ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

DEPARTAMENTO DE ENGENHARIA MECÂNICA

APLICAÇÃO DE REDES NEURAIS PARA RECONHECIMENTO DE SINAIS ACÚSTICOS

DISSERTAÇÃO APRESENTADA À ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO PARA OBTENÇÃO DO TÍTULO DE GRADUAÇÃO EM ENGENHARIA.

CHANG CHIH WEI

NO. USP: 185014

PROF. ORIENTADOR:
MARCELO GODOY SIMÕES

São Paulo

1998

FICHA CATALOGRÁFICA

WEI, Chang C., **Aplicação de redes neurais para reconhecimento de sinais acústicos**, pp, 1998.

PARA AQUELES QUE CONSIDERAM A EDUCAÇÃO O MAIS IMPORTANTE PARA UM PAÍS E SEU POVO.

À MINHA MÃE PELA SUA PREOCUPAÇÃO COM A MINHA FORMAÇÃO.

AGRADECIMENTOS

Ao amigo e orientador Prof. Marcelo Godoy Simões pelas diretrizes seguras e permanente incentivo neste trabalho.

À minha namorada Letéiya, cujo amor fez aumentar ainda mais a minha dedicação.

Ao Cássio, que permitiu a conclusão do curso com as suas retificações de matrícula.

A todos que direta ou indiretamente colaboraram na execução deste trabalho.

ÍNDICE

1. INTRODUÇÃO

Este projeto mecânico visa a aplicação de redes neurais artificiais no reconhecimento de sinais em um sistema de transmissão acústica. Este sistema é aplicado em poços de extração petrolífera onde os sinais a serem reconhecidos permitem o monitoramento constante da temperatura e pressão no fundo do poço.

Neste sistema, o sinal acústico é utilizado para a transmissão de informações do poço para a superfície através de uma tubulação, o que faz com que este sistema seja independente de fios ou outros tipos de ligações bastante sujeitos à deterioração rápida.

As informações transmitidas pelo sensor no fundo do poço ao receptor compõem-se de códigos binários transmitidos em duas freqüências, utilizando-se a modulação FSK (*Frequency Shift Keying*). Este tipo de transmissão está sujeita à problemas de ruído, atenuação, distorções não lineares e múltiplas reflexões nas conexões da tubulação, fazendo com que técnicas convencionais de demodulação não possam se aplicados de maneira satisfatória.

A proposta apresentada neste trabalho tem como objetivo resolver dois problemas principais: a captura da dinâmica do sinal (freqüência de chaveamento) e a habilidade de classificação desta dinâmica.

Para o primeiro problema, a proposta é um pré-processamento do sinal de entrada transformando a dinâmica (função temporal) em uma imagem tridimensional (superfície).

RESUMO

Neste projeto mecânico utilizou-se redes neurais artificiais no reconhecimento de sinais em um sistema de transmissão acústica aplicado em poços de extração petrolífera. Os sinais a serem reconhecidos permitem o monitoramento constante da temperatura e pressão no fundo do poço mas estão sujeitos à problemas de ruído, atenuação, distorções não lineares e múltiplas reflexões nas conexões da tubulação, fazendo com que técnicas convencionais de demodulação não possam se aplicados de maneira satisfatória.

A proposta apresentada resolveu dois problemas principais: a captura da dinâmica do sinal (freqüência de chaveamento) e a habilidade de classificação desta dinâmica.

Para a captura da dinâmica do sinal, utilizou-se um pré-processamento do sinal de entrada transformando a dinâmica (função temporal) em uma imagem tridimensional (superfície).

Para a classificação das imagens obtidas no pré-processamento, utilizou-se a rede neural counterpropagation apresentando resultados bastante satisfatórios.

Com a validação e verificação dos resultados, o projeto foi concluído tendo-se como produtos dois programas de redes neurais em C++ que também podem ser utilizados para outras aplicações.

Para o segundo problema, a proposta é a utilização de redes neurais para a classificação das imagens obtidas no pré-processamento.

Com a validação e verificação dos resultados, os princípios utilizados neste trabalho de formatura poderão ser aplicados de maneira satisfatória em outros sistemas de reconhecimento de padrões dinâmicos.

2. METODOLOGIA DE TRABALHO

A metodologia de trabalho utilizada para a realização deste projeto teve as seguintes etapas:

1. Especificação sucinta do problema
2. Apresentação da proposta para a solução do problema
3. Estudo de modelos de redes neurais
4. Equacionamento do modelo de rede neural e do algoritmo de treinamento.
5. Implementação da rede neural em software utilizando-se a linguagem C++.
6. Testes de validação e verificação da rede neural para a aplicação
7. Avaliação das dificuldades deste projeto
8. Propostas para desenvolvimento futuro
9. Conclusão

3. ESPECIFICAÇÃO DO PROBLEMA

O modelo simplificado do sistema da plataforma de extração de petróleo está representado na Figura 1. O petróleo flui por uma tubulação interna, que é composta por diversos tubos e conexões. A figura também mostra uma tubulação externa, que protege a interna do meio em que se encontra. O espaço entre a tubulação externa e a interna é preenchida por água e o comprimento deste sistema é normalmente maior que 3000m.

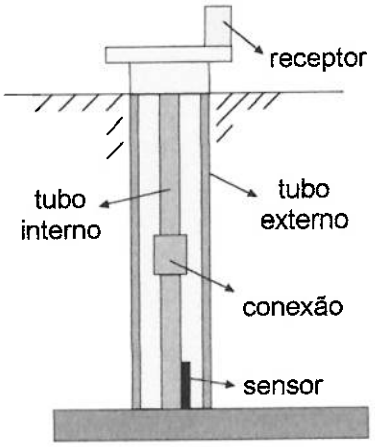


Figura 1 - Plataforma de extração petrolífera

Para uma boa operação em poços de petróleo, deve-se realizar um monitoramento periódico da temperatura e pressão do fundo do poço. Utiliza-se um sensor PDS (*Permanent Downhole Sensor*) para aquisição de dados sobre a temperatura e a pressão. Os responsáveis pela manutenção monitoram diariamente estas variáveis e injetam água entre a tubulação interna e externa para que os sistema se mantenha nas condições ótimas. Entretanto, os sensores são conectados à superfície através de cabos, o que faz com que o sistema fique sujeito a danos além de ter alto custo de instalação e manutenção.

A proposta que foi encontrada é a de se utilizar transmissão acústica para este sistema, já que transmissão rádio não é apropriada devido à atenuação obtida com a propagação da onda na água e na tubulação. Este tipo de transmissão é bastante usada em submarinos por exemplo.

Uma simples técnica de modulação utilizada para transmitir informações binárias é o chaveamento de freqüências ou FSK (*Frequency Shift Keying*), que utiliza duas freqüências diferentes f_1 e f_2. Estas duas freqüências correspondem aos bits "0" e "1" respectivamente.

Apesar de parecer bastante simples esta técnica, a detecção dos bits "0" e "1" no receptor é de extrema dificuldade. Isto ocorre devido à distorções e atenuações que o sinal recebe até chegar no receptor. As reflexões que ocorrem nas conexões da tubulação resultam em interferências não lineares no sinal. Além disso, o fluxo de óleo que passa pela tubulação interna também produz distúrbios que aumentam ainda mais a complexidade do sinal.

Para a solução deste problema, a proposta é a utilização de redes neurais que realizem o reconhecimento do sinal e tratem de forma satisfatória as características não lineares do sistema.

4. FUNDAMENTOS SOBRE REDES NEURAIS ARTIFICIAIS

4.1 Introdução

O desafio de se entender o cérebro motivou o pioneiro Ramón y Cayal (1911) a introduzir a idéia de neurônios como sendo constituintes estruturais do cérebro. Os neurônios são bem mais lentos que as portas lógicas de silicone. Os eventos em um chip ocorrem em nano-segundos, enquanto que em um neurônio ocorrem em mili-segundos. Entretando, a atuação do neurônio é compensada pelo número imenso de células nervosas e interconexões entre elas. É estimado que o córtex humano contenha aproximadamente 10 bilhões de neurônios e 60 trilhões de sinapses ou interconexões. Em termos de eficiência energética, o cérebro consome aproximadamente 10^{-16} joules por operação por segundo enquanto que o valor correspondente para os melhores computadores em uso atualmente é de aproximadamente 10^{-6} joules por operação por segundo.

O cérebro é um computador complexo, não-linear e com paralelismo no processamento de informações. Ele possui a capacidade de organizar neurônios assim como realizar processamentos como reconhecimento de padrões, percepção e controle motor muitas vezes mais rápido que o mais veloz computador digital.

Desde o nascimento, o cérebro tem uma ótima estrutura e a habilidade para construir suas próprias regras que podemos chamar de experiência. Esta experiência é construída ao longo dos anos e durante este estágio de desenvolvimento, aproximadamente um milhão de sinapses são formadas por segundo.

As sinapses são um unidades elementares estruturais e funcionais que mediam as interações entre os neurônios. Uma sinapse é uma simples conexão que pode impor excitação ou inibição em um certo neurônio receptor.

No desenvolvimento dos neurônios uma característica é extremamente importante: a plasticidade, que permite que o sistema nervoso em desenvolvimento se adapte ao ambiente externo. Em um cérebro humano esta plasticidade pode ser vista por dois mecanismos: a criação de novas conexões sinápticas entre os neurônios e a modificação de antigas sinapses.

Observando-se a plasticidade dos neurônios como unidades de processamento de informações em um cérebro humano, toma-se como analogia a criação de redes neurais com neurônios artificiais. De forma genérica, uma rede neural é uma máquina que é projetada para modelar a forma como o cérebro desempenha uma tarefa ou função. A rede é normalmente implementada utilizando-se componentes eletrônicos ou então simuladas em software em um computador. Para se alcançar um grande desempenho, as redes neurais empregam uma imensa interconexão de unidades de processamento chamadas de neurônios.

Uma rede neural é um imenso processador paralelo distribuído que tem a capacidade de armazenar experiências e torná-las viáveis para o uso. Ela assemelha-se ao cérebro por dois motivos:

1. A experiência é adquirida através de um processo de aprendizagem.

2. Os pesos das conexões entre neurônios são utilizados para armazenar as experiências.

O processo para se executar a aprendizagem chama-se de algoritmo de aprendizagem, que é uma função que irá modificar os pesos sinápticos em rede para ater-se ao objetivo do projeto.

A utilização de redes neurais oferece, como benefícios, as seguintes propriedades:

- ✓ Não-linearidade;
- ✓ Mapeamento de entradas e saídas num processo de aprendizagem;
- ✓ Adaptabilidade às mudanças no ambiente através de treinamentos;
- ✓ Resposta evidencial para padrões conflitantes (entradas iguais com saídas diferentes dados durante o treinamento);
- ✓ Informações contextuais, onde cada neurônio é afetado pela atividade global da rede;
- ✓ Robustez do sistema, ou seja, tolerância em relação à erros (como perda de conexões entre neurônios);
- ✓ Processamento paralelo simultâneo através dos neurônios;
- ✓ Uniformidade de análise e projeto que confere à rede um caráter universal.

As redes neurais artificiais são desenvolvidas em uma grande quantidade de configurações. Apesar dessa diversidade, elas têm vários aspectos em comum. A seguir serão descritos temas, notações e representações básicas que serão constantemente utilizadas neste trabalho.

4.2 O modelo biológico

Por serem inspirados em aspectos biológicos, os pesquisadores têm procurado aprender mais sobre a organização do cérebro em relação à configurações de redes e algoritmos. Os projetistas de redes neurais devem partir de conhecimentos biológicos, procurando por estruturas adequadas que gerem funções úteis. A Figura 2 mostra a estrutura de um par neurônios biológicos. Os dendritos se estendem do corpo celular para outros neurônios, onde eles recebem sinais em um ponto de conexão chamado de sinapse. No lado receptor da sinapse, as entradas são conduzidas para a célula corpo. As entradas são somadas, e algumas tendem a excitar a célula e outras a inibir o disparo. Quando a excitação acumulada em um corpo celular exceder um valor de *threshold*, a célula dispara, mandando um sinal através do axônio para outros neurônios. Este tipo básico de funcionamento tem várias complexidades e exceções, entretando, a maioria das redes neurais artificiais modelam apenas estas simples características.

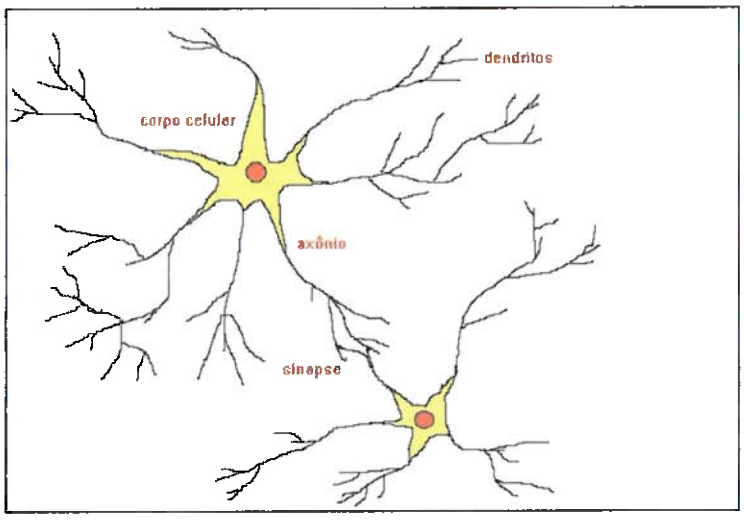


Figura 2 - O neurônio biológico

4.3 O neurônio artificial

O neurônio artificial foi projetado para desempenhar as características de primeira ordem dos neurônios biológicos. Em essência, um conjunto de entradas é aplicado, cada qual representando a saída de outro neurônio. Cada entrada é multiplicada por um peso correspondente (peso sináptico), e todas as entradas já ponderadas pelos pesos são somadas para determinar o nível de ativação do neurônio. A Figura 3 mostra um modelo representa esta idéia. Apesar da diversidade de configurações possíveis, quase todas se baseiam neste tipo de configuração.

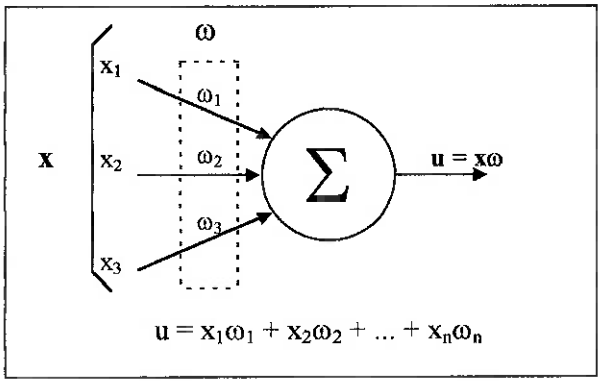


Figura 3 - O Neurônio Artificial

4.4 As funções de ativação

O sinal u (soma das entradas ponderadas) é geralmente processado por uma função de ativação φ para produzir um sinal de saída y do neurônio. Isto pode ser realizado através de uma simples função linear,

$$y = \varphi(u) = k \cdot u \qquad \text{onde k é uma constante}$$

ou através de uma função *threshold*,

$$y = \varphi(u) = \begin{cases} 1 & se\ u \geq \theta \\ 0 & se\ u < \theta \end{cases}$$ onde θ é uma constante de *threshold*.

Na Figura 4, a função de ativação φ aceita a saída u e produz um sinal y. Se o processamento em φ comprime a faixa de valores de u, então y nunca excede um valor limite.

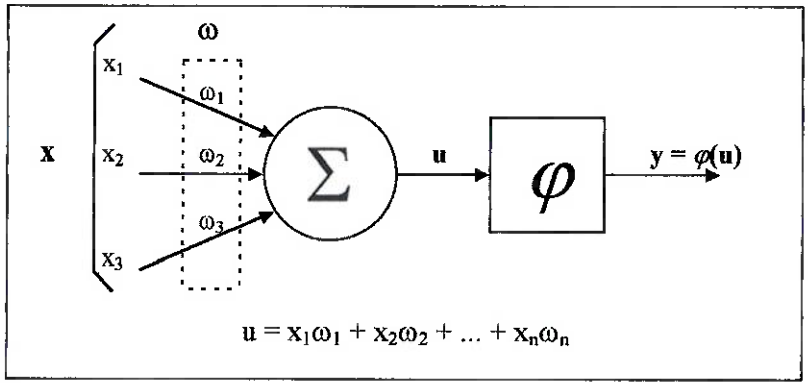


Figura 4 -Neurônio Artificial com a Função de Ativação

A função mais escolhida é a função logística ou sigmoidal (em forma de "S") como mostra a Figura 5.

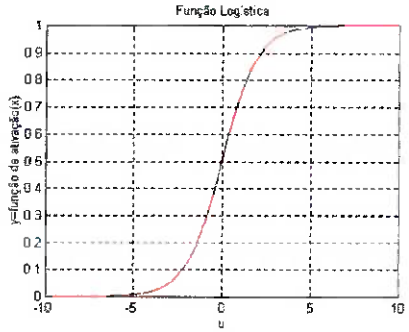


Figura 5 - Função de Ativação Sigmoidal (Logística)

A função logística é dada por:

$$y = \varphi(u) = \frac{1}{1 + e^{-u}}$$

Pela analogia a sistemas eletrônicos, pode-se pensar que a função de ativação é definida por um ganho não linear para o neurônio artificial. Este ganho é calculado pela taxa de mudança em y para uma pequena mudança em u. Então, o ganho é a rampa da curva para um nível de excitação especificado. O ganho varia de um valor baixo para grandes excitações negativas (a curva é quase horizontal), até um valor alto para excitação zero. O valor torna a ser baixo novamente para grandes excitações negativas. Esta característica não linear resolve o problema de processamento de sinais pequenos e grandes por uma mesma rede. A parte central de grande ganho da função logística resolve o problema de processamento de sinais pequenos, enquanto que as regiões laterais de ganhos pequenos (extremos positivos e negativos) resolvem o problema para grandes excitações.

Outra função de ativação amplamente utilizada é a função tangente hiperbólica (Figura 6), com o mesmo formato de curva da função logística (curva em 'S'). Entretanto, a função tangente-hiperbólica é simétrica na origem, resultando em valor zero quando a entrada é zero.

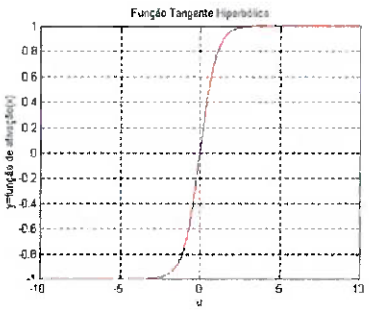


Figura 6 - Função de Ativação Hiperbólica

A função tangente hiperbólica é dada por:

$$y = \varphi(u) = tanh(x)$$

Este modelo simples para o neurônio artificial ignora muitas características biológicas como por exemplo, o atraso de tempo que afeta a dinâmica do sistema. Uma entrada gera imediatamente uma saída, e mais, o modelo não leva em conta os efeitos do sincronismo dos neurônios, o que vários pesquisadores acham de extrema importância.

4.5 Redes neurais de camada única

Apesar de um neurônio ter a capacidade de desempenhar certas funções simples, o poder da utilização computacional advém de se conectar os neurônios em redes. A rede mais simples é formado por um grupo de neurônios dispostos em uma única camada. (Figura 7).

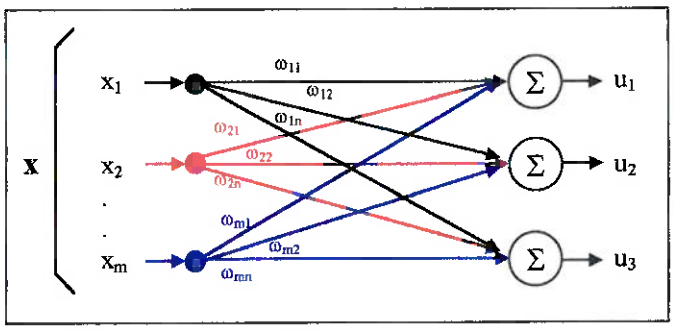


Figura 7 - Rede neural de camada única

Deve-se notar que existem nós que apenas distribuem as entradas, e portanto não devem ser considerados como constituintes de uma camada. Um conjunto de entradas **x** tem cada um dos seus elementos conectados em cada um dos neurônios passando por um peso determinado. Cada neurônio é simplesmente processado gerando uma saída que é somada às outras. Deve-se considerar os pesos como sendo elementos de uma matriz ω. A dimensão desta matriz é *mxn*, onde *m* é o número de entradas e *n* o número de neurônios. Por exemplo, o peso da conexão da quarta entrada com o segundo neurônio é

ω_{32}. Sendo assim, o cálculo das saídas **u** para a camada é uma simples multiplicação matricial:

$$\mathbf{u} = \mathbf{x}\ \omega$$

onde **u** e ω são vetores coluna.

4.6 Redes neurais de camadas múltiplas

Maiores e mais complexas, estas redes oferecem geralmente maiores capacidades computacionais. Apesar das redes serem construídas em quaisquer configurações, o arranjo de redes em estrutura de camadas assimila-se mais ainda a certas partes do cérebro. Estas redes de camadas múltiplas contém mais capacidades do que a de camada única e, recentemente, muitos algoritmos foram desenvolvidos para treinar estas redes.

As redes de camadas múltiplas são formadas pelo agrupamento de redes de camadas únicas. A saída de uma camada será a entrada da camada subseqüente. A Figura 8 mostra este tipo de configuração.

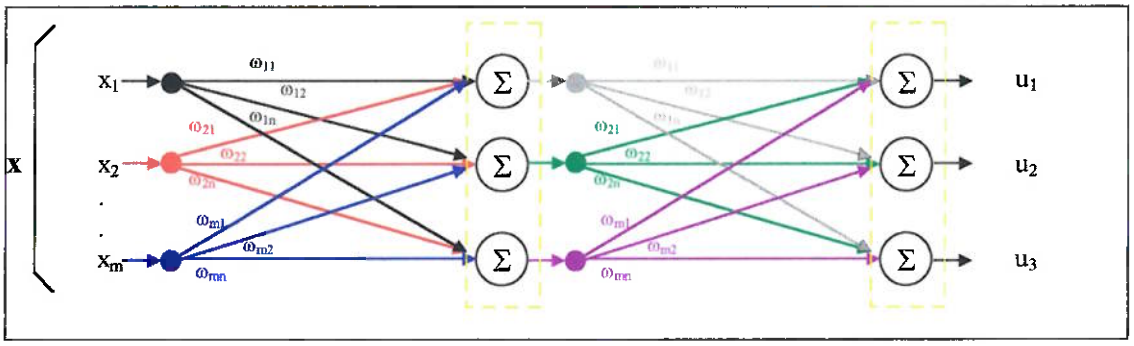


Figura 8 - Rede neural de camadas múltiplas

4.7 A função de ativação não linear entre as camadas

As redes neurais da camadas múltiplas não fornecem nenhuma vantagem computacional sobre as redes de camada única a não ser que haja uma função de ativação não linear entre as camadas. O cálculo da saída de uma camada consiste em se multiplicar o vetor de entrada por uma primeira matriz de pesos, e depois (se não houver uma função de ativação não linear) multiplicar o vetor resultante por uma segunda matriz de pesos. Isto pode ser expresso por:

$$(x\omega_1)\omega_2$$

Como a multiplicação de matrizes é associativa, pode-se reagrupar:

$$x(\omega_1\omega_2)$$

Isto mostra que uma rede neural de duas camadas lineares é exatamente equivalente à uma rede de camada única, considerando-se os pesos desta rede como sendo o produto $\omega_1\omega_2$. Portanto, a função de ativação não linear entre as camadas é de vital importância para a expansão das capacidades das redes de camadas múltiplas em relação as de camada única.

4.8 Redes neurais recorrentes

As redes consideradas até agora não tinham nenhuma conexão para realimentação, ou seja, não tinham conexões com pesos que ligavam a saída de uma camada à entrada da camada anterior. Este grupo de redes é chamado de não-recorrentes, ou de alimentação positiva. Para redes que contém conexões para realimentação são consideradas como recorrentes. Para redes não-recorrentes não existe memória, ou seja, sua saída é

determinada apenas a partir das entradas e dos pesos da rede. Para redes recorrentes, a saída é determinada pelas entradas. pelos pesos e pelas saídas anteriores. Por esta razão, a redes recorrentes podem exibir propriedades muito similares à memória a curto prazo de seres humanos.

4.9 Treinamento de redes neurais artificiais

Entre todas as características interessantes das redes neurais artificiais, nenhuma é mais importante do que a sua capacidade de aprendizagem. O treinamento das redes é uma analogia ao desenvolvimento intelectual humano. Apesar disso, deve-se atentar com o fato de que a aprendizagem das redes é limitada.

Uma rede é considerada treinada quando, com a aplicação de um conjunto de entradas, obtém-se um conjunto de saídas desejado. Cada conjunto de entradas é representado por um vetor. Então, o treinamento consiste em se aplicar seqüencialmente conjuntos de vetores, enquanto os pesos da rede são ajustados conforme um procedimento pré-determinado. Durante o treinamento, os pesos da rede convergem para valores que fazem com que a aplicação do vetor de entradas produza o vetor de saídas desejado.

4.10 Treinamento supervisionado e não supervisionado

Os algoritmos de treinamento são classificados como supervisionados e não-supervisionados. O treinamento supervisionado requer o conjunto formado pelo vetor de entradas e pela de saídas desejadas para as respectivas entradas. Normalmente, a rede é treinada com um certo número de pares de treinamento (nome dado ao conjunto entradas+saídas). Um vetor de entradas é aplicado, então a saída da rede é calculada e comparada com a saída desejada correspondente. A diferença (erro) entre elas é

realimentada para a correção dos pesos da rede utilizando-se um certo algoritmo de treinamento que minimizará o erro. Os vetores de entrada são aplicados seqüencialmente, os seus respectivos erros calculados, e os pesos da rede modificados até que o erro gerado pela rede para todo o conjunto de dados seja aceitável.

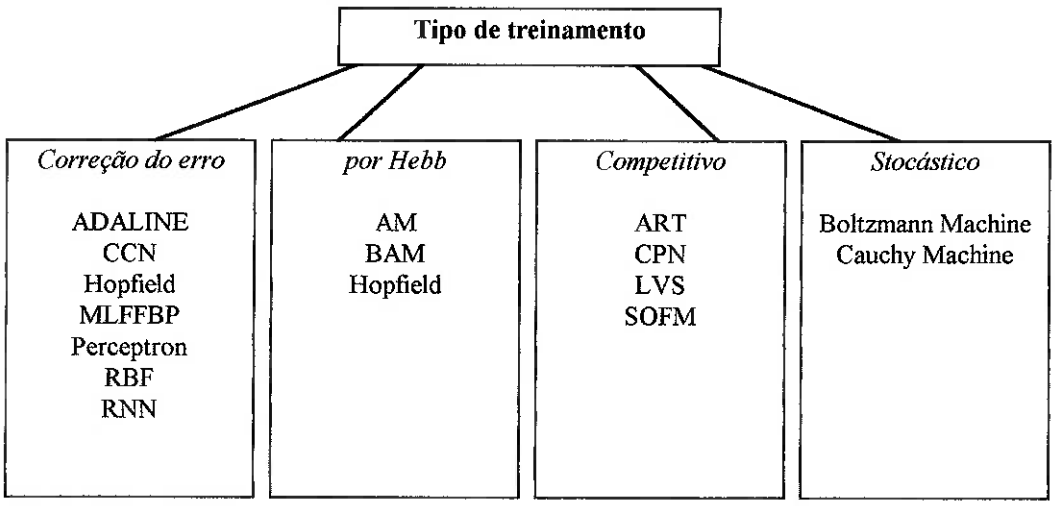
O treinamento não-supervisionado não necessita de um vetor de saídas desejadas. O conjunto de dados para treinamento consiste apenas nos vetores de entradas. O algoritmo de treinamento modifica os pesos da rede para gerar uma saída consistente. Essa saída consistente é a aplicação entradas similares que geram o mesmo padrão de saída. O processo de treinamento, portanto, envolve propriedades estatísticas do conjunto de dados para treinamento e agrupa vetores similares em classes. Para caracterizar essas classes deve-se então verificar as relações entre entrada e saída.

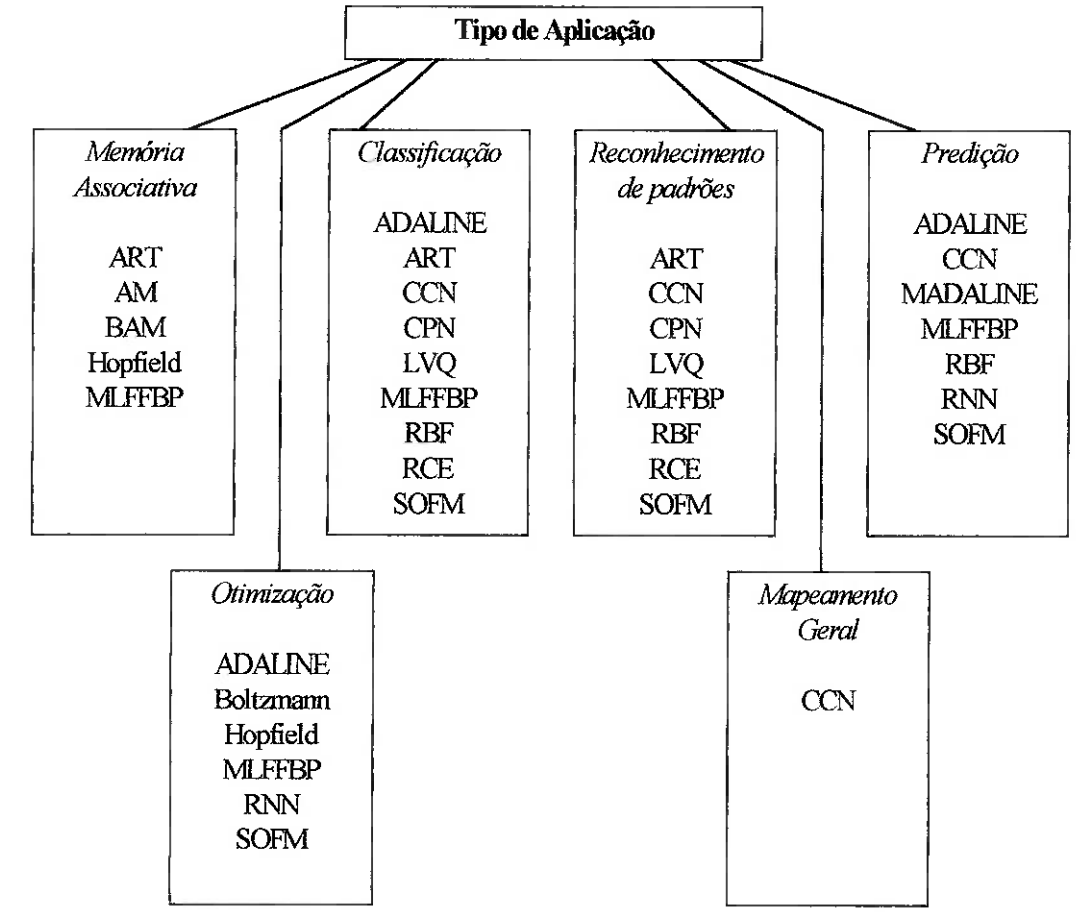
4.11 Classificação das Redes Neurais

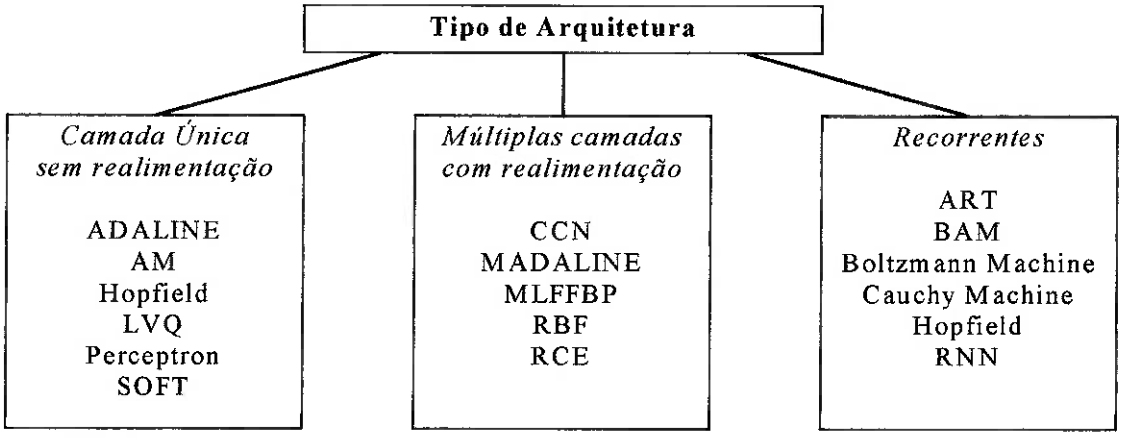
As redes neurais podem ser classificadas quanto à sua estratégia de treinamento, quanto ao tipo de aprendizagem, quanto ao tipo de aplicação e quanto à sua arquitetura. A seguir, serão classificadas as redes mais usuais segundo estes critérios. Algumas redes neurais existentes são:

- ADALINE - Adaptive-Linear Neural Element
- ART - Adaptative Ressonant Theory
- AM - Associative Memories
- BAM - Bidirectional Associative Memories
- Boltzmann Machine
- CCN - Cascade Correlation
- Cauchy Machine
- CPN - Counter Propagation
- Hamming
- Hopfield

- LVQ - Linear Vector Quantization
- MADALINE
- MLFFBP - Multilayer Feedfoward Back-Propagation
- Perceptron
- PNN - Probabilistic Neural Network
- RBF - Radial Basis Function
- RNN -Recurrent Neural Networks
- SOFM - Self-Organizing Feature Map




Tipo de Aplicação
Memória Associativa
ART
AM
BAM
Hopfield
MLFFBP
Classificação
ADALINE
ART
CCN
CPN
LVQ
MLFFBP
RBF
RCE
SOFM
Reconhecimento de padrões
ART
CCN
CPN
LVQ
MLFFBP
RBF
RCE
SOFM
Predição
ADALINE
CCN
MADALINE
MLFFBP
RBF
RNN
SOFM
Otimização
ADALINE
Boltzmann
Hopfield
MLFFBP
RNN
SOFM
Mapeamento Geral
CCN


Tipo de Arquitetura
Camada Única sem realimentação
ADALINE
AM
Hopfield
LVQ
Perceptron
SOFT
Múltiplas camadas com realimentação
CCN
MADALINE
MLFFBP
RBF
RCE
Recorrentes
ART
BAM
Boltzmann Machine
Cauchy Machine
Hopfield
RNN

5. REDE NEURAL BACKPROPAGATION

5.1 Introdução

Durante muitos anos não existia um algoritmo conciso e prático para o treinamento de rede neurais com múltiplas camadas. Além disso, as redes neurais de uma única camada mostravam-se limitados. Com isso, surgiu-se um novo tipo de algoritmo denominado backpropagation.

O backpropagation é um método sistemático para treinamento de redes de camadas múltiplas, e contém fundamentos matemáticos bastante práticos. Apesar de suas limitações, este algoritmo expandiu muito a aplicação de redes neurais, gerando ótimas demonstrações de seu potencial.

5.2 Configuração da rede

5.2.1 O neurônio

A Figura 9 mostra o neurônio utilizado como a unidade fundamental para redes com backpropagation. Um conjunto de entradas lhe é aplicado, tanto externamente como de uma camada anterior. Cada uma destas entradas é multiplicada por um peso e então somam-se os produtos obtidos. Esta soma de produtos é representada por u e deve ser calculado para cada neurônio da rede. Após o cálculo de u, uma função de ativação $\varphi(.)$ lhe é aplicada para sua modificação, produzindo um sinal y.

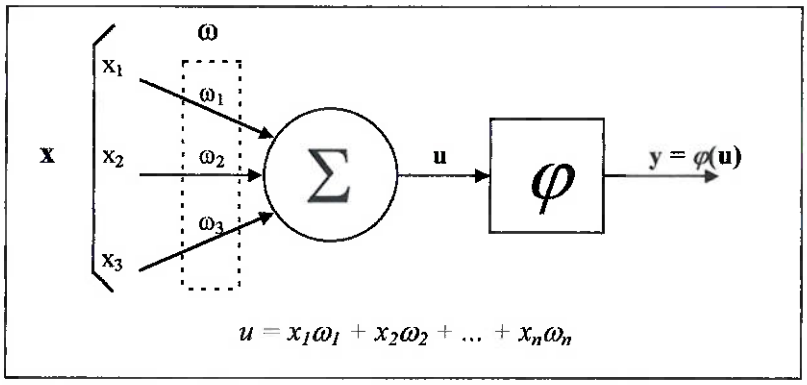


Figura 9 - Neurônio da Rede com Backpropagation

A função de ativação mais utilizada é a função logística ou sigmoidal, dada por:

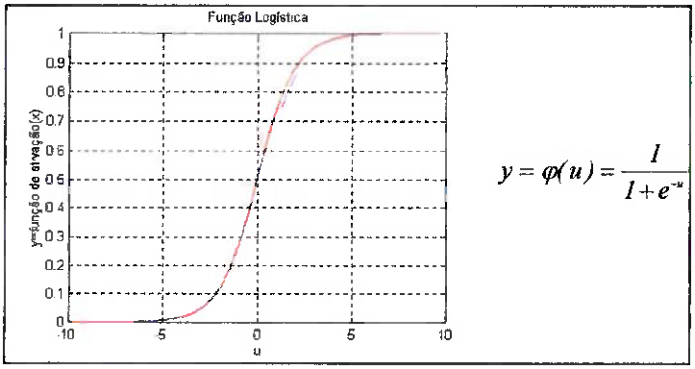


Figura 10 - Função de Ativação Sigmoidal

Observando-se a função sigmoidal acima, pode-se deduzir que a sua derivada é dada por:

$$\frac{\partial y}{\partial u} = y(1-y)$$

Pode-se utilizar também outras expressões matemáticas, mas desde que elas sejam diferenciáveis em todas as regiões. A função logística atende a esse requisito e, além do mais, tem a vantagem de ter um controle automático do ganho. Quando aplica-se sinais de pequena amplitude obtém-se amplitudes grandes e, ao se aumentar a amplitude do sinal, diminui-se o seu ganho. Deste jeito, sinais de grande amplitude podem ser

aplicadas à rede sem saturação, enquanto que sinais de pequena amplitude não irão receber atenuação excessiva.

5.2.2 A rede de camadas múltiplas

A Figura 11 mostra uma rede de camadas múltiplas própria para o treinamento com o backpropagation.

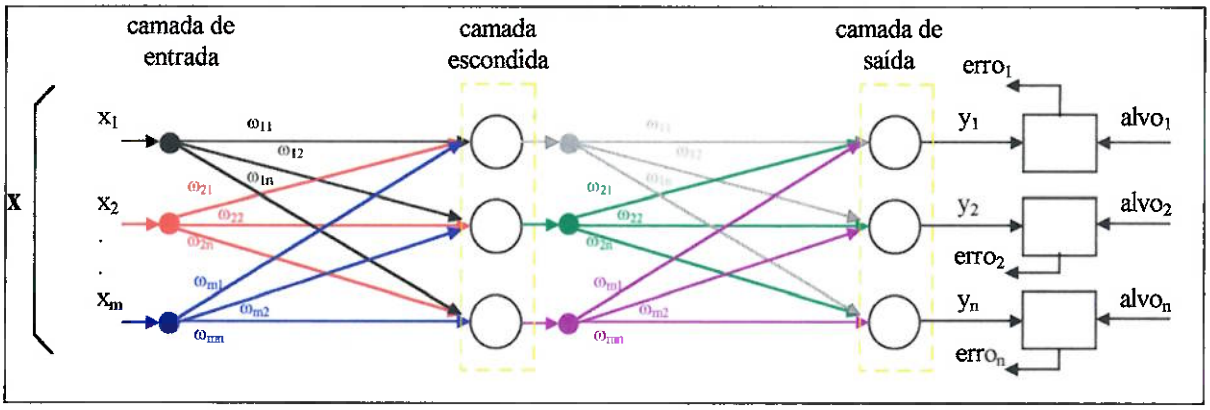


Figura 11 - Rede neural de três camadas para backpropagation

Neste caso, obtém-se uma rede neural com três camadas. O backpropagation pode ser aplicado à redes com qualquer número de camadas, entretanto, apenas duas camadas são necessárias para demonstrar este algoritmo.

5.3 O treinamento da rede neural backpropagation

O objetivo do treinamento da rede é ajustar os pesos para que com a aplicação de um conjunto de entradas seja gerado um conjunto de saídas desejado. Desta forma, pode-se utilizar a notação de vetores para o conjunto de entradas e saídas. O treinamento irá assumir que cada conjunto de entradas tenha uma par de saídas desejado, ou seja, os dados para o treinamento sempre encontram-se aos pares (entrada/saída). O treinamento se utiliza de um número de pares de dados, que pode variar em cada caso. Por exemplo,

para reconhecer a imagem de um caractere (supondo esta imagem dividida em vários quadrantes) utilizar-se-ia um conjunto com 26 pares de dados para treinamento.

Antes de se iniciar o processo de treinamento, todos os pesos devem ser inicializados com pesos pequenos e aleatórios. Isto irá garantir que a rede não se sature com grandes valores de entrada e previne contra problemas de desempenho quando a rede exige valores desiguais.

Na Figura 12, é apresentado o fluxograma do treinamento da rede backpropagation.

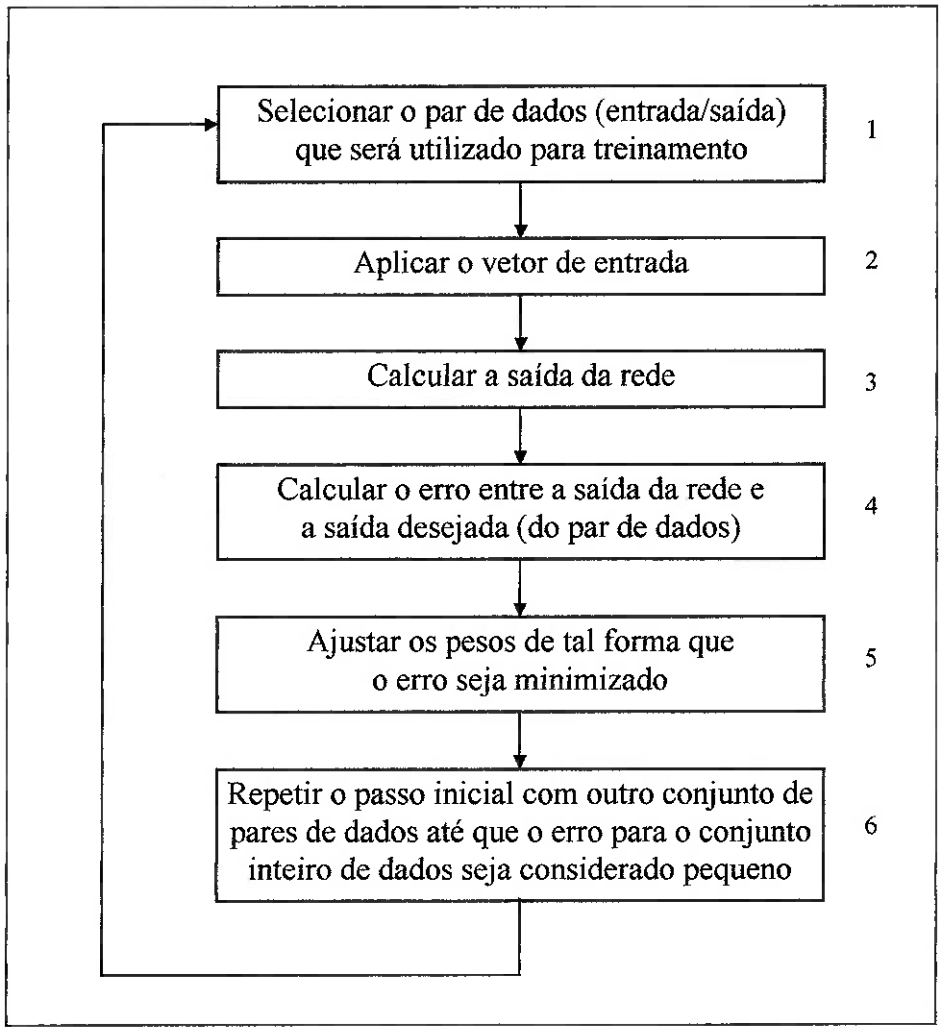


Figura 12 - Fluxograma do Treinamento Backpropagation

Os três primeiros passos são similares ao modo como a rede irá calcular as saídas após o treinamento. Os cálculos devem ser feitos camada por camada, ou seja, em um exemplo com três camadas, primeiro são calculados as saídas da camada de entrada, depois da camada escondida e finalmente da camada de saída.

No passo 4, cada saída y da rede será subtraída da saída desejada gerando um erro. Este erro é utilizado na etapa 5 para ajustar os pesos da rede. A polaridade e a magnitude da variação do peso serão determinados pelo algoritmo de treinamento.

Após várias iterações destas etapas, o erro entre as saídas da rede e as desejadas deve reduzir-se a um valor aceitável. Então ter-se á uma rede treinada.

Deve-se observar que as etapas 1, 2 e 3 constituem-se em "passos para frente", já que o sinal se propaga da entrada para a saída. As etapas 4 e 5 são "passos para trás", já que há a propagação do erro da saída para a entrada visando o ajuste de erros. Os sinais que constituem os passos para frente são ditos como *sinais da função*, enquanto que os que constituem os passos para trás são ditos *sinais dos erros* (Figura 13).

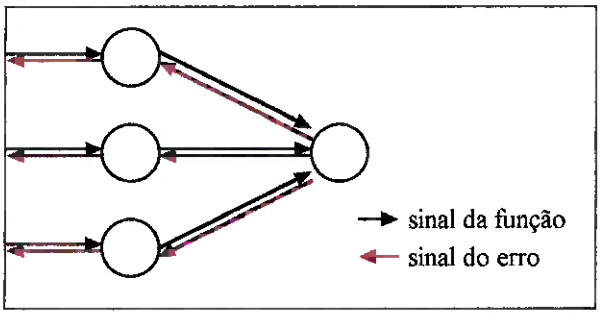


Figura 13 - Sinais da função e do erro

Com já foi visto anteriormente, o cálculo em redes neurais de camadas múltiplas é processado camada por camada, iniciando-se pelas camadas de entrada e terminando nas camadas de saída. O valor de u para cada neurônio da primeira camada é calculado

a partir da soma dos produtos dos sinais de entrada pelos respectivos pesos na conexão sináptica. A função de ativação $\varphi(.)$ faz com que o sinal proveniente u se limite certos valores, gerando o sinal y para cada neurônio da camada. Quando um conjunto de saídas para uma camada é encontrado, estas saídas se transformam em entradas para a camada seguinte. Este processo se repete, camada por camada, até que um conjunto final de saídas é gerado.

