UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE COMUNICAÇÕES
DECOM-FEEC-UNICAMP

A UTILIZAÇÃO DE REDES NEURAIS ARTIFICIAIS NO PROJETO DE RECEPTORES FH-CDMA

Getúlio Antero de Deus Júnior
Orientador: **Jaime Portugheis**

Dissertação de Mestrado

Campinas - SP - Brasil
1998

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE COMUNICAÇÕES
DECOM-FEEC-UNICAMP

A UTILIZAÇÃO DE REDES NEURAIS ARTIFICIAIS NO PROJETO DE RECEPTORES FH-CDMA

Getúlio Antero de Deus Júnior
Orientador: **Jaime Portugheis**

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica
Área de Concentração: Eletrônica e Comunicações.

Campinas - SP - Brasil
1998

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

D488u

Deus Júnior, Getúlio Antero de
A utilização de redes neurais artificiais no projeto de receptores FH-CDMA. / Getúlio Antero de Deus Júnior.--Campinas, SP: [s.n.], 1998.

Orientador: Jaime Portugheis
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Redes neurais (Computação). 2. Rádio – Receptores e recepção. 3. Sistemas de comunicação móvel. 4. Algoritmos. I. Portugheis, Jaime. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Agradecimentos

Como disse certa vez um amigo, muito provável esta seja a parte mais difícil por se tratar de muitas pessoas envolvidas diretamente ou indiretamente neste trabalho. Agradecer sempre é uma tarefa muito difícil, principalmente quando se tem tantas variáveis em um contexto tão amplo e complexo, o qual faz parte este trabalho.

Um editoriral do "Daily Yomiuri" do Japão, em 5 de maio de 1976 observou: "A desarmonia e a fricção que sentimos em nossa vida diária seguramente podem ser atribuídos à nossa má conduta social. Nós, como indivíduos, nos esquecemos de ser mutualmente acomodatícios e gratos uns aos outros." Animando os leitores a expressarem gratidão, o editorial observou adicionalmente: "Grande parte da atual desconfiança internacional resulta desta falta dum espírito acomodatício. Afinal, a gratidão não é algo vergonhoso. Ajuda-nos a assegurar a harmonia social".

Não é fora de propósito a exortação de sermos gratos e acomodatícios. Contudo, para não cometer a injustiça de esquecer algumas das pessoas às quais sou eternamente grato, farei apenas algumas citações de pessoas e instituições que foram fundamentais no contexto deste trabalho:

- Ao único Deus Verdadeiro e Feliz cujo nome é Jeová;
- Aos meus pais Getúlio e Iranísia por sempre confiarem e acreditarem em minha pessoa;

- À toda a minha família, pelos encorajamentos e por terem suportado a minha ausência, em especial aos meus irmãos Cláudio, Eduardo e Ricardo e aos meus avós João e Maria;
- À todos os meus amigos, em especial, Jés de Cerqueira, ao casal Welington-Adriana, ao casal Odacir-Francis e seus filhos Alessandro-Daniela, ao casal Leandro-Cynthia, ao casal Luiz Carlos-Roberta, Alisson, Márcio, Fábio, Cintia, Fernando, Luiz Gustavo, Marcelo, Maria Luiza, Naylamp, Alexandre Osório, Luiz Rômulo, Antônio Alberti, Paula, Veloso, Jaudelice, Kakimoto e tantos outros amigos.
- Ao Prof. Dr. Jaime Portugheis por ter aceito a minha sugestão de tema de trabalho, pela exemplar orientação e por ter se empenhado tanto neste trabalho;
- Ao Prof. Dr. Márcio Luiz de Andrade Netto, pela co-orientação amiga e atenciosa;
- Aos membros da banca examinadora desta dissertação, o Prof. Dr. Márcio Luiz de Andrade Netto e Prof. Dr. Sindi, por ter aceito o convite;
- Ao Prof. Dr. José Fernando Von Zuben por sugestões práticas na implementação de algoritmos por aprendizado;
- Ao Prof. Dr. Weber Martins da Escola de Engenharia Elétrica de Goiás pelos comentários extremamente apreciados;
- Ao Prof. Dr. Lee Luan Ling pela cooperação técnica;
- À todos os funcionários da FEEC, em especial os funcionários Juraci, Mário, Ricardo e Washington, às funcionárias Cristina, Lúcia, Mara e Noêmia e a ex-funcionária Renata;
- À UNICAMP por ter me proporcionado esta oportunidade de pesquisa;
- Ao CPQD/TELEBRÁS pelo convênio de pesquisa extra-curricular, em especial à pessoa do Eng. Dr. Sindi;
- À CAPES pela concessão de uma bolsa de estudos durante toda a realização do curso.

" Ó Jeová, tu és o meu Deus. Eu te exalto, elogio o teu nome, pois fizeste maravilhas, conselhos desde tempos anteriores, em fidelidade, em fidedignidade. E Jeová dos exércitos há de fazer para todos os povos, neste monte, um banquete de pratos bem azeitados, um banquete de [vinhos guardados com a] borra, de pratos bem azeitados, cheios de tutano, de [vinhos guardados com a] borra, filtrados. E neste monte ele há de tragar a face do envoltório que envolve todos os povos e o trabalho tecido que está entretecido sobre todas as nações. Ele realmente tragará a morte para sempre, e o Soberano Senhor Jeová certamente enxugará as lágrimas de todas as faces. E de toda a terra ele tirará o vitupério de seu povo, pois o próprio Jeová falou [isso]."

Isaías 25:1,6-8 *Tradução do Novo Mundo das Escrituras Sagradas*

Listas de Figuras

Listas de Tabelas

Sumário

Resumo

DEUS JÚNIOR, G. A. **A utilização de Redes Neurais Artificiais no projeto de receptores FH-CDMA.** Campinas: FEEC, UNICAMP, 1998. Dissertação (Mestrado) - Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, 1998. 100p.

Uma das possibilidades do serviço de comunicações digitais é a investigação da técnica de modulação de espalhamento espectral utilizando chaveamento por deslocamento em freqüência.

Um sistema FH-CDMA proposto na literatura pode ser representado por matrizes de transmissão e de recepção. A regra de decisão que minimiza a probabilidade de erro de mensagem no receptor é conhecida por lógica majoritária.

A partir do diagrama de blocos para o receptor de lógica majoritária, é apresentada uma forma de Rede Neural construída sem treinamento. Apresenta-se a análise de uma rede paramétrica de *Kohonen* com o objetivo de mostrar que suas operações resultam no cálculo de valores intermediários da rede construída sem treinamento. Propõe-se também uma estrutura de rede multi-camada. Além disso, avalia-se a utilização de códigos de bloco para a representação dos padrões de saída para as Redes Neurais *feedforward,* com o objetivo de reduzir o número de neurônios da camada de saída da rede.

Palavras Chave: Redes Neurais (Computação), Receptor por Lógica Majoritária, Receptores FH-CDMA, Algoritmos por Aprendizado.

Abstract

DEUS JÚNIOR, G. A. **The project of FH-CDMA receivers using Neural Networks.** Campinas: FEEC, UNICAMP, 1998. Dissertação (Mestrado) - Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, 1998. 100p.

A possible application of digital communication service is the investigation of spread spectrum modulation using *frequency shifting keying* technique and a majority rule receiver.

As proposed in the literature, a FH-CDMA system can be represented in terms of transmission and reception matrixes. The decision rule that minimizes the probability of message mistake in the receiver is known as majority logic.

Starting from block diagram of the majority logic receiver, a kind of neural network built-up without training is then presented. A Kohonen parametric neural network is analysed. Its operations result in the calculation of the intermediary values of the neural network built-up without training. A multilayer neural network structure is also proposed. In order to reduce the number of neuron in the output layer of the neural network, a study using block codes to represent the output patterns of the feedforward networks is evaluated.

Key words: Neural Networks (Computation), Majority Rule Receiver, FH-CDMA Receivers, Learning Algorithms.

Introdução

Por mais de 130 anos a humanidade vem observando grandes mudanças com respeito à tecnologia. Atividades que antes eram imaginadas apenas pela ficção científica, hoje são realidades. Por exemplo, poderia alguém ter previsto, mais de 130 anos atrás, o surgimento de automóveis, aviões, computadores e muitos outros inventos? O escritor *Júlio Verne*, previu! No seu romance, não publicado na época, *Verne* até mesmo descreveu um maquinismo que se parece demais com um moderno aparelho de *fax*! (Nas palavras de *Verne*, "um telégrafo fotográfico [que] permitia o despacho a longas distâncias do fac-símile de qualquer escrito, assinatura ou projeto". - Revista *Newsweek*, 10 de outubro de 1994.)

No entanto, a busca de solução de problemas através da tecnologia muitas vezes levam ao surgimento de novos problemas que antes não existiam. Quem não se lembra da criação de um dos primeiros computadores, o velho ENIAC? Para o "perfeito" funcionamento do equipamento, foi necessário a busca de soluções de muitos outros problemas, tais como a refrigeração de circuitos, enorme quantidade de componentes, que não se adaptavam facilmente em qualquer bastidor e assim por diante. Mas, os problemas foram solucionados e hoje temos um computador super poderoso que cabe em nossas mãos!

Se de um lado, a tecnologia agrícola expulsou o homem do campo trazendo-lhe muitos problemas, de outro lado, ela trouxe uma super produção de alimentos! Deste modo, mais uma vez, vemos que a tecnologia vem sempre para a ajudar o homem a resolver determinados problemas trazendo uma nova gama de problemas. Assim, antes da aplicação de novas tecnologias, o homem deveria analisar bem o que o emprego delas causaria ao próprio homem e ao meio ambiente.

M. G. Marconi quando fez a sua primeira transmissão sem fio, conectando o continente europeu a um navio, através do envio de sinais elétricos codificados (telegrafia), talvez nem imaginava que o homem algum dia poderia se comunicar através da fala utilizando um equipamento tão promissor, como o atual telefone móvel, embora *A. G. Bell* tivesse consolidado a invenção do primeiro telefone.

Já há alguns anos têm sido realizadas pesquisas na área de sistemas de comunicações móveis com grandes avanços [1, 2, 3]. Na verdade, o surgimento de uma grande demanda pelo uso do telefone móvel pressionou a implementação prática de diversos sistemas. Existem dois tipos básicos de técnicas de espalhamento espectral para a implementação de um sistema de comunicação móvel utilizando tecnologia de acesso CDMA (Code Division Multiple Access): *técnica de espalhamento por sequência direta (sistemas DS)* e *técnica de espalhamento por salto em freqüência (sistemas FH)*. O desempenho dos sistemas FH são mais significantes do que os sistemas DS. Também a capacidade dos sistemas FH é relativamente alta. [4]. No entanto, a implementação de sistemas DS é mais simples devido o fato da grande dificuldade de se realizar os saltos das freqüências nos sistemas FH.

Também na área de Redes Neurais Artificiais, tanto a pesquisa teórica quanto a pesquisa aplicada, têm demonstrado avanços [5, 6, 7]. As Redes Neurais Artificiais são modelos matemáticos inspirados no sistema nervoso biológico, que dentre outras propriedades, são capazes de aprenderem pela experiência, de generalizarem exemplos e de abstraírem características de um meio. Os elementos de processamento de uma Rede Neural Artificial são os neurônios, cuja função de transferência não-linear gera uma saída respondendo à uma determinada entrada fornecida.

Este trabalho inédito motivou-se na utilização das Redes Neurais Artificiais no projeto de receptores FH-CDMA.

Alguns objetivos principais deste trabalho podem ser citados. São eles:

i. Proposta de um receptor FH-CDMA baseado em modelos por de Redes Neurais Artificiais;
ii. Aquisição de conhecimentos básicos em Neurais Artificiais que possibilitem, no futuro, outras aplicações em sistemas de comunicações;

Este trabalho se encontra dividido em 4 capítulos. No capítulo 1 é feito um estudo dos conceitos básicos de álgebra de campos finitos, de códigos *Reed-Solomon* e da decodificação por máxima verossimilhança, tópicos estes relacionados a um sistema FH-CDMA.

No capítulo 2 é realizada uma introdução às Redes Neurais Artificiais. Basicamente, as redes descritas neste capítulo são baseadas nas estruturas *perceptrons* estáticas e nas redes paramétricas de *Kohonen.*

No capítulo 3 é apresentada uma introdução aos sistemas FH-CDMA. É apresentado o detector FH-FSK *(Frequency Hopping - Frequency Shift Keying)* como uma das possíveis técnicas de espalhamento espectral de sinais digitais utilizando-se a modulação FSK. Em seguida, são apresentados os sistemas de endereçamento proposto por *Einarsson* e os limitantes de união das probabilidades de erro para estes sistemas. Os sistemas propostos por Einarsson sugerem a aplicação de uma regra de decodificação da mensagem recebida no receptor, que minimiza a probabilidade de erro, conhecida por *lógica majoritária.* No final do capítulo, são apresentados outros sistemas de recepção multi-usuários.

No capítulo 4 são realizadas as aplicações de Redes Neurais Artificiais para a obtenção de um receptor de lógica majoritária. É proposto um receptor de lógica majoritária baseado numa Rede Neural construída sem treinamento. Em seguida, é feita uma análise para a rede de *Kohonen* que levam aos resultados

para o cálculo do valor de uma variável da Rede Neural construída sem treinamento. Os estudos com Redes Neurais *feedforward* e códigos de bloco são realizados com o objetivo de reduzir o número de neurônios da camada de saída da rede.

Por fim, conclusões, comentários sobre os resultados obtidos e sugestões para trabalhos futuros são apresentados após o último capítulo.

Capítulo 1

1. Conceitos Básicos

Este Capítulo apresenta uma introdução à álgebra dos campos finitos, aos códigos *Reed-Solomon* (RS) e à decodificação por máxima verossimilhança (MV). Na primeira parte deste capítulo é apresentada uma introdução à álgebra de campos finitos, sobretudo visando o estudo elementar de campos finitos *GF(p)* e $GF(p^k)$ [8, 9, 10], mais comumente conhecidos como *campos de Galois*. Numa segunda parte deste capítulo são apresentados os códigos *Reed-Solomon* (RS), muito utilizados para combater surtos de erros [8]. Os códigos *Reed-Solomon* compreendem uma subclasse especial dos códigos BCH q-ários (*BCH - de Bose, Chaudhuri, and Hocquenghem*). Na parte final deste capítulo é apresentada a decodificação por máxima verossimilhança. A complexidade do detector por MV pode ser bastante reduzida utilizando-se o algoritmo de *Viterbi* ou outros algoritmos.

1.1 Introdução à Álgebra de Campos Finitos

A seguir são consideradas algumas definições e conceitos importantes relacionados à álgebra de campos finitos e muito utilizo neste trabalho.

1.1.1 Campos

A definição formal de *campo* é dada a seguir.

Definição 1 *Um campo* $\langle C,+,. \rangle$ *é um conjunto C junto com duas operações binárias "+" e "." (as quais são chamadas adição e multiplicação) tais que as seguintes propriedades são satisfeitas:*

(i) C é um grupo comutativo sob a adição "+". O elemento identidade com respeito à adição é chamado de elemento nulo ou identidade aditiva de C e é denotado por 0.

(ii) O conjunto de elementos não nulos em C é um grupo comutativo sob a multiplicação ".". O elemento identidade com respeito à multiplicação é chamado elemento unitário ou identidade multiplicativa em C e é denotado por 1.

(iii) Para os três elementos a, b, c em C, a multiplicação é distributiva sobre a adição à esquerda, a (b + c) = (a b) + (a c) e distributiva à direita, (a + b) c = (a c) + (b c).

Como exemplos de campos podemos citar: o conjunto dos número racionais sob adição e multiplicação, o conjunto dos número reais sob adição e multiplicação, o conjunto Z_p = { 0, 1, 2, ..., p-1 }, (p primo), sob adição e multiplicação módulo p. Note entretanto que os inteiros **Z** *não* formam um campo sob adição e multiplicação, assim como os conjuntos Z_m = { 0, 1, 2, ..., m-1 } sob adição e multiplicação módulo m, onde m não é primo.

Neste trabalho, estaremos interessados em campos finitos sobre os quais os códigos estarão definidos, isto é, os campos Z_p, e suas extensões (a serem apresentadas no próximo item). Um campo finito contendo p elementos é denotado por *GF(p)*.

1.1.1.1 Campos Finitos *GF(p)*

As propriedades para campos finitos em *GF(p)* estão apresentadas a seguir através das propriedades 1 e 2.

Propriedade 1 *Para qualquer primo p, existe um campo finito de p elementos denotado por GF(p).*

Propriedade 2 *Para qualquer inteiro positivo k, é possível estender o campo primo GF(p) para um campo de p^k elementos que é chamado extensão do campo GF(p) e é denotado por $GF(p^k)$.*

Os campos finitos definidos nesta seção são comumente conhecidos na literatura especializada por *Campos de Galois* (em homenagem ao matemático francês *Évarist Galois*). O motivo de se aprofundar o estudo nos campos finitos *GF(p)* reside no fato de que as palavras-código do endereçamento dos sistema FH-CDMA utilizados neste trabalho estarem definidas sobre um campo de Galois.

As regras de adição e multiplicação para um campo em *GF(p)* são definidas por aritmética módulo p. Um exemplo de campo finito sob a *operação adição módulo-5* e sob a *operação multiplicação módulo-5* é apresentado pela tabela 1.1 e tabela 1.2, respectivamente. O exemplo 1 mostra o uso de adição módulo-11 aplicados aos elementos de *GF(11)*.

$\oplus$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Tabela 1.1: Adição módulo-5

$\otimes$	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

Tabela 1.2: Multiplicação módulo-5

Exemplo 1: *Seja q = 11. Então temos, por exemplo: 3.3=9, 1+6=7, 3.4=1 (=12 mod 11), 10+3=2 (=13 mod 11).*

A *ordem* de um elemento é dado pela definição 2 apresentada logo a seguir.

Definição 2 *Seja G um campo finito. A ordem de* $g \in G$ *é o menor inteiro tal que* $g^m = 1$, $m > 0$.

Com a definição de *ordem* de um elemento, pode-se definir o que vem a ser um *elemento primitivo*.

Definição 3 *Um elemento primitivo de GF(p) é um elemento que apresenta ordem m = q -1.*

O exemplo 2, mostra um *elemento primitivo* para o *GF(11)*.

Exemplo 2: *Em GF(11), o elemento a = 2 tem as potências* a^0, a^1, a^2, ... *= 1, 2, 4, 8, 5, 10, 9, 7, 3, 6, 1, 2..., mostrando que o elemento a = 2 é um elemento primitivo, pois possui ordem igual a 10 (m = 10). Já o elemento a = 3 tem as potências* a^0, a^1, a^2, ... *= 1, 3, 9, 5, 4, 1 não é um elemento primitivo pois a sua ordem é igual a 5 (m = 5).*

Através dos exemplos apresentados anteriormente fica bem evidente que as potências de um *elemento primitivo* são todos os elementos de um campo finito. A seguir veremos um outro importante *campo*, os campo finitos $GF(p^k)$.

1.1.1.2 Campos Finitos $GF(p^k)$

Um campo finito $GF(p^k)$, possui regras para adição e multiplicação um pouco diferentes das apresentadas no item anterior. Considere o caso em que $p = 2$, que é um caso pois as operações podem ser implementadas através de circuitos digitais.

A adição definida no campo finito $GF(2^k)$ é baseada nas operações binárias dos vetores de tamanho k, i.e., é feita para cada componente do vetor uma adição módulo 2. Veja o exemplo 3 logo a seguir.

Exemplo 3: *Seja $q = 2^2 = 4$. Então, os vetores podem ser representados como vetores de tamanho 2: 0 = 00, 1 = 01, 2 = 10 e 3 = 11. A adição binária de 3 + 2 = 1, pois 3 + 2 = 11 + 10 = 01 = 1.*

A multiplicação em $GF(2^k)$ é realizada módulo $p(\alpha)$, onde $p(\alpha)$ é um polinômio em α com grau $k - 1$. Os vetores de tamanho k são transformados em polinômios em α, através do auxílio de um polinômio irredutível de grau k. A seguir é apresentada uma definição do que vem a ser um *polinômio irredutível.*

Definição 4 *Um polinômio $P(\alpha)$ de grau k, é irredutível se ele não é divisível por um polinômio de grau menor do que k (exceto o polinômio de grau zero).*

O polinômio gerador de grau 2, $p(\alpha) = \alpha^2 + \alpha + 1$ é um *polinômio irredutível*, pois não é divisível pelos polinômios $p_1(\alpha) = \alpha$ e $p_2(\alpha) = \alpha + 1$. Este polinômio e o polinômio mínimo α, são utilizados para gerar o campo $GF(2^2)$

dado pela tabela 1.3 . O exemplo 4 considera a operação de multiplicação para o campo $GF(2^2)$ ($p = 2$ e $k = 2$).

Símbolo	Polinômio	binário	octal
b^0	1	0 1	1
b^1	α	1 0	2
b^2	$\alpha^2 = \alpha + 1$	1 1	3
b^3	$\alpha^3 = \alpha^2 + \alpha = 1$	0 1	1

Tabela 1.3: Potências de p = 2 e k = 2. Onde $p(\alpha) = \alpha^2 + \alpha + 1$.

Exemplo 4: *A multiplicação de 2x3 é igual a 1, pois* $\alpha . (\alpha + 1) = \alpha^2 + \alpha = 1$. *De outra forma:* $b^1 . b^2 = b^3 = 1 \bmod p(\alpha) = 1$.

1.2 Códigos *Reed-Solomon*

Uma importante subclasse dos códigos BCH não binários é conhecida por *códigos Reed-Solomon* (RS). Esses códigos foram descobertos por *Reed* e *Solomom* em 1960. A real importância do uso dos códigos RS vem do fato de que esses códigos são capazes de corrigir os padrões contendo até t erros, através de *algoritmo simples (ao menos do ponto de vista prático) e facilmente implementável.*

1.2.1 Definições e Propriedades Básicas dos Códigos RS

Antes de definirmos os códigos RS, vejamos as definições de códigos cíclicos e códigos BCH.

Definição 5 *Um código linear (n,k) C (código com 2^k palvras-código de comprimento n) é chamado de código cíclico se todo deslocamento cíclico em C é também palavra-código em C.*

Propriedade 3 *Todo código cíclico pode ser gerado por um único $g(x)/x^n - 1$.*

Propriedade 4 *Todas as palavras-códigos c(x) de um código cíclico (n,k), podem ser expressas por c(x) = i(x) g(x), onde i(x) é o polinômio informação e g(x) é o polinômio gerador.*

Definição 6 *Um código com palavras-códigos $c(x) = \sum_{i=0}^{n-1} c_i x^i$ que tem $\alpha, \alpha^2, \cdots, \alpha^{n-k}$ como zeros é chamado de código BCH.*

A classe de códigos RS é uma importante classe de códigos cíclicos sobre *GF(p)*. Segue-se uma definição para os códigos RS.

Definição 7 *Um código RS* $\langle RS(n,k)\rangle$ *é um código cíclico de comprimento* $n = q - 1$ *sobre GF(p). O polinômio gerador desse código tem a forma* $g(x)=\prod_{i=1}^{d-1}(x-\alpha^i)$, *onde* α *é um elemento primitivo de GF(p) e d é a distância mínima de projeto do código.*

Uma importante medida para a decodificação por MV é a distância de *Hamming*. Segue-se uma definição para distância de *Hamming*.

Definição 8 *A distância de Hamming entre duas palavras-código é dada pelo número de posições que elas diferem.*

Um exemplo da aplicação da definição para a distância de *Hamming* é mostrado através do exemplo 5.

Exemplo 5: *Aplicando a definição acima e utilizando a definição de adição módulo-2, pode-se calcular a distância de Hamming entre duas palavras-código de um certo código. As palavras-código, e.g., são:*

$$\bar{v}=(1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1)\quad e\quad \bar{w}=(1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 0)$$

Portanto, a distância de Hamming entre $\bar{v}$ *e* $\bar{w}$ *é 4.*

Assim, o decodificador por MV considera a métrica de *Hamming*, i.e., dada uma palavra recebida $\bar{r}$, o decodificador decodifica-a como sendo a palavra-código $\bar{v}$ que está mais próxima a $\bar{r}$, no sentido da distância de *Hamming*.

Pelo limitante do BCH [8] (parâmetro δ denominado distância de projeto do código BCH), os códigos RS têm distância mínima maior ou igual a *d*. O número de *bits* de paridade dos códigos RS é dado por $n-k=2t$ e a distância mínima sendo $d_{min}=2t+1=n-k+1$, pois

$$d_{min} \geq n-k+1 \quad \text{Limitante BCH}$$

$$d_{min} \leq n-k+1 \quad \text{Limitante } \textit{Singleton}.$$

1.3 O papel do decodificador em um sistema de comunicações

O decodificador em um sistema de comunicações é dado pela figura 1.1.

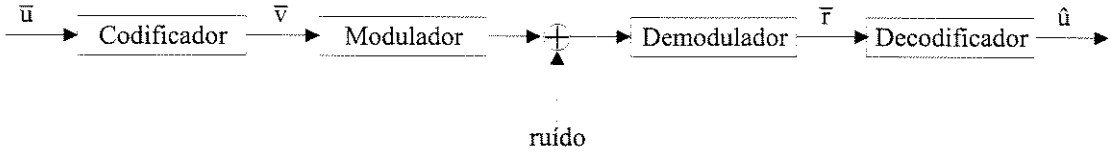

Figura 1.1: Modelo simplificado para um sistema de codificação sem memória.

O decodificador procura estimar sua saída $\hat{u}$ de uma informação enviada $\bar{u}$ baseado na sequência $\bar{r}$ recebida do demodulador. Seja $\bar{v}$ a saída no codificador. Desde que exista uma correspondência par a par entre a sequência de informação $\bar{u}$ e a palavra-código $\bar{v}$, o decodificador estima $\hat{u}$ da palavra-código real $\bar{v}$. Assim, $\hat{u} = \bar{u}$ se somente se $\hat{v} = \bar{v}$. Uma *regra de decisão* é uma estratégia para uma escolha da palavra-código $\bar{v}$ para cada possível sequência recebida $\bar{r}$. Se a palavra-código $\bar{v}$ foi transmitida, um *erro de decodificação* ocorre se somente se $\hat{v} \neq \bar{v}$. Dado que $\bar{r}$ é recebido, a *probabilidade condicional de erro do decodificador* é definida como

$$P(E|\bar{r}) \stackrel{\Delta}{=} P(\hat{v} \neq \bar{v}|\bar{r}) \qquad (1\text{-}1)$$

A *probabilidade de erro do decodificador* é então dada por

$$P(E) = \sum_{\bar{r}} P(E|\bar{r}) \cdot P(\bar{r}) \qquad (1\text{-}2)$$

$P(\bar{r})$ é independente da regra de decodificação utilizada desde que $\bar{r}$ seja produzida a *priori* para decodificação. Então, uma regra de decisão ótima [i.e., uma que minimiza $P(E)$] minimiza $P(E|\bar{r}) = P(\hat{v} \neq \bar{v}|\bar{r})$ para todo $\bar{r}$. Desde que minimizando $P(\hat{v} \neq \bar{v}|\bar{r})$ é equivalente maximizar $P(\hat{v} = \bar{v}|\bar{r})$, $P(E|\bar{r})$ é

minimizado para um dado $\bar{r}$ por escolher $\hat{v}$ como a palavra-código $\bar{v}$ que maximiza

$$P(\bar{v}|\bar{r}) = \frac{P(\bar{r}|\bar{v}) \cdot P(\bar{v})}{P(\bar{r})} \qquad (1\text{-}3)$$

isto é, $\hat{v}$ é escolhido como a palavra-código mais provável dado que $\bar{r}$ é recebido. Se todas as sequências informação são equiprováveis [i.e., $P(\bar{v})$ é a mesma para todo $\bar{v}$], maximizar $P(\bar{v}|\bar{r}) = P(\bar{r}|\bar{v}) \cdot P(\bar{v}) / P(\bar{r})$ é equivalente a maximizar $P(\bar{r}|\bar{v})$. Para um canal sem memória discreto (CMD)

$$P(\bar{r}|\bar{v}) = \prod_i P(r_i|v_i) \qquad (1\text{-}4)$$

desde que para um canal sem memória cada símbolo recebido depende apenas de um símbolo correspondente transmitido. Um decodificador que escolhe sua estimação com a maximização de $P(\bar{r}|\bar{v}) = \prod_i P(r_i|v_i)$ é chamado de *decodificador por máxima verossimilhança.*

Capítulo 2

2. Redes Neurais Artificiais

Este capítulo apresenta de maneira resumida o desenvolvimento das *Redes Neurais Artificiais* (RNA's). As redes descritas neste capítulo são baseadas nas estruturas *perceptrons* estáticas. Serão apresentados os algoritmos de alteração de pesos para o *perceptron* de uma camada única e de camadas múltiplas [19, 23, 24]. O algoritmo de camadas múltiplas é baseado na lei do gradiente descendente e na retro-propagação (*back-propagation*) de sinais através da rede. No final deste capítulo será apresentado o algoritmo para o mapeamento auto-organizado desenvolvido por *Kohonen* [16, 23, 24].

