beta_1^{2.t} + \beta_2^{2.t} + \ldots + \beta_v^{2.t} \end{aligned} \qquad \text{(II-6)}$$

S_1, S_2, ..., $S_{2.t}$ são conhecidas como funções simétricas de soma de potências ("power-sum symmetric functions").

Define-se o **polinômio de localização de erros** como:

$$\sigma(X) = (1+\beta_1 X).(1+\beta_2 X) \ldots (1+\beta_v X) = \sigma_0+\sigma_1 X+\ldots+\sigma_v X^v$$

Onde $\sigma_0 = 1$

$$\sigma_1 = \beta_1 + \beta_2 + \ldots + \beta_v$$

$$\sigma_2 = \beta_1.\beta_2 + \beta_1.\beta_3 + \ldots + \beta_{v-1}.\beta_v$$

$$\ldots \qquad \text{(II-7)}$$

$$\sigma_v = \beta_1.\beta_2.\beta_3\ldots\beta_v$$

E as raízes são β_1^{-1}, β_2^{-1}, ..., β_v^{-1}. De (II-6) e (II-7), vê-se que os coeficientes de $\sigma(X)$ relacionam-se com os componentes S_i, i=1,2,...,2.t, e, se for possivel encontrar $\sigma(X)$ dos S_i's, então os números de localização de erros podem ser encontrados e e(X) pode ser determinado. Os σ_1, σ_2, ..., σ_v são chamados de funções simétricas elementares de β_1, β_2, ..., β_v. O procedimento baseia-se nos seguintes 3 passos:

(1) Calcular a síndrome **S** = (S_1, S_2, ..., $S_{2.t}$)

(2) Encontrar $\sigma(X)$ (polinômio de localização de erros) de S_1, S_2, ..., $S_{2.t}$

(3) Determinar β_j (números de localização de erro), encontrando as raízes de $\sigma(X)$.

O primeiro passo é a equação (II-3). Os dois passos seguintes podem ser resolvidos com o algoritmo descrito na próxima seção.

II.8 - *Algoritmo para Decodificação de Códigos BCH*

Um problema computacional pode ser de classe P ou de classe NP, [7].

Um problema computacional de **classe P** é definido como o que pode ser resolvido num número de passos limitado por uma função polinomial do número de entradas.

Um problema de **classe NP** é tal que pode ser resolvido por um algoritmo não-determinístico num número de passos limitado por um polinômio no número de entradas. Um **algoritmo não-determinístico** é aquele que, quando confrontado

com uma escolha entre duas opções, cria uma cópia de si próprio, e, simultaneamente, segue executando por ambas as escolhas. A classe NP é a classe dos algoritmos Não-determinísticos de tempo Polinomial.

Finalmente, o conjunto de problemas (21, conforme Karp, *in* [7]) NP equivalentes, no sentido de que se se provar que um deles é da classe P, os demais também serão, é chamado de classe dos problemas **NP-completos.**

Assim, se for encontrado um algoritmo polinomial para um dos problemas NP-completo, o problema de decodificação também poderá ser resolvido em tempo polinomial.

Em particular, o problema de decodificação é NP-completo, [7], ou mesmo *NP-hard*, [11], isto é, não existe um procedimento geral, mas soluções para instâncias já estudadas.

Apresentamos a versão simplificada do algoritmo de Berlekamp para cálculo das posições de erro, para o caso binário, [41], também discutido em [18]. (Uma extensão do algoritmo para aumentar a sua capacidade de correção pode ser vista em [50].)

Dadas as síndromes $\mathbf{s}_i = r(\alpha^i)$, i=1,2...,2.t, seja a tabela a seguir construída recursivamente:

μ	σ_μ	d_μ	l_μ	$(2.\mu - l_\mu)$
$-\frac{1}{2}$	1	1	0	-1
0	1	$\mathbf{s}_1$	0	0
1				
2				
...				
t				

Presumindo que completamos até a linha μ, preenchemos a linha $(\mu+1)$:

(1) Se $d_\mu = 0$ então $\sigma^{(\mu+1)}(X) = \sigma^{\mu}(X)$;

(2) Se $d_\mu \neq 0$ então encontre outra linha ρ ($\rho < \mu$) tal que o número $(2.\rho - l_\rho)$ da última coluna seja o maior possível e $d_\rho \neq 0$. Então:

$$\sigma^{(\mu+1)}(X) = \sigma^{\mu} + d_{\mu} . d_{\rho}^{-1} . X^{2(\mu-\rho)} . \sigma^{(\rho)}(X)$$

Tanto no caso (1) como no caso (2), $l_{\mu+1}$ é o grau de $\sigma^{(\mu+1)}(X)$ e $d_{(\mu+1)}$ é dado por:

$$d_{\mu+1} = s_{2.\mu+3} + \sigma_1^{(\mu+1)} . s_{2.\mu+2} + \ldots = \sum_{j=0}^{l_{\mu+1}} \sigma_j . s_{2.\mu+3-j} \quad , \quad \sigma_0 = 1$$

O polinômio $\sigma^{(t)}(X)$, na última linha, deve equivaler ao vetor decodificado. Se o grau do polinômio for maior que t, houve mais de t erros e, no caso geral, não é possível localizá-los.

De posse do polinômio, verificamos quais elementos de GF(2^m), m inteiro, são suas raízes. Se α^l é raiz de $\sigma(X)$, então o dígito r_{n-l} na posição $(n-l)$ contem erro, e basta invertê-lo para corrigí-lo.

Este procedimento pode ser implementado em computador da seguinte forma (para m=4, por exemplo):

Seja o elemento de GF(2^4), GF(2^4) gerado pelo polinômio primitivo $p(X) = X^4 + X + 1$, representado por $f(\alpha) = f_0 + f_1\alpha + f_2\alpha^2 + f_3\alpha^3$. Isto é possível, pois para $j > 3$, α^j é uma soma de potências 0, 1, 2 ou 3 de α. Então o valor 1 pode ser representado pelo vetor [1 0 0 0], α por [0 1 0 0], zero por [0 0 0 0], etc, formando a tabela G_POTALFA, elmentos de GF(2^m), ordenados pelo expoente de α, estando o zero, por convenção, em primeiro lugar:

$$G\_POTALFA = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ & \ldots & & \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Um polinômio em X pode ser representado por uma matriz, em que a linha j é o coeficiente de $X^{(j-1)}$. Por

exemplo, $p(X) = \alpha X^2+\alpha^6X^3 = \alpha X^2+(\alpha^3+\alpha^2).X^3$ pode ser representado por:

$$\overline{P} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Podemos definir a operação de soma de elementos de $GF(2^m)$ como soma de vetores, módulo-2:

$$\alpha^6 + \alpha^5 = \alpha^3 + \alpha^2 + \alpha^2 + \alpha = \alpha^3 + \alpha = \alpha^9$$

$$[0\ 0\ 1\ 1] + [0\ 1\ 1\ 0] = [0\ 1\ 0\ 1]$$

E a multiplicação de dois elementos pode ser resolvida como $\alpha^i.\alpha^j = \alpha^{(i+2+j)-2}$, (i+2+j) sendo o subscripto do elemento procurado (lembrar que para qualquer elemento α^i, seu subscrito *k* é dado por k=i+2).

A potenciação é resolvida de maneira semelhante, lembrando que $(\alpha^j)^1 = \alpha^{j.1}$.

A multiplicação de 2 polinômios é um polinômio, também representado por matriz. É obtida tomando-se cada linha *i* da primeira matriz, e *j* da segunda matriz, e fazendo a multiplicação conforme acima, formando a linha *k* da matriz resultado: k = i+j-1. Exemplo:

$$(1+\alpha^5.X).(\alpha.X^2) = \alpha.X^2+\alpha^6.X^3 = \alpha.X^2+(\alpha^3+\alpha^2).X^3$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Este algoritmo e sua implementação fizeram parte do estudo de códigos. Uma descrição semelhante de implementação é encontrada em [6]. No entanto, para esta dissertação, o mais importante são os conceitos apresentados, fundamentais na definição do problema em estudo.

Além dos códigos apresentados até aqui, foi usado um dos códigos apresentados por Gamal e outros, [22], obtidos pelo algoritmo de *simulated annealing* (ou *SA*, resfriamento

simulado, técnica de otimização estatística, veja [35] ou [1]): dado um comprimento de código, um peso fixo para cada palavra-código, e um código inicial; o algoritmo executa "perturbando" (alterando randomicamente alguns bits) o código inicial e seguindo as perturbações que resultarem em maior distância de Hamming mínima para o código. Não se trata de um código linear, mas de peso constante. Além disso, não existe um algoritmo para codificação, que é feita por associação simplesmente.

No próximo capítulo, veremos os modelos de rede que foram usados.

Capítulo III

Redes Neurais Artificiais

Existem muitas definições para Redes Neurais Artificiais (ou Redes Neurais, ou ainda RN). E existem vários paradigmas (modelos) para implementação. Descreveremos o que são as Redes Neurais e os paradigmas usados neste trabalho.

III.1 - *Conceitos Básicos*

Uma **Rede Neural Artificial** é um conjunto de unidades de processamento relativamente simples (chamados nós, neurônios ou processadores), altamente interconectadas e que funcionam em paralelo. Basicamente, são projetadas para emular redes neuro-biológicas, e encontram aplicações em diversas áreas, como reconhecimento de voz, reconhecimento de padrões, aproximação de funções, etc, [2].

Uma definição alternativa é a de um **modelo matemático de rede neural**: um sistema não-linear de equações algébricas ou diferenciais de n dimensões que respondem pela dinâmica de n **neurônios.** No caso de um sistema dinâmico, cada neurônio é matematicamente um estado a_i (um número real) com uma saída associada $f_i = f_i(a_i)$, [31].

Características importantes das redes neurais são o paralelismo de seu funcionamento, a simplicidade de seus elementos (neurônios) e a sua capacidade de simular habilidades cognitivas.

Segundo Rumelhart e McClelland [47], os principais aspectos que devem ser considerados numa rede neural - chamada também de modelo de processamento distribuído paralelo - são: o conjunto de **unidades de processamento**, o **estado de ativação da rede** (vetor dos estados de cada unidade), a **função de saída** para cada unidade, o **padrão de conexão** entre as unidades, a **regra de propagação** dos padrões de atividades pela rede de conexões, a **regra de ativação** , a **regra de aprendizado**, e o **ambiente** onde este sistema opera. A Figura III-1 ilustra os componentes básicos dos quais se constitui uma rede.

Em cada instante t, cada unidade u_i (uma das N unidades da rede) tem um valor de ativação $a_i(t)$; esta ativação passa por uma função f_i para gerar uma saída $o_i(t)$. Este valor é passado pelas conexões unidirecionais para outras unidades da rede. Para cada conexão da rede, existe um número real associado chamado de **peso** ou **força** da conexão, w_{ij}, da unidade u_i para a unidade u_j. Se o peso é positivo, a entrada correspondente é **excitatória**, se negativo, é **inibitória**. Na unidade u_j, as entradas são combinadas por algum operador (em geral, a adição, resultando no valor *net*, *entrada líquida*) e, junto com a ativação atual da unidade, são parâmetros para a função F (a regra de ativação), que fornece o novo valor de ativação da unidade.

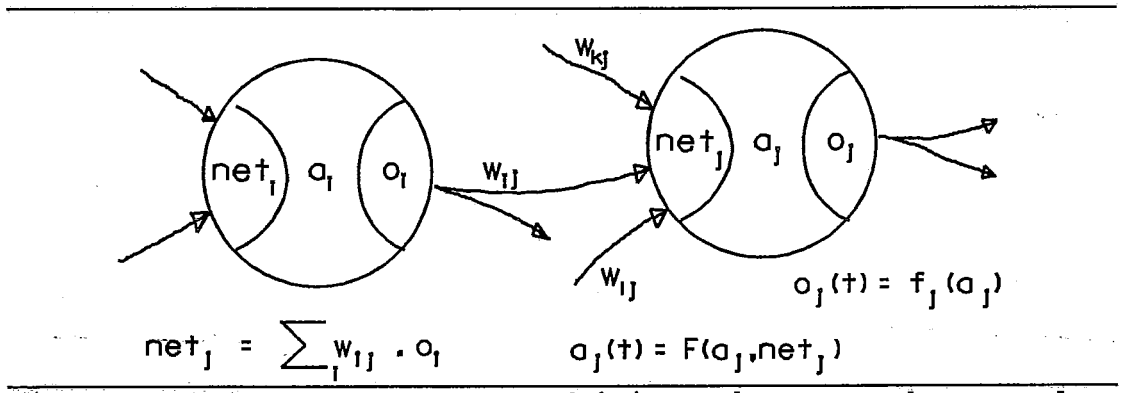


Figura III-1. Os componentes básicos de uma rede neural.

Para cada rede, podemos chamar de unidades de **entrada** aquelas que recebem valores do exterior (ambiente), unidades **escondidas**, aquelas que não se comunicam com o exterior, e unidades de **saída**, aquelas cujos valores de saída são enviados para o exterior. Cada sub-conjunto de unidades que operam de maneira semelhante é chamado de **camada** da rede. Temos então a camada de entrada, camadas escondidas e a camada de saída (*input*, *hidden*, e *output*).

O problema do aprendizado é definido como encontrar um algoritmo para modificar os pesos e os limiares das conexões e dos neurônios da rede que minimizem um critério (função) externo, relevante para a tarefa (mapeamento da entrada para a saída) em questão, [2].

Existem muitas regras de aprendizado, que são funções pelas quais os pesos da rede se alteram de forma a que a rede se adapte ao meio (isto é, forneça saídas por algum critério melhores para as entradas que lhe são aplicadas). A maioria destas regras são heurísticas baseadas na regra de Hebb (hebbiana): se uma unidade recebe um valor de outra e ambas estão ativadas, então o peso da conexão entre ambas deve ser reforçado. Matematicamente, de uma maneira geral:

$$\Delta w_{ij} = g(a_j(t), t_j(t)).h(o_i(t), w_{ij})$$

onde $t_j(t)$ é uma espécie de **valor professor** da unidade u_j, que "mostra" à unidade qual o valor desejado.

Em sua versão mais simples, sem o valor professor, a regra hebbiana se reduz a:

$$\Delta w_{ij} = \eta a_j o_i$$

onde η é chamado de **taxa de aprendizado.**

Uma variação comum é a regra de Widrow-Hoff ou **regra delta,** na qual o aprendizado (variação no peso) é proporcional à diferença (ou delta) entre a ativação corrente e a ativação desejada, fornecida pelo "professor":

$$\Delta w_{ij} = \eta (t_j(t) - a_j(t)).o_i(t)$$

O ambiente é representado pelos valores de entrada na primeira camada da rede (camada de entrada), que se presumem sempre pertencentes a um determinado domínio. No caso de valores discretos de um conjunto finito, pode-se supor uma lista de m padrões, numerados de 1 a m. Cada padrão (ou vetor) de entrada é formado pelos valores de entrada em cada unidade.

Redes neurais podem operar em dois modos: **aprendizado** (ou *learning*) e **recuperação** (ou *recall*). No

primeiro, padrões são apresentados na entrada e seus correspondentes são apresentados na saída, e os pesos variam conforme a regra de aprendizado. Durante a recuperação, a saída da rede é lida ao serem apresentados padrões na entrada, e os pesos não sofrem variação.

Dois paradigmas de aprendizado importantes são associação de padrões e a auto-associação. Na **associação de padrões**, o objetivo é mapear padrões definidos na entrada da rede com outros padrões definidos na camada de saída da rede, encontrando o conjunto de pesos que fará com que a rede forneça na saída o padrão correspondente ao da entrada, quando este for apresentado. Em geral, usa-se o **aprendizado supervisionado**, onde o valor desejado é fornecido à camada de saída da rede, até serem encontrados os pesos mais apropriados.

Um caso particular importante é a **auto-associação**, em que a rede deve mostrar na saída o mesmo padrão que lhe é apresentado na camada de entrada. A aplicação deste paradigma é a **completação de padrões**, isto é, a rede deve recuperar um padrão completo ao ser aplicada uma **parte** do padrão, implementando uma **memória de acesso por conteúdo.**

Dois outros paradigmas são a **classificação**, em que padrões e suas respectivas categorias são apresentados à rede, que deve tornar-se capaz de classificar um padrão ligeiramente alterado; e o **detector de regularidades**, onde não existe um conjunto fixo de categorias dado, mas a rede é que detecta estatisticamente as características mais importantes [21].

Uma questão importante também a ser considerada ao se modelar uma rede neural é o modo ("ritmo") com o qual as funções de ativação são atualizadas (recalculadas). Pode ser **síncrono**, quando há um relógio central e a cada intervalo todas as unidades determinam o novo valor de ativação. Ou **assíncrono**, quando em cada instante, existe uma probabilidade fixa de cada unidade executar a atualização do seu valor de ativação. Este modo tem uma importante vantagem teórica. Num intervalo suficientemente pequeno, somente uma unidade estará

sofrendo atualização, e este sistema não tem tendência de ficar oscilando entre um número de estados.

Uma corrente de pesquisadores pretende com os modelos de redes neurais (ou **modelos conexionistas**) substituir a "metáfora computacional", o uso de modelos de computador para explicar os mecanismos da mente, por uma "metáfora do cérebro". Uma propriedade interessante das redes neurais que pode não ter ficado clara é a de que o conhecimento está nas *conexões*, e não em pontos localizados. As ativações dos neurônios servem apenas como memória de curto prazo. A memória de longo prazo está nas conexões entre os neurônios, e o conhecimento está muito mais *implícito* na estrutura do que explícito em regras.

Veremos agora os modelos (paradigmas) de redes usadas neste trabalho.

III.2 - *Modelo Backpropagation*

Uma **rede backpropagation** (retro-propagação) basicamente constitui-se de 3 camadas de unidades de processamento: 1 camada de **entrada F_A ou I** (*input*), 1 (ou mais) camada **escondida F_B ou H** (*hidden*), e 1 camada de **saida F_C ou O** (*output*), veja a Figura III-2. A informação é levada no sentido da camada **I** para a camada **O**. Ocorre aprendizado tanto na camada **H** (conexões de **I** para **H**), como na camada **O**.

É uma rede **hetero-associativa**, isto é, associa pares arbitrários de padrões, sendo capaz de estimar valores de funções de $\mathbf{R}^n$ em $\mathbf{R}^q$ (capacidade de generalização), [49]. Opera no modo de aprendizado e no modo de recuperação de informação. O algoritmo de aprendizado (algoritmo *backpropagation*) visa a correção do erro em relação ao sinal "professor", baseado no método do gradiente aplicado às múltiplas camadas, desenvolvido independetemente por Bryson e Ho (1969), Werbos (1974), Parker (1982), e Rumelhart, Hinton e Williams (1986) (veja [47]).

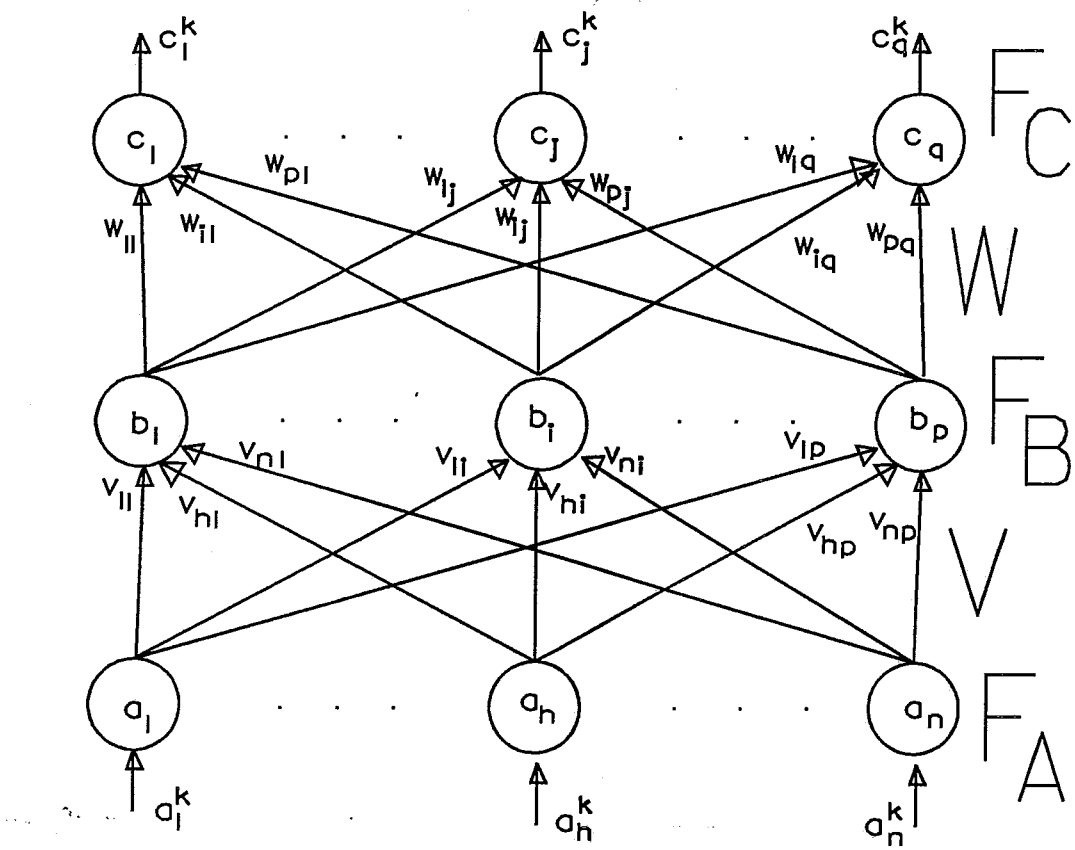


Figura III-2. Rede backpropagation.