Em notação vetorial, obtém-se:

$$\mathbf{u} = \mathbf{x}.\omega$$

$$\mathbf{y} = \varphi(\mathbf{u})$$

Estas equações são então aplicadas para cada camada individualmente, desde a de entrada até a de saída.

5.3.1 Ajuste dos pesos da camada de saída

Com o valor de saída desejado disponível durante o treinamento da rede, a operação de ajuste de pesos torna-se extremamente simples, utilizando-se uma modificação da regra delta. O processo de treinamento inicia-se tomando o peso entre um neurônio i da camada escondida e um neurônio j da camada de saída. A saída do neurônio j é subtraída do valor da saída desejada gerando um sinal de erro e_j.

$$e_j = d_j - y_j$$

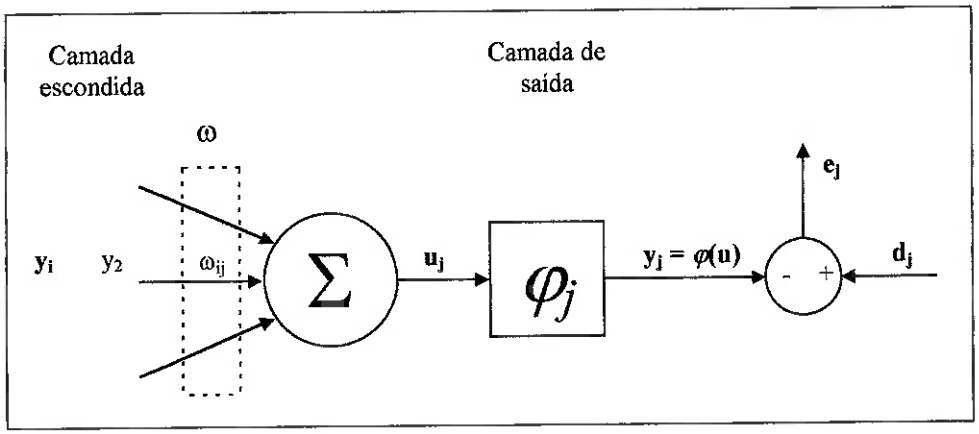


Figura 14 - Ajuste de pesos na camada de saída

Define-se como erro quadrático global a soma dos erros quadráticos de cada neurônio da camada de saída, ou seja, é expresso por:

$$\mathscr{E} = \frac{1}{2}\sum_{j} e_j^2$$

A atividade interna do neurônio *j* antes de ser ou não ativado é dado por:

$$u_j = \sum_i \omega_{ij} \cdot y_i$$

Aplicando-se a função de ativação obtém-se:

$$y_j = \varphi(u_j)$$

O algoritmo do backpropagation visa aplicar uma correção $\Delta\omega_{ij}$ para o sinal sináptico ω_{ij}:

$$\omega_{ij}(n+1) = \omega_{ij}(n) + \Delta\omega_{ij} \qquad \text{onde } n \text{ são as iterações}$$

A correção $\Delta\omega_{ij}$, é proporcional ao gradiente:

$$\frac{\partial \mathscr{E}}{\partial \omega_{ij}} = \frac{\partial \mathscr{E}}{\partial e_j} \cdot \frac{\partial e_j}{\partial y_j} \cdot \frac{\partial y_j}{\partial u_j} \cdot \frac{\partial u_j}{\partial \omega_{ij}}$$

Este gradiente representa um fator de sensibilidade, que determina a direção de busca para o peso ω_{ij}.

Derivando-se a equação do erro quadrático, obtém-se:

$$\frac{\partial \mathscr{E}}{\partial e_j} = e_j$$

Derivando-se a equação do erro no neurônio j, obtém-se:

$$\frac{\partial e_j}{\partial y_j} = -1$$

Derivando-se a equação da função de ativação:

$$\frac{\partial y_j}{\partial u_j} = \varphi_j'(u_j)$$

Derivando-se a equação da atividade interna do neurônio j antes de passar pela função de ativação:

$$\frac{\partial u_j}{\partial \omega_{ij}} = y_i$$

Substituindo-se na equação do gradiente:

$$\frac{\partial \mathscr{E}}{\partial \omega_{ij}} = -e_j \cdot \varphi_j'(u_j) \cdot y_i$$

A correção $\Delta\omega_{ij}$ aplicada ao peso ω_{ij} é dada pela regra delta:

$$\Delta\omega_{ij} = -\eta \frac{\partial \mathscr{E}}{\partial \omega_{ij}}$$

A constante η determina a taxa de aprendizagem e é chamada de *coeficiente de aprendizagem*.

Substituindo nas equações anteriores, obtém-se:

$$\Delta\omega_{ij} = -\eta\delta_j y_i$$

onde δ_j é o gradiente local, definido por:

$$\delta_j = e_j \cdot \varphi_j'(u_j)$$

Se a função de ativação for a logística, pode-se deduzir que:

$$\varphi_j'(u_j) = y_j(1 - y_j)$$

Portanto, obtém-se que o gradiente local é dado por:

$$\delta_j = e_j u_j(1 - u_j)$$

5.3.2 Ajuste dos pesos da camada escondida

As camadas escondidas não possuem informações sobre as saídas desejadas, então o processo de treinamento descrito anteriormente não pode ser utilizado. O backpropagation treina as camadas escondidas através da propagação do erro de saída para dentro das camadas. Esta propagação do erro é feito camada por camada até atingir a entrada com os respectivos ajustes de pesos.

Considera-se agora a seguinte situação para o neurônio *j* da camada escondida:

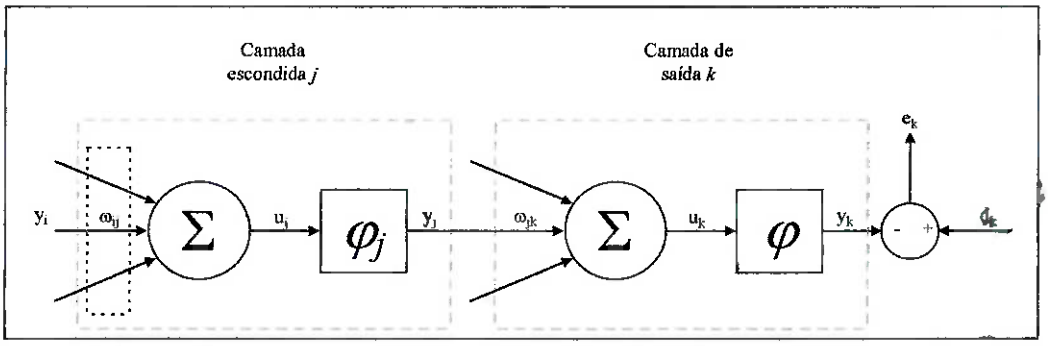


Figura 15 - Ajuste de pesos na camada escondida

O gradiente local para um neurônio escondido j é dado por:

$$\delta_j = -\frac{\partial \mathscr{E}}{\partial y_j} \cdot \frac{\partial y_j}{\partial u_j}$$

ou seja, $$\delta_j = -\frac{\partial \mathscr{E}}{\partial y_j} \cdot \varphi_j'(u_j)$$

Rescrevendo a equação do erro quadrático global para os novos índices:

$$\mathscr{E} = \frac{1}{2}\sum_k e_k^2$$

Diferenciando-se esta equação:

$$\frac{\partial \mathscr{E}}{\partial y_j} = \sum_k e_k \cdot \frac{\partial e_k}{\partial u_k} \cdot \frac{\partial u_k}{\partial y_j}$$

Entretanto, pela figura acima pode-se notar que:

$$e_k = d_k - y_k = d_k - \varphi_k(u_k)$$

Então, pode-se obter:

$$\frac{\partial e_k}{\partial u_k} = -\varphi_k'(u_k)$$

Tomando-se a atividade interna do neurônio k da camada de saída:

$$u_k = \sum_j \omega_{jk} \cdot y_j$$

Derivando-se esta equação:

$$\frac{\partial u_k}{\partial y_i} = \omega_{jk}$$

Então, pode-se obter a derivada parcial:

$$\frac{\partial \mathscr{E}}{\partial y_j} = -\sum_k e_k \cdot \varphi_k'(u_k) \cdot \omega_{jk} = -\sum_k \delta_k \omega_{jk}$$

Finalmente, pode-se obter o gradiente local pelo rearranjo dos termos:

$$\delta_j = \varphi_j'(u_j) \sum_k \delta_k \omega_{jk}$$

O fator $\varphi_j'(u_j)$ que aparece no gradiente local δ_j depende da ativação do neurônio escondido j. A somatória que compõe o segundo fator da multiplicação depende de dois conjuntos de dados: δ_k que requer conhecimento dos erros dos sinais para a camada imediatamente seguinte da camada escondida j e que é diretamente ligada à ela, e ω_{jk} que consiste nos pesos sinápticos associados a essas conexões.

Pode-se então resumir as relações obtidas para o backpropagation:

1. A correção $\Delta\omega_{ij}$ aplicada ao peso sináptico que conecta o neurônio i com o neurônio j é definida pela regra delta:

2. $\begin{pmatrix}\text{Correção do erro} \\ \Delta\omega_{ij}\end{pmatrix} = \begin{pmatrix}\text{Coeficiente de aprendizagem} \\ \eta\end{pmatrix} \cdot \begin{pmatrix}\text{Gradiente local} \\ \delta_j\end{pmatrix} \cdot \begin{pmatrix}\text{Sinal de entrada no neurônio } j \\ y_i\end{pmatrix}$

   O gradiente local δ_j depende em qual camada está se analisando:

   - se o neurônio j pertencer à camada de saída, δ_j é igual ao produto entre a derivada da função de ativação e o erro, ambos associados ao neurônio j.
   - se o neurônio j pertencer à camada escondida, δ_j é igual ao produto entre a derivada da função de ativação e a somatória dos produtos entre os pesos sinápticos e os δ's da camada seguinte que estão conectados ao neurônio j.

5.3.3 Coeficiente de momento

Rumelhart, Hilton e Williams (1986) descreveram um método para melhorar o tempo de treinamento do algoritmo do backpropagation sem perda de estabilidade do processo. Chamado de *coeficiente de momento*, o método envolve a adição de um termo no ajuste de pesos que se seja proporcional ao valor anterior de ajuste de pesos. Quando este termo é adicionado, pode-se dizer que há a presença de memória no ajuste dos pesos. Então a equação de ajuste de pesos fica modificada para:

$$\Delta\omega_{ij}(n+1) = -\eta\delta_j(n)y_i(n) + \alpha\Delta\omega_{ij}(n)$$

$$\omega_{ij}(n+1) = \omega_{ij}(n) + \Delta\omega_{ij}(n+1)$$

Utilizando-se o método do momento, a rede tende a seguir mas suavemente a curva de erro. Outros métodos de ajuste também são eficazes, como os baseados em atenuação exponencial.

6. REDE NEURAL COUNTERPROPAGATION

6.1 Introdução

A rede counterpropagation foi desenvolvida por Robert Hecht-Nielsen (1987) e vai além das redes de camada única no que se refere a representatividade. Comparada com a rede backpropagation, o tempo de treinamento é reduzido na ordem de centenas de vezes. A counterpropagation não é tão genérica quanto a backpropagation, mas permite a obtenção de soluções para aplicações que não toleram grandes tempos de treinamento.

A rede counterpropagation é uma combinação de duas configurações: a mapa auto-organizador de Kohonen (1988) e a camada outstar de Grossberg (1969, 1971, 1982). As duas configurações juntas agrupam propriedades peculiares que cada uma, individualmente, não possuiria. Esta combinação de paradigmas pode produzir redes neurais mais próximas ao modelo biológico do cérebro do que uma estrutura homogênea, já que o cérebro possui módulos especializados que operam em cascata para produzir uma determinada atividade.

O processo de treinamento da rede counterpropagation associa os vetores de entrada com o correspondente vetor de saída. Estes vetores podem ser binários, consistindo de zero e um, ou então contínuos. Uma vez que a rede neural é treinada, a aplicação de um vetor de entradas produz uma saída desejada. A capacidade de generalização da rede permite que a saída correta seja obtida mesmo que o vetor de entrada esteja parcialmente incompleto ou incorreto. Isto faz com que a rede seja bastante utilizada em reconhecimento de padrões e tratamento de sinais.

6.2 A estrutura da rede counterpropagation

A Figura 16 mostra uma versão simplificada da rede counterpropagation. Ela ilustra as características funcionais deste paradigma.

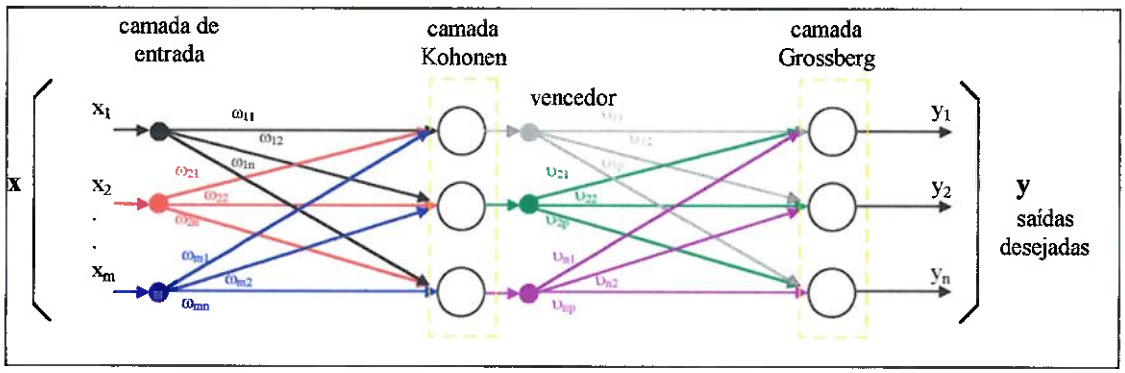


Figura 16 - Rede Counterpropagation

Cada neurônio da camada de entrada conecta todos os neurônios da camada Kohonen, fazendo a distribuição do sinal, sem apresentar aspectos computacionais. Esta conexão entre a camada de entrada e a Kohonen possui um peso denominado ω_{mn}. Analogamente, cada neurônio da camada Kohonen também está conectado à todos os neurônios da camada Grossberg através de um peso υ_{np} . Esta configuração assemelha-se à configuração de outras redes, entretanto, a diferença está no processamento realizado pelos neurônios da camada Kohonen e Grossberg.

6.3 Funcionamento da rede counterpropagation

6.3.1 Camada Kohonen

Na sua forma mais simples, a camada Kohonen funciona no modo "winner-takes-all". Isto significa que, para um dado vetor de entradas, um único neurônio da Kohonen dispara a saída lógica 1, enquanto que os outros neurônios não disparam (ou disparam 0).

Associado a cada neurônio Kohonen está um conjunto de pesos conectados a cada entrada. Por exemplo, na Figura 16, o neurônio Kohonen K_1 tem pesos ω_{11}, ω_{21}, ... , ω_{m1}. A camada de entrada é representada por x_1, x_2, .. , x_m. Assim como na maioria das redes neurais, a atividade de cada neurônio Kohonen é a soma das entradas multiplicada pelos respectivos pesos, ou seja:

$$u_j = \sum_i x_i \cdot \omega_{ij}$$

ou em notação vetorial:

$$\mathbf{u} = \mathbf{x}.\omega$$

onde: **u** é o vetor de atividade dos neurônios da camada Kohonen.

Para finalizar o processamento da camada Kohonen, o neurônio de maior atividade (maior valor de u_i) será considerado vencedor (ou "winner"). A saída deste neurônio será "1" e a saída dos demais neurônios será "0".

6.3.2 Camada Grossberg

A camada Grossberg funciona de forma similar a outras redes neurais. Sua saída é composta pela soma ponderada das saídas da camada Kohonen k_1, k_2, ... , k_n pelos seus respectivos pesos υ_{11}, υ_{12}, ... , υ_{np}. A saída para cada neurônio Grossberg é então dada por:

$$y_j = \sum_i k_i \cdot \upsilon_{ij}$$

ou em notação vetorial:

$$\mathbf{y} = \mathbf{k}.\upsilon$$

onde: **y** é o vetor de saída da camada Grossberg;

k é o vetor de saída da camada Kohonen;

υ é a matriz de pesos da camada Grossberg.

Se a rede neural utilizar apenas um neurônio vencedor para a camada Kohonen, apenas um elemento do vetor **k** é não nulo, resultando num processo de cálculo bastante simples de ser realizado. De fato, a única ação de cada neurônio Grossberg é gerar um vetor de saída que corresponde ao vetor de pesos que conecta a camada Grossberg ao neurônio vencedor.

6.4 Treinamento da Camada Kohonen

A camada Kohonen classifica os vetores de entrada em grupos semelhantes. Isto faz com que os ajustes dos pesos da camada Kohonen sejam tais que vetores de entrada semelhantes disparem sempre o mesmo neurônio Kohonen (vencedor). Posteriormente, a camada Grossberg se responsabilizará por produzir as saídas desejadas.

O treinamento Kohonen é um algoritmo que se auto-organiza, ou seja, é um treinamento não-supervisionado. Por esta razão, existe a dificuldade (além de não necessidade) de se prever qual neurônio Kohonen específico será ativado para uma dada entrada. É apenas necessário que se assegure que o treinamento separe vetores de entrada não similares.

6.4.1 Pré-processamento dos vetores de entrada

É altamente aconselhável, mas não obrigatório, que todos os vetores de entrada sejam normalizados antes de serem aplicados na rede neural. A normalização pode ser expressa por:

$$x_i' = \frac{x_i}{\sqrt{(x_1^2 + x_2^2 + \ldots + x_n^2)}}$$

A Figura 17 representa alguns vetores de entrada bi-dimensionais que estão em um círculo unitário. Nesta situação, tem-se apenas duas entradas para a rede neural. Esta idéia pode se estendida para um número arbitrário de entradas, gerando um vetor unitário de dimensões maiores contidos em uma hiper-esfera.

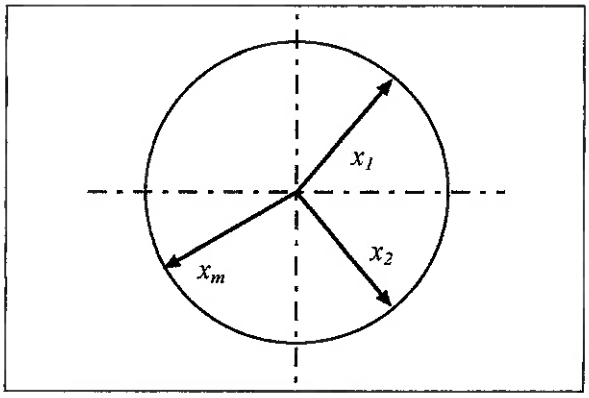


Figura 17- Vetores de entrada normalizados contidos em um círculo unitário

6.4.2 Competição dos neurônios e ajuste dos pesos

Para treinar a camada Kohonen, um vetor de entradas é aplicado e sua distância euclidiana é calculada em relação ao vetor de pesos associado a cada neurônio Kohonen. O neurônio que tiver a menor distância euclidiana é declarado vencedor (winner) e seus pesos são ajustados. A distância euclidiana é dada por:

$$d_i = \sqrt{(x_1 - \omega_{1i})^2 + (x_2 - \omega_{2i})^2 + \ldots + (x_m - \omega_{mi})^2}$$

Desta forma, estará se procurando o neurônio cujo vetor de pesos mais se assemelha ao vetor de entradas. O vetor de pesos deste neurônio vencedor será ajustado de modo a toná-lo ainda mais similar à entrada. É importante frisar que este treinamento é não-supervisionado, ou seja, a rede se auto-organiza para que a resposta de um dado neurônio Kohonen seja máxima para um dado vetor de entradas.

O ajuste dos pesos para um neurônio vencedor *j* é feito seguindo a seguinte equação:

$$\omega_{ij}^{novo} = \omega_{ij}^{antigo} + \alpha(x_i - \omega_{ij}^{antigo})$$

onde: α é o coeficiente de aprendizagem Kohonen

x_i é a componente *i* do vetor de entradas.

Cada peso ω_{ij} associado ao neurônio vencedor *j* é ajustado com uma parcela proporcional à diferença entre seu valor e o respectivo valor da entrada x_i. .O objetivo deste ajuste é minimizar a distância entre os pesos e as entradas.

A Figura 18 mostra o processo de treinamento de um caso de apenas duas entradas (bi-dimensional). Inicialmente, é encontrado a diferença ($\mathbf{x} - \omega^{\mathbf{velho}}$) para construir um vetor que partirá da ponta do vetor ω até o vetor **x**. Em seguida, este vetor tem seu comprimento ajustado com a multiplicação de um escalar α (cujo valor é menor do que 1), produzindo um vetor de ajuste δ. Finalmente, o novo vetor $\omega^{\mathbf{novo}}$ é o segmento orientado que sai da origem e vai até a ponta de δ. Desta forma, o efeito do treinamento é rotacionar o vetor de pesos em direção ao vetor de entradas.

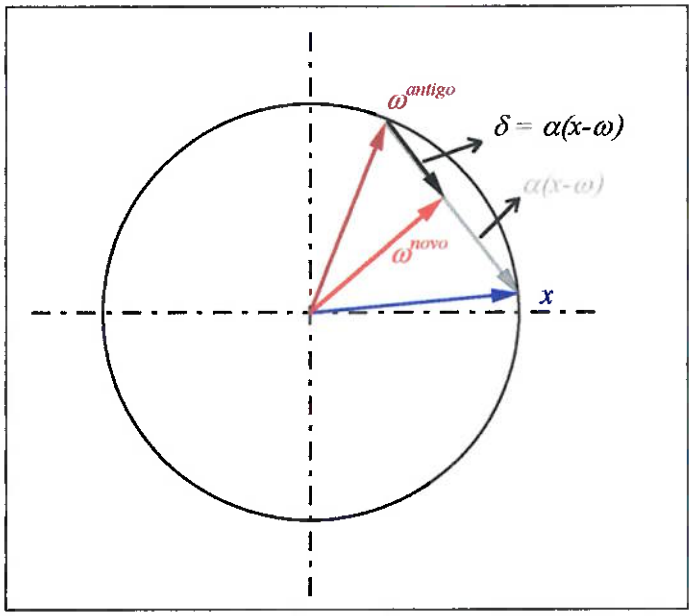


Figura 18 - Ajuste de pesos no treinamento Kohonen

A variável α é o coeficiente de aprendizagem que usualmente inicia-se com o valor de aproximadamente 0,7 e pode se reduzido gradualmente durante o treinamento para uma melhor convergência para o valor final.

Normalmente, os vetores de entrada contidos nos dados para treinamento são agrupados de acordo com suas similaridades. Estes grupos irão ativar o mesmo neurônio Kohonen. Neste caso, o vetor de pesos para este neurônio será a média dos vetores de entrada que lhe serão aplicados para dispará-lo.

Ao utilizar-se um valor de α baixo, os efeitos decorrentes das etapas do treinamento serão diminuídos, ou seja, os pesos associados a cada neurônio irão assumir um valor próximo ao "central" dos valores dos vetores de entrada para os quais o neurônio foi considerado vencedor.

6.4.3 Inicialização dos pesos

Todos os pesos da rede neural devem ser inicializados com determinados valores antes de se iniciar o treinamento. É comum a escolha de valores aleatórios pequenos para os pesos, entretanto, esta escolha para os pesos da camada Kohonen pode causar sérios problemas no treinamento. Os vetores de entrada não estão normalmente dispostos de modo a serem facilmente agrupados em partes diferentes da hiper-esfera. Isto faz com que muitos dos vetores de pesos estejam tenham uma grande distância euclidiana tal que não sejam considerados os mais similares para a ativação do neurônio vencedor.

A conseqüência deste problema é a perda dos neurônios que sempre irão ter a saída zero. Além disso, os vetores restantes que podem ser ativados serão tão poucos em quantidade que não possibilitarão a distinção dos vetores de entrada que estarão dispostos em uma mesma porção da hiper-esfera. Este fenômeno pode ser visto na Figura 19.

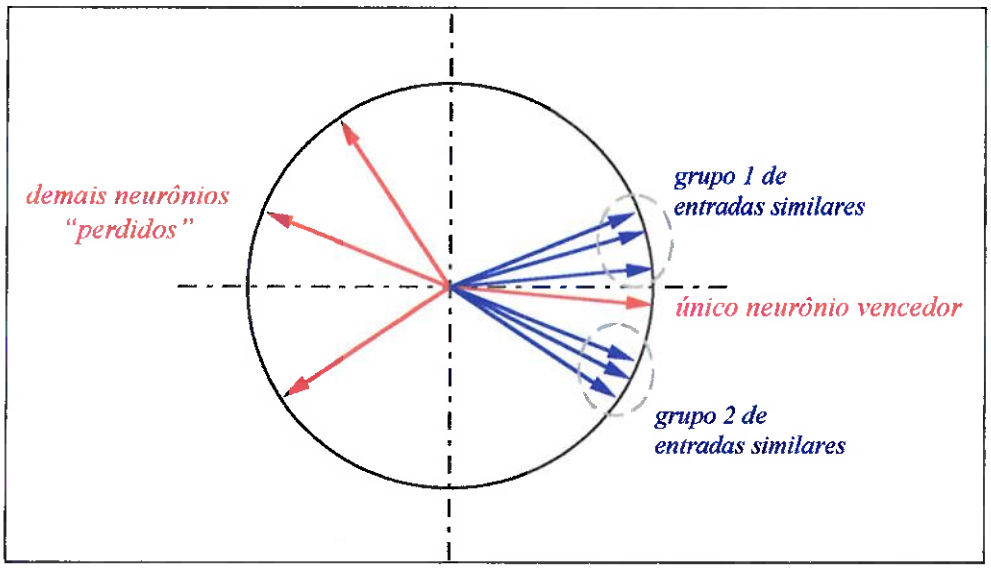


Figura 19 - Problema de inicialização aleatória dos pesos da camada Kohonen

A solução mais apropriada seria o de analisar a densidade de distribuição dos vetores de entrada, observando-se então em que regiões da hiper-esfera ocorrem as maiores incidências dos vetores pesos. A partir daí, os pesos seria distribuídos aleatoriamente nesta região. Entretanto, esta implementação é impraticável diretamente, mas existem diversas técnicas que aproximam os seus efeitos.

A primeira solução, chamada de *método de combinação convexa*, assume que todos os pesos tenham os pesos iniciais iguais a $(1/\sqrt{n})$, onde n é o número de entradas e, portanto, o número de componentes do vetor de pesos. Isto faz com que os vetores de pesos sejam unitários e todos coincidentes. Além disso, a entrada é ajustada da seguinte forma:

$$x_i' = (1-\alpha)\cdot x_i + \frac{1}{\sqrt{n}}\cdot\alpha$$

Inicializa-se α, com valores grandes (o que gera proximidade com $(1/\sqrt{n})$) para depois se diminuir seu valor (proximidade com x_i). Os vetores de pesos irão seguir um ou um pequeno grupo de vetores de entradas e, no fim do treinamento, poderão produzir os padrões de saída desejados. Este procedimento aumenta de forma bastante significativa do tempo de treinamento, já que os vetores de pesos têm que seguir um alvo móvel.

Uma outra solução seria o de atribuir a cada neurônio Kohonen uma "consciência". Este método foi proposto por DiSieno (1988) e consiste em se penalizar os neurônios que vencem a competição muito freqüentemente. Considerando-se que p_i seja a freqüência em que o neurônio i vença a competição, pode-se definir:

$$p_i^{novo} = p_i^{antigo} + b(y_i - p_i^{antigo}) \qquad \text{onde } b \text{ é uma constante entre 0 e 1.}$$

Se z_i representar o neurônio vencedor, então um valor de *bias* B_i deve ser adicionado para modificar a competição de tal forma que:

$$z_i = \begin{cases} 1 & \text{se } (|\omega_i - x|^2 - B_i) \leq (|\omega_j - x|^2 - B_j) \quad \text{com i} \neq \text{j} \\ 0 & \text{caso contrário} \end{cases}$$

onde o valor da penalidade B_i é dado por:

$$B_i = C \cdot \left(\frac{1}{n} - p_i \right)$$

onde C é um fator de *bias* e n é o número de neurônios da camada Kohonen.

Finalmente, pode-se ajustar os pesos do neurônio vencedor de acordo com:

$$\omega_i^{novo} = \omega_i^{antigo} + \alpha \cdot (x - \omega_i^{antigo}) \cdot z_i$$

O método de "consciência" descrito é muito eficaz no desenvolvimento de características equiprováveis das entradas. É comprovado que a sua aplicação melhora o desempenho do treinamento de diferentes redes neurais que utilizam o treinamento competitivo.

6.4.4 Modo interpolativo (mais de um único vencedor)

Até agora, foi descrito o algoritmo de treinamento competitivo onde apenas um neurônio Kohonen é considerado vencedor. Este algoritmo tem uma precisão limitada, já que a sua saída é determinada apenas pelo neurônio vencedor.

Um modo de melhorar a precisão é utilizando-se um método interpolativo, onde os neurônios de maiores atividades são agrupados para serem apresentados à camada Grossberg. Estes grupos de neurônios vencedores não têm tamanho definido, variando de aplicação para aplicação. Uma vez que o grupo é determinado, as saídas dos

neurônios vencedores da camada competitiva são normalizados para valores unitários. Os demais neurônios que não pertencem a este grupo têm em suas saídas o valor nulo.

O modo interpolativo é mais capaz de representar mapeamentos mais complexos e produz resultados mais precisos. Entretanto, não existe nenhuma evidência para se avaliar os dois modos citados anteriormente.

6.5 Treinamento da camada Grossberg

A camada Grossberg é relativamente simples de ser treinada. Um vetor de entradas é aplicado na rede, a saída da camada Kohonen é obtida, e a saída da camada Grossberg é então calculada numa operação normal. Posteriormente, cada peso que está conectado ao neurônio vencedor é ajustado. O ajuste é proporcional à diferença entre o peso e a saída desejada, ou seja:

$$\upsilon_{ij}^{novo} = \upsilon_{ij}^{antigo} + \beta(y_j - \upsilon_{ij}^{antigo})k_i$$

onde: k_i é a saída do neurônio Kohonen i;

y_i é a componente j do vetor de saídas desejadas;

β é o coeficiente de aprendizagem da camada Grossberg.

Inicialmente, o valor de β é de aproximadamente 0,1 e é gradualmente reduzido ao longo do processo de treinamento.

A partir do que foi apresentado, nota-se que os pesos da camada Grossberg irão convergir para os valores médios das saídas desejadas, apesar dos pesos da camada Kohonen serem treinados para os valores médios das entradas. O treinamento Grossberg é supervisionado, sendo que o algoritmo se utiliza da saída desejada. O treinamento

não-supervisionado da camada Kohonen produz saídas em posições indeterminadas; estas saídas serão mapeadas para produzir as saídas desejadas na camada Grossberg.

6.6 Resumo do treinamento da rede counterpropagation

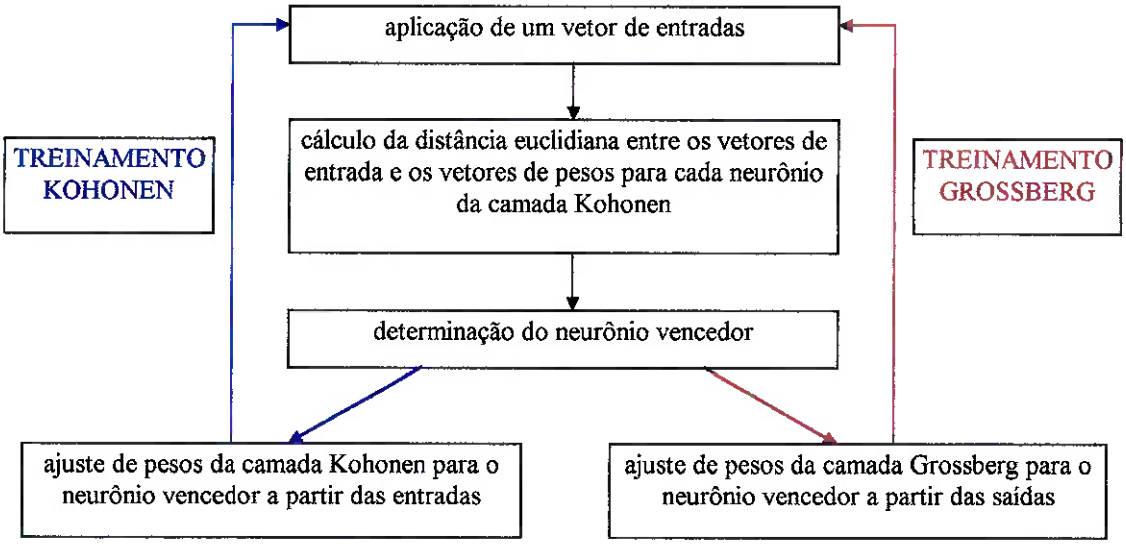


Figura 20 - Treinamento da rede counterpropagation

7. IMPLEMENTAÇÃO DA REDE BACKPROPAGATION

7.1 Introdução

A implementação da rede neural foi feita utilizando-se a programação orientada à objetos na linguagem C++. As principais rotinas são:

•*Rotina principal*: controla o fluxo de processamento das outras rotinas recebendo todos os parâmetros da simulação;

•*Rotina de construção da rede*: cria ou abre uma rede neural com a inicialização dos neurônios, dos pesos e das entradas;

•*Rotina de treinamento*: a rede recebe os padrões de treinamento de forma seqüencial e se adequa para a obtenção das saídas desejadas;

•*Rotina de teste:* a rede recebe os padrões de teste e obtém o erro das respostas cabendo ao usuário decidir ou não utilizar a rede ou realizar outro treinamento;

•*Rotina de execução:* a rede recebe os padrões de entrada e obtém a resposta que será utilizada em reconhecimento de padrões;

•*Biblioteca de rotinas*: demais funções relevantes como: procedimentos específicos para a treinamento, entrada, saída e outros.

O *black-box* da rede neural implementada é dado a seguir na Figura 21

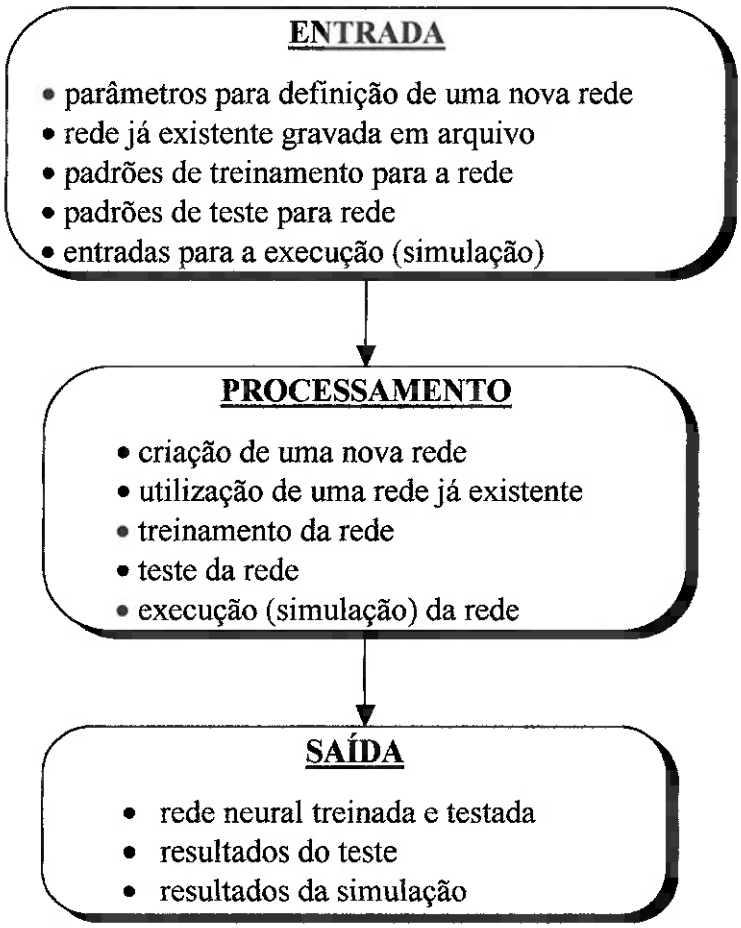


Figura 21 - Black-box da rede neural

7.2 Descrição do programa implementado

A implementação da rede neural é feita através das matrizes *w1* e *w2* que representam os pesos, e dos vetores *s1*, *s2* e *s3* que representam os sinais sinápticos.

A inicialização de um rede neural foi feita criando-se as matrizes *w1* e *w2* (através de alocação dinâmica) dados o número de entradas, de saídas e de neurônios. Os pesos iniciais foram atribuídos aleatoriamente dentro do programa.

A abertura de uma rede consistiu em criar-se as matrizes *w1* e *w2* a partir de um arquivo cuja extensão é **.sz** (onde são fornecidos o número de entradas, de saídas e de

neurônios). Para a leitura dos pesos, utilizaram-se os arquivos de extensão ***.w1*** e ***.w2*** definido-se então a rede.

Para o cálculo das saídas, deve-se igualar *s3* à entrada. Percorre-se a rede através dos sinais sinápticos, aplicando-se os pesos de tal forma que pode-se obter então a saída dado pelo vetor *s1*. Para o treinamento, utilizou-se o cálculo das saídas e comparou-se com as saídas desejadas (encontradas no arquivo de treinamento cuja extensão é ***.trn***). Com isso, pôde-se corrigir os pesos iterativamente até que o erro quadrático fosse menor do que um valor admissível, ou então até que o número máximo de iterações fosse atingido. Durante o treinamento, houve a alteração dos coeficientes de aprendizado e de momento segundo parâmetros solicitados pelo programa. A alteração é feita de acordo com o número de iterações para escalonamento. Se esse número for atingido, corrige-se os coeficientes e zera-se novamente a contagem. Existe uma barra de erro (Figura 22) que indica o comportamento do erro durante o programa. O erro é representado pela parte central da barra, aumentado e diminuindo conforme o desenvolvimento do treinamento. Inicialmente, o menor unidade de erro corresponde a 1% mas quando o erro é menor do que este valor, a escala da barra de erro se altera. Isto pode ser observado a partir da mudança de cores nas partes laterais da barra. Além disso, o fator multiplicativo na escala aparece logo abaixo da barra de erro, como pode ser observado na Figura 22.

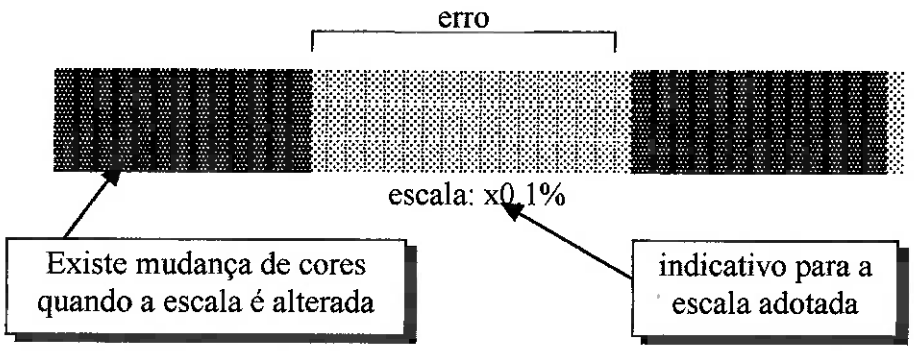


Figura 22 - Barra de erro do programa backpropagation durante treinamento

Para o teste, o procedimento inicial é análogo ao do treinamento, porém, não há a correção dos pesos. Apresenta-se apenas o erro quadrático resultante de aplicações utilizando-se 30% dos padrões de teste contidos em um arquivo (definido pelo usuário).

A execução da rede neural (cálculo de ume saída dada a entrada) é feita simplesmente chamando o mesmo procedimento utilizado no treinamento e no teste. O cálculo do erro e a correção dos pesos não são efetuados e a entrada é dada através do teclado de um arquivo.

A arquitetura do software é apresentada a seguir mostrando a hierarquia e a relação entre as partes funcionais do sistema.