2.1 Introdução

Nestes finais de século XX, as RNA's surgem como uma nova ferramenta para tentar solucionar problemas em sua grande maioria multidisciplinares. As RNA's, baseadas biologicamente na anatomia do cérebro humano, são compostas de elementos que trabalham de maneira análoga a um neurônio biológico. Tal semelhança é enfatizada pelas características similares do funcionamento das redes neuronais biológicas, ou seja, de aprender pela experiência, de generalizar exemplos através de outros e de abstrair características.

'Embora muitos progressos notáveis nesta área tenham sido descobertos, a maioria da sociedade humana parece conhecer o cérebro humano da mesma maneira que os antigos gregos conheciam, ou seja, da maneira que *Hipócrates* observara no passado: *"... o cérebro parece uma glândula, é branco e separado em pequenas massas como as glândulas..."* [11]. Por outro lado, isso mudou um pouco e muitos progressos em neurociências foram adquiridos em nossos dias, graças, dentre outras, aos avanços em tecnologias de aquisição de imagens do corpo humano, como o surgimento dos tomógrafos e de aparelhos utilizando a técnica por ressonância magnética. Por se conhecer muito pouco sobre o cérebro humano em função de sua complexidade, ainda se está muito longe das RNA's produzirem resultados tão completos e perfeitos quanto aqueles do cérebro humano.

Recentemente foi publicado no *Journal of Comparative Neurology*, os resultados de uma pesquisa dinamarquesa para saber a quantidade de neurônios em pessoas saudáveis para compará-los com os de quem sofre com o mal de *Parkinson*. Nos 94 cadáveres analisados contou-se o número de neurônios do *neocórtex* (ligado ao raciocínio abstrato) e verificou-se que em média, os homens têm 4 bilhões de neurônios a mais que as mulheres. Foi perguntado à pesquisadora *Bente Pakkenburg*, da Universidade de Copenhague, quem era mais inteligente, o homem ou a mulher? Visto que já fora provado que não há

diferença de Quoeficiente de Inteligência (QI) entre homens e mulheres, ela respondeu categoricamente que a inteligência *não* está ligada ao número de neurônios de uma pessoa, e sim, *na quantidade e na eficiência das conexões* entre elas. Do ponto de vista computacional, as RNA's exploram justamente este fato, pois o processamento paralelo de informações das RNA's é baseado nas forças das conexões dos neurônios da rede. O conhecimento da RNA's está na definição destas conexões no final do treinamento. Por isso, requer um crescente estudo sobre novas topologias de redes, de algoritmos de aprendizagem e *hardware* mais robusto.

A figura 2.1 mostra o desenho esquemático de um neurônio biológico ou célula nervosa. Estas células nervosas podem estar agrupadas em redes com formas de árvores. O coletivo ou a rede de neurônios do ser humano está na faixa 10^{10} a 10^{11} neurônios. Vale lembrar que o coletivo de neurônios de uma população de cupins está nesta mesma ordem de grandeza, pois, uma população com um milhão de cupins com seus aproximadamente dez mil neurônios individualmente, nos dará um valor nesta mesma ordem.

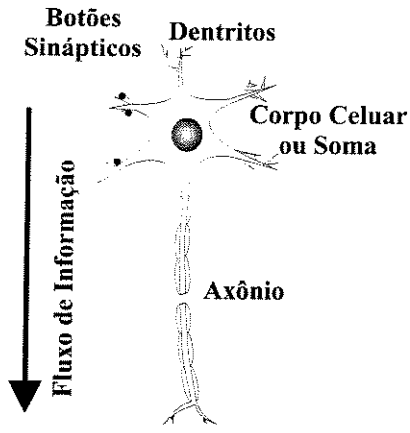

Figura 2.1: Modelo do neurônio biológico

Um modelo matemático simples para o neurônio foi proposto por *McCulloch* e *Pitts* [12]. O modelo do neurônio proposto apresentado por eles, foi baseado na soma das entradas ponderadas por seus respectivos pesos (entrada-líquida), *bias* ou limiar de ativação, aplicado à função de ativação *limiar*. A função de

ativação *limiar* vai limitar a saída, ou resposta do neurônio, para dois valores, i.e., se a entrada-líquida estiver acima ou igual ao limiar de ativação, a saída será *um* e se a entrada-líquida estiver abaixo do limiar de ativação, a saída será *zero*. O modelo proposto por *McCulloch* e *Pitts* é dado pela figura 2.2.

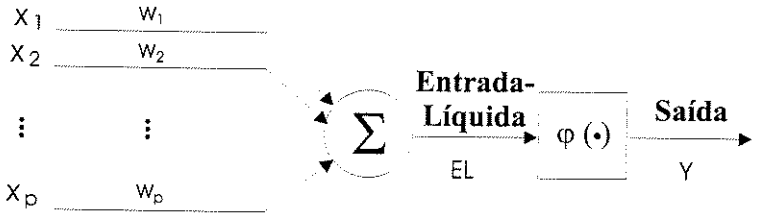

Figura 2.2: Modelo não-linear para o neurônio artificial, proposto por *McCulloch* e *Pitts*.

Matematicamente tem-se para o modelo de *McCulloch* e *Pitts* as equações dadas pela tabela 2.1.

Entrada-Líquida (EL)	$EL = \sum_{i=1}^{p} w_i x_i$
Saída dada pela função de ativação	$Y = \varphi(EL) = \begin{cases} 1; \textit{ para } EL \geq \textit{Limiar} \\ 0; \textit{ para } EL < \textit{Limiar} \end{cases}$

Tabela 2.1: Equações para o modelo do neurônio artificial de *McCulloch* e *Pitts*.

2.2 Um Breve Histórico

A história das RNA's é marcada por fases de efervescência e de descrédito, devido a resultados de pesquisas (obstáculos ou novos algoritmos) e avanços tecnológicos (máquinas mais potentes).

Em 1943, uma primeira onda de interesse emergente veio após a introdução simplificada de neurônios realizadas por *McCulloch* e *Pitts* [12], apresentado de forma bem simples no item anterior. *Rosenblatt* [13], em 1958, desenvolve os *perceptrons* realizando simulações em computador digital. Em 1960, *Widrow e Hoff* [14] contribuem com uma adaptação da teoria estatística de erros adaptada para os modelos de RNA's. *Em* 1969, *Minsky* e *Papert* [15] provaram em seu livro, que os *perceptrons* de uma única camada não conseguem separar classes que não sejam linearmente separáveis, destruindo formalmente qualquer esperança para os *perceptrons* de *Rosenblatt*. A partir daí, muitos outros pesquisadores publicam artigos na área, com destaque para os trabalhos em memória associativa auto-organizada realizados por *Teuvo Koehonen* [16], trabalhos em otimização realizados por *Hopfield* [17] e por *Hinton* e *Sejnowski* [18]. Contudo, o interesse efervescente apareceu apenas com resultados teóricos importantes em 1985, consolidados por *Rumelhart*, *Hinton* e *Williams*, com a descoberta do algoritmo de alteração dos pesos sinápticos pela retro-propagação do erro (*error back-propagation*) para os *perceptrons* com camadas múltiplas [19].

A seguir é apresentado pela tabela 2.2 um breve sumário do que foi desenvolvido em termos de RNA's, em ordem cronológica, e associada a seus respectivos pesquisadores. A tabela 2.2 é uma extensão de uma outra tabela, dada pelas anotações de sala de aula do professor Dr. *Weber Martins* (UFG/GO).

Ano	*Acontecimento*
1943	*McCulloch* e *Pitts* demostram que as RNA's têm capacidade de processar dados, com base na concepção do cérebro como um computador de elementos computacionais bem definidos: os neurônios.
1948	*Shannon* desenvolve teoria da informação, incentivando o desenvolvimento das RNA's indiretamente. *Von Neumann* utilizou as idéias de McCulloch e Pitts para fazer redes de válvulas comutativas, definindo uma lógica exata (dispositivos binários realizando operações AND/OR).
1949	*Hebb* descreve o primeiro algoritmo de aprendizagem.
1950	*Lashley* defende idéia de que a representação do conhecimento no cérebro é distribuída.
1951	*Edmonds* e Minsky constróem fisicamente a primeira RNA.
1956	*Rochester* apresentou um modelo de RNA na 1ª Conferência Internacional de Inteligência Artificial (1956), com a finalidade de verificar o comportamento desta rede, sem no entanto conseguir interpretar seus resultados.
1958	*Rosenblatt* desenvolve os perceptrons com um algoritmo de aprendizagem mais poderoso e fazendo simulação em computador digital, sendo que seus resultados bem sucedidos provocaram um grande interesse por RNA's.
1969	*Willshaw* contribui com análises matemáticas. Minsky e Papert destroem matematicamente que os perceptrons de uma única camada de *Rosenblat* eram incapazes de reconhecer classes linearmente separáveis, tornando pouco interessante o modelo apresentado. Em função disso, a concepção conexionista foi colocada em segundo plano por uma década.
1972	*Kohonen* divulga pesquisa em memória associativa auto-organizadora. *Anderson* analisa memórias associativas lineares.
1974	Paul *Werbos* descobre o algoritmo de back-propagation, mas não o divulga suficientemente.
1977	*Rumelhart* e *McClelland* fundam o grupo "Processamento Paralelo Distribuído". *Hetch-Nielsen* desenvolve dois neurocomputadores.
1982	*Hopfield* reavalia o modelo do Perceptron e demonstra a capacidade de encontrar mínimos de energia, muito utilizado em problemas de otimização.
1983	*Hinton* e *Sejnowski* realiza um treinamento de rede de forma estocástica.
1985	*Rumelhart*, *Hinton* e *Williams* demonstram o algoritmo de alteração de pesos sinápticos back-propagation.
1987	*Bart Kosko* abre a primeira conferência mundial sobre RNA's dizendo: "A Inteligência Artificial está morta. Longa vidas às Redes Neurais!"

Tabela 2.2: Breve histórico das RNA's.

2.3 Momento Atual

A simulação de RNA's em computadores digitais requer um grande número de cálculos, não deixando de ser um problema quando se acredita que propriedades relevantes somente aparecerão em grande redes. Por outro lado, a estrutura das RNA's indica para implementações físicas, i.e., máquinas dedicadas comumente chamadas por *neurocomputadores*. Os *neurocomputadores* realizariam o que uma rede neuronal biológica faz naturalmente, ou seja, os seus inúmeros processadores se comunicariam para realizar uma determinada tarefa. Entretanto, construir sinapses (elementos que devem mudar para garantir a aprendizagem) dentro de *chips* vem se constituindo em grande desafio. Surgiu então, uma corrente que tenta implementar RNA's através de memórias (usadas em computadores convencionais).

Muito provavelmente, as RNA's estão hoje onde as comunicações estavam na década de 60. Dispositivos eletro-mecânicos eram amplamente utilizados em comunicações. Atualmente não é viabilizado este tipo de tecnologia em comunicações, embora se conviva a atual tecnologia (centrais necessariamente digitais) com as mais antigas (centrais eletro-mecânicas). Principalmente, o que talvez mais intriga os pesquisadores que trabalham com RNA's é o fato de que os parâmetros da rede (tais como, número de neurônios da camada escondida, taxas de aprendizado e funções de ativação) devem ser passados à ela de forma implícita (não tendo uma fórmula fechada para o cálculo). Felizmente, alguns pesquisadores estão desenvolvendo métodos e obtendo bons resultados, embora não tão genéricos, para se conseguir RNA's que definam estes parâmetros automaticamente [25].

2.4 Modelo de Uma Rede Neural Artificial

Antes de abordar o modelo de uma RNA, faz-se necessário apresentar o funcionamento básico do sistema nervoso natural e a partir daí, conseguir explicar o modelo para uma RNA.

2.4.1 A Principal Unidade: O Neurônio

A figura 2.1 mostra o modelo esquemático do neurônio biológico. O sistema nervoso é constituído por essas células nervosas, por onde flui a informação e seu controle. Estas células recebem impulsos elétricos (provenientes de um processo eletroquímico), integram estes impulsos e os retransmitem a outras células. São essas células nervosas, ou neurônios, que servem de modelo às RNA's.

Os dendritos tem por função captar através de seus botões sinápticos os estímulos nervosos, que podem ser excitatórios ou inibitórios, de outros neurônios e conduzi-los ao corpo celular e podem, se alcançarem um certo valor de limiar, desencadear um novo estímulo transmitido ao axônio da célula constituída por uma fibra nervosa única. Por meio das ramificações do axônio, este impulso é transmitido a outros neurônios (veja o sentido deste impulso através da representação pela seta da figura 2.1).

As propriedades físicas e eletroquímicas da membrana axonal são explicadas pelo mecanismo da *bomba de sódio*, consolidados em 1952 pelos pesquisadores *Hodgkin* e *Huxley* [11]. Através de reações químicas do organismo, ocorre o aparecimento de uma carga elétrica na membrana do neurônio, sendo que a ruptura da membrana ocasiona a entrada de Na+. Esta carga elétrica cria um desequilíbrio eletrolítico e, portanto, uma polaridade na membrana (mais negativa dentro e mais positiva fora). Como o neurônio possui um nível de despolarização, se o potencial eletrônico chegar ao limiar, será ativado um potencial de ação. Esse potencial é do tipo "completo ou vazio" (comparando

com a saída dada pela função de ativação do modelo do neurônio da figura 2.2, será igual a *1* ou a *0*). Quando ativado, este potencial de ação é propagado ao longo da membrana, despolarizando pontos da mesma e criando, assim, a propagação. Esta propagação pode ser entendida como um sinal-estímulo. Se o potencial eletrônico ultrapassa o limiar (ou seja, é positivo), diz-se que ocorre a conexão (sinapse). As sinapses são os pontos de ligação que funcionam como reguladores capazes de controlar a transmissão de impulsos e regulando a sua intensidade se preciso. Durante a vida do sistema nervoso, as sinapses estão em constante modificação e formação. Quando a pessoa envelhece, muitas destas conexões são perdidas, o que ocasiona a perda de informações adquiridas durante a vida da pessoa.

A figura 2.2 mostra no modelo artificial do neurônio uma função não linear capaz de aproximar grosseiramente as características de transferência não linear de um neurônio biológico. Essa função não linear do modelo é chamada de *função de ativação* do neurônio (denotada por $\varphi(.)$). A saída do neurônio Y é processada por esta função.

A escolha da função de ativação depende de cada aplicação a ser implementada por meio de RNA's. Sobretudo, o que mais será levado em conta para se definir a função de ativação é de como as entradas e saídas serão representadas, ou seja, serão binárias ou contínuas. O próximo item apresentará algumas funções de ativação mais utilizadas.

2.4.2 Funções de Ativação

As funções de ativação amplamente utilizadas em RNA's são geralmente bem comportadas e estão muito distantes do modelo ideal para a função de ativação do neurônio biológico. Cada área de atuação de um neurônio em um ser humano, possuirá uma determinada função de ativação para a reposta aos estímulos. Por exemplo, a função de ativação para os estímulos de células nervosas de um músculo é completamente diferente da função de ativação

responsáveis pelos estímulos das células do hipocampo. Daí o motivo do uso de vários tipos de funções de ativação para uma RNA. A escolha da função (ou funções) para as respostas de uma RNA dependerá de cada aplicação.

Sendo assim, diversas funções de ativação são conhecidas. Uma lista das principais funções de ativação amplamente utilizadas em RNA's é dada pela tabela 2.3.

Nome	*Função*
Limiar (*Threshold*)	$\varphi(x) = \begin{cases} +1 \text{ se } x \geq 0 \\ -1 \text{ se } x < 0 \end{cases}$
Sigmoid	$\varphi(x) = \frac{1}{1+\exp(-ax)}$
Tangente Hiperbólica	$\varphi(x) = \frac{1-\exp(-x)}{1+\exp(-x)}$
Linear	$\varphi(x) = a \cdot x$, para $a \neq 0$
Função Limiar Lógica (*Threshold Logic Function* ou *Piecewise-Linear Function*)	$\varphi(x) = \begin{cases} 1 & \text{v} \geq \frac{1}{2} \\ x & \frac{1}{2} > \text{v} > -\frac{1}{2} \\ 0 & \text{v} \leq -\frac{1}{2} \end{cases}$

Tabela 2.3: Principais funções de ativação utilizadas em RNA's.

2.4.3 Arquitetura de Rede Neural *Feedforward* com Camada Única

As RNA's *feedforward* se organizam em Redes Neurais de camada única ou com camadas múltiplas, dependendo do número de camadas que compõem a topologia da rede.

A figura 2.3 apresenta uma arquitetura de Rede Neural mais simples existente. São as RNA's que apresentam uma única camada para os dados de entrada. Observe que os círculos cheios representam o modelo de apenas um neurônio, dado em pormenores pela figura 2.2 do modelo básico de um neurônio artificial. O número de entradas é denotado por ne. Cada entrada x_i é conectada a cada neurônio, sendo portanto apresentado como entrada à RNA o vetor $\bar{x}$. O

número de neurônios da camada única é denotado por *nns*. A conexão de cada entrada é ponderada por um peso w_{ij}. O matriz de pesos, denominado $\overline{W}$, gera o vetor de saída $\overline{Y}$ através da função de ativação $\varphi(\cdot)$ aplicada ao vetor $\overline{X} \cdot \overline{W}$, ou seja,

$$\overline{Y} = \varphi(\overline{X} \cdot \overline{W}), \qquad (2\text{-}1)$$

onde $\overline{X}$ é o vetor linha de entrada de dimensão *ne* e $\overline{Y}$ é o vetor linha de saída de dimensão *nns*.

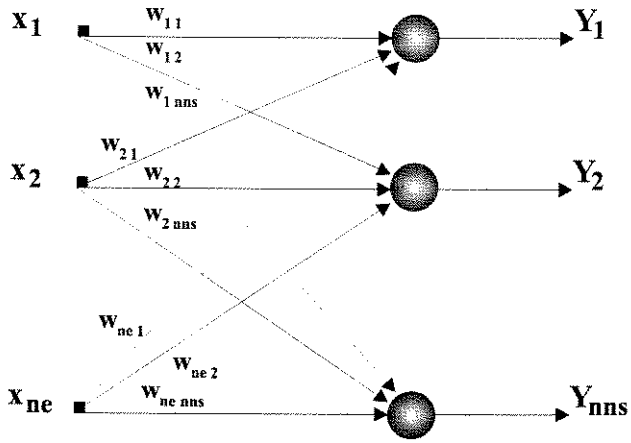

Figura 2.3: Arquitetura de uma RNA *feedforward* com uma única camada.

2.4.4 Arquitetura de Rede Neural *Feedforward* com Camada Múltipla

Embora a rede com camadas múltiplas seja uma rede um pouco mais complexa é a rede que apresenta melhores resultados e portanto, maiores aplicações.

A arquitetura de uma RNA com camada múltipla possibilita resultados superiores em relação às redes com camada única, e tem despertado muito interesse no que se refere à novos algoritmos de treinamento, onde reside os principais problemas desta topologia. Com o aparecimento de novos algoritmos

de treinamento, esta topologia, voltou a ser valorizada e atualmente é uma das arquiteturas mais utilizadas.

Uma RNA com camadas múltiplas pode ser observada como se fosse uma rede compostas de várias redes de uma única camada, onde os neurônios de camadas adjacentes estão conectados. A figura 2.4 Ilustra uma RNA com uma camada escondida.

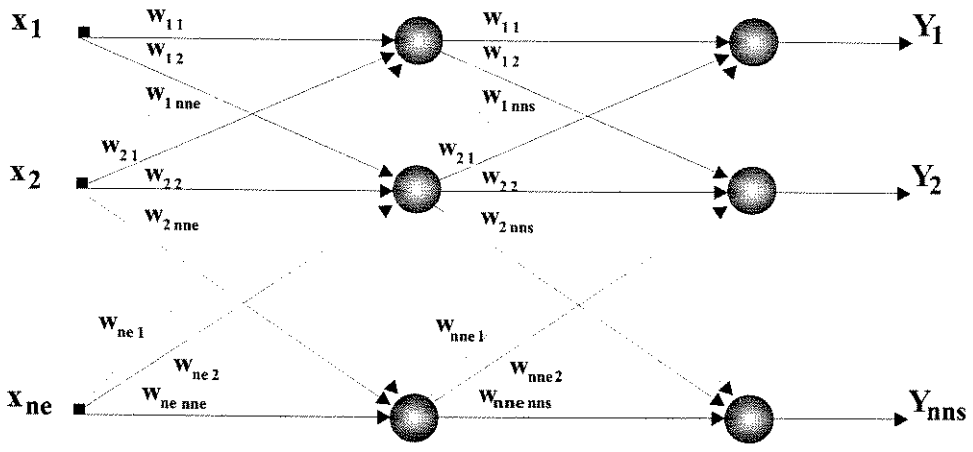

Figura 2.4: Arquitetura de uma RNA *feedforward* com uma camada escondida.

A arquitetura de RNA's com camadas múltiplas possibilitou a resolução de problemas cujas classes fossem não separáveis (item 2.5.2). Daí o surgimento recentemente de grande interesse em seu estudo e novas aplicações em problemas variados.

2.4.5 Métodos de Aprendizagem de uma RNA

O treinamento de uma RNA é um fator determinante para um bom desempenho na resolução de um problema por meio de Redes Neurais. Se uma rede não estiver bem treinada, ela não poderá fazer coisa alguma que nos seja de interesse.

Há pesquisadores como *Hopfield* [17], que acredita que a questão de treinar uma rede não está em ensiná-la, mas sim, em construir convenientemente. O conhecimento durante o funcionamento é embutido na fase de projeto. Não existe fase de treinamento em tais redes. O projetista já constrói a rede de acordo com suas necessidades. Basicamente, ele modela uma função "energia" que a rede, quando em operação, por si mesma aponta para os pontos de mínimo ou de máximo desejado.

A maioria, porém, preferem desenvolver algoritmos ou técnicas para se conseguir que a rede possa aprender. Não existe ainda nenhum método de aprendizagem infalível capaz de solucionar qualquer tipo de problema. No entanto, consegue-se resultados satisfatórios em muitos casos. Por exemplo, se conseguirmos uma rede que consiga a mesma percepção auditiva de um papagaio falador que reconhecesse os sons de seu dono, já teríamos sobrepujado muitíssimo, o melhor desempenho obtido por técnicas tradicionais.

Dois grande métodos de treinamento se destacam: o treinamento supervisionado e o treinamento não supervisionado. No treinamento supervisionado, é necessário um instrutor para que a rede possa aprender, ou seja, é apresentado uma entrada (estímulo) à rede e uma saída desejada (alvo) para a mesma. Seria como ensinar um papagaio a falar "CURRUPACO, DÁ O PÉ LOURO!". Primeiro repetimos a fala desejada (estímulo) ao animal por várias vezes até que o animal repita corretamente (alvo). Por outro lado, no treinamento não supervisionado, o próprio ambiente pode auxiliar o aprendiz em seu esforço (em sua adaptação). No caso do papagaio, seria os sons que o animal em seu esforço aprenderia em seu *habitat* natural, como por exemplo, se sua mãe lhe oferece uma refeição em sua boca no ninho, talvez o simples assobio de sua mãe distinguiria ou não a sua chegada, e é neste momento que o ambiente se encarrega de dar os últimos respaldos em seu sistema de percepção.

Dentre as redes mais utilizadas na atualidade estão as redes de camadas múltiplas constituídas de *perceptrons* com treinamento supervisionado. No

entanto, primeiramente é importante a verificação do funcionamento da rede *perceptron* de uma única camada.

2.5 Rede Neural Utilizando Perceptron

2.5.1 O *Perceptron* de Camada Única

Em meados dos anos 50, *Frank Rosenblatt* [13], do *Cornell Aeronautical Laboratories*, idealizou um modelo computacional para a retina para ser empregado no reconhecimento de padrões ópticos. Este modelo foi denominado *perceptron*. A meta pretendida era muito ambiciosa, pois buscava-se a interpretação e a modelagem dos mecanismos de reconhecimento de padrões do sistema visual.

A idéia essencial em apoio ao *perceptron* estava calcada no fato de que a retina contém vários sensores de luminosidade, os bastonetes, arranjados matricialmente. As saídas destes sensores estão conectadas a uma rede neuronal biológica capaz de reconhecer tipos particulares de padrões, ou seja, a resposta produzida é uma função de uma lógica de limiar, em que nenhuma saída é fornecida até que um determinado nível ou tipo de entrada ocorram. Estas características foram inspiradas em observações que mostraram que um neurônio não dispara até a ocorrência de um determinado balanceamento da atividade de entrada da rede neuronal biológica, onde algum limiar deva ser excedido.

Rosemblatt abordou o problema das aptidões dos *perceptrons* quando os padrões de entrada são binários. Neste caso, cada padrão de entrada é um vértice de um hipercubo, que possui dimensão igual ao número de entradas. Se N for o número de entradas, então existem 2^N padrões de entrada possíveis. Ele também provou que se padrões de duas classes de entrada podem ser separados por um *perceptron*, então o processo de convergência terá sucesso e será determinada

uma solução. Tais conjuntos de padrões são denominados *linearmente separáveis.*

A rede *perceptron* é capaz de aprender a classificar apenas os padrões linearmente separáveis, não podendo executar um *ou exclusivo*, por exemplo. Por isto, em meados dos anos 60, *Marvin Minsky* e *Seymour Papert* iniciaram um trabalho com a finalidade de avaliar o *perceptron.* O livro publicado em 1969 [15] foi fruto de uma análise matemática detalhada dos *perceptrons* de *Rosenblatt.* O principal resultado apresentado no livro era a conclusão de que o *perceptron* de uma camada e a computação neural não eram assuntos de interesse, sendo necessários ainda muitos estudos para a sua viabilização.

O *perceptron* de uma única camada com apenas um neurônio pode ser representado pela figura 2.5. Pode-se notar que existe um único neurônio com p entradas e seu limiar de ativação Θ, com a sua saída y. Internamente o neurônio pode ser representado pelo grafo de sinais dado pela figura 2.6. Do modelo da figura 2.6 pode ser encontrado o valor de v_j dado pela combinação linear das p entradas e pelo seu limiar de ativação Θ, ponderados pelos respectivos pesos, ou seja:

$$v = \sum_{i=1}^{p} w_i x_i - \Theta$$

Observe que o valor da saída y depende da função de ativação $\varphi(v_j)$ do neurônio. Esta função de ativação pode ser a função sinal, a função tangente hiperbólica ou qualquer outra função não linear apropriada conforme visot na seção 2.4.2.

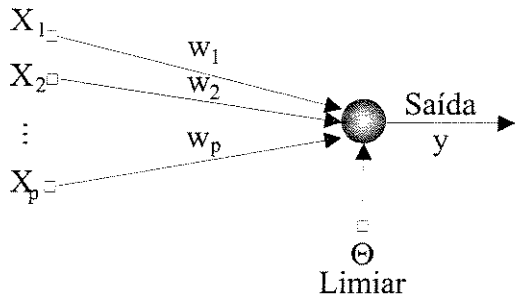

Figura 2.5: O *perceptron* de uma única camada com um único neurônio.

Fonte: *Haykin* [23]

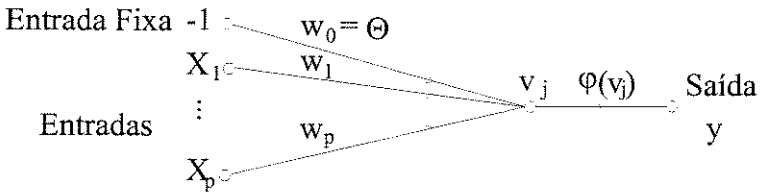

Figura 2.6: Grafo de sinal equivalente do *perceptron.*

Fonte: *Haykin* [23]

2.5.2 O *Perceptron* e a Classificação de Padrões Linearmente Separáveis

O propósito do *perceptron* de uma única camada com um único neurônio é de classificar duas classes ζ_1 e ζ_2 que sejam linearmente separáveis. Os padrões da classe ζ_1 (entradas $x_1, x_2, ..., x_p$) são apresentados ao algoritmo de treinamento de tal forma que o *perceptron* responda com *+1* na sua saída e caso os padrões sejam da classe ζ_2 com *-1*. Isto leva a duas regiões de decisão que são separadas por um *hiperplano* dado por:

$$\sum_{i=1}^{p} w_i x_i - \Theta = 0$$

No caso de duas variáveis, o *hiperplano* será uma reta em $\mathbf{R}^2$. E caso sejam duas variáveis de entrada, o *hiperplano* será um plano em $\mathbf{R}^3$.

A figura 2.7 mostra uma de par de classes não linearmente separáveis, e um par de classes linearmente separáveis. Note que se as duas classes tiverem valores comuns, ou seja, a interseção das duas classes não seja nula (classes não linearmente separáveis), o *perceptron* apresentará problemas de convergência no algoritmo de treinamento.