O algoritmo de aprendizado baseia-se em minimizar uma função de custo, em geral, a função erro quadrático, equação (III-1), soma dos erros obtidos para cada diferente padrão apresentado, equação (III-2).

$$E = \sum E_k \qquad \text{(III-1)}$$

$$E_k = \frac{1}{2} \sum_j (t_{kj} - o_{kj})^2 \qquad \text{(III-2)}$$

sendo E o erro a ser minimizado; E_k, o erro em um determinado padrão k; t_{kj}, o sinal professor (valor desejado) no neurônio (unidade de processamento) j; e o_{kj}, o valor obtido em cada neurônio da camada **O**.

O algoritmo pode ser delineado da seguinte forma:
(1) Assinale valores randômicos no intervalo $[-1/\sqrt{N+1} \ , \ 1/\sqrt{N+1}]$ a cada peso das conexões v_{hi} e w_{ij} entre

as camadas **I** e **H**, e entre as camadas **H** e **O**, respectivamente; e também para cada limiar T_i dos processadores da camada **H**, e cada limiar S_j dos processadores da camada **O** (aqui, N é a quantidade de conexões que alimentam cada neurônio - $N=n$ para a camada **H**, e $N=p$ para a camada **O**).

(2) Para cada par de padrões (A_k, C_k), k=1, 2, ..., m, faça:
(2.a) Aplique os valores de A_k sobre a camada de entrada, e os valores de saída da camada de entrada são os valores de entrada da camada seguinte, a escondida. As ativações de cada neurônio da camada escondida são calculadas por:

$$b_i = f\left(\sum_{h=1}^{n} a_h v_{hi} + T_i \right)$$

Para todo i=1,2,...,p, p número de neurônios na camada escondida, onde b_i é o valor de ativação do neurônio i da camada escondida, T_i é o seu valor de limiar, e $f()$, em geral, é a função sigmóide logística de limiar dada por:

$$f(x) = \frac{1}{(1 + e^{-x})} \qquad \text{(III-3)}$$

(2.b) Filtre os valores das ativações da camada escondida usando W, matriz de conexões entre as camadas **H** e **O**, para aplicar os valores na camada **O**, de saída, usando:

$$c_j = f\left(\sum_{i=1}^{q} b_i w_{ij} + S_j \right)$$

Para todo j=1,2,...,q, q número de unidades na camada de saída, onde c_j é o valor de ativação da unidade j na camada de saída, e S_j, o seu valor de limiar, e $f()$ dada pela equação (III-3).

(2.c) Calcule a discrepância (erro) entre os valores de saída obtidos e os desejados ($c_j^k - c_j$), e multiplique pela derivada da função de ativação. No caso da função logística:

$$d_j = (c_j^k - c_j)(1 - c_j)c_j$$

Para todo j=1,2,...,q, onde d_j é o valor de erro a ser retropropagado.

(2.d) Calcule o erro a ser retropropagado de cada unidade de processamento da camada escondida relativo a cada d_j:

$$e_i = b_i(1-b_i)\sum_{j=1}^{q} w_{ij}d_j$$

Para todo i=1,2,...,p; onde e_i é o erro retropropagado relativo ao valor no i-ésimo processador da camada escondida.

(2.e) Ajuste as conexões entre as camadas escondida e de saída (regra Delta de aprendizado):

$$\Delta w_{ij} = \Delta w_{ij}^{(t)} = \eta_1 . b_i . d_j$$

Para todo i=1,2,...,p, e para todo j=1,2,...,q; onde Δw_{ij} é a variação feita na conexão do neurônio i da camada escondida para o neurônio j da camada de saída, η_1, constante positiva, é a taxa de aprendizado.

(2.f) Ajuste os limiares da camada **O**:

$$\Delta S = \lambda_1 d_j$$

Para todo j=1,2,...,q, onde ΔS_j é a variação no limiar do processador j da camada de saída.

(2.g) Ajuste as conexões entre as camadas de entrada e escondida:

$$\Delta v_{hi} = \eta_2 . a_h . e_i$$

Para todo h=1,2,...,n, e para todo i=1,2,...,p; onde Δv_{hi} é a variação na conexão do processador h da camada de

entrada para o processador i na camada do meio (escondida), e η_2 é a taxa de aprendizado.

(2.h) Ajuste os limiares da camada escondida:

$$\Delta T_i = \lambda_2 e_i$$

Para todo i=1,2,...,n; onde ΔT_i é a variação no limiar do processador i da camada escondida.

(3) Repita o passo (2) até que o valor do erro d_j para cada j=1,2,...,p, e para cada k=1,2,...,m, seja suficientemente pequeno.

A regra de aprendizado pode ser aplicada a cada apresentação de um grupo de pares de padrões (em **batelada**), usando-se a média dos erros na retroprogação.

Também pode ser aplicada a regra Delta com **momentum** (λ), constante positiva que traz uma parte da variação do peso na iteração anterior, $\Delta w_{ij}^{(t-1)}$:

$$\Delta w_{ij} = \eta_1 . b_i . d_j + \lambda_1 . \Delta w_{ij}^{(t-1)}$$

para a camada de saída, e equivalentemente para a camada do meio:

$$\Delta v_{hi} = \eta_2 . a_h . e_i + \lambda_2 . \Delta v_{hi}^{(t-1)}$$

A rede backpropagation opera no modo de recuperação a partir de valores lidos em sua camada de entrada **I**, e passados, pelas conexões entre **I** e **H** (matriz V de conexões), para a camada escondida **H**, onde as ativações b_i são calculadas:

$$b_i = f(\sum_{h=1}^{n} a_h . v_{hi} + T_i)$$

para todo i=1,2,...p.

Obtidos todos os valores b_i, o processo se repete para a camada seguinte (de saída):

$$c_j = f(\sum_{i=1}^{p} b_i . w_{ij} + S_j)$$

para todo j=1,2,...,q. Obtem-se, então, o valor de saída correspondente à entrada.

Garantidamente, o algoritmo encontrará um mínimo local para a função de custo (erro), mas como garantir que se encontre o mínimo global ainda é uma questão em aberto. Outras questões em aberto são o número de processadores (ou neurônios) da camada escondida, os valores para as taxas de aprendizado, e a quantidade de padrões necessária para o aprendizado.

Lehmen [39] faz um estudo sobre os fatores que influenciam o desempenho deste algoritmo. Usa ruído nos pesos das conexões (para que a busca na vizinhança do ponto seja completa), compara pesos analógicos e discretos, e limitação (*clamping*) nos valores dos pesos (para limitar a vizinhança de busca). O uso de ruído parece melhorar bastante o desempenho.

Aazhang e Henson [2] estudam backpropagation como um "otimizador local", aliado ao *simulated annealing* (resfriamento simulado - técnica de otimização estatística) como um "otimizador global". Isto é simular ao uso de ruído citado anteriormente. Em ambos os casos, em princípio, há o custo de tempo de processamento elevado.

A vantagem do algoritmo está em sua grande capacidade de armazenamento, sua capacidade de aproximar funções não-lineares, generalização a partir dos dados fornecidos, e sua tolerância a falhas (perda de um número pequeno de processadores).

III.3 - *Modelo de Hopfield*

A **rede de Hopfield**, descrita em [25], mas proposta por outros pesquisadores já em 1943 (McCulloch e Pitts), tem uma única camada (F_A) e possui conexões simétricas. É um

codificador não-linear e auto-associativo, que armazena padrões $A_k = (a_1^k,...,a_n^k)$, k=1,2,...,m, binários (em {0,1}) ou bipolares ({-1,+1}), usando aprendizado hebbiano (veja seção III.1). A atualização dos processadores é assíncrona (em cada instante, um dos processadores, escolhido randomicamente, é atualizado [16]), e opera em instantes discretos do tempo.

Cada nó tem uma conexão de e para cada um dos outros nós, e uma conexão recorrente (para si próprio) - veja Figura III-3, de Simpson [49].

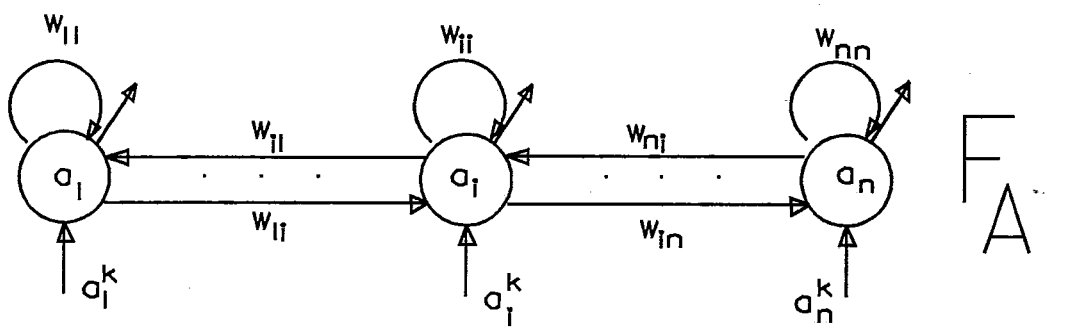


Figura III-3: Modelo de Hopfield.

Este modelo tem particular importância por Hopfield ter popularizado uma analogia entre a estabilidade da rede e uma função de energia de Lyapunov, mostrando ser este um modelo simples e poderoso, com propriedades coletivas não totalmente previsíveis a partir do modelo do neurônio individual, e capaz de implementar **memória associativa** (de acesso pelo conteúdo, e não a partir de um endereço).

A codificação é feita usando a equação:

$$W = \sum_{k=1}^{m} A_k^t . A_k$$

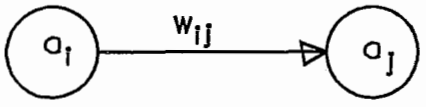
onde W é a matriz dos pesos das conexões, e A_k é o vetor-linha representando o padrão. É equivalente a:

$$w_{ij} = \sum_{k=1}^{m} a_i^k . a_j^k$$

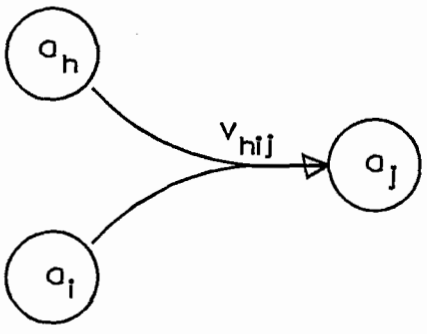
$$i=1,2,\ldots,n;\ j=1,2,\ldots,n$$

onde w_{ij} é o peso da conexão entre o processador i e o processador j, e $w_{ij} = w_{ji}$.

Uma extensão que pode ser feita é usar a codificação de segunda ordem (ou uma ordem mais alta). A Figura III-4 mostra conexões de primeira e de segunda ordem.



conexão de primeira ordem



conexão de segunda ordem

Figura III-4. Conexões de primeira e de segunda ordem.

A codificação de segunda ordem atualiza os pesos da seguinte maneira:

$$v_{hij} = \sum_{k=1}^{m} a_h^k . a_i^k . a_j^k$$

$$h=1,2,\ldots,n;\ i=1,2,\ldots,n;\ j=1,2,\ldots,n$$

Onde v_{hij} é o peso da conexão dos nós h e i para o nó j, e $v_{hij} = v_{ijh} = v_{jhi}$. A matriz de conexões V é uma matriz n x n x n.

A recuperação na rede de Hopfield de primeira ordem é feita com a equação (III-4):

$$a_i(t+1) = f(\sum_{j=1}^{n} w_{ij} . a_j(t)) \qquad \text{(III-4)}$$

onde $a_j(t)$ é ativação do processador j no instante t, e $f()$ é a função limiar:

$$f(x) = \begin{cases} 1 & se\ x > 0 \\ 0 & de\ outra\ forma \end{cases}$$

A recuperação é uma operação que usa feedback (retro-alimentação) durante a qual aplica-se repetidamente a equação (III-4), até que todos os processadores não mais sofram alteração.

Para redes de segunda ordem, a equação (III-4) é substituída por:

$$a_j(t+1) = f(\sum_{h=1}^{n} \sum_{i=1}^{n} a_h(t) . a_i(t) . v_{hij})$$

Para mostrar a estabilidade da rede, Hopfield usou a função de Lyapunov (expressando uma "energia computacional") (veja também Carvalho, [16]):

$$L(A) = -\frac{1}{2}\sum_{\substack{i=1 \\ i\neq j}}^{n}\sum_{j=1}^{n} a_i . a_j . w_{ij} \qquad \text{(III-5)}$$

Onde $A=(a_1,\ldots,a_n)$ é um vetor arbitrário das ativações em F_A, e os pesos $w_{ii}=0$ (depois, esta restrição foi relaxada, por McEliece, para $0\leq w_{ii}\leq n$, [57]).

A partir da equação (III-5), tomando a derivada a instantes discretos de L(A) em relação a a_i, pode-se obter:

$$\Delta L(A) = -[\sum_{j=1}^{n} a_j w_{ij}] . \Delta a_i$$

Onde para qualquer $\Delta a_i \neq 0$, $\Delta L(A) < 0$, ou seja, a energia do sistema sempre tende a diminuir e o sistema é estável.

O número de padrões que podem ser armazenados por uma rede de Hopfield é, aproximadamente:

$$m = \frac{n}{2.\log n + \log\log n} \approx \frac{n}{2.\log n}$$

A baixa capacidade é a principal limitação deste modelo (para um estudo da capacidade da rede de Hopfield, veja [3]). As vantagens são sua capacidade de recompor padrões a partir de entradas incompletas, sua simplicidade, e sua tolerância a falhas. Além disso, por ser assíncrona, é uma rede que se presta bem a implementação.

III.4 - *Modelo BAM*

Uma **rede BAM - Bidirectional Associative Memory** (Memória Associativa Bi-direcional), Figura III-5, codifica (armazena) **pares** de padrões (A_k, B_k), e não apenas padrões simples (A_k, A_k). É uma rede de duas camadas com *feedback* (retro-alimentação), que armazena padrões binários ($\{0,1\}$) ou bipolares ($\{-1, 1\}$).

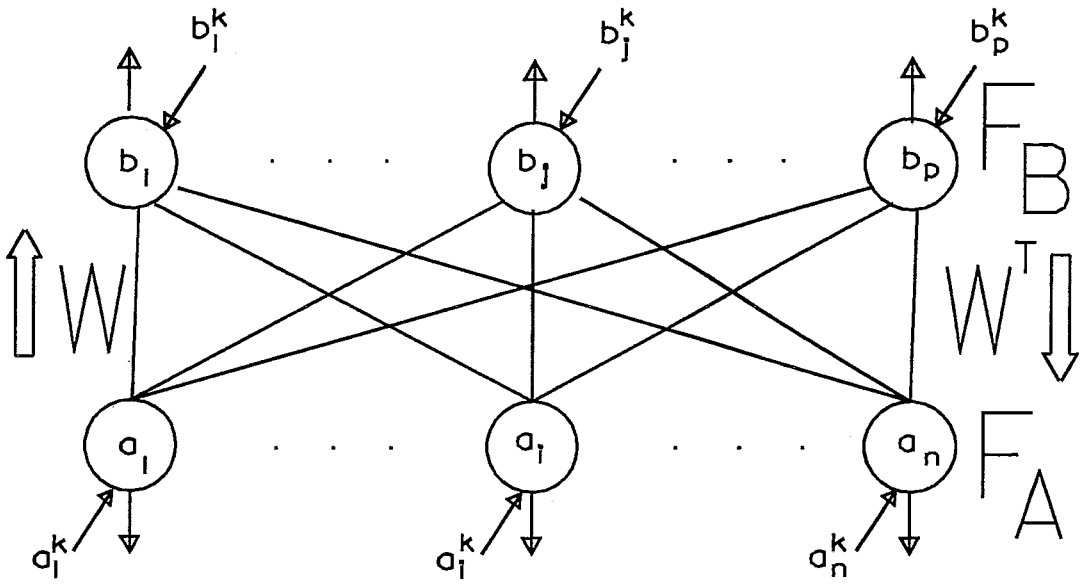


Figura III-5. Modelo BAM.

A rede BAM, proposta por Bart Kosko [37], é um correlacionador de padrões, hetero-associativo, que fornece o padrão correspondente mais próximo dentre os padrões armazenados, ao ser apresentado um padrão na entrada. O aprendizado é hebbiano (veja seção III-1), onde o par k de padrões (A_k, B_k), $A_k = (a_1^k, \ldots, a_n^k)$, $B_k = (b_1^k, \ldots, b_p^k)$, é representado pelas camadas F_A e F_B da rede, respectivamente ([49]).

O aprendizado é feito segundo a equação:

$$W = \sum_{k=1}^{m} A_k^t \times B_k$$

Onde os vetores-linha A_k e B_k representam os m padrões sendo codificados. W é a **matriz de memória** (n x p), que armazena as associações. Uma forma equivalente é:

$$w_{ij} = \sum_{k=1}^{m} a_i^k . b_j^k$$

onde w_{ij} é o peso da conexão do processador i da camada F_A para o processador j da camada F_B.

A recuperação é feita a partir de um padrão inicial ou a partir de um par incompleto, com os seguintes passos:

(1) Apresentar um padrão para a camada F_A, ou um padrão para a camada F_B, ou, ainda, parte de padrões tanto para a camada F_A como para F_B.
(2) Usar as ativações da camada F_A, através das conexões (matriz W), em F_B .
(3) Calcular as ativações em F_B.
(4) Re-alimentar as ativações de F_B, através de W^t (conexões de F_B para F_A).
(5) Calcular as ativações de F_A.
(6) Repetir os passos (3), (4), e (5) até que as ativações em F_A e em F_B não mais se alterem. A rede estará num estado de **ressonância.**

A atualização dos valores pode ser **síncrona** ou **assíncrona.**

As ativações em F_B são calculadas por:

$$b_j(t+1) = \begin{cases} 1 & se\ y_j > 0 \\ b_j(t) & se\ y_j = 0 \\ -1 & se\ y_j < 0 \end{cases}$$

Onde $b_j(t+1)$ é a ativação do processador j de F_B no instante (t+1), e y_j é o valor de pré-ativação do processador, definido por:

$$y_j = \sum_{i=1}^{n} a_i(t) . w_{ij}$$

Onde $a_i(t)$ é a ativação da unidade processadora i de F_A no instatnte t.

Equivalentemente, as ativações em F_A são dadas por:

$$a_i(t+1) = \begin{cases} 1 & se\ x_i>0 \\ a_i(t) & se\ x_i=0 \\ -1 & se\ x_i<0 \end{cases}$$

Onde:

$$x_i = \sum_{j=1}^{p} b_j(t) \cdot w_{ji}$$

A estabilidade da rede BAM é comprovada usando-se uma função de energia de Lyapunov que incorpora todos os parâmetros do sistema BAM ([49]):

$$L(A,B) = -\frac{1}{2}A.W.B^t - \frac{1}{2}A.W^t.B^t = -A.W.B^t$$

A é o vetor (vetor-linha) ativação de F_A, B é o de F_B, e W, a matriz de conexões entre as duas camadas, sendo W^t sua transposta.