7.3 Arquitetura do Programa

A hierarquia entre as rotinas e os objetos estão descritas logo a seguir:

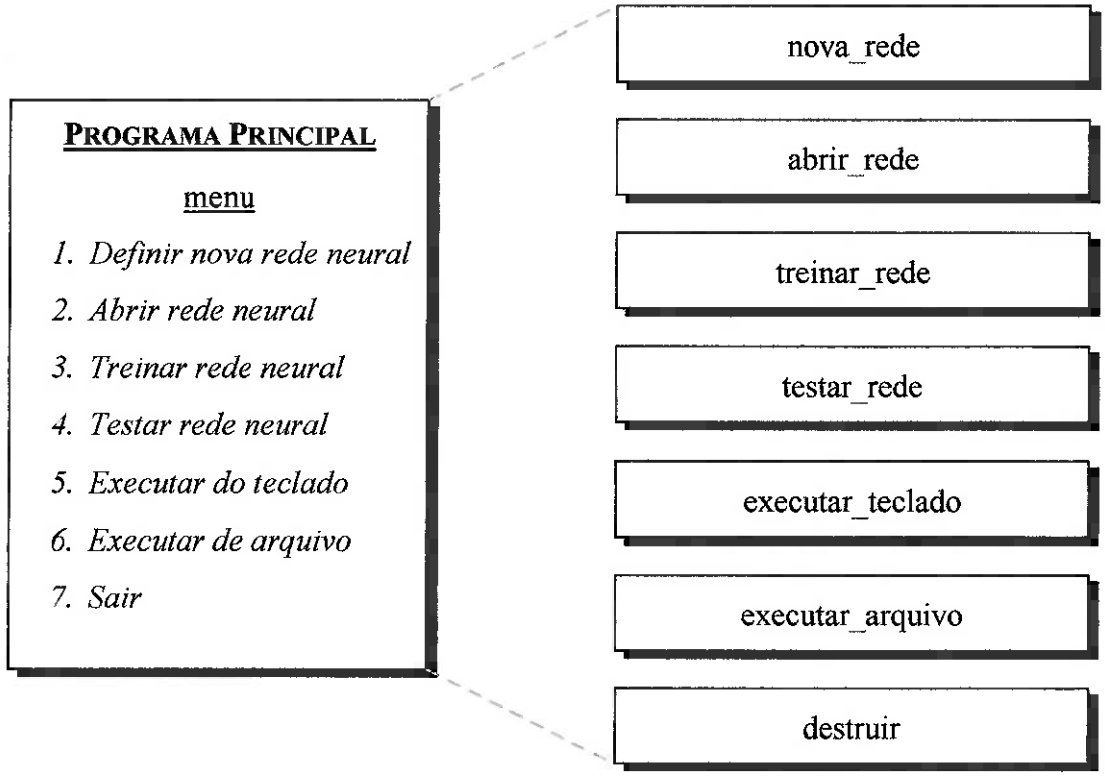


Figura 23 - Arquitetura principal do programa backpropagation

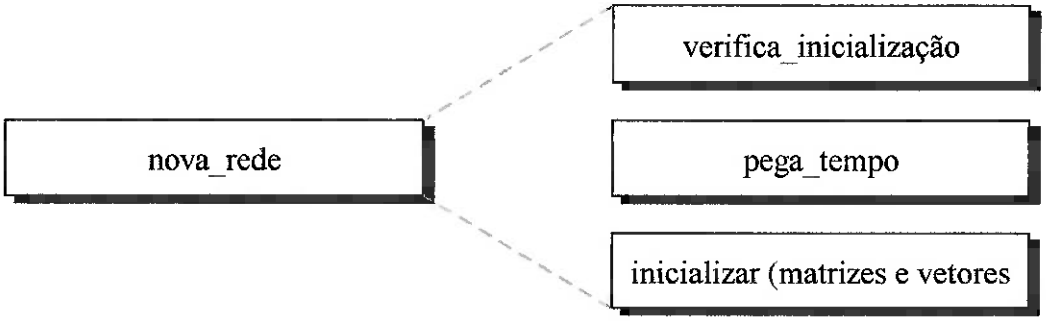


Figura 24 - Programa Backpropagation: Rotina nova_rede

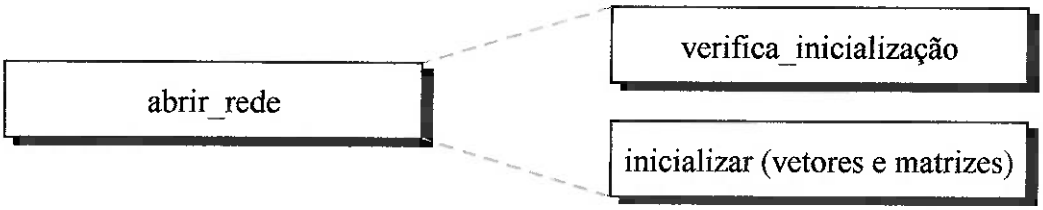


Figura 25 - Programa Backpropagation: Rotina abrir_rede

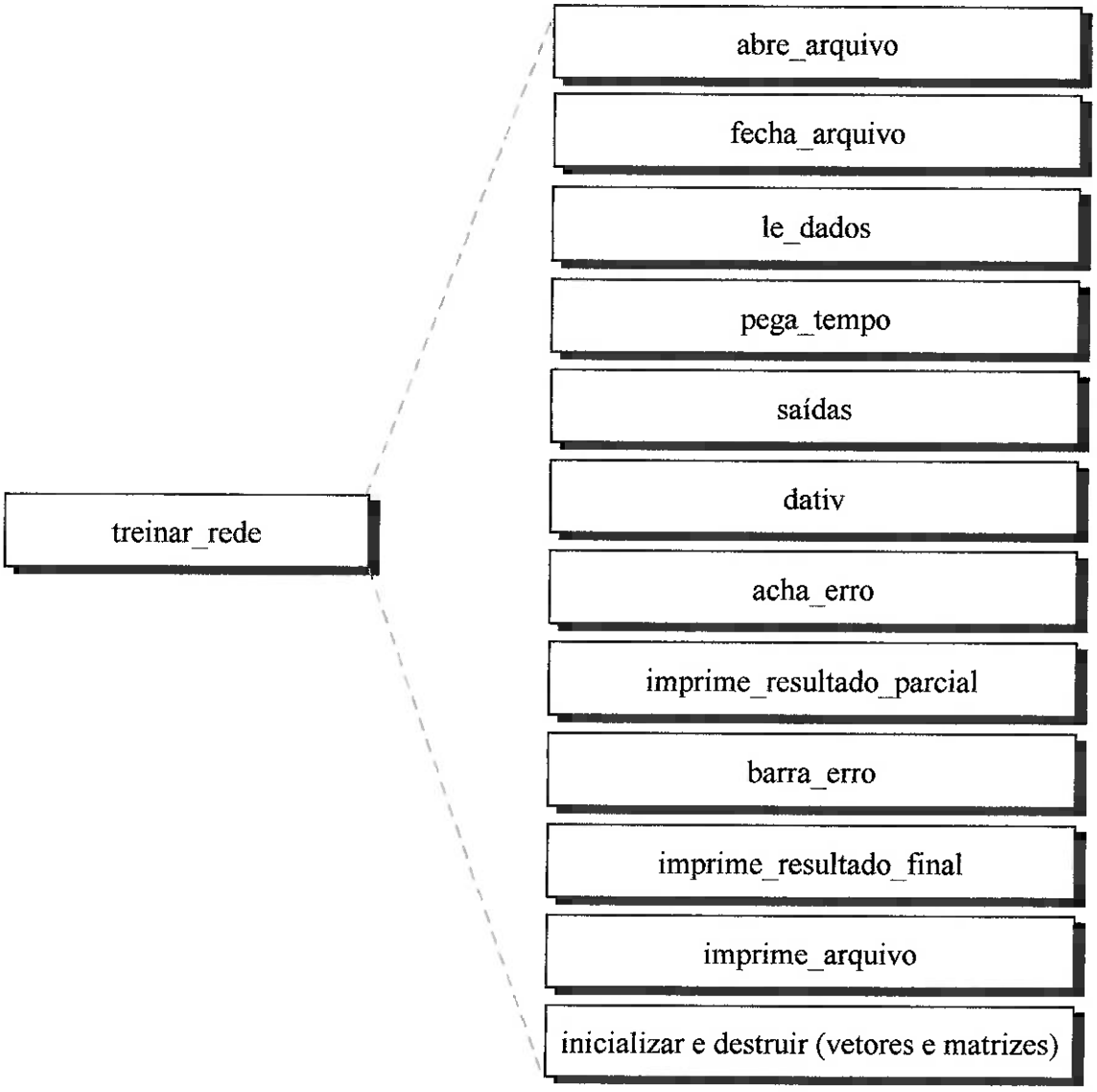


Figura 26 - Programa Backpropagation: Rotina treinar_rede

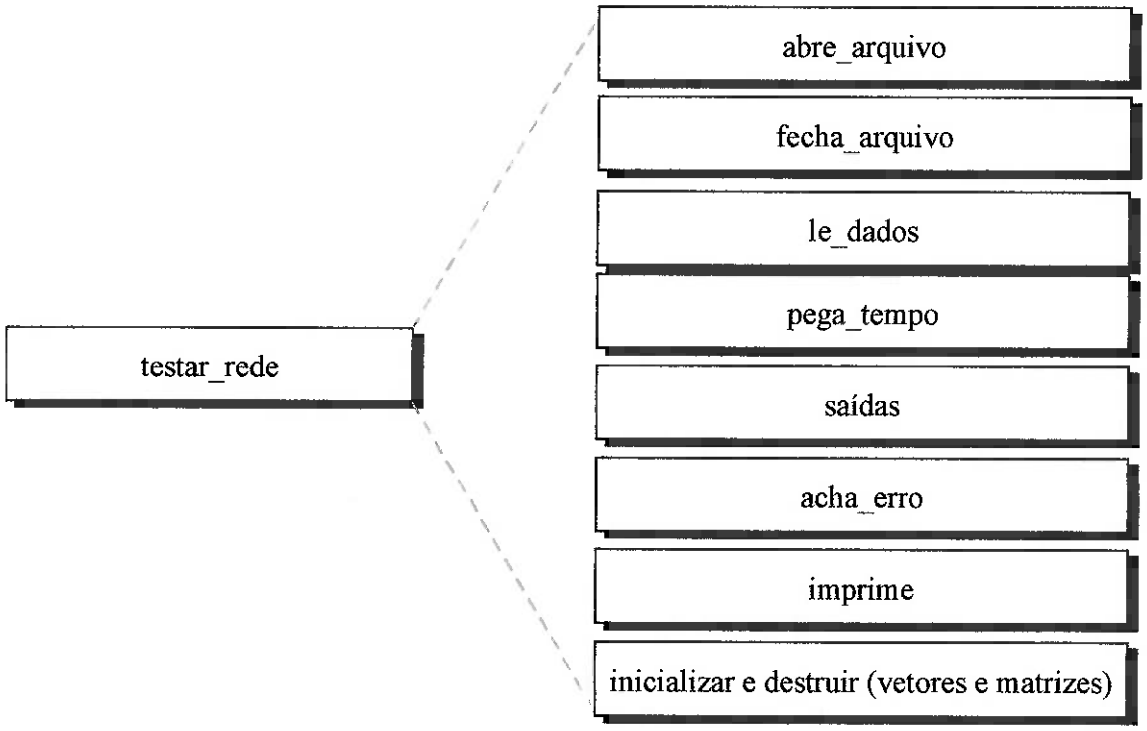


Figura 27 - Programa Backpropagation: Rotina testar_rede

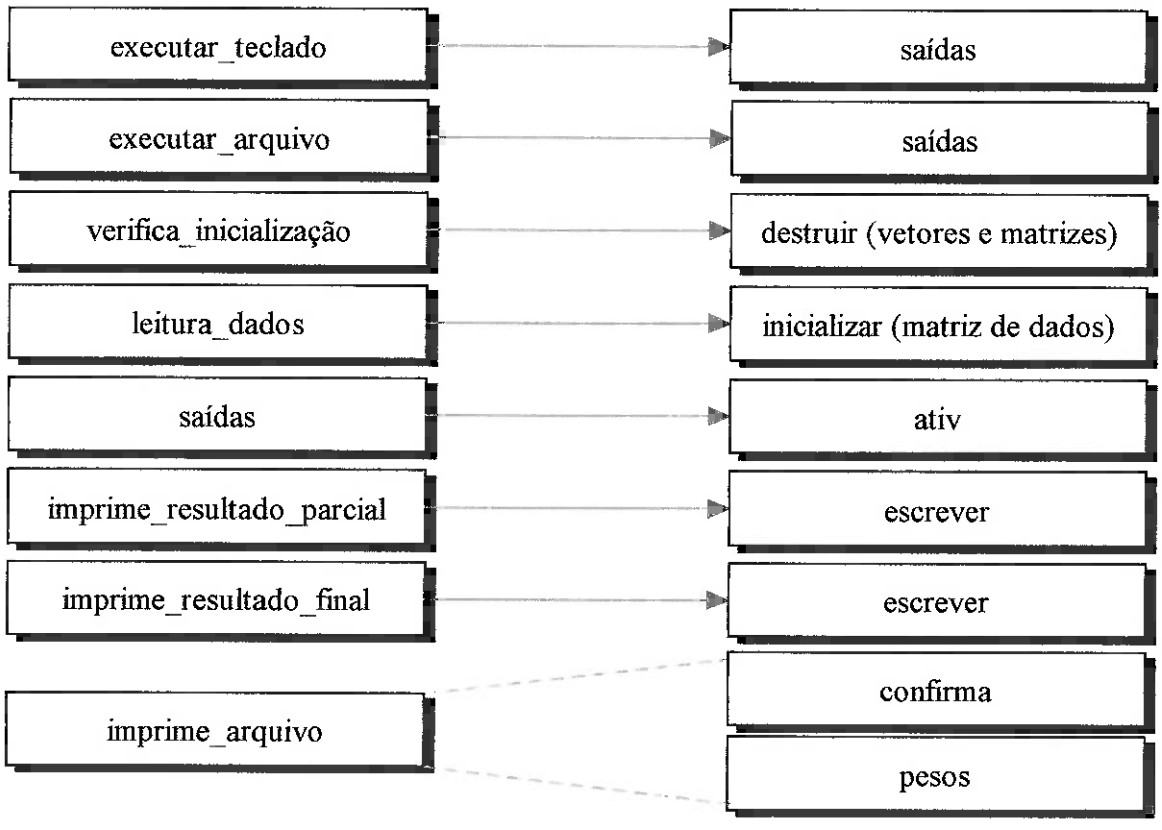


Figura 28 - Programa Backpropagation: Outras rotinas

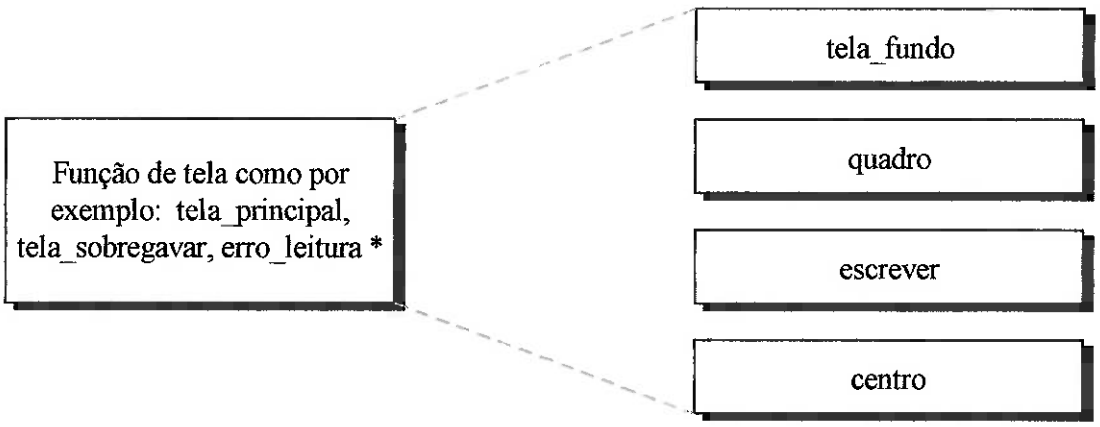


Figura 29 - Programa Backpropagation: Funções básicas de tela

7.4 Os objetos utilizados

A implementação do programa foi feita utilizando-se objetos. Na programação orientada à objetos, tem-se que cada objeto é uma "caixa preta" onde existe o processamento de dados utilizando-se funções que somente alteram variáveis relativas a ele, ou então a outros objetos.

Os objetos utilizados neste programa são os seguintes:

- *vector:* é um objeto que representa um vetor de tamanho *n* e que contém rotinas de inicialização (desde que *n* seja dado anteriormente) e destruição (desde que ele já esteja inicializado).

- *matrix:* é um objeto que representa uma matriz de dimensões *mxn*, dados por *lin* e *col*, e que contém rotinas de inicialização (desde que *lin* e *col* sejam dados anteriormente) e destruição (desde que ele já esteja inicializado).

- *screen:* é um objeto que contém as principais rotinas gráficas do programa como, por exemplo, as rotinas de limpar a tela, de fazer quadros, de apresentar a tela

principal, de apresentar mensagens de erro, etc... Este objeto servirá de base para os outros relacionados às telas gráficas.

- *tela define:* é um objeto que apresenta todas as telas gráficas durante a definição de uma nova rede. Neste objeto serão utilizados todas as funções básicas de tela, ou seja, ele é um objeto derivado do *screen.*

- *tela abrir:* é um objeto que apresenta todas as telas gráficas durante a abertura de uma rede contida em um arquivo. Assim como o anterior, ele é um objeto derivado do *screen.*

- *tela treino:* é um objeto que apresenta todas as telas gráficas durante o treinamento de uma rede neural. Analogamente, ele é um objeto derivado do *screen.*

- *tela execução:* é um objeto que apresenta todas as telas gráficas durante a execução (simulação) de uma rede neural. Analogamente, ele é um objeto derivado do *screen.*

- *impressão:* é um objeto que apresenta todas as telas gráficas durante a impressão dos pesos da rede neural. Analogamente, ele é um objeto derivado do *screen.*

- *tela teste:* é um objeto que apresenta todas as telas gráficas durante o teste de uma rede neural. Analogamente, ele é um objeto derivado do *screen.*

- *treino:* é um objeto relacionado ao treinamento da rede neural. Ele cria variáveis auxiliares (para os dados, por exemplo), além de possuir todo um conjunto de subrotinas que manipulam os pesos até que a rede esteja treinada. Entre essas

subrotinas pode-se destacar as funções *ativ* e *dativ* que representam as funções de ativação (sigmoidal) e sua derivada. Outras rotinas como a leitura dos parâmetros de treinamento e impressão da rede treinada em um arquivo também estão contidas neste objeto.

- *teste:* é um objeto relacionado ao teste da rede neural. Ele cria as variáveis relacionadas ao teste e possui uma rotina de impressão do resultado do teste.

- *net:* é um objeto que representa a rede neural. Nele estão contidos os pesos, os sinais sinápticos da rede e as subrotinas que o manipulam. Dentro destas subrotinas podem-se destacar a subrotina que verifica a inicialização, a que define uma nova rede, a de abertura de uma rede, a de treinamento e a de teste. Ele também possui uma rotina de destruição de todas as variáveis utilizadas quando se finaliza o programa. O objeto *net* é a base deste programa, tendo contato com qualquer objeto quando necessário sem que afete a integridade da estrutura hierárquica do programa.

7.5 O relacionamento entre os objetos

O objeto *net* possui em sua parte privada dois objetos *matrix* e três objetos *vector*. Este é o objeto principal, porque contém a própria rede (representada pelos pesos e sinais sinápticos) e chama todos os outros objetos no decorrer do programa. O objeto é melhor representado na Figura 30:

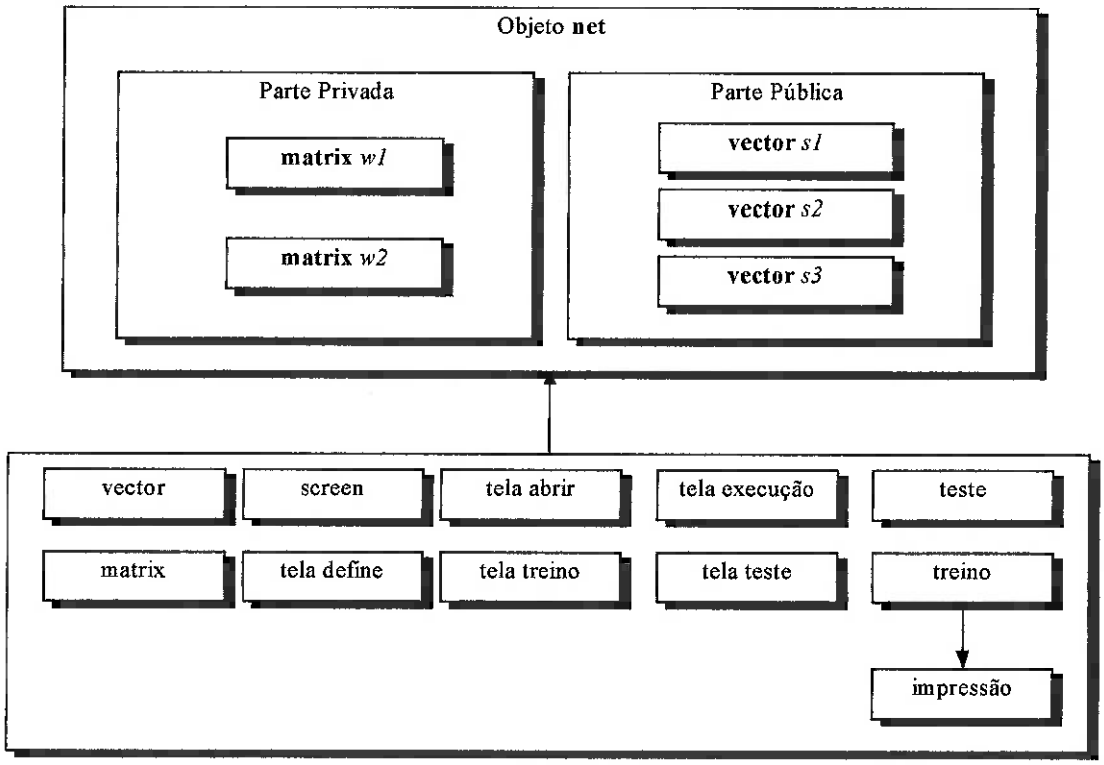


Figura 30 - Relacionamento entre objetos no programa da rede backpropagation

Os objetos de tela (*tela_define*, *tela_abrir*, *tela_treino*, *tela_execução*, *tela_teste* e *impressão*) são derivados do objeto *screen*. Portanto, esses objetos contém todas as funções que *screen* encerra em sua parte pública.

A listagem do programa da rede neural backpropagation encontra-se no apêndice 1, assim como os diagramas de Nassi_Schneidermann e a descrição detalhada das rotinas mais importantes.

8. IMPLEMENTAÇÃO DA REDE COUNTERPROPAGATION

8.1 Introdução

A implementação da rede neural counterpropagation foi feita de forma análoga à rede counterpropagation. Utilizou-se também a programação orientada à objetos na linguagem C++. As rotinas principais são semelhantes às da rede backpropagation

8.2 Descrição do programa implementado

A implementação da rede neural é feita através das matrizes *weight_koh* e *weight_gross* que representam os pesos da camada Kohonen e da camada Grossberg respectivamente. Os vetores *entrada* e *saida* são utilizados para todo o processamento da rede (teste, treino e execução).

A **inicialização** de um rede neural foi feita criando-se as matrizes *weight_koh* e *weight_gross* (através de alocação dinâmica). As dimensões das matrizes são dadas pelo número de entradas, de saídas e de neurônios da camada intermediária. Os pesos iniciais da camada Grossberg foram atribuídos aleatoriamente, enquanto que os pesos da camada Kohonen foram atribuídos segundo os pesos iniciais do **método de combinação convexa**. O método de combinação convexa inicializa os pesos com o valor de $1/\sqrt{n}$. Não foi utilizado o resto deste método por ele necessitar de um tratamento de entradas ao longo do treinamento, o que faz com que o tempo de processamento aumente devido ao fato dos pesos serem ajustados em relação a um alvo móvel

A abertura de uma rede é feita a partir de um arquivo cuja extensão é ***.sz*** (onde são fornecidos o número de entradas, de saídas e de neurônios). Para a leitura dos pesos, utilizaram-se os arquivos de extensão ***.wkh*** e ***.wgr*** definido-se então a rede.

Para o **cálculo das saídas**, utiliza-se o vetor *entrada* e a matriz *weight_koh* para a obtenção do neurônio vencedor. Processa-se então a camada Grossberg, atribuindo ao vetor *saída* o valor encontrado na coluna correspondente ao neurônio vencedor da matriz de pesos *weight_gross*. Isto só pode ser feito porque está sendo utilizado apenas um neurônio vencedor que emite um sinal sináptico de valor "1".

Para o **treinamento**, utilizou-se um arquivo de extensão é ***.trn***. Foram realizados as leituras dos vetores *entrada* e *saída* para um par de treinamento. O vetor *entrada* é processado pela rotina *treino_kohonen*, obtendo-se o vencedor *winner*. Na rotina *treino_kohonen* os pesos da camada Kohonen são ajustados com uma outra rotina *ajusta_kohonen*. Após o treinamento Kohonen (competitivo), processa-se a rotina *treino_grossberg*, que utiliza o vetor *saída* e o *winner*. Na rotina *treino_grossberg*, os pesos são ajustados com uma outra rotina *ajusta_grossberg*. Ao contrário do treinamento da rede backpropagation, o treinamento desta rede é feito em duas etapas. Em cada etapa é perguntado ao usuário os parâmetros de treinamento (coeficiente de aprendizagem, número de iterações e erro). Os **coeficientes de aprendizagem** sofrem decaimento exponencial automático para melhorar a convergência na direção de diminuir o erro. O erro no treinamento Kohonen corresponde à distância euclidiana média das entradas em relação aos pesos da camada Kohonen. Analogamente, o erro no treinamento Grossberg corresponde à distância euclidiana média das saídas em relação aos pesos da camada Grossberg.

Após o treinamento Kohonen o programa pergunta ao usuário se que iniciar o treinamento Grossberg. Caso não se queira continuar, o programa volta à tela principal.

Com isso, foi possível corrigir os pesos iterativamente até que o erro fosse menor do que um valor admissível, ou então até que o número máximo de iterações fosse atingido.

No treinamento Kohonen, para que o tempo de treinamento fosse reduzido e a convergência fosse aumentada, optou-se por diversas soluções. A primeira seria o de **inicializar os pesos da rede através da média dos arquivos de treinamento**. Isto foi possível utilizando-se o programa de geração e tratamento de sinais para a rede counterpropagation, que cria arquivos próprios para possibilitar a abertura na rede no programa de rede neural. Este método diminui o tempo inicial de treinamento, mas não garante a sia convergência.

Existe uma outra opção bastante similar que é o "**treinamento Kohonen supervisionado**". Neste treinamento, as entradas são apresentadas à rede neural e, utilizando-se da saída, obriga-se um certo neurônio a ser o vencedor. A partir disso, os pesos conectados a este neurônio são ajustados. Este método não aumenta muito a eficiência do treinamento e só pode ser aplicado quando utiliza-se apenas uma saída binária ("0" ou "1") e quando tem-se apenas dois neurônios na camada Kohonen.

Em uma outra alternativa, tem-se o vetor *winners* do objeto *treino* que faz o papel similar ao conceito de "**consciência**" proposto por Di Sieno (1988). O vetor *winners* marca quantas vezes cada neurônio foi considerado vencedor. Durante o treinamento Kohonen, se algum neurônio não foi considerado vencedor (valor "0" para o elemento

do vetor *winners*), corrige-se os pesos conectados a este neurônio atribuindo-lhes os valores dos pesos correspondentes ao neurônio que mais vezes foi vencedor. Isto foi feito para que o vetor de pesos ser aproximasse mais da região da hiper-esfera onde estão concentrados os vetores das entradas.

Para o **teste**, o procedimento inicial é análogo ao do treinamento, porém, não há a correção dos pesos. Por esta razão, os objetos utilizados no teste são os mesmos que o do treinamento.

A **execução da rede neural** (cálculo de ume saída dada a entrada) é feita simplesmente chamando o mesmo procedimento utilizado no treinamento e no teste. O cálculo do erro e a correção dos pesos não são efetuados e a entrada é dada através do teclado de um arquivo.

A arquitetura do software é apresentada a seguir mostrando a hierarquia e a relação entre as partes funcionais do sistema.

8.3 Arquitetura do Programa

A hierarquia entre as rotinas e os objetos estão descritas logo a seguir:

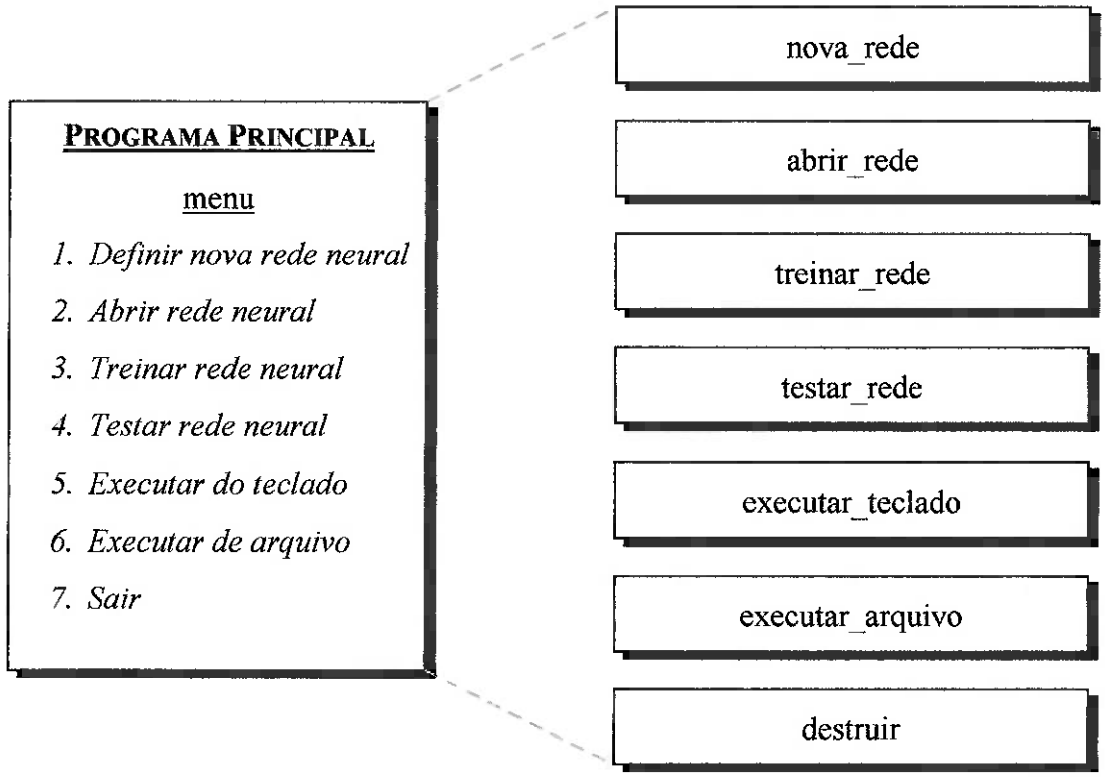


Figura 31 - Arquitetura principal do programa counterpropagation

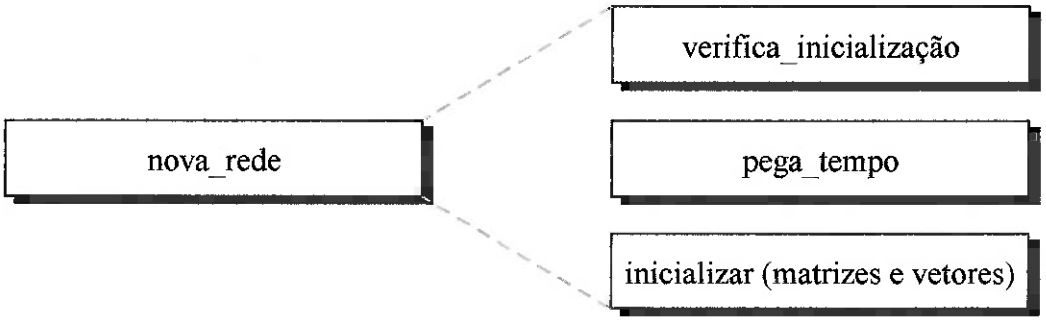


Figura 32 - Programa Counterpropagation: Rotina nova_rede

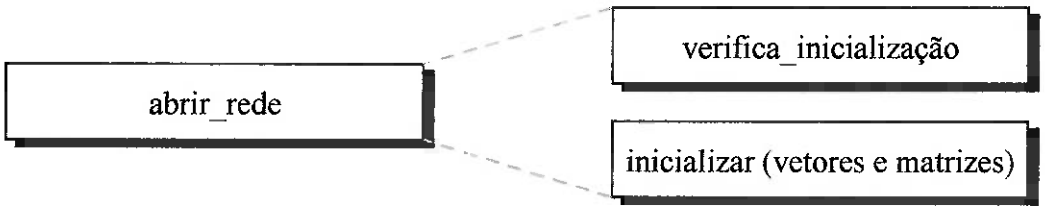


Figura 33 - Programa Counterpropagation: Rotina abrir_rede

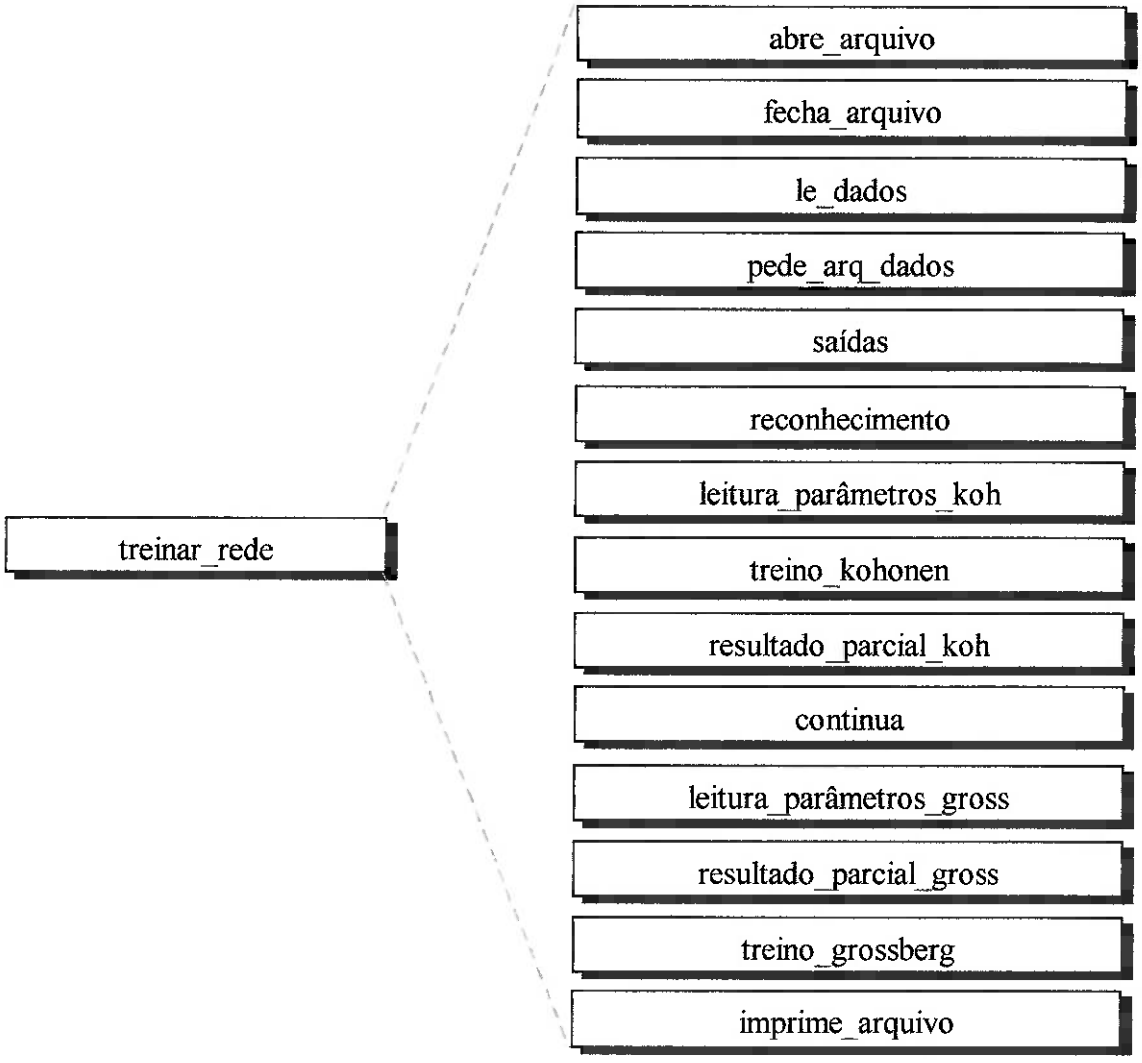


Figura 34 - Programa Counterpropagation: Rotina treinar_rede

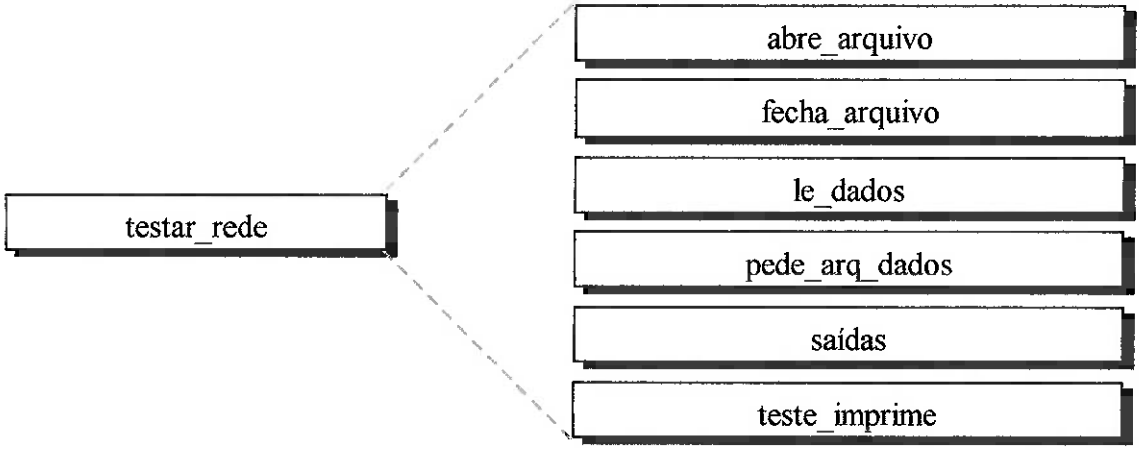


Figura 35 - Programa Counterpropagation: Rotina testar_rede

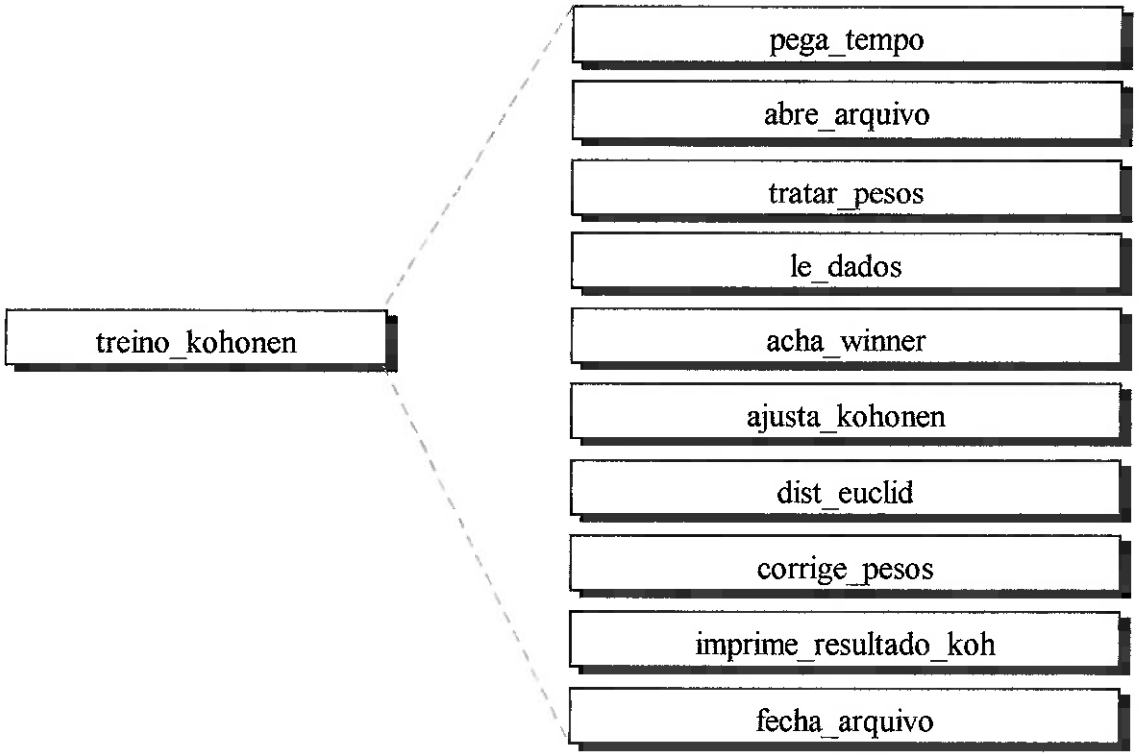


Figura 36 - Programa Counterpropagation: Rotina treino_kohonen

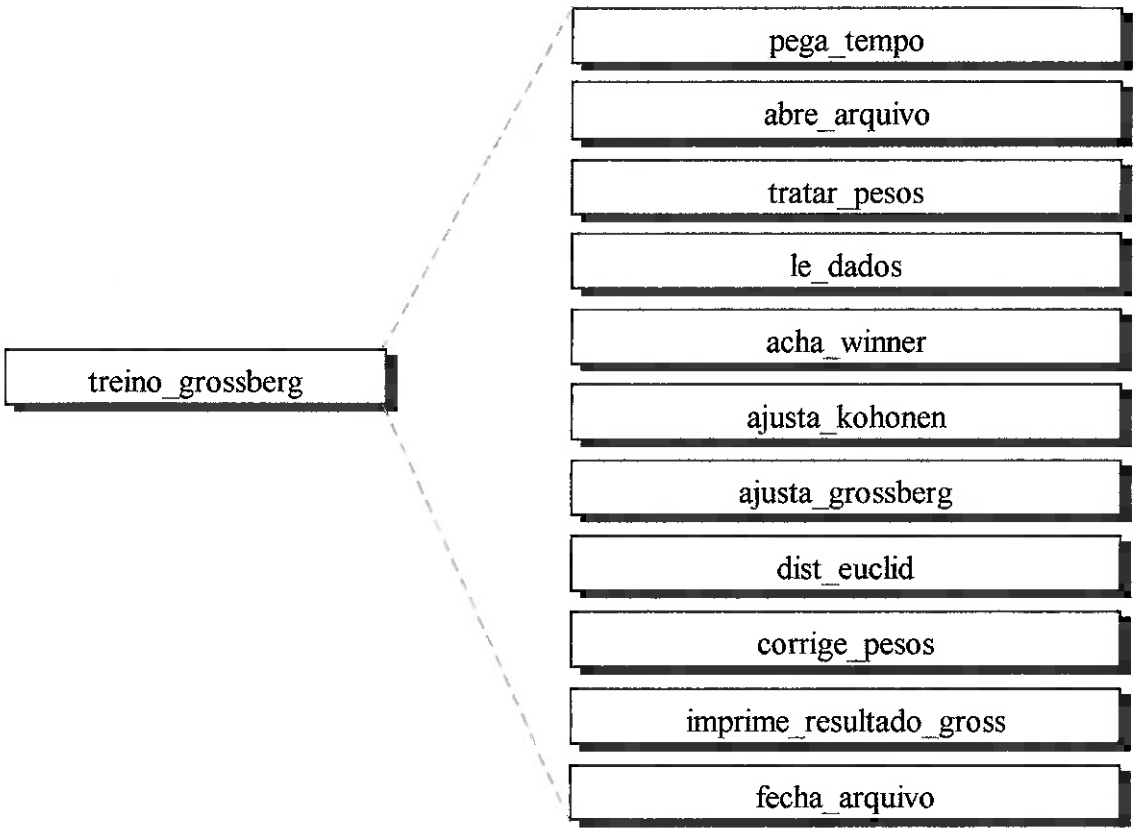


Figura 37 - Programa Counterpropagation: Rotina treino_grossberg

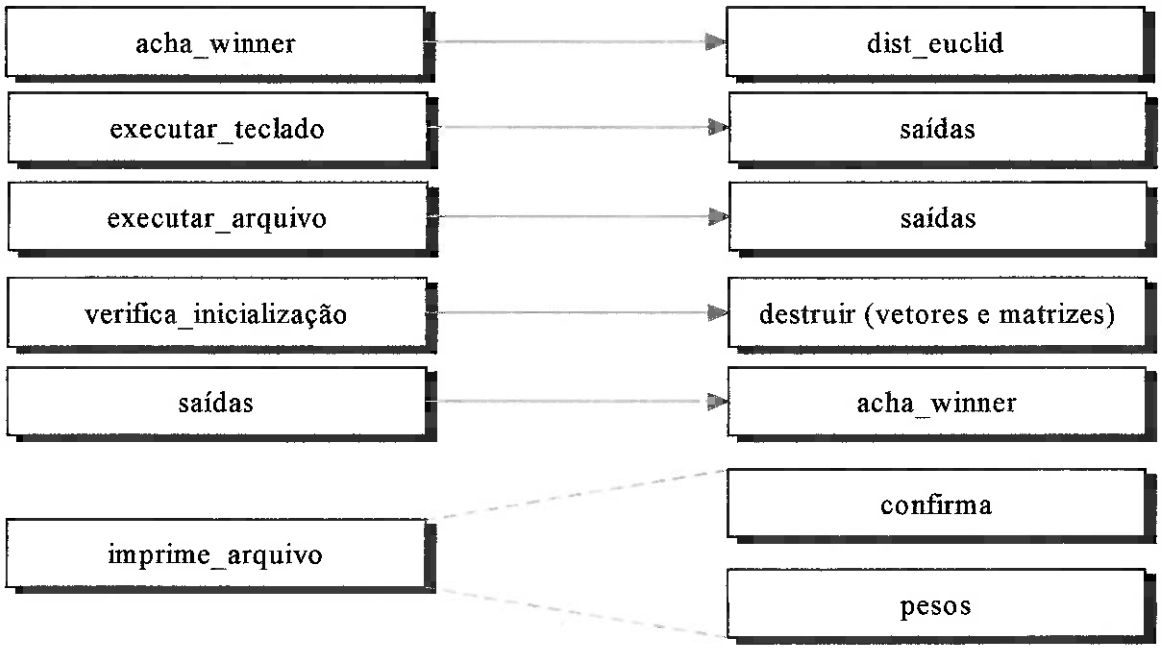


Figura 38 - Programa Counterpropagation: Outras rotinas

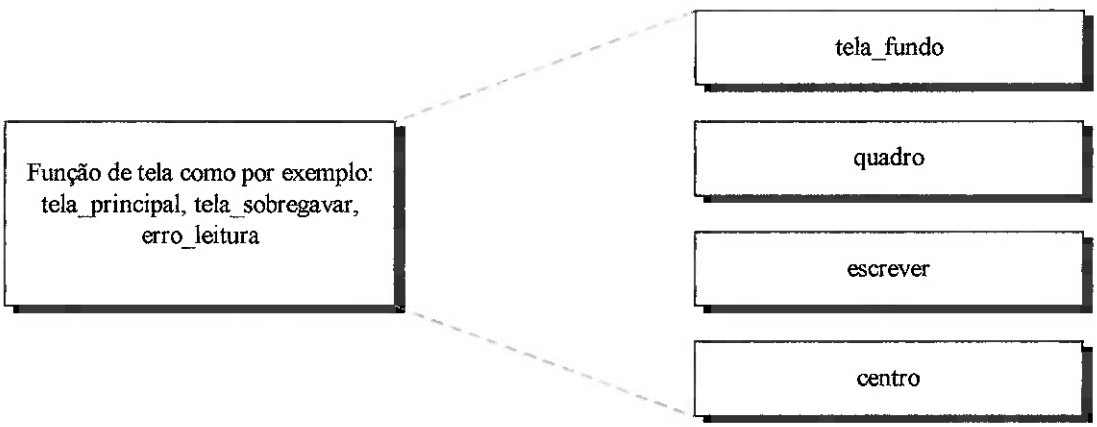


Figura 39 - Programa Counterpropagation: Funções básicas de tela

8.4 Os objetos utilizados

A implementação do programa counterpropagation foi feita tomando-se como base o programa da rede backpropagation. Portanto, vários objetos descritos anteriormente foram reutilizados não necessitando de maiores esclarecimentos. Estes objetos são: *vector*, *matrix*, *screen*, *tela_abrir*, *tela_define*, *tela_treino*, *tela_execução* e *impressão*. Os demais objetos que são próprios do programa de rede neural counterpropagation são os seguintes:

- *treino:* é um objeto relacionado ao treinamento e ao teste da rede neural. Ele cria variáveis auxiliares, como por exemplo o vetor *winners* que representa a "consciência", e possui todo um conjunto de subrotinas que manipulam os pesos até que a rede esteja treinada. Entre essas subrotinas pode-se destacar as funções *abre_arquivo*, *lê_dados*, *leitura_parametros_koh*, *leitura_parametros_gross* e *teste_imprime*.

- *net:* é um objeto que representa a rede neural. Nele estão contidos os pesos, os sinais de entrada e saída da rede, o neurônio vencedor e as subrotinas que o manipulam. Dentro destas subrotinas podem-se destacar a subrotina que verifica a inicialização, a que define uma nova rede, a de abertura de uma rede, a de treinamento (Kohonen e Grossberg) e a de teste. Ele também possui uma rotina de destruição de todas as variáveis utilizadas quando se finaliza o programa. É neste objeto que estão contidos as subrotinas de manipulação dos pesos durante o treinamento e também de determinação do neurônio vencedor. O objeto *net* é a base deste programa, tendo contato com qualquer objeto quando necessário sem que afete a integridade da estrutura hierárquica do programa.

8.5 O relacionamento entre os objetos

O objeto *net* possui em sua parte privada dois objetos *matrix* e três objetos *vector*. Este é o objeto principal, porque contém a própria rede (representada pelos pesos e sinais sinápticos) e chama todos os outros objetos no decorrer do programa. O objeto é melhor representado na Figura 40:

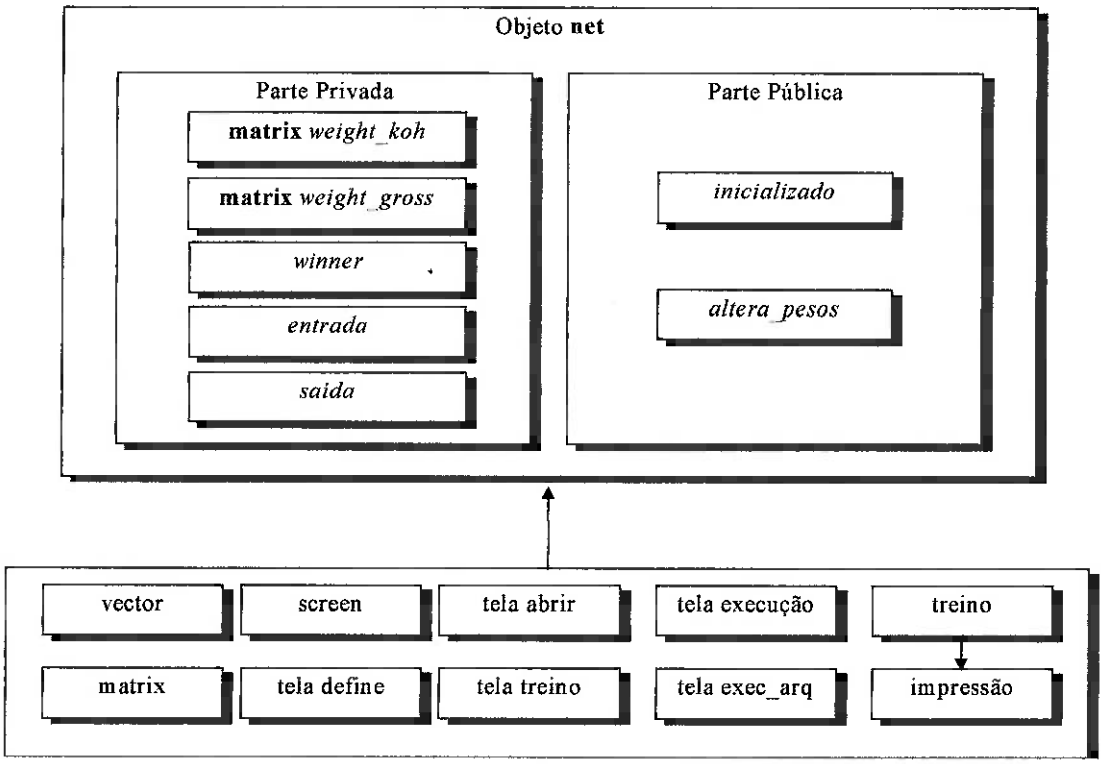


Figura 40 - Relacionamento entre objetos no programa da rede counterpropagation

Os objetos de tela (*tela_define*, *tela_abrir*, *tela_treino*, *tela_execução*, *tela_exec_arq* e *impressão*) são derivados do objeto *screen*. Portanto, esses objetos contém todas as funções que *screen* encerra em sua parte pública.