(a) (b)

Figura 2.7: (a) Um par de padrões linearmente separáveis. (b) Um par de padrões não linearmente separáveis. Fonte: *Haykin* [23].

2.5.3 O Algoritmo de Treinamento para o *Perceptron*

O algoritmo do *perceptron* adaptado por *Haykin* [23] é dado pela tabela 2.4. Pode ser definida uma nova variável baseada no número de interações *n*. Trata-as do número de épocas do treinamento. Uma época é definida como sendo a apresentação de todos os padrões de treinamento ao *perceptron*, ou seja, uma época é obtida quando *n* for igual ao número de padrões de treinamento. Para uma certa época, se para todos os padrões de treinamento *x(n)* a saída *y* do *perceptron* estiver correspondendo ao valor desejado *d*, então diz-se que o *perceptron* convergiu e os pesos *w* obtidos no treinamento poderão ser utilizados para a classificação numa fase chamada teste.

Passo 1 ***Inicialização:*** *w(0)=0*

Passo 2 ***Ativação:*** no instante aplique x(n) e d(n)

Passo 3 ***Cálculo da Resposta Observada:***

$$y(n) = Sinal[w^T(n) \cdot x(n)],$$

onde a função *Sinal(x)* é dada por:

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

Passo 4 ***Alteração dos Pesos:***

$$w(n+1) = w(n) + \eta \cdot [d(n) - y(n)] \cdot x(n), \text{ onde}$$

onde η é uma constante denominada *taxa de aprendizado* e

$$d(n) = \begin{cases} +1; se\ x(n) \in \zeta_1 \\ -1;\ se\ x(n) \in \zeta_2 \end{cases}.$$

Passo 5 Se o número de interações *n* for menor ou igual ao número de padrões de treinamento faça:

$$(n = n+1)$$ e volte ao passo 2

Senão vá ao passo 6.

Passo 6: Se o *perceptron* estiver respondendo corretamente para todos os padrões de treinamento, o algoritmo convergiu e vá ao passo 7. Caso contrário, volte ao passo 2;

Passo 7 ***Fim.***

Tabela 2.4: Algoritmo do *perceptron.* Fonte: *Haykin* [23].

A mudança do valor dos pesos é dada pela medida de desempenho $\hat{J}(n)$ expressa por:

$$\hat{J}(n) = -e(n) \cdot v(n) = -[d(n) - y(n)] \cdot v(n)$$

Multiplicando a expressão acima pelo gradiente em relação a w vem que:

$$\nabla_w \cdot \hat{J}(n) = -[d(n) - y(n)] \cdot \nabla_w \cdot v(n)$$

E pela regra de correção de erros [14], o incremento dos pesos fica sendo dado por:

$$\Delta w(n) = -\eta \cdot \nabla_w \cdot \hat{J}(n) = \eta \cdot [d(n) - y(n)] \cdot x(n)$$

Onde:

$x(n) = [-1, x_1(n), x_2(n), \ldots, x_p(n)]^T$: Vetor de entradas

$w(n) = [\Theta(n), w_1(n), w_2(n), \ldots, w_p(n)]^T$: Vetor pesos

$y(n)$: Resposta observada

$d(n)$: Resposta desejada

$v(n) = w^T(n)\, x(n)$: cominação linear de $w(n)$ e $x(n)$

η : Taxa de aprendizado (constante positiva menor do que *1*)

A seguir é apresentado o *perceptron* com camadas múltiplas e o seu algoritmo de alteração dos pesos sinápticos, proposto inicialmente por *Werbos* e consolidado por *Rumelhart*.

2.5.4 O *Perceptron* de Camadas Múltiplas

O *perceptron* de camadas múltiplas são redes sem realimentação com uma ou mais camadas de nós, entre os nós de entrada e saída. Estas camadas adicionais

contém unidades ou nós escondidos que não são conectados diretamente aos nós de entrada e saída. A figura 2.8 ilustra uma rede de *perceptron* com três camadas escondidas.

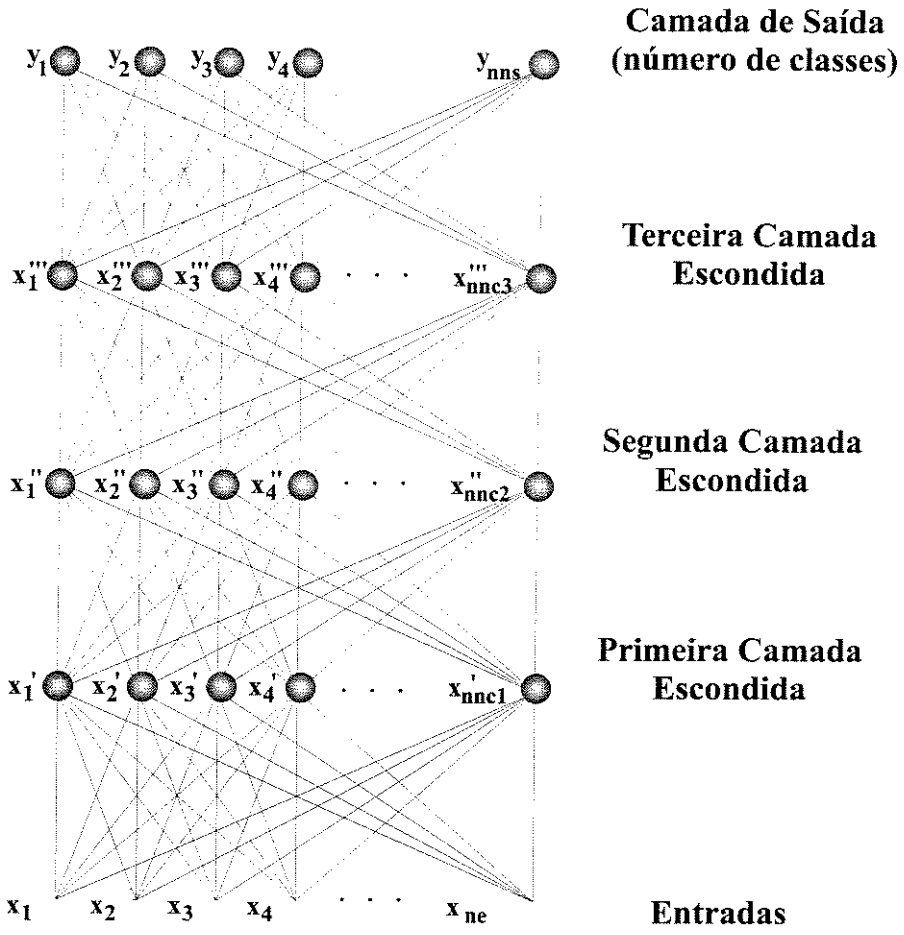

Figura 2.8: Exemplo de uma RNA com três camadas escondidas, onde *ne* é o valor do número de entradas da rede, *nnc1*, *nnc2* e *nnc3* o número de neurônios das respectivas camadas escondidas e *nns* o número de neurônios da camada de saída.

Embora a convergência dos algoritmos para este tipo de rede não possa ser provada, como no caso do algoritmo de camada simples, eles tem-se mostrado úteis na solução de muitos problemas devido a capacidade de formação de novas regiões de decisão.

O objetivo básico do algoritmo é de minimizar o erro quadrático médio entre a saída atual e a desejada. O algoritmo assume um conjunto inicial de pesos aleatórios no início do treinamento e durante o treinamento o algoritmo busca o valor correto dos pesos adaptados ao problema a ser resolvido.

Tipicamente, uma rede *perceptron* com camadas múltiplas possui uma camada de entrada, uma camada de saída e no mínimo uma camada interna oculta. Não existe limite teórico quanto ao número de camadas internas, mas na maioria das aplicações utiliza-se uma ou duas. Por exemplo, *Cybenko* demonstrou rigorosamente que apenas uma única camada escondida seria suficiente para aproximar funções contínuas através de RNA's [21].

Durante a fase de treinamento a informação se propaga da camada de entrada para a camada de saída e o erro no sentido contrário. A rede *perceptron* com camadas múltiplas assume que todos os neurônios e conexões têm influência na resposta errada, sendo o erro propagado reversamente através das conexões das camadas anteriores. Este processo é repetido até que o erro seja minimizado. A rede aprende com os exemplos apresentados durante a *fase de treinamento* e é capaz de generalizar diante de uma entrada desconhecida na *fase de classificação*. Portanto, as RNA's, na maioria das vezes, são mais vantajosas que os métodos estatísticos por não necessitarem de conhecimento a *priori* das distribuições estatísticas das classes, sendo este conhecimento incorporado através do treinamento.

A denominação *back-propagation* deriva-se deste processo de distribuição de influência dos erros. O algoritmo *back-propagation* tem como base a regra delta ou do gradiente descendente, que tem como principal objetivo minimizar o erro médio quadrático entre a resposta obtida e a desejada para um determinado padrão. Logo a seguir é apresentado a lei de aprendizado baseada no gradiente descendente para uma rede *perceptron* (algoritmo *back-propagation*) com uma única camada escondida, utilizada nas simulações deste trabalho.

2.5.5 A Lei de Aprendizado Baseada no Gradiente Descendente

A lei de aprendizado baseada no gradiente descendente é provada na maioria dos livros de RNA's [23].

O algoritmo de *back-propagation*, em termos bem simples minimiza o erro quadrático médio, ou seja minimiza a função:

$$\varepsilon_{av} = \frac{1}{N}\sum_{n=1}^{N} \varepsilon(n)$$

onde N é o número de exemplos de treinamento e $\varepsilon(n)$ é a soma instantânea dos erros quadráticos de todos os neurônios na camada de saída. Portanto, o objetivo do processo de aprendizagem do algoritmo é de ajustar os parâmetros da rede de forma a minimizar ε_{av} (medida de desempenho de aprendizado do conjunto de treinamento). Fazendo algumas aproximações usadas para minimizar $\varepsilon(n)$, isto é, minimizando padrão a padrão, obtemos um algoritmo similar ao adaptado por *Fausett* [24] dado pela tabela 2.5, para o ajuste dos pesos sinápticos de rede *perceptron* com uma única camada escondida (algoritmo *back-propagation)*.

Algoritmo básico *back-propagation*

passo 0. Inicialize todos os pesos:

(Fixe-os com valores aleatórios pequenos)

passo 1. Enquanto a condição de parada for falsa, faça os passos 2-9.

passo 2. Para cada par de treinamento, faça os passos 3-8.

Alimentação progressiva (*feedforward*)

passo 3. Para unidade de entrada (X_i, i = 1, ... , n) receba o sinal de entrada x_i e propague este sinal para todas as unidades na camada seguinte (as unidades escondidas).

passo 4. Para cada unidade de entrada (Z_j, j = 1, ..., p) some o sinal de entrada ponderado pelo seu respectivo peso sináptico,

$$z\_in_j = v_{0j} + \Sigma\ (1 \text{ até } n)\ x_i\ v_{ij},$$

aplique a função de ativação para elas para calcular o sinal de cada saída,

$$z_j = f\ (z\_in_j).$$

e envie este sinal para todas as unidades da camada seguinte (unidades de saída).

passo 5. Para cada unidade de saída (Y_k, k = 1, ..., m) some o sinal entrada,

$$y\_in_k = w_{0j} + \Sigma\ (1 \text{ até } p)\ z_j\ w_{jk},$$

aplique a função de ativação para calcular o sinal de cada saída,

$$y_k = f\ (y\_in_k).$$

Retro-propagação do erro (*back-propagation of error*)

Passo 6. Para cada unidade de saída (Y_k, k = 1, ..., m) receba um padrão alvo correspondendo ao padrão em treinamento, calcule o erro informação,

$$\delta_k = (\ t_k - y_k\ )\ f\ '\ (y\_in_k),$$

calcule o termo para correção dos pesos (usado para

modificar w_{jk} posteriormente),

$$\Delta w_{jk} = \alpha \, \delta_k \, z_j,$$

calcule o termo para correção do limiar de ativação (usado para atualizar w_{0j} posteriormente),

$$\Delta w_{0j} = \alpha \, \delta_k$$

e envie δ_k para as unidades da camada anterior.

Passo 7. Para cada unidade escondida (Z_j, j = 1, ..., p) some os deltas de entrada (da unidade da camada seguinte),

$$\delta\_in_j = \Sigma \, (k = 1 \text{ até } m) \, \delta_k \, w_{jk}$$

multiplique pela derivada da função de ativação para calcular o termo de informação do erro,

$$\delta_j = \delta\_in_j \, f' \, (z\_in_j),$$

calcule o termo para correção dos pesos (usado para atualizar v_{ij} posteriormente),

$$\Delta v_{ij} = \alpha \, \delta_j \, x_i,$$

e calcule o termo para correção do limiar de ativação (usado para atualizar v_{0j} posteriormente),

$$\Delta v_{0j} = \alpha \, \delta_j.$$

Passo 8. Para cada unidade de saída (Y_k, k = 1, ..., m) atualize o limiar de ativação e os pesos (j = 0, ..., p):

$$w_{jk} \text{ (novo)} = w_{jk} \text{ (velho)} + \Delta w_{jk}.$$

Para cada unidade escondida (Y_k, k = 1, ..., m) atualize o limiar de ativação e os pesos (i = 0, ..., n):

$$v_{jk} \text{ (novo)} = v_{jk} \text{ (velho)} + \Delta v_{jk}.$$

Passo 9. Teste a condição de parada.

Tabela 2.5: Passos para o algoritmo *back-propagation* para uma rede com *n* neurônios na camada de entrada, com *p* neurônios na camada oculta e *m* neurônios na camada de saída, onde α é a taxa de aprendizado da RNA. Fonte: *Fausett* [24].

O algoritmo descrito pela tabela 2.5 é facilmente implementado em qualquer linguagem de programação estruturada ou interpretada, como por exemplo, linguagem *C*, *Pascal*, *Matlab* e *Basic*. No entanto, melhores desempenhos no que diz respeito à velocidade de convergência do algoritmo podem ser encontradas independentemente da linguagem de programação utilizada. Isto é conseguido com um acréscimo de um parâmetro μ denominado de *momento*. A nova regra de alteração dos pesos sinápticos consiste numa combinação do gradiente em uma determinada direção e o gradiente previsto. Esta é uma modificação do gradiente descendente.

As novas equações para os incrementos dos pesos sinápticos para o algoritmo *back-propagation* com *momento* são:

$$\Delta w_{jk}(t+1) = \alpha\, \delta_k\, z_j + \mu\, \Delta w_{jk}(t) \text{ e} \qquad (2\text{-}2)$$

$$\Delta w_{ij}(t+1) = \alpha\, \delta_i\, x_i + \mu\, \Delta v_{ij}(t) \qquad (2\text{-}3)$$

onde o parâmetro *momento* μ é limitado à faixa de valores entre 0 e 1, excluindo os extremos finais. Os parâmetros *t*, *(t-1)* e *(t+1)* são os parâmetros da variável considerada iteração atual, anterior e posterior, respectivamente. A convergência é rápida quando o momento resultante da combinação estiver no caminho correto (levando à convergência).

Muitas outras modificações peculiares poderão ser analisadas pelo leitor se desejar. Como por exemplo, a inclusão de um controlador *fuzzy* para o treinamento *back-propgation* proposto por *Lee* [22] ou uma regra relativamente simples para o algoritmo de treinamento *back-propagation* dado pela regra *delta-bar-delta*, proposto por *Jacobs* [20].

2.6 Redes Neurais Baseadas em Competição

2.6.1 Mapas auto-organizados de *Kohonen*

Muitas Redes Neurais usam a idéia de competição associado aos neurônios, para aumentar o contraste nas ativações dos neurônios. Por exemplo, apresenta-se à rede um determinado padrão. Um dos neurônios da rede vai ser o que possui o maior potencial de ativação e será chamado de *neurônio vencedor*. A região próxima a este *neurônio vencedor* no mapa de *Kohonen* será responsável pela classe a qual pertence o padrão. O algoritmo de alteração de pesos, vai "reforçando" os valores dos pesos para cada *neurônio vencedor* e para os seus vizinhos. Quando as regiões (conjunto de neurônios) responsáveis por cada classe estiverem bem definidas, o treinamento é terminado.

A figura 2.9 mostra a arquitetura bidimensional para um mapa de *Kohonen,* para um conjunto hipotético de padrões. Note que após o treinamento, a rede de *Kohonen* convergiu para a classificação de quatro classes distintas.

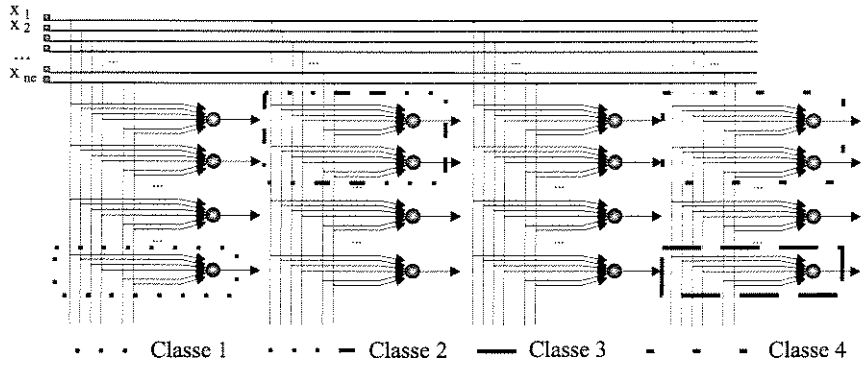

Figura 2.9: Mapa bidimensional de *Kohonen,* com uma solução hipotética encontrada após o treinamento.

Neste trabalho, foi utilizado uma rede de *Kohonen* unidimensional denominada *rede paramétrica de Kohonen.* No próximo item será apresentado para esta rede, a arquitetura neural e o algoritmo de treinamento.

2.6.2 Rede Paramétrica de *Kohonen*

A arquitetura para uma rede paramétrica de *Kohonen* é dada pela figura 2.10.

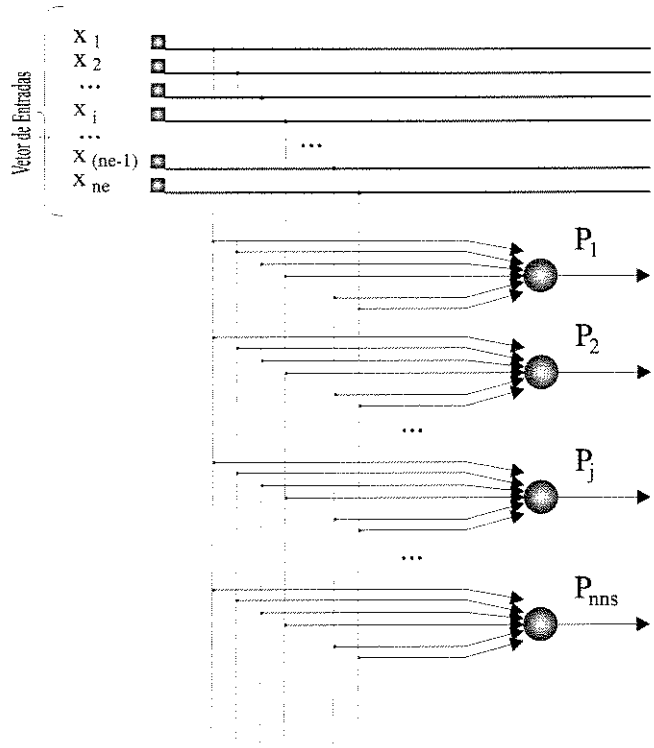

Figura 2.10: Rede paramétrica de *Kohonen* com *nns* neurônios.

O algoritmo para o treinamento da rede paramétrica de *Kohonen* foi adaptado do algoritmo apresentado por *Fausett* [24]. O novo algoritmo sugerido apresenta uma inovação com respeito ao número de neurônios da rede [35]. Durante o treinamento, o número de neurônios da rede terá um valor menor ou igual ao do valor inicial. "Podas" ou eliminação de neurônios vão ocorrer se uma determinada condição de eliminação for satisfeita. Porém, a eliminação de neurônios não poderá ser realizada se o número mínimo de neurônios da rede for atingido. Após uma determinada condição de redução para o raio de vizinhança for atingido, uma redução no valor do raio de vizinhança será efetuada, não podendo ser inferior ao valor unitário. O algoritmo para a rede paramétrica de *Kohonen* é apresentado pela tabela 2.6.

Algoritmo para a rede paramétrica de *Kohonen*

passo 0. Inicialize todos os pesos:

(Fixe-os com valores aleatórios pequenos)

Inicialize o número de entradas da rede, o raio de vizinhança inicial, o número de neurônios inicial e final, a taxa de aprendizado.

passo 1. Enquanto condição de parada for falsa, faça os passos 2-9.

passo 2. Para cada vetor de entrada x (padrão de treinamento), faça os passos 3-5.

passo 3. Para cada j , calcule:

$$D(j) = \sum_i (w_{ij} - x_i)^2$$

passo 4. Encontre o índice J tal que $D(J)$ seja mínimo.

passo 5. Para todas as unidades j e para os vizinhos de J, e para todo i:

w_{ij} (novo) = w_{ij} (velho) + α [x_i - w_{ij} (velho)].

passo 6. Altere o valor da taxa de aprendizado:

$$\alpha(t+1) = \text{taxa} \cdot \alpha(t)$$

(Valor para *taxa* próximo de 0.5)

passo 7. Se a condição de redução do raio de vizinhança foi atingida, reduza o raio de vizinhança.

passo 8. Se a condição de eliminação do número de neurônios foi atingida, reduza o número de neurônios, ou seja, realize a "poda" de neurônios.

passo 9. Testa a condição de parada.

Tabela 2.6: Passos para o algoritmo da rede paramétrica de *Kohonen*. O peso w_{ij} corresponde à conexão entre a *i-ésima* entrada e o *j-ésimo* neurônio. A *i-ésima* entrada é denotada por x_i. Fonte: *Fausett* [24] e adaptado com a introdução do passo 8.

Capítulo 3

3. Sistemas FH-CDMA (*Frequency Hopping - Code Division Multiple Access*)

Este capítulo apresenta uma introdução aos sistemas FH-CDMA. Na primeira parte deste capítulo é apresentado o detector não coerente FH-FSK (*Frequency Hopping - Frequency Shifit Keying*) proposto pelo laboratório da *Bell,* como uma das possíveis técnicas de espalhamento espectral de um sinal digital utilizando a modulação FSK [26]. Numa segunda parte deste capítulo, os sistemas de endereçamento propostos por *Einarsson* [9, 27] são apresentados como proposição para as matrizes tempo-freqüência da sequência de salto em freqüência (FH). Também é apresentada uma generalização para os sistemas de duas mensagens de *Einarsson*, realizada por *Vajda* [28]. Em seguida os limitantes de união das probabilidades de erro dos sistemas propostos por *Einarsson* são dados de forma resumida. No fim deste capítulo, é apresentado outros sistemas de recepção multi-usuários [29, 30].

3.1 O Transmissor e o Receptor de um Sistema FSK-FHMA

3.1.1 O Transmissor Multi-nível FH-CDMA

A figura 3.1 mostra o diagrama de blocos do transmissor multi-nível FH-CDMA. O usuário gera a sequência *Q-ária* de mensagem, $\bar{x}$, de comprimento L. Um endereço $\bar{a}$, também de comprimento L, é adicionado à mensagem do usuário (componente a componente). A palavra-código resultante

$$\bar{y} = \bar{a} + \bar{x} \qquad (3\text{-}1)$$

é enviada através do canal utilizando-se uma sequência de L tons FSK. Esta sequência FSK pode ser representada como entradas de uma matriz de Q linhas e L colunas [26]. A duração de cada um destes sinais é o tempo de *chip* (cada entrada da matriz) de duração T segundos.

Como exemplo, para um sistema com $Q = 7$ e $L = 3$, a matriz de transmissão da figura 3.1, corresponde a mensagem do usuário igual a $\bar{x} = (3, 3, 3)$ e o endereço do usuário igual a $\bar{a} = (4, 5, 6)$, resultando na palvra-código $\bar{y} = \bar{a} + \bar{x} = (0, 1, 2)$. Note que neste caso, a palavra-código de endereço $\bar{a}$ foi deslocada ciclicamente de 3 passos (operação de adição módulo-7).

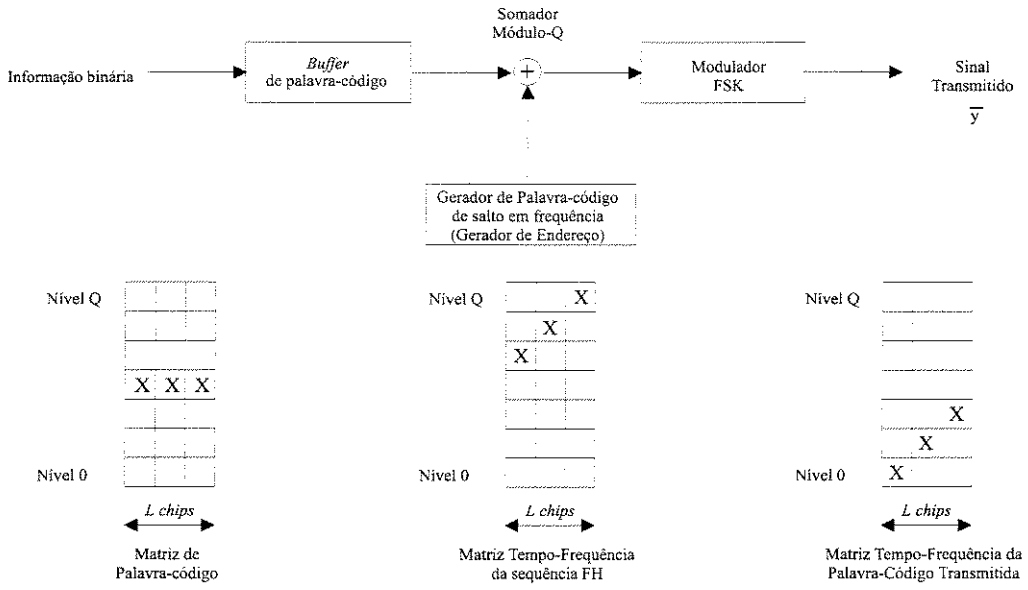

Figura 3.1: Diagrama de Blocos do Transmissor Multi-nível FH-CDMA.

A seqüência de tons FSK transmitida pode ser afetada por $M < Q$ sequências de outros usuários interferentes e também por desvanecimentos e ruídos. Neste trabalho, considerou-se apenas a interferência de outros usuários na sequência de tons transmitida.

Considere agora o caso em $Q = L$. Se for utilizado um endereçamento apropriado (endereçamento ortogonal) para as matrizes de transmissão, não ocorrerá erros na decodificação dos Q possíveis usuários ativos, pois os M usuários não causarão interferência entre si. No entanto, se isso fosse construído, não estaríamos reduzindo a faixa espectral. Assim, atribui-se sempre$L < Q$, reduzindo a faixa espectral utilizada. Evidentemente, o número de usuários "alocados" numa mesma faixa espectral será bem maior do que se cada usuário fosse "alocado" em uma faixa dedicada. Mas também, a probabilidade de erro de mensagem para um determinado usuário irá aumentar.

3.1.2 O Receptor Multi-nível FH-CDMA

O receptor consiste num *frequency-dehopper*, que extrai o endereço da sequência de tons recebida, seguido de Q detectores de energia. Durante L intervalos de *chips*, as saídas dos detectores são comparadas com um limiar e é feita uma decisão relativa à presença ou ausência da freqüência correspondente. A figura 3.2 mostra o diagrama de blocos do receptor multi-nível FH-CDMA.

Um receptor de lógica majoritária (LM), maximiza

$$\max_{m} p(\bar{r}|\bar{y}_m),\ m = 0,\ 1,\ 2,\ \cdots,\ M-1 \qquad (3\text{-}2)$$

onde y_m, $m = 0,\ 1,\ 2,\ \cdots,\ M-1$, são todas as M possíveis matrizes transmitidas pelo usuário que está sendo decodificado.

Portanto, o receptor de máxima verossimilhança, pode ser implementado através de um receptor de LM: escolha $\bar{y}$ como a palavra-código transmitida (matriz) se ele possui mais coincidências com a matriz recebida do que $\bar{y}^{*} (\neq \bar{y})$ [26].

Como exemplo, para um sistema com $Q = 7$ e $L = 3$, um usuário interferente contribui com a palavra-código resultante $\bar{y}_1 = (\ 3,\ 0,\ 2\ )$, que interfere em apenas uma posição na palavra-código de um segundo usuário com $\bar{y}_2 = (\ 0,\ 1,\ 2\ )$. Para decodificação da mensagem transmitida do usuário de endereço $\bar{a} = (\ 4,\ 5,\ 6\ )$, basta subtrair $\bar{y}_1$ e $\bar{y}_2$ do endereço $\bar{a}$ (operação de subtração módulo Q). Isto resulta em uma nova matriz $\bar{y}^{*}$. Aplicando a regra de decisão por lógica majoritária verificamos que a mensagem transmitida pelo usuário é $\bar{x} = (\ 3\ ,\ 3,\ 3\ )$. O efeito da interferência entre usuários pode ser visualizado na matriz de recepção da figura 3.2 (efeito de desespalhamento na matriz $\bar{y}^{*}$).