Uma forma equivalente é:

$$L(A,B) = -\sum_{i=1}^{n}\sum_{j=1}^{p} a_i b_j w_{ij}$$

A variação da energia em relação a variação em A é:

$$\Delta L_A(A,B) = -(\Delta A).W.B^t = -\sum_{i=1}^{n} \Delta a_i . \sum_{j=1}^{p} b_j w_{ij}$$

$$\Delta A = (\Delta a_1, \Delta a_2, \ldots, \Delta a_n)$$

É possível mostrar que:

$$se\ \Delta A \neq 0 \Rightarrow \Delta L(A,B) < 0$$
$$e\ \Delta A = 0 \Rightarrow \Delta L(A,B) = 0$$

Ou seja, o sistema é estável (a energia não cresce, para qualquer variação de A).

Equivalentemente, o sistema é estável para variações em B.

A principal limitação do modelo BAM é sua baixa capacidade de armazenamento, aproximadamente m pares de padrões armazenados, m dado por (veja também [38]):

$$m = \frac{q}{4.\log_2 q}$$

$$onde\ q = \min(n,p)$$

Suas vantagens são a simplicidade, a estabilidade e a possibilidade de adicionar padrões rapidamente. Aplica-se especialmente quando se deseja relacionar um padrão ao correspondente ao seu vizinho mais próximo, dentre os padrões armazenados, mas com um número pequeno de pares.

III.5 - *Modelo de Hamming*

Uma **rede de Hamming**, descrita por Lippmann [40], é uma rede recursiva, que faz uma aproximação para o vizinho mais próximo, dentre os padrões armazenados.

A rede (ver figura III-6) possui 3 camadas: F_A, de entrada; F_B (a **camada competitiva**), onde ocorre uma competição entre as unidades de processamento para realizar a classificação, de forma que somente uma permaneça ativa após o processamento, aquela que melhor representa o padrão de entrada; e F_C, cujas unidades tem valor de ativação 0 ou 1, e função de ativação tal que é ativada a unidade cuja entrada (saída de uma unidade de F_B) é positiva.

Um modelo simplificado da camada competitiva é apresentado em [24], veja Figura III-7. Numa rede de aprendizado por competição, existe uma camada de entrada F_X

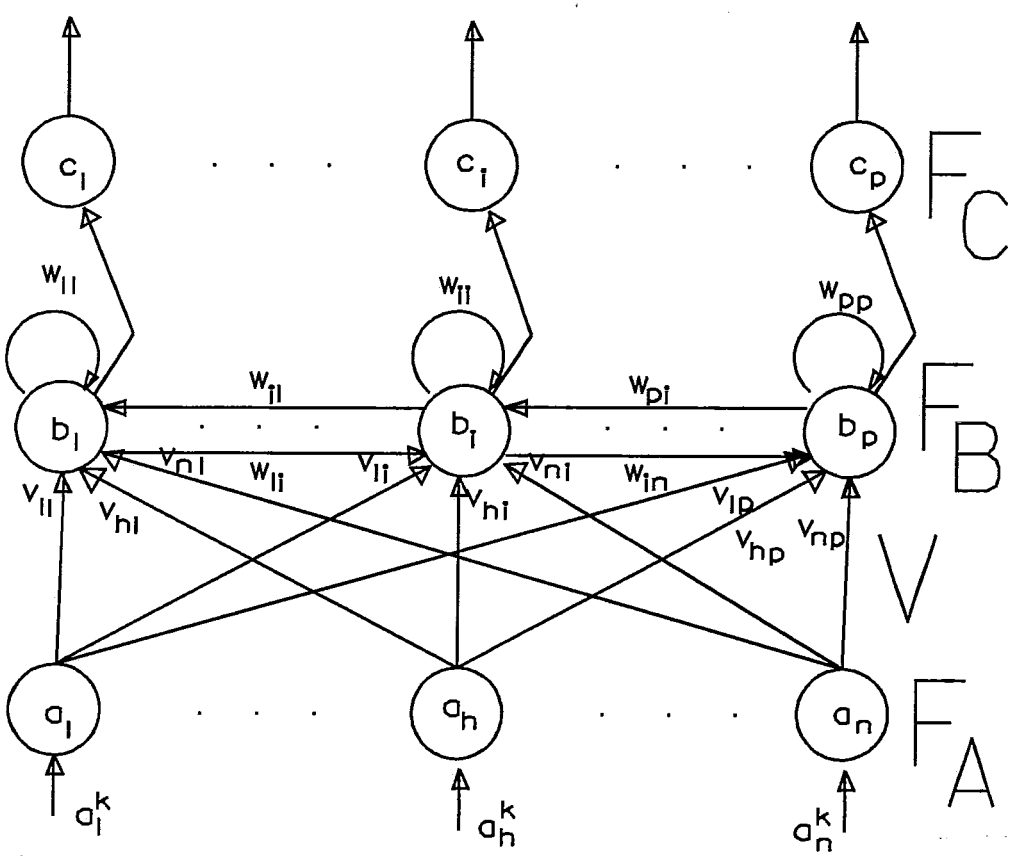


Figura III-6. Modelo de Hamming.

para distribuir os sinais (binários ou bipolares) de entrada, e uma camada de saída F_Y, onde se dá a competição. Cada nó y_j da camada de saída está conectado a todos os nós x_i da camada de entrada.

Somente uma unidade de F_Y pode estar ativa num dado instante, e é chamada de "vencedora". Normalmente é a unidade com maior valor de entrada net_j, para a entrada corrente X:

$$net_j = \sum_i u_{ij}.x_i = \boldsymbol{u}_j.\boldsymbol{a}$$

Onde u_{ij} é o peso da conexão entre a unidade i de F_X e a unidade j de F_Y. Dito de outra forma:

$$\boldsymbol{w}_{j^*}.\boldsymbol{a} \geq \boldsymbol{w}_j.\boldsymbol{a}, \ \forall j$$

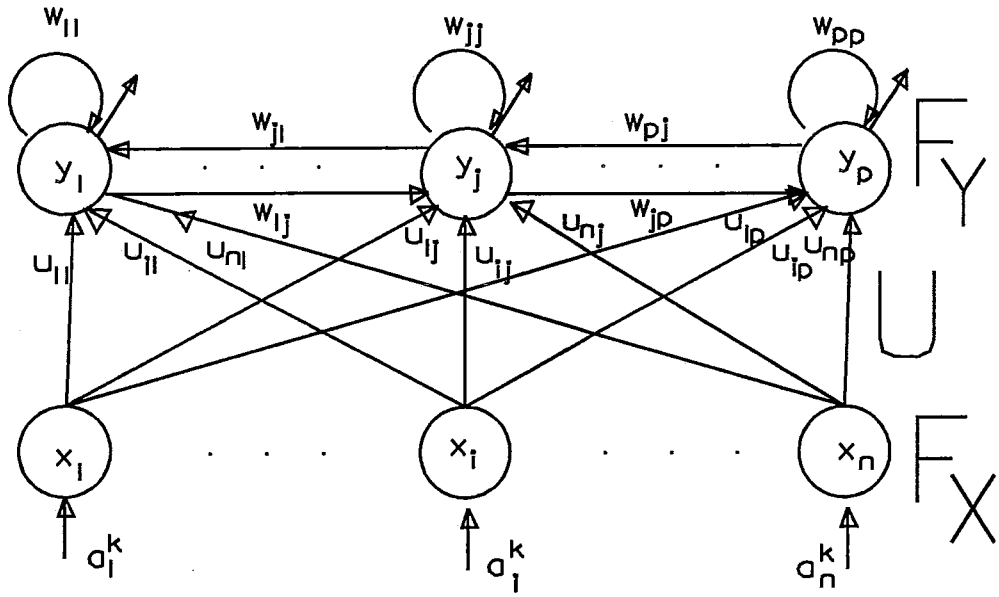


Figura III-7. Camada competitiva.

onde j^* identifica a unidade vencedora, e:

$$b_j = f(net_j) = 1 \qquad \text{(III-6)}$$

Se os pesos de cada unidade são normalizados de forma que $||w_j||=1$, para todo j, então a equação (III-6) equivale a:

$$\|\boldsymbol{w}_{j^*} - \boldsymbol{a}\| \leq \|\boldsymbol{w}_j - \boldsymbol{a}\|, \ \forall j$$

Esta equação diz que a unidade vencedora é aquela cujo vetor de pesos normalizado está mais próximo do vetor de entrada.

A característica de somente uma unidade (a vencedora) ser ativada pode ser implementada (numa rede) por conexões inibitórias laterais, além de uma conexão recorrente excitatória.

A rede de Hamming aprende um conjunto de m padrões numa única **epoch** - apresentação completa do conjunto de entrada. A cada padrão é atribuído um processador da camada F_B, que o representará. O número de unidades em F_B é igual ao número de padrões a serem armazenados, p = m, e o número de unidades em F_C é igual ao de F_B.

O aprendizado serve para atribuir valores às conexões e limiares de acordo com as seguintes regras:

$$v_{hi} = \frac{a_h^i}{2}$$

$$T_i = \frac{n}{2}$$

$$1 \leq h \leq n; \quad 1 \leq i \leq p$$

onde v_{hi} é o peso da conexão entre o processador h de F_A e o processador i de F_B, e T_i é valor do limiar do processador i de F_B.

Além disso:

$$w_{ik} = 1 \qquad se\ i=k$$

$$w_{ik} = -\lambda \qquad se\ i \neq k$$

$$1 \leq i,k \leq m$$

$$e \quad \lambda < \frac{1}{m}$$

onde w_{ik} é o peso da conexão entre a unidade i de F_B e a unidade k também de F_B. Nesta camada, os limiares de cada unidade têm valor nulo.

A recuperação (classificação) é feita pelo seguinte procedimento:

(1) Atribuir os valores iniciais com o padrão de entrada (desconhecido):

$$b_i(0) = f\left(\sum_{h=1}^{n} v_{hi} . a_h - T_i \right)$$

$$1 \leq i \leq m$$

$$f(x) = \begin{cases} 1 & se\ x \geq 0 \\ 0 & se\ x < 0 \end{cases}$$

(2) Atualização:

$$b_i(t+1) = f\left(b_i(t) - \lambda \sum_{k \neq i} b_k(t) \right)$$

$$1 \leq i,k \leq m$$

(3) Repetir passo (2) até que $b_i(t+1) = b_i(t)$ (convergência), e só haja um processador ativo em F_B.
(4) Os valores de ativação de F_B são passados para F_C, de forma que, em F_C, só haja um processador ativo com valor 1, e os demais tenham valor de ativação nulo. Este processador ativo representa o padrão armazenado mais próximo (em termos de distância de Hamming) do padrão apresentado na entrada.

Na rede de Hamming, é necessário uma unidade para cada classe de padrões. No entanto, por implementar um classificador ótimo de erro mínimo (quando os bits em erro são randômicos e independentes), tem desempenho igual ou melhor que a rede de Hopfield. Além disso, pode ter capacidade de armazenamento maior que uma rede de Hopfield de mesmo número de entradas, pois na rede de Hamming a capacidade é dada pelo número de unidades na camada F_B.

III.6 - *Modelo ART1*

O modelo **ART1 (Adaptive Resonance Theory - Teoria da Ressonância Adaptativa)**, descrito por Carpenter e Grossberg [14], [15], é baseado numa rede de duas camadas que usa aprendizado competitivo. É um classificador que armazena um número arbitrário de padrões binários $A_k=(a_1^k, \ldots, a_n^k)$, k=1,2,...,m. (Padrões analógicos são tratados numa extensão

do modelo chamada ART2). Os n processadores da camada F_A, veja Figura III-8, representam os componentes do padrão A_k, e os p processadores da camada F_B representam, cada um, uma classe, [49]. Um modelo semelhante é descrito em [48].

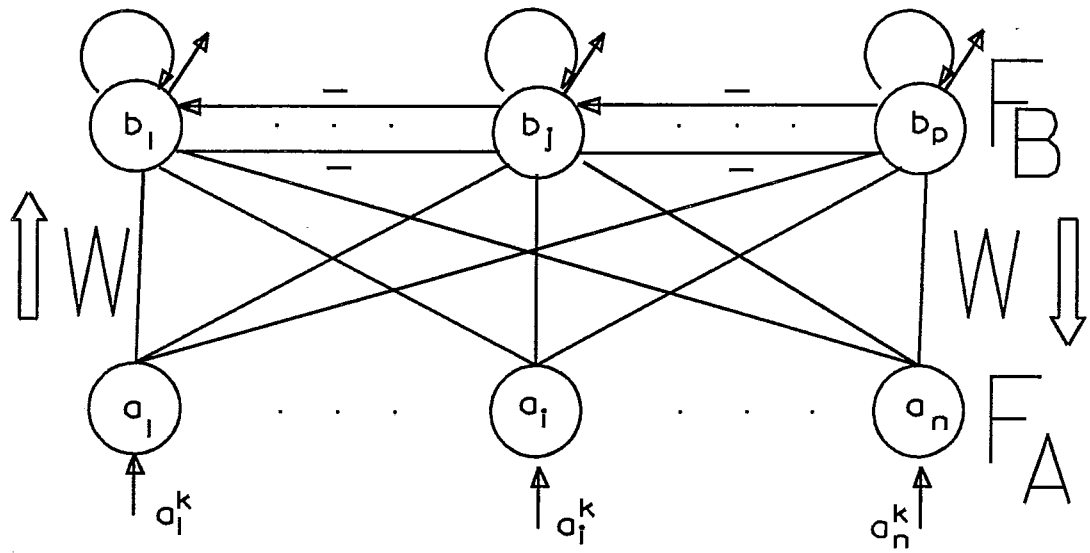


Figura III-8. Modelo ART1.

Cada processador de F_A conecta-se a cada um dos processadores de F_B, e vice-versa. Todos os processadores em F_B estão ligados entre si por conexões inibitórias e possuem, cada um, uma conexão recorrente, que implementam o aprendizado competitivo.

A rede ART1 também pode ser estruturada em dois subsistemas: o **subsistema de atenção (ou atencional)**, e o **subsistema de orientação (ou orientador)**, veja Figura III-9. O subsistema de atenção faz com que a camada F_A opere somente quando existe um padrão de entrada. O subsistema de orientação inibe um processador de F_B de ser ativado, após ser verificado que o processador não representa a classe do padrão de entrada, operação chamada **STM reset** (reposicionamento da memória de curto prazo - *Short Term Memory* - as ativações dos neurônios). Estes subsistemas podem ser inerentes ao algoritmo de processamento do sistema. Os

subsistemas são restrições que afetam o processamento da informação.

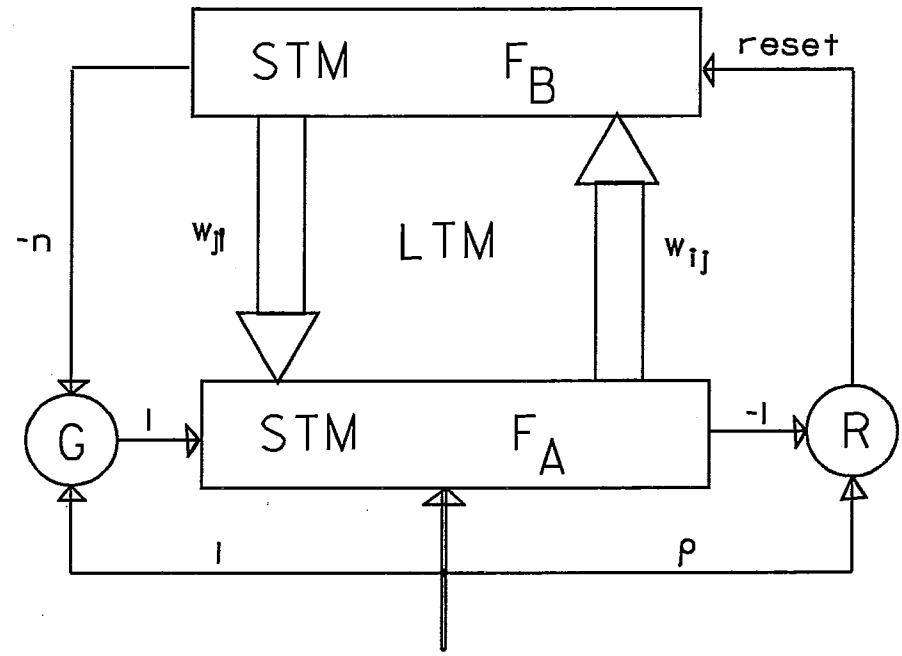


Figura III-9. Sistema ART1.

A camada F_A recebe os sinais externos em suas entradas a_i^k, e os sinais das conexões **top-down** w_{ji} (de F_B para F_A); e F_B recebe os sinais de F_A pelas conexões **bottom-up** w_{ij}.

Um procedimento de codificação, conforme Hertz e outros [24], é esboçado a seguir:

(0) Atribuições iniciais:

$\mathbf{w}_j = \mathbf{1}$

$\mathbf{w}_j$ é o vetor dos pesos das conexões que terminam no neurônio j de F_B; e **1** é o vetor cujos componentes todos têm valor 1.

É atribuído um valor ao **parâmetro de vigilância** ρ.

(1) Uma nova entrada **x** é apresentada na entrada de F_A e copiada em sua saída $\mathbf{a}=(a_1,\ldots,a_n)$.

(2) São habilitadas todas as unidades de F_B.

(3) Os sinais a_i, i=1,2,...,n, se transferem para F_B pelas conexões bottom-up w_{ij}.

Em F_B ocorre uma competição entre todos os neurônios habilitados (se não houver mais neurônios habilitados, o processamento termina).

O neurônio vencedor j^* é aquele para o qual:

$$\|\boldsymbol{w}_{j*}\| . \boldsymbol{a} = \max_{1 \le j \le p} (\|\boldsymbol{w}_j\| . \boldsymbol{a})$$

onde

$$\|\boldsymbol{w}_j\| := \frac{\boldsymbol{w}_j}{L + \sum_{i=1}^{n} w_{ij}}$$

onde w_j é o vetor dos pesos das conexões que terminam no neurônio j da camada competitiva, w_{j*} é o vetor correspondente ao neurônio vencedor j^*; $|| w_j ||$ é o vetor w_j normalizado; e L é um valor positivo (e pequeno).

As conexões w_{ij} e w_{ji} formam a **LTM** (*Long Term Memory* - Memória de Longo Prazo).

(4) Testar se a semelhança entre **a=x** e $\mathbf{w}_{j*}$ é suficiente (teste de hipótese):

$$v = \frac{\boldsymbol{w}_{j*} \cdot \boldsymbol{a}}{\sum_{i=1}^{n} a_i}$$

v, índice de coincidência, é a fração de bits em **a** que também está ligada em $\mathbf{w}_{j*}$. (Quanto mais bits, maiores as chances de **a** pertencer à classe j^*, conforme o valor do parâmetro ρ, definido *a priori*, no passo (0)).

(4.1) Se $\mathbf{v} \geq \rho$ então há **ressonância** (semelhança suficiente), e o padrão de entrada é classificado na classe j^*, vá para o passo (5);

(4.2) Se não, isto é: $\mathbf{v} < \rho$, o protótipo w_{j*} é rejeitado, a unidade j^* de F_B é desabilitada (não mais participando da competição para classificar este padrão A_k), volta para o passo (3).

(5) Adapta w_{j*} (operação AND - E - binária):

$$\boldsymbol{w}_{j*} := \boldsymbol{w}_{j*} \wedge \boldsymbol{a}$$

(6) Volta para o passo (1).

É feita uma busca entre os padrões armazenados (as classes conhecidas) até que se encontre um suficientemente próximo (segundo o parâmetro de vigilância).

Na rede ART1, não existe um modo de recuperação completamente distinto do aprendizado. Se o padrão na entrada estiver armazenado, a unidade de saída correspondente é ativada. Se não, ou a unidade correspondente à classe (já armazenada) é ativada (quando a entrada está suficientemente próxima do protótipo armazenado), ou uma nova unidade de saída é recrutada para representar o padrão na entrada. Se não existirem mais neurônios disponíveis em F_B, nenhuma resposta é obtida (saturação da rede).