A listagem do programa da rede neural counterpropagation encontra-se no apêndice 2, assim como os diagramas de Nassi_Schneidermann e a descrição detalhada das rotinas mais importantes.

9. PRÉ-PROCESSAMENTO DOS SINAIS DE ENTRADA

Com o intuito de fazermos com que as redes neurais implementadas fossem as mais genéricas possíveis, foi implementado um pré-processamento dos sinais de entrada para que fosse possível a aplicação das redes no reconhecimento de sinais (objetivo do projeto).

Inicialmente dois sinais são obtidos em uma amostragem da aplicação. A freqüência de amostragem deve ser maior do que duas vezes a freqüência do sinal para que o Teorema da Amostragem seja atendido e a leitura seja correta. Os pontos de cada sinal são gravados seqüencialmente em um arquivo texto (*.txt) e, a partir destes dois arquivos, foram implementados duas abordagens diferentes.

A primeira abordagem seria o de gerar arquivos para o treinamento da rede backpropagation, considerando-se que as entradas da rede estariam atrasadas uma em relação às outras. A segunda seria a de um tratamento do sinal transformando a sua dinâmica em uma imagem tridimensional para que possa ser utilizada na rede counterpropagation. Estas abordagens serão detalhadas logo a seguir e o motivo da aplicação nas respectivas arquiteturas de redes neurais será discutido no capítulo de testes e resultados da rede neural backpropagation.

9.1 Entradas da rede neural atrasadas uma em relação às outras

O conceito de atraso nas entradas da rede neural é uma tentativa de se capturar uma dinâmica utilizando-se a rede backpropagation na sua forma mais simples. O conceito é melhor representado pela Figura 41, onde a entrada x_1 representa a entrada em um

instante de amostragem k, x_2 representa a amostragem no instante seguinte $k+1$, e assim por diante até $k+n$ (onde n é o número de amostragens realizadas).

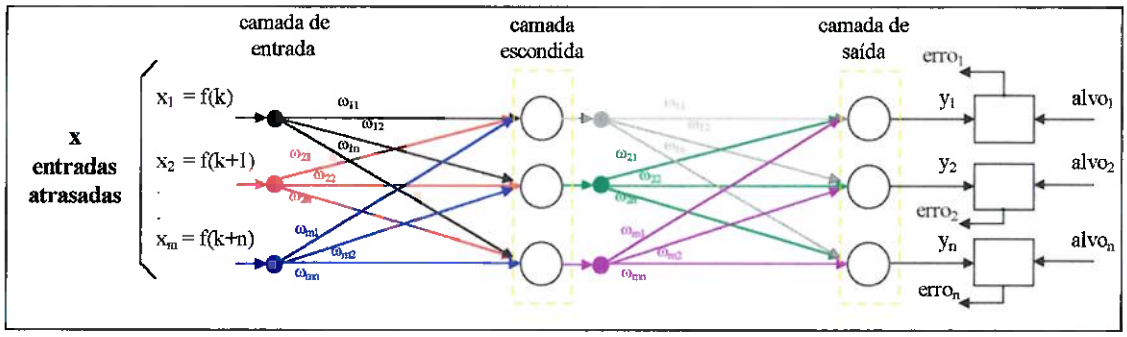


Figura 41 - Rede Backpropagation com entradas atrasadas

Os dados do arquivo do sinal foram lidos e colocados em um vetor. Iniciou-se então a distorção do sinal mantendo-se o fator de potência. Com um número razoável de entradas distorcidas, pode-se construir um arquivo de treinamento (*.trn) adequado à rede neural. A construção deste arquivo de treinamento foi uma simples gravação dos pontos com distorção de forma seqüencial, garantindo-se apenas que os pontos correspondam às respectivas entradas.

Um exemplo dos sinais de entrada para o arquivo de treinamento é dado na Figura 42.

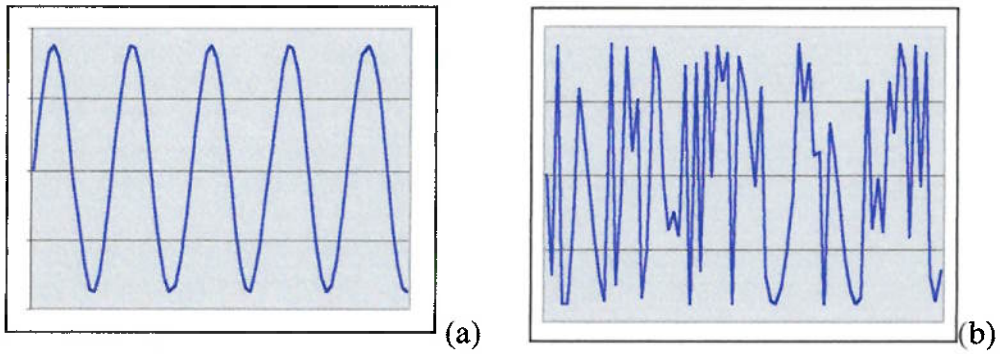


Figura 42 - Entrada para a rede Backpropagation: (a)senoidal; (b)senoidal com distorção

O programa de geração do arquivo de treinamento para a rede backpropagation foi implementado de forma bastante simples na linguagem C (ver tela inicial na Figura 43) e pode ser encontrado no apêndice 3.

Figura 43 - Tela principal do programa de geração de entradas para a rede Backpropagation

9.2 Transformação da dinâmica do sinal em uma superfície

Com o intuito de melhorar a convergência do treinamento da rede neural counterpropagation e o correspondente desempenho na classificação, o sinal é pré-processado antes de ser aplicado nas entradas da rede. Com os sinais contidos em dois vetores (lidos dos arquivos *.txt), procede-se o mapeamento segundo a função caos modificada, que pode ser representada pela Figura 44.

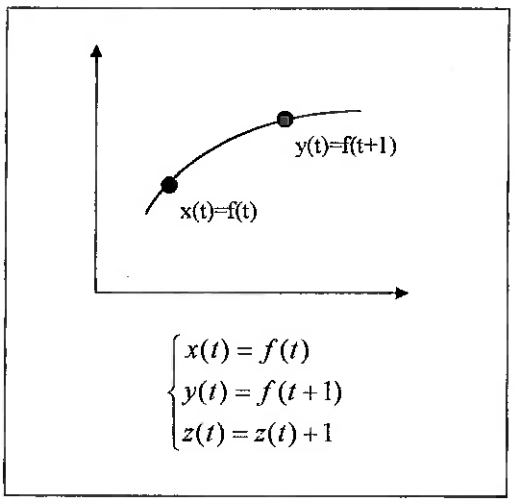


Figura 44 - Mapeamento da dinâmica do sinal

A modificação da função caos está no incremento de *z(t)*. Isto foi aplicado para que pontos com freqüências mais comuns não sejam desprezados. Entretanto, com esta aplicação, pode-se ter picos em algumas regiões que fazem com que o efeito dos trechos de menor freqüência sejam desprezados. Para evitar este problema, utilizou-se uma compressão-expansão dos sinais com o auxílio de uma progressão geométrica de razão 0,5:

$$z(x,y) = \frac{1-0,5^{z(x,y)}}{1-0,5}$$

Com esta aplicação, a matriz fica limitada a valores entre 0 e 2 tendo duas vantagens sobre outras funções (como log ou sigmoidal):

- é computacionalmente mais simples a determinação do valor
- oferece maior controle no comportamento assintótico.

Com este tratamento, pode-se obter a Figura 45, que representa uma onda senoidal.

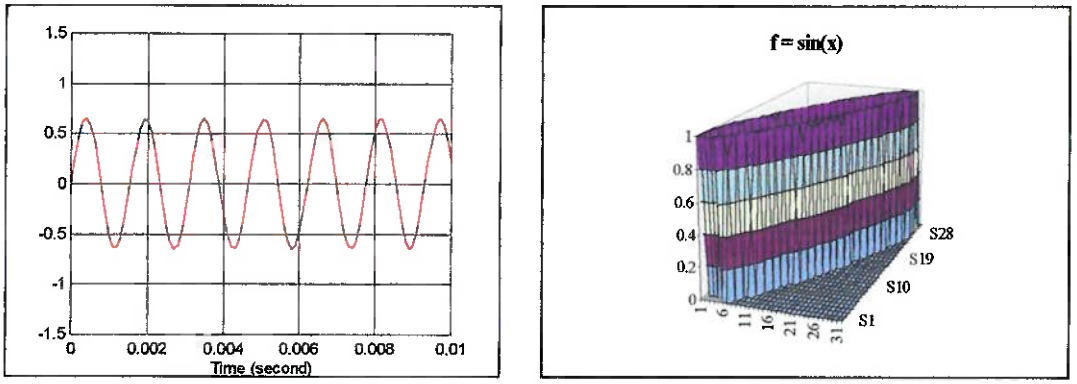


Figura 45 - Tratamento de um sinal senoidal

Aplicando a mesma distorção que a praticada no pré-processamento das entradas da rede backpropagation, pode-se obter a Figura 46 que corresponde a um sinal senoidal com 40% de distorção.

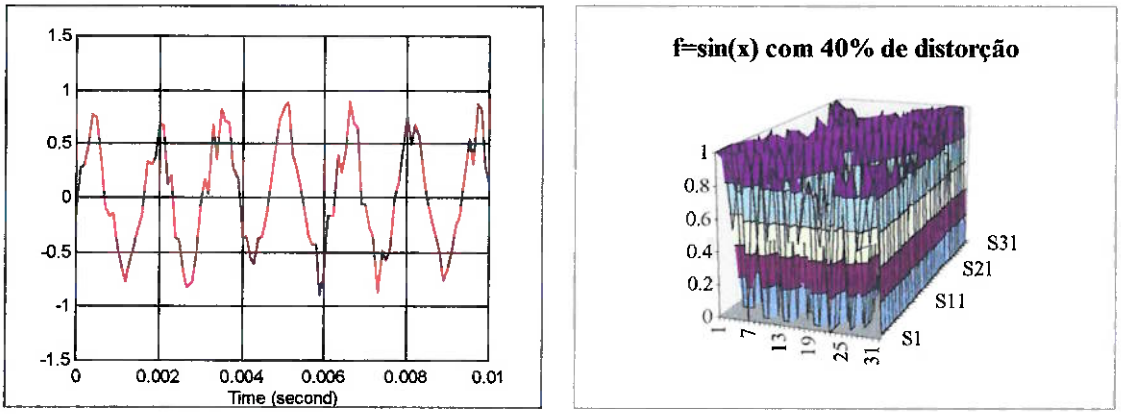


Figura 46 - Tratamento de um sinal senoidal com 40% de distorção

O programa de tratamento de sinal e geração do arquivo de treinamento para a rede counterpropagation foi implementado de forma bastante simples na linguagem C (ver tela principal na Figura 47) e sua listagem pode ser encontrado no apêndice 4. Existe uma opção de inicialização de pesos que cria uma rede neural com a média dos pesos (melhora o treinamento da rede).

Figura 47 - Tela do programa de tratamento de sinais para a rede Counterpropagation

10. VALIDAÇÃO E VERIFICAÇÃO - FORMULAÇÃO

10.1 Introdução

Antes de ser utilizado, um sistema computacional deve ser avaliado em vários aspectos. Com isso, a validação do desempenho pode ser o mais significante. A validação refere-se a determinar se o sistema tem um certo desempenho em um nível aceitável de precisão e eficiência. A verificação, que é um passo anterior à validação, refere-se a determinar se o sistema foi corretamente implementado. Por isso, um sistema precisa ser verificado primeiro para depois ser validado.

10.2 Verificação

De forma resumida, a verificação de um sistema computacional de redes neurais deve englobar os seguintes aspectos:

- *Propriedades da rede:* arquitetura (camada única ou múltiplas camadas), fluxo de informações (alimentação positiva ou realimentação/recorrente), e padrões de interconexões (totalmente ou parcialmente conectadas).
- *Propriedades da célula:* conexões de entrada e saída e não linearidades (funções de ativação logística ou tangente-hiperbólica).
- *Propriedades neurodinâmicas:* inicialização dos pesos, cálculo das ativações, e ajuste dos pesos.

Um modelo de rede neural é usualmente caracterizada por um conjunto de equações matemáticas. A verificação da implementação correta destas equações é de extrema importância.

Sob o ponto de vista dos procedimentos, é importante a verificação das seguintes propriedades:

- *Convergência:* verificar se a rede converge para um valor após iterações.
- *Estabilidade:* verificar se sucessivas iterações produzem menores oscilações na saída.

10.3 Validação

A definição de "desempenho" varia com a natureza das aplicações. Para aplicações de classificação, o desempenho pode ser definido em termos de uma taxa de acertos sobre o total de casos testados.

O'Keefe, Balci e Smith (1987) separaram metodologia de validação em métodos quantitativos e qualitativos. Métodos qualitativos adotam comparações subjetivas de desempenho, enquanto que métodos quantitativos empregam as técnicas estatísticas para comparar desempenho. Definindo de outra maneira, validação qualitativa refere-se avaliação de resultados como respostas categóricas, enquanto que a validação quantitativa envolve resultados numéricos.

Em validações quantitativas, os métodos como testes de hipóteses são amplamente aplicados de forma satisfatória. Utiliza-se usualmente o teste t-Student ou o Qui-quadrado para a validação, entretanto, no caso do reconhecimento de sinais estes testes não podem ser aplicados.

10.3.1 Teste de validação

Na aplicação deste projeto, deve-se considerar o arranjo em pares de possibilidades de respostas, ou seja, (sim sim), (não não), (sim não) e (não sim). Deve-se utilizar a estimativa de quantas vezes estes eventos ocorrem. nos diferentes grupos.

Para simplificar o cálculo das variáveis inerentes ao teste de aplicação, pode-se utilizar um teste de proporção considerando-se que os erros de classificação encontram-se distribuídos uniformemente.

Para comprovar a validação da rede neural aplicada no reconhecimento de sinais, considera-se o seguinte teste:

- Assume-se a hipótese:

$$H_0: p = p_0$$

onde p representa a proporção de classificação errada do sistema e p_0 a proporção de classificação errada máxima desejada.

Caso não haja evidências de se rejeitar a hipótese H_0, aceito-a como verdadeira, ou seja, a rede classifica de maneira satisfatória. O teste a ser realizado deve ser unilateral, que confirma que o sistema tem no máximo um erro de classificação dado por p_0 .

- Estipular um nível de significância α (probabilidade do erro tipo I, ou seja, probabilidade de se rejeitar a hipótese H_0 sendo ela verdadeira).
- Dado o nível de significância α, calcula-se o valor de z_α pela tabela da distribuição normal reduzida.

- Para que a hipótese H_0 não seja rejeitada (rede satisfatória) deve-se ter obrigatoriamente:

$$\left|\frac{(p-p_0)\pm 1/2n}{\sqrt{p_0(1-p_0)/n}}\right| < z_\alpha$$

- Portanto, pode-se ter um valor máximo de p que, quando comparada com os resultados das amostras obtidas, torna possível concluir a validade desta rede.

10.3.2 Tamanho da amostra a ser utilizada

Dado um grau de significância α, uma precisão e_0 e a proporção a ser considerada p, pode-se calcular o tamanho da amostra a partir de:

$$n = \left(\frac{z_{\alpha/2}}{e_0}\right)^2 p(1-p)$$

Para se verificar a probabilidade de se cometer o erro tipo II, para isso deve-se calcular β a partir de:

$$n = \left(\frac{z_\alpha\sqrt{p_0(1-p_0)} + z_\beta\sqrt{p(1-p)}}{p-p_0}\right)^2$$

onde p é o valor da proporção populacional além do qual é fixado em, no máximo, β a probabilidade de se cometer o erro tipo II (aceitar a hipótese H_0 sendo ela falsa).

11. TESTES E RESULTADOS

Para comprovar-se a validade da aplicação de redes neurais no reconhecimento dos sinais acústicos, foram realizados diversos testes de verificação e validação.

Alguns testes simples de verificação dos programas foram realizados para verificar-se o sistema. As propriedades mais observadas no programa foram a convergência, estabilidade e robustez do sistema. Observou-se também a generalidade do sistema. Os testes realizados para a rede backpropagation foi o simulação de lançamento de projétil, enquanto que o da rede counterpropagation foi o de classificação simples.

Portanto, será ressaltado os testes de validação da rede aplicados ao escopo do problema. Os procedimentos adotados serão descritos logo a seguir.

11.1 Cálculo da proporção de erros máxima desejada

A proporção de erros máxima admissível será calculada, admitindo-se que existem quatro classificações possíveis:

saída obtida	saída desejada	classificação	proporções de erro
sim	sim	correta	0.25
não	não	correta	0.25
sim	não	incorreta	0.25
não	sim	incorreta	0.25

Outra restrição será imposta à proporção de 0,25 de erros obtida. Considerando-se uma rede counterpropagation de 20 neurônios, a probabilidade de que um deles classifique errado é de:

$$p = \frac{1}{20} \cdot 0,25 \quad \Rightarrow \quad p = 0,0125$$

Esta proporção de 0,0125 será utilizado nos testes a serem realizados posteriormente por se tratar tornar o teste mais rigoroso do que o real, já que será utilizado redes com menos de 20 neurônios.

11.2 Cálculo da proporção de erros máxima permitida

Fixado um nível de significância α de 1%, obtém-se:

- $z_\alpha = 2,325$ (obtido pela tabela normal)
- calculando-se o tamanho da amostra:

$$n = \left(\frac{2,325}{0,01}\right)^2 0,0125(1-0,0125) \quad \Rightarrow \quad n = 667$$

Será utilizado um tamanho de amostra *n=1000* para que se tenha uma margem de segurança.

- $\left|\frac{(p-0,0125) \pm 1/(2 \cdot 1000)}{\sqrt{0,0125(1-0,0125)/1000}}\right| < 2,325 \quad \Rightarrow \quad p = 0,0212$

Portanto, obtém-se que a proporção de erro máxima permitida é de 0,0212.

11.3 Testes de validação realizados para a rede backpropagation

11.3.1 Sinais com degradação

Para a rede neural backpropagation, utilizou-se o programa de geração de sinais, obtendo-se arquivos de treinamento com 10, 15, 20, 25 e 30% de degradação, nomeando-os como RN1, RN2, RN3, RN4 e RN5. Estes arquivos continham 50 padrões de treinamento dos dois tipos de sinais. Para que não houvesse a hipótese do vício da rede neural em relação aos dados de treinamento, foram criados 10 amostras contendo 1000 conjuntos de entradas cada para cada tipo de degradação. As redes neurais foram criadas segundo os parâmetros:

- número de entradas: 256
- número de saídas: 1
- número de neurônios na camada intermediária: 6

As redes foram treinadas segundo os seguintes parâmetros:

- coeficiente de aprendizagem: 0.7
- coeficiente de momento: 0.01
- escalonamento a cada 250 iterações
- fator de escalonamento para o coeficiente de aprendizagem: 0.99
- fator de escalonamento para o coeficiente de momento: 0.99
- número máximo de iterações: 5000
- erro quadrático máximo: 0.001

A tela do programa encontra-se na Figura 48.

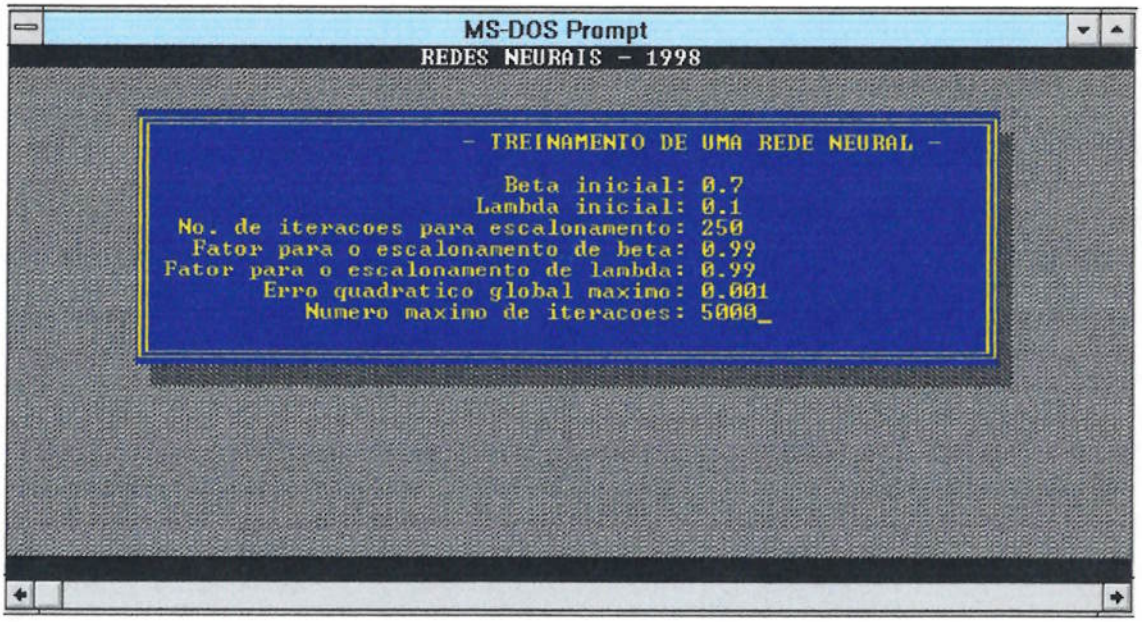


Figura 48 - Treinamento da rede backpropagation para os testes

Obteve-se resultados de treinamento como pode ser observado na Figura 49.

Figura 49 - Resultado do treinamento da rede backpropagation

A média de tempo de treinamento é de 2 minutos.

Foram criadas e treinadas 5 redes neurais diferentes. Com os padrões de 1000 conjunto de dados cada, as redes foram submetidas aos testes de validação. Os resultados foram:

Proporção dos erros no teste de validação da rede backpropagation para 10 a 50% de degradação

	grau de degradação				
Rede neural	10%	20%	30%	40%	50%
RN1	0.001	0.001	0.001	0.002	0.002
RN2	0.001	0.001	0.001	0.001	0.002
RN3	0.001	0.001	0.001	0.001	0.001
RN4	0.000	0.000	0.001	0.001	0.000
RN5	0.000	0.000	0.000	0.000	0.000

Obs: valores obtidos com a média de 10 amostras de 1000 dados cada.

Observa-se que a rede neural é satisfatória para todos os casos, pois para todos os arquivos de teste a proporção de erros manteve-se abaixo do valor de 0.0212 admissíveis.

11.3.2 Sinais com harmônicas

Foram realizados também testes com a terceira e quinta harmônica do sinal. Novas redes foram treinadas incluindo-se os dados relativos às harmônicas no arquivo de treinamento. As redes neurais RN6 e RN7 correspondem às redes treinadas com 10% e 30% de degradação do sinal com 3a. harmônica, enquanto que as redes RN8 e RN9 correspondem à 10 % e 30% de degradação do sinal com 3a. e 5a. harmônica Os resultados do teste são:

Proporção dos erros no teste de validação da rede backpropagation para 3a. e 5a. harmônicas

	grau de degradação			
Rede neural	15% (3a. harm)	30% (3a. harm)	15% (3a. e 5a. harm)	30% (3a. e 5a. harm)
RN6	0.001	0.002	0.167	0.211
RN7	0.001	0.001	0.141	0.207
RN8	0.001	0.002	0.001	0.002
RN9	0.000	0.001	0.000	0.001

Obs: valores obtidos com a média de 10 amostras de 1000 dados cada.

Constata-se que após o treinamento da rede neural com a 3a. e 5a. harmônica, obtém-se um bom reconhecimento dos sinais do arquivo de testes.

11.3.3 Perda dos pontos iniciais do sinal

Foram realizados testes para o caso de se perder os pontos iniciais do sinal. Neste caso, tendo perdidos k pontos, a entrada x_0 é o valor $f(k)$, ou seja, a primeira entrada já está atrasada de k, fazendo com que as outras também esteja. O número de pontos para serem colocados nas entradas deve ser o suficiente para que todas as entradas contenham seus respectivos pontos. Esta perda de pontos iniciais corresponderia a um adiantamento ou atraso de fase numa onda senoidal.

Para que fosse realizado o teste, foram retirados os 10 primeiros pontos dos sinais que foram amostrados do sistema. Foram amostrados 320 pontos, do qual será utilizado apenas 256, ou seja, a perda de 10 pontos ainda mantém todas as entradas preenchidas, mas com o atraso de 10/(freqüência de amostragem em Hz) segundos.

A figura abaixo mostra os pontos iniciais que foram retirados.

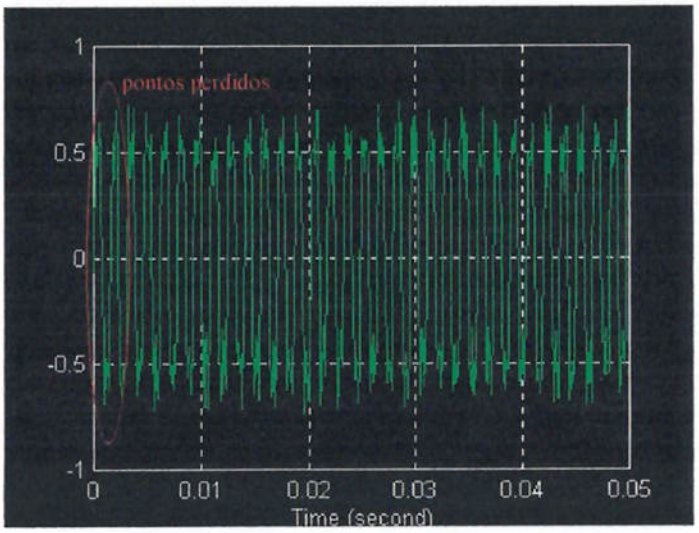


Figura 50 - Problema de perda de pontos iniciais do sinal amostrado

Aplicando-se as redes neurais já treinadas e testadas na validação do item 11.3.1 e treinando-as novamente com a inclusão de novos padrões de treinamento obtidos com a perda dos pontos iniciais do sinal, obtém-se os seguintes resultados utilizando-se arquivos de teste contendo padrões com e sem perda de pontos iniciais.

Proporção dos erros no teste de validação da rede backpropagation para 10 a 50% de degradação com perda dos pontos iniciais no sinal amostrado

	grau de degradação do sinal (com perdas nos pontos iniciais)				
Rede neural	10%	20%	30%	40%	50%
RN1	0.221	0.301	0.275	0.189	0.359
RN2	0.134	0.124	0.136	0.216	0.278
RN3	0.133	0.128	0.138	0.145	0.241
RN4	0.126	0.131	0.132	0.142	0.244
RN5	0.145	0.135	0.145	0.161	0.196

Obs: valores obtidos com a média de 10 amostras de 1000 dados cada.

Portanto, observa-se que os resultados não estão satisfatórios, já que as proporções de erros estão acima de 0.0212.

11.3.4 Discussão sobre os resultados obtidos pela rede backpropagation

De acordo com os resultados obtidos anteriormente, foi possível constatar que a rede backpropagation possui um alto poder de classificação para um tempo de treinamento pequeno. Entretanto, isto pode ser aplicado apenas nos casos mais simples. Para o caso da perda de pontos iniciais do sinal, o erro de classificação aumenta de tal forma que torna a rede inadequada para a aplicação. Como a possibilidade de ocorrência deste problema é alta, a interpretação dos resultados pode ser errônea. Portanto, outras abordagens devem ser consideradas para esta aplicação.

11.4 Testes de validação realizados para a rede counterpropagation

11.4.1 Redes neurais treinadas com os sinais com degradação

Para a rede neural counterpropagation, utilizou-se o programa de geração e tratamento de sinais, obtendo-se arquivos de treinamento com 10, 15, 20, 25 e 30% de degradação, nomeando-os como RN1, RN2, RN3, RN4 e RN5. Estes arquivos continham 50 padrões de treinamento dos dois tipos de sinais. Para que não houvesse a hipótese do vício da rede neural em relação aos dados de treinamento, foram criados outras 10 amostras contendo 1000 conjuntos de entradas cada para cada tipo de degradação. As redes neurais foram criadas segundo os parâmetros:

- número de entradas: 1024
- número de saídas: 1
- número de neurônios na camada intermediária: 2

Para diminuir o tempo de treinamento das redes neurais, utilizou-se a opção de inicializar os pesos da rede a partir da média dos dados do sinal. No programa de geração de arquivos de treinamento existe a opção de criar arquivo de pesos para a rede. Este arquivo contém a média das entradas que serão apresentadas à rede durante o treinamento. Com a geração dos arquivos de treinamento (50 superfícies) e a inicialização dos pesos pela média, foi possível iniciar o treinamento com coeficiente de aprendizado bastante baixo, já que necessita-se apenas de um ajuste mais preciso.

As redes foram treinadas segundo os seguintes parâmetros:

Treinamento Kohonen

- coeficiente de aprendizagem inicial: 0.005
- coeficiente de aprendizagem final: 0.0001
- erro (média da distância euclidiana): 0.001
- número máximo de iterações: 50

Treinamento Grossberg

- coeficiente de aprendizagem inicial: 0.2
- coeficiente de aprendizagem final: 0.001
- erro (média da distância euclidiana): 0.001
- número máximo de iterações: 50

Não foi utilizado o modo supervisionado do treinamento Kohonen.

A tela do programa da rede counterpropagation no treinamento Kohonen encontra-se na Figura 51.

Figura 51 - Treinamento Kohonen da rede counterpropagation

Obteve-se resultados de treinamento como pode ser observado na Figura 52.

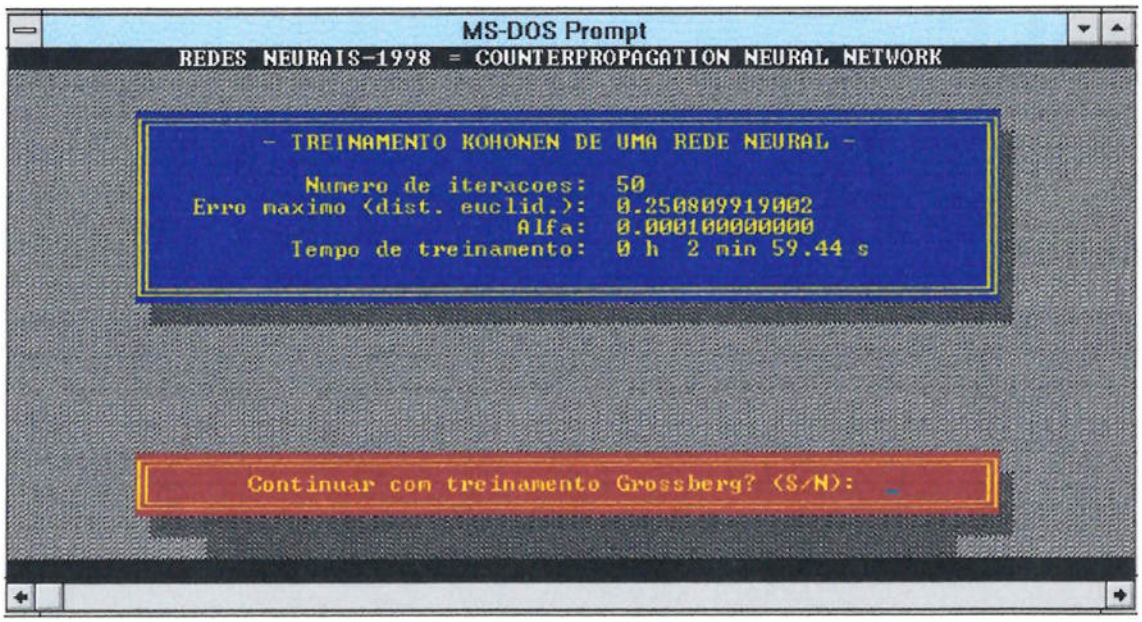


Figura 52 - Resultado do treinamento Kohonen da rede counterpropagation

A média de tempo de treinamento competitivo é de 3 minutos.

A tela do treinamento Grossberg do programa é apresentado na figura

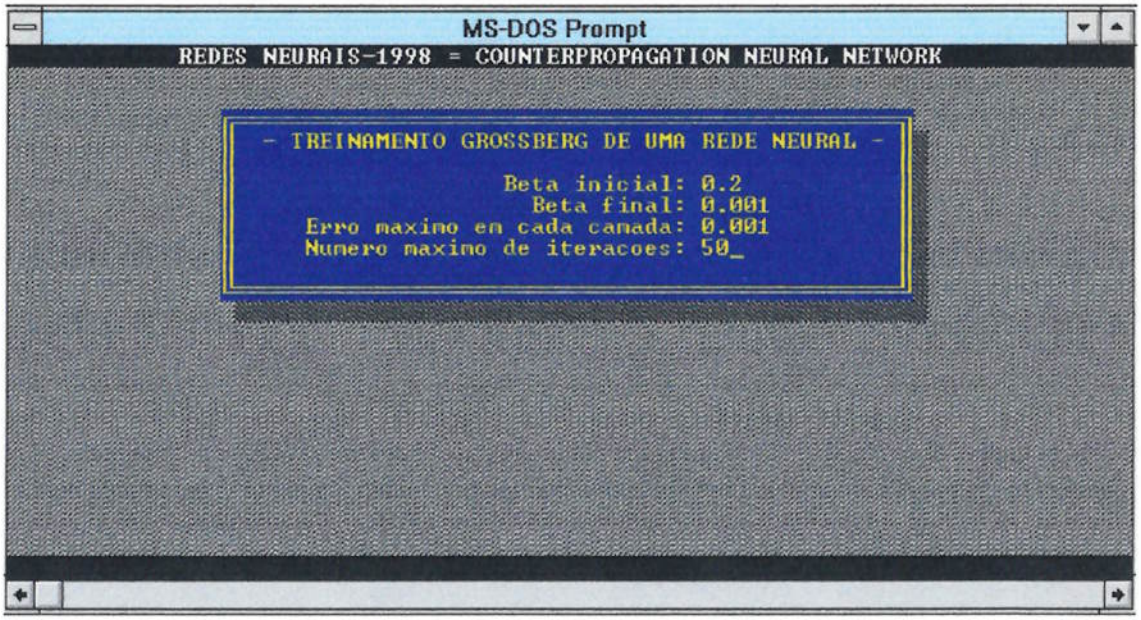


Figura 53 - Treinamento Grossberg da rede counterpropagation

Obteve-se resultados de treinamento como pode ser observado na Figura 54.

Figura 54 - Resultado do treinamento Grossberg da rede counterpropagation

O tempo de treinamento da camada Grossberg é em média 5 segundos.

Foram criadas e treinadas 4 redes neurais diferentes. Estas redes foram nomeadas como RN1, RN2, RN3 e RN4 sendo treinadas com 2%, 15%, 30% e 45% de degradação do sina, respectivamente.

Com os padrões de 1000 conjuntos de dados cada, as redes foram submetidas aos testes de validação. Os resultados foram:

Proporção dos erros no teste de validação da rede counterpropagation para 10 a 50% de degradação

	grau de degradação				
Rede neural	10%	20%	30%	40%	50%
RN1	0.000	0.001	0.001	0.002	0.002
RN2	0.000	0.000	0.001	0.001	0.002
RN3	0.000	0.000	0.000	0.000	0.001
RN4	0.000	0.000	0.000	0.000	0.000

Obs: valores obtidos com a média de 10 amostras de 1000 dados cada.

Observa-se que a rede neural é satisfatória para todos os casos, pois para todos os arquivos de teste a proporção de erros manteve-se abaixo do valor de 0.0212 admissíveis.

Utilizando-se estas mesmas redes, observa-se que estas não captam a dinâmica do sinal quando submetidas à sinais com 3a. e 5a. harmônica.

Proporção dos erros no teste de validação da rede counterpropagation sinais com 3a. e 5a. harmônica e treinadas apenas para degradação simples

	grau de degradação			
Rede neural	15% (3a. harm)	30% (3a. harm)	15% (3a. e 5a. harm)	30% (3a. e 5a. harm)
RN1	0.500	0.500	0.501	0.500
RN2	0.499	0.500	0.499	0.500
RN3	0.498	0.501	0.501	0.500
RN4	0.447	0.500	0.499	0.485

Obs: valores obtidos com a média de 10 amostras de 1000 dados cada.

11.4.2 Sinais com harmônicas

Foram realizados também testes com a terceira e quinta harmônica do sinal. Novas redes similares às anteriormente citadas foram treinadas incluindo-se os dados relativos às harmônicas no arquivo de treinamento. As redes neurais RN6 e RN7 correspondem às redes treinadas com 10% e 30% de degradação do sinal com 3a. harmônica, enquanto que as redes RN8 e RN9 correspondem à 10 % e 30% de degradação do sinal com 3a. e 5a. harmônica Os resultados do teste são:

Proporção dos erros no teste de validação da rede counterpropagation para 3a. e 5a. harmônicas com rede treinada para 3a. e 5a. harmônicas

	grau de degradação			
Rede neural	15% (3a. harm)	30% (3a. harm)	15% (3a. e 5a. harm)	30% (3a. e 5a. harm)
RN6	0.001	0.002	0.499	0.499
RN7	0.000	0.000	0.500	0.499
RN8	0.499	0.501	0.000	0.001
RN9	0.500	0.500	0.000	0.000

Obs: valores obtidos com a média de 10 amostras de 1000 dados cada.

Observa-se que as redes treinadas para reconhecer a 3a. harmônica não reconhecem os sinais com 3a. e 5a. harmônicas e vice-versa. Aplicando os sinais de degradação de um sinal sem harmônicas obtém-se:

Proporção dos erros no teste de validação da rede counterpropagation para teste com 10 a 50% de degradação do sinal sem harmônicas e com redes treinadas para 3a. e 5a. harmônicas

	grau de degradação				
Rede neural	10%	20%	30%	40%	50%
RN1	0.500	0.504	0.501	0.500	0.500
RN2	0.500	0.503	0.502	0.499	0.501
RN3	0.499	0.500	0.500	0.498	0.499
RN4	0.502	0.500	0.501	0.499	0.499

Obs: valores obtidos com a média de 10 amostras de 1000 dados cada.

A partir destes resultados, conclui-se que, para cada freqüência, tem-se 3 padrões distintos de entradas que a rede tenta agrupar em um neurônio só durante o treinamento competitivo. Estes três padrões são:

- sinal sem harmônica,
- sinal com 3a. harmônica e
- sinal com 3a. e 5a. harmônica.

Como foi utilizado uma rede com apenas 2 neurônios na camada competitiva, a rede não teve a capacidade de agrupar os três padrões em um só neurônio vencedor.

11.4.3 Solução do problema das harmônicas

Para resolver o problema do reconhecimento das freqüências quando o sinal possui interferência das harmônicas, foi criada uma nova rede neural com 6 neurônios. As redes criadas foram denominadas RN-A e RNB, e foram treinadas com sinais com, respectivamente, 10% e 30% de degradação de cada tipo (sem harmônicas + com 3a. harmônica + com 3a. e 5a. harmônica).

Os resultados obtidos com arquivos de testes contendo 1000 padrões sem harmônicas, 1000 com 3a. harmônica e 1000 com 3a. e 5a. harmônica são apresentados a seguir:

Proporção dos erros no teste de validação da rede counterpropagation para redes neurais com 6 neurônios na camada competitiva

	Rede neural	
Tipo de arquivo teste	RN-A	RN-B
10% de degradação com e sem harmônicas	0.000	0.000
20% de degradação com e sem harmônicas	0.001	0.001
30% de degradação com e sem harmônicas	0.001	0.000
40% de degradação com e sem harmônicas	0.001	0.001

Obs: valores obtidos com a média de 10 amostras de 3000 dados cada.

Portanto, constata-se que a solução de adicionar neurônios na camada competitiva faz com que a rede ficasse satisfatória, já que a proporção de erro é bastante pequena.

11.4.4 Perda dos pontos iniciais do sinal

O tratamento de sinais transformando-os em superfícies faz com que a rede não sofra um efeito muito grande. Isto é devido à compressão que é feita quando se aplica a transformação que utiliza a progressão geométrica de razão 0,5. Esta situação foi testada e os resultados obtidos foram também bastante satisfatórios.

11.4.5 Discussão sobre os resultados obtidos pela rede counterpropagation

De acordo com os resultados obtidos anteriormente, foi possível constatar que a rede counterpropagation possui um alto poder de classificação para um tempo de treinamento pequeno. Isto é devido às técnicas utilizadas para melhorar a eficiência do treinamento, que são: inicialização dos pesos pela média e conceito adaptado de "consciência".

Em uma rede neural com poucos neurônios na camada competitiva pode ter o seu poder de reconhecimento de padrões comprometido, principalmente se as entradas aplicadas na rede serem muito próximas. Isto foi claramente observado nos resultados obtidos com a inserção de mais neurônios para se poder classificar as harmônicas.

Conclui-se que o pré-processamento de sinais foi aplicado com sucesso nas entradas da rede, já que a rede neural capturou a dinâmica do sinal e reconheceu as freqüências com sucesso.

Deve-se ressaltar que a não aplicação da rede neural com o pré-processamento do sina em superfícies é devido ao fato do treinamento backpropagation ser centenas de vezes mais lento do que o counterpropagation. Nos resultados, nota-se que o tempo de treinamento para o backpropagation é menor, porém os dados de entrada possuem dimensões diferentes, o que influi na velocidade de processamento dos pesos.

12. DISCUSSÃO SOBRE AS DIFICULDADES DO PROJETO

De acordo com a metodologia adotada para este projeto, serão citadas a seguir as dificuldades que foram encontradas no decorrer do projeto. As suas possíveis soluções serão discutidas e será feita uma análise crítica sobre a decisão tomada.

Inicialmente, constatou-se que a literatura sobre redes neurais é bastante restrita, sendo que os livros mencionam sempre o aspecto teórico do problema. Na prática, aparecem dificuldades que não podem ser encontradas na literatura, ou então, são muito superficialmente abordadas. Isto é devido ao fato de que as redes neurais artificiais são, de certa forma, conceitos recentes.

Apesar das várias arquiteturas de redes neurais existentes, a avaliação da melhor arquitetura depende da implementação da rede no problema proposto. Diversos autores tentam classificar as redes mais adequadas para cada tipo de problema, ou então, generalizar uma arquitetura para vários tipos de problemas.

Foram utilizadas duas arquiteturas muito conhecidas: a backpropagation e a counterpropagation. Inicialmente, estas redes neurais foram implementadas na sua forma mais simples, ou seja, sem que fossem verificados critérios como convergência, robustez ou velocidade de treinamento. A partir destas implementações notou-se que a rede backpropagation tem aplicação mais genérica, mas o seu tempo de treinamento é muito maior do que o da rede counterpropagation. Isto foi observado na literatura e no próprio fluxograma de treinamento da rede (onde todos os pesos da rede backpropagation devem necessariamente ser processados).

Ao pesquisar-se sobre a rede counterpropagation, encontrou-se diversas formas de se representar esta rede. Por exemplo, alguns autores consideram o neurônio vencedor aquele que tiver o maior produto escalar entre os seus pesos e o vetor de entrada. Outros consideram o vencedor aquele que tiver a menor distância euclidiana entre o vetor d pesos e o de entrada. Ambas as abordagens estão corretas quando os dados são normalizados.

Outro conceito pouco abordado na literatura é o treinamento da camada competitiva. Como foi explicado durante o trabalho, podem existir neurônios que nunca serão vencedores porque o seu vetor de pesos está contido em uma parte isolada da hiper-esfera, longe dos vetores de entrada. A literatura cita dois métodos de solução do problema: o método da combinação convexa, que não costuma ser aplicada porque aumenta bastante o tempo de treinamento; e o estabelecimento de uma consciência, que faz com a probabilidade de que um neurônio seja vencedor seja igual para todos. Este estabelecimento da consciência depende de constantes que são obtidos pela experiência da pessoa que irá implementar a rede. Devido a este fato, optou-se por uma solução alternativa, já que se ocorresse algum erro, muito tempo seria gasto para se descobrir se o erro era proveniente do programa ou da escolha de constantes.

A solução alternativa para o problema do treinamento consistiu em utilizar o conceito de "consciência", fazendo-se a análise das freqüências com que os neurônios são vencedores. A partir daí, se algum neurônio não era vencedor após a aplicação de todo o conjunto de entradas, isto significaria que os seus pesos estariam numa região isolada da hiper-esfera. Então, a solução adotada foi a de igualar o vetor de pesos do neurônio isolado com o que foi mais vezes vencedor, ou seja, o vetor de pesos estaria na região

em que os vetores de entrada são mais freqüentes. A classificação, a partir daí, seria mais refinada, com a utilização de todos os neurônios.

Outro ponto importante é relativo ao número de neurônios necessários para a camada competitiva. Como foi visto nos resultados, o número de neurônios pode influir de maneira decisiva no resultado. Entretanto, as entradas desta aplicação são bastante próximas e com dimensões bastante grandes, não favorecendo este tipo de análise.

A literatura sobre a validação e verificação da rede neural implementada é bastante escassa. Alguns autores citaram sobre o tratamento estatístico dos resultados, mostrando alguns exemplos bem vagos. Os testes de validação aplicados neste relatórios foram baseados nas citações destes autores e em livros de estatística básica.