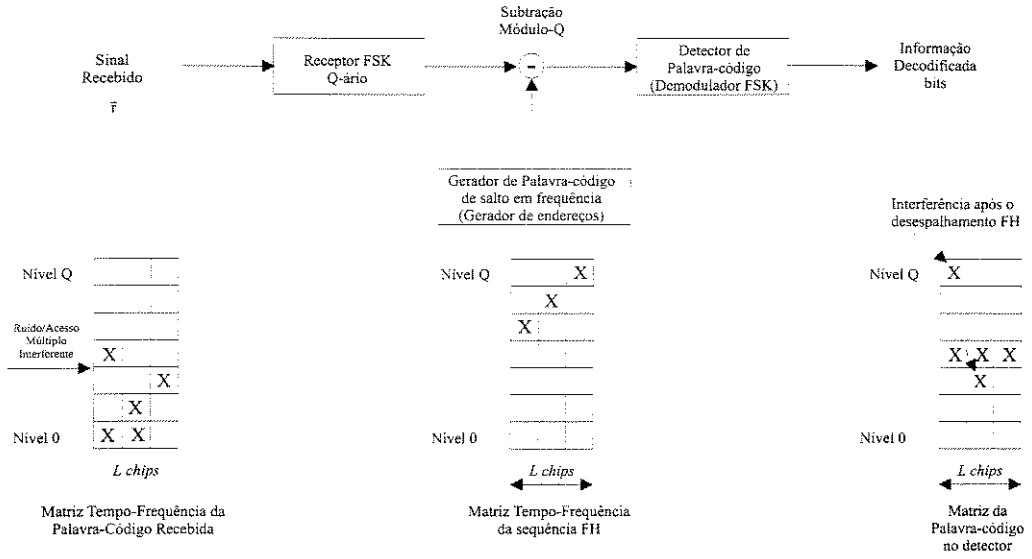

Figura 3.2: Diagrama de Blocos do Receptor Multi-nível FH-CDMA.

3.2 Transmissão e Recepção para os Sistemas FSK-FHMA Segundo o tipo de Endereçamento

Duas propostas mais apropriadas de endereços foram apresentadas posteriormente. *Einarsson* [9, 27] apresenta um sistema de endereçamento para uma ou duas mensagens. *Vadja* [28] apresenta um sistema de endereçamento para um número de mensagens qualquer. A seguir são apresentados os estudos realizados por esses dois pesquisadores.

3.2.1 Matriz de Transmissão através do Endereçamento dado por *Einarsson*

Einarsson sugere que o m-ésimo usuário gere uma sequência Q-*ária* de mensagem, $\bar{x}_m$, de comprimento L ($L < Q$). Um endereço $\bar{a}_m$, também de comprimento L, é adicionado à mensagem do usuário (componente a componente). Todas as operações binárias consideradas a partir daqui são sobre um campo de *Galois* $GF(Q)$. A palavra-código resultante

$$\bar{y}_m = \bar{a}_m + \bar{x}_m \qquad (3\text{-}3)$$

é enviada através do canal utilizando-se uma sequência de L tons FSK. A duração de cada um destes sinais é o tempo de *chip*. A equação (3-3) difere da equação (3-1), no sentido de que os endereços $\bar{a}$ da equação (3-1) pode ser qualquer sequência. Por outro lado, a equação (3-3) sugere um endereçamento que minimize a probabilidade de erro.

Seja $\bar{\beta} = (1, \beta, \beta^2, \ldots, \beta^{L-1})$ um vetor cujas componentes são potências de um elemento primitivo fixo, β, de $GF(Q)$, e $\bar{1} = (1, 1, \ldots, 1)$. Seja γ_m, $\bar{x}_m \in GF(Q)$. *Einarsson* sugere em [9] o conjunto de Q endereços e mensagens que se segue:

$$\bar{a}_m = \gamma_m . \bar{\beta}\,; \quad \bar{x}_m = x_m \cdot \bar{1}\,, \tag{3-4}$$

para um sistema síncrono (todos os usuários estão alinhados no tempo), e:

$$\bar{a}_m = \gamma_m . \bar{1}\,; \quad \bar{x}_m = x_m \cdot \bar{\beta}\,, \tag{3-5}$$

para um sistema assíncrono.

Posteriormente, *Einarsson* propôs em [27] um conjunto de Q^2 mensagens e Q endereços. Na verdade, o que *Einarsson* propôs foi a transmissão de Q^2 palavras-código de um código *Reed-Solomon* com dois símbolos de informações que mais tarde *Vadja* generalizou para um número qualquer de símbolos de informação. Vejamos os estudos de endereçamento dado por *Vadja*.

3.2.2 Matriz de Transmissão através do Endereçamento dado por Vajda

Vadja [28] assume $\bar{y}$ uma palavra-código de um código *Reed-Solomon* de comprimento n e k símbolos de informação, gerada como descrito abaixo:

$$\bar{y} = x_0 \cdot \bar{1} + x_1 \cdot \bar{\beta}^{(1)} + x_2 \cdot \bar{\beta}^{(2)} + x_3 \cdot \bar{\beta}^{(3)} + \cdots + x_{k-1} \cdot \bar{\beta}^{(k-1)}; \quad \text{onde } x_i \in GF(Q), i = 0, 1, \cdots, k-1 \tag{3-6}$$

e $\bar{\beta}^{(i)} = (1, \beta^i, \beta^{2i}, \cdots, \beta^{(n-1)i})$, $i = 0,\ 1,\ \cdots, k-1$, com β uma raiz *n-ésima* da unidade e primitiva. No apêndice A encontra-se a verificação de que o endereçamento da equação 3-5 é um código RS.

Seja $k = 3$, $n = 2L$, $x_0 = \gamma_m, x_1 = x_{m1}$ e $x_2 = x_{m2}$. Então,

$$\bar{y}_m = x_{m1} \cdot \bar{\beta}^{(1)} + x_{m2} \cdot \bar{\beta}^{(2)} + \gamma_m \cdot \bar{1} \tag{3-7}$$

é uma palavra de um código RS (possivelmente encurtado) de comprimento $2L$ e três símbolos de informação. Portanto, o conjunto de endereços e mensagens que se segue é possível:

$$\bar{a}_m = \gamma_m \cdot \bar{1}\,; \quad \bar{x}_m = x_{m1} \cdot \bar{\beta}^{(1)} + x_{m2} \cdot \bar{\beta}^{(2)} \qquad (3\text{-}8).$$

3.2.3 Limitantes de união da probabilidade de erro para os sistemas síncronos e assíncronos de *Einarsson*

O limitante de união da probabilidade de erro de um sistema proporciona um valor máximo para a probabilidade de erro simulada para um determinado número de usuários no sistema. Os limitantes de união da probabilidade de erro de palavra dados por *Einarsson* em [9], estão mostrados pela tabela 3.1. As probabilidades P_{P1} e P_{P2}, representam os limitantes de união da probabilidade de erro de palavra para o sistema síncrono dado pela equação 3-3 e para o sistema assíncrono dado pela equação 3-4, respectivamente. M é o número de usuários ativos no sistema. Como em [27], as probabilidades de erro de *palavra* podem ser convertidas em probabilidades de erro de *bit* aplicando o fator $\frac{Q}{2(Q-1)}$.

A figura 3.3 mostra os limitantes de união da probabilidade de erro de palavra para os sistemas síncrono e assíncrono com Q=11 e L=6.

Limitante	Probabilidade de Erro de Palavra
$P_P < P_{P1}$	$P_{P1} = (Q-1)\cdot\dfrac{(M-1)(M-2)\cdots(M-L)}{Q^L}$.
$P_P < P_{P2}$	$P_{P2} = (Q-1)\cdot\left\{\prod_{k=0}^{L/2-1}\left[1-\left(1-\dfrac{1}{Q-k}\right)^{M-k-1}\right]\right\}^2$.

Tabela 3.1: Limitantes de união da probabilidade de erro de palavra para os sistemas de *Einarsson*. P_{P1} é para o sistema síncrono (3-3) e P_{P2} é para o sistema assíncrono (3-4).

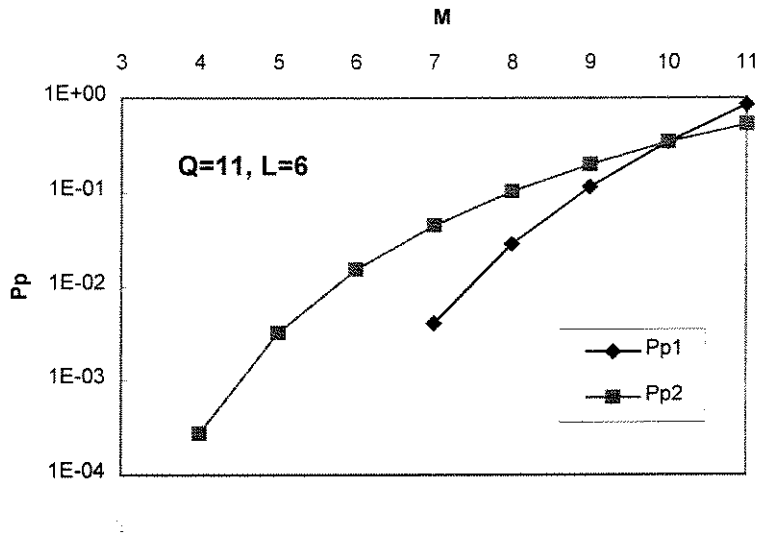

Figura 3.3: Probabilidade de erro de palavra como função do número de usuários M para um sistema com Q=11 e L=6. P_{P1} e P_{P2} são os limitantes de união para o sistema síncrono (3-3) e asssíncrono (3-4), respectivamente.

Como apresenta a figura 3.3, o desempenho esperado para o sistema síncrono é superior em relação ao sistema assíncrono. Neste trabalho, o desempenho nas simulações será idêntico para ambos os sistemas, pois não foi realizada a simulação para os erros de quadro dos sistemas.

3.3 Recepção multi-usuários para os sistemas FSK-FHMA

Esta seção descreve um esquema de decodificação que reduz a probabilidade de erro dos sistemas descritos anteriormente através da deteção conjunta das matrizes de recepção dos diversos usuários. Para tal, usa-se a estrutura algébrica bem definida dos endereços destes usuários. Timor [29, 30] mostrou que, para uma dada probabilidade de erro, pode-se aumentar o número de usuários simultâneos em até 60%.

Basicamente, existem duas técnicas para a implementação da nova decodificação. A primeira analisa se a matriz de recepção possui mais de uma linha completa. Se existir, a regra de decodificação será realizada através de alguns cálculos baseados na estrutura algébrica do código a fim de identificar se todos os *chips* ativos de uma determinada linha pertence ou não ao usuário que está sendo decodificado. A segunda maneira, utiliza o mesmo princípio da técnica anterior com uma lista de usuários ativos no sistema. Isto diminui ainda mais a probabilidade de erro. Vejamos os resultados de forma sistemática para os limitantes de união das probabilidades de erro utilizando tais técnicas.

3.3.1 Decodificação estágio 2 [29, 30]:

Esta decodificação utiliza a estrutura algébrica do código para determinar qual das linhas completamente cheia de uma matriz de recepção é a mensagem a ser decodificada. A *decodificação estágio 2* é empregada se somente se, existir mais de uma linha cheia na matriz de recepção. Caso contrário, a decodificação utilizada será o algoritmo de lógica majoritária que é muito mais simples e rápido [30].

As operações para o teste de interferência podem ser realizada através de registradores. As operações da *decodificação estágio 2* estão bem apresentadas em [29].

Erros podem ser causados na decodificação quando as duas seguintes condições forem satisfeitas:

(i) Existir mais do que uma linha completa na matriz de recepção
(ii) A linha correta passa pelo teste de interferência

Quando isto acontece, escolhe-se aleatoriamente uma das linhas que é decodificada como a mensagem recebida. A figura 3.4 mostra uma matriz de recepção com mais de uma linha completa onde se aplica a *decodificação estágio 2*.

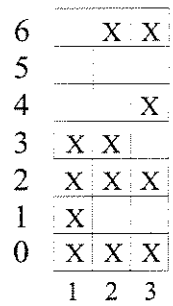

Figura 3.4: Matriz de recepção com mais de uma linha completa. Sistema síncrono com Q=7 e L=3.

O limitante de união para a probabilidade de erro de *bit* para a *decodificação estágio 2* [29] é dado pela tabela 3.2. M é o número de usuários ativos no sistema.

Limitante	**Probabilidade de Erro de *Bit***
$P_b < P_{bE2}$	$P_{bE2} = (Q/2)\cdot(1+S)^L \cdot p^{2L}$,
onde:	$S = (Q-1)\cdot(1-p)\cdot p^{(L-2)}$ $p = 1-(1-Q^{-1})^{M-1}$

Tabela 3.2: Limitante de união da probabilidade de erro de *bit* para a *decodificação estágio 2* (P_{bE2}).

3.3.2 Decodificação estágio 3/2:

A *decodificação estágio 3/2* é realizada se a *decodificação estágio 2* não conseguir decodificar a mensagem recebida (houver ambiguidade, conforme condições (i) e (ii) do item anterior). Então , a decodificação poderá ser realizada de duas maneiras, conhecendo uma lista de usuários ativos no sistema (aplicado para a estação rádio base) ou não (aplicado para a estação móvel) [30].

Os limitantes de união para as probabilidades de erro de *bit* para a *decodificação estágio 3/2* [30] é dado pela tabela 3.3. Novamente, M é o número de usuários ativos no sistema.

Limitante	**Probabilidade de Erro de *Bit***
$P_b < P_{bE3/2}$: Lista de Usuários Conhecida	$P_{bE3/2} = (Q/2)\cdot(1+S\cdot\tilde{p}\cdot P_2^{E2})^L \cdot p^{2L}$ e
$P_b < P_{bE2/3}$: Lista de Usuários Não conhecida	$P_{bE3/2} = (Q/2)\cdot[1+S\cdot(\tilde{p}\cdot P_2^{E2}+1-\tilde{p})]^L \cdot p^{2L}$,
onde:	$p = 1-(1-Q^{-1})^{M-1}$ $S = (Q-1)\cdot(1-p)\cdot p^{(L-2)}$ $\tilde{p} = \frac{M-1}{Q-1}$ $P_2^{E2} < p^L \cdot (1+S)^L$

Tabela 3.3: Limitantes de união da probabilidade de erro de *bit* para a *decodificação estágio 3/2* ($P_{bE3/2}$).

O gráfico da figura 3.5 apresenta os limitantes de união das probabilidades de erro de *bit* para a *decodificação convencional*, para a *decodificação estágio 2* e, para a *decodificação estágio 3/2*, para o sistema com Q=11 e L=6 síncrono. Para encontrar as probabilidades de erro de *bit* para a *decodificação convencional*,

basta multiplicar os resultados obtidos anteriormente para as probabilidades de erro de palavra (figura 3.3), pelo fator $\frac{Q}{2(Q-1)}$.

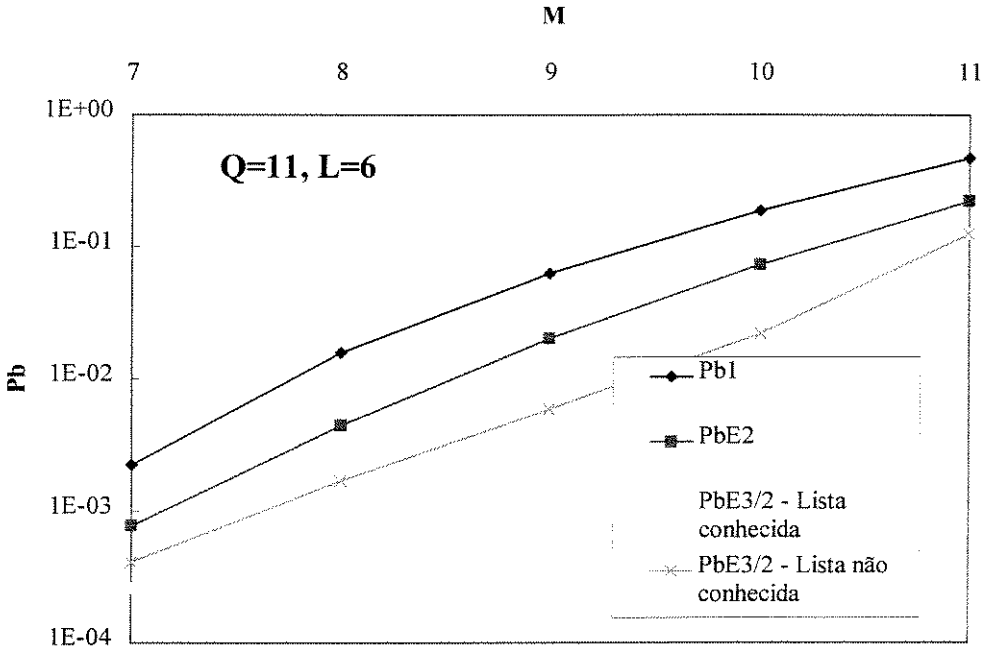

Figura 3.5: Limitantes da probabilidade de erro de *bit* como função do número de usuários M para um sistema com Q=11 e L=6 síncrono. P_{b1} são os limitantes de união para a *decodificação convencional*, P_{bE2} são os limitantes de união para a *decodificação estágio 2*, $P_{bE3/2}$ - *Lista conhecida* são os limitantes de união para a *decodificação estágio 3/2* com lista de usuários ativos conhecida e $P_{bE3/2}$ - *Lista não conhecida* são os limitantes de união para a *decodificação estágio 3/2* sem lista de usuários ativos conhecida.

3.3.3 Decodificação conjunta por máxima verossimilhança

A decodificação conjunta por máxima verossimilhança consiste em um receptor multi-usuário por máxima verossimilhança que provê uma estimativa para todos os Q possíveis símbolos transmitidos. O sistema suporta no máximo Q usuários. Deste modo, o receptor provê uma estimativa para todos os Q possíveis

símbolos transmitidos (k=1, 2, ..., Q) e para uma deteção conjunta por máxima verossimilhança, a regra maximiza

$$\max_{n} \; p(\bar{r}|\bar{u}_n),\; n = 0,\; 1,\; 2,\; \cdots,\; Q^k - 1 \qquad (3\text{-}9)$$

onde e a matriz $\bar{u}_n$ é o resultado da aplicação da lógica OU sobre uma combinação distinta de todos as Q matrizes de transmissão dos usuários, isto é, $\bar{y}_k$, $k = 1,\; 2,\; \cdots,\; Q$. Portanto, para as diversas probabilidades condicionais conjuntas, é necessário avaliar Q^Q matrizes candidatas à possível mensagem enviada.

No entanto, o número de matrizes candidatas à possível mensagem enviada é extremamente grande para um sistema com parâmetro relativamente pequeno. Por exemplo, se Q=11, um pouco menos de $3 \cdot 10^{11}$ matrizes são matrizes candidatas. Por este motivo, *Fiebig* propõe em [31] um algoritmo de deteção sub-ótima que reduz o número de matrizes candidatas para a tomada de decisão da mensagem enviada através da deteção conjunta por máxima verossimilhança.

Capítulo 4

4. Resultados e Simulações

Este Capítulo apresenta os resultados e simulações realizadas neste trabalho e são todos de carácter inédito. Primeiramente, o receptor de lógica majoritária é apresentado. A partir do diagrama de blocos para o receptor de lógica majoritária, é proposta uma forma de construção de uma Rede Neural sem treinamento. Uma comparação sobre o número de transistores utilizando uma rede construída sem treinamento e o respectivo receptor de lógica majoritária construído digitalmente é então avaliado. Uma análise para a rede de *Kohonen* é apresentada mostrando que as suas operações resultam no cálculo de valores intermediários da Rede Neural construída sem treinamento. Por fim, um estudo utilizando códigos de bloco para a representação dos padrões de treinamento para as Redes Neurais *feedforward* é avaliado com o objetivo de reduzir o número de neurônios da camada de saída da rede.

4.1 A implementação da regra de lógica majoritária através de uma RNA sem treinamento

No capítulo 3, foi apresentado que a decodificação mais simples possível para a decodificação da mensagem recebida é o receptor de máxima verossimilhança de LM, que minimiza a probabilidade de erro de mensagens igualmente prováveis. De fato, isto pode ser implementado através de um receptor de LM que implementa a seguinte regra: escolha $\bar{y}$ como a palavra-código transmitida (matriz) se ele possui mais coincidências com a matriz recebida do que $\bar{y}^{*}(\neq \bar{y})$.

Considere o endereçamento dado pela equação (3-3), ou seja, o endereçamento para o sistema síncrono. A figura 4.1 mostra as possíveis mensagens para este sistema. Então, de maneira construtiva podemos implementar um receptor de lógica majoritária com um conjunto de Q neurônios, usando como função de ativação, a função *threshold-logic*. Todas as saídas são comparadas e a saída vencedora (maior), será a mensagem recebida. A figura 4.2 mostra a arquitetura desta Rede Neural construída sem treinamento para as operações de lógica majoritária, para um sistema síncrono com Q mensagens.

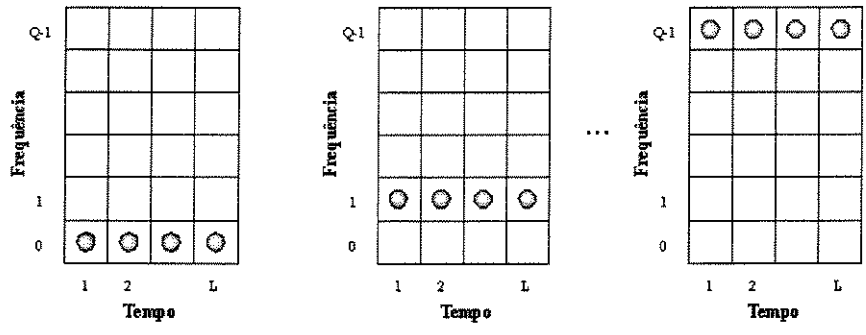

Figura 4.1: Padrões de treinamento para um sistema síncrono.

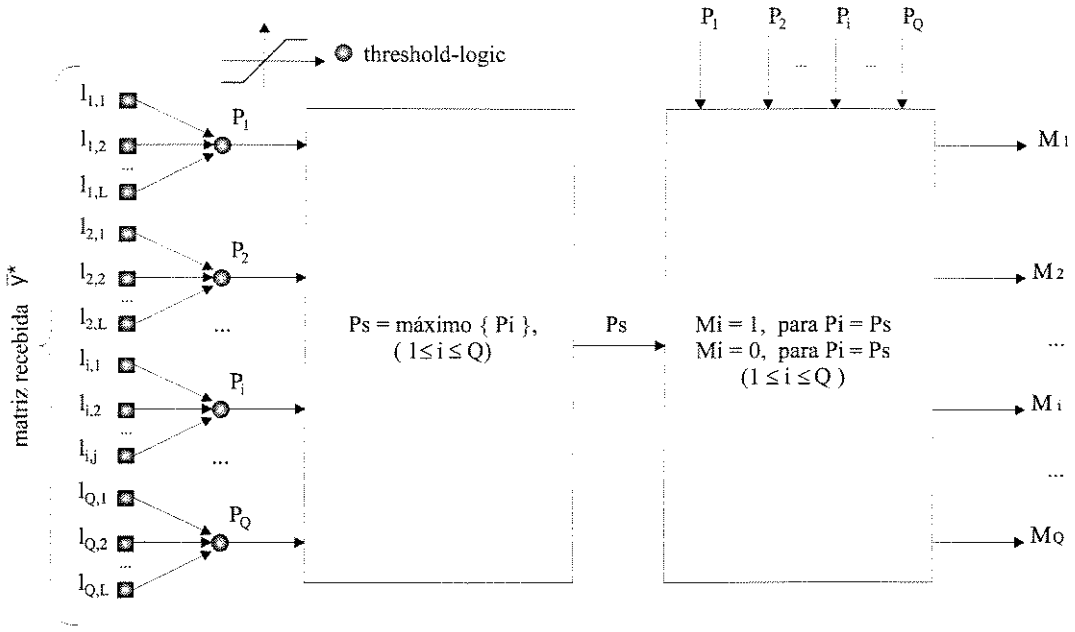

Figura 4.2: Arquitetura de uma Rede Neural construída sem treinamento para as operações de lógica majoritária, para um sistema síncrono com Q mensagens.

Para um sistema assíncrono, as possíveis mensagens são dadas pelo endereçamento dado pela equação (3-4) e podem ser representadas conforme a figura 4.3. Uma Rede Neural semelhante à apresentada pela figura 4.2, poderá ser facilmente construída, alterando apenas as conexões dos *chips* da matriz recebida (entradas) aos neurônios de entrada da rede.

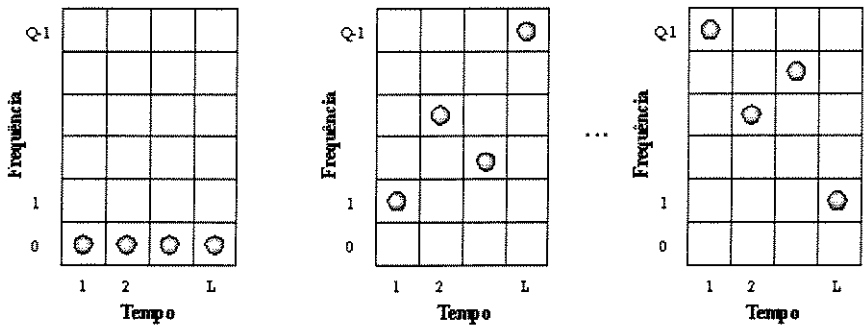

Figura 4.3: Padrões de treinamento para um sistema assíncrono.

O bloco que seleciona o valor de P_S e os valores de M_i (diagrama de blocos dado pela figura 4.2) pode ser implementado através de subredes de seleção-

comparativa. A seguir é apresentado uma descrição mais detalhada de como ficaria a arquitetura neural completa para as operações de lógica majoritária.

4.1.1 Arquitetura neural para o valor de P_S

Considere a subrede de seleção-comparativa dada em [7]. Para que seja selecionado o maior valor entre dois números, basta que seja implementado uma subrede de seleção-comparativa que implemente a seguinte operação:

$$P = P_1 + \text{máximo}\left\{P_2 - P_1,\ 0\right\} \qquad (4\text{-}1).$$

A figura 4.4 apresenta a implementação da equação (4-1) através de dois neurônios.

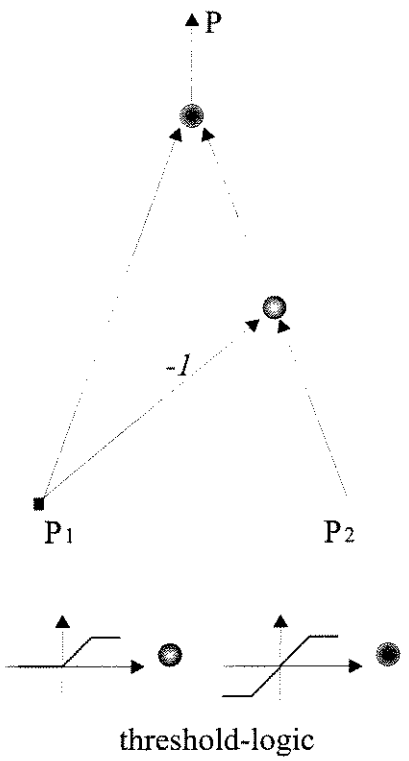

Figura 4.4: Subrede seleção-comparativa

Pelo diagrama da figura 4.2, podemos encontrar o valor de P_S através de várias subredes seleção-comparativa. A figura 4.5 apresenta como ficaria a arquitetura neural para o bloco que seleciona o valor de P_S.

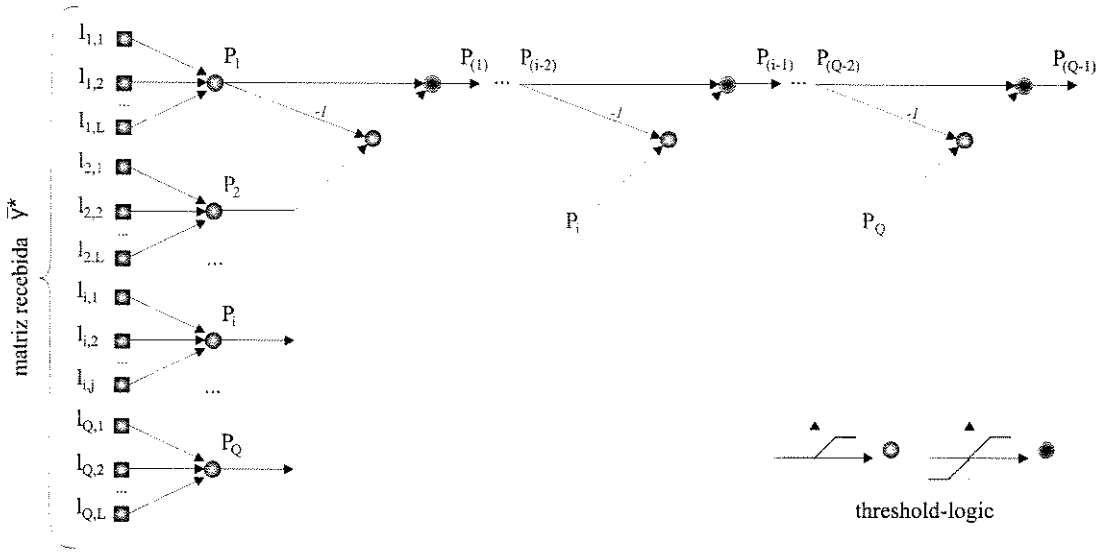

Figura 4.5: Arquitetura neural para bloco que encontra o valor de P_S. Note que $P_{(Q-1)} = P_S = \text{máximo}\left\{P_1, P_2, \cdots, P_i, \cdots, P_Q\right\}$.

