Instituto de Ciências Matemáticas de São Carlos

ISSN-0109-2569

Utilização do Matlab para o desenvolvimento de de redes neurais

VIVIANI PASQUINI SANTOS

ROSELI A. FRANCELIN ROMERO

Nº 18

RELATÓRIOS TÉCNICOS DO ICMSC

SÃO CARLOS
jun. / 1994

UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS DE SÃO CARLOS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO E ESTATÍSTICA

UTILIZAÇÃO DO MATLAB PARA O DESENVOLVIMENTO DE REDES NEURAIS

Viviani Pasquini Santos

Roseli A. Francelin Romero *

SYSNO 2336455
DATA 14/03/13
ICMC - SBAB

junho/1994

* Este trabalho foi elaborado com o apoio da FAPESP - Processo nro. 92/2151-8

Índice

1 Introdução 4
2 O que é o MATLAB? 5
2.1 Utilizando o MATLAB 5
2.2 M-files 6
2.3 Toolboxes 6
2.4 Help 7
2.5 Save/Load 7
2.5.1 Savemat.c 8
2.6 Matrizes Simples 9
2.7 Elementos de uma Matriz 9
2.8 Comandos e Variáveis 10
2.9 Números e Expressões Aritméticas 11
2.10 Operações com Matrizes 12
2.10.1 Transposição 12
2.10.2 Adição e Subtração 12
2.10.3 Multiplicação de Matrizes 13
2.10.4 Divisão de Matrizes 13
2.11 Operações com Vetores 14
2.12 Geração de Vetores 14
2.13 Gráficos 16
2.14 Impressão de Gráficos 21
3 Redes Neurais 24
3.1 Perceptron 25
3.2 Regra Delta 25
3.3 Redes Multi-Camadas 26
3.4 Algoritmo Backpropagation 28
4 Redes Neurais no Ambiente MATLAB 34
4.1 Modelo de Neurônio e Arquiteturas de Rede 34
4.1.1 Neurônio com uma entrada 34
4.1.2 Funções de Transferência 35
4.1.3 Neurônio com Múltiplas Entradas 36
4.1.4 Uma camada de Neurônios 37
4.1.5 Múltiplas Camadas de Neurônios 38
4.1.6 Batching : Múltiplos Vetores de Entrada 39
4.2 Implementando o Perceptron no Ambiente MATLAB. 40
4.2.1 Inicialização 40
4.2.2 Regra de Aprendizagem 40
4.2.3 Treinamento 41
4.2.4 Exemplo 43

4.3 Implementando Backpropagation no Ambiente MATLAB 47
4.3.1 Inicialização 47
4.3.2 Regra de Aprendizagem 48
4.3.3 Treinamento 50
4.3.4 Exemplos 52
4.4 Melhorando o Algoritmo Backpropagation 60
4.4.1 Aprendizagem com momentum 60
4.4.2 Melhores Condições Iniciais 61
4.4.3 Taxa de Aprendizagem Adaptativa 61
4.4.4 Adotando o momentum e taxa de aprendizagem adaptatva 62
4.5 Outras Redes Neurais disponíveis no MATLAB 62
5 Aplicação da Backpropagation no Reconhecimento de Doenças do Coração 63
5.1 Testes 64
5.2 Resultados 73
7 Conclusão 74
8 Bibliografia 75
9 Apêndice 76

1 Introdução

O objetivo desse trabalho é descrever os principais procedimentos existentes no Neural Network TOOLBOX do MATLAB para a implementação de redes neurais artificiais.

Inicialmente, as características básicas operacionais do ambiente MATLAB são brevemente apresentados no Capítulo 2, tais como, manipulação de matrizes, geração e impressão de gráficos, linguagem utilizada no MATLAB e edição de arquivos .m.

Em seguida, dois importantes modelos de redes neurais artificiais: PERCEPTRON e Redes Multi-Camadas são discutidos no Capítulo 3.

No capítulo 4, apresentamos como essas duas arquiteturas de redes neurais artificiais são implementadas no ambiente MATLAB, ilustrando o desempenho dessas redes através de exemplos simples.

Finalmente, uma aplicação da Rede Multi-Camadas para Reconhecimento de Doenças do Coração é realizada no Capítulo 5. Topologias diferentes foram utilizadas para mostrar o desempenho desta rede.

Gostaríamos de salientar que o presente trabalho não esgota o potencial disponível no TOOLBOX NEURAL NETWORKS, pois existem alguns modelos de Redes Neurais juntamente com outros recursos que não são apresentados aqui.

2 O que é o MATLAB?

O MATLAB é um pacote de softwares para computação numérica científica e de engenharia. O MATLAB integra análise numérica, cálculo de matrizes, processamento de sinais e construção de gráficos em um desenvolvimento de fácil utilização onde problemas e soluções são expressados exatamente como eles são e escritos matematicamente sem programação tradicional.

O nome MATLAB significa MATriz LABoratory, pois ele é um sistema interativo onde o elemento básico é uma matriz. Isto permite resolver muitos problemas numéricos em um tempo bem menor do que levaria para escrever um programa em linguagens tais como Fortran ou C.