Durante a implementação do pré-processamento do sinal, notou-se a peculiaridade ocorrida com a superfície resultante da aplicação do sinal com 3a. e 5a. harmônicas. Este fato poderá ser melhor explorado futuramente para várias outras aplicações.

Para concluir esta discussão, deve-se ressaltar que o tema "redes neurais artificiais" está em um estágio de amadurecimento ainda. Muitos tópicos ainda deverão ser estudados para que se possa generalizar as aplicações das redes neurais, assim como analisar a sua eficiência frente a outros sistemas.

13. RESULTADOS DO PROJETO

Este projeto resultou em:

- um estudo detalhado sobre duas arquiteturas de redes neurais
- implementação de uma rede neural backpropagation em C++ cuja aplicação pode ser estendida para problemas genéricos
- implementação de uma rede neural counterpropagation em C++ cuja aplicação pode ser estendida para problemas genéricos
- avaliação da validade das redes implementadas
- implementação de um pré-processador de sinais que geram arquivos de treinamento e teste
- documentação detalhada dos programas de redes neurais implementados

14. PROPOSTAS PARA DESENVOLVIMENTO FUTURO

Seguem-se abaixo alguns tópicos que pode ser desenvolvidos no futuro.

- melhoria do algoritmo relativo à consciência de forma que ele possa ter aplicação mais abrangente;
- reconhecimento da componente fundamental a partir de suas harmônicas;
- reconhecimento das componentes harmônicas de uma sinal para, por exemplo, identificar distúrbios em sistemas elétricos e acionar os compensadores adequados.

15. CONCLUSÃO E CONSIDERAÇÕES FINAIS

No reconhecimento de sinais acústicos para monitoramento de condições de um poço de extração petrolífera, podem ser aplicadas redes neurais artificiais de maneira bastante satisfatória. Neste trabalho, foi possível observar que, com um pré-processamento de sinais, as redes neurais conseguem aliar o poder de captura da dinâmica de uma sinal ao poder de classificação e reconhecimento de padrões. Este resultado foi atingido de maneira simples, com a geração de dois programas em C++ devidamente documentados. A solução de se utilizar redes neurais pode ser implementada a um custo muito mais baixo em relação ao custo operacional que seria gasto caso fosse utilizado fios para a comunicação entre o sensor e o receptor.

Para concluir este trabalho, deseja-se que as aplicações das redes neurais artificiais sejam estendidas às outras áreas da engenharia e que a nova geração de estudantes percebam, desde já, o poder que esta ferramenta tem para a solução de problemas, ferramenta esta criada pelo Homem.

16. REFERÊNCIAS BIBLIOGRÁFICAS

HAYKIN, Simon S.; **Neural Networks: a Comprehensive Foundation**. New York, Macmillan, 1994.

HERTZ, John; **Introduction to the theory of neural computation**, by J. Hertz, A. Krogh and R.G. Palmer. Addison-Wesley, 1991.

WASSERMAN, Philip D.; **Neural Computing: theory and practice**. New York, Van Nostrand Reinhold, 1989.

CICHOCKI, Andrzej; **Neural Networks for optimization and signal processing**. John Wiley & Sons, 1993.

WASSERMAN, Philip D.; **Advanced methods in neural computing**. New York, Van Nostrand Reinhold, 1989.

PAO, Yoh-Han; **Adaptative pattern recognition and neural networks**. Addison-Wesley, 1989.

FU, Limin; **Neural Networks in computer intelligence**. Florida, MacGraw-Hill, 1994.

PATTERSON, Dan W.; **Artificial Neural Networks**. Pingapore, Prentice-Hall, 1996.

COSTA NETO, Pedro L. de O.; **Estatística**. São Paulo, Edgard Blücher, 1977.

A. APÊNDICE 1 - A REDE BACKPROPAGATION EM C++

A.1 Detalhamento dos objetos

Nesta seção, serão detalhados todos os objetos da rede backpropagation explicando-se a finalidade de todas as variáveis e funções que os compõem.

- *vector:* este objeto representa um vetor de números reais de tal forma que o tamanho deste vetor está em seu campo *size*. Existem funções que o inicializam e o destroem (*inicializar e destruir*) sem afetar outros objetos. A alocação de memória e a desalocação é feita de maneira dinâmica utilizando-se ponteiros. Pode-se atribuir um valor para os elementos na inicialização do vetor.
- *matrix:* este objeto representa uma matriz de números reais de tal forma que o tamanho desta matriz está em seus campos *lin* e *col* (número de linhas e colunas respectivamente). Existem funções que o inicializam e o destroem (*inicializar e destruir*) sem afetar outros objetos. A alocação de memória e a desalocação é feita de maneira dinâmica utilizando-se ponteiros. Pode-se atribuir um valor para os elementos na inicialização da matriz.
- *screen:* o objeto contém as seguintes funções de tela:
  - *limpa tela:* como o próprio nome diz, esta função limpa a tela atual.
  - *quadro:* desenha um quadro dados as coordenadas dos vértices de sua diagonal e as cores de texto e de fundo.
  - *centro:* centraliza um texto na tela dado a linha e as cores de texto e de fundo.
  - *escrever:* escreve um texto na tela dado as coordenadas do ponto de partida e as cores de texto e de fundo.
  - *tela_fundo:* cria a tela de fundo deste programa.
  - *tela_principal:* cria a tela principal deste programa.
  - *opção:* cria o quadro de opções para a tela principal deste programa.

- *tela_sobregravar:* cria o quadro de aviso quando existe a possibilidade da rede neural ser sobregravada.
- *impossível_abrir:* cria o quadro de aviso quando for impossível abrir o arquivo especificado.
- *erro_leitura:* cria o quadro de aviso quando há um erro na leitura de um arquivo.
- *não_definida:* cria um quadro de aviso quando a rede neural não foi inicializada anteriormente.

- *tela_define:* além das funções de *screen* possui funções próprias para a definição de uma nova rede. Essas funções são as seguintes:
  - *principal:* cria a tela principal para esta parte do programa.
  - *entradas:* cria o quadro pedindo o número de entradas da rede neural.
  - *saídas:* cria o quadro pedindo o número de saídas da rede neural.
  - *neurônios:* cria o quadro pedindo o número de neurônios da rede neural.
  - *aviso:* cria um aviso informando que a rede está definida.

- *tela_abrir:* além das funções de *screen* possui funções próprias para a abertura de uma nova rede. Essas funções são as seguintes:
  - *principal:* cria a tela principal para esta parte do programa.
  - *pede_arquivo:* cria o quadro pedindo o arquivo que contém a rede neural.
  - *aviso:* cria um aviso informando que a rede está definida.

- *tela_treino:* além das funções de *screen* possui funções próprias para o treinamento de uma nova rede. Essas funções são as seguintes:
  - *principal:* cria a tela principal para esta parte do programa.
  - *pede_dados:* cria o quadro pedindo o arquivo de dados para o treinamento.
  - *pede_parâmetros:* cria o quadro pedindo os parâmetros do treinamento (coeficiente de aprendizado, de momento, número de iterações, erro, etc.).

- *em_treinamento:* cria o quadro informando que a rede neural está em treinamento.
- *resultado_parcial:* cria o quadro informando os resultados parciais ao longo do treinamento.
- *resultado_final:* cria o quadro informando os resultados finais do treinamento.
- *barra_erro:* cria uma barra de erro de acordo com o erro dado em cada etapa do treinamento. O erro é indicado pela parte central e a mudança de escala é percebida quando há a mudança de cores nas partes laterais desta barra.
- *aviso:* cria um aviso informando que a rede está treinada.

- *tela_execução:* além das funções de *screen* possui uma função própria para a execução (simulação) da rede neural. A função cria a tela principal para esta parte do programa.
- *tela_teste:* além das funções de *screen* possui funções próprias para o teste da rede neural. Essas funções são as seguintes:
  - *principal:* cria a tela principal para esta parte do programa.
  - *pede_arquivo:* cria o quadro pedindo o arquivo de dados para o teste da rede neural.
  - *resultado:* cria um quadro de resultados ao final do teste.

- *impressão:* além das funções de *screen* possui funções próprias para a impressão da rede em um arquivo. Essas funções são as seguintes:
  - *confirma:* cria o quadro onde se pede a confirmação para a impressão.
  - *pesos:* cria o quadro pedindo o nome do arquivo onde a rede neural será guardada.

- *treino:* objeto que contém todas as funções necessárias para o treinamento da rede neural. Além disso, possui variáveis que auxiliam esta parte do programa. A matriz *dados* contém os padrões de treinamento; as matrizes *delta1* e *delta2* são as correções para os pesos, *e1* e *e2* são os vetores das ativações de entrada; e *ent*

e *sai* são os vetores que contém as entradas e saídas respectivamente. As funções deste objeto são:

- *leitura_parâmetros:* faz a leitura dos parâmetros do treinamento e os passa por referência. Os parâmetros são: o coeficiente de aprendizado, o de momento, o número de iterações para escalonamento, os fatores para a correção dos coeficientes, o erro quadrático admissível e o número de iterações máximo.
- *ativ:* função ativação que, dado um número, retorna o resultado obtido com a função sigmoidal.
- *dativ:* derivada da função de ativação.
- *imprime_resultado_parcial:* como o próprio nome diz, faz a impressão dos resultados parciais durante o treinamento.
- *imprime_resultado_final:* faz a impressão dos resultados definitivos após o treinamento.
- *imprime_arquivo:* faz a impressão da rede neural para um arquivo.
- *seleção_dados:* faz a seleção e quais dados do arquivo do EMTP serão utilizados no treinamento da rede neural.
- *normalizar_dados:* faz a normalização dos dados coletados, sendo que essa normalização é feita por coluna, isto é, cada variável é normalizada pelo seu valor máximo.

- *teste:* objeto que contém todas as funções necessárias para o teste da rede neural. Além disso, possui variáveis que auxiliam esta parte do programa. A matriz *dados* contém os padrões de teste (mas utiliza apenas 30% deles); *e1* e *e2* são os vetores das ativações de entrada; e *ent* e *sai* são os vetores que contém as entradas e saídas respectivamente. Para o teste, a única unção necessária é a *imprime* que faz a impressão do resultado do teste realizado.
- *net:* objeto contém os pesos, dados pelas matrizes *w1* e *w2*, e os sinais sinápticos, dados pelos vetores *s1*, *s2* e *s3*. Possui um *flag* para indicar se a rede neural já foi ou não inicializada e um número inteiro *tipo* para indicar que tipo de treinamento foi realizado (a pertir do EMTP ou não). Além disso, possui um vetor de números

reais *fator_normal* que contém os fatores normalizantes obtidos no treinamento ou no teste. As funções deste objeto são:

- *pega_tempo:* pega o tempo e retorna os centésimos de segundo para a geração dos randômicos.
- *espera_tecla:* cria o quadro para esperar uma tecla ser pressionada.
- *verifica_inicialização:* verifica se a rede já foi ou não inicializada.
- *nova_rede:* cria uma nova rede.
- *abrir_rede:* abre uma rede existente em arquivo.
- *leitura_dados:* realiza a leitura de dados tanto para o treinamento como para o teste da rede neural.
- *acha_erro:* dado o vetor de saída desejada, retorna o erro quadrático gerado a partir da saída obtida.
- *treinar_rede:* realiza o treinamento de uma rede neural já inicializada.
- *testar_rede:* realiza o teste de uma rede neural já inicializada.
- *saídas:* dadas as entradas, acha as saídas através da rede neural.
- *destruir:* destrói a rede neural desalocando da memória os vetores *s1*, *s2* e *s3* e as matrizes *w1* e *w2*.

Deve-se ressaltar que os objetos de tela são chamados ao longo de todo o programa e também por outros objetos.

A.2 Diagramas de Nassi-Schneidermann

Agora que os objetos foram definidos e detalhados, haverá uma breve apresentação do algoritmo implementado através dos diagramas de Nassi-Schneidermann. As funções de telas não irão ser apresentadas por não serem relevantes ao programa. Apenas a rotina que cria a barra de erro deverá ser esquematizada a seguir.

Objeto: *vector* - subrotina *inicializar* parâmetros: *val* (valor dos elementos sendo nulo caso não seja referido) variáveis auxiliares: *i* (contador)
aloca vetor *v* do tamanho *size*
para *i* de zero a *size*
v[i] = val

Objeto: *vetor* - subrotina *destruir* parâmetros: nenhum variáveis auxiliares: nenhuma
desaloca *v*
aterra o ponteiro *v* (*v* = NULL)

Objeto: *matrix*- subrotina *inicializar* parâmetros: *val* (valor dos elementos sendo nulo caso não seja referido) variáveis auxiliares: *i* e *j* (contadores)
aloca ponteiro para o vetor *m* do tamanho *lin*
para *i* de zero a *lin*
para cada elemento do vetor anterior, alocar outro vetor de tamanho *col*
para *j* de zero a *col*
m[i][j] = val

Objeto: *screen* - subrotina *barra_erro*

parâmetros: *erro* (número real)

variáveis auxiliares: *i*, *a*, *b* e *tipo* (inteiros); *nível_max* (real) e *txt* (vetor de caracteres)

para *i* de zero 50

guarda caracter 178 no vetor de texto (cor da parte central da barra)

tipo recebe o caracter 177

o erro é multiplicado por 100 (conversão para erro %)

se erro for maior que 50

T

F

a recebe 1 e *b* recebe 49 (extremos do erro)

enquanto erro for menor do que 1 (1%)

multiplica erro por 10 (para mudar a escala)

divide nível por 10 (para indicar mudança)

troca-se o tipo de caracter de *tipo* de 176 para 177 ou vice-versa)

a recebe 25 - erro/2 (extremo esquerdo)

b recebe 25 + erro/2 (extremo direito)

para i de zero até *a*

txt[i] recebe tipo (parte lateral esquerda)

para i de 50 até *b* (decrementando)

txt[i] recebe tipo (parte lateral direita)

imprime *txt* na tela

imprime escala utilizada com *nível_max*

Objeto: net - subrotina *verifica_inicialização*

parâmetros: nenhum (utiliza o campo inicializado deste objeto)

variáveis auxiliares: *c* (caracter)

se *inicializado* for igual a *sim*

T

F

dá mensagem de sobregravar

lê *c*

se *c* for *'N'* ou *'n'*

T

F

ir para *end*

se *c* for *'s'* ou *'S'*

T

destrói *w1*, *w2*, *s1*, *s2* e *s3*

retorna 1

enquanto n for diferente de 's', 'S', 'n'ou 'N'

retorna 0

end: retorna 0

Objeto: *net* - subrotina *nova_rede*

parâmetros: nenhum (utiliza os campos de *net*)
variáveis auxiliares: *i* e *j* (contadores) e objeto *tela* (do tipo tela_define)

chama *principal* de *tela*

T se *verifica_inicialização* retornar 0

ir para *end*

chama as telas de entradas, saídas e neurônios para a leitura dos números de entradas, de saídas e de neurônios

chama *pega_tempo* para a semente para a geração de randômicos

inicializa *w1* dado seus campos: *lin=smax e col=nmax+1*

para *i* de 0 a *smax*

- para *j* de 0 a *nmax*
  - *w1[i][j]* recebe um randômico inteiro até no máximo 1000
  - divide *w1[i][j]* por 1000 e soma 0.2

chama *pega_tempo* para a semente para a geração de randômicos

inicializa *w2* dado seus campos: *lin=nmax e col=emax+1*

para *i* de 0 a *nmax*

- para *j* de 0 a *emax*
  - *w2[i][j]* recebe um randômico inteiro até no máximo 1000
  - divide *w2[i][j]* por 1000 e soma 0.1

inicializado = sim

inicializar *s1* dado seu campo: *size=smax+1* atribuindo valor diferente de zero (0.7)

inicializar *s2* dado seu campo: *size=nmax+1* atribuindo valor diferente de zero (0.4)

inicializar *s3* dado seu campo: *size=emax+1* atribuindo valor diferente de zero (0.6)

end:

Objeto: net - subrotina *acha_erro*

parâmetros: *sai* (objeto *vector*)
variáveis auxiliares: *i* (contador); *erro* (número real)

zera *erro*

para *i* de zero a *smax*

- *erro* é acrescido da metade de (*sai[i]-s1[i]*) ao quadrado

retorna *erro*

Objeto: net - subrotina *abrir_rede*

parâmetros: nenhum (utiliza os campos de *net*)

variáveis auxiliares: *i* e *j* (contadores); *c* (caracter); *s*, *nome1*, *nome2*, *nome3* e *nomearq* (string); *arquivo1*, *arquivo2*, *arquivo3* (arquivos) e objeto *tela* (tela_abrir)

chamar tela principal

se *verifica_inicialização* retornar 0 ir para *end*

alocar memória para *nomearq*, *nome1*, *nome2*, *nome3* e *s* (de tamanho 20)

imprime tela pedindo arquivo e lê o nome do arquivo em *nomearq*

copia *nomearq* para *nome1*, *nome2* e *nome3* e adiciona as extensões *.w1*, *.w2* e *.sz* à *nome1*, *nome2* e *nome3* respectivamente

T se *arquivo1*, *arquivo2* e *arquivo3* retornarem NULL ao tentar abrí-los

apresentar mensagem de *impossível abrir*, fazer com que *inicializado* receba *não* e ir para *erro*

enquanto não atingir o fim de *arquivo3*

T se ao ler *emax*, *smax* e *nmax* retornar valor NULL

imprimir mensagem de *erro de leitura* e ir para *erro*

fechar *arquivo3*

inicializar *w1* dado seus campos: *lin=smax* e *col=nmax+1*

enquanto não atingir o fim de *arquivo1*

para *i* de zero a *smax*

para *j* de zero a *nmax*

T se ao ler *w1[i][j]* retornar valor NULL

imprimir mensagem de *erro de leitura* e ir para *erro*

fechar *arquivo1*

inicializar *w2* dado seus campos: *lin=nmax* e *col=emax+1*

enquanto não atingir o fim de *arquivo2*

para *i* de zero a *nmax*

para *j* de zero a *emax*

T se ao ler *w2[i][j]* retornar valor NULL

imprimir mensagem de *erro de leitura* e ir para *erro*

inicializar *s1*, *s2* e *s3* dados que *size* são *smax+1*, *nmax+1* e *emax+1* respectivamente

inicializado recebe *sim*

desaloca e aterra *nomearq*, *nome1*, *nome2*, *nome3* e *s* e imprime aviso de OK

erro: fecha arquivo1, arquivo2 e arquivo3; desaloca e aterra *nomearq*, *nome1*, *nome2*, *nome3* e *s*; e *espera_tecla*

end:

Objeto: *treino* - subrotina *imprime_resultado_parcial* parâmetros: *iter*, *erro_max*, *beta_ini*, *lamb_ini* variáveis auxiliares: *conv* (ponteiro para caracteres); scr (objeto *screen*)
inicializa *conv* com tamanho 20
grava *iter* em *conv*
chama *escrever* (de *screen*) com *conv* para uma certa posição e cores
grava *erro_max* em *conv*
chama *escrever* (de *screen*) com *conv* para uma certa posição e cores
grava *beta_ini* em *conv*
chama *escrever* (de *screen*) com *conv* para uma certa posição e cores
grava *lamb_ini* em *conv*
chama *escrever* (de *screen*) com *conv* para uma certa posição e cores
desaloca *conv*

Objeto: *treino* - subrotina *leitrua_parâmetros* parâmetros: *beta_ini*, *lamb_ini*, *n_escf*, *beta_pos*, *lamb_pos*, *errod*, *n_max* variáveis auxiliares: *tela* (tela_treino)
vai para a posição certa e lê *beta_ini*
vai para a posição certa e lê *lamb_ini*
vai para a posição certa e lê *n_escf*
enquanto *n_escf* < 0
vai para a posição certa e lê *beta_pos*
vai para a posição certa e lê *lamb_pos*
vai para a posição certa e lê *errod*
enquanto *errod* < 0
vai para a posição certa e lê *n_max*
enquanto *n_max* < 0

Objeto: net - subrotina *leitura_dados*

parâmetros: *caso* (teste ou treino); *dados* (matrix)
variáveis auxiliares: *arqdad* (string); *i*, *j* e n_dados(inteiros); *fdados* (arquivo); *tela1* (tela_teste); e *tela2* (tela_treino)

se *caso* = *teste*

T: imprime tela pedindo arquivo para teste

F: imprime tela pedindo arquivo para treino

inicializa *arqdad* com tamanho 20 e lê *arqdad*

adiciona extensão *.trn* à arqdad

se ao tentar abrir *fdados* com *arqdad* retornar valor NULL

T: imprimir mensagem de *impossível_abrir*

F:

- se a leitura de *n_dados* retornar NULL
  - T: imprimir *erro_leitura* de *tela2* e ir para *erro*
- inicializar *dados* com os campos *lin*=*n_dados* e *col*=*(emax+smax)*
- enquanto não atingir o fim de *fdados*
  - para *i* de 0 a *n_dados*
    - para j de 0 a *(emax+smax)*
      - se ao ler *dados[i][j]* retornar NULL, imprimir mensagem *erro_leitura* e ir para *erro*
- fechar arquivo *fdados*
- desalocar *arqdad* e retornar 1

espera_tecla e ir para *erro*

erro: fechar arquivo *fdados*; desalocar *arqdad*; e retornar 0

Objeto: *treino* - subrotina *imprime_resultado_final*

parâmetros: *iter* e *erro_max*
variáveis auxiliares: *conv* (ponteiro para caracteres); *scr* (screen); *tela* (tela_treino)

inicializa *conv* com tamanho 20

imprime tela com *resultado_final* (de *tela*)

grava *iter* em *conv*

chama *escrever* (de *screen*) com *conv* para uma certa posição e cores

grava *erro_max* em *conv*

chama *escrever* (de *screen*) com *conv* para uma certa posição e cores

desaloca *conv*

Objeto: *treino* - subrotina *ativ*

parâmetros: x (número real)

variáveis auxiliares: nenhuma

se x for maior que 50

T: retorna 1

F: se x for menor que -50

T: retorna 0

F: retorna $1/(1+\exp(-x))$ (sigmoidal)

Objeto: *treino* - subrotina *dativ*

parâmetros: x (número real)

variáveis auxiliares: nenhuma

retorna $x*(1-x)$

Objeto: *net* - subrotina *treinar_rede*

parâmetros: *caso* (inteiro)

variáveis auxiliares: *aprf*=1000; *erro*, *errod*, *erro_max*, *max*, *beta_ini*, *lamb_ini*, *beta_pos*, *lamb_pos (reais)*; *i*, *j*, *n_esc*, *n_escf*, *apr*, *qdad*, *n_max*, *iter* (inteiros);*temp* (matrix); *tela* (tela_treino); *trn* (treino); *key* (char)

se *caso* for *0*

T

imprime tela pincipal com *principal* de *tela*

se *inicializado* for *não*

T

imprimir mensagem de rede não inicializada; esperar uma tecla e ir para *end*

se ao chamar *leitura_dados* para o treino retornar zero

T

ir para *end*

F

imprime tela pincipal para o EMTP com *principal_EMTP* de *tela*

tipo recebe 1

se *inicializado* for *não*

T

imprimir mensagem de rede não inicializada; esperar uma tecla e ir para *end*

se ao chamar *dados_EMTP* retornar *0*

T

ir para *end*

executar *seleção_dados* de *trn* dados *temp*, *variáveis* (de *trn)* e *padrões* (de *trn*)

executar *normalizar_dados* de *trn* dados *dados* (de *trn*) e *fator_normal*

destruir *temp*

inicializar *delta1*, *delta2*, *e1*, *e2*, *ent* e *sai* dados os seus respectivos tamanhos

chamar *leitura_parâmetros* do objeto *trn* passando-se os parâmetros necessários

n_esc recebe *n_escf*

imprime tela para resultado parcial (*resultado_parcial* de *tela*)

imprime mensagem *em_treinamento* (de *tela*)

apr recebe *aprf*

chama *pega_tempo* para a semente para a geração de randômicos

qdad recebe um valor randômico menor do que *lin* (campo de *dados* de *trn*)

incrementa *iter* e decrementa *n_esc* e *apr*

se *n_esc* for zero

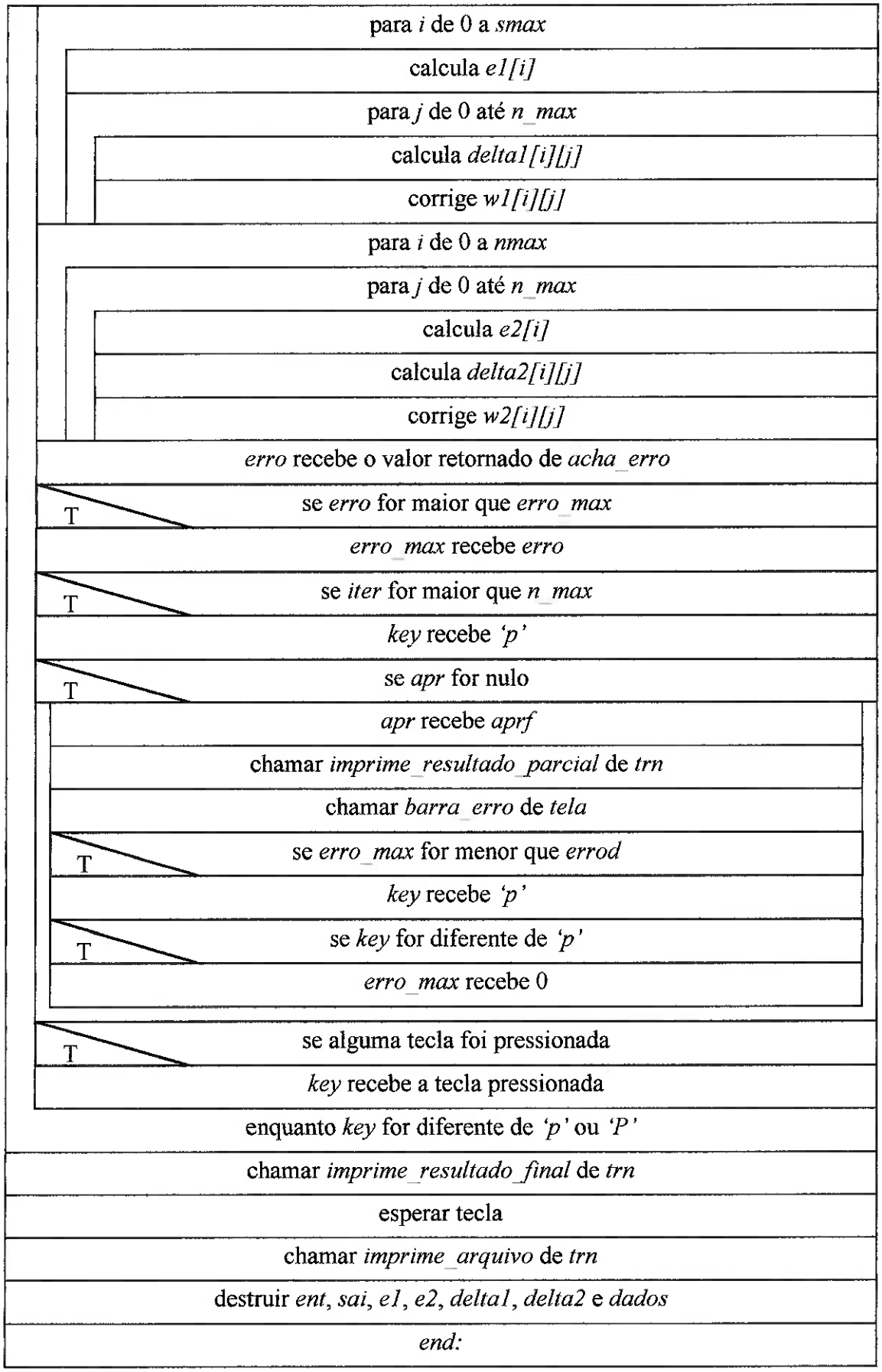
T

n_esc recebe *n_escf*; corrige *beta_ini* e *lamb_ini* com *beta_pos* e *lamb_pos*

pegar um conjunto de dados qualquer (dado por *qdad*) e atribuir à *ent* e *sai* os seus valores

chamar a rotina *saídas* dado *ent*

continua na póxima página


para i de 0 a smax
calcula e1[i]
para j de 0 até n_max
calcula delta1[i][j]
corrige w1[i][j]
para i de 0 a nmax
para j de 0 até n_max
calcula e2[i]
calcula delta2[i][j]
corrige w2[i][j]
erro recebe o valor retornado de acha_erro
T
se erro for maior que erro_max
erro_max recebe erro
T
se iter for maior que n_max
key recebe 'p'
T
se apr for nulo
apr recebe aprf
chamar imprime_resultado_parcial de trn
chamar barra_erro de tela
T
se erro_max for menor que errod
key recebe 'p'
T
se key for diferente de 'p'
erro_max recebe 0
T
se alguma tecla foi pressionada
key recebe a tecla pressionada
enquanto key for diferente de 'p' ou 'P'
chamar imprime_resultado_final de trn
esperar tecla
chamar imprime_arquivo de trn
destruir ent, sai, e1, e2, delta1, delta2 e dados
end:

Objeto: *net* - subrotina *testar_rede* parâmetros: nenhum (utiliza campos de *net*) variáveis auxiliares: *erro*, *erro_max* (reais); *i*, *qdad*, *iter* (inteiros); *tela* (tela_teste); *test* (teste)
imprime tela principal (*principal* de *tela*)
T se *inicializado* for *não*
imprimir mensagem de rede não definida; esperar tecla e ir para *end*
T se ao chamar *leitura_dados* para o caso do teste retornar NULL
ir para *end*
inicializar *ent* e *sai*
chama *pega_tempo* para a semente para a geração de randômicos
enquanto *iter* for menor do que 0,3**dados.lin* (utiliza apenas 30% dos dados)
qdad recebe um número aleatório menor que o campo *lin* de *dados*
incrementa *iter*
ent recebe os valores de entrada para o conjunto de dados referido por *qdad*
sai recebe os valores de saída para o conjunto de dados referido por *qdad*
chama *saídas* dado *ent*
erro recebe *acha_erro* dado *sai*
T se *erro* for maior que *erro_max*
erro_max = *erro*
chama *imprime* de *test* para a impressão dos resultados
destrói *dados*, *ent* e *sai*
espera_tecla

Objeto *test* - subrotina *imprime* parâmetros: *iter* (inteiro), *erro_max* (real) variáveis auxiliares: *conv* (string), *scr* (screen) e *tela* (tela_teste)
alocar *conv* com tamanho 20
imprimir tela para resultados
guardar *iter* em *conv*
chamar *escrever* de *scr* para imprimir *conv* na posição correta
guardar *erro* em *conv*
chamar *escrever* de *scr* para imprimir *conv* na posição correta
deslocar *conv*

Objeto: *net* - subrotina *saídas*
parâmetros: *ent* (vector) e os campos de *net* variáveis auxiliares: *i*, *j* (contadores) e *trn* (treino)
zerar todos os elementos de *s1*
zerar todos os elementos de *s2*
igualar *s3* ao vetor *ent*
fazer *s3[emax]* receber -1
para *i* de 0 até *nmax*
para *j* de 0 até *emax*
calcular *s2* a partir de *s3* e de *w2* (usando também a função de ativação)
s2[nmax] recebe 1
para *i* de 0 até *smax*
para *j* de 0 até *nmax*
calcular *s1* a partir de *s2* e de *w1* (usando também a função de ativação)

Objeto: *net* - subrotina *destruir*
parâmetros: nenhum (utiliza campos do objeto *net*) variáveis auxiliares: nenhum
desaloca da memória as variáveis: *w1*, *w2*, *s1*, *s2* e *s3*

Objeto: *net* - subrotina *executar_teclado* parâmetros: nenhum (utiliza os campos de *net*) variáveis auxiliares: *i* (inteiro), *conv* (string), *tela* (tela_execução), *scr* (screen), *ent* (vector)
imprime a tela principal desta parte do programa
T se *inicializado* for *não*
imprimir mensagem de rede não definida; esperar tecla e ir para *end*
alocar *conv* com tamanho 40
inicializar *ent*
para *i* de zero até *emax*
ler *ent[i]*
chamar *saídas* dado *ent*
para *i* de zero até *smax*
imprimir *s1[i]* na posição correta
espera_tecla; destruir *ent*; desalocar *conv*
end:

PROGRAMA PRINCIPAL

variáveis: *scr* (screen); *i*, *opção*(inteiros); *c* (caracter); *pára* (flag); *rede* (net)

o campo *inicializado* de *rede* recebe *não*

de_novo:

chamar *tela_principal* de *scr*

ler *opção* na posição correta

pára recebe *não*

se *opção* for menor do que 1 ou maior do que 6

T

ir para *de_novo*

se *opção* for menor do que 6

T

F

caso *opção* tenha valor:

1	chamar *nova_rede* de *rede*
2	chamar *abrir_rede* de *rede*
3	chamar *treinar_rede* de *rede*
4	chamar *testar_rede* de *rede*
5	chamar *executar_teclado* de *rede*