4.1.2 Arquitetura neural para o valores de M_i

M_i assume dois valores: no estado habilitado M_i será igual a *um* e no estado desabilitado M_i será igual a *zero*. Sabendo que $P_i \leq P_{(Q-1)}$, então podemos construir uma subrede que implemente a operação para o valor de M_i dada por:

$$M_i = \text{máximo}\left\{ P_i - P_{(Q-1)} + 1,\ 0 \right\},\ (1 \leq i \leq Q). \qquad (4\text{-}2)$$

A operação dada pela equação (4-2) pode ser representa por uma subrede neural conforme mostra a figura 4.6.

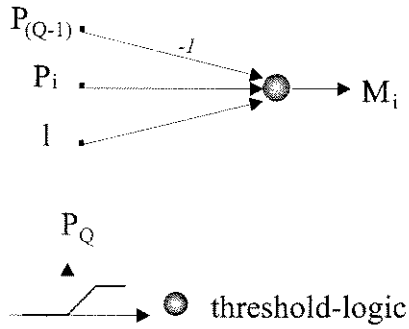

Figura 4.6: Subrede para o valor de M_i.

Pelo diagrama da figura 4.2, podemos encontrar a arquitetura neural para o bloco que seleciona os valores de M_i. A figura 4.7 mostra como ficaria este bloco, utilizando Q subredes para o valor de M_i.

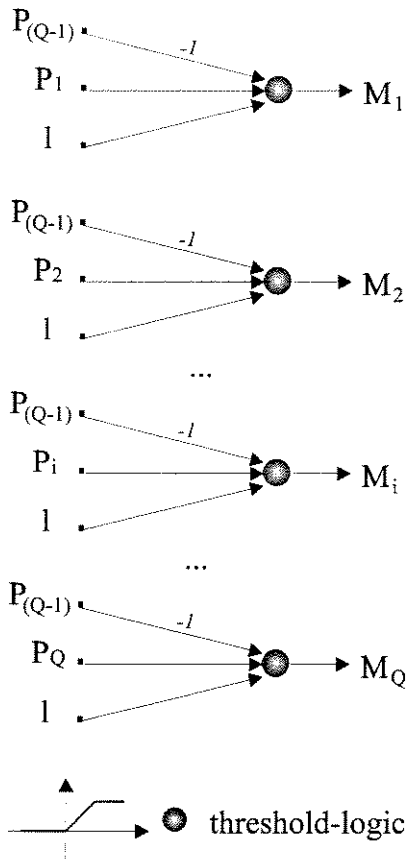

Figura 4.7: Arquitetura neural para bloco que encontra os valores de M_i.

É digno de nota que a arquitetura neural para o bloco que encontra o valor de P_S (figura 4.5) juntamente com a arquitetura neural para o bloco que encontra os valores de M_i (figura 4.7), se aplica para sistemas com interferência de acesso múltiplo bem como para sistemas que além da interferência apresentam ruído do canal de telefonia móvel [27].

Portanto, uma nova estrutura baseada em RNA's realizada de maneira construtiva, é capaz de implementar perfeitamente o receptor de lógica majoritária.

Em [7] foi sugerida a implementação de um decodificador de *Viterbi* baseado em uma RNA que utiliza a subrede de seleção-comparativa (figura 4.4). Esta implementação possui uma complexidade aproximadamente seis vezes menor do que uma implementação convencional que utiliza operações ACS (*add-compare-select*) [6, 32]. Assim, o bloco da RNA construída sem treinamento responsável pelo cálculo de P_S, pode ser construído utilizando a técnica de subrede seleção-comparativa, podendo reduzir bastante o número de transistores utilizados em comparação com o mesmo receptor construído de maneira convencional, isto é, construído digitalmente levando-se em conta um determinado número de operações ACS's necessárias para implementação do receptor.

No próximo tópico será analisado uma outra forma de encontrar a arquitetura neural proposta para os valores de P_i, $1 \leq i \leq Q$ (dado pela figura 4.2). Será utilizada a técnica de treinamento de padrões por aprendizado conhecida como Redes de *Kohonen*.

4.2 Rede de *Kohonen* para as operações de P_i

Uma Rede Neural paramétrica de *Kohonen* pode ser obtida, utilizando como padrões de treinamento, os padrões dados pela figura 4.1, com $Q=7$ e $L=3$. Para este sistema, é exigido uma arquitetura de rede conforme mostra a figura 4.8.

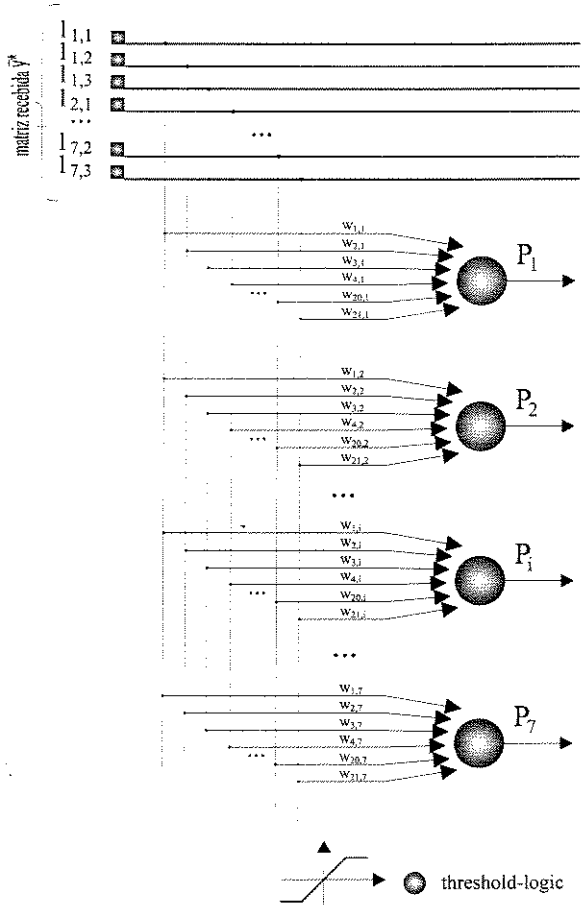

Figura 4.8: Arquitetura de uma Rede Neural paramétrica de *Kohonen* com $Q=7$ e $L=3$.

As conexões dos *chips* da matriz recebida (entradas) aos neurônios de entrada da rede são feitas através dos pesos sinápticos $w_{i,j}$. A matriz de pesos sinápticos W para o sistema com $Q=7$ e $L=3$ tem dimensão 21x7 (21 linhas e 7 colunas). Esta dimensão está relacionada com o número de entradas da rede igual a 21 e com o número de neurônios de saída da rede igual a 7. A matriz W obtida através de uma simulação em um computador digital é mostrada pela figura 4.9.

$$W = \begin{pmatrix}
-0.033 & -0.033 & -0.033 & \mathbf{0.198} & -0.033 & -0.033 & -0.033 \\
-0.033 & -0.033 & -0.033 & \mathbf{0.198} & -0.033 & -0.033 & -0.033 \\
-0.033 & -0.033 & -0.033 & \mathbf{0.198} & -0.033 & -0.033 & -0.033 \\
-0.033 & -0.033 & -0.033 & -0.033 & -0.033 & \mathbf{0.198} & -0.033 \\
-0.033 & -0.033 & -0.033 & -0.033 & -0.033 & \mathbf{0.198} & -0.033 \\
-0.033 & -0.033 & -0.033 & -0.033 & -0.033 & \mathbf{0.198} & -0.033 \\
\mathbf{0.198} & -0.033 & -0.033 & -0.033 & -0.033 & -0.033 & -0.033 \\
\mathbf{0.198} & -0.033 & -0.033 & -0.033 & -0.033 & -0.033 & -0.033 \\
\mathbf{0.198} & -0.033 & -0.033 & -0.033 & -0.033 & -0.033 & -0.033 \\
-0.033 & -0.033 & -0.033 & -0.033 & \mathbf{0.198} & -0.033 & -0.033 \\
-0.033 & -0.033 & -0.033 & -0.033 & \mathbf{0.198} & -0.033 & -0.033 \\
-0.033 & -0.033 & -0.033 & -0.033 & \mathbf{0.198} & -0.033 & -0.033 \\
-0.033 & -0.033 & \mathbf{0.198} & -0.033 & -0.033 & -0.033 & -0.033 \\
-0.033 & -0.033 & \mathbf{0.198} & -0.033 & -0.033 & -0.033 & -0.033 \\
-0.033 & -0.033 & \mathbf{0.198} & -0.033 & -0.033 & -0.033 & -0.033 \\
-0.033 & \mathbf{0.198} & -0.033 & -0.033 & -0.033 & -0.033 & -0.033 \\
-0.033 & \mathbf{0.198} & -0.033 & -0.033 & -0.033 & -0.033 & -0.033 \\
-0.033 & \mathbf{0.198} & -0.033 & -0.033 & -0.033 & -0.033 & -0.033 \\
-0.033 & -0.033 & -0.033 & -0.033 & -0.033 & -0.033 & \mathbf{0.198} \\
-0.033 & -0.033 & -0.033 & -0.033 & -0.033 & -0.033 & \mathbf{0.198} \\
-0.033 & -0.033 & -0.033 & -0.033 & -0.033 & -0.033 & \mathbf{0.198}
\end{pmatrix}$$

Figura 4.9: Matriz de pesos sinápticos obtida no treinamento de uma Rede Neural paramétrica de *Kohonen* com $Q=7$ e $L=3$.

Observando a matriz de pesos sinápticos W, verificamos que neste treinamento, a primeira coluna da matriz W (primeiro neurônio da camada de saída da rede) é responsável pela classificação da terceira mensagem treinada. A segunda coluna da matriz W é responsável pela classificação da sexta mensagem treinada. E assim por diante. Devido ao caráter auto-organizado da rede, se treinarmos novamente a rede, encontraremos uma nova matriz W e muito provavelmente, o primeiro neurônio da camada de saída da rede não será mais responsável pela classificação da terceira mensagem.

Analisando o resultado do treinamento da matriz de pesos sinápticos W notamos que todas as entradas das redes estão conectadas aos neurônios da rede, pois não temos valores nulos na referida matriz. Assim, aparentemente se compararmos a arquitetura da Rede Neural paramétrica de *Kohonen* que implementa o cálculo dos valores de P_i, chegamos a conclusão que é mais complexa do que a Rede Neural implementada de maneira construtiva (figura 4.2). Entretanto, se substituirmos os valores positivos da matriz de pesos sinápticos por *um* e os valores negativos por *zero*, o desempenho em termos de probabilidade de erro da rede não é alterado. Isto se deve ao fato de que a regra de decisão dada pelo algoritmo para a rede paramétrica de *Kohonen* (passo 3 da tabela 2.6),

$$D(j) = \sum_i (w_{ij} - x_i)^2 , \qquad (4\text{-}3)$$

$$(1 \le j \le Q \text{ e } 1 \le i \le Q.L),$$

encontre o índice J tal que $D(J)$ seja mínimo,

implementa as operações de distância Euclidiana entre o vetor de entradas $\bar{x}$ e a i-ésima coluna da matriz de pesos sinápticos W. Portanto, para este treinamento se a quarta coluna da matriz W apresentar a menor distância Euclidiana (menor distância de *Hamming*) para um determinado padrão de teste, decodificaríamos como sendo enviada a primeira mensagem.

Portanto, a Rede Neural paramétrica de *Kohonen* é na verdade uma forma de efetuar o cálculo para os valores de P_i $(1 \le i \le Q)$ da rede construída sem treinamento, ou seja, $P_i = \alpha D(i) + \beta$, onde α e β são constantes.

4.3 Simulações utilizando redes *feedforward*

O uso de uma rede *feedforward* para a solução do problema proposto no item 4.1, pode ser justificado se uma RNA treinada possuir na camada de saída, um número de neurônios menor do que Q. O principal motivo, é o fato de que uma Rede Neural *feedforward* com uma camada escondida de neurônios seria muito mais complexa do que uma rede de *Kohonen*. Assim, para a representação de cada padrão da rede, poderá ser usado um código de bloco com Q palavras-códigos de comprimento menor do que Q.

Na fase de testes, a Rede Neural *feedforward* apresenta em suas saídas valores reais. Assim, para uma decisão de um possível padrão ruidoso apresentado à rede (decisão de qual mensagem foi transmitida), calculam-se as métricas de distância Euclidiana para todas as palavras-códigos envolvidas, sendo que a mensagem decodificada, será a que possuir a menor métrica. No entanto, existem outras técnicas mais eficientes para a decodificação da mensagem. A parte da teoria da codificação que analisa tais técnicas é denominada de decodificação suave. Treliças baseadas na arquitetura dos códigos facilitam encontrar a menor métrica para a tomada da decisão pois podemos utilizar o algoritmo de *Viterbi*. Mais adiante, é feita uma análise melhor para uma decodificação específica tratada neste trabalho.

4.3.1 Representação do padrão de saída e regra de decisão

O que segue está baseado em [23]. O diagrama de blocos para um classificador de m padrões para uma rede *perceptron* com múltipla camada é apresentado pela figura 4.10.

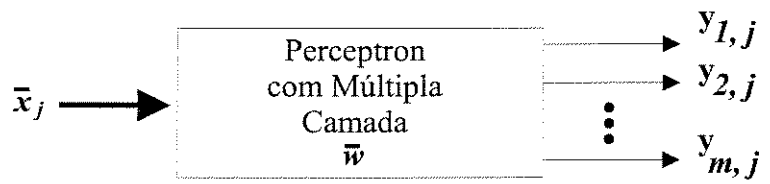

Figura 4.10: Diagrama de blocos de classificador de padrões.

Seja $\bar{x}_j$ o j-ésimo padrão de treinamento e $y_{k,j} = F_k(\bar{x}_j)$, $k = 1, 2, ..., m$ a k-ésima saída da rede, onde a função $F_k(.)$ define o mapeamento treinado pela rede neural da entrada para a k-ésima saída. Por conveniência, seja

$$\bar{y}_j = [y_{1,j}, y_{2,j}, \cdots, y_{m,j}]^T$$
$$\bar{y}_j = [F_1(\bar{x}_j), F_2(\bar{x}_j), \cdots, F_m(\bar{x}_j)]^T$$
$$\bar{y}_j = \bar{F}(\bar{x}_j)$$

onde $\bar{F}(.)$ é um vetor operador de funções.

O problema da classificação de m padrões, se resume na busca de uma solução através do operador de funções $\bar{F}(.)$, que minimiza o erro quadrático médio da função custo dada por:

$$L(\bar{F}) = \frac{1}{2N}\left\|\bar{d}_j - \bar{F}(\bar{x}_j)\right\|^2 \qquad (4\text{-}4)$$

onde $\bar{d}_j$ é o padrão de saída desejado (alvo) para um padrão de treinamento $\bar{x}_j$, $\|\,.\,\|$ é a norma Euclidiana ao vetor atribuído e N é o número total de padrões entrada-saída apresentados à rede para o treinamento.

Suponha agora que a rede é treinada com valores binários para a representação do padrão de saída, dado por:

$d_{k,j} = 1$ quando o protótipo $\bar{x}_j$ pertence à classe ζ_k e

$d_{k,j} = 0$ quando o protótipo $\bar{x}_j$ não pertence à classe ζ_k.

Baseado nesta notação, a classe ζ_k é representada pelo vetor alvo *m*-dimensional:

$$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow \textit{elemento de ordem } k .$$

Suponha também que o classificador *perceptron* de camada múltipla é treinado com o algoritmo *backpropagation* com um conjunto finito de exemplos independentes e identicamente distribuídos (i.i.d.), tomando assim uma aproximação assintótica sobre as probabilidades de classe *a posteriori*. Deste modo, para um problema de classificação de *m* padrões, o *k*-ésimo elemento da resposta desejada $\bar{d}$ é igual a *um* se o vetor pertence a classe ζ_k e *zero* caso contrário. Sob esta condição, a esperança condicional $E[\bar{d}_k|\bar{x}]$ é igual a probabilidade de classe *a posteriori* $P(\zeta_k|\bar{x})$, $k = 1, 2, ..., m$. Também pela *lei dos grandes números*, se o conjunto de treinamento N possui tamanho infinito, o vetor de pesos $\bar{w}$ que minimiza a função custo dada pela equação (4-3) aproxima o vetor de pesos ótimo $\bar{w}^*$.

Assim, se o tamanho do conjunto de treinamento N é grande (não infinito), a rede treinada convergirá para um mínimo local e não global. Mesmo assim, poderemos aplicar a regra de decisão da saída baseada nas probabilidades de classe *a posteriori* $P(\zeta_k|\bar{x})$, $k = 1, 2, ..., m$ e portanto a regra mais apropriada é:

Classifique um vetor $\bar{x}$ *como pertencente à classe* ζ_k *se*

$$F_k(\bar{x}) > F_j(\bar{x}) \text{ para todo } j \neq k,$$

onde $F_k(\bar{x})$ *e* $F_j(\bar{x})$ *são elementos do vetor da função de mapeamento*

$$\bar{F}(\bar{x}) = \begin{bmatrix} F_1(\bar{x}) \\ F_2(\bar{x}) \\ \vdots \\ F_m(\bar{x}) \end{bmatrix}.$$

Alguns códigos foram utilizados para tentar reduzir o número de neurônios da camada de saída da RNA. Evidentemente, quanto menor for o tamanho da palavra-código utilizada, pior será o desempenho em termos de probabilidade de erro da rede. O desempenho será pior, sobretudo porque estaremos confundindo a rede e não teríamos uma aproximação assintótica sobre as probabilidades de classe *a posteriori*. Para se ter uma idéia, se a representação da tabela 4.1 fosse utilizada, o desempenho da RNA seria o mesmo para os resultados obtidos para as simulações de lógica majoritária, pois neste caso, teríamos uma aproximação assintótica sobre as probabilidades de classe *a posteriori*. Entretanto, a rede seria muito complexa, pois não estaríamos reduzindo o número de saídas da rede, pois a rede treinada com essa codificação necessitaria de Q neurônios na camada de saída.

Classe	*Representação da saída*
1	$\bar{y}_1 = [+1, -1, -1, \cdots, -1]^T$
2	$\bar{y}_2 = [-1, +1, -1, \cdots, -1]^T$
...	...
Q	$\bar{y}_Q = [-1, -1, -1, \cdots, +1]^T$

Tabela 4.1: Representação do vetor de saída $\bar{y}_k$ de Q possíveis mensagens. ($\bar{y}_1, \bar{y}_2, \cdots, \bar{y}_Q$ possuem Q elementos).

A seguir, é apresentado os resultados para umas redes *feedforward* treinadas. Alguns códigos foram sugeridos, como tentativa de redução do número de neurônios da rede. O sistema assíncrono com Q=31 e L=15 é então considerado.

4.3.2 Uma RNA *feedforward* com um código binário mais um bit de paridade na camada de saída.

Para a representação das 31 mensagens de um sistema síncrono com *Q=31* e *L=15*, seriam necessários 5 *bits* para o código binário. Assim, com 5 neurônios na camada de saída seriam suficientes para representar as 31 possíveis mensagens do sistema. Um *bit* de verificação de paridade foi introduzido para melhorar o desempenho da rede. A palavra toda nula foi descartada.

Uma Rede Neural *feedforward* com 465 entradas, 50 neurônios na camada escondida (ainda não otimizado) e 6 neurônios na camada de saída foi treinada para a decodificação das 31 possíveis mensagens do sistema (fase de treinamento). A partir das probabilidades de erro de palavra encontradas na simulação (fase de testes), as probabilidades de erro de *bit* foram encontradas através da correção pelo fator $\frac{Q}{2(Q-1)}$. Os resultados para as probabilidades de erro de bit para um sistema assíncrono com *Q=31* e *L=15*, estão apresentados na tabela 4.2.

Probabilidades de Erro de Bit para a representação binária mais paridade

NUA*	Pb**	NUA*	Pb**	NUA*	Pb**	NUA*	Pb**
1	0,0E+00	10	5,7E-02	19	2,4E-01	28	3,6E-01
2	0,0E+00	11	8,0E-02	20	2,6E-01	29	3,7E-01
3	0,0E+00	12	1,0E-01	21	2,8E-01	30	3,7E-01
4	5,2E-05	13	1,2E-01	22	2,9E-01	31	3,8E-01
5	7,8E-04	14	1,4E-01	23	3,0E-01		
6	3,7E-03	15	1,6E-01	24	3,2E-01		
7	1,2E-02	16	1,9E-01	25	3,3E-01		
8	2,3E-02	17	2,1E-01	26	3,4E-01		
9	3,9E-02	18	2,4E-01	27	3,5E-01		

* NUA é o número de usuários ativos no sistema; ** Pb é a probabilidade de erro de bit.

Tabela 4.2: Probabilidades de Erro de Bit para um sistema síncrono com Q=31 e L=15, representação binária mais um bit de paridade.

Para uma simulação de lógica majoritária com um número de usuários variando de 1 à 15, a probabilidade de erro de bit seria nula. No entanto, nota-se através dos resultados apresentados pela tabela 4.2 que não foi possível obter tal desempenho com a representação binária mais um bit de paridade. Portanto, necessitaremos de mais neurônios na camada de saída para aumentar o poder de classificação da RNA. A seguir é apresentado uma solução através de um outro código de bloco, o código *Reed-Müller* (16,5,8).

4.3.3 RNA com um código *Reed-Müller* (16,5,8) na camada de saída.

Um código *Reed-Müller* de comprimento 16, com 32 palavras-código, foi escolhido para a representação das 31 mensagens do sistema síncrono com *Q=31* e *L=15*. Uma Rede Neural *feedforward* com 465 entradas, 50 neurônios na camada escondida (ainda não otimizado) e 16 neurônios na camada de saída foi treinada para a decodificação das 31 possíveis mensagens do sistema (fase de treinamento). Os resultados para as probabilidades de erro de bit encontradas (fase de testes) estão apresentados na tabela 4.2.

Probabilidades de Erro de Bit para uma representação com um código (16,5,8)

NUA*	Pb**	NUA*	Pb**	NUA*	Pb**	NUA*	Pb**
1	0,0E+00	10	5,2E-05	19	3,1E-02	28	1,5E-01
2	0,0E+00	11	5,2E-05	20	4,0E-02	29	1,7E-01
3	0,0E+00	12	7,8E-04	21	5,2E-02	30	1,8E-01
4	0,0E+00	13	1,5E-03	22	6,5E-02	31	2,0E-01
5	0,0E+00	14	3,6E-03	23	8,1E-02		
6	0,0E+00	15	5,6E-03	24	8,7E-02		
7	0,0E+00	16	9,8E-03	25	1,1E-01		
8	0,0E+00	17	1,4E-02	26	1,2E-01		
9	5,2E-05	18	2,1E-02	27	1,4E-01		

* NUA é o número de usuários ativos no sistema; ** Pb é a probabilidade de erro de bit.

Tabela 4.3: Probabilidades de Erro de Bit para um sistema síncrono com Q=31 e L=15, representação com o código de bloco (16,5,8).

O gráfico para as probabilidades de erro de bit dados pelas tabelas 4.2 e 4.3 é mostrado pela figura 4.11.

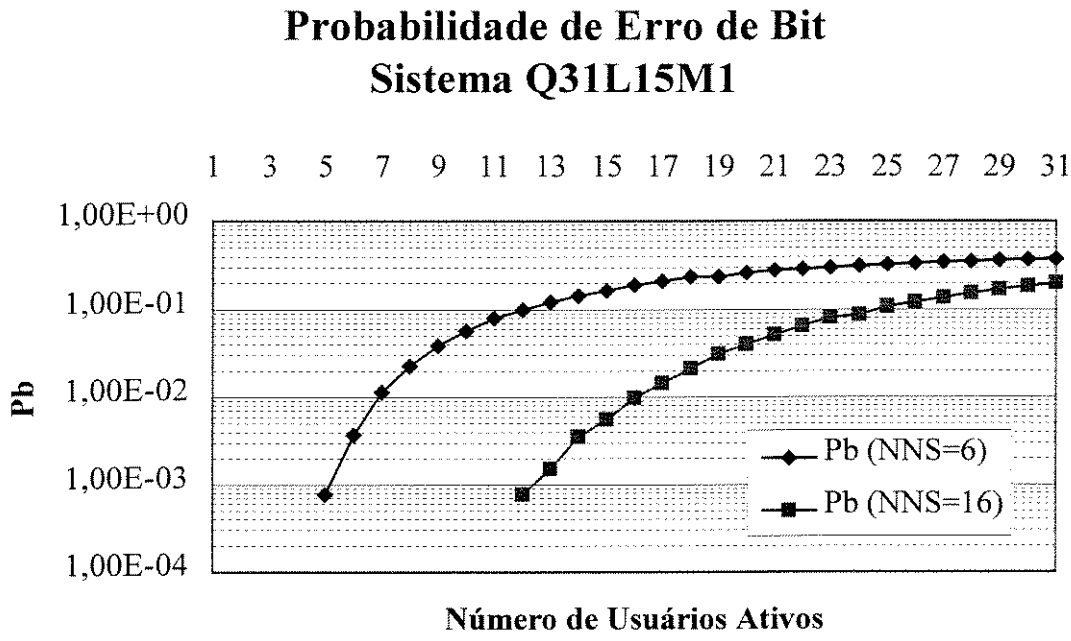

Figura 4.11: Probabilidades de erro de bit para um sistema assíncrono com Q=31 e L=15. NNS é o número de neurônios na camada de saída que depende do código utilizado.

Novamente, notamos que o desempenho de lógica majoritária não foi atingido. No entanto, este receptor sub-ótimo poderia ser implementado para um número de usuários ativos inferior ou igual a *doze*, com probabilidades de erro de bit inferior a 10^{-3}. Uma justificativa da implementação da regra de lógica majoritária utilizando uma RNA *feedforward* pode ser atribuída a não realização de um receptor de LM quando o valor do parâmetro Q for muito elevado. Assim, utilizando uma RNA *feedforward* com um número reduzido para o número de neurônios da camada de saída (menor do que Q), receptores subótimos poderiam ser obtidos.

4.3.4 Decodificação suave

Para a decodificação das mensagens no receptor, um método seria o cálculo das distâncias Euclidianas para todas as palavras-códigos envolvidas. Este método

de decodificação é conhecido por força bruta. No entanto, isto pode ser um pouco complexo se o número de palavras-códigos for numeroso. Assim, surge pela teoria da codificação o estudo de treliças construídas a partir da arquitetura de construção do código. O número de somas e comparações para a decodificação por treliça é menor do que para uma decodificação por força bruta. E com algoritmos ainda mais eficientes, o número dessas somas e comparações pode ser ainda menor, como é no caso da aplicação do algoritmo de *Wagner* [33].

A figura 4.11 mostra a treliça para o código utilizado anteriormente, ou seja, o código *Reed-Müller* (16,5,8) [34]. A treliça constitui-se de 4 treliças idênticas em paralelo, onde cada treliça representa um código de verificação de paridade (4,3,2). Assim, seria possível decodificar cada um destes códigos através do algoritmo de *Wagner*.