No desenvolvimento de pesquisa o MATLAB tem se tornado a ferramenta padrão para cursos introdutórios em álgebra linear aplicada, assim como para cursos avançados em outras áreas. Na indústria, o MATLAB é usado para pesquisa e para resolver problemas de engenharia e matemática.

2.1 Utilizando o MATLAB

Para chamar o MATLAB, basta digitarmos o comando matlab. Após um instante, a apresentação do MATLAB aparece, e um prompt da seguinte forma é mostrado: ">>". A seguir o interpretador MATLAB fica aguardando instruções.

Para entrarmos com uma pequena matriz, colocamos os dados dentro de colchetes, e separamos as linhas da matriz com ponto e vírgula como mostrado abaixo.

```
A = [1 2 3; 4 5 6; 7 8 10]
```

O MATLAB responderá com

```
A =
     1 2 3
     4 5 6
     7 8 10
```

Para inverter a matriz, digitamos o seguinte comando:

```
B = inv(A)
```

O ponto de exclamação ! no MATLAB é usado para indicar que a linha de comando após o ! deverá ser executada como um comando do sistema operacional. Assim você pode rodar qualquer utilitário do DOS ou outro programa, dentro do MATLAB. Quando a execução do programa termina, o controle retorna ao MATLAB.

As teclas ↑ e ↓ podem ser usadas para chamar linhas de comandos anteriores.

Por exemplo, se você entrar com

```
log(sqt(atan(2*(3+4))))
```

o MATLAB responde com uma mensagem de erro,

```
Undefined variable or function
Symbol in question → sqt
```

pois a função sqt não existe. A função correta é sqrt. Ao invés de reescrevermos a linha de entrada, simplesmente usamos a tecla ↑. A linha incorreta será mostrada novamente e poderemos mover o cursor usando ← até o ponto onde o r deve ser inserido:

```
log(sqrt(atan(2*(3+4))))
ans =
        0.2026
```

Para sair do MATLAB basta digitarmos quit ou exit.

2.2 M-files

Para executarmos um programa no MATLAB, basta criarmos um arquivo com a extensão ".M" em qualquer editor de texto e chamarmos este programa dentro do MATLAB.

Por exemplo, para executarmos o programa exemplo.m digitamos apenas o nome do programa após o prompt:

```
>> exemplo
```

2.3 Toolboxes

Toolboxes opcionais são disponíveis no MATLAB fornecendo aplicações específicas adicionais. Os toolboxes são arquivos .M fornecidos em diretórios adicionais no MATLAB.

Alguns toolboxes pertencentes ao MATLAB são:

1. O Signal Processing Toolbox que adiciona 50 comandos para processamento de sinais de 1 e 2 dimensões.
2. O Control System Toolbox que inclui mais de 40 comandos adicionais para engenharia de controle e teoria de sistemas.
3. O Robust-Control Toolbox que adiciona 35 comandos para o projeto de sistema de controle robusto.

4. O Neural Network Toolbox com mais de 40 comandos adicionais para o desenvolvimento de redes neurais artificiais.

2.4 Help

Um help está disponível no MATLAB, fornecendo informação "on-line", na maioria dos tópicos do MATLAB. Para conseguir listar o help, digite:

```
help
```

Para conseguir o help de um tópico específico, digite help *tópico*. Por exemplo,

```
help eig
```

fornece informações no uso da função "eigenvalue"

```
help [
```

mostra como usar colchetes para construir matrizes.

2.5 Save/Load

"save" e "load" são comandos do MATLAB para armazenar e recuperar variáveis no disco.

MAT-files são arquivos num formato específico para o MATLAB criados pelo comando save e recuperados pelo comando load.

O comando load é o inverso de save.

Por exemplo, o código seguinte cria um arquivo teste.mat no disco contendo os vetores X, Y e Z.

```
X = [1 2 3];
Y = [2 3 4];
Z = X + Y;
save teste X Y Z
```

Se quisermos utilizar o vetor Z do arquivo teste.mat em algum momento, basta digitarmos

```
load teste
Z
```

e o resultado será

```
Z =
     3   5   7
```

2.5.1 Savemat.c

Savemat.c é uma rotina em linguagem C encontrada no MATLAB para converter um matriz de um arquivo em linguagem C para um arquivo .MAT.

savemat(fp, type, pname, mrows, ncols, imagf, preal, pimag)

FILE *fp	ponteiro para o arquivo
int tipo	Tipo flag: normalmente 0 para PC, 1000 para Sun, Mac, e Apollo, 2000 para VAX D-float, 3000 para VAX G-float. Adiciona 1 para variáveis texto.
int mrows	Dimensão da linha
int ncols	Dimensão da coluna
int imagf	Flag imaginário
char *pname	Ponteiro para o nome da matriz
double *preal	Ponteiro para dados reais
double *pimag	Ponteiro para dados imag

Aqui está um exemplo usando "savemat" para gravar uma matriz de um arquivo em linguagem C no disco no formato .MAT.