pára recebe *sim*

enquanto *pára* for igual a *não*

se o campo *inicializado* de *rede* for *sim*

T

chamar *destruir* de *rede*

chamar *limpa_tela*

A.3 Listagem do Programa

```
//=========================================//
// REDES NEURAIS - BACKPROPAGATION   //
//=========================================//
// Trabalho de Formatura - 1998      //
// CHANG CHIH WEI - No. USP 185014 //
// Reconhecimento de sinais          //
//=========================================//


#include <stdio.h>
#include <math.h>
#include <string.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>

#define SL 81

enum flag { nao, sim };

/* DEFINICAO DE CLASSES */

class vector          //classe de vetores com suas principais rotinas
{
  public:
    double  *v;
    int  size;
    void  inicializar(double val=0);
    void  destruir(void);
};

class matrix          //classe de matrizes com suas principais rotinas
{
  public:
    double  **m;
    int     lin,col;
    void  inicializar(double val=0);
    void  destruir(void);
};

class screen    // objeto contendo todas as funcoes de tela
{
  public:
    virtual void limpa_tela(void);
    virtual void quadro(int X1,int Y1,int X2,int Y2,int C1,int C2);
    virtual void centro(char TXT[],int Y,int C1,int C2);
    virtual void escrever(char txt[],int x,int y,int c1,int c2);
    virtual void tela_fundo(void);
    virtual void tela_principal(void);
    virtual void opcao(void);
    virtual void tela_sobregravar(void);
    virtual void impossivel_abrir(void);
    virtual void erro_leitura(void);
```

```
    virtual void nao_definida(void);
};

class tela_define:public screen // objeto contendo todas as funcoes de tela
{                       // necessarias para a definicao da nova rede
  public:
    void principal(void);
    void entradas(void);
    void saidas(void);
    void neuronios(void);
    void aviso(void);
};

class tela_abrir:public screen    // objeto contendo todas as funcoes de tela
{                       // necessarias para abrir uma rede
  public:
    void principal(void);
    void pede_arquivo(void);
    void aviso(void);
};

class tela_treino:public screen  // objeto contendo todas as funcoes de tela
{                       // necessarias para treinar a rede
  public:
    void principal(void);
    void pede_dados(void);
    void pede_parametros(void);
    void em_treinamento(void);
    void resultado_parcial(void);
    void resultado_final(void);
    void barra_erro(double erro);
    void aviso(void);
};

class tela_execucao:public screen  // objeto contendo todas as funcoes de tela
{                       // necessarias para a execucao da nova rede
  public:
    void principal(int emax);
};

class tela_exec_arq:public screen  // objeto contendo todas as funcäes de tela
{                       // necessarias para o teste da rede
  public:
    void principal(void);    // tela principal
    void pede_arquivo(void); // tela que pede arquivo de dados para execucÆo da rede
    void fim_da_execucao(void); // tela indicando fim da execucÆo
};


class impressao:public screen    // objeto contendo as funcoes para a impressao
{                     // da rede em arquivo
  public:
    void confirma(void);
    void pesos(void);
};

class tela_teste:public screen  // objeto contendo todas as funcoes de tela
{                     // necessarias para o teste da rede
  public:
```

```
    void pede_arquivo(void);
    void resultado(void);
    void principal(void);
};

class treino          // objeto contendo todas as funcoes e variaveis
{                     // necessarias para o treinamento da rede
  public:
    vectore1,
        e2,
        ent,
        sai,
        dados;
    matrixdelta1,
        delta2;
    FILE *arqdad;
    void  espera_tecla(void);   // funcÆo que espera uma tecla ser pressionada
    int   abre_arquivo(char *arqdad);  // funcÆo que abre o arquivo de dados
    void  fecha_arquivo(void); // funcÆo que fecha o arquivo de dados para treinamento
    int   le_dados(int emax, int smax, vector &entrada, vector &saida); // funcÆo que l^ os dados de
treinamento do arquivo
    void  leitura_parametros(double &beta_ini,double &lamb_ini,int &n_escf,double &beta_pos,
          double &lamb_pos,double &errod,unsigned long int &n_max);
    double  ativ(double x);
    double  dativ(double x);
    void  imprime_resultado_parcial(unsigned long int iter,double erro_max,double beta_ini,double lamb_ini);
    void  imprime_resultado_final(unsigned long int iter,double erro_max);
    void  imprime_arquivo(int emax, int smax, int nmax, matrix w1, matrix w2);
};

class teste           // objeto contendo todas as funcoes e variaveis
{                     // necessarias para o teste da rede
  public:
    vectore1,
        e2,
        ent,
        sai;
    matrixdados;

    void imprime(int iter, double erro_max);
};

class net             // Objeto que representa a rede neural.
{                     // As variaveis e as principais subrotinas da
                      // rede estao incluidas neste objeto
  private:
    vectors1,
        s2,
        s3;
    matrixw1,
        w2;
  public:
    flag inicializado;
    screenscr;
    int   pega_tempo(void); // Pega o tempo para gerar os randomicos
    int   emax,smax,nmax;
    void  espera_tecla(void);
    int   pede_arq_dados(char *(&arqdad));
    int   verifica_inicializacao(void); // Verifica inicializacao da rede
```

```
    void   nova_rede(void);// Cria nova rede
    void   abrir_rede(void);// Abre rede ja existente
    int    leitura_dados(int caso, matrix &dados);  // Leitura de dados para treino e teste
    double  acha_erro(vector sai);     // Acha o erro referente ao treino
    void   treinar_rede(void);   // Treina a rede
    void   testar_rede(void);  // Testa a rede
    void   saidas(vector ent);    // Acha as saidas
    void   destruir(void);    // Destroi (desaloca) as variaveis utilizadas
    void   executar_teclado(void);     // Calcula a saida dada as entradas via teclado
    void   executar_arquivo(void);     // Calcula a saida dada as entradas via arquivo
};


/* DEFINICAO DAS FUNCOES MEMBRO DE CADA CLASSE */

void vector::inicializar(double val)
/* Inicializa um vetor atribuindo a seus elementos um valor inicial */
{
  int i;

  v = new double [size];
  for(i=0;i<size;i++)
    v[i]=val;
}

void vector::destruir(void)
/* Destroi (desaloca) o vetor existente */
{
  delete v;
  v = NULL;
}

void matrix::inicializar(double val)
/* inicializa matriz atribuindo a seus elementos um valor inicial */
{
  int i,j;

  m = new double *[lin];
  for(i=0;i<lin;i++)
  {
    m[i] = new double [col];
    for(j=0;j<col;j++)
      m[i][j]=val;
  }
}

void matrix::destruir(void)
/* destroi (desaloca) matrix existente */
{
  int i;

  for(i=0;i<lin;i++)
    delete m[i];
  delete m;
  m=NULL;
}

void screen::limpa_tela(void)
/*Limpa a tela */
```

```
{
  clrscr();
}

void screen::quadro(int X1,int Y1,int X2,int Y2,int C1,int C2)
/*Funcao de tela que desenha um quadro dados os extremos de sua diagonal
  e as cores do texto e do fundo                                    */
{
  char T[SL];
  int F;

  for(F=0;F<SL;F++)
    T[F]=0;
  textbackground(C2);
  textcolor(C1);
  T[0]=201;
  for(F=X1+1;F<X2;F++)
    T[F-X1]=205;
  T[X2-X1]=187;
  gotoxy(X1,Y1);
  cprintf(T);
  T[0]=186;
  for(F=X1+1;F<X2;F++)
    T[F-X1]=32;
  T[X2-X1]=186;
  for(F=Y1+1;F<Y2;F++)
  {
    gotoxy(X1,F);
    cprintf(T);
  }
  T[0]=200;
  for(F=X1+1;F<X2;F++)
    T[F-X1]=205;
  T[X2-X1]=188;
  gotoxy(X1,Y2);
  cprintf(T);
  textcolor(7);
  textbackground(0);
  for(F=X1;F<=X2;F++)
    T[F-X1]=176;
  gotoxy(X1+1,Y2+1);
  cprintf(T);
  T[0]=176;
  T[1]=0;
  for(F=Y1+1;F<=Y2;F++)
  {
    gotoxy(X2+1,F);
    cprintf(T);
  }
  textbackground(0);
  textcolor(15);
}

void screen::centro(char TXT[],int Y,int C1,int C2)
/*funcao que centraliza um texto dado a linha e as cores do texto e de fundo */
{
  int X;

  X=(80-strlen(TXT))/2;
```

```
    textcolor(C1);
    textbackground(C2);
    gotoxy(X,Y);
    cprintf(TXT);
    textcolor(15);
    textbackground(0);

}

void screen::escrever(char txt[],int x,int y,int c1,int c2)
/*escreve um texto na posicao (x,y) dado as cores de texto e de fundo */
{
    textcolor(c1);
    textbackground(c2);
    gotoxy(x,y);
    cprintf(txt);
    textcolor(15);
    textbackground(0);
}

void screen::tela_fundo(void)
/*Cria a tela de fundo deste programa */
{
    char TXT[SL];
    int F;

    limpa_tela();

    textbackground(0);
    textcolor(15);
    for (F=0;F<80;F++)
      TXT[F]=32;
    TXT[80]=0;
    cprintf(TXT);
    sprintf(TXT,"REDES NEURAIS-1998 = BACKPROPAGATION NEURAL NETWORK");
    centro(TXT,1,15,0);
    gotoxy(1,2);
    textcolor(7);
    for (F=0;F<80;F++)
      TXT[F]=178;
    TXT[80]=0;
    for (F=0;F<23;F++)
      cprintf(TXT);
}

void screen::tela_principal(void)
/*Cria a tela principal com o seu menu */
{
    tela_fundo();
    quadro(14,4,66,15,14,1);
    centro("MENU PRINCIPAL",5,15,1);
    escrever("1.DEFINIR NOVA REDE NEURAL",17,7,15,1);
    escrever("2.ABRIR REDE NEURAL",17,8,15,1);
    escrever("3.TREINAR REDE NEURAL EXISTENTE",17,9,15,1);
    escrever("4.TESTAR A REDE NEURAL",17,10,15,1);
    escrever("5.EXECUTAR REDE COM DADOS DO TECLADO",17,11,15,1);
    escrever("6.EXECUTAR REDE COM ARQUIVO",17,12,15,1);
    escrever("7.SAIR",17,13,15,1);
    quadro(25,20,55,22,14,1);
```

```
  escrever("Digite a sua opcao: ",27,21,15,1);
}

void net::espera_tecla(void)
/*Cria a tela esperando que uma tecla seja pressionada  */
{
  screenscr;

  scr.quadro(20,21,60,23,14,4);
  scr.escrever("Tecle algo para prosseguir: ",23,22,15,4);
  getch();
}

void screen::opcao(void)
/*Cria o quadro onde se deve digitar a opcao desejada  */
{
  quadro(30,18,50,20,14,1);
  escrever("Opcao:  ",34,19,15,1);
}

void screen::tela_sobregravar(void)
/*Cria o quadro que pergunta se ira sobregravar os dados ja existentes */
{
  quadro(14,12,66,14,14,4);
  escrever("Dados ja existentes!!! Sobregravar (s/n)? ",17,13,15,4);
}

void screen::impossivel_abrir(void)
/*  Apresenta um aviso de erro     */
{
  quadro(10,12,70,14,14,4);
  escrever("ERRO: NAO FOI POSSIVEL CRIAR OS ARQUIVOS ESPECIFICADOS!!!",12,13,15,4);
}

void screen::erro_leitura(void)
/*Apresenta um aviso para um erro de leitura  */
{
  quadro(10,12,70,14,14,4);
  escrever("ERRO NA LEITURA DOS ARQUIVOS ESPECIFICADOS!!!",14,13,15,4);
}

void screen::nao_definida(void)
/*Apresenta o aviso quando a rede neural nao esta definida  */
{
  quadro(10,12,69,14,14,4);
  escrever("ERRO: REDE NEURAL NAO DEFINIDA!!!",23,13,15,4);
}

void tela_define::principal(void)
/*Cria a tela principal na definicao de uma nova rede neural */
{
  tela_fundo();
  quadro(14,4,66,9,14,1);
  escrever("O programa cria uma rede neural de 3 camadas com",16,5,15,1);
  escrever("no de polarizacao.",16,6,15,1);
  escrever("Esta rede sera definida pelos arquivos de ",16,7,15,1);
  escrever("extensoes: .w1, .w2 e .sz",16,8,15,1);
}
```

```
void tela_define::entradas(void)
/* Cria o quadro pedindo as entradas  */
{
  quadro(14,12,66,14,14,1);
  escrever("Numero de entradas:  ",17,13,15,1);
}

void tela_define::saidas(void)
/*Cria o quadro pedindo as saidas   */
{
  quadro(14,12,66,14,14,1);
  escrever("Numero de saidas:  ",17,13,15,1);
}

void tela_define::neuronios(void)
/*Cria o quadro pedindo op numero de neuronios da rede */
{
  quadro(14,12,66,14,14,1);
  escrever("Numero de neuronios:  ",17,13,15,1);
}

void tela_define::aviso(void)
/*Apresenta um aviso dizendo que os arquivos foram criados  */
{
  quadro(9,12,71,14,14,4);
  escrever("Rede definida. Tecle algo para voltar ao menu principal",11,13,15,4);
}

void tela_abrir::principal(void)
/*Cria a tela principal na abertura de uma rede neural ja existente  */
{
  tela_fundo();
  quadro(14,4,66,8,14,1);
  escrever("O programa abre uma rede neural ja existente.",18,5,15,1);
  escrever("Estes arquivos devem possuir extensoes: .w1, ",18,6,15,1);
  escrever(".w2 e .sz",18,7,15,1);
}

void tela_abrir::pede_arquivo(void)
/*Cria o quadro onde se pede o nome do arquivo da rede  */
{
  quadro(14,12,66,14,14,1);
  escrever("Rede a ser aberta (sem extensao):  ",17,13,15,1);
}

void tela_abrir::aviso(void)
/*Apresenta o aviso de que os dados foram coletados */
{
  quadro(24,16,56,18,14,1);
  escrever("Dados Coletados",32,17,15,1);
}

void tela_treino::principal(void)
/*Cria a tela principal no treinamento da rede neural  */
{
  tela_fundo();
  quadro(14,4,66,7,14,1);
  escrever("O programa treina uma rede neural.",24,5,15,1);
  escrever("E necessario que a rede neural esteja predefinida",16,6,15,1);
```

```
}

void tela_treino::pede_dados(void)
/*Cria um quadro pedindo o arquivo de dados  */
{
  quadro(14,12,66,14,14,1);
  escrever("Arquivo de dados (sem extensao):  ",17,13,15,1);
}

void tela_treino::pede_parametros(void)
/*Cria tela para a coleta dos parametros  */
{
  tela_fundo();
  quadro(10,4,70,15,14,1);
  escrever(" - TREINAMENTO DE UMA REDE NEURAL - ",32,5,14,1);
  escrever("                  Beta inicial: ",12,7,14,1);
  escrever("                 Lambda inicial: ",12,8,14,1);
  escrever(" No. de iteracoes para escalonamento: ",12,9,14,1);
  escrever("  Fator para o escalonamento de beta: ",12,10,14,1);
  escrever("Fator para o escalonamento de lambda: ",12,11,14,1);
  escrever("      Erro quadratico global maximo: ",12,12,14,1);
  escrever("        Numero maximo de iteracoes: ",12,13,14,1);
}

void tela_treino::em_treinamento(void)
/*Cria o aviso durante o treinamento   */
{
  quadro(14,21,66,23,14,2);
  escrever("EM TREINAMENTO... Pressione P para interromper.",16,22,143,2);
}

void tela_treino::resultado_parcial(void)
/*Apresenta os resultados parciais do treinamento   */
{
  tela_fundo();
  quadro(10,4,70,16,14,1);
  escrever(" - TREINAMENTO DE UMA REDE NEURAL - ",20,5,14,1);
  escrever("        Numero de iteracoes: ",12,7,14,1);
  escrever("Erro quadratico global maximo: ",12,8,14,1);
  escrever("                  Beta: ",12,9,14,1);
  escrever("                 Lambda: ",12,10,14,1);
}

void tela_treino::resultado_final(void)
/*Apresenta o resultado final do treinamento */
{
  quadro(10,18,70,23,14,4);
  escrever(" - FIM DO TREINAMENTO - ",28,19,14,4);
  escrever("   Numero de iteracoes realizadas: ",14,21,15,4);
  escrever("Erro maximo nas ultimas iteracoes: ",14,22,15,4);
}

void tela_treino::barra_erro(double erro)
/*Cria uma barra onde o erro e apresentado. Quando a barra fica estatica
  nao ocorre mais variacao do erro                          */
{
  char TXT[SL];
  int i,a,b,tipo;
  double nivel_max=1.0;
```

```
    for (i=0;i<51;i++)
      TXT[i]=178;
    TXT[51]=0;

    tipo = 177;
    erro *= 100;
    if( erro >= 50 )      // Acima de 50% de erro a barra fica no maximo
    {
      a = 1;
      b = 49;
    }
    else
    {
      while( erro <= 1 )
      {
        erro *= 10;         // Varia-se a escala da barra e a cada
        nivel_max /= 10; // variacao altera-se a cor da barra
        if(tipo==176)
          tipo = 177;
        else
          tipo = 176;
      }
      if( erro >= 50 )
      {
        a = 1;
        b = 49;
      }
      else
      {
        a = 25 - (int)(erro/2);
        b = 25 + (int)(erro/2);
      }
    }
    for(i=0;i<a;i++)
      TXT[i] = tipo;
    for(i=50;i>b;i--)
      TXT[i] = tipo;
    gotoxy(15,12);
    cprintf(TXT);
    gotoxy(15,13);
    cprintf(TXT);
    gotoxy(15,14);
    cprintf(TXT);
    gotoxy(32,15);
    textcolor(14);
    textbackground(1);
    cprintf("escala: %lf %",nivel_max);
}

void tela_treino::aviso(void)
/* Apresenta um aviso apos a criacao dps arquivos  */
{
  quadro(9,12,71,14,14,4);
  escrever("Arquivos Criados. Tecle algo para voltar ao menu principal",11,13,15,4);
}

void tela_execucao::principal(int emax)
/* Cria a tela principal na execucao(calculo) na rede neural */
```

```
{
  char *conv;
  conv = new char [20];

  tela_fundo();
  quadro(14,3,66,6,14,1);
  escrever(" - EXECUCAO DA REDE NEURAL - ",26,4,14,1);
  sprintf(conv,"Numero de entradas: %d",emax);
  escrever(conv,30,5,15,1);
  quadro(8,8,38,19,14,1);
  escrever("- ENTRADAS -",17,9,14,1);
  quadro(40,8,71,19,14,1);
  escrever("- SAIDAS -",50,9,14,1);
  delete conv;
}

void tela_exec_arq::pede_arquivo(void)
/*Cria um quadro pedindo o arquivo de execucao  */
{
  quadro(14,8,66,12,14,1);
  escrever("Obtencao das saidas a partir de um arquivo *.ent",17,9,15,1);
  escrever("Arquivo de entradas (sem extensao):  ",17,11,15,1);
}

void tela_exec_arq::fim_da_execucao(void)
/*Apresenta o fim da execucao  */
{
  quadro(10,16,70,18,14,4);
  escrever(" - FIM DA EXECU□ÇO - ",28,17,14,4);
}

void tela_exec_arq::principal(void)
/*Cria a tela principal na execucao da rede neural com um arquivo de entradas  */
{
  tela_fundo();
  quadro(14,3,66,5,14,1);
  escrever(" - EXECU□ÇO DA REDE NEURAL - ",27,4,14,1);
}


void tela_teste::pede_arquivo(void)
/*Cria um quadro pedindo o arquivo de teste*/
{
  quadro(14,8,66,10,14,1);
  escrever("Arquivo de teste (sem extensao):  ",17,9,15,1);
}

void tela_teste::resultado(void)
/*Apresenta a tela de resultados do teste */
{
  quadro(10,14,70,19,14,4);
  escrever(" - RESULTADO DO TESTE - ",28,15,14,4);
  escrever("Numero de dados utilizados: ",18,17,15,4);
  escrever("      Erro maximo obtido: ",18,18,15,4);
}

void tela_teste::principal(void)
/*Cria a tela principal no teste da rede neural */
{
```

```
  tela_fundo();
  quadro(14,3,66,5,14,1);
  escrever(" - TESTE DA REDE NEURAL - ",27,4,14,1);
}

void impressao::confirma(void)
/*Cria o quadro onde voce diz se quer ou nao imprimir em arquivo*/
{
  tela_fundo();
  quadro(14,12,66,14,14,1);
  escrever("Deseja guardar a rede em um arquivo (s/n)? ",17,13,15,1);
}

void impressao::pesos(void)
/*  Cria a tela de para a impressao dos pesos em arquivo   */
{
  tela_fundo();
  quadro(14,12,66,14,14,1);
  escrever("Entre nome para o arquivo da rede: ",17,13,15,1);
  quadro(14,6,66,9,14,1);
  escrever("IMPRESSAO DOS PESOS DA REDE EM UM ARQUIVO",20,7,15,1);
  escrever("Os arquivos terao extensoes: .w1, .w2 e .sz",16,8,15,1);
  quadro(14,12,66,14,14,1);
  escrever("Entre nome para o arquivo da rede: ",17,13,15,1);
}

int net::pega_tempo(void)
/*Pega o tempo para a utilizacao dos randomicos */
{
  struct dostime_t t;

  _dos_gettime(&t);
  return t.hsecond;
}

int net::verifica_inicializacao(void)
/*Verifica a inicializacao das variaveis a serem utilizadas   */
{
  char c;

  if(inicializado==sim)
  {
    do
    {
      scr.tela_sobregravar();
      c=getchar();
      if( c == 'N' || c == 'n')
        goto end;
      else
        if( c == 'S' || c == 's')
        {
          w1.destruir();
          w2.destruir();
          s1.destruir();
          s2.destruir();
          s3.destruir();

          return 1;
        }
```

```
    }while( c != 'N' && c != 'n' && c != 'S' && c != 's' );
  }
  else
    return 1;
  end:
  return 0;
}

void net::nova_rede(void)
/*Cria uma nova rede  */
{
  int i,j;
  tela_define tela;

  tela.principal();

  if( !(verifica_inicializacao()) )
    goto end;

  tela.entradas();
  scanf("%d",&emax);
  tela.saidas();
  scanf("%d",&smax);
  tela.neuronios();
  scanf("%d",&nmax);

  srand(pega_tempo());
  w1.lin=smax;
  w1.col=nmax+1;
  w1.inicializar();
  for(i=0;i<smax;i++)
  {
    for(j=0;j<=nmax;j++)
    {
      w1.m[i][j]= random(1001);
      w1.m[i][j] /=1000;
      w1.m[i][j] += 0.2;
    }
  }

  srand(pega_tempo());
  w2.lin=nmax;
  w2.col=emax+1;
  w2.inicializar();
  for(i=0;i<nmax;i++)
  {
    for(j=0;j<=emax;j++)
    {
      w2.m[i][j]=random(1001);
      w2.m[i][j]/=1000;
      w2.m[i][j]+= 0.1;
    }
  }
  inicializado = sim;
  s1.size = smax+1;
  s2.size = nmax+1;
  s3.size = emax+1;
  s1.inicializar(0.7);
  s2.inicializar(0.4);
```

```
  s3.inicializar(0.6);

  end:
}

void net::abrir_rede(void)
/* Le (abre) uma rede de um dado arquivo */
{
  int  i,j;
  char  c, *nome1=NULL, *nome2=NULL, *nome3=NULL,
       *nomearq=NULL, *s=NULL, palavra[12];
  FILE *arquivo1=NULL,*arquivo2=NULL,*arquivo3=NULL;
  tela_abrir tela;

  tela.principal();

  if( !(verifica_inicializacao()) )
    goto end;

  nomearq = new char [15];
  nome1 = new char [15];
  nome2 = new char [15];
  nome3 = new char [15];
  s = new char [15];

  tela.pede_arquivo();
  scanf("%s",nomearq);
  strcpy(nome1,nomearq);
  strcpy(nome2,nomearq);
  strcpy(nome3,nomearq);
  strcat(nome1,".w1");
  strcat(nome2,".w2");
  strcat(nome3,".sz");

  if ( ((arquivo1=fopen(nome1,"rt"))==NULL)||((arquivo2=fopen(nome2,"rt"))==NULL)
    ||((arquivo3=fopen(nome3,"rt"))==NULL) )
  {
    tela.impossivel_abrir();
    inicializado = nao;
    goto erro;
  }

  while( !feof(arquivo3) )
  {
    if ( !fscanf(arquivo3, "%d %d %d", &emax, &smax, &nmax) )
    {
      tela.erro_leitura();
      goto erro;
    }
  }
  fclose( arquivo3 );

  w1.lin=smax;
  w1.col=nmax+1;
  w1.inicializar();

  while( !feof(arquivo1) )
  {
    for(i=0;i<smax;i++)
```

```
      for(j=0;j<=nmax;j++)
        if( !fscanf(arquivo1, "%lf", &w1.m[i][j]) )
        {
          tela.erro_leitura();
          goto erro;
        }
}
fclose( arquivo1 );

w2.lin=nmax;
w2.col=emax+1;
w2.inicializar();

while( !feof(arquivo2) )
{
  for(i=0;i<nmax;i++)
    for(j=0;j<=emax;j++)
      if( !fscanf(arquivo2, "%lf", &w2.m[i][j]) )
      {
        tela.erro_leitura();
        goto erro;
      }
}
fclose( arquivo2 );

s1.size = smax+1;
s2.size = nmax+1;
s3.size = emax+1;
s1.inicializar(0.3);
s2.inicializar(0.4);
s3.inicializar(0.6);
inicializado = sim;

delete nomearq;
delete nome1;
delete nome2;
delete nome3;
nomearq = NULL;
nome1 = NULL;
nome2 = NULL;
nome3 = NULL;
delete s;
s = NULL;
tela.aviso();

erro:

fclose( arquivo1 );
fclose( arquivo2 );
fclose( arquivo3 );
delete nomearq;
delete nome1;
delete nome2;
delete nome3;
nomearq = NULL;
nome1 = NULL;
nome2 = NULL;
nome3 = NULL;
delete s;
```

```
  s = NULL;
  espera_tecla();

  end:
}

double net::acha_erro(vector sai)
/*Calcula o erro obtido com as entradas e saidas  */
{
  double erro;
  int i;

  erro = 0;
  for(i=0;i<smax;i++)
  {
    erro += pow((sai.v[i]-s1.v[i]),2);
  }
  erro /= 2;
  return erro;
}

int net::leitura_dados(int caso, matrix &dados)
/*Funcao que faz a leitura dos dados para treinamento(0)  e teste(1)  */
{
  char *arqdad;
  int  i, j, n_dados;
  FILE *fdados=NULL;
  tela_teste  tela1;
  tela_treinotela2;

  if(caso)
    tela1.pede_arquivo();// pede arquivo de dados para o teste
  else
    tela2.pede_dados();      // pede arquivo de dados para o treinamento

  arqdad = new char [20];
  scanf("%s",arqdad);
  strcat(arqdad,".trn");            // extensao .trn para arq de dados
  if( (fdados=fopen(arqdad,"rt"))==NULL)
  {
    tela2.impossivel_abrir();
    espera_tecla();
    goto erro;
  }
  else
  {
    if(!fscanf(fdados,"%d",&n_dados))
    {
      tela2.erro_leitura();
      goto erro;
    }
    dados.lin = n_dados;
    dados.col = emax+smax;
    dados.inicializar();
    while( !feof(fdados) )
    {
      for(i=0;i<n_dados;i++)
        for(j=0;j<(emax+smax);j++)
          if(!fscanf(fdados,"%lf",&dados.m[i][j]))
```

```
            {
              tela2.erro_leitura();
              goto erro;
            }
      }
      fclose(fdados);
      delete arqdad;
      return 1;
    }

    erro:
      fclose(fdados);
      delete arqdad;
      return 0;
}

void treino::espera_tecla(void)
/* Tela esperando que uma tecla seja pressionada */
{
  screen scr;

  scr.quadro(20,21,60,23,14,4);
  scr.escrever("Tecle algo para prosseguir: ",23,22,15,4);
  getch();
}



int treino::abre_arquivo(char *nome_arqdad)
/* Abre arquivo de dados para treinamento e retorna o numero de dados para treinamento */
{
  int  i, j, n_dados;
  tela_treino tela;

  if( (arqdad=fopen(nome_arqdad,"rt"))==NULL)     // abre arquivo
  {
    tela.impossivel_abrir();
    espera_tecla();
    fclose(arqdad);
    return 0;
  }
  else
  {
    if(!fscanf(arqdad,"%d",&n_dados))  // l^ numero de dados
    {
      tela.erro_leitura();
      fclose(arqdad);
      espera_tecla();
      return 0;
    }
    else
    {
      rewind(arqdad);
      return n_dados;
    }
  }
}

void treino::fecha_arquivo(void)
```

```
/*Fecha arquivo de dados para treinamento   */
{
  fclose(arqdad);
}


int treino::le_dados(int emax, int smax, vector &entradas, vector &saidas)
/*Realiza a leitura dos dados para treinamento e retorna zero caso isso nao seja possivel */
{
  int  j;
  double norma=0.0;
  tela_treinotela;

  for(j=0;j<emax;j++)
  {
    if(!fscanf(arqdad,"%lf",&entradas.v[j]))
    {
      tela.erro_leitura();
      fclose(arqdad);
      return 0;
    }
    norma += entradas.v[j]*entradas.v[j];
  }
  norma=sqrt(fabs(norma));
  for(j=0;j<emax;j++)
    entradas.v[j] /= norma;

  for(j=0;j<smax;j++)
    if(!fscanf(arqdad,"%lf",&saidas.v[j]))
    {
      tela.erro_leitura();
      fclose(arqdad);
      return 0;
    }
  return 1;
}



void treino::leitura_parametros(double &beta_ini,double &lamb_ini,int &n_escf,double &beta_pos,
            double &lamb_pos,double &errod,unsigned long int &n_max)
/*Realiza a leitura dos parametros do treino  */
{
  tela_treinotela;

  tela.pede_parametros();      // tela dando pedindo todos os parametros
  gotoxy(50,7);
  scanf("%lf",&beta_ini);
  gotoxy(50,8);
  scanf("%lf",&lamb_ini);
  do
  {
    gotoxy(50,9);
    scanf("%d",&n_escf);
  }
  while(n_escf<0);
  gotoxy(50,10);
  scanf("%lf",&beta_pos);
  gotoxy(50,11);
```

```
  scanf("%lf",&lamb_pos);
  do
  {
    gotoxy(50,12);
    scanf("%lf",&errod);
  }
  while(errod<0);
  gotoxy(50,13);
  scanf("%ld",&n_max);
}

void treino::imprime_resultado_parcial(unsigned long int iter,double erro_max,double beta_ini,double lamb_ini)
/*Imprime resultados parciais para o treinamento */
{
  char *conv;
  screenscr;

  conv = new char [20];
  sprintf(conv,"  %ld ",iter);
  scr.escrever(conv,42,7,14,1);
  sprintf(conv," %12.8lf ",erro_max);
  scr.escrever(conv,42,8,14,1);
  sprintf(conv," %12.8lf ",beta_ini);
  scr.escrever(conv,42,9,14,1);
  sprintf(conv," %12.8lf ",lamb_ini);
  scr.escrever(conv,42,10,14,1);
  delete conv;
}

void treino::imprime_resultado_final(unsigned long int iter,double erro_max)
/*Imprime o resultado final do treinamento*/
{
  char *conv;
  tela_treinotela;
  screenscr;

  conv = new char [20];

  tela.resultado_final();   // Indica fim de treinamento e cria quadro
                // para impressao dos resultados
  sprintf(conv,"  %ld ",iter);
  scr.escrever(conv,50,21,14,4);
  sprintf(conv,"  %lf ",erro_max);
  scr.escrever(conv,50,22,14,4);
  delete conv;
}

void treino::imprime_arquivo(int emax, int smax, int nmax, matrix w1, matrix w2)
/*  Imprime nos arquivos (.w1, .w2, .sz) a rede neural */
{
  int i,j;
  char c, *nome1=NULL, *nome2=NULL, *nome3=NULL, *nomearq=NULL;
  FILE *arquivo1 = NULL,*arquivo2 = NULL,*arquivo3 = NULL;
  impressao impr;
  tela_treinotela;

  do
  {
    impr.confirma();
```

```
    c=getchar();
    if( c == 'N' || c == 'n')
      goto end;
  }while( c != 'N' && c != 'n' && c != 'S' && c != 's' );

  nomearq = new char [15];
  nome1 = new char [15];
  nome2 = new char [15];
  nome3 = new char [15];

  impr.pesos();
  scanf("%s",nomearq);
  strcpy(nome1,nomearq);
  strcat(nome1,".w1");
  strcpy(nome2,nomearq);
  strcat(nome2,".w2");
  strcpy(nome3,nomearq);
  strcat(nome3,".sz");

  if ( ((arquivo1=fopen(nome1,"wt"))==NULL)||((arquivo2=fopen(nome2,"wt"))==NULL)
    ||((arquivo3=fopen(nome3,"wt"))==NULL) )
  {
    tela.impossivel_abrir();
  }
  else
  {
    fprintf(arquivo3,"\n%d",emax);
    fprintf(arquivo3,"\n%d",smax);
    fprintf(arquivo3,"\n%d",nmax);
    fclose(arquivo3);

    for(i=0;i<smax;i++)
      for(j=0;j<=nmax;j++)
      {
        fprintf(arquivo1," %10.6lf",w1.m[i][j]);
        if(j==nmax)
          fprintf(arquivo1,"\n");
      }
    fclose(arquivo1);

    for(i=0;i<nmax;i++)
      for(j=0;j<=emax;j++)
      {
        fprintf(arquivo2," %10.6lf",w2.m[i][j]);
        if(j==emax)
          fprintf(arquivo2,"\n");
      }

    fclose(arquivo2);
    tela.aviso();
  }
  delete(nomearq);
  delete(nome1);
  delete(nome2);
  delete(nome3);

  end:
}
```

```
double treino::ativ(double x)
/* Funcao de ativacao */
{
  if(x>50.0)
    return 1.0;
  else
  {
    if(x<-50.0)
      return 0.0;
    else
      return 1.0/(1.0+exp(-x));
  }
}

double treino::dativ(double x)
/* derivada da funcao de ativacao */
{
  return x*(1-x);
}

int net::pede_arq_dados(char *(&arqdad))
/*Funcao que faz a leitura dos dados para treinamento  e teste  */
{
  tela_treinotela;
  FILE *f_arqdad;

  tela.pede_dados();      // pede arquivo de dados para o treinamento

  arqdad = new char [20];
  scanf("%s",arqdad);
  strcat(arqdad,".trn");   // arquivo de extensao .trn para dados de treinamento

  if( (f_arqdad=fopen(arqdad,"rt"))==NULL)  // verifica se este arquivo existe. Em
  {                          // caso negativo, retorna o valor zero.
    tela.impossivel_abrir();
    espera_tecla();
    fclose(f_arqdad);
    return 0;
  }
  else
  {
    fclose(f_arqdad);
    return 1;
  }
}


void net::treinar_rede(void)
/*Treina a rede neural  */
{
  const aprf=100;
  char key, *nome_arqdad;
  double  erro, errod, erro_max=0, max,
      beta_ini, lamb_ini, beta_pos, lamb_pos;
  unsigned long int n_max,iter =0;
  int  i, j, n_esc, qdad, n_escf, apr, quant_dados;
  treino trn;
  tela_treinotela;
```

```
tela.principal();  //tela inicial

if( inicializado == nao )
{
  tela.nao_definida();   // tela: rede nao inicializada
  espera_tecla();
  goto end;
}

if(!pede_arq_dados(nome_arqdad))  // tela pedindo arquivo de dados para treinamento
  goto end;

trn.delta1.lin = smax;
trn.delta1.col = nmax+1;
trn.e1.size = smax;
trn.e2.size = nmax+1;
trn.delta2.lin = nmax;
trn.delta2.col = emax+1;
trn.ent.size = emax;
trn.sai.size = smax;
trn.ent.inicializar();
trn.sai.inicializar();
trn.e1.inicializar();
trn.e2.inicializar();
trn.delta1.inicializar();
trn.delta2.inicializar();

trn.leitura_parametros(beta_ini,lamb_ini,n_escf,beta_pos,lamb_pos,errod,n_max);

n_esc=n_escf;
tela.resultado_parcial(); // Indica o numero de iteracoes
            // o erro global maximo, o beta e o lambida
tela.em_treinamento();  // Mensagem: Em treinamento. Pressione P para interromper
apr = aprf;
srand(pega_tempo());

quant_dados = trn.abre_arquivo(nome_arqdad);   // abre arquivo de dados e recebe o
if( !quant_dados )                             // numero de dados para treinamento
  goto end;

do
{
  iter++;
  n_esc--;
  apr--;
  fscanf(trn.arqdad,"%d",&quant_dados);
  if(feof(trn.arqdad))
  {
    rewind(trn.arqdad);
    fscanf(trn.arqdad,"%d",&quant_dados);
  }

  if( !n_esc )
  {
    n_esc = n_escf;
    beta_ini*=beta_pos;
    lamb_ini*=lamb_pos;
  }
  trn.le_dados(emax,smax,trn.ent,trn.sai);     // leitura de um conjunto de dados de treinamento
```

```
    saidas(trn.ent);

    for(i=0;i<smax;i++)
    {
      trn.e1.v[i] = (trn.sai.v[i]-s1.v[i])*trn.dativ(s1.v[i]);
      for(j=0;j<=nmax;j++)
      {
        trn.delta1.m[i][j] = beta_ini*trn.e1.v[i]*s2.v[j] + lamb_ini*trn.delta1.m[i][j];
        w1.m[i][j] += trn.delta1.m[i][j];

      }
    }
    for(i=0;i<nmax;i++)
    {
      for(j=0;j<smax;j++)
        trn.e2.v[i] += trn.e1.v[j]*w1.m[j][i];
      trn.e2.v[i] *= trn.dativ(s2.v[i]);
      for(j=0;j<=emax;j++)
      {
        trn.delta2.m[i][j] = beta_ini*trn.e2.v[i]*s3.v[j] + lamb_ini*trn.delta2.m[i][j];
        w2.m[i][j] += trn.delta2.m[i][j];
      }
    }

    erro = acha_erro(trn.sai);

    if(erro>erro_max)
      erro_max = erro;

    if(iter>=n_max)
      key = 'p';

    if(!apr)
    {
      apr = aprf;
      trn.imprime_resultado_parcial(iter,erro_max,beta_ini,lamb_ini);
      tela.barra_erro(erro_max);
      if(erro_max<errod)
        key = 'p';
      if(key!='p')
        erro_max = 0;
    }
    if(kbhit())
      key = getch();
  }
  while( (key!='p') && (key!='P') );

  trn.imprime_resultado_final(iter,erro_max);
  getch();
  trn.imprime_arquivo(emax,smax,nmax,w1,w2);

  trn.ent.destruir();
  trn.sai.destruir();
  trn.e1.destruir();
  trn.e2.destruir();
  trn.delta1.destruir();
  trn.delta2.destruir();
```

```
  end:
}

void net::testar_rede(void)
/* Subrotina que faz o teste de uma rede neural ja treinada ou nao  */
{
  int i,iter=0,quant_dados;
  double erro=0;
  char *nome_arqdad;
  tela_teste tela;
  teste test;
  treino trn;

  tela.principal();

  if( inicializado == nao )
  {
    tela.nao_definida();   // tela: rede nao inicializada
    espera_tecla();
    goto end;
  }

  if(!pede_arq_dados(nome_arqdad))  // tela pedindo arquivo de dados para treinamento
    goto end;

  test.ent.size = emax;
  test.sai.size = smax;
  test.ent.inicializar();
  test.sai.inicializar();
  srand(pega_tempo());

  quant_dados = trn.abre_arquivo(nome_arqdad);  // abre arquivo de dados e recebe o
  if( !quant_dados )                            // numero de dados para treinamento
    goto end;
  fscanf(trn.arqdad,"%d",&quant_dados);

  erro=0;
  while( iter <= quant_dados )
  {
    iter++;

    if(feof(trn.arqdad))
    {
      rewind(trn.arqdad);
      fscanf(trn.arqdad,"%d",&quant_dados);
    }

    trn.le_dados(emax,smax,test.ent,test.sai);      // leitura de um conjunto de dados de treinamento

    saidas(test.ent);
    for(i=0;i<smax;i++)
    {
      if(s1.v[i]>=0.5)
        s1.v[i] = 1;
      else
        s1.v[i] = 0;
    }

    erro += acha_erro(test.sai);
```

```
    }
    erro /= quant_dados;

    test.imprime(iter-1,erro);

    test.ent.destruir();
    test.sai.destruir();
    espera_tecla();

    end:
}

void teste::imprime(int iter, double erro_max)
/*Imprime os resultados do teste realizado */
{
    char *conv;
    screen scr;
    tela_teste  tela;

    conv = new char [20];

    tela.resultado();      // Tela onde os resultados do teste sao impressos
    sprintf(conv,"  %d ",iter);
    scr.escrever(conv,48,17,14,4);
    sprintf(conv,"  %lf ",erro_max);
    scr.escrever(conv,48,18,14,4);
    delete conv;
}


void net::destruir(void)
/*Destroi as variaveis ainda alocadas na memoria */
{
    w1.destruir();
    w2.destruir();
    s1.destruir();
    s2.destruir();
    s3.destruir();
}

void net::saidas(vector ent)
/*Acha as saidas dada as entradas */
{
    int i,j;
    treino trn;

    for(i=0;i<smax;i++)
        s1.v[i] = 0;
    for(i=0;i<nmax;i++)
        s2.v[i] = 0;
    for(i=0;i<emax;i++)
        s3.v[i]=ent.v[i];
    s3.v[emax]=-1;

    for(i=0;i<nmax;i++)
    {
        for(j=0;j<=emax;j++)
            s2.v[i] += s3.v[j]*w2.m[i][j];
        s2.v[i] = trn.ativ(s2.v[i]);
```

```
    }
    s2.v[nmax]=1;
    for(i=0;i<smax;i++)
    {
      for(j=0;j<=nmax;j++)
        s1.v[i] += s2.v[j]*w1.m[i][j];
      s1.v[i] = trn.ativ(s1.v[i]);
    }
}

void net::executar_teclado(void)
/*Calcula a saida a ser obtida em uma rede previamente treinada atraves
  do teclado                                        */
{
  int i;
  char *conv;
  tela_execucao tela;
  screen scr;
  vector ent;

  tela.principal(emax);

  if( inicializado == nao )
  {
    tela.nao_definida();   // tela: rede nao inicializada
    espera_tecla();
    goto end;
  }

  conv = new char [40];
  ent.size = emax;
  ent.inicializar();

  for(i=0;i<emax;i++)
  {
    sprintf(conv,"Entrada %2d: ",i);
    scr.escrever(conv,10,11+i,15,1);
    gotoxy(23,11+i);
    scanf("%lf",&ent.v[i]);
  }
  saidas(ent);

  for(i=0;i<smax;i++)
  {
    sprintf(conv,"Saida %2d: %11.6lf",i,s1.v[i]);
    scr.escrever(conv,42,11+i,15,1);
  }
  espera_tecla();
  ent.destruir();
  delete conv;

  end:
}


void net::executar_arquivo(void)
/*Calcula as saidas a partir das entradas contidas em um arquivo */
{
  char *arq1, *arq2;
```

```
int  i, j, n_dados;
FILE *arq_ent=NULL, *arq_sai=NULL;
tela_exec_arq  tela;
vector ent;

tela.principal();

ent.size = emax;
ent.inicializar();


if( inicializado == nao )
{
  tela.nao_definida();   // tela: rede nao inicializada
  espera_tecla();
  goto end;
}

tela.pede_arquivo();  // pede arquivo de dados para a execucao

arq1 = new char [20];
arq2 = new char [20];
scanf("%s",arq1);
strcpy(arq2,arq1);
strcat(arq1,".ent");       // extensao *.ent para arq de entradas
strcat(arq2,".sai");       // extensao *.sai para arq de saidas
if( ((arq_ent=fopen(arq1,"rt"))==NULL) || ((arq_sai=fopen(arq2,"wt"))==NULL) )
{
  tela.impossivel_abrir();
  espera_tecla();
  goto erro;
}
else
{
  if(!fscanf(arq_ent,"%d",&n_dados))
  {
    tela.erro_leitura();
    goto erro;
  }
  for(i=0;(i<n_dados)&&(!feof(arq_ent));i++)
  {
    for(j=0;j<emax;j++)
      if(!fscanf(arq_ent,"%lf",&ent.v[j]))  // leitura de arquivo
      {
        tela.erro_leitura();
        goto erro;
      }
    saidas(ent);  // calculo das saidas
    for(j=0;j<smax;j++)
    {
      if(s1.v[j]>=0.5)
        s1.v[j] = 1;
      else
        s1.v[j] = 0;
      fprintf(arq_sai," %lf ",s1.v[j]);// impressao em arquivo
    }
    fprintf(arq_sai,"\n");
  }
```

```
    }
    tela.fim_da_execucao();
    espera_tecla();

    erro:
      fclose(arq_ent);
      fclose(arq_sai);
      delete arq1;
      delete arq2;
    end:
      ent.destruir();
}



/*PROGRAMA PRINCIPAL */

void main (void)
{
  screen scr;
  int i,opcao;
  char c;
  flag para;
  net rede;

  rede.inicializado = nao;

  do
  {
    de_novo:

    scr.tela_principal();
    gotoxy(48,21);
    if ( !scanf("%d",&opcao) ) goto de_novo;
    para = nao;
    if (opcao < 1 || opcao > 7)   goto de_novo;
    else
    {
      if ( opcao < 7 )
      {
        switch (opcao)
        {
          case 1:
            rede.nova_rede();
            break;
          case 2:
            rede.abrir_rede();
            break;
          case 3:
            rede.treinar_rede();
            break;
          case 4:
            rede.testar_rede();
            break;
          case 5:
            rede.executar_teclado();
            break;
```

```
            case 6:
              rede.executar_arquivo();
              break;
          }
        }
        else para = sim;
      }
    } while ( !para );

    if(rede.inicializado==sim)
      rede.destruir();
    scr.limpa_tela();
}
```

B. APÊNDICE 2 - A REDE COUNTERPROPAGATION EM C++

B.1 Detalhamento dos objetos

Nesta seção, serão apenas os objetos mais importantes da rede counterpropagation, explicando-se a finalidade de todas as variáveis e funções que os compõem.

- *treino:* objeto que contém todas as funções necessárias para o treinamento da rede neural. Além disso, possui variáveis que auxiliam esta parte do programa. O vetor winners indica a freqüência que um neuônio foi vencedor; o arquivo *arqdad* contém os padrões de treinamento; os inteiros *t_ini* e *t_fim* marcam os instantes iniciais e finais do treinamento; e *t_koh* e *t_gross* marcam a duração do treinamento kohonen e grossberg respectivamente. As funções deste objeto são:
  - *leitura_parâmetros_koh e leitura_parâmetros_gross:* faz a leitura dos parâmetros dos treinamentos kohonen e grossberg, e os passa por referência. Os parâmetros são: o coeficiente de aprendizado inicial e final, o erro admissível e o número de iterações máximo.
  - *abre_arquivo:* função que abre o arquivo de dados para treinamento e retorna o número de padrões de treinamento.
  - *fecha_arquivo:* função que fecha o arquivo de dados para treinamento.
  - *le_dados:* função que lê os dados o arquivo de treinamento.
  - *imprime_resultado_parcia_koh e imprime_resultado_parcia_gross:* como o próprio nome diz, faz a impressão dos resultados parciais durante o treinamento kohonen e grossberg.
  - *continua*: função que pergunta ser que continuar com o treinamento Grossberg.
  - *imprime_arquivo:* faz a impressão da rede neural para um arquivo.
  - *teste_imprime:* imprime os resultados do teste realizado
- *net:* objeto contém os pesos, dados pelas matrizes *weight_koh* e *weight_gross*, e as entradas e saídas, dadas pelos vetores *entrada* e *saida*. Possui um *flag* para indicar se a rede neural já foi ou não inicializada e um número inteiro *winner* para indicar qual o neurônio vencedor. As funções deste objeto são:
  - *pega_tempo:* pega o tempo e retorna os centésimos de segundo para a geração dos randômicos.

- *espera_tecla:* cria o quadro para esperar uma tecla ser pressionada.
- *verifica_inicialização:* verifica se a rede já foi ou não inicializada.
- *nova_rede:* cria uma nova rede.
- *abrir_rede:* abre uma rede existente em arquivo.
- *pede_arq_dados:* pede arquivo de dados tanto para o treinamento como para o teste da rede neural.
- *acha_erro:* dado o vetor de saída desejada, retorna o erro quadrático gerado a partir da saída obtida.
- *treinar_rede:* realiza o treinamento de uma rede neural já inicializada.
- *testar_rede:* realiza o teste de uma rede neural já inicializada.
- *saídas:* dadas as entradas, acha as saídas através da rede neural.
- *destruir:* destrói a rede neural desalocando da memória todos os vetores e matrizes.
- *executar_teclado:* calcula a saída dadas as entradas via teclado.
- *executar_arquivo:* calcula a saída dadas as entradas via arquivo.
- *dist_euclid:* calcula a distância euclidiana entre um vetor e a coluna ou linha de uma matriz de pesos.
- *acha_winner:* acha o neurônio vencedor.
- *ajusta_kohonen:* ajusta os pesos da camada kohonen.
- *ajusta_grossberg:* ajusta os pesos da camada grossberg.
- *tratar_pesos:* trata os pesos para melhorar o treinamento
- *corrige_pesos:* corrige os pesos para melhorar o treinamento
- *treino_kohonen:* chama todo o treinamento kohonen
- *treino_grossberg:* chama todo o treinamento grossberg

Deve-se ressaltar que os objetos de tela são chamados ao longo de todo o programa e também por outros objetos.

B.2 Diagramas de Nassi-Schneidermann

Agora que os objetos foram definidos e detalhados, haverá uma breve apresentação do algoritmo implementado através dos diagramas de Nassi-Schneidermann. As funções de telas não irão ser apresentadas por não serem relevantes ao programa. Apenas a rotina que cria a barra de erro deverá ser esquematizada a seguir.

Objeto: *net* - subrotina *treinar_rede*
imprime tela pincipal com *principal* de *tela*
T se *inicializado* for *não*
imprimir mensagem de rede não inicializada; esperar uma tecla e ir para *end*
T se ao chamar *leitura_dados* para o treino retornar zero
ir para *end*
chama *leitura_parametros_koh*
chama *resultado_parcial_koh*
chama *treino_kohonen*
T se *continua* for *não*
chama *leitura_parametros_gross*
chama *resultado_parcial_gross*
chama *treino_grossberg*
chama tela *fim_de_treino*
chama *imprime_arquivo*

Objeto: *net* - subrotina *treino_kohonen*

inicializa *winners*

t_ini chama *pega_tempo*

quant_dados chama *abre_arquivo*

se *quant_dados* for nulo

T

ir para *end*

se *altera_pesos* for *altera*

T

chamar *tratar_pesos*

inicializa vetor *winners*

para i de 0 a *quant_dados*

chama *le_dados*

chama *acha_winner*

incrementa *vetor winners*

chama *ajusta_kohonen*

voltar arquivo *arqdad* ao início

cálculo do *erro*

se nãofor a primeira iteração

T

chamar *corrige_pesos*

ajustar *alfa*

t_ini chama *pega_tempo*

t_fim recebe *pega_tempo*

imprimir tela de treinamento parcial

se a tecla "p" for pressionada

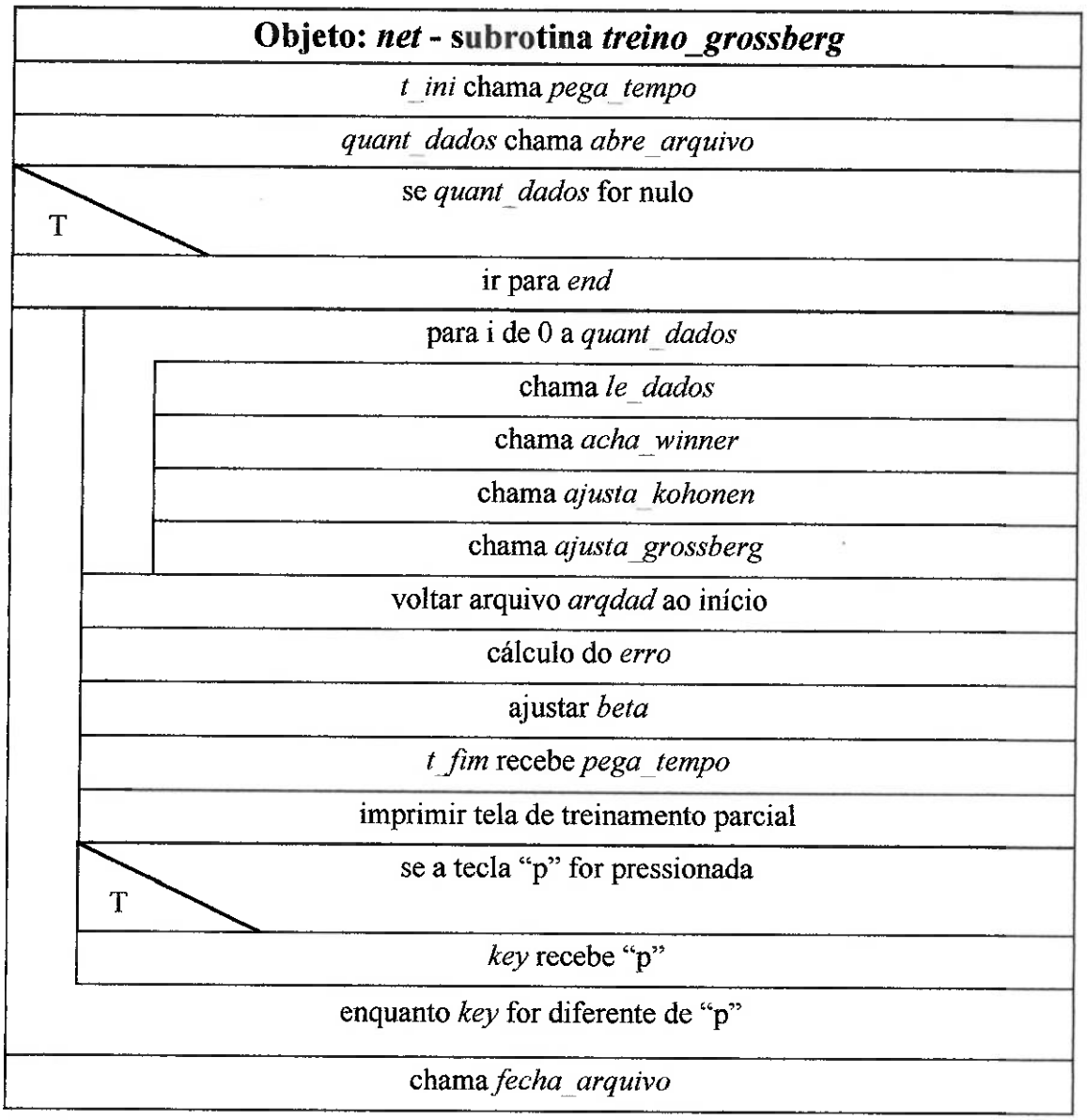
T

key recebe "p"

enquanto *key* for diferente de "p"

chama *fecha_arquivo*

destruir *winners*


Objeto: net - subrotina treino_grossberg
t_ini chama pega_tempo
quant_dados chama abre_arquivo
se quant_dados for nulo
T
ir para end
para i de 0 a quant_dados
chama le_dados
chama acha_winner
chama ajusta_kohonen
chama ajusta_grossberg
voltar arquivo arqdad ao início
cálculo do erro
ajustar beta
t_fim recebe pega_tempo
imprimir tela de treinamento parcial
se a tecla "p" for pressionada
T
key recebe "p"
enquanto key for diferente de "p"
chama fecha_arquivo

Objeto: *net* - subrotina *testar_rede*
inicializar *saida_desejada*
se *inicializado* for não T
imprime tela de não inicialização e vai para *end*
chama *pede_arq_dados* e se retornar 0 T
vai para *end*
quant_dados recebe a chamada de *abre_arquivo*
se *quant_dados* for nulo T
ir para *end*
para i de 0 a *quant_dados*
chama *le_dados*
iguala *saida_desejada* a *saida*
chama *saidas*
acha *erro*
divide *erro* por *quant_dados* para achar erro médio
chama *teste_imprime* para os resultados
destruir *saida_desejada*
chamar *fecha_arquivo* para fechar *arqdad*

Objeto: *net* - subrotina *dist_euclid*	
inicializar *dist=0*	
se *quant_dados* for nulo T	F
para *i* de 0 a *weight.col*	para *i* de 0 a *weight.lin*
incrementar *dist* com quadrado da diferença de *v* com a linha dada por *winner*	incrementar *dist* com quadrado da diferença de *v* com a coluna dada por *winner*
retorna *dist*	

Objeto: *net* - subrotina *acha_winner*
inicializar *min_dist=0* e *winner*=0
aux recebe a distância euclidiana das entradas em relação aos pesos chamando *dist_euclid*
com *i* de 0 a *weight_koh.lin*
aux recebe a distância euclidiana das entradas em relação aos pesos chamando *dist_euclid*
se *aux* for menor que *min_dist* (T)
min_dist recebe *aux* e *winner* recebe *i*

Objeto: *net* - subrotina *saídas*
zerar todos os elementos de *saida*
chamar *acha_winner*
igualar *saida* ao vetor de pesos *weight_gross* correspondente ao neurônio vencedor

Objeto: *net* - subrotina *destruir*
desaloca da memória as variáveis: *weight_koh*, *weight_gross*, *entrada* e *saida*

PROGRAMA PRINCIPAL

variáveis: *scr* (screen); *i*, *opção*(inteiros); *c* (caracter); *pára* (flag); *rede* (net)

o campo *inicializado* de *rede* recebe *não*

de_novo:

chamar *tela_principal* de *scr*

ler *opção* na posição correta

pára recebe *não*

se *opção* for menor do que 1 ou maior do que 7

T: ir para *de_novo*

F:

se *opção* for menor do que 7

T:

caso *opção* tenha valor:

1	chamar *nova_rede* de *rede*
2	chamar *abrir_rede* de *rede*
3	chamar *treinar_rede* de *rede*
4	chamar *testar_rede* de *rede*
5	chamar *executar_teclado* de *rede*
6	chamar *executar_arquivo* de *rede*