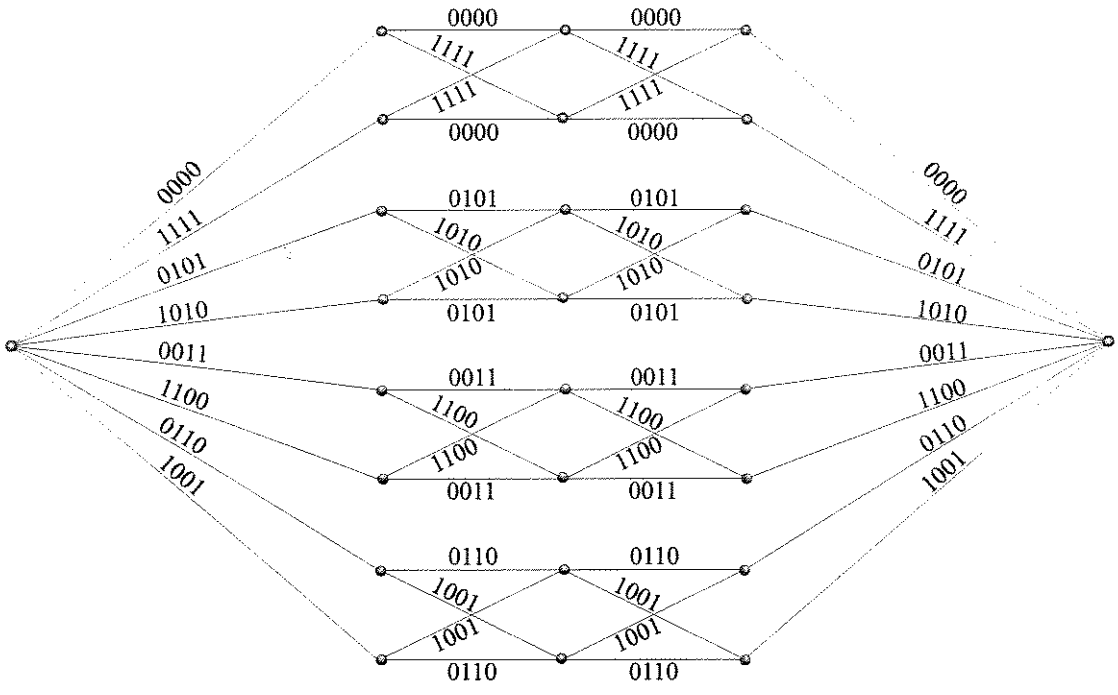

Figura 4.12: Treliça para o código *Reed-Müller* (16,5,8). Notação para a representação do código utilizado pela RNA: o bit "1" representa "1" e o bit "0" representa "-1".

A distância Euclidiana entre um vetor $\bar{x}$ e um vetor $\bar{y}$ é dado por:

$$d_E^2(\bar{x},\bar{y}) = \|\bar{x}\|^2 + \|\bar{y}\|^2 - 2\,\bar{x}\cdot\bar{y}\,.$$

Assim, para uma dada palavra recebida $\bar{r}$ e para uma certa palavra-código y_i, temos que a distância Euclidiana será dada por:

$$d_E^2(\bar{r},\bar{y}_i) = \|\bar{r}\|^2 + \|\bar{y}_i\|^2 - 2\,\bar{r}\cdot\bar{y}_i \qquad (4\text{-}5).$$

Para a regra de máxima verossimilhança, a palavra-código y_i que possuir a *menor* distância Euclidiana com relação a palavra recebida $\bar{r}$, será a palavra-código mais próxima da palavra recebida. Se todas as palavras-código tiverem módulo constante (mesma energia), então a regra será equivalente a escolher a palavra-código y_i que tiver o *maior* produto interno $\bar{r}\cdot\bar{y}_i$.

A decodificação através da força bruta, consiste no cálculo da distância Euclidiana para todas as palavras-código e a palavra-código mais próxima da palavra recebida, será a que apresentar a menor distância Euclidiana. No entanto, para o código (16,5,8) utilizado anteriormente, podemos aplicar a regra do produto interno pois temos a mesma energia para toda as palavras do código. Assim, é necessário a realização de 15 operações de soma para o cálculo do produto interno de cada palavra-código. Para a decodificação de uma palavra recebida seriam necessárias um total de 465 somas e de 30 comparações.

A aplicação do algoritmo de *Wagner* utilizando a treliça do código *Reed-Müller* (16,5,8) consiste na aplicação do algoritmo de *Wagner* quatro vezes, pois temos neste caso, 4 treliças em paralelo. A aplicação do algoritmo de *Wagner* reduz o número de somas necessárias, pois o algoritmo necessita de 32 somas para o cálculo das métricas dos bits "-1" e "1", de 12 operações de soma para a métrica de cada código (4,3,2) e de 16 operações de subtração para o cálculo da confiabilidade de cada decisão abrupta, contribuindo com um total de 60

operações. O número de comparações necessárias caso o algoritmo de *Wagner* falhar é igual a 12 e para a decisão final é igual a 3. Portanto, 15 comparações no total seriam necessárias para a aplicação do algoritmo de *Wagner*.

A tabela 4.3 apresenta o número de comparações e de operações que seriam necessárias para decodificação através da treliça do código *Reed-Müller* (16,5,8) e para a decodificação utilizando força bruta. Observando o número de somas e o número de comparações necessárias para a implementação da decodificação da mensagem recebida, a melhor técnica seria a que utiliza o algoritmo de *Wagner*.

Técnica	**Número de Operações**	**Número de Comparações**
Força Bruta	31x15 = 465	30
Algoritmo de *Wagner*	4x8+4x3+4x4+15 = 75	4x3+3 = 15

Tabela 4.4: Tabela comparativa para a decodificação da mensagem recebida através do algoritmo de *Wagner* utilizando a treliça do código *Reed-Müller* (16,5,8) e utilizando força bruta.

Considerações Finais e Perspectivas Futuras

Como principais contribuições deste trabalho podemos citar:

i. A elaboração de uma nova estrutura baseada em Redes Neurais Artificiais realizada de maneira construtiva, capaz de implementar perfeitamente o receptor proposto;

ii. A implementação do receptor de lógica majoritária através da rede paramétrica de *Kohonen,* juntamente com a alteração dos valores dos pesos sinápticos, proposto neste trabalho, levam aos resultados e à mesma estrutura proposta através das RNA's realizada de maneira construtiva. Pode ser ressaltado ainda, que se uma estrutura com todas as conexões da rede paramétrica de *Kohonen* já existente para a realização de uma determinada tarefa, pode ser facilmente modificada para a realização do receptor de lógica majoritária que não precisa de todas as conexões;

iii. Proposta de uma estrutura por aprendizado, utilizando RNA *feedforward* e códigos corretores de erro para a representação dos padrões de saída da rede. A regra de decisão através de algoritmos de decodificação por decisão suave, como o uso da treliça de um código corretor de erro, pode facilitar a classificação dos padrões treinados, principalmente porque reduz o número de saídas da Rede Neural. Neste trabalho, a aplicação de códigos corretores de erro e de decodificação por decisão suave não levou ao desempenho ótimo do receptor de lógica majoritária. No entanto, uma classe de receptores sub-ótimos podem ser implementados e dependendo do tamanho do sistema, a arquitetura da RNA *feedforward* pode ser interessante.

Trabalhos futuros poderão ser realizados e como sugestões podemos apontar:

i. A Implementação do receptor projetado de maneira construtiva através de um *chip* dedicado, visto que o número de transistores utilizados neste caso, seria bem menor do que se fosse realizado uma implementação digital.

ii. Um estudo mais aprofundado da lei de aprendizado pelo gradiente descendente, aplicado a algoritmos de treinamento que utilizam derivadas de segunda ordem ou superior, com o objetivo de levar a convergência a um ponto de mínimo global, para aplicações de RNA que utilizem códigos corretores de erro e regra de decisão através de algoritmos de decodificação por decisão suave. Assim, a aplicação de código corretor de erro para a redução do número de saídas da RNA, cria uma potencialidade para ser aceito com paradigmas para soluções de outros problemas de comunicações, com possibilidades de ampliação a outras esferas.

iii. A aplicação de Redes Neurais Artificiais para a decodificação conjunta de máxima-verossimilhança (equação 3-2) de multi-usuários de um sistema de comunicação (item 3.3 do capítulo 3).

Referências Bibliográficas

[1] POTTER, R. Personal communications for the mass market. *Telecommunications*, int. ed., v.24, Sep., 1990.

[2] HUFF, D. L. Advanced mobile phone service: the developmental system. *B.S.T.J.*, v.58, 249, Jan., 1979.

[3] MARAL, G., BOUSQUET, M. *Satellite Communications Systems.* New York: John Wiley & Sons, 1976.

[4] PROAKIS, J. G. *Digital communications.* New York: McGraw-Hill, 1995.

[5] YUAN, J., CHEN, C. S. Correlation decoding of the (24,12) Golay code using neural networks. *IEE Proceedings-I*, v.138, n.6, p.517-524, Dec., 1991.

[6] KECHRIOTIS, G. I., MANOLAKOS, E. S. Hopfield Neural Network Implementation of the Optimal CDMA Multiuser Detector. *IEEE Trans. on Neural Networks*, v.7, n.1, Jan., 1996.

[7] WANG, X., WICKER, S., An Artificial Neural Net Viterbi Decoder. *IEEE Trans. on Comm.*, v.44, n.2, Feb., 1996.

[8] LIN, S., COSTELLO JR., D. J. *Error control coding - Fundamentals and Applications.* New Jersey: Prentice-Hall, 1993.

[9] EINARSON, G. Address Assignment for a Time-Frequency-Coded, Spread-Spectrum System. *B.S.T.J.*, v.59, n.7, p.1241-1255, Sep., 1980.

[10] MAC WILIANS, F. J., SLOANE N. J. *The Theory of Error-Correcting Codes.* Netherlands: Elsevier, 1977.

[11] KÓVACS, Z. L. *O cérebro e a sua mente - uma introdução à neurociência computacional.* Brasil: Edição acadêmica, 1997.

[12] MCCULLOCH, J. L., PITTS, W. A logical calculus of ideas immanent in nervons activity. *Bulletin of Mathematical Biophysics*, v.5, p.115-133, 1943.

[13] ROSENBLATT, F. The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, v.65, p.386-408, 1958.

[14] WIDROW, B., HOFF, M. E. Jr. Adaptive switching circuits. *IRE WESCON Convention Record*, p.96-104, 1960.

[15] MINSKY, M. L., PAPERT, S. A. *Perceptrons*. Cambridge: MIT Press, 1969.

[16] KOEHONEN, T. Correlation matrix memories. *IEEE Trans. on Computers*, v.C-21, p.353-359, 1972.

[17] HOPFIELD, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the U.S.A.*, v.79, p.2554-2558, 1982.

[18] HINTON, G. E., SEJNOWSKY, T. J. Optimal perceptual inference. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, p.448-453, 1983.

[19] RUMELHART, D. E., HINTON, G. E., WILLIAMS, R. J. Learning internal representations by back-propagation errors. *Nature*, v.323, p.533-536, 1985.

[20] JACOBS, R. A. Increased rates of convergence through learning rate adaptations. *Neural Networks*, v.1, p.295-307, 1988.

[21] CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, v.2, p.303-314, 1989.

[22] LEE, C. C. Fuzzy logic in control systems: Fuzzy logic controller - Part I and Part II. *IEEE Trans. on Systems, Man, and Cybernetics*, v.20, p.404-418 e p.419-435, 1990.

[23] HAYKIN, S. *Neural Networks - A Comprehensive Foundation.* New York: Prentice-Hall, 1994.

[24] FAUSETT, L. *Fundamentals of Neural Networks.* New Jersey: Prentice Hall International, 1994.

[25] VON ZUBEN, F. J. *Modelos paramétricos e não paramétricos de redes neurais artificiais e aplicações.* Campinas: Faculdade de Engenharia

Elétrica e de Computação da UNICAMP, 1996, 244p. (Tese, Doutorado em Engenharia Elétrica).

[26] GOODMAN, B. J., HENRY, P. S., PRABHU V. K. Frequency-Hopped Multilevel FSK for Mobile Radio. *B.S.T.J.*, v.59, n.7, p.1257-1275, Sept., 1980.

[27] EINARSSON, G. Coding for a Multiple-Access Frequency-Hopping System. *IEEE Trans. on Comm.*, v.COM-32, n.5, p.589-597, May, 1984.

[28] VADJA, I. Code sequences for frequency-hopping multiple-access systems. *IEEE Trans. on Comm.*, v.43, n.10, p.2553-2554, Oct., 1995.

[29] TIMOR, U. Improved Decoding Scheme for Frequency-Hopped Multilevel FSK System. *B.S.T.J.*, v.59, n.10, p.1839-1855, Dec., 1980.

[30] TIMOR, U. Multistage Decoding of Frequency-Hopped FSK System. *B.S.T.J.*, v.60, n.4, p.471-483, Apr., 1981.

[31] FIEBIG, C. G. Iterative interference cancellation for FFH/MFSKMA systems. *IEE Proc. on Comm.*, v.143, n.6, Dec., 1996.

[32] SPARSO, J., JORGENSEN, H. N., PAASKE, E., PEDERSEN, S., RUBER-PETERSEN, T. An area-efficient topology for VLSI Viterbi decoders and other shuffle-exchange type structures. *IEEE J. Sol. Circ.*, v.26, n.2, p.90-97, Feb. 1991.

[33] MICHELSON, A. M., LEVESQUE, A. H. *Error-Control Techniques for Digital Communication.* USA: Wiley, 1985.

[34] FORNEY JR., G. D. Coset Codes - Part II: binary patterns and related codes. *IEEE Trans. Information Theory*, v.IT-34, p.1152-1187, part II, Sep., 1988.

[35] NUNES, L. C. Comunicação privada.

APÊNDICE A

Verificação de que o endereçamento dado pela equação 3-5 é um código RS.

Seja $\overline{c} = (u(1), u(\alpha), \cdots, u(\alpha^{N-1})) = (c_0, c_1, \cdots, c_{n-1})$, uma palavra-código de um código *Reed-Solomon*, onde $u(z) = \sum_{i=0}^{K-1} u_i \cdot z^i$ e $\overline{u} = (u_0, u_1, \cdots, u_{K-1})$ para $u_i \in$ *GF(p)*. No capítulo 1, foi definido que a palavra-código $\overline{c}$ pertence a um código BCH, por verificar que $c(x) = \sum_{i=0}^{N-1} c_i \cdot x^i$ tem $\alpha, \alpha^2, \cdots, \alpha^{D-1}$ como zeros.

Portanto, fazendo K=k, $\alpha = \beta$, $u_i = x_i$ para i de 0 até k-1, vem que cada componente de $\overline{c}$ do código RS é dado pelo conjunto de equações:

$$
\begin{aligned}
c_0 &= x_0 + x_1 + x_2 + \cdots + x_{k-1} \\
c_1 &= x_0 + x_1 \cdot \beta + x_2 \cdot \beta^2 + \cdots + x_{k-1} \cdot \beta^{k-1} \\
&\vdots \\
c_{n-1} &= x_0 + x_1 \cdot \beta^{(n-1)} + x_2 \cdot \beta^{(n-1)\cdot 2} + \cdots + x_{k-1} \cdot \beta^{(n-1)\cdot(k-1)}
\end{aligned}
$$

De outra forma as equações dada acima, podem representar plenamente a equação 3-5, dada por:

$$
\overline{c} = (c_0, c_1, \cdots, c_{n-1}) = x_0 \cdot \overline{1} + x_1 \cdot \overline{\beta}^{(1)} + x_2 \cdot \overline{\beta}^{(2)} + x_3 \cdot \overline{\beta}^{(3)} + \cdots + x_{k-1} \cdot \overline{\beta}^{(k-1)};
$$
$$
x_i \in GF(Q),\ i = 0,\ 1,\ \cdots, k-1
$$

onde $\overline{\beta}^{(i)} = (1, \beta^i, \beta^{2i}, \cdots, \beta^{(n-1)i})$.

APÊNDICE B

Relação dos principais programas desenvolvidos para a realização deste trabalho.

Arquivo: endepbas.c (i)

Responsável pela geração dos padrões entrada-saída da RNA para o treinamento.

Sistema Q31L15, com o código *Reed-Müller* (16,5,8) para os padrões de saída.

```
#include <apiDOS.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <io.h>
#include <process.h>
#include <conio.h>
#include <math.h>
#define TAMANHO_DO_BUFFER 512
#define NUMERO_DE_PARAMETROS 17
unsigned char Buffer[TAMANHO_DO_BUFFER];
MTX MatrizAux1,MatrizAux2,Mensagem,Enderecos,Vetor,Matriz1,Matriz2,Padrao,Padroes,Sai;
double Q,NUSistema;
int Mensagem1,Mensagem2,Mensagem3,Mensagem4,Mensagem5,Mensagem6,Mensagem7,EP1,EP2,L,k;
int Endereco1,Endereco2,Endereco3,Endereco4,Endereco5,Endereco6,Endereco7;
char *Resultado;
double Aux1 [1] [256];
long NumPad;
FILE *PontArq1;
int i,j,k,NColunas,NLinhas;
char AuxString[50];
double Numero_Sorteado;
struct BJ {
  double m[4];
};
double CParidade[9][4];
struct BJ Bj[4][2];
/* ---------------------------------------------------------------------------------------------------------------------*/
void Inicializa_Formacao_do_Codigo()
{
 /* Codigo de paridade */
 CParidade[1][0]=0; CParidade[1][1]=0; CParidade[1][2]=0; CParidade[1][3]=0;
 CParidade[2][0]=0; CParidade[2][1]=0; CParidade[2][2]=1; CParidade[2][3]=1;
 CParidade[3][0]=0; CParidade[3][1]=1; CParidade[3][2]=0; CParidade[3][3]=1;
 CParidade[4][0]=1; CParidade[4][1]=0; CParidade[4][2]=0; CParidade[4][3]=1;
 CParidade[5][0]=1; CParidade[5][1]=0; CParidade[5][2]=1; CParidade[5][3]=0;
 CParidade[6][0]=1; CParidade[6][1]=1; CParidade[6][2]=0; CParidade[6][3]=0;
 CParidade[7][0]=1; CParidade[7][1]=1; CParidade[7][2]=1; CParidade[7][3]=1;
 CParidade[8][0]=0; CParidade[8][1]=1; CParidade[8][2]=1; CParidade[8][3]=0;
 /* Codigos Bj's para formacao do codigo (16,5.8) */
 Bj[0][0].m[0]=-1; Bj[0][0].m[1]=-1; Bj[0][0].m[2]=-1; Bj[0][0].m[3]=-1;
 Bj[0][1].m[0]=1; Bj[0][1].m[1]=1; Bj[0][1].m[2]=1; Bj[0][1].m[3]=1;
 Bj[1][0].m[0]=-1; Bj[1][0].m[1]=1; Bj[1][0].m[2]=-1; Bj[1][0].m[3]=1;
 Bj[1][1].m[0]=1; Bj[1][1].m[1]=-1; Bj[1][1].m[2]=1; Bj[1][1].m[3]=-1;
 Bj[2][0].m[0]=-1; Bj[2][0].m[1]=-1; Bj[2][0].m[2]=1; Bj[2][0].m[3]=1;
 Bj[2][1].m[0]=1; Bj[2][1].m[1]=1; Bj[2][1].m[2]=-1; Bj[2][1].m[3]=-1;
 Bj[3][0].m[0]=-1; Bj[3][0].m[1]=1; Bj[3][0].m[2]=1; Bj[3][0].m[3]=-1;
 Bj[3][1].m[0]=1; Bj[3][1].m[1]=-1; Bj[3][1].m[2]=-1; Bj[3][1].m[3]=1;
}
/* ---------------------------------------------------------------------------------------------------------------------*/
void Tela()
{
 clrscr();
 gotoxy(5,3); printf("Universidade Estadual de Campinas");
 gotoxy(5,4); printf("Faculdade de Engenharia Elétrica e de Computação - FEEC");
 gotoxy(5,5); printf("Cidade:");
```

```
 gotoxy(5,6); printf("Departamento:");
 gotoxy(5,7); printf("Tópicos:");
 gotoxy(5,8); printf("Orientador:");
 gotoxy(5,9); printf("Estudante:");
 gotoxy (5,23);
 printf("CopyRight(C) 1997 by Getúlio Júnior");
 gotoxy(13,5); printf("Campinas");
 gotoxy(19,6); printf("Departamento de Comunicações");
 gotoxy(14,7); printf("Protótipos de Redes Neurais - Geração de Padrões ");
 gotoxy(17,8); printf("Jaime Portugheis");
 gotoxy(16,9); printf("Getúlio A. de Deus Júnior");
 }
/* ------------------------------------------------------------------------------------------------------------------------------------------------*/
double Combinacao(int a, int b)
{
int i,x,y,z;
  x=1;
  y=1;
  z=1;
  for (i=2;i<=a;i++)
     x=x*i;
  for (i=2;i<=b;i++)
     y=y*i;
  for (i=2;i<=(a-b);i++)
     z=z*i;
  return (x/(y*z));
}
/* ------------------------------------------------------------------------------------------------------------------------------------------------*/
void Elemento_Primitivos()
/* O elemento Primitivo EP2 não será  usado */
{
int Mal,i,j,flag,contador;
double a1,a2;
 Tela();
 contador=0;
 for (k=1;k<=(Q-1);k++) {
    for (j=1;j<=(Q-1);j++) {
                    Aux1[0][j-1]=fmod(pow(k,j),Q);
    }
    flag=1;
    for (j=0;j<=(Q-3);j++) {
     if (Aux1[0][j]==1) {
                    flag=0;
                    if(contador==0) {
  EP1=-1;
                    }
                    else {
                EP2=-1;
                    }
     }
    }
    if (((flag==1)&(Aux1[0][Q-2]==1))&(contador<=2)) {
     contador=contador+1;
     if (contador==1) {
                    EP1=k;
                    gotoxy (5,12); printf("Elemento Primitivo 1 =%d",EP1);
     }
     else {
                    if (contador==2) {
     EP2=k;
     k=Q;
     gotoxy (5,14); printf("Elemento Primitivo 2 =%d",EP2);
                    }
     }
    }
 }
 gotoxy (5,21);
 printf("Cálculo Elemento Primimitivo - Pressione Alguma Tecla [ ]");
 gotoxy(60,21);
 getch();
}
/* ------------------------------------------------------------------------------------------------------------------------------------------------*/
void Matriz_de_Enderecos()
{
int j,i,Mal;
 CreateMatrix0(&Enderecos,Q,L,1,1);
 for (i=1;i<=L;i++) {
    Enderecos._[1][i]=0;
    for (j=1;j<=(Q-1);j++) {
     Enderecos._[j+1][i]=fmod(j*pow(EP1,i-1),Q);
    }
 }
 Tela();
 ShowMatrix(&Enderecos,5,12);
 Mal=SaveMatrix("c:\\getulio\\sincrono\\q31115rm\\endepbas\\eq31115",&Enderecos,MTX_DOUBLE);
 if (!Mal) { gotoxy (5,22); printf("Salvar Arquivo OK: e1q31115.mtx - Pressione Alguma Tecla [ ]"); }
    else { gotoxy (5,22); printf("Erro: %d",Mal); }
 gotoxy (5,21);
 printf("Matriz de Endereços");
 gotoxy(62,22);
```

```
 getch();
}
/* ---------------------------------------------------------------------------------------------------------------------------------------*/
void Parametros()
{
FILE *Arquivo_Corrente;
static char *Chave[NUMERO_DE_PARAMETROS]={"Campo de Galois=","L=","NUSistema","Mensagem1=",
"Mensagem2=","Mensagem3=","Mensagem4=","Mensagem5=","Mensagem6=","Mensagem7=","Endereco1=","Endereco2=","Endereco3=","Endereco4=","Endereco5
=","Endereco6=","Endereco7="};
char dflt[]="Impossível",
char string[] = "Valores Iniciais";
char fn[40];
char msg[80];
int lc;
char *pret;
int i,Vl,Vc,Valor_Contado;
char *Ponteiro_Fim;
 Tela();
 gotoxy (5,21);
 printf("Parâmetros do Sistema - Pressione Alguma Tecla [ ]");
 Vc=5; Vl=12;
 Valor_Contado=1;
 if ((Arquivo_Corrente=fopen("c:\\getulio\\sincrono\\q31115rm\\endepbas\\endepbas.mlp", "r"))==NULL) {
    perror("Não posso abrir este arquivo! [ apiDOS ]");
    exit(1);
 }
 for (i=0;i<NUMERO_DE_PARAMETROS;i++) {
    msg[0]=0x00;
    for (lc=1;;lc++) {
     if (fgets(msg, 80, Arquivo_Corrente)==NULL) {
                    if (ferror(Arquivo_Corrente))
    {
    perror("Não posso abrir este arquivo! [ apiDOS ]");
    fclose(Arquivo_Corrente);
    exit(2); }
    else
     Resultado=dflt;
     break; /* NÆo Posso Encontrar a chave! */
      } /* if */
      pret=strstr(msg, Chave[i]);
      if ((pret) && (pret==msg)) { /* A Chave "mente" nesta linha! */
      Resultado=pret+strlen(Chave[i]);
      Resultado[strlen(Resultado)]=0x00;
      break ; }
      } /* for */
                    gotoxy(Vc,Vl);
                    printf("%s%s",Chave[i],Resultado);
                    if(Valor_Contado>=7) {
    Vc=Vc+20;
    Valor_Contado=1;
    Vl=11;
                    }
                    Valor_Contado++;
                    Vl++;
     switch (i) {
      case 0: Q = strtod(Resultado,&Ponteiro_Fim); break;
      case 1: L = strtod(Resultado,&Ponteiro_Fim); break;
      case 2: NUSistema = strtod(Resultado,&Ponteiro_Fim); break;
      case 3: Mensagem1 = strtod(Resultado,&Ponteiro_Fim); break;
      case 4: Mensagem2 = strtod(Resultado,&Ponteiro_Fim); break;
      case 5: Mensagem3 = strtod(Resultado,&Ponteiro_Fim); break;
      case 6: Mensagem4 = strtod(Resultado,&Ponteiro_Fim); break;
      case 7: Mensagem5 = strtod(Resultado,&Ponteiro_Fim); break;
      case 8: Mensagem6 = strtod(Resultado,&Ponteiro_Fim); break;
      case 9: Mensagem7 = strtod(Resultado,&Ponteiro_Fim); break;
      case 10: Endereco1 = strtod(Resultado,&Ponteiro_Fim); break;
      case 11: Endereco2= strtod(Resultado,&Ponteiro_Fim); break;
      case 12: Endereco3 = strtod(Resultado,&Ponteiro_Fim); break;
      case 13: Endereco4 = strtod(Resultado,&Ponteiro_Fim); break;
      case 14: Endereco5 = strtod(Resultado,&Ponteiro_Fim); break;
      case 15: Endereco6 = strtod(Resultado,&Ponteiro_Fim); break;
      case 16: Endereco7 = strtod(Resultado,&Ponteiro_Fim); break;
    } /* switch */
 } /* for */
 /* Número de Padrões */
 /* NumPad=Q+Combinacao(Q,2); */
 NumPad=Q;
 fclose(Arquivo_Corrente);
 gotoxy(53,21);
 getch();
}
/* ---------------------------------------------------------------------------------------------------------------------------------------*/
double Numero_Aleatorio(long Valor_Inicial, long Valor_Final)
{
 int m;
 m=((double)rand()/(double)RAND_MAX)*(Valor_Final-Valor_Inicial)+Valor_Inicial;
/* if (m>0) m=1;
    else m=-1; */
 return (m);
}
```

```
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Padrao_tx(int ender,int usuario)
{
int s,d;
  CreateMatrix0(&MatrizAux1,Q,L,1,1);
  SetMatrix(&MatrizAux1,-1);
  for(s=1;s<=(L);s++) {
   for(d=1;d<=Q;d++) {
     MatrizAux1._[Q-fmod((usuario+Enderecos._[ender][s]),Q)][s]=1;
   }
  }
  for(s=1;s<=(L);s++) {
    for(d=1;d<=Q;d++) {
       if (Padrao._[1][s+(d-1)*(L)] < 0) {
     Padrao._[1][s+(d-1)*(L)]=MatrizAux1._[d][s];
       }
     }
  }
 DestroyMatrix(&MatrizAux1);
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Padrao_rx(int ender)
{
int pos,num,s,d,i,flagaux;
  CreateMatrix0(&MatrizAux2,Q,(L),1,1);
  CreateMatrix0(&MatrizAux1,Q,(L),1,1);
  SetMatrix(&MatrizAux1,-1);
  for(s=1;s<=(L);s++) {
    for(d=1;d<=Q;d++) {
   MatrizAux1._[d][s]=Padrao._[1][s+(d-1)*(L)];
    }
  }
  SetMatrix(&MatrizAux2,-1);
  for(s=1;s<=(L);s++) {
     for(d=1;d<=Q;d++) {
      if (MatrizAux1._[d][s]>0) {
   num=((Q-d)-Enderecos._[ender][s]); //ender+1
      if (num<0) {
   num=num+Q; }
     pos=(Q-num);
     if ((pos>0)&&(pos<=Q)) {
   MatrizAux2._[pos][s]=1; }
      }
   } //for
  } //for
  for(s=1;s<=(L);s++) {
    for(d=1;d<=Q;d++) {
       Padrao._[1][s+(d-1)*(L)]=MatrizAux2._[d][s];
    }
  }
  DestroyMatrix(&MatrizAux1);
  DestroyMatrix(&MatrizAux2);
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Cria_dados_arquivo_bin()
{
FILE *fptr1,*fptr2;
int d,s,contador,contadoraux,i,j,k,iaux,jaux,kaux,quativos,l,n_linhas,b,c,NNSai,o,p,f;
float padrao[466],saida[17],conta_padroes;
float saida_aux[50][17],contador_sai,valor, somador;

CreateMatrix0(&Padrao,1,Q*L,1,1);
 if((fptr1=fopen("c:\\getulio\\sincrono\\q31115rm\\endepbas\\xistrein.bin","wb"))==NULL) {
    printf("NÆo posso abrir o arquivo: xistrein.bin");
    exit(1);
 }
 if((fptr2=fopen("c:\\getulio\\sincrono\\q31115rm\\endepbas\\ysaidas.bin","wb"))==NULL) {
    printf("NÆo posso abrir o arquivo: ysaidas.bin");
    exit(1);
 }
 conta_padroes=0;
/* Padräes Básicos */
    contador=0;
    for (k=1;k<=8;k++) {
     for (l=0;l<=3;l++) {
                    for (i=0;i<=3;i++) {
                                for (j=0;j<=3;j++) {
                                 saida[(j+1)+i*4]=Bj[l][CParidade[k][i]].m[j];
                                }
                    }
                    if((k==1)&&(l==0)) {
                                                /* nao faz nada */
                    } else {
                                fwrite(saida,sizeof(saida),1,fptr2);
                                contador=contador+1;
                    }
    }
 }
 clrscr();
 printf("contador: %d",contador);
```

```
  getch();
  for(k=1;k<=Q;k++) {
    SetMatrix(&Padrao,-1);
    Padrao_tx(k,0);
    for(iaux=1;iaux<=Q*L;iaux++) {
                padrao[iaux]=Padrao._[1][iaux];
    }
    conta_padroes=conta_padroes+1;
    fwrite(padrao,sizeof(padrao),1,fptr1);
  }
  fclose(fptr1);
  fclose(fptr2);
  Tela();
  gotoxy(5,12);
  printf("Número de Padrões %3.0lf",conta_padroes);
  gotoxy(5,21);
  printf("Arquivos Gravados !");
  gotoxy(5,22);
  printf("Tecle algo [ ]");
  gotoxy(17,22);
  DestroyMatrix(&Padrao);
  getch();
}
/* ---------------------------------------------------------------------------------------------------------------------------------------*/
void Cria_matrizes_de_enderecos_bin()
{
FILE *fptr;
double aux1[31+1], aux2[31+1];
clrscr();
printf("Matrizes de enderecos *.bin foram criadas");
if((fptr=fopen("c:\\getulio\\sincrono\\q31l15rm\\endepbas\\e1q31l15.bin","wb"))==NULL) {
    printf("Não posso abrir o arquivo: eq31l15.bin");
    exit(1);
 }
 /* transferencia dos dados para o arquivo */
 for (i=1;i<=L;i++) {
aux1[1]=Enderecos._[1][i];
    for (j=1;j<=(Q-1);j++) {
                aux1[j+1]=Enderecos._[j+1][i];
    }
    fwrite(aux1,sizeof(aux1),1,fptr);
 }
 fclose(fptr);
 getch();
}
/* ---------------------------------------------------------------------------------------------------------------------------------------*/
main()
{
 randomize();
 Parametros();
 Elemento_Primitivos();
 Matriz_de_Enderecos();
 Inicializa_Formacao_do_Codigo();
 Cria_dados_arquivo_bin();
 Cria_matrizes_de_enderecos_bin();
 DestroyMatrix(&Enderecos);
}
/* ---------------------------------------------------------------------------------------------------------------------------------------*/
```

Arquivo: celular.c (i)

Responsável pelo treinamento da RNA e cálculo das probabiliades de erro da RNA.