Na Figura III-9, os pesos bottom-up w_{ij} são cópias normalizadas dos pesos top-down w_{ji} ($w_{ij}=||w_{ji}||$). Note-se que os valores de w_{ji}, a_i, b_j, G e R são binários (pertencem a {0,1}).

Se o sinal de **reset** R estiver ligado enquanto um neurônio de F_B estiver ativo (passo (4.2) - caso $r \leq \rho$), este neurônio vencedor é desabilitado e não participa das competições seguintes para esta classificação. (Todas as unidades em F_B podem ser reabilitadas por um sinal não mostrado, ao se iniciar a classificação de um novo padrão de entrada).

As unidades de F_A trabalham segundo a regra de ativação:

$$a_i = \begin{cases} x_i & \text{se } b_j = 0,\ j=1,2,\ldots,p \ (F_B \text{ inativa}) \\ x_i \wedge \sum_{j=1}^{p} w_{ji}.b_j & \text{de outra forma} \end{cases}$$

(III-7)

Onde ^ significa a operação lógica de AND (E). Isto é feito usando a unidade G (subsistema de atenção) que está ativa (G=1) quando existe um padrão na entrada da rede ($\mathbf{x \neq 0}$), e inativa (G=0), se não existe.

G pode ser um neurônio cuja ativação seja dada por:

$$G = f(\sum_{i=1}^{n} x_i - n.\sum_{j=1}^{p} b_j - 0.5)$$

onde

$$f(z) = \begin{cases} 1 & \text{se } z > 0 \\ 0 & \text{de outra forma} \end{cases}$$

Para as unidades de F_A, a ativação é:

$$a_i = f(x_i + \sum_{i=1}^{n} w_{ji}.b_j + G - 1.5) \qquad \text{(III-8)}$$

A equação (III-8) é equivalente a (III-7). (III-8) é conhecida como a **regra de 2/3:** duas de suas três entradas, (x_i; o somatório de $\{w_{ji}.b_j\}$; e G), têm que estar ativas para que a_i seja ativada.

Para R, vale:

$$R = f(\rho.\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} a_i)$$

A atualização dos pesos top-down ocorre com:

$$\frac{dw_{ji}}{dt} = \eta.b_j.(a_i - w_{ji})$$

De forma que o vetor protótipo $w_{j*}=(w_{1j*}, w_{2j*}, \ldots, w_{nj*})$ correspondente ao neurônio vencedor j^* de F_B torna-se igual à entrada a_i mascarada após ocorrer a ressonância. Os pesos bottom-up (w_{ij}) têm uma regra de atualização semelhante, que leva aos valores de w_{ji} normalizados.

A rede ART1 opera autonomamente. Pode lidar, sem perda de estabilidade, com um fluxo infinito de dados. Tem acesso rápido às categorias conhecidas e faz busca automática, quando a entrada não é conhecida. Também cria novas categorias quando é necessário e não fornece resposta quando sua capacidade está esgotada (não fornece uma resposta falsa).

Os cuidados necessários com este modelo devem-se a dificuldade de ajuste e a sua sensibilidade à ruído nos dados de entrada. Se bits aleatórios faltarem nos padrões de entrada, os padrões armazenados podem sofrer degradação pelo mascaramento da regra de adaptação (passo (5) do algoritmo). Outra desvantagem é a necessidade de muitas conexões para cada categoria (2.n), além de conexões de peso fixo, conforme a implementação.

Em suma, o interesse deste modelo está em sua capacidade de armazenar qualquer quantidade de padrões em classes de variados graus de complexidade. E tem importância também por ser uma solução para o dilema de **estabilidade** (reconhecer num padrão novo uma classe já codificada, sem ficar oscilando indefinidamente)-**plasticidade** (o sistema se adapta a um novo padrão criando uma nova classe, quando é o caso).

III.7 - *Modelo Counterpropagation*

Uma **rede counterpropagation** (contra-propagação) é um modelo de 3 camadas, hetero-associativo, que executa um mapeamento para o padrão vizinho-mais-próximo. É capaz de armazenar pares de padrões analógicos, (A_k, C_k), $k=1,2,\ldots,m$. Para a codificação (aprendizado) usa uma combinação de aprendizado de Grossberg (ou outros), para a camada de saída, e de Kohonen, para a camada escondida, (onde é encontrado um conjunto ótimo de vetores de referência - conexões que chegam a um determinado processador - para um dado conjunto de treinamento) (veja [49]). Um modelo similar, mas não tão genérico, é proposto em [27], chamada de *feature map* (mapa de características).

Na Figura III-10, as n unidades processadoras da camada F_A correspondem aos elementos de A_k, e as q unidades de F_C correspondem aos de C_k.

O procedimento de codificação é esboçado a seguir:

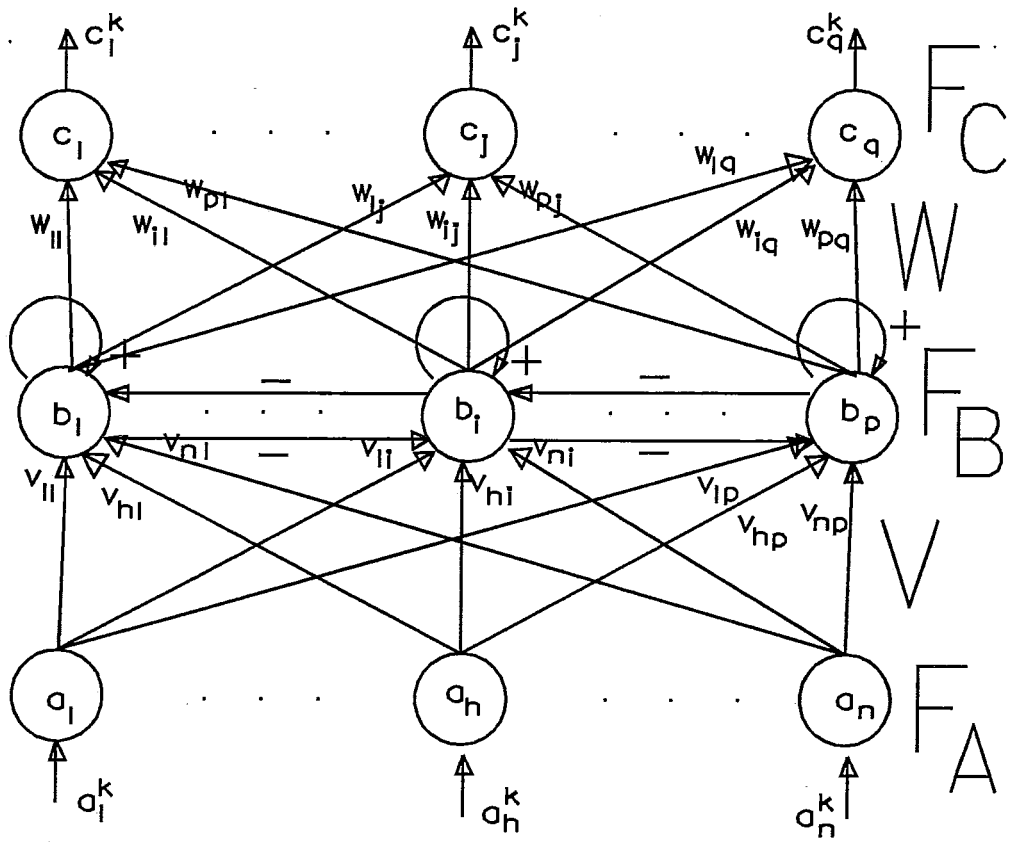


Figura III-10. Modelo Counterpropagation.

(1) Normalizar os padrões A_k para o comprimento unitário:

$$a_h^k = \frac{a_h^k}{\| A_k \|}$$

$h=1,2,\ldots,n;\ k=1,2,\ldots,m$

(2) Atribuir valores randômicos, no intervalo [0,1], aos pesos v_{hi} das conexões entre F_A e F_B.

(3) As conexões entre a camada F_A e o processador i da camada F_B, que formam o vetor de pesos $V_i = (v_{1i}, \ldots, v_{hi}, \ldots, v_{ni})$, são normalizadas:

$$v_{hi} = \frac{v_{hi}}{\| V_i \|}$$

$h=1,2,\ldots,n;\ i=1,2,\ldots,p$

(4) Para cada padrão A_k, k=1,2,...,m:

(4.a) Achar V_g mais próximo de A_k:

$$\| A_k - V_g \| = \min_{i=1}^{p} \| A_k - V_i \|$$

(4.b) Aproximar V_g de A_k:

$$\Delta v_{hg} = \beta(t) . [a_h^k - v_{hg}]$$

$$h=1,2,\ldots,n$$

v_{hg} é o peso da conexão entre a unidade h de F_A e a unidade g de F_B, e $\beta(t)$ é a taxa de aprendizado no instante t=1,2,... (valores discretos), definida por:

$$\beta(t) = t^{-1}$$

ou

$$\beta(t) = 0.2\left[1 - \frac{t}{10000}\right]$$

(4.c) Renormalizar V_g:

$$v_{hg} = \frac{v_{hg}}{\| V_g \|}$$

(4.d) Codificar o padrão C_k:

$$\Delta w_{gj} = -\xi w_{gj} + \gamma b_g c_j^k$$

$$j=1,2,\ldots,q$$

Onde Δw_{gj} é a variação no peso da conexão entre o processador g de F_B e o processador j de F_C; ξ é uma constante positiva chamada taxa de decaimento, γ é a constante de aprendizado, e b_g é a ativação do processador de F_B correspondente a V_g:

$$b_g = \sum_{h=1}^{n} v_{hg} . a_h^k$$

(5) Repetir o passo (4) para t=1,2,...,z, z em [500,10000].

A recuperação é feita passando os valores de ativação da camada de entrada F_A, após lido um padrão, pela matriz V de conexão com F_B:

$$b_i = \sum_{h=1}^{n} v_{hi} . a_h$$

$$i=1,2,...,p$$

Em seguida, o valor máximo de ativação em F_B, b_g, é passado para F_C, obtendo-se o padrão C na saída:

$$c_j = w_{gj} . b_g$$

$$j=1,2,...,q$$

A principal limitação da rede counterpropagation é a necessidade de haver uma unidade na camada do meio (F_B) para cada par armazenado. As principais vantagens são a capacidade de mapeamento para o padrão vizinho mais próximo, a partir do padrão de entrada, o aprendizado de mapeamentos não-lineares, e a velocidade de aprendizado (10 a 100 vezes mais rápido que para uma backpropagation, embora não se aplique em todos os casos que esta última).

Estes são os fundamentos dos modelos usados neste trabalho, apresentados de uma forma resumida. No próximo capítulo, veremos a aplicação destes paradigmas ao problema de decodificação.

Capítulo IV

Códigos e Redes Neurais

Veremos uma aplicação de Redes Neurais a Códigos de Controle de Erros, especificamente, no processo de decodificação, motivação central deste trabalho. Os modelos vistos no Capítulo III foram usados, com resultados variados. Sem ter pretensão alguma de esgotar o assunto, mesmo porque existem outras abordagens para o problema, o trabalho serviu para o estudo e comparação dos paradigmas apresentados, para esta determinada tarefa.

IV.1 - *Por que Redes Neurais para Decodificação ?*

Simulações feitas demonstram a capacidade de redes neurais funcionarem como decodificadores tanto para códigos blocados, como para códigos convolucionais (estes não tratados nesta dissertação). Caid e Means [13], interessados em contra-medidas eletrônicas - quando o canal não é simétrico e binário (veja Capítulo II) e não existe um ruído "branco gaussiano" - usaram redes *feedforward* de aprendizado supervisionado (backpropagation) e o código de Hamming (7,4), além de um código convolucional para as simulações.

Hussain e outros, [28] e [29], aplicam backpropagation e otimização randômica ao problema de decodificação. Afirmam que o problema de código de controle de erros é do tipo autoassociativo e que se aplica bem ao ambiente backpropagation (porém, observamos que não, quando se tratam de entradas binárias, ou arredondadas).

O problema de decodificação é apresentado como um problema de classificação de sinal ou de padrões, com vantagens na relação sinal/ruído e na capacidade de correção, comparando-se com os sistemas existentes.

Além de o problema de decodificação poder ser visto como um problema de classificação, também este pode ser visto como um problema de decodificação, tal como proposto em [19]. Neste artigo, descreve-se um modo de classificação baseado em transformar um vetor de entrada numa palavra-código possivelmente com ruído, a qual é decodificada, encontrando-

se a palavra-código fundamental, a classe a qual pertence o vetor de entrada original. Foram usados códigos da matriz de Hadamard e códigos de sequência de máximo comprimento, ambos os tipos de códigos são tais que quaisquer duas palavras-códigos tem distância de Hamming constante entre si, e tão grande quanto possível.

A classificação por vizinho mais próximo, descrita em [36], é semelhante a decodificação em códigos não-perfeitos. O vetor-código é o fundamental de uma classe, vetores com até t diferenças pertencem a esta classe (distância de Hamming até t), com mais de t diferenças não pertencem à nenhuma classe; ou, podemos interpretar, podem pertencer à classe (mas com distância de Hamming maior que t do vetor fundamental), ou a duas ou mais classes (quando não seriam aceitos, pedindo-se retransmissão, ou seriam decodificados com ajuda de algum processo de votação).

Vimos que o problema de decodificação guarda semelhança com o reconhecimento de padrões. Cherkassky e Vassilas [17] relatam o uso de backpropagation na recuperação de padrões (nomes) de um banco de dados, funcionando como memória associativa. Para cada nome armazenado, existe um neurônio na camada de saída. Algumas conclusões genéricas são alcançadas (veja seção IV.3). Apesar de serem encontradas taxas de recuperação correta de até 100 %, redes backpropagtion não são consideradas de uso prático para este tipo de aplicação devido ao tempo proibitivamente longo necessário para se adicionar ou apagar um registro. Praticamente, é necessário retreinar a rede utilizando todo o conjunto de dados. Nessa aplicação, as comparações indicaram a superioridade dos modelos baseados em memórias associativas distribuidas sobre o modelo redes backpropagation, inclusive por causa do custo de se treinar uma rede backpropagation. Huang e Lippmann [27] também apontam outros modelos em detrimento da backpropagation, quando se requer classificação com regiões de decisão complexas.

Zeng, Hush e Ahmed [56] usam rede de Hamming para decodificar códigos blocados lineares e não-lineares, apontando vantagens sobre os computadores Von Neuman.

Em [42], usa-se rede de Hamming para decodificar o código BCH (7,4) num experimento acústico submarino.

Em resumo, existem diversos trabalhos em decodificação - ou similarmente em reconhecimento de padrões - usando diversos modelos de redes neurais, com casos de resultados bastante animadores. A seguir, apresentamos exemplos de aplicações realizadas para este trabalho.

IV.2 - *Rede de Hamming e o Código de Hamming*

Nas referências [42] e [56] é citado o uso de uma rede de Hamming para decodificar o código BCH (7,4) (código de Hamming). Conforme [56], a rede calcula as distâncias de Hamming entre o vetor recebido e os vetores (palavras-códigos) armazenados, em tempo polinomial (o tempo de processamento é uma função polinomial do número de entradas) graças ao paralelismo. Num computador Von Neuman, este cálculo consome um tempo exponencial (o tempo é uma função exponencial do número de entradas).

Baseados na referência [42], experimentamos a rede de Hamming na decodificação do código blocado BCH (7,4) . A topologia da rede (7-16-16) é mostrada na Figura IV-1. O polinômio gerador do código é:

$$g(X) = 1 + X + X^3$$

A rede foi treinada com a representação bipolar completa do código, isto é, os vetores fundamentais (palavras-código) sem ruído. A representação binária do código é dada na Figura IV-2. É suficiente uma apresentação completa (que serve para a atribuição dos valores dos pesos das conexões).

Para testar a rede, foram apresentados todos os 128 (= 2^7) vetores do espaço $\{-1,1\}^7$, representando os vetores fundamentais e os possíveis vetores com ruído. A eficácia é de 100%, ou seja, todos os vetores "recebidos" foram decodificados corretamente.

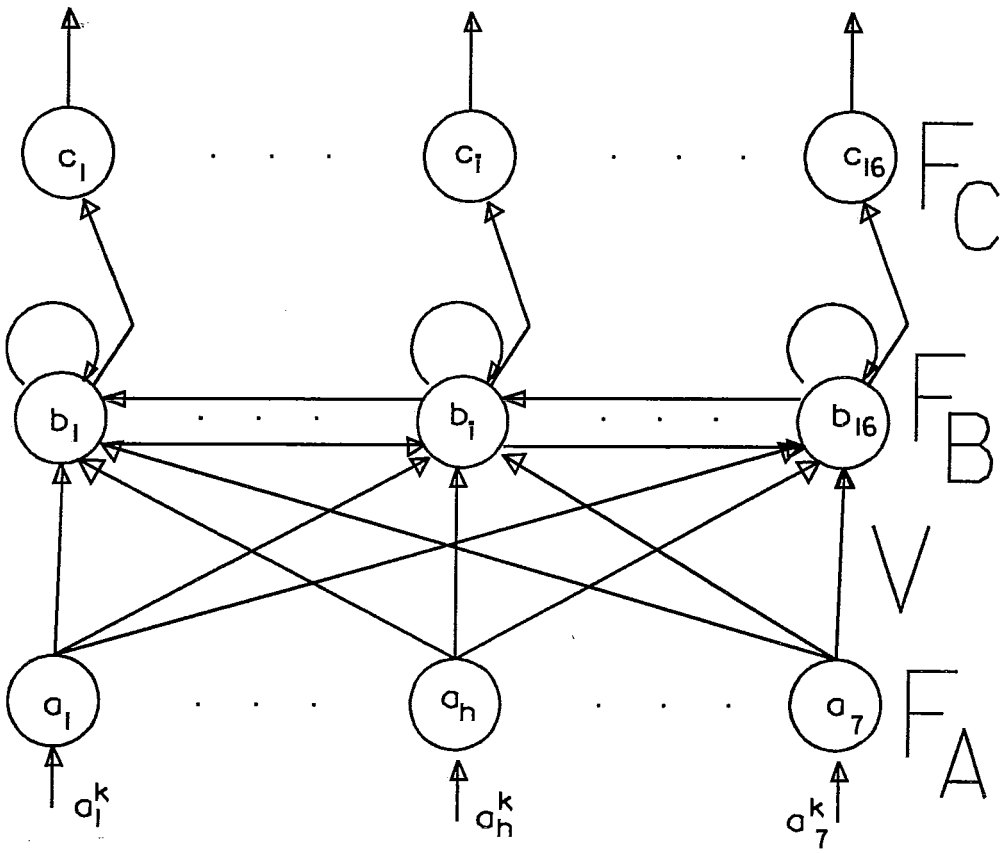


Figura IV-1. Rede de Hamming (7-16-16), para o código BCH (7,4)..

0	0	0	0	0	0	0
1	1	0	1	0	0	0
0	1	1	0	1	0	0
1	0	1	1	1	0	0
1	1	1	0	0	1	0
0	0	1	1	0	1	0
1	0	0	0	1	1	0
0	1	0	1	1	1	0
1	0	1	0	0	0	1
0	1	1	1	0	0	1
1	1	0	0	1	0	1
0	0	0	1	1	0	1
0	1	0	0	0	1	1
1	0	0	1	0	1	1
0	0	1	0	1	1	1
1	1	1	1	1	1	1

Figura IV-2. O código de Hamming (7,4).

Lembramos que, sendo o código de Hamming um código perfeito que corrige um erro, se ocorrerem dois erros ao se transmitir um vetor **v**, o vetor recebido **r** será

interpretado como um outro vetor-código **v'**, contendo um único erro, do qual **r** estará mais próximo em termos de distância de Hamming.

A rede de Hamming foi usada também na decodificação do código de Golay (23,12), que possui $2^{12} = 2048$ palavras-código, capacidade de corrigir até 3 erros, e também é um código perfeito. Devido às restrições de equipamento, não foram feitos testes exaustivos, mas pelas observações feitas, e pelas definições tanto do código como da rede, pode-se afirmar a aplicabilidade da rede de Hamming sobre este código, com eficácia de 100 %, tal como para o código de Hamming.

IV.3 - *Redes Backpropagation e Código de Hamming*

Caid e Means [13], e Hussain, [28] e [29], usam redes backpropagation com o código (7,4).