F: *pára* recebe *sim*

enquanto *pára* for igual a *não*

se o campo *inicializado* de *rede* for *sim*

T:

chamar *destruir* de *rede*

chamar *limpa_tela*

B.3 Listagem do Programa

```
//=========================================//
// REDES NEURAIS - COUNTERPROPAGATION //
//=========================================//
// Trabalho de Formatura - 1998      //
// CHANG CHIH WEI - No. USP 185014 //
// Reconhecimento de sinais          //
//=========================================//

#include <stdio.h>
#include <math.h>
#include <string.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>

#define SL 81
#define EPS 1E-5

enum flag { nao, sim };
enum tipo { KOHONEN, GROSSBERG };     // tipo de treinamento
enum t_pesos { altera, nao_altera }; // indica se vai haver tratamento de pesos

/* DEFINI□ÇO DE CLASSES */

class vector          //classe de vetores com suas principais rotinas
{
  public:
    double  *v;       // valor dos elementos do vetor
    int  size;     // tamanho do vetor
    void  inicializar(int n, double val=0);  // inicializacao do vetor
    void  destruir(void);    // destruicao do vetor
};

class matrix          //classe de matrizes com suas principais rotinas
{
  public:
    double  **m;     // valor dos elementos da matriz
    int     lin,col;  // dimensoes da matriz
    void  inicializar(int l, int c, double val=0); // inicializacao da matriz
    void  destruir(void);    // destruicao da matriz
};

class screen    // objeto contendo todas as funcoes de tela
{
  public:
    virtual void limpa_tela(void);     // limpa a tela
    virtual void quadro(int X1,int Y1,int X2,int Y2,int C1,int C2);  // desenha quadro
    virtual void centro(char TXT[],int Y,int C1,int C2);  //  escreve no centro de uma linha da tela
    virtual void escrever(char txt[],int x,int y,int c1,int c2);  // escreve um texto dada a posicao
    virtual void tela_fundo(void);    // cria tela de fundo
    virtual void tela_principal(void);// cria tela principal
    virtual void opcao(void);        // cria quadro de opcoes
    virtual void tela_sobregravar(void);  // cria a tela perguntando se ira sobregravar a rede existente
```

```
    virtual void impossivel_abrir(void);  // cria a tela informando que e impossivel abrir
    virtual void erro_leitura(void);   // cria a tela avisando que houve erro de leitura
    virtual void nao_definida(void); // cria a tela avisando que a rede nÆo foi definida
};

class tela_define:public screen // objeto contendo todas as funcoes de tela
{                    // necessarias para a definicao da nova rede
  public:
    void principal(void);  // tela principal
    void entradas(void);  // tela pedindo numero de entradas
    void saidas(void);     // tela pedindo numero de saidas
    void neuronios(void);   // tela pedindo numero de neuronios da camada intermediaria
    void aviso(void);   // tela avisando que a rede foi criada
};

class tela_abrir:public screen    // objeto contendo todas as funcoes de tela
{                    // necessarias para abrir uma rede
  public:
    void principal(void);     // tela principal
    void pede_arquivo(void); // tela pedindo nome do arquivo que contem a rede
    void aviso(void);       // tela avisando que a rede foi lida a partir do arquivo
};

class tela_treino:public screen  // objeto contendo todas as funcoes de tela
{                    // necessarias para treinar a rede
  public:
    void principal(void);          // tela principal
    void pede_dados(void);          // tela que pede o arquivo de dados para treinamento
    void continua(void);          // tela que pergunta se ira continuar com o treinamento GROSSBERG
    void pede_parametros_koh(void);    // tela pedindo parƒmetros para treinamento KOHONEN
    void pede_parametros_gross(void); // tela pedindo parƒmetros para treinamento GROSSBERG
    void em_treinamento(void);       // tela indicando que a rede esta em treinamento
    void resultado_parcial_koh(void); // tela que apresenta resultados parciais do treinamento KOHONEN
    void resultado_parcial_gross(void);  // tela que apresenta resultados parciais do treinamento GROSSBERG
    void fim_treino(void);          // tela indicando fim de treinamento
    void aviso(void);            // tela avisando o fim do treinamento
    void reconhecimento(char &resp);// tela perguntando se e ou nÆo reconhecimento de 2 padroes
};

class tela_execucao:public screen  // objeto contendo todas as funcoes de tela
{                    // necessarias para a execucao da nova rede
  public:
    void principal(int emax);  //  tela principal para execucao por teclado
};

class impressao:public screen     // objeto contendo as funcoes para a impressÆo
{                  // da rede em arquivo
  public:
    void confirma(void); // tela perguntando se confirma a impressÆo em arquivo
    void pesos(void);      // tela para impressÆo dos pesos em arquivo
};

class tela_teste:public screen  // objeto contendo todas as funcoes de tela
{                    // necessarias para o teste da rede
  public:
    void principal(void);     // tela principal
    void pede_arquivo(void); // tela que pede arquivo de dados para teste da rede
    void resultado(void);    // tela indicando resultados do teste realizado
    int   quantas_vezes(void);  // tela pedindo e retornando o no.de vezes se e testado o conjunto de dados
```

```
    void em_teste(void);    // tela indicando que a rede esta em treinamento
};

class tela_exec_arq:public screen  // objeto contendo todas as funcoes de tela
{                       // necessarias para o teste da rede
  public:
    void principal(void);    // tela principal
    void pede_arquivo(void); // tela que pede arquivo de dados para execucao da rede
    void fim_da_execucao(void); // tela indicando fim da execucao
};

class treino          // objeto contendo todas as funcoes e variaveis
{                 // necessarias para o treinamento da rede
  public:
    vector  winners;    // vetor que indica se o neuronio foi ou nÆo vencedor;        // vetor de dados para
treinamento
    FILE *arqdad;      // arquivo com dados para treinamento
    long  t_ini, t_fim,  // instantes iniciais e finais do treinamento
        t_koh, t_gross;  // tempos de treinamento Kohonen e Grossberg
    void  espera_tecla(void);   // funcao que espera uma tecla ser pressionada
    long  pega_tempo(void);   // funcao que pega os instantes iniciais e finais de treinamento
    int    abre_arquivo(char *arqdad);  // funcao que abre o arquivo de dados
    void  fecha_arquivo(void); // funcao que fecha o arquivo de dados para treinamento
    int    le_dados(int emax, int smax, vector &entrada, vector &saida); // funcao que l^ os dados de treinamento
do arquivo
    void  leitura_parametros_koh(double &alfa_ini,double &alfa_fim, // l^ os par fm. de trein. KOHONEN
                double &errod,unsigned long int &n_max);
    void  leitura_parametros_gross(double &beta_ini,double &beta_fim,// l^ os par fm. trein. GROSSBERG
                double &errod,unsigned long int &n_max);
    int    continua(void);      // funcao que pergunta se ira continuar o treinamento GROSSBERG
    void  imprime_resultado_parcial_koh(unsigned long int iter_koh,  // imprime os resultados par-
                double erro_max_koh,double alfa, long tempo);   // ciais do treinamento KOHONEN
    void  imprime_resultado_parcial_gross(unsigned long int iter_gross, //imprime os resultados par-
                double erro_max_gross,double beta, long tempo); // ciais do treinamento GROSSBERG
    void  imprime_arquivo(int emax, int smax, int nmax, matrix weight_gross, matrix weight_koh);
                // imprime as matrizes dos pesos em um arquivo
    void  teste_imprime(int iter, double erro);  // imprime os resultados do teste realizado
};

class net          // Objeto que representa a rede neural.
{                // As variaveis e as principais subrotinas da
                 // rede estÆo incluidas neste objeto
  private:
    int    winner;    // indica o neuronio vencedor
    vectorentrada, // vetor de entradas da rede
        saida;    // vetor de saidas da rede
    matrixweight_gross,  // matriz com os pesos da camada KOHONEN
        weight_koh;   // matriz com os pesos da camada GROSSBERG
    char  reconhec;       // indica se e ou nÆo reconhecimento de 2 padroes com saidas 0 ou 1
  public:
    flag inicializado;    // indica se houve ou nÆo inicializacao da rede neural
    t_pesos altera_pesos;      // indica se vai haver alteracao dos pesos para melhorar o treinamento
    int    pega_tempo(void); // pega o tempo para gerar os randomicos
    int    emax,smax,nmax;     // numero de entradas, saidas e neuronios respectivamente
    void  espera_tecla(void); // espera uma tecla ser pressionada
    int    verifica_inicializacao(void); // Verifica inicializacao da rede
    void  nova_rede(void);// Cria nova rede
    void  abrir_rede(void); // Abre rede ja existente
    int    pede_arq_dados(char *(&arqdad));  // pede arquivo de dados para treinamento e teste
```

```
    void  treinar_rede(void);    // Treina a rede
    void  testar_rede(void);     // Testa a rede
    void  saidas(void);      // Acha as saidas a partir do vetor de entradas e dos pesos
    void  destruir(void);      // Destrói (desaloca) as variaveis utilizadas
    void  executar_teclado(void);  // Calcula a saida dada as entradas via teclado
    double  dist_euclid(int winner, matrix w, vector v, tipo treino);  // Calcula a distancia euclidiana
    void  acha_winner(void);// Acha o neuronio vencedor e retorna erro
    void  ajusta_kohonen(double alfa);  // Ajusta pesos da rede Kohonen
    void  ajusta_grossberg(double beta);  // Ajusta pesos da rede Grossberg */
    void  tratar_pesos(FILE *arqdad);     // trata os pesos para melhorar o treinamento
    void  corrige_pesos(vector winners, FILE *arqdad);  // corrige os pesos para melhorar o treinamento
    void  treino_kohonen(double alfa_ini,double alfa_fim,double erro_max,  // treinamento KOHONEN
                double &erro,int n_max,char *nome_arqdad);
    void  treino_grossberg(double beta_ini,double beta_fim,double erro_max,  // treinamento GROSSBERG
                double &erro,int n_max, double alfa, char *nome_arqdad);
    void  executar_arquivo(void);      // acha as saidas a partir de entradas dadas em um arquivo
};


/* DEFINICAO DAS FUNCOES MEMBRO DE CADA CLASSE */

void vector::inicializar(int n, double val)
/* Inicializa um vetor atribuindo a seus elementos um valor inicial */
{
  int i;

  size = n;
  v = new double [size];
  for(i=0;i<size;i++)
    v[i]=val;
}

void vector::destruir(void)
/* Destroi (desaloca) o vetor existente */
{
  delete v;
  v = NULL;
}

void matrix::inicializar(int l, int c, double val)
/* inicializa matriz atribuindo a seus elementos um valor inicial */
{
  int i,j;

  lin = l;
  col = c;
  m = new double *[lin];
  for(i=0;i<lin;i++)
  {
    m[i] = new double [col];
    for(j=0;j<col;j++)
      m[i][j]=val;
  }
}

void matrix::destruir(void)
/* destroi (desaloca) matrix existente */
{
  int i;
```

```
  for(i=0;i<lin;i++)
    delete m[i];
  delete m;
  m=NULL;
}

void screen::limpa_tela(void)
/*Limpa a tela */
{
  clrscr();
}

void screen::quadro(int X1,int Y1,int X2,int Y2,int C1,int C2)
/*Funcao de tela que desenha um quadro dados os extremos de sua diagonal
  e as cores do texto e do fundo                              */
{
  char T[SL];
  int F;

  for(F=0;F<SL;F++)
    T[F]=0;
  textbackground(C2);
  textcolor(C1);
  T[0]=201;
  for(F=X1+1;F<X2;F++)
    T[F-X1]=205;
  T[X2-X1]=187;
  gotoxy(X1,Y1);
  cprintf(T);
  T[0]=186;
  for(F=X1+1;F<X2;F++)
    T[F-X1]=32;
  T[X2-X1]=186;
  for(F=Y1+1;F<Y2;F++)
  {
    gotoxy(X1,F);
    cprintf(T);
  }
  T[0]=200;
  for(F=X1+1;F<X2;F++)
    T[F-X1]=205;
  T[X2-X1]=188;
  gotoxy(X1,Y2);
  cprintf(T);
  textcolor(7);
  textbackground(0);
  for(F=X1;F<=X2;F++)
    T[F-X1]=176;
  gotoxy(X1+1,Y2+1);
  cprintf(T);
  T[0]=176;
  T[1]=0;
  for(F=Y1+1;F<=Y2;F++)
  {
    gotoxy(X2+1,F);
    cprintf(T);
  }
  textbackground(0);
```

```
  textcolor(15);
}

void screen::centro(char TXT[],int Y,int C1,int C2)
/*funcao que centraliza um texto dado a linha e as cores do texto e de fundo */
{
  int X;

  X=(80-strlen(TXT))/2;
  textcolor(C1);
  textbackground(C2);
  gotoxy(X,Y);
  cprintf(TXT);
  textcolor(15);
  textbackground(0);

}

void screen::escrever(char txt[],int x,int y,int c1,int c2)
/*escreve um texto na posicao (x,y) dado as cores de texto e de fundo */
{
  textcolor(c1);
  textbackground(c2);
  gotoxy(x,y);
  cprintf(txt);
  textcolor(15);
  textbackground(0);
}

void screen::tela_fundo(void)
/*Cria a tela de fundo deste programa */
{
  char TXT[SL];
  int F;

  limpa_tela();

  textbackground(0);
  textcolor(15);
  for (F=0;F<80;F++)
    TXT[F]=32;
  TXT[80]=0;
  cprintf(TXT);
  sprintf(TXT,"REDES NEURAIS-1998 = COUNTERPROPAGATION NEURAL NETWORK");
  centro(TXT,1,15,0);
  gotoxy(1,2);
  textcolor(7);
  for (F=0;F<80;F++)
    TXT[F]=178;
  TXT[80]=0;
  for (F=0;F<23;F++)
    cprintf(TXT);
}

void screen::tela_principal(void)
/*Cria a tela principal com o seu menu */
{
  tela_fundo();
  quadro(14,4,66,15,14,1);
```

```
   centro("MENU PRINCIPAL",5,15,1);
   escrever("1.DEFINIR NOVA REDE NEURAL",17,7,15,1);
   escrever("2.ABRIR REDE NEURAL",17,8,15,1);
   escrever("3.TREINAR REDE NEURAL EXISTENTE",17,9,15,1);
   escrever("4.TESTAR A REDE NEURAL",17,10,15,1);
   escrever("5.EXECUTAR REDE COM DADOS DO TECLADO",17,11,15,1);
   escrever("6.EXECUTAR REDE COM ARQUIVO",17,12,15,1);
   escrever("7.SAIR",17,13,15,1);
   quadro(25,20,55,22,14,1);
   escrever("Digite a sua opcao: ",27,21,15,1);
}

void net::espera_tecla(void)
/*Cria a tela esperando que uma tecla seja pressionada  */
{
   screenscr;

   scr.quadro(14,21,66,23,14,4);
   scr.escrever("Tecle algo para prosseguir: ",23,22,15,4);
   getch();
}

void screen::opcao(void)
/*Cria o quadro onde se deve digitar a opcao desejada  */
{
   quadro(30,18,50,20,14,1);
   escrever("Opcao:  ",34,19,15,1);
}

void screen::tela_sobregravar(void)
/*Cria o quadro que pergunta se ira sobregravar os dados ja existentes */
{
   quadro(14,12,66,14,14,4);
   escrever("Dados ja existentes!!! Sobregravar (s/n)? ",17,13,15,4);
}

void screen::impossivel_abrir(void)
/*  Apresenta um aviso de erro     */
{
   quadro(10,12,70,14,14,4);
   escrever("ERRO: NÇO FOI POSSÖVEL CRIAR OS ARQUIVOS ESPECIFICADOS!!!",12,13,15,4);
}

void screen::erro_leitura(void)
/*Apresenta um aviso para um erro de leitura  */
{
   quadro(10,12,70,14,14,4);
   escrever("ERRO NA LEITURA DOS ARQUIVOS ESPECIFICADOS!!!",14,13,15,4);
}

void screen::nao_definida(void)
/*Apresenta o aviso quando a rede neural nao esta definida  */
{
   quadro(10,12,69,14,14,4);
   escrever("ERRO: REDE NEURAL NÇO DEFINIDA!!!",23,13,15,4);
}

void tela_define::principal(void)
/*Cria a tela principal na definicao de uma nova rede neural  */
```

```
{
  tela_fundo();
  quadro(12,4,66,8,14,1);
  escrever("O programa cria uma rede neural counterpropagation",14,5,15,1);
  escrever("Esta rede sera definida pelos arquivos de extensoes",14,6,15,1);
  escrever("*.wgr, *.wkh e *.sz",16,7,15,1);
}

void tela_define::entradas(void)
/* Cria o quadro pedindo as entradas  */
{
  quadro(14,12,66,14,14,1);
  escrever("Numero de entradas:  ",17,13,15,1);
}

void tela_define::saidas(void)
/*Cria o quadro pedindo as saidas   */
{
  quadro(14,12,66,14,14,1);
  escrever("Numero de saidas:  ",17,13,15,1);
}

void tela_define::neuronios(void)
/*Cria o quadro pedindo o numero de neuronios da rede  */
{
  quadro(14,12,66,14,14,1);
  escrever("Numero de neuronios:  ",17,13,15,1);
}

void tela_define::aviso(void)
/*Apresenta um aviso dizendo que os arquivos foram criados  */
{
  quadro(9,12,71,14,14,4);
  escrever("Rede definida. Tecle algo para voltar ao menu principal",11,13,15,4);
}

void tela_abrir::principal(void)
/*Cria a tela principal na abertura de uma rede neural ja existente  */
{
  tela_fundo();
  quadro(12,4,68,9,14,1);
  escrever("   O PROGRAMA ABRE UMA REDE NEURAL Jµ EXISTENTE",14,5,15,1);
  escrever("ExtensÆo: *.wgr - pesos da camada KOHONEN",14,6,15,1);
  escrever("ExtensÆo: *.wkh - pesos da camada GROSSBERG",14,7,15,1);
  escrever("ExtensÆo: *.sz  - no. de entradas, saidas e neuronios",14,8,15,1);
}

void tela_abrir::pede_arquivo(void)
/*Cria o quadro onde se pede o nome do arquivo da rede */
{
  quadro(14,12,66,14,14,1);
  escrever("Rede a ser aberta (sem extensÆo):  ",17,13,15,1);
}

void tela_abrir::aviso(void)
/*Apresenta o aviso de que os dados foram coletados */
{
  quadro(24,16,56,18,14,1);
  escrever("Dados Coletados",32,17,15,1);
```

```
}

void tela_treino::principal(void)
/*Cria a tela principal no treinamento da rede neural  */
{
  tela_fundo();
  quadro(12,4,68,7,14,1);
  escrever("TREINAMENTO DE UMA REDE NEURAL COUNTERPROPAGATION",15,5,15,1);
  escrever(" O arquivo de dados deve conter a extensÆo *.trn",15,6,15,1);
}

void tela_treino::pede_dados(void)
/*Cria um quadro pedindo o arquivo de dados  */
{
  quadro(14,12,66,14,14,1);
  escrever("Arquivo de dados (sem extensÆo):  ",17,13,15,1);
}

void tela_treino::continua(void)
/*Cria tela perguntando se continua com o treinamento Grossberg */
{
  quadro(10,20,70,22,14,4);
  escrever("Continuar com treinamento Grossberg? (S/N): ",18,21,14,4);
}

void tela_treino::pede_parametros_koh(void)
/*Cria tela para a coleta dos parametros de treinamento Kohonen  */
{
  tela_fundo();
  quadro(16,4,64,12,14,1);
  escrever(" - TREINAMENTO KOHONEN DE UMA REDE NEURAL - ",18,5,14,1);
  escrever("          Alfa inicial: ",22,7,14,1);
  escrever("            Alfa final: ",22,8,14,1);
  escrever("Erro maximo em cada camada: ",22,9,14,1);
  escrever("Numero maximo de iteracoes: ",22,10,14,1);
}

void tela_treino::pede_parametros_gross(void)
/*Cria tela para a coleta dos parametros de treinamento Grossberg */
{
  tela_fundo();
  quadro(16,4,64,12,14,1);
  escrever(" - TREINAMENTO GROSSBERG DE UMA REDE NEURAL - ",18,5,14,1);
  escrever("          Beta inicial: ",22,7,14,1);
  escrever("            Beta final: ",22,8,14,1);
  escrever("Erro maximo em cada camada: ",22,9,14,1);
  escrever("Numero maximo de iteracoes: ",22,10,14,1);
}

void tela_treino::em_treinamento(void)
/*Cria o aviso durante o treinamento  */
{
  quadro(14,21,66,23,14,2);
  escrever("EM TREINAMENTO... Pressione P para interromper.",16,22,143,2);
}

void tela_treino::resultado_parcial_koh(void)
/*Apresenta os resultados parciais do treinamento Kohonen */
{
```

```
  tela_fundo();
  quadro(10,4,70,12,14,1);
  escrever(" - TREINAMENTO KOHONEN DE UMA REDE NEURAL - ",18,5,14,1);
  escrever("        Numero de iteracoes: ",12,7,14,1);
  escrever("  Erro maximo (dist. euclid.): ",12,8,14,1);
  escrever("                       Alfa: ",12,9,14,1);
  escrever("        Tempo de treinamento: ",12,10,14,1);
}

void tela_treino::resultado_parcial_gross(void)
/* Apresenta os resultados parciais do treinamento   Grossberg */
{
  tela_fundo();
  quadro(10,4,70,12,14,1);
  escrever(" - TREINAMENTO GROSSBERG DE UMA REDE NEURAL - ",16,5,14,1);
  escrever("        Numero de iteracoes: ",12,7,14,1);
  escrever("  Erro maximo (dist. euclid.): ",12,8,14,1);
  escrever("                       Beta: ",12,9,14,1);
  escrever("        Tempo de treinamento: ",12,10,14,1);
}


void tela_treino::fim_treino(void)
/* Apresenta o resultado final do treinamento */
{
  quadro(10,16,70,18,14,4);
  escrever(" - FIM DO TREINAMENTO - ",28,17,14,4);
}

void tela_treino::aviso(void)
/* Apresenta um aviso apos a criacao dos arquivos  */
{
  quadro(9,12,71,14,14,4);
  escrever("Arquivos Criados. Tecle algo para voltar ao menu principal",11,13,15,4);
}

void tela_treino::reconhecimento(char &resp)
/* Verifica se e reconhecimento de 2 padroes cujas saidas sÆo 0 ou 1*/
{
  quadro(8,12,72,14,14,1);
  escrever("Rede neural com 2 neuronios com saidas 0 ou 1 ?  (S/N):  ",11,13,15,1);
  do
  {
    gotoxy(66,13);
    scanf("%c",&resp);
  }
  while((resp!='n')&&(resp!='N')&&(resp!='s')&&(resp!='S'));
}

void tela_execucao::principal(int emax)
/* Cria a tela principal na execucao(calculo) na rede neural */
{
  char *conv;
  conv = new char [20];

  tela_fundo();
  quadro(14,3,66,6,14,1);
  escrever(" - EXECU□ÇO DA REDE NEURAL - ",26,4,14,1);
  sprintf(conv,"Numero de entradas: %d",emax);
```

```
   escrever(conv,30,5,15,1);
   quadro(8,8,38,19,14,1);
   escrever("- ENTRADAS -",17,9,14,1);
   quadro(40,8,71,19,14,1);
   escrever("- SAÖDAS -",50,9,14,1);
   delete conv;
}

void tela_teste::principal(void)
/*Cria a tela principal no teste da rede neural */
{
   tela_fundo();
   quadro(14,3,66,5,14,1);
   escrever(" - TESTE DA REDE NEURAL - ",27,4,14,1);
}

void tela_teste::pede_arquivo(void)
/*Cria um quadro pedindo o arquivo de teste */
{
   quadro(14,8,66,10,14,1);
   escrever("Arquivo de teste (sem extensÆo):  ",17,9,15,1);
}

void tela_teste::resultado(void)
/*Apresenta a tela de resultados do teste */
{
   quadro(10,12,70,17,14,4);
   escrever(" - RESULTADO DO TESTE - ",28,13,14,4);
   escrever("          Numero de dados utilizados: ",14,15,15,4);
   escrever("Erro quadratico por conj. de entradas: ",14,16,15,4);
}

int tela_teste::quantas_vezes(void)
/*Cria um quadro e retorna quantas vezes se e testado o conjunto de dados */
{
   int n;

   quadro(12,12,68,14,14,1);
   escrever("Quantas vezes deseja testar o conjunto de dados? ",15,13,15,1);
   do
   {
     gotoxy(65,13);
     scanf("%d",&n);
   }
   while(n<1);

   return n;
}

void tela_teste::em_teste(void)
/*Cria o aviso durante o teste */
{
   quadro(25,21,55,23,14,2);
   escrever("EM TESTE!!!  AGUARDE...",29,22,143,2);
}

void tela_exec_arq::pede_arquivo(void)
/*Cria um quadro pedindo o arquivo de execucao   */
{
```

```
   quadro(14,8,66,12,14,1);
   escrever("Obtencao das saidas a partir de um arquivo *.ent",17,9,15,1);
   escrever("Arquivo de entradas (sem extensÆo):  ",17,11,15,1);
}

void tela_exec_arq::fim_da_execucao(void)
/* Apresenta o fim da execucao  */
{
   quadro(10,16,70,18,14,4);
   escrever(" - FIM DA EXECU□ÇO - ",28,17,14,4);
}

void tela_exec_arq::principal(void)
/* Cria a tela principal na execucao da rede neural com um arquivo de entradas  */
{
   tela_fundo();
   quadro(14,3,66,5,14,1);
   escrever(" - EXECU□ÇO DA REDE NEURAL - ",27,4,14,1);
}

void impressao::confirma(void)
/* Cria o quadro onde voce diz se quer ou nao imprimir em arquivo*/
{
   tela_fundo();
   quadro(14,12,66,14,14,1);
   escrever("Deseja guardar a rede em um arquivo (s/n)? ",17,13,15,1);
}

void impressao::pesos(void)
/*  Cria a tela de para a impressao dos pesos em arquivo  */
{
   tela_fundo();
   quadro(14,12,66,14,14,1);
   escrever("Entre nome para o arquivo da rede: ",17,13,15,1);
   quadro(14,6,66,9,14,1);
   escrever("IMPRESSÇO DOS PESOS DA REDE EM UM ARQUIVO",20,7,15,1);
   escrever("Os arquivos terÆo extensoes: .wgr, .wkh e .sz",16,8,15,1);
   quadro(14,12,66,14,14,1);
   escrever("Entre nome para o arquivo da rede: ",17,13,15,1);
}

int net::pega_tempo(void)
/* Pega o tempo para a utilizacao dos randomicos */
{
   struct dostime_t t;

   _dos_gettime(&t);
   return t.hsecond;
}

int net::verifica_inicializacao(void)
/* Verifica a inicializacao das variaveis a serem utilizadas  */
{
   char c;
   screen scr;

   if(inicializado==sim)
   {
      do
```

```
    {
      scr.tela_sobregravar();     // pergunta se quer sobregravar a rede ja existente
      c=getchar();
      if( c == 'N' || c == 'n')
        goto end;  // em caso negativo, vai para o fim e retorna 0
      else
        if( c == 'S' || c == 's')
        {
          weight_gross.destruir();  // em caso afirmativo, destr¢i-se as matrizes
          weight_koh.destruir();    // e vetores alocados para a nova inicializacao
          saida.destruir();         // com as funcoes nova_rede e abrir_rede
          entrada.destruir();
          return 1;
        }
    }while( c != 'N' && c != 'n' && c != 'S' && c != 's' );
  }
  else
    return 1;
  end:
  return 0;
}

void net::nova_rede(void)
/*Cria uma nova rede  */
{
  int i,j;
  tela_define tela;

  tela.principal();

  if( !(verifica_inicializacao()) ) // se a rede nÆo estiver inicializada, vai para o final
    goto end;

  tela.entradas();// tela pedindo o numero de entradas
  scanf("%d",&emax);
  tela.saidas();   // tela pedindo o numero de saidas
  scanf("%d",&smax);
  tela.neuronios(); // tela pedindo o numero de neuronios
  scanf("%d",&nmax);

  srand(pega_tempo());
  weight_gross.inicializar(smax,nmax);  // inicializa matriz de pesos Grossberg
  for(i=0;i<smax;i++)                   // com valores aleat¢rios
  {
    for(j=0;j<nmax;j++)
    {
      weight_gross.m[i][j]= random(100);
      weight_gross.m[i][j] /=100;
    }
  }

  srand(pega_tempo());

  weight_koh.inicializar(nmax,emax); // inicializa matriz de pesos Kohonen
  for(i=0;i<nmax;i++)                // com valores aleat¢rios
    for(j=0;j<emax;j++)
      weight_koh.m[i][j]=(1.0/sqrt(fabs(nmax)));

  inicializado = sim;
```

```
  saida.inicializar(smax,0.7);
  entrada.inicializar(emax,0.6);

  winner=0;
  altera_pesos = altera;// indica que os pesos foram ditribuidos aleatoriamente, sem previo tratamento,
                        // o que faz com que os pesos necessitem de alteracoes para melhorar o treinamento
  end:
}

void net::abrir_rede(void)
/* L^ (abre) uma rede de um dado arquivo */
{
  int  i,j;
  double  norma;
  char  c, *nome1=NULL, *nome2=NULL, *nome3=NULL,
       *nomearq=NULL, *s=NULL, palavra[12];
  FILE  *arquivo1=NULL,*arquivo2=NULL,*arquivo3=NULL;
  tela_abrir tela;

  tela.principal();

  if( !(verifica_inicializacao()) ) // se a rede nÆo estiver inicializada, vai para o final
    goto end;

  nomearq = new char [15];
  nome1 = new char [15];
  nome2 = new char [15];
  nome3 = new char [15];
  s = new char [15];

  tela.pede_arquivo();  // tela pedindo o nome do arquivo que contem a rede neural
  scanf("%s",nomearq);
  strcpy(nome1,nomearq);
  strcpy(nome2,nomearq);
  strcpy(nome3,nomearq);
  strcat(nome1,".wgr");
  strcat(nome2,".wkh");
  strcat(nome3,".sz");

  if ( ((arquivo1=fopen(nome1,"rt"))==NULL)||((arquivo2=fopen(nome2,"rt"))==NULL)
    ||((arquivo3=fopen(nome3,"rt"))==NULL) )
  {
    tela.impossivel_abrir();
    inicializado = nao;
    goto erro;
  }

  while( !feof(arquivo3) )// leitura do numero de entradas, saidas e neuronios
  {
    if ( !fscanf(arquivo3, "%d %d %d", &emax, &smax, &nmax) )
    {
      tela.erro_leitura();
      goto erro;
    }
  }
  fclose( arquivo3 );

  weight_gross.inicializar(smax,nmax);
```

```
while( !feof(arquivo1) )// leitura dos pesos da camada Grossberg
{
  for(i=0;i<smax;i++)
    for(j=0;j<nmax;j++)
      if( !fscanf(arquivo1, "%lf", &weight_gross.m[i][j]) )
      {
        tela.erro_leitura();
        goto erro;
      }
}
fclose( arquivo1 );

weight_koh.inicializar(nmax,emax);

while( !feof(arquivo2) )// leitura dos pesos da camada Kohonen
{
  for(i=0;i<nmax;i++)
    for(j=0;j<emax;j++)
      if( !fscanf(arquivo2, "%lf", &weight_koh.m[i][j]) )
      {
        tela.erro_leitura();
        goto erro;
      }
}
fclose( arquivo2 );

for(i=0;i<nmax;i++)
{
  norma = 0.0;
  for(j=0;j<emax;j++)
    norma+=weight_koh.m[i][j]*weight_koh.m[i][j];
  norma=sqrt(fabs(norma));
  for(j=0;j<emax;j++)
    weight_koh.m[i][j]/=norma;
}

saida.inicializar(smax,0.3);
entrada.inicializar(emax,(1.0/sqrt(nmax)));
winner = 0;

inicializado = sim;

delete nomearq;
delete nome1;
delete nome2;
delete nome3;
nomearq = NULL;
nome1 = NULL;
nome2 = NULL;
nome3 = NULL;
delete s;
s = NULL;
tela.aviso();

erro:

fclose( arquivo1 );
fclose( arquivo2 );
```

```
    fclose( arquivo3 );
    delete nomearq;
    delete nome1;
    delete nome2;
    delete nome3;
    nomearq = NULL;
    nome1 = NULL;
    nome2 = NULL;
    nome3 = NULL;
    delete s;
    s = NULL;
    espera_tecla();

    end:
}

int net::pede_arq_dados(char *(&arqdad))
/*Funcao que faz a leitura dos dados para treinamento   e teste   */
{
    tela_treinotela;
    FILE *f_arqdad;

    tela.pede_dados();        // pede arquivo de dados para o treinamento

    arqdad = new char [20];
    scanf("%s",arqdad);
    strcat(arqdad,".trn");    // arquivo de extensÆo .trn para dados de treinamento

    if( (f_arqdad=fopen(arqdad,"rt"))==NULL)  // verifica se este arquivo existe. Em
    {                         // caso negativo, retorna o valor zero.
      tela.impossivel_abrir();
      espera_tecla();
      fclose(f_arqdad);
      return 0;
    }
    else
    {
      fclose(f_arqdad);
      return 1;
    }
}

void treino::espera_tecla(void)
/*Tela esperando que uma tecla seja pressionada  */
{
    screenscr;

    scr.quadro(20,21,60,23,14,4);
    scr.escrever("Tecle algo para prosseguir: ",23,22,15,4);
    getch();
}

long treino::pega_tempo(void)
/*Pega os instantes iniciais e finais do treinamento   */
{
    long int tempo;
    struct time t;

    gettime(&t);
```

```
  tempo = (int)t.ti_hund;
  tempo += 100*(int)t.ti_sec;
  tempo += (long)6000*(long)t.ti_min;
  tempo += (long)360000*(long)t.ti_hour;
  return tempo;
}

int treino::abre_arquivo(char *nome_arqdad)
/*Abre arquivo de dados para treinamento e retorna o numero de dados para treinamento  */
{
  int  i, j, n_dados;
  tela_treinotela;

  if( (arqdad=fopen(nome_arqdad,"rt"))==NULL)     // abre arquivo
  {
    tela.impossivel_abrir();
    espera_tecla();
    fclose(arqdad);
    return 0;
  }
  else
  {
    if(!fscanf(arqdad,"%d",&n_dados))  // l^ numero de dados
    {
      tela.erro_leitura();
      fclose(arqdad);
      espera_tecla();
      return 0;
    }
    else
    {
      rewind(arqdad);
      return n_dados;
    }
  }
}

void treino::fecha_arquivo(void)
/*Fecha arquivo de dados para treinamento  */
{
  fclose(arqdad);
}


int treino::le_dados(int emax, int smax, vector &entradas, vector &saidas)
/*Realiza a leitura dos dados para treinamento e retorna zero caso isso nÆo seja possivel  */
{
  int  j;
  double norma=0.0;
  tela_treinotela;

  for(j=0;j<emax;j++)
  {
    if(!fscanf(arqdad,"%lf",&entradas.v[j]))
    {
      tela.erro_leitura();
      fclose(arqdad);
      return 0;
    }
```

```
      norma += entradas.v[j]*entradas.v[j];
    }
    norma=sqrt(fabs(norma));
    for(j=0;j<emax;j++)
      entradas.v[j] /= norma;

    for(j=0;j<smax;j++)
      if(!fscanf(arqdad,"%lf",&saidas.v[j]))
      {
        tela.erro_leitura();
        fclose(arqdad);
        return 0;
      }
    return 1;
}

void treino::leitura_parametros_koh(double &alfa_ini,double &alfa_fim,
                    double &errod,unsigned long int &n_max)
/*Realiza a leitura dos parametros do treino Kohonen */
{
    tela_treinotela;

    tela.pede_parametros_koh();     // tela pedindo os parametros de treinamento Kohonen

    do
    {
      gotoxy(50,7);
      scanf("%lf",&alfa_ini);    // leitura do coeficiente de aprendizagem inicial
    }
    while( (alfa_ini>1) || (alfa_ini<0) );   // alfa entre 0 e 1

    do
    {
      gotoxy(50,8);
      scanf("%lf",&alfa_fim);   // leitura do coeficiente de aprendizagem final
    }
    while( (alfa_fim>1) || (alfa_fim<0) );  // alfa entre 0 e 1

    do
    {
      gotoxy(50,9);
      scanf("%lf",&errod); // leitura do erro maximo desejado
    }
    while(errod<0);           // apenas valores positivos para erro

    do
    {
      gotoxy(50,10);
      scanf("%ld",&n_max);    // leitura do numero maximo de iteracoes
    }
    while(n_max<1);
}

void treino::leitura_parametros_gross(double &beta_ini,double &beta_fim,
                    double &errod,unsigned long int &n_max)
/*Realiza a leitura dos parametros do treino grossberg   */
{
    tela_treinotela;
```

```
   tela.pede_parametros_gross();   // tela pedindo os parametros de treinamento Grossberg

   do
   {
     gotoxy(50,7);
     scanf("%lf",&beta_ini);    // leitura do coeficiente de aprendizagem inicial
   }
   while( (beta_ini>1) || (beta_ini<0) );

   do
   {
     gotoxy(50,8);
     scanf("%lf",&beta_fim);      // leitura do coeficiente de aprendizagem final
   }
   while( (beta_fim>1) || (beta_fim<0) );

   do
   {
     gotoxy(50,9);
     scanf("%lf",&errod); // leitura do erro maximo desejado
   }
   while(errod<0);            // apenas valores positivos para erro

   do
   {
     gotoxy(50,10);
     scanf("%ld",&n_max);     // leitura do numero maximo de iteracoes
   }
   while(n_max<1);
}

int treino::continua(void)
/*Verifica se continua com o treinamento Grossberg  */
{
   char resposta;
   tela_treinotela;

   tela.continua();      // pergunta se continua treino grossberg

   do
   {
     gotoxy(63,21);
     scanf("%c",&resposta);
   }
   while( (resposta!='s') && (resposta!='n') && (resposta!='S') && (resposta!='N'));

   if(resposta=='s' || resposta=='S')
     return 1;
   else
     return 0;
}


void treino::imprime_resultado_parcial_koh(unsigned long int iter_koh,double erro_max_koh,
                        double alfa, long tempo)
/*Imprime resultados parciais para o treinamento Kohonen  */
{
   int horas=0, minutos=0, segundos=0, centesimos=0;
   char *conv;
```

```
    screenscr;

    conv = new char [20];

    centesimos = tempo%100;      // Tratamento do tempo de treinamento
    tempo /= 100;
    segundos = tempo%60;
    tempo /=60;
    minutos = tempo %60;
    tempo /=60;
    horas = tempo%24;

    sprintf(conv,"  %ld ",iter_koh);
    scr.escrever(conv,42,7,14,1);
    sprintf(conv,"%16.12lf ",erro_max_koh);
    scr.escrever(conv,42,8,14,1);
    sprintf(conv,"%16.12lf ",alfa);
    scr.escrever(conv,42,9,14,1);
    sprintf(conv," %2d h %2d min %2d.%2d s",horas,minutos,segundos,centesimos);
    scr.escrever(conv,42,10,14,1);
    delete conv;
}

void treino::imprime_resultado_parcial_gross(unsigned long int iter_gross,double erro_max_gross,
                                double beta, long tempo)
/* Imprime resultados parciais para o treinamento Grossberg */
{
    int horas=0, minutos=0, segundos=0, centesimos=0;
    char *conv;
    screenscr;

    conv = new char [20];

    centesimos = tempo%100;      // Tratamento do tempo de treinamento
    tempo /= 100;
    segundos = tempo%60;
    tempo /=60;
    minutos = tempo %60;
    tempo /=60;
    horas = tempo%24;

    sprintf(conv,"  %ld ",iter_gross);
    scr.escrever(conv,42,7,14,1);
    sprintf(conv,"%16.12lf ",erro_max_gross);
    scr.escrever(conv,42,8,14,1);
    sprintf(conv,"%16.12lf ",beta);
    scr.escrever(conv,42,9,14,1);
    sprintf(conv," %2d h %2d min %2d.%2d s",horas,minutos,segundos,centesimos);
    scr.escrever(conv,42,10,14,1);

    delete conv;

}


void treino::imprime_arquivo(int emax, int smax, int nmax, matrix weight_gross, matrix weight_koh)
/*  Imprime nos arquivos (.wgr, .wkh, .sz) a rede neural */
{
    int i,j;
```

```
char c, *nome1=NULL, *nome2=NULL, *nome3=NULL, *nomearq=NULL;
FILE *arquivo1 = NULL,*arquivo2 = NULL,*arquivo3 = NULL;
impressao impr;
tela_treinotela;

do
{
  impr.confirma(); // confirma a impressÆo
  c=getchar();
  if( c == 'N' || c == 'n')
    goto end;  // em caso negativo vai para o final da funcao
}while( c != 'N' && c != 'n' && c != 'S' && c != 's' );

nomearq = new char [15];
nome1 = new char [15];
nome2 = new char [15];
nome3 = new char [15];

impr.pesos();
scanf("%s",nomearq);
strcpy(nome1,nomearq);
strcat(nome1,".wgr");
strcpy(nome2,nomearq);
strcat(nome2,".wkh");
strcpy(nome3,nomearq);
strcat(nome3,".sz");

if ( ((arquivo1=fopen(nome1,"wt"))==NULL)||((arquivo2=fopen(nome2,"wt"))==NULL)
  ||((arquivo3=fopen(nome3,"wt"))==NULL) )
{
  tela.impossivel_abrir();
}
else
{
  fprintf(arquivo3,"\n%d",emax);  // impressÆo das dimensoes da rede neural
  fprintf(arquivo3,"\n%d",smax);
  fprintf(arquivo3,"\n%d",nmax);
  fclose(arquivo3);

  for(i=0;i<smax;i++)        // impressÆo dos pesos da camada Grossberg
    for(j=0;j<nmax;j++)
    {
      fprintf(arquivo1," %10.6lf",weight_gross.m[i][j]);
      if(j==(nmax-1))
        fprintf(arquivo1,"\n");
    }
  fclose(arquivo1);

  for(i=0;i<nmax;i++)        // impressÆo dos pesos da camada Kohonen
    for(j=0;j<emax;j++)
    {
      fprintf(arquivo2," %10.6lf",weight_koh.m[i][j]);
      if(j==(emax-1))
        fprintf(arquivo2,"\n");
    }

  fclose(arquivo2);
  tela.aviso();
}
```

```
    delete(nomearq);
    delete(nome1);
    delete(nome2);
    delete(nome3);

    end:
}

void treino::teste_imprime(int iter, double erro)
/*Imprime os resultados do teste realizado */
{
    char *conv;
    screen scr;
    tela_teste  tela;

    conv = new char [20];

    tela.resultado();     // Tela onde os resultados do teste sao impressos
    sprintf(conv,"  %d ",iter);
    scr.escrever(conv,52,15,14,4);
    sprintf(conv,"  %12.8lf ",erro);
    scr.escrever(conv,52,16,14,4);
    delete conv;
}

double net::dist_euclid(int winner, matrix w, vector v, tipo treino)
/* Calcula a distancia euclidiana entre a i-esima linha da matriz w e o vetor v  */
{
    int i;
    double dist=0.0;
    if(treino==KOHONEN)
      for(i=0;i<w.col;i++)
        dist+=pow((w.m[winner][i]-v.v[i]),2);
    else
      for(i=0;i<w.lin;i++)
        dist+=pow((w.m[i][winner]-v.v[i]),2);
    return dist;
}

void net::acha_winner(void)
/*Acha o neuronio vencedor e a distância euclidiana (erro)  */
{
    int i;
    double aux=0.0, min_dist=0.0;

    winner=0;

    aux = dist_euclid(0,weight_koh,entrada,KOHONEN);
    min_dist = aux;
    for(i=1;i<weight_koh.lin;i++)
    {
      aux = dist_euclid(i,weight_koh,entrada,KOHONEN);
      if(aux<min_dist)
      {
        min_dist = aux;
        winner = i;
      }
    }
}
```

```
void net::ajusta_kohonen(double alfa)
/* Ajusta pesos da camada Kohonen */
{
  int i;
  double norma;

  for(i=0;i<weight_koh.col;i++)
    weight_koh.m[winner][i] += alfa*( entrada.v[i] - weight_koh.m[winner][i] );

  norma = 0.0;
  for(i=0;i<emax;i++)
    norma+=weight_koh.m[winner][i]*weight_koh.m[winner][i];
  norma=sqrt(fabs(norma));
  for(i=0;i<emax;i++)
    weight_koh.m[winner][i]/=norma;
}

void net::ajusta_grossberg(double beta)
/* Ajusta pesos da camada Grossberg  */
{
  int i;
  for(i=0;i<weight_gross.lin;i++)
    weight_gross.m[i][winner] += beta*( saida.v[i] - weight_gross.m[i][winner] );
}

void net::tratar_pesos(FILE *arqdad)
/*   Tratamento dos pesos utilizando randomicos dentro de uma faixa de valores (dados pelo arquivo de dados)
  para uma nova rede (cujos pesos estÆo inicializados aleatoriamente sem faixa de valores)        */
{
  int i, j, quant_dados;
  double min=0.0, max=0.0, aux, norma=0.0;
  tela_treino tela;

  srand(pega_tempo());
  rewind(arqdad);
  if(!fscanf(arqdad,"%d",&quant_dados))
  {
    tela.erro_leitura();
    fclose(arqdad);
    goto erro;
  }
  else
  {
    for(i=1; i<quant_dados; i++)
    {
      for(j=0;j<emax;j++)
      {
        if(!fscanf(arqdad,"%lf",&aux))
        {
          tela.erro_leitura();
          fclose(arqdad);
          goto erro;
        }
        else
        {
          if(aux>max)
            max = aux;
          else
```

```
            if(aux<min)
              min = aux;
          }
        }
        for(j=0;j<smax;j++)
          if(!fscanf(arqdad,"%lf",&aux))
          {
            tela.erro_leitura();
            fclose(arqdad);
            goto erro;
          }
      }
    }

    if(max==min)
      min = max/1.1;

    for(i=0;i<weight_koh.lin;i++)
      for(j=0;j<weight_koh.col;j++)
        weight_koh.m[i][j] = (((double)(rand()%1000))/1000)*(max-min)+min;

    for(i=0;i<nmax;i++)
    {
      norma = 0.0;
      for(j=0;j<emax;j++)
        norma+=weight_koh.m[i][j]*weight_koh.m[i][j];
      norma=sqrt(fabs(norma));
      for(j=0;j<emax;j++)
        weight_koh.m[i][j]/=norma;
    }


    rewind(arqdad);

    erro:
}

void net::corrige_pesos(vector winners, FILE *arqdad)
/*   Correcao dos pesos utilizando randomicos dentro de uma faixa de valores (dados pelo arquivo de dados)
    para melhorar o treinamento. A correcao s¢ e feita com os neuronios que nÆo foram vencedores (estes
    neuronios constituiriam uma redundƒncia da rede caso continuassem nÆo vencedores).          */
{
    int i, j, k, quant_dados, max_winner;
    double aux, norma=0.0;
    tela_treino tela;

    srand(pega_tempo());

    for(i=0;i<winners.size;i++)
      if(winners.v[i]==0)       // verifica qual neuronio nÆo foi vencedor
      {
        max_winner = 0;
        aux = winners.v[0];
        for(j=1;j<winners.size;j++)
          if(winners.v[j]>aux)
          {
            aux = winners.v[j];
            max_winner = j;
          }
```

```
        for(j=0;j<weight_koh.col;j++)
          weight_koh.m[i][j] = weight_koh.m[max_winner][j];
      }

    rewind(arqdad);

    erro:
}

void net::treino_kohonen(double alfa_ini,double alfa_fim,double erro_max,double &erro,
                int n_max,char *nome_arqdad)
/*Realiza o treinamento da camada Kohonen - "winner takes all"*/
{
    const aprf=1;  // numero de iteracoes para exibicao dos resultados parciais
    char key, *conv;
    double  erro_anterior, alfa, fraction;
    unsigned long int iter=0;
    int  i, j, quant_dados, apr;
    treino trn;
    tela_treinotela;

    trn.winners.inicializar(nmax,0);  // inicializacao do vetor que indica se o neuronio e ou nÆo vencedor

    trn.t_ini = trn.pega_tempo();     // pega instante em que se iniciou o treinamento
    trn.t_koh = 0;

    tela.em_treinamento();   // Mensagem: Em treinamento. Pressione P para interromper
    apr = aprf;

    fraction = (double)pow( (alfa_fim/alfa_ini),  1.0/(n_max) ); // fator para decaimento exponencial
                                        // do coeficiente de aprendizagem
    alfa=alfa_ini;
    erro_anterior = 0;
    iter = 0;

    quant_dados = trn.abre_arquivo(nome_arqdad);  // abre arquivo de dados e recebe o
    if( !quant_dados )                            // numero de dados para treinamento
      goto end;

    if(altera_pesos == altera)    // altera pesos quando a rede foi inicializada com valores quaisquer
      tratar_pesos(trn.arqdad); // e trata com faixa de valores contidos no arquivo de dados

    do
    {
      iter++;
      apr--;
      fscanf(trn.arqdad,"%d",&quant_dados);
      for(i=0;i<trn.winners.size;i++)      // inicializa vetor que indica se os neuronios sÆo vencedores
        trn.winners.v[i] = 0;

      for(i=0;i<quant_dados;i++)
      {
        trn.le_dados(emax,smax,entrada,saida);     // leitura de um conjunto de dados de treinamento
        if((reconhec=='n')||(reconhec=='N')) // caso nÆo seja reconhecimento de padroes (caso geral)
        {
          acha_winner();         // obtencao do neuronio vencedor
          trn.winners.v[winner]++;
        }
        else
```

```
        winner = (int)saida.v[0];  // indica qual deve ser o neuronio vencedor (para reconheci-
                                   // mento de padroes forca-se a competicao a partir da saida)
        ajusta_kohonen(alfa);  // ajuste dos pesos da camada Kohonen
      }
      rewind(trn.arqdad);
      fscanf(trn.arqdad,"%d",&quant_dados);

      erro_anterior = erro;
      erro = 0.0;

      for(i=0;i<quant_dados;i++)    // calculo do erro (distância euclidiana media)
      {
        trn.le_dados(emax,smax,entrada,saida);
        acha_winner();
        erro+=dist_euclid(winner,weight_koh,entrada,KOHONEN);
      }
      rewind(trn.arqdad);
      erro/=quant_dados;


      if((reconhec=='n')||(reconhec=='N'))
        if( (iter!=1) )
          corrige_pesos(trn.winners, trn.arqdad);  // correcao dos pesos p/ neuronios nÆo vencedores

      alfa*=fraction; //correcao do coef. de aprend.

      if(!apr)
      {
        apr = aprf;
        trn.t_fim = trn.pega_tempo();
        trn.t_koh = abs(trn.t_fim - trn.t_ini);       // imprime resultados parciais do treinamento
        trn.imprime_resultado_parcial_koh(iter,erro,alfa,trn.t_koh);
      }

      if(kbhit())
        key = getch();

      if( (iter>=n_max) || (erro<erro_max) )
      {
        trn.t_fim = trn.pega_tempo();    // imprime resultados final do treinamento
        trn.t_koh = abs(trn.t_fim - trn.t_ini);
        trn.imprime_resultado_parcial_koh(iter,erro,alfa,trn.t_koh);
        key = 'p';
      }

    }
    while( (key!='p') && (key!='P') );

    trn.fecha_arquivo();
    trn.winners.destruir();

    end:
}

void net::treino_grossberg(double beta_ini,double beta_fim,double erro_max,double &erro,int n_max,
                double alfa, char *nome_arqdad)
/*Realiza o treinamento da camada Grossberg  */
{
    const aprf=1;
```

```
char key, *conv;
double  beta, fraction;
unsigned long int iter=0;
int  i, j, quant_dados, apr;
treino trn;
tela_treinotela;

trn.t_ini = trn.pega_tempo();
trn.t_gross = 0;

tela.em_treinamento();   // Mensagem: Em treinamento. Pressione P para interromper
apr = aprf;
iter = 0;
fraction = (double)pow( (beta_fim/beta_ini),1.0/(n_max) );  // fator para decaimento exponecial
beta=beta_ini;                          // do coeficiente de aprendizagem

quant_dados = trn.abre_arquivo(nome_arqdad);  // abre arquivo de dados
if(!quant_dados)
  goto end;

do
{
  iter++;
  apr--;
  fscanf(trn.arqdad,"%d",&quant_dados);
  for(i=0;i<quant_dados;i++)
  {
    trn.le_dados(emax,smax,entrada,saida);     // leitura de um conjunto de dados de treinamento
    acha_winner();            // obtencao do neuronio vencedor
    ajusta_kohonen(alfa);     // pequeno ajuste dos pesos da camada Kohonen
    ajusta_grossberg(beta);   // ajuste dos pesos da camada Grossberg
  }
  rewind(trn.arqdad);
  fscanf(trn.arqdad,"%d",&quant_dados);

  erro = 0.0;
  for(i=0;i<quant_dados;i++)    // calculo do erro (distância euclidiana media)
  {
    trn.le_dados(emax,smax,entrada,saida);
    acha_winner();
    erro+=dist_euclid(winner,weight_gross,saida,GROSSBERG);
  }
  rewind(trn.arqdad);

  erro/=quant_dados;

  beta*=fraction;         //correcao do coeficiente de aprendizagem

  if(!apr)
  {
    apr = aprf;
    trn.t_fim = trn.pega_tempo();          // imprime resultados parciais do treinamento
    trn.t_gross = abs(trn.t_fim - trn.t_ini);
    trn.imprime_resultado_parcial_gross(iter,erro,beta,trn.t_gross);
  }

  if(kbhit())
    key = getch();
```

```
      if( (iter>n_max) || (erro<erro_max) )
      {
        trn.t_fim = trn.pega_tempo();          // imprime resultados finais do treinamento
        trn.t_gross = abs(trn.t_fim - trn.t_ini);
        trn.imprime_resultado_parcial_gross(iter,erro,beta,trn.t_gross);
        key = 'p';
      }

    }
    while( (key!='p') && (key!='P') );

    trn.fecha_arquivo();

    end:
}

void net::treinar_rede(void)
/*Treina a rede neural  */
{
    char key, *conv, *nome_arqdad;
    double  erro, erro_max=0.0,alfa, beta, alfa_ini, beta_ini, alfa_fim, beta_fim;
    unsigned long int n_max;
    int  i, j, quant_dados, apr;
    treino trn;
    tela_treinotela;

    tela.principal();  //tela inicial

    if( inicializado == nao )
    {
      tela.nao_definida();   // tela: rede nao inicializada
      espera_tecla();
      goto end;
    }

    if(!pede_arq_dados(nome_arqdad))  // tela pedindo arquivo de dados para treinamento
      goto end;

    tela.reconhecimento(reconhec);     // pergunta se e ou nÆo reconhecimento de 2 padroes com saidas 0 ou 1

    trn.leitura_parametros_koh(alfa_ini,alfa_fim,erro_max,n_max);   // leitura dos par*f*metros de treinamento

    tela.resultado_parcial_koh(); // Tela para indicar o numero de iteracoes realizado
                        // o erro obtido, o alfa final e o tempo de treinamento

    treino_kohonen(alfa_ini,alfa_fim,erro_max,erro,n_max, nome_arqdad);// realiza o treinamento Kohonen

    if(trn.continua()) // caso se deseje continuar com o treinamento Grossberg
    {
      reconhec = 's';
      trn.leitura_parametros_gross(beta_ini,beta_fim,erro_max,n_max);//leitura dos par*f*metros de treinamento

      tela.resultado_parcial_gross();     // Tela para indicar o numero de iteracoes realizado
                          // o erro obtido, o alfa final e o tempo de treinamento
      treino_grossberg(beta_ini,beta_fim,erro_max,erro,n_max,alfa_fim,nome_arqdad); //treinamento Grossberg
    }
    tela.fim_treino();
    espera_tecla();
```

```
  trn.imprime_arquivo(emax,smax,nmax,weight_gross,weight_koh);//imprime a rede para o arquivo caso
necessario

  delete nome_arqdad;

  end:
}

void net::testar_rede(void)
/* Subrotina que faz o teste de uma rede neural ja treinada ou nÆo */
{
  int i,j,iter=0,qual_dado, quant_dados,n_vezes;
  double erro=0.0;
  char *nome_arqdad;
  vector saida_desejada;
  tela_teste tela;
  treino trn;

  tela.principal();

  saida_desejada.inicializar(smax);

  if( inicializado == nao )
  {
    tela.nao_definida();   // tela: rede nao inicializada
    espera_tecla();
    goto end;
  }

  if(!pede_arq_dados(nome_arqdad))  // tela pedindo arquivo de dados
    goto end;

  quant_dados = trn.abre_arquivo(nome_arqdad);  // abre arquivo de dados
  if(!quant_dados)
    goto end;

  tela.em_teste();

  iter = 0;
  fscanf(trn.arqdad,"%d",&quant_dados);
  while( iter <= quant_dados )
  {
    iter++;
    trn.le_dados(emax,smax,entrada,saida);
    for(i=0;i<smax;i++)
      saida_desejada.v[i] = saida.v[i];
    saidas();// acho as saidas a partir das entradas
    for(i=0;i<smax;i++)
      erro+=pow(saida_desejada.v[i]-saida.v[i],2);
  }
  rewind(trn.arqdad);

  erro/=quant_dados;

  trn.teste_imprime(iter-1,erro);   // imprime resultado do teste na tela

  espera_tecla();

  saida_desejada.destruir();
```

```
    trn.fecha_arquivo();

    delete nome_arqdad;


    end:

}

void net::destruir(void)
/*Destr¢i as variaveis ainda alocadas na mem¢ria */
{
    weight_gross.destruir();
    weight_koh.destruir();
    saida.destruir();
    entrada.destruir();
}

void net::saidas(void)
/*Acha as saidas a partir das entradas  */
{
    int i,j;

    for(i=0;i<smax;i++)
        saida.v[i] = 0;

    acha_winner();// acha o neuronio vencedor

    for(i=0;i<smax;i++)
        saida.v[i]=weight_gross.m[i][winner]; // calculo das saidas

}

void net::executar_teclado(void)
/*Calcula a saida a ser obtida em uma rede previamente treinada atraves
    do teclado                                          */
{
    int i;
    char *conv;
    tela_execucao tela;
    screen scr;

    tela.principal(emax);

    if( inicializado == nao )
    {
        tela.nao_definida();   // tela: rede nao inicializada
        espera_tecla();
        goto end;
    }

    conv = new char [40];

    for(i=0;i<emax;i++)
    {
        sprintf(conv,"Entrada %2d: ",i);
        scr.escrever(conv,10,11+i,15,1);
        gotoxy(23,11+i);
        scanf("%lf",&entrada.v[i]);
```

```
    }

    saidas();

    for(i=0;i<smax;i++)
    {
      sprintf(conv,"Saida %2d: %11.6lf",i,saida.v[i]);
      scr.escrever(conv,42,11+i,15,1);
    }
    espera_tecla();
    delete conv;

    end:
}

void net::executar_arquivo(void)
/* Calcula as saidas a partir das entradas contidas em um arquivo */
{
    char *arq1, *arq2;
    int  i, j, n_dados;
    double norma=0.0;
    FILE *arq_ent=NULL, *arq_sai=NULL;
    tela_exec_arq  tela;

    tela.principal();

    if( inicializado == nao )
    {
      tela.nao_definida();   // tela: rede nao inicializada
      espera_tecla();
      goto end;
    }

    tela.pede_arquivo();  // pede arquivo de dados para a execucao

    arq1 = new char [20];
    arq2 = new char [20];
    scanf("%s",arq1);
    strcpy(arq2,arq1);
    strcat(arq1,".ent");       // extensÆo *.ent para arq de entradas
    strcat(arq2,".sai");       // extensÆo *.sai para arq de saidas
    if( ((arq_ent=fopen(arq1,"rt"))==NULL) || ((arq_sai=fopen(arq2,"wt"))==NULL) )
    {
      tela.impossivel_abrir();
      espera_tecla();
      goto erro;
    }
    else
    {
      if(!fscanf(arq_ent,"%d",&n_dados))
      {
        tela.erro_leitura();
        goto erro;
      }
      for(i=0;(i<n_dados)&&(!feof(arq_ent));i++)
      {
        norma = 0.0;
        for(j=0;j<emax;j++)
        {
```

```
            if(!fscanf(arq_ent,"%lf",&entrada.v[j]))   // leitura de arquivo
            {
              tela.erro_leitura();
              goto erro;
            }
            norma += entrada.v[j]*entrada.v[j];
          }
          norma = sqrt(fabs(norma));
          for(j=0;j<emax;j++)
            entrada.v[j] /= norma;

          saidas();// calculo das saidas

          for(j=0;j<smax;j++)
            fprintf(arq_sai," %lf ",saida.v[j]);  // impressÆo em arquivo
          fprintf(arq_sai,"\n");
        }

    }
    tela.fim_da_execucao();
    espera_tecla();

    erro:
      fclose(arq_ent);
      fclose(arq_sai);
      delete arq1;
      delete arq2;
    end:
}


/*PROGRAMA PRINCIPAL */

void main (void)
{
    screen scr;
    int i,opcao;
    char c;
    flag para;
    net rede;

    rede.inicializado = nao;

    do
    {
      de_novo:

      scr.tela_principal();
      gotoxy(48,21);
      if ( !scanf("%d",&opcao) ) goto de_novo;
      para = nao;
      if (opcao < 1 || opcao > 7)   goto de_novo;
      else
      {
        if ( opcao < 7 )
        {
          switch (opcao)
          {
            case 1:
```

```
                rede.nova_rede();
                break;
              case 2:
                rede.abrir_rede();
                break;
              case 3:
                rede.treinar_rede();
                break;
              case 4:
                rede.testar_rede();
                break;
              case 5:
                rede.executar_teclado();
                break;
              case 6:
                rede.executar_arquivo();
                break;
            }
          }
          else para = sim;
        }
      } while ( !para );

      if(rede.inicializado==sim)
        rede.destruir();
      scr.limpa_tela();
    }
```