Sistema Q31L15, com o código *Reed-Müller* (16,5,8) para os padrões de saída.

```
#include <string.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <math.h>
#define NUMERO_DE_PARAMETROS17
#define NHN 50
#define NON 16
#define NIN 465
#define Q 31
#define L 15
FILE *fptr1,*fptr2,*fptr3;
int pad, kaux,contador,i,j,k,iaux,fs,posicao;
float maior,diferenca_outneuron[17],saida[17],padrao[466]; /* Mesmo Valor da Geração dos Padrões */
int ad,d,s,q;
FILE *fptr;
int u,i,j,ncols,nrows;
```

```
char string[25];
double erro_total,flag,np,nin,nhn,non,lr,momento,tolerance,minerror,nme,beta,
value,fullerror,usuario_amarrado,numero_usuarios_ativos,modo,numero_de_repeticoes;
char *activfunction, *initmethod;
char *valuestr, *printresult, *result;
double m0[17],m1[17],M1[17],M0[17],cparidade[5],cpar[5];
float zin[2][NHN+1], yin[2][NON+1], z[2][NHN+1], y[2][NON+1], deltak[2][NON+1];
float deltaw[NON+1][NHN+2], deltain[2][NHN+1], deltaj[2][NHN+1], wnew[NON+1][NHN+2];
float hidden[2][NHN+1], outneuron[2][NON+1], testpattern[2][NIN+1], adress[Q+1][L+1];
float matrixaux[Q+1][L+1], matrixauxii[Q+1][L+1], x[2][NIN+1], yt[2][NON+1];
float v[NHN+1][NIN+2], w[NON+1][NHN+2], vnew[NHN+1][NIN+2], deltav[NHN+1][NIN+2];
int mnum[49],musuarios[32],mmensagens[32],usuariorec[1],posmusuariossorteada[1];
double srv,erv;
int patt,patt2,countnumpoint;
int mensagemdecodificada,numlinha[49],nlinhascompletas,contauns;
long numacertos,numacertoslm,numvezes,n1,epocas;
time_t first, second;
float na,nv1;
double mprobabilidade[32][32],mprobabilidadelm[32][32],macertos[32][32],macertoslm[32][32],nquepodesersorteado[32],numlinhascompletas,fator_correcao;
float saida_aux[50][17];
struct BJ {
  double m[4];
};
double CParidade[9][4];
struct BJ Bj[4][2];
/* ------------------------------------------------------------------------------------------------------------------------------------------------------*/
void Inicializa_Formacao_do_Codigo()
{
  /* Codigo de paridade */
  CParidade[1][0]=0; CParidade[1][1]=0; CParidade[1][2]=0; CParidade[1][3]=0;
  CParidade[2][0]=0; CParidade[2][1]=0; CParidade[2][2]=1; CParidade[2][3]=1;
  CParidade[3][0]=0; CParidade[3][1]=1; CParidade[3][2]=0; CParidade[3][3]=1;
  CParidade[4][0]=1; CParidade[4][1]=0; CParidade[4][2]=0; CParidade[4][3]=1;
  CParidade[5][0]=1; CParidade[5][1]=0; CParidade[5][2]=1; CParidade[5][3]=0;
  CParidade[6][0]=1; CParidade[6][1]=1; CParidade[6][2]=0; CParidade[6][3]=0;
  CParidade[7][0]=1; CParidade[7][1]=1; CParidade[7][2]=1; CParidade[7][3]=1;
  CParidade[8][0]=0; CParidade[8][1]=1; CParidade[8][2]=1; CParidade[8][3]=0;
  /* Codigos Bj's para formacao do codigo (16,5.8) */
  Bj[0][0].m[0]=-1; Bj[0][0].m[1]=-1;  Bj[0][0].m[2]=-1;  Bj[0][0].m[3]=-1;
  Bj[0][1].m[0]=1; Bj[0][1].m[1]=1;  Bj[0][1].m[2]=1;  Bj[0][1].m[3]=1;
  Bj[1][0].m[0]=-1; Bj[1][0].m[1]=1;  Bj[1][0].m[2]=-1;  Bj[1][0].m[3]=1;
  Bj[1][1].m[0]=1; Bj[1][1].m[1]=-1;  Bj[1][1].m[2]=1;  Bj[1][1].m[3]=-1;
  Bj[2][0].m[0]=-1; Bj[2][0].m[1]=-1;  Bj[2][0].m[2]=1;  Bj[2][0].m[3]=1;
  Bj[2][1].m[0]=1; Bj[2][1].m[1]=1;  Bj[2][1].m[2]=-1;  Bj[2][1].m[3]=-1;
  Bj[3][0].m[0]=-1; Bj[3][0].m[1]=1;  Bj[3][0].m[2]=1;  Bj[3][0].m[3]=-1;
  Bj[3][1].m[0]=1; Bj[3][1].m[1]=-1;  Bj[3][1].m[2]=-1;  Bj[3][1].m[3]=1;
}
/* ------------------------------------------------------------------------------------------------------------------------------------------------------*/
float Sorteia_Numero_Aleatorio(double fStart,double fEnd)
{
 float i;
 i=((double)rand()/(double)RAND_MAX)*(fEnd-fStart)+fStart;
 return(i);
}
/* ------------------------------------------------------------------------------------------------------------------------------------------------------*/
double sigmoide_bipolar (double x)
{
double a, b;
 a=1;
 b=1;
 if (x>(75)) x=75;
 else  if (x<-75) x=-75;
 return (((2*a)/(1+exp(-b*x)))-a);
}
/* ------------------------------------------------------------------------------------------------------------------------------------------------------*/
double derivada_sigmoide_bipolar (double x)
{
double a, b;
 a=1;
 b=1;
 if (x>75) x=75;
 else  if (x<-75) x=-75;
 return ((2*a*b*exp(-b*x))/pow(1+exp(-b*x),2));
}
/* ------------------------------------------------------------------------------------------------------------------------------------------------------*/
void Tela()
{
 clrscr();
 gotoxy(5,3); printf("Universidade Estadual de Campinas");
 gotoxy(5,4); printf("Faculdade de Engenharia Elétrica e de Computação");
 gotoxy(5,5); printf("Cidade:");
 gotoxy(5,6); printf("Departamento:");
 gotoxy(5,7); printf("Tópicos:");
 gotoxy(5,8); printf("Orientador:");
 gotoxy(5,9); printf("Pós-graduando:");
 gotoxy(5,20); printf("Mensagem");
 gotoxy(13,5); printf("Campinas");
 gotoxy(19,6); printf("Departamento de Comunicações");
 gotoxy(14,7); printf("Protótipos de Redes Neurais");
 gotoxy(17,8); printf("Jaime Portugheis");
```

```
 gotoxy(20,9); printf("Getúlio A. de Deus Júnior");
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Parametros()
{
    FILE *stream;
    static char *key[NUMERO_DE_PARAMETROS]={"número de padrões=","número de neurônios de entrada =",
                        "número de neurônios de entrada =","número de neurônios de saída =",
                        "taxa de aprendizado =","momento","tolerância=","erro mínimo =",
                        "valor aleatório inicial=","valor aleatório final=","número máximo de épocas =",
                        "função de ativação =","método de inicialização =","número de usuários ativos =",
                        "modo=","usuários amarrado =","número de repetições ="};
    char dflt[]="missing";
    char string[] = "initial values";
    char fn[40];
    char msg[80];
    int lc;
    char *pret;
    int i,lv,cv,countervalue;
    char *endprt;
    Tela();
    gotoxy (5,22);
    printf("Algoritmo Backpropagation - CopyRight(C) 1997, by EletroGet");
    gotoxy (5,23);
    printf("Parâmetros do Sistema.      Pressione Algo... [ ]");
    cv=5; lv=11;
    countervalue=1;
    if ((stream=fopen("c:\\getulio\\sincrono\\q31115rm\\backlog\\celular.mlp", "r"))==NULL) {
     perror("Unable to open that file! [ apiDOS ]");
     exit(1); }
    for (i=0;i<NUMERO_DE_PARAMETROS;i++) {
     msg[0]=0x00;
     for (lc=1;;lc++) {
                        if (fgets(msg, 80, stream)==NULL) {
    if (ferror(stream))
     {
     perror("I can't load that file! [ apiDOS ]") ;
     fclose(stream);
     exit(2); }
     else
      result=dflt;
      break; // I didn't find the key!
                        }
      pret=strstr(msg, key[i]);
      if ((pret) && (pret==msg)) { // The key lies on this line!
     result=pret+strlen(key[i]);
     result[strlen(result)-1]=0x00;
     break ; }
     }
                        gotoxy(cv,lv);
                        printf("%s%s",key[i],result);
                        if(countervalue>=9) {
    cv=cv+35;
    countervalue=1;
    lv=10;
                        }
                        countervalue++;
                        lv++;
     switch (i) {
      case 0: np = strtod(result,&endprt); break;
      case 1: nin = strtod(result,&endprt); break;
      case 2: nhn = strtod(result,&endprt); break;
      case 3: non = strtod(result,&endprt); break;
      case 4: lr = strtod(result,&endprt); break;
      case 5: momento = strtod(result,&endprt); break;
      case 6: tolerance = strtod(result,&endprt); break;
      case 7: minerror = strtod(result,&endprt); break;
      case 8: srv = strtod(result,&endprt); break;
      case 9: erv = strtod(result,&endprt); break;
      case 10: nme = strtod(result,&endprt); break;
      case 11: activfunction = result; break;
      case 12: initmethod = result; break;
      case 13: numero_usuarios_ativos = strtod(result,&endprt); break;
      case 14: modo = strtod(result,&endprt); break;
      case 15: usuario_amarrado = strtod(result,&endprt); break;
      case 16: numero_de_repeticoes = strtod(result,&endprt); break;
     }
    }
    flag=0;
    result="Nguyen-Widrow";
    if (*initmethod==*result) {
     flag=1;
    }
    fclose(stream);
    countnumpoint = nme/100; // maximum 100 points for graph
    gotoxy(52,23);
    getch();
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Sorteia_Numeros (int maux[49],int qnum,double numinicial,double numfinal,int repeticao)
```

```
{
// numvezes = 1,000,000
// qnum: no maximo = NNEn
// mnum: matriz de numeros sorteados de 1 ate NNEn
// repeticao = 0 -> nÆo pode repetir os n£meros sorteados
// repeticao = 1 -> pode repetir os n£meros sorteados
double num1,num2;
double integer;
long int cont0=0,cont1=0,cont2=0,cont3=0,cont4=0,cont5=0,cont6=0,cont7=0;
long int flag,flagaux,g,j;
 if (repeticao==1) {
    for(g=1;g<=qnum;g++) {
                    num2=Sorteia_Numero_Aleatorio(numinicial,numfinal+1);
                    num1=modf(num2,&integer);
                    maux[g]=integer;
                    if (maux[g]==(numfinal+1)) {
    maux[g]=numfinal;
                    }
    } //for
 } // if
 else {
    if (repeticao==0) {
   if (modo==1) {
    maux[1]=usuario_amarrado;
   }
   else {
    num2=Sorteia_Numero_Aleatorio(numinicial,numfinal+1);
    num1=modf(num2,&integer);
    maux[1]=integer;
    if (maux[1]==(numfinal+1)) {
                    maux[1]=numfinal;
    }
   }
   for(g=2;g<=qnum;g++) {
                flag=0;
                while(flag==0) {
                 num2=Sorteia_Numero_Aleatorio(numinicial,numfinal+1);
                 num1=modf(num2,&integer);
                 maux[g]=integer;
                 flag=1;
                 if (maux[g]==(numfinal+1)) {
                    maux[g]=numfinal;
                 } //if
                 for(j=1;j<=(g-1);j++) {
                    if (maux[g]==maux[j]) {
                 flag=0;
                    }
                 } //for
                } //while
   } //for
    } // if
    else {
                printf("erro no parametro repeticao");
    }
 } // else
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Atribui_valor_aleatorios_para_pesos()
{
 int i, j;
 for (i=1;i<=NHN+1;i++) {
    for (j=1;j<=NIN+2;j++) {
                v[i][j]=Sorteia_Numero_Aleatorio(srv,erv);
    }
 }
 for (i=1;i<=NON+1;i++) {
    for (j=1;j<=NHN+2;j++) {
                w[i][j]=Sorteia_Numero_Aleatorio(srv,erv);
    }
 }
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Treina_padrao()
{
int i,j;
   for (i=0;i<=1;i++){
    for (j=0;j<=NIN ;j++){
                zin[i][j]=0;
    }
   }
   for (i=0;i<=1;i++){
    for (j=0;j<=NON ;j++){
                yin[i][j]=0;
    }
   }
   for (i=0;i<=NHN;i++){
    for (j=0;j<=NIN+1 ;j++){
                deltav[i][j]=0;
    }
   }
```

```
    for (i=0;i<=NON;i++){
     for (j=0;j<=NHN+1 ;j++){
                    deltaw[i][j]=0;
     }
    }
    for (i=0;i<=1;i++){
     for (j=0;j<=NIN+1 ;j++){
                    deltain[i][j]=0;
     }
    }
    for (i=0;i<=1;i++){
     for (j=0;j<=NON ;j++){
                    deltak[i][j]=0;
     }
    }
    // Feedforward
    for (j=1;j<=nhn;j++) {
     for (i=1;i<=nin;i++) {
                                zin[1][j] = x[1][i]*v[j][i] + zin[1][j];
     }
     zin[1][j] = zin[1][j] + v[j][0];
    }
    for(j=1;j<=nhn;j++) {
     z[1][j]=sigmoide_bipolar((double)*&zin[1][j]);
    }
    for (j=1;j<=non;j++) {
     for (i=1;i<=nhn;i++) {
                                yin[1][j] = z[1][i]*w[j][i] + yin[1][j];
     }
     yin[1][j] = yin[1][j] + w[j][0];
    }
    for(j=1;j<=non;j++) {
     y[1][j]=sigmoide_bipolar ((double)*&yin[1][j]);
    }
    // Backpropgation of error
    for(i=1;i<=non;i++) {
     deltak[1][i]=((yt[1][i]-y[1][i]))*derivada_sigmoide_bipolar((double)*&yin[1][i]);
     for (j=1;j<=nhn;j++) {
                                deltaw[i][j]=lr*deltak[1][i]*z[1][j]+momento*deltaw[i][j];
     }
     deltaw[i][0]=lr*deltak[1][i];
    }
    for(i=1;i<=nhn;i++) {
     for (j=1;j<=non;j++) {
                    deltain[1][i]=deltak[1][j]*w[j][i]+deltain[1][i];
     }
    }
    for(i=1;i<=nhn;i++) {
     deltaj[1][i]=deltain[1][i]*derivada_sigmoide_bipolar((double)*&zin[1][i]);
     for (j=1;j<=nin;j++) {
                                deltav[i][j]=lr*deltaj[1][i]*x[1][j]+momento*deltav[i][j];
     }
     deltav[i][0]=lr*deltaj[1][i];
    }
    // Update weights and biases
    for (j=0;j<=nhn;j++) {
     for (i=1;i<=non;i++) {
                    wnew[i][j]=w[i][j]+deltaw[i][j];
     }
    }
    for (j=1;j<=nhn;j++) {
     for (i=0;i<=nin;i++) {
                    vnew[j][i]=v[j][i]+deltav[j][i];
     }
    }
    for (j=0;j<=nhn;j++) {
     for (i=1;i<=non;i++) {
                    w[i][j]=wnew[i][j];
     }
    }
    for (j=1;j<=nhn;j++) {
     for (i=0;i<=nin;i++) {
                    v[j][i]=vnew[j][i];
     }
    }
}
/* ------------------------------------------------------------------------------------------------------------------------------*/
void Aplica_padrao_na_rede()
{
 int i,j;
 double hiddenin,outputneuron;
 // Primeiro nivel
     for (j=1;j<=nhn;j++) {
                    hiddenin=0;
                    for (i=1;i<=nin;i++) {
    hiddenin = x[1][i]*v[j][i] + hiddenin;
                    }
                    hiddenin = hiddenin + v[j][0];
                    hidden[1][j]=sigmoide_bipolar(hiddenin);
     }
```

```
    // segundo nivel
    for (j=1;j<=non;j++) {
                outputneuron=0;
                for (i=1;i<=nhn;i++) {
                        outputneuron = hidden[1][i]*w[j][i] + outputneuron;
                }
                outputneuron = outputneuron + w[j][0];
                outneuron[1][j]=sigmoide_bipolar(outputneuron);
    }
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Aplica_padrao_na_rede_para_teste()
{
 int i,j;
 double hiddenin,outputneuron;
 // primeiro nivel
    for (j=1;j<=nhn;j++) {
                hiddenin=0;
                for (i=1;i<=nin;i++) {
                        hiddenin = testpattern[1][i]*v[j][i] + hiddenin;
                }
                hiddenin = hiddenin + v[j][0];
                hidden[1][j]=sigmoide_bipolar(hiddenin);
     }
     // segundo nivel
    for (j=1;j<=non;j++) {
                outputneuron=0;
                for (i=1;i<=nhn;i++) {
                  outputneuron = hidden[1][i]*w[j][i] + outputneuron;
                }
                outputneuron = outputneuron + w[j][0];
                outneuron[1][j]=sigmoide_bipolar(outputneuron);
    }
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Treinamento_principal()
{
int kaux, i, j, contador, pad, ordem[32], pad2, n1, n2;
    epocas=0;
    Tela();
    printf("\n\nAlgoritmo BackPropagation - CopyRight(C) 1997, by EletroGet");
    printf("\n\nTreinando...");
    do
    {
     Sorteia_Numeros(ordem,1,5,Q-5,0);
     for(pad=1;pad<=2;pad++) {
      if((fptr1=fopen("c:\\getulio\\sincrono\\q31115rm\\backlog\\xistrein.bin","rb"))==NULL) {
                                  printf("\n\nNão posso abrir o arquivo: xistrein.bin");
                                  exit(1);
                                 }
                                 if((fptr2=fopen("c:\\getulio\\sincrono\\q31115rm\\backlog\\ysaidas.bin","rb"))==NULL) {
                                  printf("\n\nNão posso abrir o arquivo: ysaidas.bin");
                                  exit(1);
                                 }
                                 if (pad==1) {
                                  n1=ordem[1];
                                  n2=np;
                                 } else  if (pad==2) {
                                                                        n1=1;
                                                                        n2=ordem[1]-1;
                                 }
                                 for(pad2=n1;pad2<=n2;pad2++) {
                                                fread(padrao,sizeof(padrao),1,fptr1);
                                                fread(saida,sizeof(saida),1,fptr2);
                                                for(kaux=1;kaux<=Q*L;kaux++) {
                                                 x[1][kaux]=padrao[kaux];
                                                }
                                                for(kaux=1;kaux<=non;kaux++) {
                                                 yt[1][kaux]=saida[kaux];
                                                }
                                                Treina_padrao();
                                 } /* pad 2 */
                                 fclose(fptr1);
                                 fclose(fptr2);
                } /* pad */
                erro_total=0;
                 if((fptr1=fopen("c:\\getulio\\sincrono\\q31115rm\\backlog\\xistrein.bin","rb"))==NULL) {
                  printf("Não posso abrir o arquivo: xistrein.bin");
                  exit(1);
                 }
                 if((fptr2=fopen("c:\\getulio\\sincrono\\q31115rm\\backlog\\ysaidas.bin","rb"))==NULL) {
                  printf("Não posso abrir o arquivo: ysaidas.bin");
                  exit(1);
                 }
                 for(pad=1;pad<=np;pad++) {
                  fread(padrao,sizeof(padrao),1,fptr1);
                  fread(saida,sizeof(saida),1,fptr2);
                  for(kaux=1;kaux<=Q*L;kaux++) {
                                x[1][kaux]=padrao[kaux];
                  }
```

```
                    for(kaux=1;kaux<=non;kaux++) {
                            yt[1][kaux]=saida[kaux];
                    }
                    Aplica_padrao_na_rede();
                    for(i=1;i<=non;i++) {
                            erro_total=erro_total + pow(yt[1][i]-(outneuron[1][i]),2);
                    }
                  }
                  fclose(fptr1);
                  fclose(fptr2);

                  erro_total=erro_total/2;
                  epocas=epocas+1;
                  gotoxy (25,15);
                  printf("Erro: %5.3e  Epocas: %5d ",erro_total,epocas);
    } while ((erro_total>minerror)&&(epocas<nme));
    printf("\n\n\nTreinamento Terminado! Tecle algo ... ");
    getch();
}
/* ---------------------------------------------------------------------------------------------------------------------------*/
void padrao_tx(int usuario, int mensagem)
{
int s,d;
    for(s=0;s<=Q;s++) {
     for(d=0;d<=L;d++) {
                  matrixaux[s][d]=-1.0;
     }
    }

    for(s=1;s<=(L);s++) {
                  matrixaux[Q-fmod((usuario+adress[mensagem][s]),Q)][s]=1;
    }
    for(d=1;d<=Q;d++) {
     for(s=1;s<=(L);s++) {
                  if (testpattern[1][s+(d-1)*(L)] < 0) {
                          testpattern[1][s+(d-1)*(L)]=matrixaux[d][s];
                  }
     }
    }
}
/* ---------------------------------------------------------------------------------------------------------------------------*/
void testa_palavra_codigo()
{
double v1,auxiliar_outneuron[32];
int somador,posicao,diferenca[5],menor;
 for(i=1;i<=non;i++) {
    m1[i]=pow(outneuron[1][i]-1,2);
 }
 for(i=1;i<=non;i++) {
    m0[i]=pow(outneuron[1][i]+1,2);
 }
 for(j=1;j<=4;j++) {
    M0[j]=0;
    for(i=((j-1)*4+1);i<=(4+(j-1)*4);i++) {
                  M0[j]=m0[i]+M0[j];
    }
 }
 for(j=1;j<=4;j++) {
    M1[j]=0;
    for(i=((j-1)*4+1);i<=(4+(j-1)*4);i++) {
                  M1[j]=m1[i]+M1[j];
    }
 }
 for(j=1;j<=4;j++) {
    if(ceil(M1[j])>=ceil(M0[j])) {
                  cpar[j]=1;
    } else {
                   cpar[j]=-1;
    }
 }
  clrscr();
    printf("M1: \n");
    for (k=1;k<=4;k++) {
     printf("%lf ",M1[k]);
    }
    printf("\n\n");
    printf("M0: \n");
    for (k=1;k<=4;k++) {
     printf("%lf ",M0[k]);
    }
    printf("\n\n");
    printf("Cpar: \n");
    for (k=1;k<=4;k++) {
     printf("%1.0lf ",cpar[k]);
    }
    printf("\n\n");

 somador=0;
 for(j=1;j<=4;j++) {
    if(cpar[j]>0) {
```

```
                somador=cpar[j]+somador;
    }
  }
  if (fmod(somador,2)!=0) {
    /* ocorreu um erro, nao e um codigo de paridade, devemos trocar
                por o bit menos confiavel */
    for(j=1;j<=4;j++) {
                diferenca[j]=abs(M0[j]-M1[j]);
    }
    menor=1000;
    for(j=1;j<=4;j++) {
                if (menor>diferenca[j]) {
                 menor=diferenca[j];
                 posicao=j;
                }
    }
    cpar[posicao]=cpar[posicao]*(-1);
  } /* if */
    printf("Cpar corrigido: \n");
    for (k=1;k<=4;k++) {
     printf("%1.0lf ",cpar[k]);
    }
    printf("\n\n");
    getch();
}
/* ------------------------------------------------------------------------------------------------------------------------------------------------------*/
void Classificacao_mv()
{
int cont, l, lg, i, k, posicao;
double saidabj[5][17],metrica[33],menor;
    Aplica_padrao_na_rede_para_teste();
    for (k=2;k<=32;k++) {
     metrica[k]=0;
     for (i=1;i<=non;i++) {
                 metrica[k]=(pow(outneuron[1][i]-saida_aux[k][i],2))+metrica[k];
     }
    }
    menor=1000;
    for (k=2;k<=32;k++) {
     if (menor>metrica[k]) {
                                menor=metrica[k];
                                posicao=k;
     }
    }
    cont=0;
    for (k=1;k<=non;k++) {
                if (saida_aux[posicao][k]==saida_aux[mmensagens[1]+1][k]) {
                                cont=cont+1;
                }
    }
    if(cont==non) {
                numacertos++;
    }
}
/* ------------------------------------------------------------------------------------------------------------------------------------------------------*/
void padrao_rx(int ad)
{
int ncoincidencias,contacoincidencias,pos,num,s,d,i,flagaux,kaux;
int Aux1[Q+1][L+1], Aux2[Q+1], contador;
    for(d=0;d<=Q;d++) {
     for(s=0;s<=L;s++) {
                matrixaux[d][s]=-1.0;
     }
    }
    for(d=1;d<=Q;d++) {
     for(s=1;s<=L;s++) {
                                matrixaux[d][s]=testpattern[1][s+(d-1)*(L)];
     }
    }
    for(d=0;d<=Q;d++) {
     for(s=0;s<=L;s++) {
                matrixauxii[d][s]=-1.0;
     }
    }
    for(s=1;s<=L;s++) {
     for(d=1;d<=Q;d++) {
                if (matrixaux[d][s]>0) {
                 num=((Q-d)-ad);
                 if (num<0) {
                                num=num+Q;
                 }
                 pos=(Q-num);
                 if ((pos>0)&&(pos<=Q)) {
                                matrixauxii[pos][s]=1; }
                }
     } //for
    } //for
    for(d=1;d<=Q;d++) {
     for(s=1;s<=L;s++) {
                testpattern[1][s+(d-1)*(L)]=matrixauxii[d][s];
```

```
    }
   }
   if((fptr3=fopen("c:\\getulio\\sincrono\\q31115rm\\backlog\\xistrein.bin","rb"))==NULL) {
    printf("Não posso abrir o arquivo: xistrein.bin");
    exit(1);
   }
   ncoincidencias=0;
   for(patt=1;patt<=np;patt++) {
    fread(padrao,sizeof(padrao),1,fptr3);
    for(kaux=1;kaux<=Q*L;kaux++) {
                x[1][kaux]=padrao[kaux];
    }
                contacoincidencias=0;
                for(s=0;s<=(Q-1);s++) {
                 for(d=1;d<=(L);d++) {
                        if((testpattern[1][d+(s*(L))]==x[1][d+(s*(L))])&&(x[1][d+(s*(L))]>0)) {
                                contacoincidencias=contacoincidencias+1;
                        }
                 } // for
                } // for
                if (contacoincidencias==L) {
                        ncoincidencias=ncoincidencias+1;
                }

   } // patt
   fclose(fptr3);

   if (ncoincidencias==1) {
    numacertoslm=numacertoslm+1;
   }
   Classificacao_mv();
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------------------*/
void probabilidade_de_erro(int quativos)
{
int l;
 Sorteia_Numeros(musuarios,quativos,0,30,0); // os usuarios ativos nao podem
                                                        // se repetirem
 Sorteia_Numeros(mmensagens,quativos,1,31,1); //as mensagens podem ser repetidas
 for(l=1;l<=quativos;l++) {
    padrao_tx(musuarios[l],mmensagens[l]);
 }
 Sorteia_Numeros(posmusuariossorteada,1,1,quativos,1);
 usuariorec[1]=musuarios[posmusuariossorteada[1]];
 padrao_rx(usuariorec[1]);
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------------------*/
void probabilidade_de_erro_amarrada(int quativos,int usuario_amarrado)
{
int l;
 Sorteia_Numeros(musuarios,quativos,0,Q-1,0), // os usuarios ativos nao podem
                                                        // se repetirem
 Sorteia_Numeros(mmensagens,quativos,1,Q,1); //as mensagens podem ser repetidas

 for(l=1;l<=quativos;l++) {
    padrao_tx(musuarios[l],mmensagens[l]);
 }
 padrao_rx(usuario_amarrado);
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------------------*/
void probabilidade_todos_amarrados()
{
int h,f,usuama[2];
double aux;
 clrscr();
 Tela();
 gotoxy (5,22);
 printf("Simulando ....");
    for(h=1;h<=Q;h++) {
                numacertos=0;
                numacertoslm=0;
                n1=1;
                numero_usuarios_ativos=h;
                Sorteia_Numeros(usuama,2,0,Q-1,1);
                usuario_amarrado=usuama[2];
                f=1;

                for(q=1;q<=n1;q++){
                 for(u=1;u<=numero_de_repeticoes;u++) {
                        for (i=1;i<=Q;i++) {
                                for (j=1;j<=L;j++) {
                                 testpattern[1][j+(i-1)*L]=-1;
                                }
                        }
                 probabilidade_de_erro_amarrada(numero_usuarios_ativos,usuario_amarrado);
                 } //for
                } //for
                macertos[f][h]=numacertos;
                macertoslm[f][h]=numacertoslm;
                nv1=numero_de_repeticoes;
```

```
                    mprobabilidadelm[f][h]=1-(macertoslm[f][h]/nv1);
                    mprobabilidade[f][h]=1-(macertos[f][h]/nv1);
    }//for
  clrscr();
  Tela();
  f=1;
  contador=0;
  gotoxy (1,11);
  printf("Pb. Erro Pal. (Numero de usuarios ativos no sistema: 1, 2, 3, 4, ..., %d)",Q);
  gotoxy (1,12);
  for(h=1;h<=Q;h++) {
     printf("%3.2e ",mprobabilidade[f][h]);
     contador=contador+1;
     if(contador>=8) {
                    printf("\n");
                    contador=0;
     }
  }
  gotoxy (1,17);
  printf("Pb. Erro Bit (Numero de usuarios ativos no sistema: 1, 2, 3, 4, ..., %d)",Q);
  gotoxy (1,18);
  contador=0;
  fator_correcao=(Q*1.0/(2.0*(Q-1)));
  for(h=1;h<=Q;h++) {
    printf("%3.2e ",mprobabilidade[f][h]*fator_correcao);
    contador=contador+1;
    if(contador>=8) {
                    printf("\n");
                    contador=0;
    }
  }
  gotoxy (5,23);
  printf("Rede Neural                    Pressione Algo... [ ]");
  gotoxy (71,23);
  getch();
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------------------*/
void regra_binaria()
{
int contador,k,l,somador;
double contador_sai,valor,v1;
   contador=0;
   for (k=1;k<=8;k++) {
    for (l=0;l<=3;l++) {
                    for (i=0;i<=3;i++) {
                                for (j=0;j<=3;j++) {
                                 if((k==1)&&(l==0)) {
                                                /* nao faz nada */
                                 } else {
                                                saida_aux[(l+1)+(k-1)*4][(j+1)+i*4]=Bj[l][CParidade[k][i]].m[j];
                                 }
                                }
                    }
    }
  }
  clrscr();
  for (k=1;k<=8;k++) {
    for (l=0;l<=3;l++) {
                  for (i=0;i<=3;i++) {
                   for (j=0;j<=3;j++) {
                                printf("%1.0lf ",saida_aux[(i+1)+(k-1)*4][(j+1)+i*4]);
                   }
                  }
                  printf("\n");
    }
  }
  getch();
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------------------*/
void Salvar_Matrizes_de_Pesos_Bin()
{
int i, j;
float aux1[NIN+2], aux2[NHN+2];
  Tela();
  if((fptr1=fopen("c:\\getulio\\sincrono\\q31115rm\\backlog\\wtreina.bin","wb"))==NULL) {
    printf("NÆo posso abrir o arquivo: wtreina.bin");
    exit(1);
  }
  if((fptr2=fopen("c:\\getulio\\sincrono\\q31115rm\\backlog\\vtreina.bin","wb"))==NULL) {
    printf("NÆo posso abrir o arquivo: vtreina.bin");
    exit(1);
  }
   /* transferencia dos dados para o arquivo */
   for(i=0;i<=NHN;i++) {
                  for(j=0;j<=NIN+1;j++) {
                                aux1[j]=v[i][j];
                  }
                  fwrite(aux1,sizeof(aux1),1,fptr2);
   }
   for(i=0;i<=NON;i++) {
```

```
                    for(j=0;j<=NHN+1;j++) {
                                aux2[j]=w[i][j];
                    }
                    fwrite(aux2,sizeof(aux2),1,fptr1);
    }
  fclose(fptr1);
  fclose(fptr2);
  printf("\n\n\Ok: Gravacao das matrizes de pesos");
  getch();
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Salvar_Pesos_Iniciais()
{
int i, j;
float aux1[NIN+2], aux2[NHN+2];
  Tela();
  if((fptr1=fopen("c:\\getulio\\sincrono\\q31115rm\\backlog\\winicial.bin","wb"))==NULL) {
     printf("NÆo posso abrir o arquivo: winicial.bin");
     exit(1);
  }
  if((fptr2=fopen("c:\\getulio\\sincrono\\q31115rm\\backlog\\vinicial.bin","wb"))==NULL) {
     printf("NÆo posso abrir o arquivo: vinicial.bin");
     exit(1);
  }
    /* transferencia dos dados para o arquivo */
    for(i=0;i<=NHN;i++) {
                    for(j=0;j<=NIN+1;j++) {
                                aux1[j]=v[i][j];
                    }
                    fwrite(aux1,sizeof(aux1),1,fptr2);
    }
    for(i=0;i<=NON;i++) {
                    for(j=0;j<=NHN+1;j++) {
                                aux2[j]=w[i][j];
                    }
                    fwrite(aux2,sizeof(aux2),1,fptr1);
    }
  fclose(fptr1);
  fclose(fptr2);
  printf("\n\n\Ok: Gravacao das matrizes de pesos iniciais");
  getch();
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Carregar_Matrizes_de_Pesos_Bin()
{
int i, j;
float aux1[NIN+2], aux2[NHN+2];
  Tela();
  if((fptr1=fopen("c:\\getulio\\sincrono\\q31115rm\\backlog\\wtreina.bin","rb"))==NULL) {
     printf("NÆo posso abrir o arquivo: wtreina.bin");
     exit(1);
  }
  if((fptr2=fopen("c:\\getulio\\sincrono\\q31115rm\\backlog\\vtreina.bin","rb"))==NULL) {
     printf("NÆo posso abrir o arquivo: vtreina.bin");
     exit(1);
  }
    /* transferencia dos dados para o arquivo */
    for(i=0;i<=NHN;i++) {
                    fread(aux1,sizeof(aux1),1,fptr2);
                    for(j=0;j<=NIN+1;j++) {
                                v[i][j]=aux1[j];
                    }

    }
    for(i=0;i<=NON;i++) {
                    fread(aux2,sizeof(aux2),1,fptr1);
                    for(j=0;j<=NHN+1;j++) {
                                w[i][j]=aux2[j];
                    }

    }
  fclose(fptr1);
  fclose(fptr2);
  printf("\n\n\Ok: Matrizes de pesos sinapticos carregadas");
  getch();

}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Carregar_Pesos_Iniciais()
{
int i, j;
float aux1[NIN+2], aux2[NHN+2];
  Tela();
  if((fptr1=fopen("c:\\getulio\\sincrono\\q31115rm\\backlog\\winicial.bin","rb"))==NULL) {
     printf("NÆo posso abrir o arquivo: winicial.bin");
     exit(1);
  }
  if((fptr2=fopen("c:\\getulio\\sincrono\\q31115rm\\backlog\\vinicial.bin","rb"))==NULL) {
     printf("NÆo posso abrir o arquivo: vinicial.bin");
     exit(1);
```

```
  }
    /* transferencia dos dados para o arquivo */
    for(i=0;i<=NHN;i++) {
                    fread(aux1,sizeof(aux1),1,fptr2);
                    for(j=0;j<=NIN+1;j++) {
                                    v[i][j]=aux1[j];
                    }

    }
    for(i=0;i<=NON;i++) {
                    fread(aux2,sizeof(aux2),1,fptr1);
                    for(j=0;j<=NHN+1;j++) {
                                    w[i][j]=aux2[j];
                    }

    }
  fclose(fptr1);
  fclose(fptr2);
  printf("\n\n\Ok: Matrizes de pesos sinapticos carregadas");
  getch();

}
/* ---------------------------------------------------------------------------------------------------------------------------------------------------------*/
void Carrega_matrizes_de_enderecos_bin()
{
FILE *fptr1;
double aux1[31+1], aux2[31+1];

    Tela();
    printf("Matrizes de enderecos *.bin foram lidas com sucesso");

  if((fptr1=fopen("c:\\getulio\\sincrono\\q31115rm\\backlog\\e1q31115.bin","rb"))==NULL) {
     printf("NÆo posso abrir o arquivo: eq31115.bin");
     exit(1);
  }
  /* transferencia dos dados para o arquivo */
  for (i=1;i<=L;i++) {
     fread(aux1,sizeof(aux1),1,fptr1);
     adress[1][i]=aux1[1];
     printf("%f ",adress[1][i]);
     for (j=1;j<=(Q-1);j++) {
                    adress[j+1][i]=aux1[j+1];
                    printf("%f ",adress[j+1][i]);
     }
     printf("\n");
  }
  fclose(fptr1);
  printf("\n\n\Ok: Matriz de enderecos carregadas");
  getch();
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------------------*/
// Programa principal
main()
{
  randomize();
  clrscr();
  Parametros();
  Inicializa_Formacao_do_Codigo();
  regra_binaria();
  Carregar_Matrizes_de_Pesos_Bin();
// Atribui_valor_aleatorios_para_pesos();
// Salvar_Pesos_Iniciais();
//  Carregar_Pesos_Iniciais();
/* aqui */
//  Tela();
//  Treinamento_principal();
//  Salvar_Matrizes_de_Pesos_Bin();
  Carrega_matrizes_de_enderecos_bin();
  probabilidade_todos_amarrados();
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------------------*/
```