No entanto, Brady e outros, [9], mostram alguns exemplos evidenciando que cuidados devem ser tomados. Tratam-se de problemas de separação de duas famílias de vetores para os quais pode-se usar uma combinação linear para realizar a separação. Porém, a solução de mínimo custo (soma dos quadrados dos erros, no algoritmo backpropagation, que usa descida por gradiente) não separa as dadas famílias da maneira desejada.

Confirma-se que soluções de mínimos erros quadrados não minimizam o número de erros de classificação (decodificação no nosso caso), tanto para casos de entradas analógicas como para entradas boolenas (isto é, em $\{0,1\}^n$). Assim, minimizar o erro quadrático não equivale a minimizar o número de erros de classificação. E os resultados apresentados no artigo mostram que tais erros de classificação não resultam de alguma regra particular nem estão restritos a casos com superfícies complicadas de energia. A referência [52] também trata deste tema.

Cherkassky e Vassilas [17], numa aplicação de reconhecimento de padrões, observam que:

(1) a escolha dos valores dos parâmetros para um bom desempenho depende do problema, é necessário um ajuste empírico dos parâmetros para cada problema

(2) o número de nós escondidos determina a capacidade da rede

(3) representações esparsas da entrada permitem uma melhor recuperação, mas não aumentam a capacidade da rede

(4) a escolha dos parâmetros de aprendizado é crítica para o desempenho como um todo da memória associativa backpropagation.

Estas considerações servem para avaliar a aplicabilidade do modelo backpropagation em decodificação. Existem pesquisas apontando num ou noutro sentido. Ao longo do desenvolvimento deste trabalho mesmo, foi vista a dificuldade de se proceder a uma análise rigorosa, pela quantidade de instâncias de redes backpropagation que surgem, com cada variação de cada parâmetro: taxa de aprendizado, *momentum*, número de neurônios na camada do meio, "temperatura" para o aprendizado (pertubação imposta nos valores dos dados na entrada dos neurônios). Além disto, estes parâmetros e também as funções de ativação dos neurônios (logística, tangente hiperbólica, degrau), podem variar de camada para camada de uma mesma rede. E, ainda, pode haver variação nos valores a medida que a rede opere no modo de aprendizado (aumentando o número de iterações).

Assim, várias simplificações foram feitas. Por exemplo, não variamos os parâmetros conforme aumentava o número de iterações numa mesma rede, nem usamos valores diferentes para diferentes camadas (em [39], é proposto algo similar à temperatura diferente de zero na camada do meio). Apresentaremos os resultados considerados mais significativos. Foi usado o código de Hamming - BCH (7,4).

Inicialmente foram usados 7 neurônios na camada de entrada, e 7 na camada de saída, variando-se a quantidade de unidades na camada do meio de 1 a 48 (não continuamente). Para um código (n,k) em geral, o número de vetores do espaço B^n, $B = \{0,1\}$, seria muito maior que a quantidade de vetores código. Por isso eram apresentados somente os vetores fundamentais, na entrada e na saída, durante o aprendizado, com representação bipolar (0 representado por -1), que evita

que o primeiro termo da regra de ajuste dos pesos (veja seção II-2) seja anulado. A rede, naturalmente, aprendeu a função identidade, simplesmente reproduzindo na saída os valores da entrada, quando em operação. A injeção de ruído analógico nos dados de aprendizado não melhorou a decodificação. Também foi tentado o aprendizado suplementar, conforme [34], onde a rede se adapta somente para os vetores que causam um erro maior que um limite determinado. A medida que a rede progride, este limite diminui para que seja feito o ajuste "fino".

Em seguida, baseados em [53], que sugere dar informações redundantes para a rede aprender o mapeamento, acrescentamos um bit na saída da rede, representando a paridade do vetor na entrada. Também somente eram apresentados os vetores fundamentais durante o modo de aprendizado. Com algumas exceções, a rede calculava a paridade, mas não corrigia o vetor recebido, sendo decodificado.

Partindo de [17], foram usados 16 neurônios na camada de saída, obtendo-se um ganho considerável na eficácia e resultados mais significativos. Foi obtida eficácia de até 95 %. Resultados melhores foram obtidos com a apresentação dos vetores contendo ruído discreto (bits em erro: invertidos de 0 para 1 ou de 1 para 0), conforme sugere Hussain, [28] e [29], possível num código pequeno como o BCH (7,4) usado.

A topologia em geral foi de 7 neurônios na camada de entrada, um número variável na camada do meio, e 16 na saída. Os valores iniciais dos pesos eram randômicos e pequenos (devem estar entre -0.1 e +0.1; foram usados valores entre -0.07 e +0.07=0.5/N, N=7). Resultados de alguns dos testes realizados estão na Figura IV-3.

"Topologia" refere-se ao número de neurônios nas camadas de entrada, do meio (e segunda do meio, na rede (3)), e de saída, respectivamente. η é o valor da taxa de aprendizado utilizado na regra de aprendizado. λ é o valor do momentum utilizado na regra de aprendizado. "fç." é a função de ativação utilizada pelos neurônios da camada de saída:

	topologia	η	λ	fç.	treino	Q	eficácia	
(1)	(7-16-16)	0.2	0.0	tanh	fund.	250	106	83 %
						5000	105	82 %
(2)	(7-16-16)	0.1	0.5	tanh	fund.	1100	107	84 %
						5000	104	81 %
(3)	(7-11-11-16)	0.2	0.	tanh	fund.	5000	68	53 %
(4)	(7-16-16)	0.2	0.5	tanh	compl.	100000	110	86 %
(5)	(7-16-16)	0.2	0.5	sign	compl.	6000	128	100 %
(6)	(7-16-16)	0.1	0.0	sign	compl.	7500	128	100 %
(7)	(7-15-16)	0.2	0.0	sign	compl.	15000	127	99 %
(8)	(7-24-16)	0.2	0.0	tanh	fund.	250	121	95 %
						500	121	95 %

Figura IV-3. Resultados com Backpropagation.

tanh é a tangente hiperbólica, e

$$sign(x) = \begin{cases} 1 & se\ x > 0 \\ 0 & se\ x = 0 \\ -1 & se\ x < 0 \end{cases}$$

"Treino" refere-se ao conjunto de vetores usados para o treinamento (aprendizado) da rede: *fund.* significa que apenas os vetores fundamentais, sem ruído, foram apresentados, *compl.* significa que o conjunto completo de vetores do espaço B^7 foi apresentado. Q é o número de iterações que a rede executou antes de ser medida a sua eficácia. "Eficácia" é a capacidade da rede de decodificação correta, medida sobre o conjunto completo de vetores: o primeiro valor (x) é a quantidade de vetores decodificados corretamente, e o segundo é o percentual equivalente (x / 128 * 100 %).

Observe-se que um maior número de iterações (apresentação de cada vetor) não significa maior eficácia;

que o uso de duas camadas escondidas não se mostrou especialmente promissor; e que uma saída discretizada melhorou o aprendizado, levando a eficácia a até 100 %. O erro RMS ao final das iterações em cada rede (usado no algoritmo) era da ordem de 5 % ou menor, mas um erro RMS menor nem sempre significava maior eficácia, como ocorreu no caso (1): cerca de 5% após 250 iterações, e 1.5% após 5000 iteraçõs; ou no caso (2): 3% após 1100 iterações, e 1.5% após 5000 iterações.

A apresentação ou não dos vetores com ruído foi fundamental para a decodificação 100% correta (como se obtem no algoritmo tradicional). No entanto, em códigos de maior comprimento, isto pode representar uma restrição intransponível ao aprendizado da rede, devido ao número de vetores que teriam que ser apresentados e ao tempo de aprendizado.

IV.4 - *Redes de Hopfield e BAM*

As redes de Hopfield e BAM têm capacidade de armazenamento abaixo das necessidades da aplicação. No entanto, existe literatura referindo-se a redes de Hopfield em decodificação, ([31], [32], [54], [55]). Buscam, entre outros pontos, garantir que as palavras-código sejam os únicos atratores (pontos de mínimo) da rede.

Aqui, simplesmente tentamos armazenar algumas palavras-código em cada rede e verificar sua capacidade de decodificação. Foi usada a representação bipolar.

Do código de Hamming, uma rede BAM (7-4), veja Figura IV-4, foi treinada em apenas 2 vetores simétricos. Estes foram perfeitamente recuperados, com um bit em erro (de 0 para 1, ou de 1 para 0) em qualquer posição. Era apresentado a palavra-código (com ruído, possívelmente), e a rede convergia para o vetor mensagem de 4 bits correspondente.

Ao se treinar a rede em mais duas palavras-códigos, houve degradação, no sentido de que ela não era mais capaz de decodificar corretamente nenhuma das quatro palavras-códigos apresentadas para aprendizado.

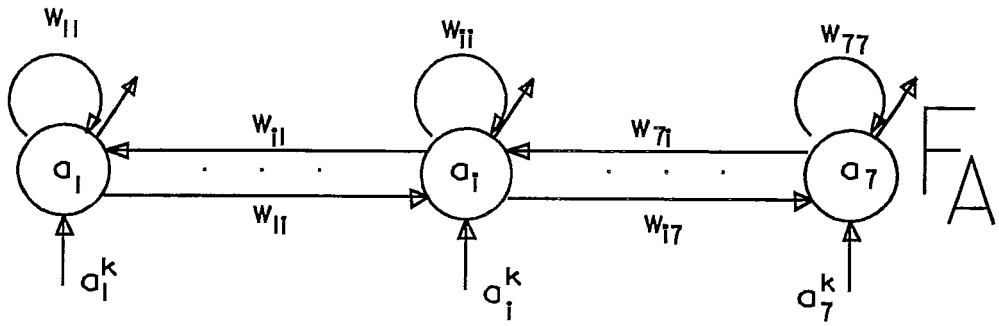


Figura IV-5. Rede de Hopfield para o código de Hamming (7,4).

recebido. A grande vantagem é o reduzido número de neurônios necessários para a decodificação (igual a *n* para um código *(n,k)*).

IV.5 - *Rede Counterpropagtion*

O modelo counterpropagation foi capaz de decodificar corretamente qualquer vetor de 7 bits, usando o código BCH (7,4) (código de Hamming). A topologia usada é mostrada na Figura IV-6. Foi usada a representação bipolar.

O aprendizado ocorreu com supervisão na camada de saída somente. Após 6000 iterações, 375 varridas sobre o conjunto dos vetores fundamentais (a rede não foi treinada com vetores contendo ruído), obteve-se uma eficácia de 100% na decodificação dos 128 vetores possíveis.

Uma rede counterpropagation, veja Figura IV-8, foi usada também na decodificação do código BCH (15,5), não-perfeito, apresentado na tabela da Figura IV-9. O polinômio gerador do código (15,5) é:

$$g(X) = 1 + X + X^2 + X^4 + X^5 + X^8 + X^{10}$$

Este código tem capacidade de correção de até 3 erros numa palavra-código. Foi usado um conjunto de teste

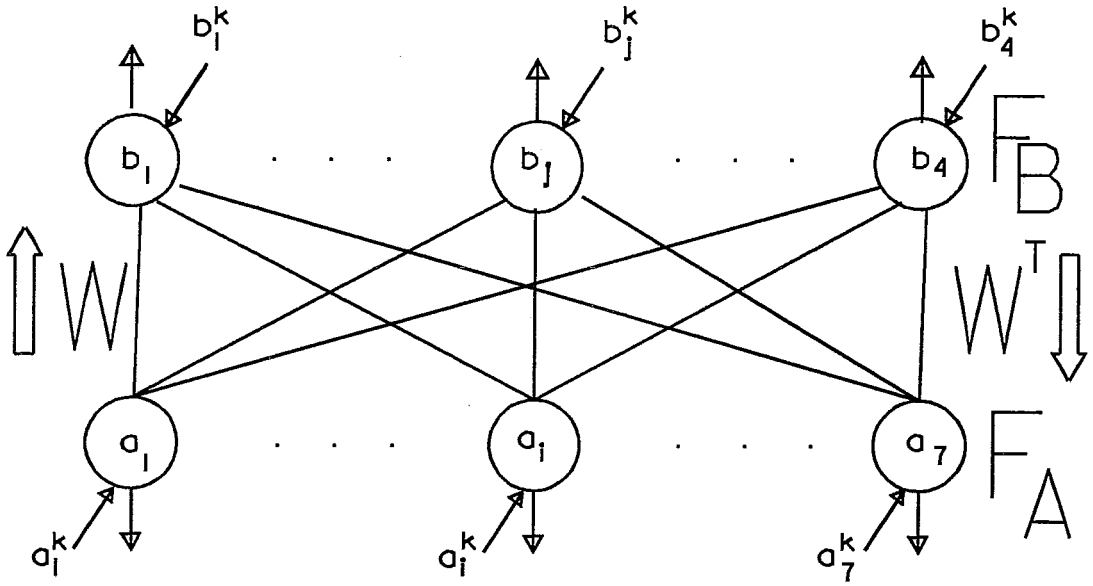


Figura IV-4. Rede BAM (7-4), para o código de Hamming (7,4).

Para verificação da rede de Hopfield, foi utilizado esquema semelhante. Duas palavras simétricas do código de Hamming foram armazenadas. A rede, Figura IV.5, era capaz de recuperá-las a partir de qualquer padrão com um bit em erro. Observou-se que a rede também corrigia dois erros em um vetor, mas com 3 erros, o comportamento não era completamente previsível, ora convergindo para o vetor correto, ora o seu simétrico. Usou-se uma probabilidade de ativação de neurônio de 50%.

Ao ser apresentado um terceiro padrão (palavra-código), a rede aprendeu-o (e ao seu simétrico), mas já não recuperava perfeitamente um padrão qualquer com ruído.

Deve-se ressaltar que Bruck [10] (e [11]) mostra que achar o mínimo **global** da função de energia da rede de Hopfield equivale à decodificação por máxima semelhança. A dificuldade é, portanto, garantir que o caminho sobre a superfície de energia leve ao mínimo correspondente ao vetor

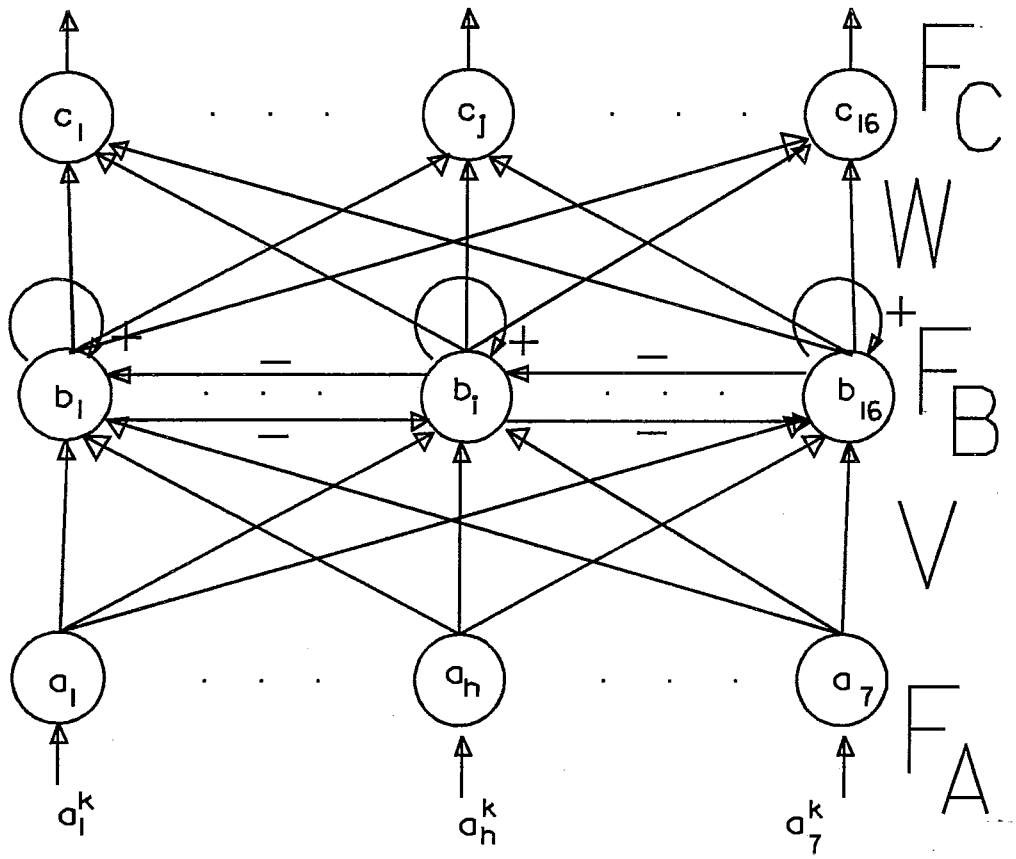


Figura IV-6. Rede Counterpropagtion (7-16-16), para o código de Hamming.

quantidade de erros	decodificações corretas	eficácia da rede
0	96	100 %
1	96	100 %
2	96	100 %
3	96	100 %
4	56	58 %

Figura IV-7. Resultados para Counterpropagation decodificando BCH (15,5).

contendo os 32 (= 2^5) vetores fundamentais e mais 2 vetores com bits em erro para cada vetor fundamental (totalizando 96 vetores). Após 6000 iterações, 187 apresentações dos vetores fundamentais (únicos usados no treinamento da rede), foram obtidas as medidas de eficácia da rede (para este conjunto de

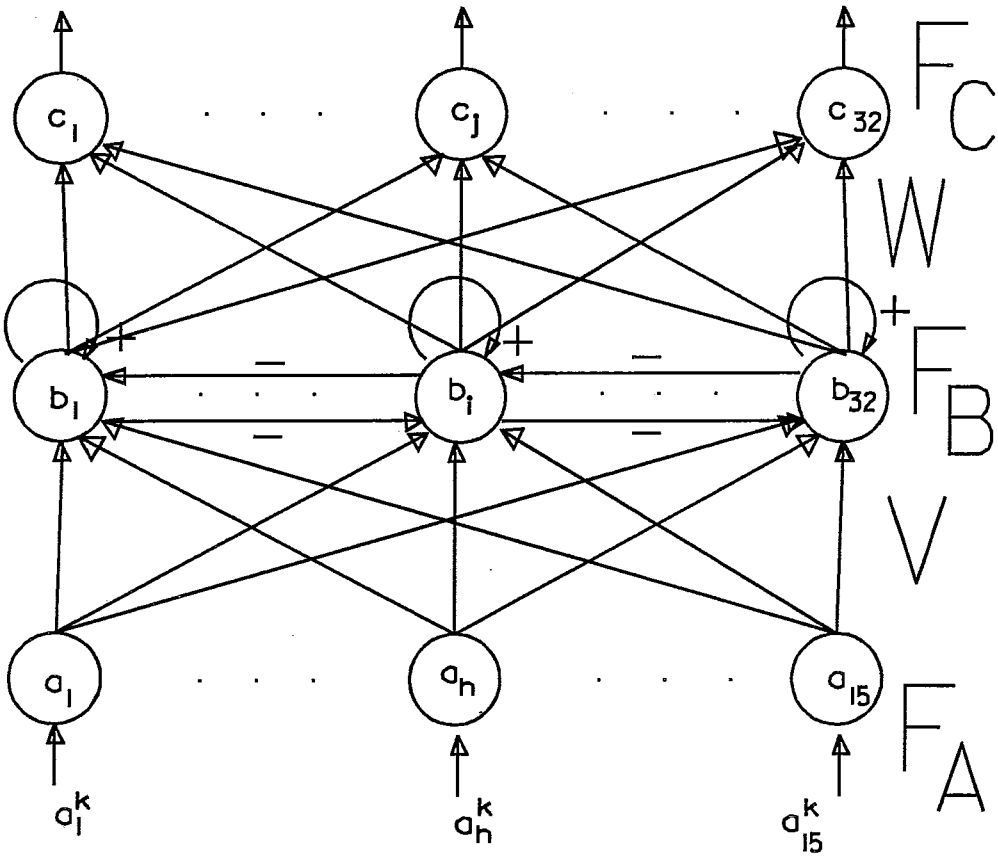


Figura IV-8. Rede counterpropagation (15-32-32) para o código BCH (15,5).

teste) apresentadas na Figura IV-7.