C. LISTAGEM DO PROGRAMA PARA GERAÇÃO DOS ARQUIVOS DE TREINAMENTO PARA O BACKPROPAGATION

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>

#define SIZE 256

enum flag { TREINAMENTO, PESOS, ENTRADAS };
enum resposta { SIM, NAO };
enum inclui { NORMAL, H3 , H5 };

int VAR;

void limpa_tela(void)
/*Limpa a tela */
{
  clrscr();
}

void quadro(int X1,int Y1,int X2,int Y2,int C1,int C2)
/*Funcao de tela que desenha um quadro dados os extremos de sua diagonal
  e as cores do texto e do fundo                                    */
{
  char T[81];
  int F;

  for(F=0;F<81;F++)
    T[F]=0;
  textbackground(C2);
  textcolor(C1);
  T[0]=201;
  for(F=X1+1;F<X2;F++)
    T[F-X1]=205;
  T[X2-X1]=187;
  gotoxy(X1,Y1);
  cprintf(T);
  T[0]=186;
  for(F=X1+1;F<X2;F++)
    T[F-X1]=32;
  T[X2-X1]=186;
  for(F=Y1+1;F<Y2;F++)
  {
    gotoxy(X1,F);
    cprintf(T);
  }
  T[0]=200;
  for(F=X1+1;F<X2;F++)
```

```
      T[F-X1]=205;
    T[X2-X1]=188;
    gotoxy(X1,Y2);
    cprintf(T);
    textcolor(7);
    textbackground(0);
    for(F=X1;F<=X2;F++)
      T[F-X1]=176;
    textbackground(0);
    textcolor(15);
}

void escrever(char txt[],int x,int y,int c1,int c2)
/*escreve um texto na posicao (x,y) dado as cores de texto e de fundo */
{
   textcolor(c1);
   textbackground(c2);
   gotoxy(x,y);
   cprintf(txt);
   textcolor(15);
   textbackground(0);
}

void centro(char TXT[],int Y,int C1,int C2)
/*funcao que centraliza um texto dado a linha e as cores do texto e de fundo */
{
   int X;

   X=(80-strlen(TXT))/2;
   textcolor(C1);
   textbackground(C2);
   gotoxy(X,Y);
   cprintf(TXT);
   textcolor(15);
   textbackground(0);
}

void tela_fundo(void)
/*Cria a tela de fundo deste programa */
{
   char TXT[81];
   int F;

   limpa_tela();

   textbackground(0);
   textcolor(15);
   for (F=0;F<80;F++)
      TXT[F]=32;
   TXT[80]=0;
   cprintf(TXT);
   sprintf(TXT,"== TRATAMENTO DE SINAIS ==");
   centro(TXT,1,14,0);
   printf("\n");
}

int le_dados(char *nome1, char *nome2, double *(&vetor1), double *(vetor2))
/*Faz a leitura dos dados do arquivo e coloca-os em vetores*/
{
```

```
  int i,j;
  double aux;
  FILE *arq1, *arq2;

  if ( (arq1=fopen(nome1,"rt"))==NULL )
  {
    quadro(14,6,66,8,14,4);
    escrever("IMPOSS+VEL ABRIR ARQUIVO!!!",20,7,15,4);
    return 0;
  }
  else
    if ( (arq2=fopen(nome2,"rt"))==NULL )
    {
      quadro(14,6,66,8,14,4);
      escrever("IMPOSS+VEL ABRIR ARQUIVO!!!",20,7,15,4);
      return 0;
    }

  for(i=0;i<SIZE;i++)
  {
    if ( !fscanf(arq1, "%lf", &vetor1[i]) )
    {
      quadro(14,6,66,8,14,4);
      escrever("ERRO NA LEITURA DE ARQUIVO!!!",20,7,15,4);
      return 0;
    }
  }
  for(i=0;i<SIZE;i++)
  {
    if ( !fscanf(arq2, "%lf", &vetor2[i]) )
    {
      quadro(14,6,66,8,14,4);
      escrever("ERRO NA LEITURA DE ARQUIVO!!!",20,7,15,4);
      return 0;
    }
  }
  fclose(arq1);
  fclose(arq2);

  return 1;

}

void tratamento(double *(&vetor), double *vec)
/*Faz o tratamento dos sinais transformando-os em superfícies a partir de um vetor  */
{
  int i,j, aux1, aux2, aux3, aux4;
  double temp;
  struct time t;

  aux3=SIZE-1;
  aux4=VAR;

  for(i=0;i<SIZE;i++)
    vetor[i]=vec[i];


  for( i=0;i<(int)(aux3/200*aux4);i++)
  {
```

```
    aux1 = rand()%(aux3-1);
    aux2 = rand()%(aux3-1);
    temp = vetor[aux1];
    vetor[aux1]=vetor[aux2];
    vetor[aux2]=temp;
  }

  temp = 0.0;                  // Normalizaç¦o da matriz
  for(i=0;i<(SIZE);i++)
    temp+=pow(vetor[i],2);
  temp = sqrt(fabs(temp));
  for(i=0;i<(SIZE);i++)
    vetor[i]/=temp;

}

void gera_arq(int n, char *nome, double *vetor1, double *vetor2,
      double  saida1, double saida2, flag tipo, inclui resp)
// Funç¦o que gera arquivos. O flag indica qual o tipo de arquivo está sendo criado.
{
  int i,j,k,aux;
  FILE *arq;
  double *matriz;

  matriz = new double [SIZE];
  for(i=0;i<(SIZE);i++)
      matriz[i]=0.0;

  if(resp==NORMAL)
  {
    if ( (arq=fopen(nome,"wt"))==NULL )
    {
      quadro(14,6,66,8,14,4);
      escrever("ERRO NA CRIAÇ¦O DE ARQUIVO!!!",20,7,15,4);
      goto end;
    }
  }
  else
  {
    if ( (arq=fopen(nome,"at"))==NULL )
    {
      quadro(14,6,66,8,14,4);
      escrever("ERRO NA CRIAÇ¦O DE ARQUIVO!!!",20,7,15,4);
      goto end;
    }
  }

  if(tipo!=PESOS)
    if( resp==NORMAL )
      fprintf(arq,"%d\n\n",2*n);   // para arquivos de treinamento e entrada

  for(k=0;k<n;k++)
  {
    printf(".");
    for(i=0;i<(SIZE);i++)
        matriz[i]=0.0;
    tratamento(matriz,vetor1);  // tratamento dos sinais
    for(i=0;i<(SIZE);i++)
        fprintf(arq," %8.6lf",matriz[i]);  // impress¦o das matrizes
```

```
    if( tipo == TREINAMENTO )
      fprintf(arq,"\n %8.6lf\n",saida1);   // impress¦o das saídas para
    else                          // os arquivos de treinamento
      fprintf(arq,"\n");
  }
  for(k=0;k<n;k++)
  {
    printf(".");
    for(i=0;i<(SIZE);i++)
        matriz[i]=0.0;
    tratamento(matriz,vetor2);  // tratamento dos sinais
    for(i=0;i<(SIZE);i++)
        fprintf(arq," %8.6lf",matriz[i]);  // impress¦o das matrizes
    if( tipo == TREINAMENTO )
      fprintf(arq,"\n %8.6lf\n",saida2);   // impress¦o das saídas para
    else                          // os arquivos de treinamento
      fprintf(arq,"\n");
  }

  end:

  delete matriz;
  matriz=NULL;
  fclose(arq);

  if(resp==H3 || resp==H5)
  {
    if ( (arq=fopen(nome,"r+t"))==NULL )
    {
      quadro(14,6,66,8,14,4);
      escrever("ERRO NA CRIAÇ¦O DE ARQUIVO!!!",20,7,15,4);
    }
    else
      if(tipo!=PESOS)
      {
        fscanf(arq,"%d",&aux);
        rewind(arq);
        fprintf(arq," %d\n",aux+2*n);
      }
    fclose(arq);
  }
}

void main(void)
{
  int i, n_dados, n;
  double *vetor1, *vetor2;
  char *nomearq1,*nomearq2, *nome_trn, *nome_tst, resp;
  FILE *arq;
  resposta h3=NAO, h5=NAO;

  nomearq1 = new char [15];
  nomearq2 = new char [15];
  nome_trn = new char [15];
  nome_tst = new char [15];

  vetor1 = new double [SIZE];
  vetor2 = new double [SIZE];
```

```
for(i=0;i<SIZE;i++)
{
  vetor1[i] = 0.0;
  vetor2[i] = 0.0;
}

do
{ erro:

  limpa_tela();
  quadro(14,4,66,14,14,1);
  centro("== TRATAMENTO DE SINAIS PARA BACKPROPAGATION ==",5,14,1);
  if(h5==NAO)
  {
    if(h3==NAO)
    {
      escrever("     Nome do arquivo do sinal 1: ",17,7,15,1);
      escrever("     Nome do arquivo do sinal 2: ",17,8,15,1);
    }
    else
    {
      escrever(" Arquivo da 3a. harm. do sinal 1: ",17,7,15,1);
      escrever(" Arquivo da 3a. harm. do sinal 2: ",17,8,15,1);
    }
  }
  else
  {
    escrever(" Arquivo da 5a. harm. do sinal 1: ",17,7,15,1);
    escrever(" Arquivo da 5a. harm. do sinal 2: ",17,8,15,1);
  }

  escrever("                    Distorcao: ",17,10,15,1);
  escrever("Número de dados para treinamento: ",17,11,15,1);

  escrever("  Nome do arquivo de treinamento: ",17,12,15,1);
  gotoxy(51,7);
  scanf("%s",nomearq1);
  gotoxy(51,8);
  scanf("%s",nomearq2);

  do
  {
    gotoxy(51,10);
    scanf("%d",&VAR);
  }
  while( VAR<0 );
  do
  {
    gotoxy(51,11);
    scanf("%d",&n_dados);
  }
  while( n_dados<0 );

  if(h3==NAO&&h5==NAO)
  {
    gotoxy(51,12);
    scanf("%s",nome_trn);
  }
  else
```

```
   escrever(nome_trn,51,12,15,1);

 strcat(nomearq1,".txt");
 strcat(nomearq2,".txt");

 printf("\n\n\n\n\nAguarde...");

 if( ! le_dados(nomearq1,nomearq2,vetor1,vetor2) ) // leitura do arquivo
 {
   getch();
   goto erro;
 }

 if(h3==NAO&&h5==NAO)
 {
   strcpy(nome_tst,nome_trn);
   strcat(nome_trn,".trn");
   strcat(nome_tst,".ent");
 }

 n=n_dados/2;

 if( h5==NAO )
 {
   if( h3==NAO)
   {
     gera_arq(n,nome_trn,vetor1,vetor2,0,1,TREINAMENTO, NORMAL);   // gera arquivo de treinamento
     gera_arq(n,nome_tst,vetor1,vetor2,0,1,ENTRADAS, NORMAL); // gera arquivo de entrada
   }
   else
   {
     gera_arq(n,nome_trn,vetor1,vetor2,0,1,TREINAMENTO,H3);   // gera arquivo de treinamento
     gera_arq(n,nome_tst,vetor1,vetor2,0,1,ENTRADAS,H3); // gera arquivo de entrada
   }
 }
 else
 {
   gera_arq(n,nome_trn,vetor1,vetor2,0,1,TREINAMENTO,H5);   // gera arquivo de treinamento
   gera_arq(n,nome_tst,vetor1,vetor2,0,1,ENTRADAS,H5); // gera arquivo de entrada
 }

 if(h3==NAO)
 {
   quadro(18,17,60,19,14,4);
   escrever("Incluir 3a. Harmônica? (S/N): ",24,18,15,4);
   gotoxy(54,18);
   do
   {
     scanf("%c",&resp);
   }
   while((resp!='n')&&(resp!='N')&&(resp!='s')&&(resp!='S'));

   if((resp=='s')||(resp=='S'))
   {
     h3 = SIM;
     goto erro;
   }
 }
 else
```

```
        if(h5==NAO)
        {
          quadro(18,17,60,19,14,4);
          escrever("Incluir 5a. Harmônica? (S/N): ",24,18,15,4);
          gotoxy(54,18);
          do
          {
            scanf("%c",&resp);
          }
          while((resp!='n')&&(resp!='N')&&(resp!='s')&&(resp!='S'));

          if((resp=='s')||(resp=='S'))
          {
            h5 = SIM;
            goto erro;
          }
        }

      limpa_tela();
      quadro(14,6,66,8,14,1);
      escrever("Todos os arquivos foram criados com sucesso!",18,7,15,1);

      quadro(16,12,64,14,14,4);
      escrever("Deseja continuar no programa? (S/N): ",22,13,15,4);
      do
      {
        gotoxy(59,13);
        scanf("%c",&resp);
      }
      while((resp!='n')&&(resp!='N')&&(resp!='s')&&(resp!='S'));
      if((resp!='s')||(resp!='S'))
      {
        h3 = NAO;
        h5 = NAO;
      }
    }
    while((resp!='n')&&(resp!='N'));

    end:

    delete nomearq1;
    delete nomearq2;
    delete nome_trn;
    delete nome_tst;
    delete vetor1;
    delete vetor2;

    nomearq1 = NULL;
    nomearq2 = NULL;
    nome_trn = NULL;
    nome_tst = NULL;
    vetor1 = NULL;
    vetor2 = NULL;
}
```

D. LISTAGEM DO PROGRAMA PARA GERAÇÃO DOS ARQUIVOS DE TREINAMENTO (PRÉ-PROCESSAMENTO DOS SINAIS) PARA O COUNTERPROPAGATION

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>

#define SIZE 2048

enum flag { TREINAMENTO, PESOS, ENTRADAS };
enum resposta { SIM, NAO };
enum inclui { NORMAL, H3 , H5 };

int VAR;

void limpa_tela(void)
/*Limpa a tela */
{
  clrscr();
}

void quadro(int X1,int Y1,int X2,int Y2,int C1,int C2)
/*Funcao de tela que desenha um quadro dados os extremos de sua diagonal
  e as cores do texto e do fundo                    */
{
  char T[81];
  int F;

  for(F=0;F<81;F++)
    T[F]=0;
  textbackground(C2);
  textcolor(C1);
  T[0]=201;
  for(F=X1+1;F<X2;F++)
    T[F-X1]=205;
  T[X2-X1]=187;
  gotoxy(X1,Y1);
  cprintf(T);
  T[0]=186;
  for(F=X1+1;F<X2;F++)
    T[F-X1]=32;
  T[X2-X1]=186;
  for(F=Y1+1;F<Y2;F++)
  {
    gotoxy(X1,F);
    cprintf(T);
  }
```

```
    T[0]=200;
    for(F=X1+1;F<X2;F++)
      T[F-X1]=205;
    T[X2-X1]=188;
    gotoxy(X1,Y2);
    cprintf(T);
    textcolor(7);
    textbackground(0);
    for(F=X1;F<=X2;F++)
      T[F-X1]=176;
    textbackground(0);
    textcolor(15);
}

void escrever(char txt[],int x,int y,int c1,int c2)
/*escreve um texto na posicao (x,y) dado as cores de texto e de fundo */
{
   textcolor(c1);
   textbackground(c2);
   gotoxy(x,y);
   cprintf(txt);
   textcolor(15);
   textbackground(0);
}

void centro(char TXT[],int Y,int C1,int C2)
/*funcao que centraliza um texto dado a linha e as cores do texto e de fundo */
{
   int X;

   X=(80-strlen(TXT))/2;
   textcolor(C1);
   textbackground(C2);
   gotoxy(X,Y);
   cprintf(TXT);
   textcolor(15);
   textbackground(0);
}

void tela_fundo(void)
/*Cria a tela de fundo deste programa */
{
   char TXT[81];
   int F;

   limpa_tela();

   textbackground(0);
   textcolor(15);
   for (F=0;F<80;F++)
     TXT[F]=32;
   TXT[80]=0;
   cprintf(TXT);
   sprintf(TXT,"== TRATAMENTO DE SINAIS ==");
   centro(TXT,1,14,0);
   printf("\n");
}

int le_dados(char *nome1, char *nome2, double *(&vetor1), double *(vetor2))
```

```
/*Faz a leitura dos dados do arquivo e coloca-os em vetores*/
{
  int i,j;
  double aux;
  FILE *arq1, *arq2;

  if ( (arq1=fopen(nome1,"rt"))==NULL )
  {
    quadro(14,6,66,8,14,4);
    escrever("IMPOSS+VEL ABRIR ARQUIVO!!!",20,7,15,4);
    return 0;
  }
  else
    if ( (arq2=fopen(nome2,"rt"))==NULL )
    {
      quadro(14,6,66,8,14,4);
      escrever("IMPOSS+VEL ABRIR ARQUIVO!!!",20,7,15,4);
      return 0;
    }

  for(i=0;i<SIZE;i++)
  {
    if ( !fscanf(arq1, "%lf", &vetor1[i]) )
    {
      quadro(14,6,66,8,14,4);
      escrever("ERRO NA LEITURA DE ARQUIVO!!!",20,7,15,4);
      return 0;
    }
  }
  for(i=0;i<SIZE;i++)
  {
    if ( !fscanf(arq2, "%lf", &vetor2[i]) )
    {
      quadro(14,6,66,8,14,4);
      escrever("ERRO NA LEITURA DE ARQUIVO!!!",20,7,15,4);
      return 0;
    }
  }
  fclose(arq1);
  fclose(arq2);

  aux = fabs(vetor1[0]);
  for(i=1;i<SIZE;i++)
    if(vetor1[i]>aux)
      aux = fabs(vetor1[i]);
  for(i=0;i<SIZE;i++)
    vetor1[i]/=aux;

  aux = fabs(vetor2[0]);
  for(i=1;i<SIZE;i++)
    if(vetor2[i]>aux)
      aux = fabs(vetor2[i]);
  for(i=0;i<SIZE;i++)
    vetor2[i]/=aux;

  return 1;

}
```

```
void tratamento(double **(&matriz), double *vec, int grade)
/*Faz o tratamento dos sinais transformando-os em superficies a partir de um vetor  */
{
  int i,j, posicao_ini, posicao_pos, aux1, aux2, aux3, aux4;
  double data_ini, data_pos, temp, *vetor;
  struct time t;

  aux3=SIZE-1;
  aux4=VAR;

  vetor = new double [SIZE];

  for(i=0;i<SIZE;i++)
    vetor[i]=vec[i];

  data_ini = vetor[0];
  posicao_ini = (int)((data_ini+1)*(grade)/2);   // posiçÞo de um ponto no instante k

  for( i=0;i<(int)(aux3/200*aux4);i++)
  {
    aux1 = rand()%(aux3-1);
    aux2 = rand()%(aux3-1);
    temp = vetor[aux1];
    vetor[aux1]=vetor[aux2];
    vetor[aux2]=temp;
  }

  for(i=1;i<SIZE;i++)
  {
    data_pos = vetor[i];
    posicao_pos=(int)((data_pos+1)*(grade)/2);  // posiçÞo de um ponto no instante k+1
    matriz[posicao_ini][posicao_pos]++;      // incrementa a matriz na linha f(k) e na coluna f(k+1)
    data_ini=data_pos;
    posicao_ini = posicao_pos;
  }

  for(i=0;i<(grade+1);i++)
    for(j=0;j<(grade+1);j++)
      matriz[i][j]=(1-pow(0.5,matriz[i][j]));     // faz tratamento de amplitude por uma PG

  delete vetor;
  vetor = NULL;
}

void gera_arq(int n, char *nome, int grade, double *vetor1, double *vetor2,
                    double  saida1, double saida2, flag tipo, inclui resp)
// FunçÞo que gera arquivos. O flag indica qual o tipo de arquivo está sendo criado.
{
  int i,j,k,aux;
  FILE *arq;
  double **matriz;

  matriz = new double *[grade+1];
  for(i=0;i<(grade+1);i++)
  {
    matriz[i] = new double [grade+1];
    for(j=0;j<(grade+1);j++)
      matriz[i][j]=0.0;
  }
```

```
if(resp==NORMAL)
{
  if ( (arq=fopen(nome,"wt"))==NULL )
  {
    quadro(14,6,66,8,14,4);
    escrever("ERRO NA CRIAÇ¦O DE ARQUIVO!!!",20,7,15,4);
    goto end;
  }
}
else
{
  if ( (arq=fopen(nome,"at"))==NULL )
  {
    quadro(14,6,66,8,14,4);
    escrever("ERRO NA CRIAÇ¦O DE ARQUIVO!!!",20,7,15,4);
    goto end;
  }
}

if(tipo!=PESOS)
  if( resp==NORMAL )
    fprintf(arq,"%d\n\n",2*n);   // para arquivos de treinamento e entrada

for(k=0;k<n;k++)
{
  printf(".");
  for(i=0;i<(grade+1);i++)
    for(j=0;j<(grade+1);j++)
      matriz[i][j]=0.0;
  tratamento(matriz,vetor1,grade);  // tratamento dos sinais
  for(i=0;i<(grade+1);i++)
    for(j=0;j<(grade+1);j++)
    {
      fprintf(arq," %8.6lf",matriz[i][j]);  // impress¦o das matrizes
      if(j==(grade))
        fprintf(arq,"\n");
    }
  if( tipo == TREINAMENTO )
    fprintf(arq," %8.6lf\n\n",saida1);   // impress¦o das saídas para
  else                        // os arquivos de treinamento
    fprintf(arq," \n\n");
}
for(k=0;k<n;k++)
{
  printf(".");
  for(i=0;i<(grade+1);i++)
    for(j=0;j<(grade+1);j++)
      matriz[i][j]=0.0;
  tratamento(matriz,vetor2,grade);  // tratamento dos sinais
  for(i=0;i<(grade+1);i++)
    for(j=0;j<(grade+1);j++)
    {
      fprintf(arq," %8.6lf",matriz[i][j]);  // impress¦o das matrizes
      if(j==(grade))
        fprintf(arq,"\n");
    }
  if( tipo == TREINAMENTO )
    fprintf(arq," %8.6lf\n\n",saida2);   // impress¦o das saídas para
```

```
      else                    // os arquivos de treinamento
        fprintf(arq," \n\n");
    }

    end:

    for(i=0;i<(grade+1);i++)
      delete matriz[i];
    delete matriz;
    matriz=NULL;
    fclose(arq);

    if(resp==H3 || resp==H5)
    {
      if ( (arq=fopen(nome,"r+t"))==NULL )
      {
        quadro(14,6,66,8,14,4);
        escrever("ERRO NA CRIAÇ¦O DE ARQUIVO!!!",20,7,15,4);
      }
      else
        if(tipo!=PESOS)
        {
          fscanf(arq,"%d",&aux);
          rewind(arq);
          fprintf(arq," %d\n",aux+2*n);
        }
      fclose(arq);
    }
}

void main(void)
{
  int i,grade, n_dados, n;
  double *vetor1, *vetor2,saida1, saida2;
  char *nomearq1,*nomearq2, *nome_trn, *nome_tst, *nome_wkh, *nome_wgr, *nome_sz,
     resp, inicia_pesos;
  FILE *arq, *arq_wkh, *arq_wgr, *arq_sz;
  resposta h3=NAO, h5=NAO;

  nomearq1 = new char [15];
  nomearq2 = new char [15];
  nome_trn = new char [15];
  nome_tst = new char [15];
  nome_wkh = new char [15];
  nome_wgr = new char [15];
  nome_sz  = new char [15];

  vetor1 = new double [SIZE];
  vetor2 = new double [SIZE];

  for(i=0;i<SIZE;i++)
  {
    vetor1[i] = 0.0;
    vetor2[i] = 0.0;
  }

  do
  { erro:
```

```
limpa_tela();
quadro(14,4,66,15,14,1);
centro("== TRATAMENTO DE SINAIS ==",5,14,1);
if(h5==NAO)
{
  if(h3==NAO)
  {
    escrever("     Nome do arquivo do sinal 1: ",17,7,15,1);
    escrever("     Nome do arquivo do sinal 2: ",17,8,15,1);
  }
  else
  {
    escrever(" Arquivo da 3a. harm. do sinal 1: ",17,7,15,1);
    escrever(" Arquivo da 3a. harm. do sinal 2: ",17,8,15,1);
  }
}
else
{
  escrever(" Arquivo da 5a. harm. do sinal 1: ",17,7,15,1);
  escrever(" Arquivo da 5a. harm. do sinal 2: ",17,8,15,1);
}

escrever("                    Grade: ",17,9,15,1);
escrever("                Distorcao: ",17,10,15,1);
escrever("Número de dados para treinamento: ",17,11,15,1);

escrever("  Nome do arquivo de treinamento: ",17,12,15,1);
escrever("   Inicializacao de pesos? (s/n): ",17,13,15,1);
gotoxy(51,7);
scanf("%s",nomearq1);
gotoxy(51,8);
scanf("%s",nomearq2);
if(h3==NAO&&h5==NAO)
  do
  {
    gotoxy(51,9);
    scanf("%d",&grade);
    grade = grade - 1;
  }
  while( grade<0 || grade>50 );
else
{
  gotoxy(51,9);
  printf("%d",grade+1);
}

do
{
  gotoxy(51,10);
  scanf("%d",&VAR);
}
while( VAR<0 );
do
{
  gotoxy(51,11);
  scanf("%d",&n_dados);
}
while( n_dados<0 );
```

```
    if(h3==NAO&&h5==NAO)
    {
      gotoxy(51,12);
      scanf("%s",nome_trn);
    }
    else
      escrever(nome_trn,51,12,15,1);

    if(h3==NAO&&h5==NAO)
    do
    {
      gotoxy(51,13);
      scanf("%c",&inicia_pesos);
    }
    while((inicia_pesos!='n')&&(inicia_pesos!='N')&&(inicia_pesos!='s')&&(inicia_pesos!='S'));

    else
      if(inicia_pesos=='s' || inicia_pesos=='S')
        escrever("sim",51,13,15,1);
      else
        escrever("n|o",51,13,15,1);

    strcat(nomearq1,".txt");
    strcat(nomearq2,".txt");

    printf("\n\n\n\n\nAguarde...");

    if( ! le_dados(nomearq1,nomearq2,vetor1,vetor2) ) // leitura do arquivo
    {
      getch();
      goto erro;
    }

    if(h3==NAO&&h5==NAO)
    {
      strcpy(nome_wkh,nome_trn);
      strcpy(nome_wgr,nome_trn);
      strcpy(nome_sz,nome_trn);
      strcpy(nome_tst,nome_trn);
      strcat(nome_trn,".trn");
      strcat(nome_wkh,".wkh");
      strcat(nome_wgr,".wgr");
      strcat(nome_sz,".sz");
      strcat(nome_tst,".ent");
    }

    n=n_dados/2;

    if( h5==NAO )
    {
      if( h3==NAO)
      {
        gera_arq(n,nome_trn,grade,vetor1,vetor2,0,1,TREINAMENTO, NORMAL);   // gera arquivo de
treinamento
        if(inicia_pesos=='s' || inicia_pesos=='S')
          gera_arq(1,nome_wkh,grade,vetor1,vetor2,0,1,PESOS, NORMAL);    // gera arquivo de pesos
//      gera_arq(n,nome_tst,grade,vetor1,vetor2,0,1,ENTRADAS, NORMAL); // gera arquivo de entrada
//      gera_arq(1,nome_wkh,grade,vetor1,vetor2,0,1,PESOS, NORMAL);    // gera arquivo de pesos
```

```
      }
      else
      {
        gera_arq(n,nome_trn,grade,vetor1,vetor2,0,1,TREINAMENTO,H3);  // gera arquivo de treinamento
        if(inicia_pesos=='s' || inicia_pesos=='S')
          gera_arq(1,nome_wkh,grade,vetor1,vetor2,0,1,PESOS,H3);  // gera arquivo de pesos
//      gera_arq(n,nome_tst,grade,vetor1,vetor2,0,1,ENTRADAS,H3); // gera arquivo de entrada
//      gera_arq(1,nome_wkh,grade,vetor1,vetor2,0,1,PESOS,H3);  // gera arquivo de pesos
      }
    }
    else
    {
      gera_arq(n,nome_trn,grade,vetor1,vetor2,0,1,TREINAMENTO,H5);  // gera arquivo de treinamento
      if(inicia_pesos=='s' || inicia_pesos=='S')
        gera_arq(1,nome_wkh,grade,vetor1,vetor2,0,1,PESOS,H5);  // gera arquivo de pesos
//    gera_arq(n,nome_tst,grade,vetor1,vetor2,0,1,ENTRADAS,H5); // gera arquivo de entrada
//    gera_arq(1,nome_wkh,grade,vetor1,vetor2,0,1,PESOS,H5);  // gera arquivo de pesos
    }

    if(h3==NAO)
    {
      quadro(18,17,60,19,14,4);
      escrever("Incluir 3a. Harmônica? (S/N): ",24,18,15,4);
      gotoxy(54,18);
      do
      {
        scanf("%c",&resp);
      }
      while((resp!='n')&&(resp!='N')&&(resp!='s')&&(resp!='S'));

      if((resp=='s')||(resp=='S'))
      {
        h3 = SIM;
        goto erro;
      }
    }
    else
      if(h5==NAO)
      {
        quadro(18,17,60,19,14,4);
        escrever("Incluir 5a. Harmônica? (S/N): ",24,18,15,4);
        gotoxy(54,18);
        do
        {
          scanf("%c",&resp);
        }
        while((resp!='n')&&(resp!='N')&&(resp!='s')&&(resp!='S'));

        if((resp=='s')||(resp=='S'))
        {
          h5 = SIM;
          goto erro;
        }
      }

    n = (grade+1)*(grade+1);

    if(inicia_pesos=='s' || inicia_pesos=='S')
    {
```

```
        if ( (arq_sz=fopen(nome_sz,"wt"))==NULL )
        {
          quadro(14,6,66,8,14,4);
          escrever("ERRO NA CRIAÇ¦O DE ARQUIVO!!!",20,7,15,4);
          goto end;
        }

        if(h3==NAO&&h5==NAO)
          fprintf(arq_sz," %d\n %d\n %d", n, 1, 2);
        else
          if(h3==SIM&&h5==NAO)
            fprintf(arq_sz," %d\n %d\n %d", n, 1, 4);
          else
            if(h3==SIM&&h5==SIM)
              fprintf(arq_sz," %d\n %d\n %d", n, 1, 6);

        fclose(arq_sz);

        saida1=0;
        saida2=1;

        if ( (arq_wgr=fopen(nome_wgr,"wt"))==NULL )
        {
          quadro(14,6,66,8,14,4);
          escrever("ERRO NA CRIAÇ¦O DE ARQUIVO!!!",20,7,15,4);
          goto end;
        }

        if(h3==NAO&&h5==NAO)
          fprintf(arq_wgr," %8.6lf %8.6lf\n",saida1,saida2);
        else
          if(h3==SIM&&h5==NAO)
            fprintf(arq_wgr," %8.6lf %8.6lf %8.6lf %8.6lf\n",saida1,saida2,saida1,saida2);
          else
            if(h3==SIM&&h5==SIM)
              fprintf(arq_wgr," %8.6lf %8.6lf %8.6lf %8.6lf %8.6lf
%8.6lf\n",saida1,saida2,saida1,saida2,saida1,saida2);

        fclose(arq_wgr);
      }


      limpa_tela();
      quadro(14,6,66,8,14,1);
      escrever("Todos os arquivos foram criados com sucesso!",18,7,15,1);

      quadro(16,12,64,14,14,4);
      escrever("Deseja continuar no programa? (S/N): ",22,13,15,4);
      do
      {
        gotoxy(59,13);
        scanf("%c",&resp);
      }
      while((resp!='n')&&(resp!='N')&&(resp!='s')&&(resp!='S'));
      if((resp!='s')||(resp!='S'))
      {
        h3 = NAO;
        h5 = NAO;
      }
```

```
    }
    while((resp!='n')&&(resp!='N'));

    end:

    delete nomearq1;
    delete nomearq2;
    delete nome_trn;
    delete nome_tst;
    delete nome_wkh;
    delete nome_wgr;
    delete nome_sz;
    delete vetor1;
    delete vetor2;

    nomearq1 = NULL;
    nomearq2 = NULL;
    nome_trn = NULL;
    nome_tst = NULL;
    nome_wkh = NULL;
    nome_wgr = NULL;
    vetor1 = NULL;
    vetor2 = NULL;
}
```