Sejam as matrizes X e Y definidas no programa em C:

$$X=\begin{bmatrix}0&0&1&1\\0&1&0&1\end{bmatrix}$$

$$Y=\begin{bmatrix}0&1&1&0\end{bmatrix}$$

```
FILE *arq;
double x[4][2];
double y[4][1];
arq = fopen("exemplo.mat","wb");
savemat(arq,0, "P", 2, 4, 0, x, (double *)0);
savemat(arq,0, "T", 1, 4, 0, y, (double *)0);
fclose(arq);
```

Esse código gera o arquivo exemplo.mat que contem as matrizes P e T, obtidas a partir das matrizes X e Y, respectivamente.

Podemos recuperar o arquivo exemplo.mat no MATLAB através da linha de código: **load exemplo**.

2.6 Matrizes Simples

O método mais fácil de inserir uma matriz pequena no MATLAB é usar uma lista explícita. A lista de elementos é separada por brancos ou vírgulas, colocada dentro de colchetes, e usa o ponto e vírgula para indicar fim de linha.

Por exemplo, se digitarmos

```
A = [1 2 3; 4 5 6; 7 8 9]
```

o resultado armazenado será:

```
A =
     1 2 3
     4 5 6
     7 8 9
```

Podemos recuperar matrizes do disco para o MATLAB através de arquivos com extensão ".M". Se um arquivo com nome teste.m contem 3 linhas do texto,

```
A = [1 2 3
     4 5 6
     7 8 9]
```

então o comando teste lê o arquivo teste.m e carrega a matriz A.

2.7 Elementos de uma Matriz

Os elementos de uma matriz podem ser qualquer expressão usada pelo MATLAB; por exemplo,

```
X = [-1.3   sqrt(3)   (1+2+3)*4/5]
```

resulta em

```
X =
   -1.3000   1.7321   4.8000
```

Elementos individuais de uma matriz podem ser referenciados com índices dentro dos parênteses. Continuando nosso exemplo,

```
X(5) = abs(x(1))
```

produz

```
X =
     -1.3000  1.7321  4.8000  0.0000  1.3000
```

Note que o tamanho de X é automaticamente aumentado para aceitar o novo elemento e que os elementos não definidos antes desse novo elemento são iguais a 0.

Matrizes grandes podem ser construídas usando pequenas matrizes como elementos.

Por exemplo, poderíamos colocar outra linha na matriz A com

```
r = [10 11 12];
A = [A ; r]
```

que resulta em

```
A =
    1    2    3
    4    5    6
    7    8    9
   10   11   12
```

Pequenas matrizes podem ser obtidas de matrizes grandes. Por exemplo,

```
A = A(1:3,:);
```

toma as primeiras três linhas e todas as colunas da matriz A para retornar a matriz original A.

2.8 Comandos e Variáveis

O MATLAB é um linguagem baseada em expressões. Expressões estabelecidas pelo usuário são avaliadas pelo sistema MATLAB. Comandos no MATLAB são geralmente da forma

```
variável = expressão
```

ou, simplesmente,

```
expressão
```

Expressões são compostas de operadores, caracteres especiais, funções, e nomes de variáveis. A avaliação da expressão produz uma matriz, que é então mostrada na tela e associada a variável para futuro uso. Se o nome da variável e o sinal "=" são omitidos, uma variável com o nome "ans" é automaticamente criada.

Por exemplo, a expressão

1900/81

produz

ans =

23.4568

Um comando é normalmente terminado com <return>. Contudo, se o último caracter do comando é um ponto e vírgula, o resultado desse comando não é mostrado na tela. Isto é útil em arquivos ".M" ou onde o resultado é muito grande com números que não interessam no resultado final. Por exemplo,

p = 1900/81;

p recebe o valor 23.4568 mas esse resultado não é mostrado na tela.

Se a expressão é tão complicada que o comando não cabe na linha, utilizamos "..." no final da linha seguidos por <return> para indicar que o comando continua na próxima linha.

Por exemplo,

S = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 ...
- 1/8 +1/9 -1/10 + 1/11 - 1/12;

avalia a soma parcial da série, associa a soma a S, mas não mostra nada na tela. Os brancos ao redor do =, + e - não são necessários, mas são incluídos para maior legibilidade.

Nomes de variáveis e funções são formados por uma letra, seguidos por qualquer número de letras e dígitos (ou underscores). Apenas os primeiros 19 caracteres de um nome são guardados.

Obs: MATLAB é "case-sensitive", isto é, distingue entre letras maiúsculas e minúsculas. Assim um a e A não são as mesmas variáveis. Todas as funções devem ser em letras minúsculas; inv(A) inverte A, mas INV(A) é uma função indefinida. Contudo, o comando casesen torna o MATLAB insensível ao tamanho das letras. Deste modo a e A são as mesmas variáveis e INV(A) inverte A.

2.9 Números e Expressões Aritméticas

A notação decimal convencional, com ponto decimal opcional e utilizando o sinal -, é usada para números no MATLAB.

Expressões aritméticas podem ser construídas usando os operadores aritméticos e ordem de precedência usuais:

+ adição
- subtração
* multiplicação
/ divisão
^ potenciação

2.10 Operações com Matrizes

Operações com matrizes são fundamentais ao MATLAB.