Observe-se que das 56 decodificações corretas no caso de haver 4 erros, 32 se referem aos vetores fundamentais, e 24 (38% de 64) aos vetores com ruído. Para efeito da aplicação, no entanto, qualquer eficácia abaixo de 100 % compromete a decodificação, pois o receptor sabe que foi aproximado o vetor mais próximo do vetor recebido, mas não tem garantia que este foi o transmitido. Seria necessário um pedido de retransmissão ou, assumindo-se algum risco, uma análise de mais alto nível para verificação de consistência de toda a mensagem, aceitando a decodificação realizada (uma espécie de "votação").

IV.6 - *Rede ART1 e Código de Hamming*

O modelo ART1 é um modelo que despertou bastante interesse, apesar de não ser de aplicação tão imediata. Normalmente, uma rede ART1 é usada para, além de reconhecer uma categoria num vetor recebido, criar novas categorias em

0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	1	0	1	0	0	0	0
1	1	1	0	1	1	0	0	1	0	1	0	0	0
0	0	1	1	0	1	0	1	1	1	1	0	0	0
1	0	1	0	1	1	1	1	0	0	0	1	0	0
0	1	1	1	0	1	1	0	0	1	0	1	0	0
0	1	0	0	0	0	1	1	1	0	1	1	0	0
1	0	0	1	1	0	1	0	1	1	1	1	0	0
1	1	0	1	0	1	1	1	1	0	0	0	1	0
0	0	0	0	1	1	1	0	1	1	0	0	1	0
0	0	1	1	1	0	1	1	0	0	1	0	1	0
1	1	1	0	0	0	1	0	0	1	1	0	1	0
0	1	1	1	1	0	0	0	1	0	0	1	1	0
1	0	1	0	0	0	0	1	1	1	0	1	1	0
1	0	0	1	0	1	0	0	0	0	1	1	1	0
0	1	0	0	1	1	0	1	0	1	1	1	1	0
1	0	1	1	0	0	1	0	1	0	0	0	0	1
0	1	1	0	1	0	1	1	1	1	0	0	0	1
0	1	0	1	1	1	1	0	0	0	1	0	0	1
1	0	0	0	0	1	1	1	0	1	1	0	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0	1
1	1	0	0	0	1	0	0	1	1	0	1	0	1
1	1	1	1	0	0	0	1	0	0	1	1	0	1
0	0	1	0	1	0	0	0	0	1	1	1	0	1
0	1	1	0	0	1	0	1	0	0	0	0	1	1
1	0	1	1	1	1	0	0	0	1	0	0	1	1
1	0	0	0	1	0	0	1	1	0	1	0	1	1
0	1	0	1	0	0	0	0	1	1	1	0	1	1
1	1	0	0	1	0	1	0	0	0	0	1	1	1
0	0	0	1	0	0	1	1	0	1	0	1	1	1
0	0	1	0	0	1	1	0	1	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figura IV-9. Código BCH (15,5).

tempo de execução, possivelmente não havendo um modo de recuperação estritamente falando. Este não é o nosso caso, em que a rede deve aprender um número finito de padrões e, depois, apenas reconhecer os vetores que lhe forem apresentados, sem criar novas categorias.

Além disso, a idéia da ART1 é reconhecer características representadas por bits de valor '1', enquanto que na decodificação um bit '1' pode significar também um erro (inversão de bit '0').

Por outro lado, o parâmetro de vigilância do modelo permite algum ajuste da quantidade de erros a serem corrigidos pela rede, além do que a rede não fornece resposta alguma (diminuindo o risco de se obter uma decodificação errada). Talvez seja possível adaptar melhor o modelo ART1 à aplicação, tornando mais clara a sua utilidade neste caso.

Para o código BCH (7,4), utilizou-se uma rede com 7 neurônios (bits) na entrada e 16 na saída, conforme a Figura IV-10.

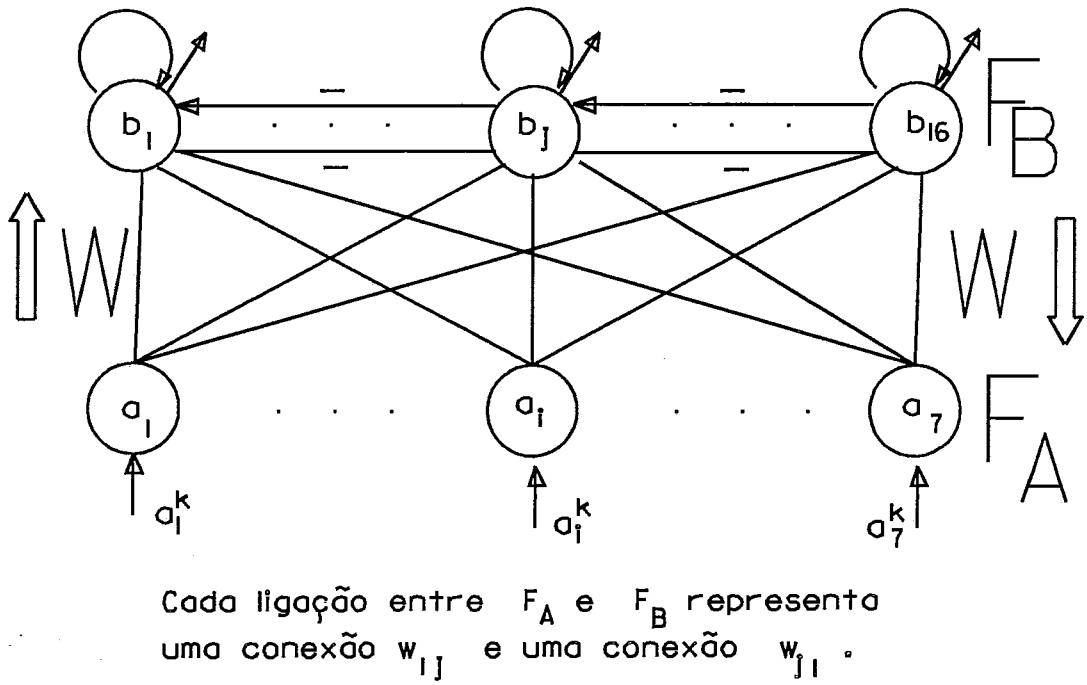


Figura IV-10. Rede ART1 (7-16) para código BCH (7,4).

Deve-se ter cuidado no ajuste do parâmetro de vigilância ρ, que é diferente no modo de aprendizado do modo de decodificação. No código BCH (7,4) existem vetores com peso 0, 3, 4 e 7. O número máximo de erros é 1 (pela definição). Assim, durante a decodificação, pode ocorrer um índice de coincidência v ($v = |\mathbf{u} \wedge \mathbf{v}| / |\mathbf{u}|$, o valor que é comparado com o parâmetro de vigilância, veja seção III.6), máximo de 0.857 = 6/7. Então devemos ter $\rho < 0.857$.

A regras de atualizações dos pesos usadas foram:

$$w_{ji} = a_i$$

para os pesos top-down, e:

$$w_{ij} = \frac{L}{L-1+\sum_{i=1}^{n} a_i}$$

onde $L = 2$

para os pesos bottom-up, que são simplificações das regras apresentadas anteriormente.

Para o aprendizado, basta apresentar os vetores fundamentais de forma que sempre haja um *reset* (criação de nova categoria) para cada novo vetor apresentado. Para que os vetores possam ser apresentados em qualquer ordem, deve-se usar $\rho = 1$, e percorrer os vetores fundamentais duas vezes. Desta forma obtivemos eficácia de 100% na decodificação do conjunto completo de vetores de B^7.

IV.7 - *Rede de Hamming e Rede ART1*

Os modelos de Hamming e ART1 foram ainda aplicados a outros códigos.

Uma rede de Hamming (15-32-32) foi utilizada para a decodificação do código BCH (15,5), que tem capacidade de corrigir até t=3 erros. Para avaliação da rede, foram usados conjuntos contendo os vetores fundamentais e mais 2 vetores com 1, 2, 3 ou 4 bits em erro, respectivamente. Veja resultados na Figura IV-11 e a rede na Figura IV-12.

quantidade de erros	decodificações corretas	eficácia da rede
0	96	100 %
1	96	100 %
2	96	100 %
3	96	100 %
4	53	55 %

Figura IV-11. Resultados para rede Hamming e código BCH (15,5).

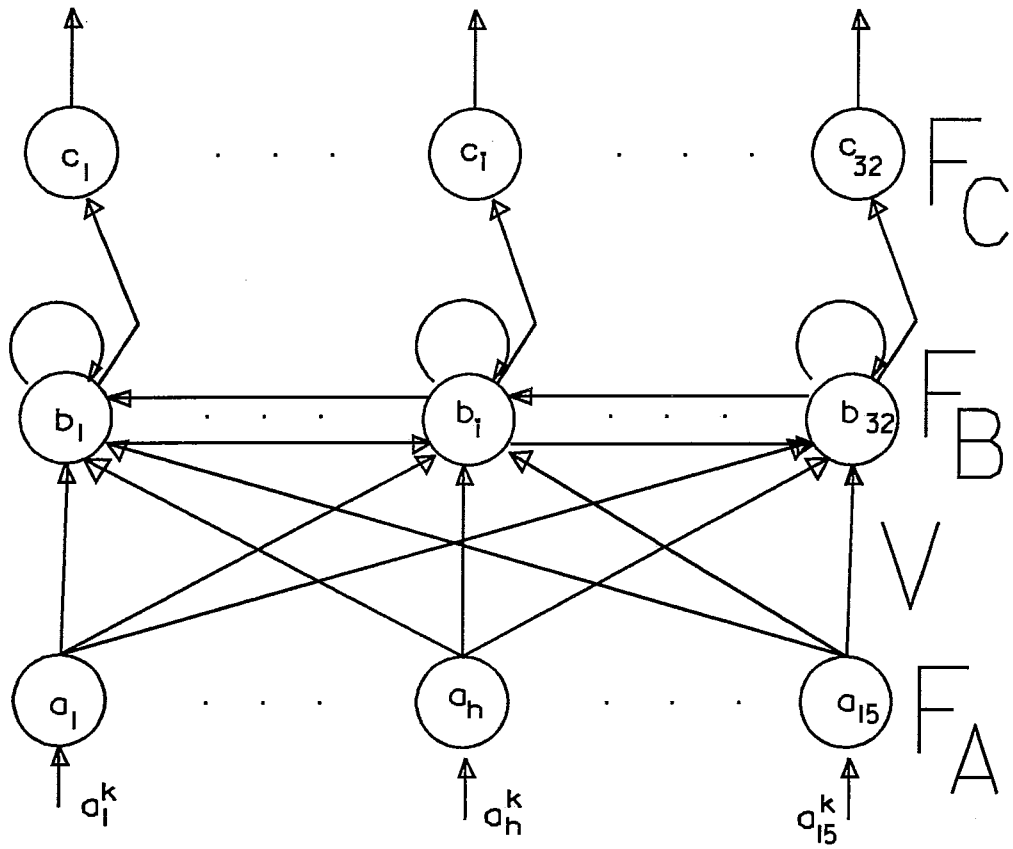


Figura IV-12. Rede de Hamming (15-32-32) para código BCH (15,5).

Observe-se que para t=4 erros, nos 53 corretamente decodificados, estão incluídos os 32 vetores fundamentais, significando 21 vetores corrigidos (32 % dos 64 com erro). A rede de Hamming teve uma eficácia pouco abaixo da Counterpropagation, para este valor de *t*.

Também foi usada uma rede de Hamming (23-18-18) para o código de peso constante com parâmetros (n=23, d=10, w=7), onde *n* é o comprimento do bloco, *d* é a distância de Hamming mínima, e *w* é o peso do código (constante por definição), [22].

Este código é blocado e não-linear, e é encontrado com o uso do algoritmo de *simullated annealing* (resfriamento simulado), veja [1]. É apresentado na Figura IV-13. A Figura IV-14 apresenta os resultados encontrados, e a Figura IV-15, a rede utilizada. Para esta avaliação foram usados 7 conjuntos de teste contendo os vetores fundamentais e mais 3

vetores contendo t erros (t=1,2,3,4,5,6,7, respectivamente a cada conjunto).

0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	1
0	0	0	0	0	1	0	1	0	1	1	0	1	0	0	1	0	0	0	1	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1
0	0	0	1	0	0	0	0	1	0	1	0	0	0	1	1	1	0	0	0	1	0	0
0	0	0	1	0	0	0	1	0	0	0	1	0	1	0	1	0	0	0	0	0	1	1
0	0	0	1	1	0	0	0	0	0	0	0	1	1	1	0	0	1	0	1	0	0	0
0	0	1	0	1	0	1	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	1
0	0	1	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0
0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1
0	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0	1	0
0	1	0	1	0	1	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	1	1	0	0	0
1	0	0	0	0	0	0	0	1	1	0	1	1	1	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	0	0	1	0	0	0	1	0
1	0	0	0	1	0	1	0	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0
1	0	1	1	1	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	1	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	1

Figura IV-13. O código de Hemachandra (23,10,7).

quantidade de erros	decodificações corretas	eficácia da rede
0	72	100 %
1	72	100 %
2	72	100 %
3	72	100 %
4	72	100 %
5	68	95 %
6	54	75 %
7	46	64 %

Figura IV-14. Resultados para a rede de Hamming e o código de Hemachandra, (23,10,7).

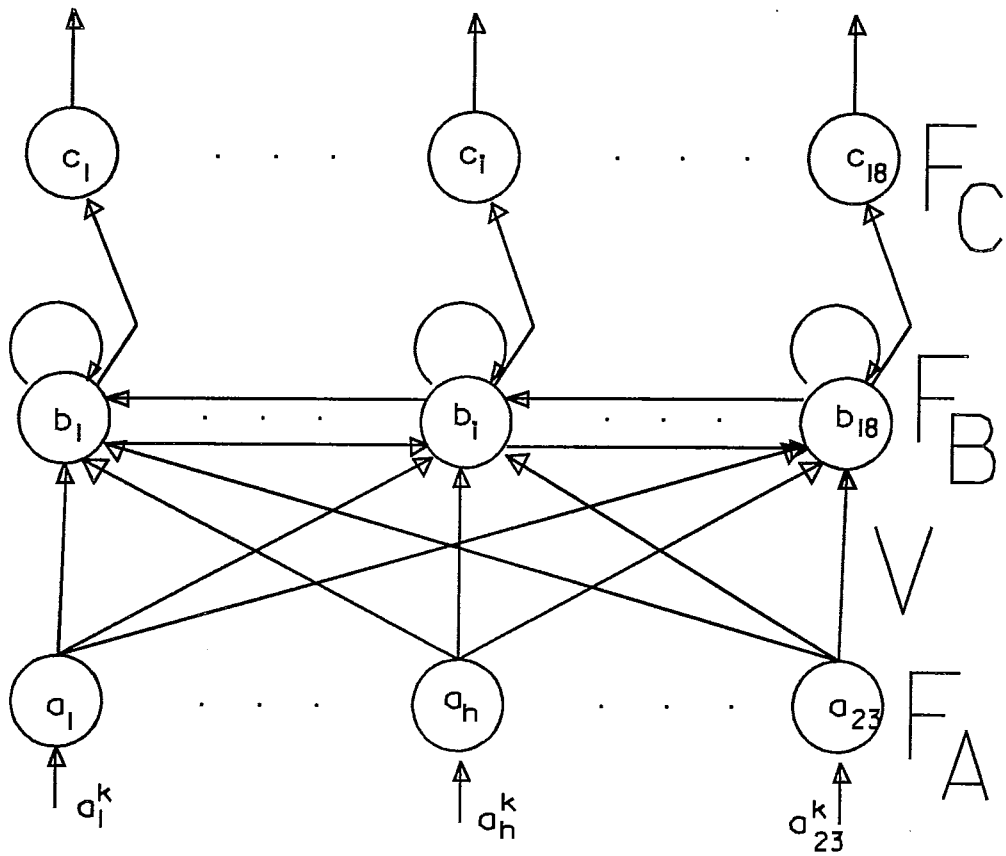


Figura IV-15. Rede Hamming (23-18-18) para o código de Hemachandra.

Os resultados (deste conjunto de teste) mostram que o código (23,10,7) pode ser explorado além de sua capacidade original (para t=5 erros, digamos), havendo uma análise de mais alto nível para aceitar ou não a decodificação realizada. A decodificação errada significa que o vetor recebido continha erros suficientes (e em posições tais) para colocá-lo mais próximo de outro vetor, não que a rede tenha "errado" no cálculo das distâncias de Hamming.

A rede ART1 também foi usada na decodificação de Hemachandra, veja Figura IV-16. Foi usado o valor de 0.29 para o parâmetro de vigilância durante a decodificação (e 1.0, durante o aprendizado, tal como no código BCH (7,4)). A eficácia foi de 100 % para o arquivo de teste. Outros resultados estão na Figura IV-17. O valor de ρ foi variado para se verificar o quanto a rede corrigia além do previsto na definição do código, um pouco superior à rede de Hamming. Eficácia de 25% significa que a rede apenas reconheceu os vetores fundamentais.

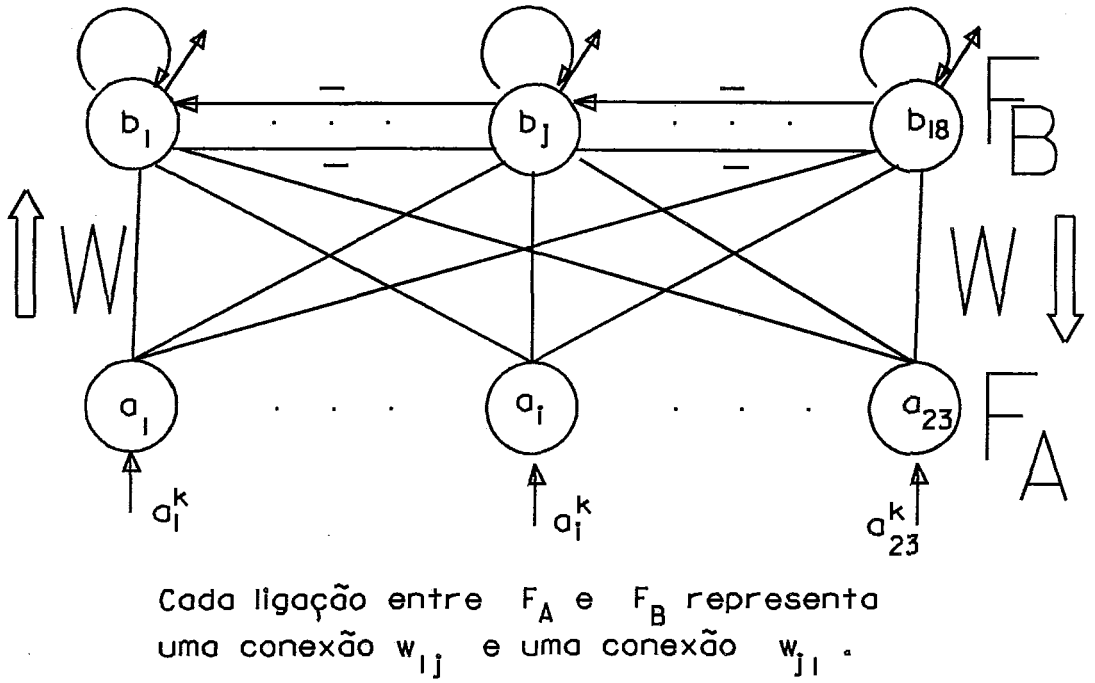


Figura IV-16. Rede ART1 para o código de Hemachandra (23,10,7).

Com a ART1, é possível ter algum controle sobre a capacidade de decodificação, e sua aplicação para códigos de peso constante pareceu mais simples (no que se refere ao cálculo do parâmetro de vigilância) do que para códigos BCH.

	t =	3		4		5		6		7	
ρ											
0.63	72	100%	72	100%	28	38%	18	25%	--		
0.58	72	100%	72	100%	69	95%	18	25%	--		
0.53	72	100%	72	100%	69	95%	58	80%	18	25%	
0.29	72	100%	72	100%	69	95%	58	80%	45	62%	

Figura IV-17. Resultados para rede ART1 e código de Hemachandra (23,10,7).