Arquivo: endepbas.c (ii)

Responsável pela geração dos padrões entrada-saída da RNA para o treinamento. Sistema Q31L15, código binário com mais um bit para a verificação de paridade.

```
#include <apiDOS.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <io.h>
#include <process.h>
#include <conio.h>
#include <math.h>
#define TAMANHO_DO_BUFFER 512
#define NUMERO_DE_PARAMETROS 17
unsigned char Buffer[TAMANHO_DO_BUFFER];
MTX MatrizAux1,MatrizAux2,Mensagem,Enderecos,Vetor,Matriz1,Matriz2,Padrao,Padroes,Sai;
double Q,NUSistema;
int Mensagem1,Mensagem2,Mensagem3,Mensagem4,Mensagem5,Mensagem6,Mensagem7,EP1,EP2,L,k;
int Endereco1,Endereco2,Endereco3,Endereco4,Endereco5,Endereco6,Endereco7, i,j,NColunas,NLinhas;
char *Resultado, AuxString[50];
long NumPad;
FILE *PontArq1;
double Numero_Sorteado, Aux1 [1] [256];
/* ---------------------------------------------------------------------------------------------------------------------------------------*/
void Tela()
{
 clrscr();
 gotoxy(5,3); printf("Universidade Estadual de Campinas");
 gotoxy(5,4); printf("Faculdade de Engenharia Elétrica e de Computação - FEEC");
 gotoxy(5,5); printf("Cidade:");
 gotoxy(5,6); printf("Departamento:");
 gotoxy(5,7); printf("Tópicos:");
 gotoxy(5,8); printf("Orientador:");
 gotoxy(5,9); printf("Estudante:");
 gotoxy (5,23);
 printf("CopyRight(C) 1997 by Getúlio Júnior");
 gotoxy(13,5); printf("Campinas");
 gotoxy(19,6); printf("Departamento de Comunicações");
 gotoxy(14,7); printf("Protótipos de Redes Neurais - Usuários Interferentes");
 gotoxy(17,8); printf("Jaime Portugheis");
 gotoxy(16,9); printf("Getúlio A. de Deus Júnior");
}
/* ---------------------------------------------------------------------------------------------------------------------------------------*/
double Combinacao(int a, int b)
{
int i,x,y,z;
  x=1;
  y=1;
  z=1;
  for (i=2;i<=a;i++)
     x=x*i;
  for (i=2;i<=b;i++)
     y=y*i;
  for (i=2;i<=(a-b);i++)
     z=z*i;
  return (x/(y*z));
}
/* ---------------------------------------------------------------------------------------------------------------------------------------*/
/* O elemento Primitivo EP2 não será  usado */
void Elemento_Primitivos()
{
int Mal,i,j,flag,contador;
double a1,a2;
 Tela();
 contador=0;
 for (k=1;k<=(Q-1);k++) {
    for (j=1;j<=(Q-1);j++) {
                    Aux1[0][j-1]=fmod(pow(k,j),Q);
    }
    flag=1;
    for (j=0;j<=(Q-3);j++) {
     if (Aux1[0][j]==1) {
                    flag=0;
                    if(contador==0) {
                                    EP1=-1;
                    }
                    else {
                                    EP2=-1;
                    }
     }
```

```
    }
    if (((flag==1)&(Aux1[0][Q-2]==1))&(contador<=2)) {
     contador=contador+1;
     if (contador==1) {
                    EP1=k;
                    gotoxy (5,12); printf("Elemento Primitivo 1 =%d",EP1);
     }
     else {
                    if (contador==2) {
     EP2=k;
     k=Q;
     gotoxy (5,14); printf("Elemento Primitivo 2 =%d",EP2);
                    }
     }
    }
 }
 gotoxy (5,21);
 printf("Cálculo Elemento Primimitivo - Pressione Alguma Tecla [ ]");
 gotoxy(60,21);
 getch();
 }
 /* ---------------------------------------------------------------------------------------------------------------------------------------------*/
 void Matriz_de_Enderecos()
 {
 int j,i,Mal;
  CreateMatrix0(&Enderecos,Q,L,1,1);
  for (i=1;i<=L;i++) {
     Enderecos._[1][i]=0;
     for (j=1;j<=(Q-1);j++) {
      Enderecos._[j+1][i]=fmod(j*pow(EP1,i-1),Q);
     }
  }
  Tela();
  ShowMatrix(&Enderecos,5,12);
 Mal=SaveMatrix("c:\\getulio\\sincrono\\q31115m1\\endepbas\\eq31115",&Enderecos,MTX_DOUBLE);
  if (!Mal) { gotoxy (5,22); printf("Salvar Arquivo OK: eq31115.mtx - Pressione Alguma Tecla [ ]"); }
     else { gotoxy (5,22); printf("Erro: %d",Mal); }
  Salva_Matriz_de_Enderecos_TXT1();
  gotoxy (5,21);
  printf("Matriz de Endereços");
  gotoxy(62,22);
  getch();
 }
 /* ---------------------------------------------------------------------------------------------------------------------------------------------*/
 void Parametros()
 {
 FILE *Arquivo_Corrente;
 static char *Chave[NUMERO_DE_PARAMETROS]={"Campo de Galois=","L=","NUSistema","Mensagem1=",
 "Mensagem2=","Mensagem3=","Mensagem4=","Mensagem5=","Mensagem6=","Mensagem7=","Endereco1=","Endereco2=","Endereco3=","Endereco4=","Endereco5
 =","Endereco6=","Endereco7=" };
 char dflt[]="Impossível";
 char string[] = "Valores Iniciais";
 char fn[40];
 char msg[80];
 int lc;
 char *pret;
 int i,Vl,Vc,Valor_Contado;
 char *Ponteiro_Fim;
  Tela();
  gotoxy (5,21);
  printf("Parâmetros do Sistema - Pressione Alguma Tecla [ ]");
  Vc=5; Vl=12;
  Valor_Contado=1;
 if ((Arquivo_Corrente=fopen("c:\\getulio\\sincrono\\q31115m1\\endepbas\\endepbas.mlp", "r"))==NULL) {
    perror("Não posso abrir este arquivo! [ apiDOS ]");
    exit(1);
 }
 for (i=0;i<NUMERO_DE_PARAMETROS;i++) {
    msg[0]=0x00;
    for (lc=1;;lc++) {
     if (fgets(msg, 80, Arquivo_Corrente)==NULL) {
                    if (ferror(Arquivo_Corrente))
    {
    perror("Não posso abrir este arquivo! [ apiDOS ]");
    fclose(Arquivo_Corrente);
    exit(2); }
    else
     Resultado=dflt;
     break; /* NÆo Posso Encontrar a chave! */
      } /* if */
      pret=strstr(msg, Chave[i]);
      if ((pret) && (pret==msg)) { /* A Chave "mente" nesta linha! */
      Resultado=pret+strlen(Chave[i]);
      Resultado[strlen(Resultado)]=0x00;
      break ; }
      } /* for */
                    gotoxy(Vc,Vl);
                    printf("%s%s",Chave[i],Resultado);
                    if(Valor_Contado>=7) {
    Vc=Vc+20;
```

```
    Valor_Contado=1;
    Vl=11;
                    }
                    Valor_Contado++;
                    Vl++;
     switch (i) {
      case 0: Q = strtod(Resultado,&Ponteiro_Fim); break;
      case 1: L = strtod(Resultado,&Ponteiro_Fim); break;
      case 2: NUSistema = strtod(Resultado,&Ponteiro_Fim); break;
      case 3: Mensagem1 = strtod(Resultado,&Ponteiro_Fim); break;
      case 4: Mensagem2 = strtod(Resultado,&Ponteiro_Fim); break;
      case 5: Mensagem3 = strtod(Resultado,&Ponteiro_Fim); break;
      case 6: Mensagem4 = strtod(Resultado,&Ponteiro_Fim); break;
      case 7: Mensagem5 = strtod(Resultado,&Ponteiro_Fim); break;
      case 8: Mensagem6 = strtod(Resultado,&Ponteiro_Fim); break;
      case 9: Mensagem7 = strtod(Resultado,&Ponteiro_Fim); break;
      case 10: Endereco1 = strtod(Resultado,&Ponteiro_Fim); break;
      case 11: Endereco2= strtod(Resultado,&Ponteiro_Fim); break;
      case 12: Endereco3 = strtod(Resultado,&Ponteiro_Fim); break;
      case 13: Endereco4 = strtod(Resultado,&Ponteiro_Fim); break;
      case 14: Endereco5 = strtod(Resultado,&Ponteiro_Fim); break;
      case 15: Endereco6 = strtod(Resultado,&Ponteiro_Fim); break;
      case 16: Endereco7 = strtod(Resultado,&Ponteiro_Fim); break;
     } /* switch */
  } /* for */
  NumPad=Q;
  fclose(Arquivo_Corrente);
  gotoxy(53,21);
  getch();
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
double Numero_Aleatorio(long Valor_Inicial, long Valor_Final)
{
 int m;
   m=((double)rand()/(double)RAND_MAX)*(Valor_Final-Valor_Inicial)+Valor_Inicial;
   return (m);
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Padrao_tx(int ender, int usuario)
{
int s,d;
   CreateMatrix0(&MatrizAux1,Q,L,1,1);
   SetMatrix(&MatrizAux1,-1);
   for(s=1;s<=(L);s++) {
      for(d=1;d<=Q;d++) {
     MatrizAux1._[Q-fmod((usuario+Enderecos._[ender][s]),Q)][s]=1;
      }
    }
    for(s=1;s<=(L);s++) {
      for(d=1;d<=Q;d++) {
      if (Padrao._[1][s+(d-1)*(L)] < 0) {
        Padrao._[1][s+(d-1)*(L)]=MatrizAux1._[d][s];
            }
      }
    }
    DestroyMatrix(&MatrizAux1);
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Padrao_rx(int ender)
{
int pos,num,s,d,i,flagaux;
    CreateMatrix0(&MatrizAux2,Q,(L),1,1);
   CreateMatrix0(&MatrizAux1,Q,(L),1,1);
   SetMatrix(&MatrizAux1,-1);
    for(s=1;s<=(L);s++) {
     for(d=1;d<=Q;d++) {
    MatrizAux1._[d][s]=Padrao._[1][s+(d-1)*(L)];
    }
   }
   SetMatrix(&MatrizAux2,-1);
   for(s=1;s<=(L);s++) {
    for(d=1;d<=Q;d++) {
    if (MatrizAux1._[d][s]>0) {
                    num=((Q-d)-Enderecos._[ender][s]); //ender+1
      if (num<0) {
                    num=num+Q; }
      pos=(Q-num);
      if ((pos>0)&&(pos<=Q)) {
                    MatrizAux2._[pos][s]=1; }
    }
     } //for
    } //for
    for(s=1;s<=(L);s++) {
     for(d=1;d<=Q;d++) {
     Padrao._[1][s+(d-1)*(L)]=MatrizAux2._[d][s];
                    }
     }
    DestroyMatrix(&MatrizAux1);
    DestroyMatrix(&MatrizAux2);
}
```

```
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Cria_dados_arquivo_bin()
{
FILE *fptr1,*fptr2;
int d,s,contador,contadoraux,i,j,k,iaux,jaux,kaux,quativos,l,n_linhas,b,c,NNSai,o,p,f;
float padrao[466],saida[7],conta_padroes;
float saida_aux[50][7],contador_sai,valor, somador;
 NNSai=5; /* 6(numero tem que ser par: nao pode ser 5)>=log2(31) */
 CreateMatrix0(&Padrao,1,Q*L,1,1);
 if((fptr1=fopen("c:\\getulio\\sincrono\\q31l15m1\\endepbas\\xistrein.bin","wb"))==NULL) {
    printf("NÆo posso abrir o arquivo: xistrein.bin");
    exit(1);
 }
 if((fptr2=fopen("c:\\getulio\\sincrono\\q31l15m1\\endepbas\\ysaidas.bin","wb"))==NULL) {
    printf("NÆo posso abrir o arquivo: ysaidas.bin");
    exit(1);
 }
 conta_padroes=0;
// Padraes Básicos
/* Gerar os números binários */
   for (k=1;k<=NNSai;k++) {
    contador_sai=0;
    valor=-1.0;
    for (l=1;l<=pow(2,NNSai),l++) { /* 2^Q elementos no máximo */
                    contador_sai=contador_sai+1;
                    if (contador_sai>pow(2,NNSai-k)) {
                                valor=-1.0*valor;
                                contador_sai=1;
                    }
                    saida_aux[l][k]=valor;
    }
   }
/* Gerar o bit de paridade */
 for (l=1;l<=pow(2,NNSai);l++) { /* 2^Q elementos no máximo */
    somador=0;
    for (k=1;k<=NNSai;k++) {
                    if (saida_aux[l][k]>0) {
                     somador=somador+saida_aux[l][k];
                    }
    }
    valor=fmod(somador,2);
    if (valor==0) {
                    valor=-1;
    }
    saida_aux[l][NNSai+1]=valor;
 }
   /* transferencia dos dados para o arquivo */
   for(k=1;k<=Q;k++) {
                    for(jaux=1;jaux<=NNSai+1;jaux++) {
                                saida[jaux]=saida_aux[k+1][jaux];
                    }
                    fwrite(saida,sizeof(saida),1,fptr2);
   }
/* Padroes basicos */
    for(k=1;k<=Q;k++) {
                    SetMatrix(&Padrao,-1);
                    Padrao_tx(k,0);
                    for(iaux=1;iaux<=Q*L;iaux++) {
                                padrao[iaux]=Padrao._[1][iaux];
                    }
                    /* for(jaux=1;jaux<=NNSai+1;jaux++) {
                                saida[jaux]=-1;
                    }
                     saida[k+(l-1)*Q]=1; */
                    conta_padroes=conta_padroes+1;
                    fwrite(padrao,sizeof(padrao),1,fptr1);
   }
 fclose(fptr1);
 fclose(fptr2);
 Tela();
 gotoxy(5,12);
 printf("Número de Padrões %3.0lf",conta_padroes);
 gotoxy(5,21);
 printf("Arquivos Gravados !");
 gotoxy(5,22);
 printf("Tecle algo [ ]");
 gotoxy(17,22);
 DestroyMatrix(&Padrao);
 getch();
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Cria_mattrizes_de_enderecos_bin()
{
FILE *fptr1;
double aux1[31+1], aux2[31+1];
  clrscr();
  printf("Matrizes de enderecos *.bin foram criadas");
 if((fptr1=fopen("c:\\getulio\\sincrono\\q31l15m1\\endepbas\\eq31l15.bin","wb"))==NULL) {
    printf("NÆo posso abrir o arquivo: eq31l15.bin");
    exit(1);
```

```
  }
  /* transferencia dos dados para o arquivo */
  for (i=1;i<=L;i++) {
     aux1[1]=Enderecos._[1][i];
     for (j=1;j<=(Q-1);j++) {
                aux1[j+1]=Enderecos._[j+1][i];
     }
     fwrite(aux1,sizeof(aux1),1,fptr1);
  }
  fclose(fptr1);
  getch();
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
void Geracao_Padroes_Basicos_Entrada()
{
int Mal,i,NPad,bytes,iaux,jaux,j,l,NNSai;
  NNSai=49;
  Tela();
  NPad=pow(Q,2);
  CreateMatrix0(&Padrao,1,Q*L,1,1);
  CreateMatrix0(&Padroes,NPad,Q*L,1,1);
  CreateMatrix0(&Sai,NPad,NNSai+1,1,1);
  for(l=1;l<=Q;l++) {
     for(k=1;k<=Q;k++) {
                SetMatrix(&Padrao,-1);
                        Padrao_tx(k,0);
                for(iaux=1;iaux<=Q*L;iaux++) {
                        Padroes._[k+(l-1)*Q][iaux]=Padrao._[1][iaux];
                }
                for(jaux=1;jaux<=NNSai;jaux++) {
                        Sai._[k+(l-1)*Q][jaux]=-1;
                }
                Sai._[k+(l-1)*Q][k+(l-1)*Q]=1;
     }
  }
  Mal=SaveMatrix("c:\\getulio\\sincrono\\q31l15m1\\endepbas\\xistreina",&Padroes,MTX_DOUBLE);
  Mal=SaveMatrix("c:\\getulio\\sincrono\\q31l15m1\\endepbas\\ysaidas",&Sai,MTX_DOUBLE);
  if (!Mal) {
    gotoxy(5,22);
    printf("Salvar OK.");
  } else printf("\nErro: %d",Mal);
  DestroyMatrix(&Padroes);
  DestroyMatrix(&Padrao);
  DestroyMatrix(&Sai);
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
main()
{
  randomize();
  Parametros();
  Elemento_Primitivos();
  Matriz_de_Enderecos();
  Cria_dados_arquivo_bin();
  Visualiza_Dados();
  Cria_matrizes_de_enderecos_bin();
  DestroyMatrix(&Enderecos);
}
/* ---------------------------------------------------------------------------------------------------------------------------------------------*/
```

Arquivo: celular.c (ii)

Responsável pelo treinamento da RNA e cálculo das probabiliades de erro da RNA. Sistema Q31L15, com o código binário mais um bit de paridade. (É semelhante ao apresentado anteriormente).