2.10.1 Transposição

O caracter especial ' denota a transposição de uma matriz. O comando

```
A = [1 2 3; 4 5 6; 7 8 0]
B = A'
```

resulta em

```
A =
     1 2 3
     4 5 6
     7 8 0

B =
     1 4 7
     2 5 8
     3 6 0
```

e

```
X = [-1 0 2]'
```

produz

```
X =
    -1
     0
     2
```

2.10.2 Adição e Subtração

A adição e a subtração de matrizes são denotadas por + e -. As operações são definidas quando as matrizes tem a mesma dimensão. Por exemplo,

```
C = A + B
```

resulta em

```
C =
     2     6    10
     6    10    14
    10    14     0
```

Adição e subtração também são definidas se um operando é um escalar, isto é, uma matriz 1x1. Neste caso, o escalar é adicionado ou subtraído de todos os elementos do outro operando. Por exemplo, considerando o vetor X da seção 2.10.1 e definindo:

```
Y = X - 1
```

obtemos

```
Y = -2
    -1
     1
```

2.10.3 Multiplicação de Matrizes

A multiplicação é denotada por *. A operação é definida apenas quando o número de colunas da primeira matriz a ser multiplicada é igual ao número de linhas da segunda matriz. Por exemplo,

$$X' * Y = \begin{bmatrix} -1 & 0 & 2 \end{bmatrix} * \begin{bmatrix} -2 \\ -1 \\ 1 \end{bmatrix}$$

resulta em

```
ans =
     4
```

2.10.4 Divisão de Matrizes

Existem dois símbolos para divisão de matrizes no MATLAB, "\" e "/". Se A é uma matriz não sigular, então A \ B e B / A correspondem formalmente a multiplicação à esquerda e à direita de B pelo inverso de A, isto é inv(A)*B e B*inv(A), mas o resultado é obtido diretamente sem o cálculo da inversa de A. Em geral,

X = A \ B é a solução para A * X = B
X = B / A é a solução para X * A = B

Por exemplo,

```
z = A \ B
```

resulta em

```
z =
   -1
    0
    2
```

2.11 Operações com Vetores

Para adição e subtração, as operações com vetores são iguais as operações para matrizes.

A multiplicação de um array é denotado por ".*". Se A e B têm as mesmas dimensões, então A .* B denota o vetor cujos elementos são apenas o produto de cada elemento da matriz A e B.

Por exemplo, se

```
X = [1 2 3];  Y = [4 5 6];
```

Z = X .* Y resulta em

```
Z =
     4 10 18
```

As expressões A ./ B e A .\ B retornam os quocientes dos elementos individuais. Assim,

```
Z = X .\ Y
```

resulta em

```
Z =
  4.0000  2.5000  2.0000
```

e

```
Z = X ./ Y
```

resulta em

```
Z = 0.2500   0.4000  0.5000
```

2.12 Geração de Vetores

O símbolo ":" é um importante caracter do MATLAB. O comando

```
X = 1:5
```

gera um vetor linha contendo os números de 1 a 5 incrementando 1 unidade. Ele produz

```
X = 1 2 3 4 5
```

Outros tipos de incrementos podem ser usados.

```
Y = 0: pi/4 :pi
```

resulta em

```
Y =
    0.0000  0.7854  1.5708  3.1416
```

Incrementos negativos também são possíveis:

```
z = 6:-1:1
```

resulta

```
z =
   6 5 4 3 2 1
```

Elementos de matrizes podem ser referenciados colocando suas posições entre parênteses. Por exemplo, dada a matriz A,

```
A =
    1  2  3
    4  5  6
    7  8  9
```

o comando

```
A(3,3) = A(1,3) + A(3,1)
```

resulta em

```
A =
     1  2  3
     4  5  6
     7  8  10
```

Outros tipos de operações são obtidos do seguinte modo:

```
A(:,[3 5 10]) = B(:,1:3)
```

O código acima troca a terceira, quinta e décima coluna de A pela primeira, segunda e terceira coluna de B. Os ":" indicam que as linhas de A continuarão as mesmas.

A(:) denota todos os elementos de A. Por exemplo:

```
A = [1 2; 3 4; 5 6]
B = A(:)
```

resulta

```
A =
     1   2
     3   4
     5   6

B =
     1
     3
     5
     2
     4
     6
```

2.13 Gráficos

O MATLAB dispõe de muitos recursos que facilitam a construção de gráficos.