Comparando-se os modelos de Hammnig e ART1, aquele é bem mais simples em sua definição e compreensão. No entanto, ART1, ou uma adaptação, pode vir a ter aplicações

interessantes em códigos de controle de erros. Por sua parametrização, este modelo possui uma flexibilidade que abre vários caminhos para estudo.

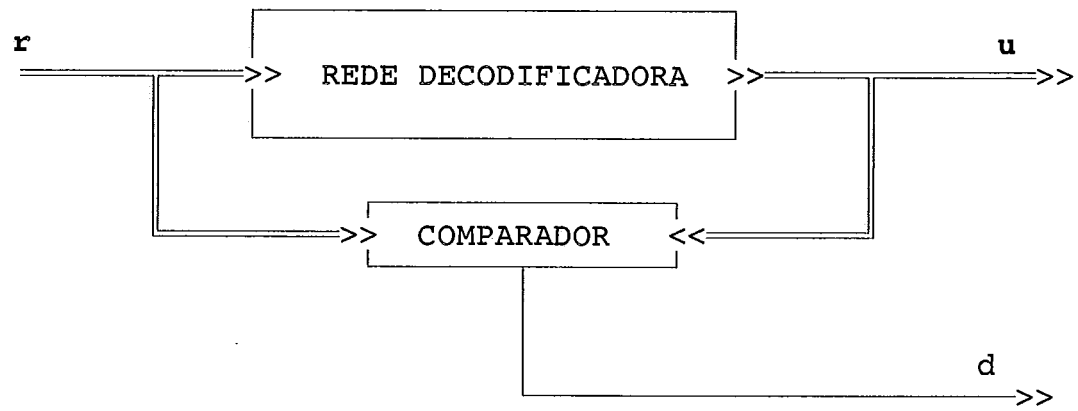


Figura IV-18. Combinação Rede e Comparador.

Para qualquer modelo usado, é conveniente que seja incluído no equipamento de decodificação um comparador que meça a distância de Hamming d entre o vetor resultado **u** e o vetor recebido **r**, conforme a Figura IV-18. Caso a distância seja maior que a capacidade de correção do código, **u** é aceito ou não dependendo de uma análise de consistência com o contexto das mensagens, e do valor de algum parâmetro de risco aceitável. No caso da ART1, este parâmetro de risco pode ser o próprio parâmetro de vigilância, para o qual um valor baixo significaria que se pode correr um risco maior (aceitar mais bits em erro). Para a rede de Hamming, este parâmetro teria que ser introduzido no sistema.

De todos os modelos, o que se aplica de maneira mais simples e é capaz de resolver o problema de decodificação é o de Hamming. Usa o paralelismo que redes neurais permitem, e sempre fornece uma resposta equivalente ao vizinho mais próximo. Resta ser verificado quais dificuldades surgirão numa implementação em hardware.

Capítulo V

Conclusão

Um resumo da dissertação é apresentado, seguido das conclusões. Pudemos ver também que muito se tem escrito sobre Redes Neurais e Decodificação. Outros temas correlatos aparecem como de interesse, mas que, por restrições diversas, não puderam ser estudados. Alguns são citados como sugestão de temas de pesquisa.

V.1 - *Resumo*

Procuramos fazer uma comparação entre 6 modelos de redes neurais na solução do problema de decodificação binária. Este problema aparece, entre outras situações, da necessidade de comunicação por meio acústico entre plataformas de petróleo na superfície do mar e unidades submarinas. O uso do meio acústico (o próprio mar, neste caso) substituiria o uso de cabos de sinal, os quais envolvem vários problemas, incluindo o alto custo.

Foi feita uma revisão teórica de Códigos de Controle de Erros, especialmente códigos BCH, incluindo um método algorítmico de decodificação.

Também foi apresentada uma revisão dos conceitos básicos de redes neurais e dos modelos de rede utilizados neste trabalho: backpropagation, de Hopfield, BAM, de Hamming, ART1 e counterpropagation.

Em seguida, apresentou-se uma argumentação da possibilidade de se aplicar redes neurais ao problema de decodificação, que pode ser entendido como um caso de reconhecimento de padrões. Simulações da aplicação propriamente ditas foram apresentadas, e medidas de eficácia para diversas combinações rede-código foram mostradas. Adiante apresentamos as conclusões.

V.2 - *Conclusões*

As principais conclusões são:

(a) Quanto à aplicação propriamente dita, a rede de Hamming, para os códigos vistos, se mostrou a solução mais simples e

melhor, aliás, como poderia ser esperado, pela sua própria definição. Este modelo usa o paralelismo que redes neurais permitem e fornece o vizinho mais próximo. Para um código BCH de tamanho maior (ou mesmo para o Golay), há que se pensar nas dificuldades de uma implementação em hardware.

(b) Redes de Hopfield têm um interesse particular por suas características e por sua história. Pesquisadores tem desenvolvido estudos para usá-las em decodificação. Quando (e se) tiverem sucesso, teremos um decodificador bastante eficiente, com apenas n (tamanho do bloco) neurônios. Trata-se de uma área bastante complexa, no entanto.

(c) Redes ART1 nos chamaram bastante atenção por sua flexibilidade. O parâmetro de vigilância permitiu algumas variações, dada uma rede, de uma maneira relativamente controlada. A Teoria da Ressonância Adaptativa têm uma importância especial no estudo de Redes Neurais.

(d) A rede counterpropagation com treinamento não-supervisionado na camada de Kohonen (camada do meio) resolveu o problema também com eficácia de 100%. Uma desvantagem frente a de Hamming é a necessidade de várias varridas do conjunto de aprendizado. Isto pode ser evitado com técnicas melhores de aprendizado. Porém, a counterpropagation terá maior número de conexões que a rede de Hamming.

(e) A já clássica rede backpropagation tem sido citada na literatura técnica, mas o seu modo de aprendizado dificulta sua utilização. Em todo caso, é um modelo que tem que ser considerado. O uso de um emulador de redes neurais ("pacote") se, por um lado, mascarou algumas dificuldades, por outro, permitiu testar um grande número de redes.

(f) A rede BAM tem grande especialmente por sua relativa simplicidade didática. No entanto, sua baixa capacidade de armazenamento torna necessário ou o uso de várias redes ou uma adaptação que a torne mais apropriada para a aplicação.

Uma dificuldade prática foi a quantidade de instâncias de rede que surgem, especialmente backpropagation. Uma recomendação para um estudo como este é que se delimite bem o escopo a ser investigado e que se organizem as diversas

instâncias, até com uma padronização de nomes de arquivos e programas.

V.3 - *Sugestões de Literatura e para Pesquisa*

A seguir citamos alguns artigos cujos temas nos pareceram merecer estudos pelos interessados na matéria:

Jeffries, referência [31], apresenta o problema de decodificação como um problema de escolha entre um número finito de opções. Compara a convergência de uma rede a um processo matemático de reconhecimento, e o atrator a uma memória matemática. Mostra indicações de que as trajetórias do modelo sempre irão para a memória mais próxima (vizinho mais próximo), realizando a decodificação, mas isto carece ainda de uma prova matemática.

Jeffries [32] também compara a decodificação a uma rede neural implementando memória de acesso pelo conteúdo. Descreve um modelo de rede cuja entrada é um vetor analógico, um valor estimado inicial, a partir do qual a rede converge para a solução, realizando uma decodificação por soft-decision. As restrições que surgem referem-se à dificuldade de armazenar palavras (padrões) arbitrárias, à memória limitada, e a não haver garantia de que os padrões armazenados sejam os únicos atratores do sistema. Para sanar estas dificuldades, é usado o modelo descrito na decodificação do código de Hamming (7,4), com ganhos de desempenho sobre um decodificador convencional.

Yuan e outros, [54] e [55], dizem ser possível obter decodificadores com hardware mais simples, e de mais rápida resposta, usando redes neurais. É descrito decodificadores de máxima-verosimilhança para códigos de peso constante e para o código de Golay (24,12). Para os primeiros, usam o modelo de Hopfield. Para o código de Golay, decomposto em 8 sub-códigos - sub-conjuntos das palavras-códigos, de mesma estrutura - usam uma rede para armazenar um dos sub-códigos, a partir do qual decodificam qualquer vetor recebido.

E para o código QR (47,24), cujo algoritmo precisa de muitas operações do tipo endereçamento por conteúdo, é

usada uma rede similar ao *perceptron* de duas camadas, com neurônios semelhantes ao ADALINE proposto por Widrow.

O problema de decodificação poder ser visto como um problema de classificação, e vice-versa. Chiueh e Goodman, [19], descrevem um modo de classificação baseado em transformar um vetor de entrada numa palavra-código possivelmente com ruído, a qual é decodificada, encontrando-se a palavra-código fundamental, a classe a qual pertence o vetor de entrada original.

Bruck e outros, [10], [11] e [12], estudam redes neurais na resolução de problemas *NP-hard*, buscando entender a relação entre rapidez de operação e tamanho da rede. Uma importante propriedade de uma rede neural, operando no modo serial, é que o seu espaço de estados não contem ciclos. A rede converge para um estado estável, de um número finito de estados estáveis, a partir do estado inicial (não oscila indefinidamente). Mostra que existe uma equivalência entre achar o máximo (ou mínimo) da função de energia e achar o corte mínimo de um grafo não-direcionado, e entre achar o máximo global da função de energia e decodificar por máxima verosimilhança. E também mostra que uma rede neural pode ser projetada para realizar uma busca local por um corte mínimo num grafo direcionado. Apresenta um teorema generalizado de convergência de redes neurais.

O problema do Corte Mínimo estudado é NP-hard. Teoricamente, então, pode-se transformar qualquer problema NP-hard no problema de corte mínimo e usar a rede neural correspondente para realizar uma busca por um mínimo local. É um problema em aberto.

Machado, Tenorio e Silva, [42], propõem, além de utilizar uma rede de Hamming, que sofreria o aprendizado em laboratório e, no campo, faria a decodificação das palavras recebidas (transmitidas segundo o código de Hamming (7,4)), uma rede backpropagation. Esta sofreria o aprendizado no campo, isto é, em operação normal, tendo como sinal professor a saída da rede de Hamming, e, como sinal de entrada, a palavra recebida.

Para a rede de Hamming, a apresentação do conjunto completo dos vetores-código uma única vez seria suficiente. Para a rede backpropagation, seria necessária a apresentação de vetores com ruído, o que torna o aprendizado oneroso (o número de vetores possíveis cresce muito). Porém, com o aprendizado no campo, o objetivo da rede backpropagation é fazer um *tracking* (rastreamento) das variações dos parâmetros não-estacionários do canal, realizando uma equalização que aumentaria o desempenho do conjunto. Esta e outras combinações de paradigmas podem trazer resultados interessantes ("o todo melhor que a soma das partes").

Uma área de pesquisa importante a ser atacada, além da combinação de dois paradigmas, é a seleção dos atratores com uma distância de Hamming previamente especificada, para a qual as redes neurais podem ser uma ferramenta útil. Gamal e outros tratam deste assunto na referência [22], usando *simulated annealing* no projeto de códigos blocados de peso constante (não-lineares). Projetar um código dedicado é uma opção para o problema de comunicação, e em particular para este caso acústico. Outra possibilidade, não investigada, seria o uso de algoritmos genéticos.

Não foram descritos neste trabalho os códigos convolucionais. Como o nome indica, baseiam-se numa convolução dos bits da mensagem. Várias referências, como Caid e Means, [13], e Provence, [44], citam-no como uma classe importante de códigos para os quais pode-se obter bons resultados usando redes neurais.

Portanto, decodificação por redes neurais além de envolver o estudo das próprias redes e da Teoria dos Códigos, beneficia-se da Teoria da Complexidade, da Teoria dos Grafos, e da Classificação de Padrões, englobando várias áreas para pesquisa.

Referências Bibliográficas

[1] AARTS, EMILE H. L., e PETER J.M VAN LAARHOVEN, "Simulated Annealing: A Pedestrian Review of the Theory and some Applications", *in Pattern Recognition Theory and Applications*, P.A. Devijver e J.Kittler, eds., Springer-Verlag, 1987

[2] AAZHANG, BEHNAAM, e TROY HENSON, "Enhanced Neural Net Learning Algorithms for Classification Problems", *Proceedings of the SPIE*, vol. 1294, *Applications of Artificial Neural Networks*, 1990, pp. 161-170

[3] ABU-MOSTAFA, YASER S., e JEANNINE-MARIE ST. JACQUES, "Information Capacity of the Hopfield Model", *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp.461, 1985

[4] ALSTON, MICHAEL D., e PAUL M. CHAU, "A Neural Network Architecture for the Decoding of Long Constraint Length Convolutional Codes", *Proceedings of International Conference on Neural Networks*, vol. I, 1990, p. 121

[5] ASSIS, FRANCISCO MARCOS DE, "Revisitando Reed-Muller", *Revista Militar de Ciência e Tecnologia*, vol.IX, no.1, pp.47-53, 1992

[6] BACKES, JOHN L., BRADLEY M. BELL, e JACK B. MILLER, "Implementation of Error Detection and Correction Codes for Acoustic Data Telemetry", *IEEE Oceans*, 1983, pp.167-175

[7] BERLEKAMP, ELWYN R., ROBERT J. McELIECE, e HENK C. A. VAN TILBORG, "On the Inherent Intractability of Certain Coding Problems", *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp.384-386, 1978

[8] BLAHUT, RICHARD E., *Theory and Practice of Error Control Codes*, Addison-Wesley, Reading, Mass, USA, 1983

[9] BRADY, M., R. RAGHAVAN, e J. SLAWNY, "Gradient Descent Fails to Separate", *Proceedings of International Conference on Neural Networks*, vol. I, 1988, p.649-656

[10] BRUCK, JEHOSHUA, e JOSEPH W. GOODMAN, "A Generalized Convergence Theorem for Neural Networks", *IEEE Transactions on Information Theory*, vol. 34, no. 5, pp.1089-1092, Setembro 1988

[11] BRUCK, JEHOSHUA, e MARIO BLAUM, "Neural Networks, Error-Correcting Codes, and Polynomials over the Binary *n*-Cube", *IEEE Transactions on Information Theory*, vol. 35, no. 5, Setembro 1989

[12] BRUCK, JEHOSHUA, e JOSEPH GOODMAN, "On the Power of Neural Networks for Solving Hard Problems", *Journal of Complexity*, vol 6, 1990, pp.129-135

[13] CAID, WILLIAM R., e ROBERT W. MEANS, "Neural Network for Error Correcting Decoders and Convolutional Codes", *Proceedings of IEEE Global Communication Conference - GLOBECOM'90*, 1990,
pp. 1028-1031

[14] CARPENTER, GAIL A., e STEPHEN GROSSBERG, "The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network", *Computer*, March 1988, vol.21, no. 3, pp. 77-88

[15] CARPENTER, GAIL A., "Adaptive Resonance Theory", *IEEE Home Video Tutorial*, cópias das transparências, 1988

[16] CARVALHO, LUÍS ALFREDO VIDAL DE, *Síntese de Redes Neuronais com Aplicações à Representação do Conhecimento e à Otimização*, Tese de Doutorado - Universidade Federal do Rio de Janeiro, COPPE, 1989

[17] CHERKASSKY, VLADIMIR, e NIKOLAOS VASSILAS, "Perfomance of Backpropagation networks for Associative Database Retrieval", *Proceedings of Internation Joint Conference on Neural Networks*, vol.I, 1989, pp. 77-84

[18] CHIEN, ROBERT T., "Block-Coding Techniques for Reliable Data Transmission", *IEEE Transactions on Communication Theory*, vol. 19, no. 5, October 1971

[19] CHIUEH, TZI-DAR, e RODNEY GOODMAN, "A Neural Network Classifier Based on Coding Theory", *Neural Information Processing Systems*, American Institute of Physics, New York, 1988, pp. 174-183

[20] ELIA, MICHELE, "Algebraic Decoding of the (23,12,7) Golay Code", *IEEE Transactions on Information Theory*, vol. 33, no. 1, January, 1987

[21] CARVALHO, LUÍS ALFREDO VIDAL DE, VALMIR BARBOSA, LEILA E. RIPOLL, SUELI B. T. MENDES, FELIPE M. G. FRANÇA, "Redes Neuronais Artificiais: A volta do Cérebro Eletrônico ?", *Ciência Hoje*, vol.12, n°. 70, p.12-21, Janeiro/Fevereiro, 1991

[22] GAMAL, ABBAS A. EL, LANE A. HEMACHANDRA, ITZHAK SHPERLING, e VICTOR K. WEI, "Using Simulated Annealing to Design Good Codes",*IEEE Transactions on Information Theory*, vol. 33, no. 1, Janeiro 1987

[23] HAMMING, R.W., "Error Detecting and Error Correcting Codes", *Bell System Technical Journal*, vol.29, Abril 1950, pp. 147-160

[24] HERTZ, JOHN, ANDERS KROGH, e RICHARD G. PALMER, *Introduction to the Theory of Neural Computation*, Cap. 9 - "Unsupervised Competitive Learning", Addison-Wesley, 1991

[25] HOPFIELD, J. J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", *Proceedings of the National Academy of Sciences, USA*, vol. 79, pp. 2554-8, 1982, *in Neurocomputing - Foundations of Research*, James A. Anderson, e Edward Rosenfeld, eds., The MIT Press, Cambridge, Masschusetts, 1989

[26] HOPFIELD, J.J., "Neurons with graded response have collective computational properties like those of two-state neurons", *Proceedings of the National Academy of Sciences, USA*, vol. 81, May 1984, pp. 3088-3092

[27] HUANG, WILLIAM Y., e RICHARD P. LIPPMANN, "Neural Net and Traditional Classifiers", *Neural Information Processing Systems*, American Institute of Physics, New York, pp.387-396, 1988

[28] HUSSAIN, M., JING SONG e J.S. BEDI, "Neural Network Application to Error Control Coding", *Proceedings of the SPIE*, vol. 1294, *Applications of Artificial Neural Networks*, 1990, pp. 502-509

[29] HUSSAIN, M., e J. S. BEDI, "Perfomance Evaluation of Different Nerual Network Training Algorithms in Error Control Coding", *Proceedings of the SPIE*, vol. 1469, *Applications of Artificial Neural Networks II*, 1991, pp. 697-706

[30] JACOBS, IRWIN M., "Practical Application of Coding", *IEEE Transactions on Information Theory*, vol. 20, no. 3, May 1974

[31] JEFFRIES, CLARK, "Code Recognition With Neural network Dynamical Systems", *SIAM Review*, vol.32, no.4, pp.636-651

[32] JEFFRIES, CLARK, "High-order Neural Models for Error Correcting Code", *Proceedings of the SPIE*, vol. 1294, *Applications of Artificial Neural Networks*, 1990, pp. 510-517

[33] JOURDAIN, G., J.Y. JOURDAIN, "Characterisation of Submarine Acoustic Transmission Channels", *Underwater Acoustics and Signal Processing*, 1981

[34] KIMOTO, TAKASHI, KAZUO ASAKAWA, MORIO YODA, e MASAKAZU TAKEOKA, "Stock Market Prediction System with Modular Neural Networks", *Proceedings of International Conference on Neural Networks*, vol. I, pp. 1-6, 1990

[35] KIRKPATRICK, S., C. D. GELATT, Jr., e M. P. VECCHI, "Optimization by Simulated Annealing", *Science*, pp. 671-680, 13 May 1983

[36] KOHONEN, TEUVO, "Self-Organizing Maps", *IEEE Home Video Tutorial*, texto da palestra, 1988

[37] KOSKO, BART, "Adaptive Bidirectional Associative Memories", *Proceedings of the SPIE*, vol. 1142, *Applications of Artificial Neural Networks*, pp. 532-545, 1987

[38] KUH, ANTHONY, e BRADLEY W. DICKINSON, "Information Capacity of Associative Memories", *IEEE Transactions on Information Theory*, vol. 35, no. 1, January 1989, pp.59-68

[39] LEHMEN, A. VON, E.G. PAEK, P.F. LIAO, A. MARRAKCHI, e J.S. PATEL, "Factors Influencing Learning by Backpropagation", *Proceedings of International Conference on Neural Networks*, vol. I, pp. 335-341, 1988.

[40] LIPPMANN, R.P., "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine*, vol.4, no. 2, Abril 1987

[41] LIN, SHU, *An Introduction to Error-Correcting Codes*, Prentice Hall, Englewood Cliffs, New Jersey, USA, 1970

[42] MACHADO, REMO ZAULI MACHADO Filho, MANOEL F. TENORIO, J. R. M. SILVA, "Neural Network Corrector for Binary Messages on Hydroacoustic Channels", *IEEE Journal of Ocean Engineering*, (versão *pre-print*), 1992

[43] PETERSON, W.W., *Error-Correcting Codes*, The M.I.T. Press, Cambridge, Mass, USA, 1961

[44] PROVENCE, JOHN D., "Neural Network Implementation for an Adaptive Maximum-Likelihood Receiver", *International Symposium on Circuits and Systems*, 1988, pp. 2381-2385

[45] ROCHA JÚNIOR, VALDEMAR CARDOSO DA, "Um Decodificador para Códigos de Hamming", UFPE - Departamento de Eletrônica e Sistemas

[46] ROTH, RON M., e GADIEL SEROUSSI, "Encoding and Decoding of BCH Codes Using Light and Short Codewords", *IEEE Transactions on Information Theory*, vol. 34, no. 3, May 1988, pp.593-597

[47] RUMELHART, DAVID E., JAMES L. McCLELLAND, e grupo de pesquisa PDP, *Parallel Distributed Processing - Explorations in the Microstructure of Cognition, (Volume 1: Foundations)*, The M.I.T. Press, Mass, USA, 1986

[48] RYAN, THOMAS W., "The Resonance Correlation Network", *Proceedings of the International Joint Conference on Neural Networks*, vol.I, pp.673-680, 1988

[49] SIMPSON, PATRICK K., *Artificial Neural Systems*, Pergamon Press, 1989

[50] STEVENS, PATRICK, "Extension of the BCH Decoding Algorithm to Decode Binary Cyclic Codes up to Their Maximum Error Correction Capacities", *IEEE Transactions on Information Theory*, vol. 34, no. 5, Setembro 1988, pp.1332-1340

[51] VITERBI, ANDREW J., "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm", *IEEE Transactions on Information Theory*, vol. 13, no. 2, 1967

[52] WITTNER, BEN S., e JOHN S. DENKER, "Strategies for Teaching Layered Networks Classification Tasks", *Neural Information Processing Systems*, American Institute of Physics, New York, 1988, pp. 850-859

[53] YU, YEONG-HO, e ROBERT F. SIMMONS, "Extra Output Biased Learning", *Proceedings of the International Joint Conference on Neural Networks* (pre-print), 1990

[54] YUAN, JING, VIJAY K. BHARGAVA, e Q. WANG, "Maximum Likelihood Decoding Using Neural Nets", *Journal of the Institution of Electronics & Telecommunication Engineers*, vol. 36, nos. 5 & 6, pp. 367-376

[55] YUAN, JING, e C.S. CHEN, "Neural Net Decoders for some Block Codes", *IEE Proceedings I (Communications, Speech and Vision)*, vol. 137, part I, no.5, pp.309-314, October,, 1990

[56] ZENG, Gengsheng, DON HUSH, e NASIR AHMED, "An Application of Neural Net in Decoding Error-Correcting Codes", *Proceedings of the IEEE International Symposium on Circuits and Systems*, 1989, pp.782-785.

[57] McELIECE, R., E.POSNER, E. RODEMICH, e S. VENKATESH, "The Capacity of the Hopfield Associative Memory", *IEEE Transactions on Information Theory*, vol. 33, pp. 461-482, 1987, *apud* [49], pp. 47 e 49.

Apêndice

Programa de Decodificação BCH

Segue-se, apenas a título de ilustração, um conjunto de rotinas que implementam o algoritmo de decodificação de Berlekamp, na linguagem MatLab, desenvolvidas como parte deste trabalho.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% eg.m - 15-jul-1991 - enhsatuf
%        Exemplo de decodificacao: BCH
!del deg.
diary deg
clear;
clear functions;
global TRUE FALSE G_POTALFA g_m g_m2 g_t g_k; % g_ ou G_ :
variaveis globais
global CASO;

disp(' ');

disp('  Codigos  de  Golay  colocados  como  exemplo  de
codificacao. ');

disp(' Codigos de Golay NAO sao BCH. ');
disp(' ');
s = [
' 1 - ( 7, 4, t=1) Codigo de Hamming '
' 2 - (15,11, t=1)                   '
' 3 - (15, 7, t=2)                   '
' 4 - (15, 5, t=3)                   '
' 5 - (23,12, t=3) Codigo de Golay  I'
' 6 - (23,12, t=3) Codigo de Golay II' ];

disp(s);
CASO = input(' Entre caso  ');
versao = input(' Entre versao (1 ou 2) ');
clear s

if CASO < 1  |  CASO > 6
   error(' CASO nao existente. ');
   return;
   end;


TRUE = 1; FALSE = 0;
g_m=4; g_m2 = 2^g_m - 1;

X_POTALFA = [

     1     0     0     0     0    -1
     2     1     0     0     0     0
```

```
      3      0      1      0      0      1
      4      0      0      1      0      2
      5      0      0      0      1      3
      6      1      1      0      0      4
      7      0      1      1      0      5
      8      0      0      1      1      6
      9      1      1      0      1      7
     10      1      0      1      0      8
     11      0      1      0      1      9
     12      1      1      1      0     10
     13      0      1      1      1     11
     14      1      1      1      1     12
     15      1      0      1      1     13
     16      1      0      0      1     14   ] ;

G_POTALFA = [

      0      0      0      0
      1      0      0      0
      0      1      0      0
      0      0      1      0
      0      0      0      1
      1      1      0      0
      0      1      1      0
      0      0      1      1
      1      1      0      1
      1      0      1      0
      0      1      0      1
      1      1      1      0
      0      1      1      1
      1      1      1      1
      1      0      1      1
      1      0      0      1   ];

g_k = 1;
msg = zeros(1,g_k); msg(g_k) = 1       % Pega mensagem
msg = input(' Entre msg(1:g_k) ')

if  versao == 2
   v = bchcod(msg)                     % Codifica
else
   v = bchcod1(msg)
   end;

r = v;
aux=input(' Erros aletorios ? (NAO = 0) ')  % Introduz erros
if aux
   for i=1:g_t;
       r = r + noise(g_m2,v);
   end;
else
    err=input(' Digite posicoes dos erros (0:n) ou 99 ')
    if err(1) == 99  break;
    else
        qterr = size(err);
        for i=1:qterr(2);
            pos = err(i) + 1;
            r(pos) = r(pos) + 1;
        end;
```

```
    end;
end;
r = rem(r,2)

u=bch(r);                % Decodifica
u(g_m2-g_k+1:g_m2)       % Mostra mensagem.

%
% Exemplo em Shu Lin, cap.6, p.123 e seguintes.
%
% Para transmitido v = zeros(1,15);
% Recebido g_r = [0 0 0 1 0 1 0 0 0 0 0 0 1 0 0];
% Resultados esperados:
%                                      Sindromes:
%  S(1) = [1 0 0 0] => 1
%  S(2) = [1 0 0 0] => 1
%  S(3) = [1 1 1 0] => alfa^10
%  S(4) = [1 0 0 0] => 1
%  S(5) = [1 1 1 0] => alfa^10
%  S(6) = [0 1 1 0] => alfa^5
%
%  SIGMA = [1 0 0 0; 1 0 0 0; 0 0 0 0; 0 1 1 0] =>
%              => 1 + X + alfa^5*X^3
%
% se transmitido v = [1 1 0 1 1 0 0 1 0 1 0 0 0 0 1]
% entao          r = [1 0 0 0 1 0 1 1 0 1 0 0 0 0 1];
% Outros exemplos: ...
% r = [0 0 0 1 0 1 0 0 0 0 0 0 1 0 0];  % shu lin: 3 erros
% r = zeros(1,15);                      % tudo zero
% r = [0 1 0 1 0 1 0 0 0 0 0 0 1 0 0];
% shu lin + um erro = 4 erros
%
clear functions
diary off;
% end eg.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function v=bchcod(msg)
% Codificador BCH - Shu Lin Cap. 6 - 12-JUN-92 - versao 2 -
% bchcod.m

s = [
'load COD741;   GMENOR = G74MENOR;   g_k =  4; g_t = 1'
'load COD15111; GMENOR = G1511MENOR; g_k = 11; g_t = 1'
'load COD1572;  GMENOR = G157MENOR;  g_k =  7; g_t = 2'
'load COD1553;  GMENOR = G155MENOR;  g_k =  5; g_t = 3'
'load COD23123; GMENOR = G2312MENOR; g_k = 12; g_t = 3'
'load COD2312C; GMENOR = G2312MENOR; g_k = 12; g_t = 3' ];

 if  ~exist('g_k')
   disp(s);
   CASO = input(' Entre caso ');
   end;

eval(s(CASO,:))
G = [GMENOR eye(g_k)];    % matriz geradora

v = rem(msg*G,2);
```

```
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function v=bchcod(msg)
% Codificador BCH - Shu Lin Cap. 6 - ago-91 - versao 1. -
% bchcod1.m

disp(' BCHCOD1 Codificador BCH versao 1. ');

A = [
   1   1   1   0   1   1   0   0   1   0   1   0 ...
0   0   0;

   0   1   1   1   0   1   1   0   0   1   0   1 ...
0   0   0;

   1   1   0   1   0   1   1   1   1   0   0   0 ...
1   0   0;

   0   1   1   0   1   0   1   1   1   1   0   0 ...
0   1   0;

   1   1   0   1   1   0   0   1   0   1   0   0 ...
0   0   1      ];

B = [1 0 0 0 1 0 1 1
     1 1 0 0 1 1 1 0
     0 1 1 0 0 1 1 1
     1 0 1 1 1 0 0 0
     0 1 0 1 1 1 0 0
     0 0 1 0 1 1 1 0
     0 0 0 1 0 1 1 1 ];


if CASO == 3
   g_k = 7;
   g_t = 2;
   B = [B eye(g_k)];
   G = B;
elseif CASO == 4
   g_k = 5;
   g_t = 3;
   G = A;
else
   error(' Caso nao previsto para esta rotina. ');
   return;
   end;


v = rem(msg*G,2);
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function ruido=noise(n,u)
% NOISE - 11-jun-1991 - satuf - gera ruido num vetor

rand('uniform') ;
```

```
ruido = zeros(1,n) ;
ruido(fix(rand*n+1)) = 1 ;
return ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function  u=bch(r)
% BCH.m - Decodificacao BCH - Shu Lin - cap.6. r: vetor
recebido

SS = sindr(r)     % --- ;
SIGMA=elp(SS)     % --- ;
%SIGMA = [1 0 0 0; 1 0 1 1; 0 1 0 1; 0 0 0 0]
beta=eln(SIGMA)     % --- ;
u=corr(r,beta)      % --- ;
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function SS=sindr(r)
% Calcula as sindromes do vetor recebido r.

t2 = 2*g_t; Y = zeros(t2,g_m); % 2.t linhas x m colunas
for i=1:t2;
    for j=1:g_m2;
        if r(j)
           k=rem(i*(j-1), g_m2) + 2;
           Y(i,:) = Y(i,:) + G_POTALFA(k,:);
           end;
        end;    % for2j
    end;        % for1i
SS = rem(Y,2);
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function SIGMA = elp(SS) ;
% ELP - Error Location Polynomial - Calcula o polinomio de
%  locacao de erros
%  - sigma(X). Ref. Shu Lin, Cap. 6 - BCH. SS: sindromes

Lcol='2*mi - 1';    % "Last column"
imax=g_t + 2;
mmi = [-0.5 0 1 2 3];

% Primeira linha ...
i = 1; mi = mmi(i); imi = i;
SIGMAS=[1 0 0 0; 0 0 0 0; 0 0 0 0; 0 0 0 0];
                                % Matriz com os polinomios
                               % Cada polinomio, uma submatriz
Dd  =  [1  0  0  0];  l  =  grau(SIGMAS,i);  ll  =  [l];
lcol=[eval(Lcol)];

% Segunda linha ...
i = i + 1; mi = mmi(i); imi = i; SIGMAS=[SIGMAS SIGMAS];
l = grau(SIGMAS,i); ll = [ll; l];
dd = SS(1,:);                   % dmi(i, mi, ll, SIGMAS, SS);
Dd = [Dd; dd]; lcol=[lcol; eval(Lcol)];
```

```
% Terceira linha e seguintes ...
while i < imax
   if ~any(dd)                    % Se d(mi) e' zero ...
      SIGMA = SIGMAS(:,(i-1)*g_m+1:g_m*i);
      SIGMAS=[SIGMAS SIGMA];  % ... e' facil ...
   else                           % ... senao ...
      aux=acharo(lcol, mmi, Dd);   % ... nem tanto.
      ro=aux(1); iro=aux(2);
      imi = i;
      SIGMA = sigmam(SIGMAS,Dd,mi,ro,imi,iro);  % chamada um
      SIGMAS = [SIGMAS SIGMA];
      end;
   l = grau(SIGMAS,imi+1); ll = [ll; l];
   if mi + 1 < g_t
      dd = dmi(imi, mi, ll, SIGMAS, SS); Dd=[Dd; dd];
      end;
   i = i + 1; mi = mmi(i);
   lcol = [lcol; eval(Lcol)];
end;
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function beta = eln(SIGMA);
% Calcula as posicoes de erro. Shu Lin p.127

beta = [-1 -1 -1]; j = 0;
for i=2:g_m2+1;
    nalfa = G_POTALFA(i,:);
    el = pol(SIGMA,nalfa);
    if ~any(el)
       a = 17 - i;
       j = j + 1;
       beta(j) = a;
       if beta(j) > g_m2-1  beta(j)=rem(beta(j),g_m2); end;
       end;
end;
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function u = corr(r,beta)
% Corrige vetor recebido r(0:) nas posicoes beta

u = r;
for i=1:g_t;
   if  beta(i) ~= -1
       a = beta(i)+1; u(a) = r(a) + 1;
       end;
   end;
u=rem(u,2);
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function l = grau(SIGMAS,i)
% Retorna o grau do "polinomio-alfa" sigmaX

col1=g_m*(i-1)+1; col2=g_m*i;
```

```
SIGMA=SIGMAS(:,col1:col2);
if ~any(any(SIGMA))  l = 0; return; end;
k = g_m;
while k > 0
   if ~any(SIGMA(k,:))
      k = k - 1;
   else
      l = k - 1; k = 0;
   end;
end;
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y=acharo(lcol, mmi, Dd)
% Achar ro e correspondente iro. Shu Lin p.125

ACHOU = FALSE; ro = 0; iro = 0; k = 0;
x = size(lcol); lcol2 = lcol(1:x(1)-1);
while ~ACHOU
      if all(lcol2 == -999)
         error(' IMPOSSIVEL calcular RO. ');
      end;
      [a b] = max(lcol2);
      if  ~any(Dd(b,:))
          lcol2(b) = -999;          % Considerar se d(ro) =
0
      % ### error(' d(ro) eh zero, nao posso calcular ro. ');
      else
          ro = mmi(b); iro = b;
          ACHOU = TRUE;
      end;
end;
y = [ro iro];
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function SIGMA=sigmam(SIGMAS,Dd,mi,ro,imi,iro)
% Calculo recursivo de sigma(X) - (mi+1). Ref. Shu Lin p.125

SIGMAMI = SIGMAS(:, g_m * (imi-1)+1 : g_m * imi);
                                          % sigma(mi)(X)
SIGMARO = SIGMAS(:, g_m * (iro-1)+1 : g_m * iro);
                                          % sigma(ro)(X)
dimi = Dd(imi,:); diro=Dd(iro,:);
diro=pot(diro,-1);
e=2*(mi-ro); k=mult(dimi,diro); P1 = coef(k,e);
P2 = pmult(P1,SIGMARO);
SIGMA=SIGMAMI + P2;
SIGMA=rem(SIGMA,2);
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function dd = dmi(imi, mi, ll, SIGMAS, SS)
% Calcula d(mi+1). Algoritmo BCH, Shu Lin, p.125

j=0; jmax=ll(imi+1,:)+1; dd = zeros(1, g_m); i = imi + 1;
```

```
while j < jmax
   ss = SS(2 * mi + 3 - j, :);
   sigma = SIGMAS(j+1,g_m*(i-1)+1:g_m*i);
   parcela = mult(sigma,ss);
   dd = soma(dd,parcela);
   j = j + 1;
end;
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Y=coef(k,e)
% Coeficiente k (elemento GF(2)), potencia e: k*X^e. Shu Lin
%  p.125

Y=zeros(g_m,g_m); Y(e+1,:) = k;
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function C=pmult(A,B)
% Polynomial MatrixMultiplication for elp - error location
%  polynomial.
% Ref. Shu Lin p.125 - A, B, C: polinomios sigma(X)

tamA=size(A); tamB=size(B); C=zeros(2*g_m - 1,g_m); I=C;
for i=1:tamA(1);
    a = A(i, :);
    for j=0:(tamB(1)-1);
        b = B(j+1, :);
        if any(a)  &  any(b)
           I(i+j,:) = mult(a, b);
           C = C + I;
           I(i+j,:) = zeros(1,g_m);
           end;
        end;
    end;
C=rem(C,2);
if any(any(C(g_m+1:2*g_m-1,:)))
                              % Existe coef <> 0 p/ X^4 ou mais.
   error(' IMPOSSIVEL decodificar. Muitos erros. ');
   C=NaN;
   % --- exit;
else
   C=C(1:g_m,:);
   end;
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y=pot(v,n)
% Potenciacao de v (alfa^b) a n (inteiro)

aux=srch(v,G_POTALFA) - 2; aux = aux * n;
if aux < 0      aux = g_m2 - rem(-aux, g_m2); end;
if aux >= g_m2     aux = rem(aux, g_m2); end;
aux = aux + 2;
y = G_POTALFA(aux,:);
return;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y=soma(a,b);
% soma dois vetores, usando modulo-2

y = a + b; y = rem(y,2) ;
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y = mult(a,b)
% Multiplica dois vetores, representando potencia de alfa

res1 = srch(a, G_POTALFA) - 2;
res2 = srch(b, G_POTALFA) - 2;
res  = res1 + res2 ; res = rem(res, g_m2) + 2;
y = G_POTALFA(res,:) ;
return ;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y = srch(n,M)
%SRCH(n,M) - search n in M - satuf - 07-JUN-1991
%            Procura vetor (ou escalar) "n"  na matriz
%            (ou vetor) "M".
% Retorna linha de "M" onde encontrou "n"

[ln cn]=size(n);
[lM cM]=size(M);
if  (ln+cn) == 2   y=find(M==n); y=primo(y); return;  end;
M1=M(:,1:cn);
N=zeros(lM,cn);
for i=1:lM ; N(i,:) = n(1,:) ; end;
DIF=M1-N;
ig=any(DIF');
y=find(ig==0);
y=primo(y);
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y = primo(v)
%PRIMO(v) - Primeiro de "v" - satuf - 10-JUN-1991
%           v => vetor
if  isempty(v)   y=0;  return; end;
y=v(1);
return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

G74MENOR =

     1     1     0
     0     1     1
     1     1     1
     1     0     1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

G2312MENOR =

```
1     0     1     0     1     1     1     0     0     0     1
1     1     1     1     1     0     0     1     0     0     1
1     1     0     1     0     0     1     0     1     0     1
1     1     0     0     0     1     1     1     0     1     1
1     0     0     0     1     1     0     1     1     0     0
0     1     1     0     0     1     1     0     1     1     0
0     0     1     1     0     0     1     1     0     1     1
1     0     1     1     0     1     1     1     1     0     0
0     1     0     1     1     0     1     1     1     1     0
0     0     1     0     1     1     0     1     1     1     1
1     0     1     1     1     0     0     0     1     1     0
0     1     0     1     1     1     0     0     0     1     1
```

%%%

G155MENOR =

```
1     1     1     0     1     1     0     0     1     0
0     1     1     1     0     1     1     0     0     1
1     1     0     1     0     1     1     1     1     0
0     1     1     0     1     0     1     1     1     1
1     1     0     1     1     0     0     1     0     1
```

%%%

G1511MENOR =

```
1     1     0     0
0     1     1     0
0     0     1     1
1     1     0     1
1     0     1     0
0     1     0     1
1     1     1     0
0     1     1     1
1     1     1     1
1     0     1     1
1     0     0     1
```

%%%