Tipos diferentes de gráficos são obtidos através de funções do MATLAB:

plot	x-y linear
mesh	superfície em três dimensões
contour	contorno de um gráfico
bar	gráfico de barras

O gráfico pode ser rotulado através do uso dos seguintes comandos:

```
title('título do gráfico')
xlabel('coordenada x')
ylabel('coordenada y')
grid - grades
```

axis(V) - cria novos limites para as coordenadas x e y. V é definido da seguinte forma: V = [x_min, x_max, y_min, x_max]

Exemplo 1:

```
Y = [0.  .48  .84  1.  .91   .6  .14]
plot(Y)
```

O resultado desses comandos é mostrado no gráfico da figura 2.1

Acrescentamos ao código anterior as seguintes linhas:

```
title('grafico')
xlabel('x')
ylabel('y')
grid
V = [-1,10,-1,2]
axis(V)
```

O resultado é mostrado na figura 2.2

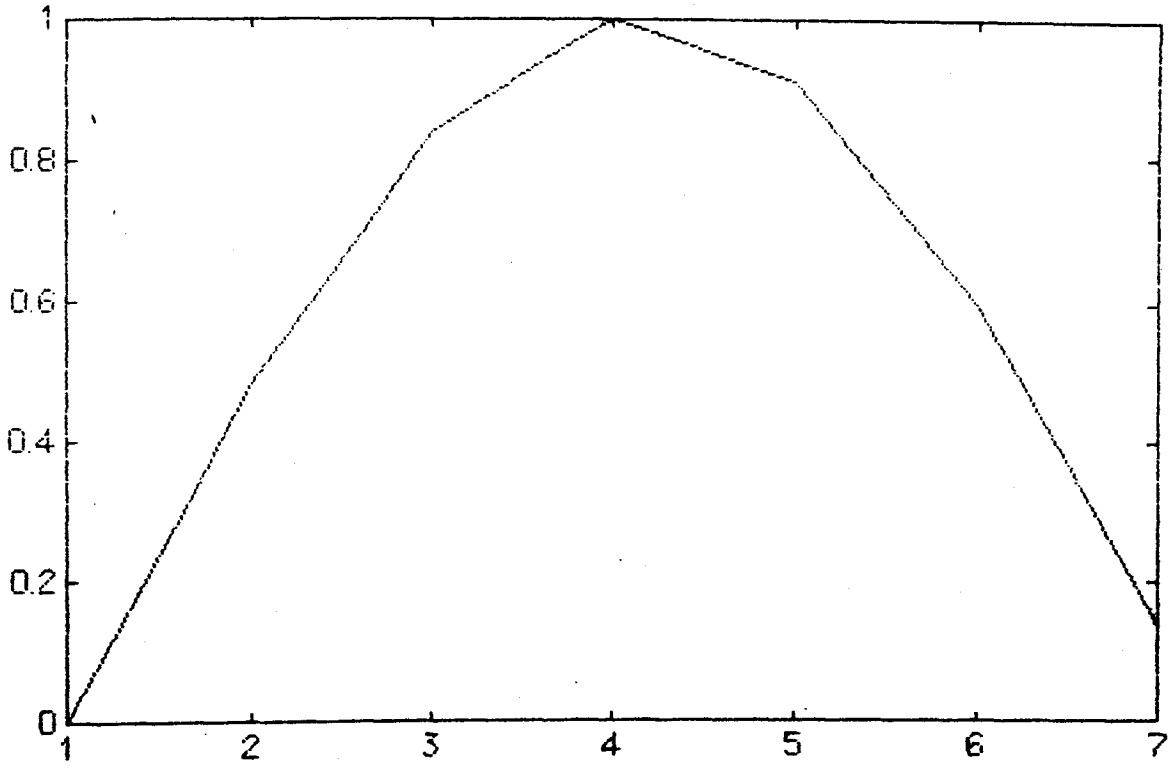


Figura 2.1

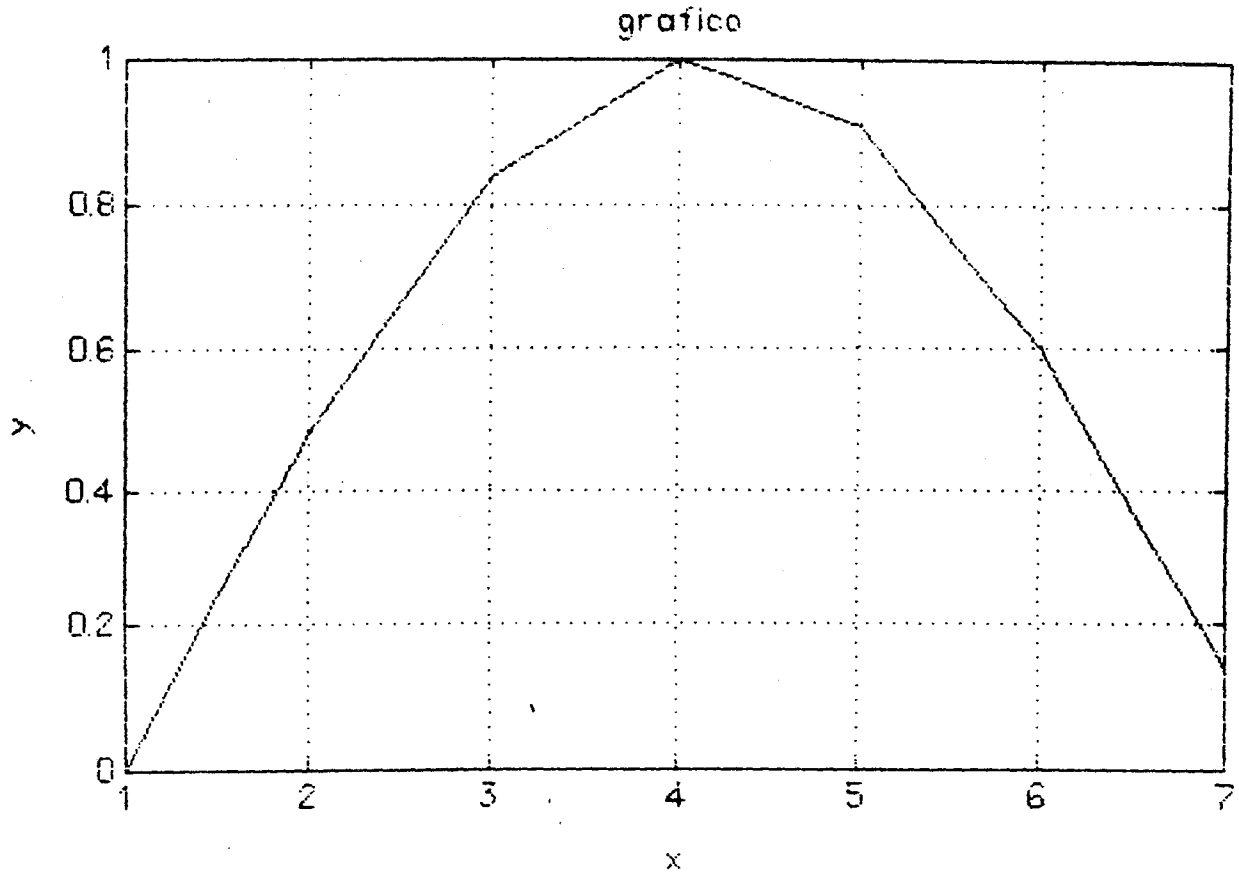


Figura 2.2

Exemplo 2:

```
t = 0:.05:4*pi;
y = sin(t);
plot(t,y)
```

O gráfico resultante pode ser visto na figura 2.3

Exemplo 3:

```
x = -8:.5:8;
y = x';
X = ones(y)*x;
Y = y*ones(x);
R = sqrt(X.^2 + Y.^2) + eps;
Z = sin(R)./R;
mesh(Z)
```

O primeiro comando define o domínio x sobre o qual a função é avaliada. A função ones(n) gera uma matriz nxn cujos elementos são 1, e portanto o terceiro comando cria uma matriz X de linhas repetidas. Após gerar um Y correspondente, uma matriz R é criada contendo a distância do centro da matriz, que é a origem (eps é uma variável permanente do MATLAB). Formando a função sinc e aplicando mesh o resultado é mostrado na figura 2.4.

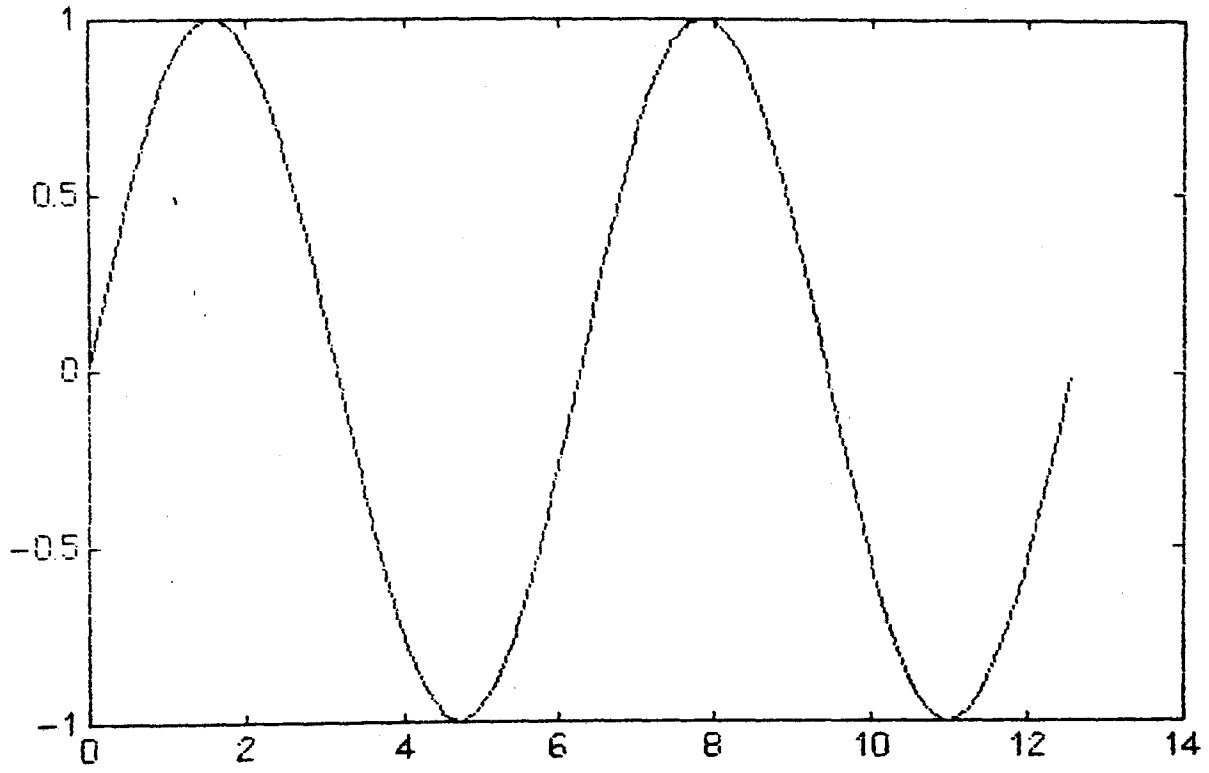


Figura 2.3

Exemplo 4:

Acrescentamos no código anterior, a função:

contour(Z)

O resultado é mostrado na figura 2.5.

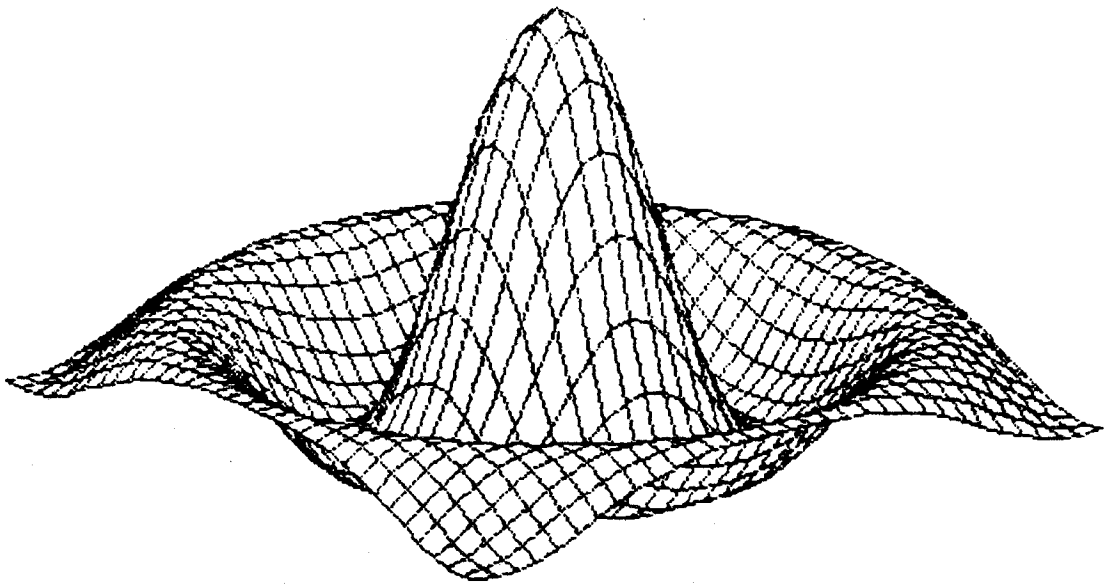


Figura 2.4

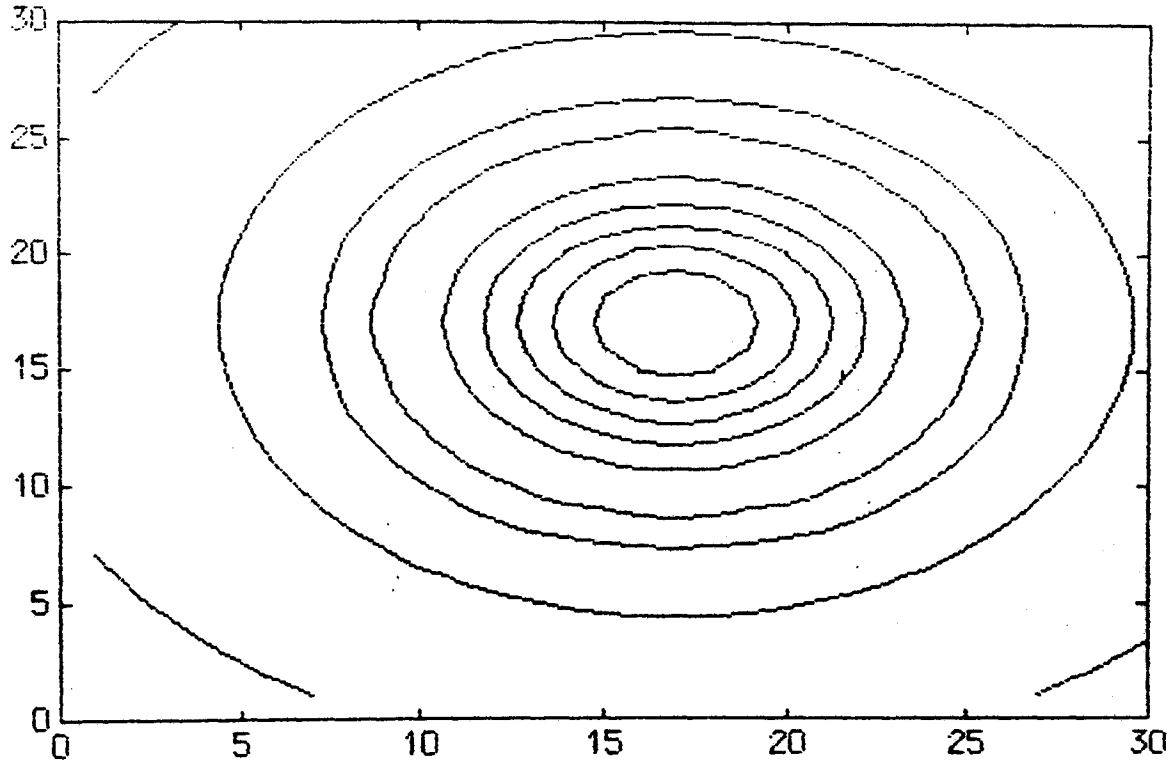


Figura 2.5

2.14 Impressão de gráficos

O MATLAB fornece impressões de gráficos através de gráficos "metafiles" (arquivos com estensão .MET) e um programa "graphics post processor" (GPP).

Os "metafiles" são gerados através do comando meta do MATLAB como no exemplo abaixo:

```
>> meta filename
```

O comando GPP utiliza os arquivos .MET para gerar arquivos que podem ser impressos por um dispositivo particular. GPP é chamado no DOS com o seguinte formato:

```
GPP filename /device [/f /p /op /ol /ot /cs /cc /t]
```

onde os argumentos são separados por espaços, os colchetes indicam argumentos opcionais, e os dispositivos podem ser os seguintes:

/dps	PostScript (Apple LaserWriter)
/deps	Encapsulated Postscript
/dpic	pic format (troff)
/dtek	Tektronix 4010/4014 output
/dtekc	Tektronix 4105/4110 output (color)
/dqms	QMS Lasergrafix Tek emulation
/dimg	Imagen Laserprinter
/dhpgl	HPGL (HP compatible plotters)
/djet	HP Laser Jet Plus (300 dpi)
/djet150	HP Laser Jet Plus (150 dpi)
/ddesk	HP Deskjet (300 dpi)
/depsd	epson 9 pin draft quality
/depsf	epson 9 pin final quality
/dmx	epson MX compatible 9 pin draft quality
/dlqd	epson 24 pin draft quality
/dlqf	epson 24 pin final quality
/dpen	Pen plot file
/dvga	VGA graphics display
/dega	EGA graphics display
/dcga	CGA graphics display
/dhgc	Hercules graphics display
/datt	AT&T 6300 graphics display

/f	output file name other than default
/p	pause after each plot
/op	portrait page orientation, half page
/ol	landscape page orientation
/ot	portrait page orientation, full page
/cs	simplex character font
/cc	complex character font
/t	output to standard output

Por exemplo, os comandos MATLAB

```
t = -50:.3:50;
y = sin(t)./t;
plot(t,y)
meta sincplot
plot(diff(y)./diff(t))
meta
```

criam um "metafile" chamado sincplot.met contendo 2 gráficos, um resultante da função sinc e outro de sua derivada. Para conseguir uma impressão do "metafile", o próximo passo é rodar o GPP. Por exemplo, no DOS

```
GPP sincplot /depsd
print sincplot.eps
```

Neste caso o arquivo resultante do comando GPP é sincplot.eps, pois o dispositivo de saída adotado foi uma impressora epson 9 pinos. A seguir podemos imprimir os gráficos através do comando print do DOS. Note que poderíamos usar esses comandos sem sair do MATLAB usando o operador !.

2.15 Fluxo de Controle

O matlab possui comandos como os encontrados na maioria das linguagens.

for variável = início:fim, comandos, end

for i = 1: n, x(i) = 0, end
atribui 0 a todos os elementos de x.

```
while expressão
        comandos
end

if condição
        comandos
elseif
      comandos
else
      comandos
end
```

break termina um loop

3. Redes Neurais

Modelos de redes neurais artificiais ou simplesmente redes neurais têm sido estudados há vários anos e no decorrer desses anos já receberam diversos nomes, tais como, modelos conexionistas, modelos de processamento distribuído paralelo, sistemas neuromórficos. Todos estes modelos tentam obter da máquina um desempenho que "imite" o comportamento humano, através de INTERCONEXÕES DE ELEMENTOS COMPUTACIONAIS SIMPLES, que "simulam" o nosso sistema nervoso biológico. A estrutura básica desse sistema é o neurônio que desempenha o papel de difusor de impulsos.

O modelo dos neurônios do sistema nervoso foi copiado para uma estrutura computacional, onde cada neurônio se torna um processador e a cada informação trocada entre esses processadores está associado um peso.

Pode-se definir então, redes neurais como sendo um sistema de interconexão de processadores, baseada no sistema nervoso, que atua após um estímulo externo.

Redes Neurais têm encontrado uma grande aplicação nas áreas de reconhecimento de padrões e imagens, processamento de sinais, controle, automação, otimização, etc.

Modelos de redes neurais são especificados pela topologia da rede, características dos nós (neurônios), regras de treinamento ou aprendizado. Estas regras baseadas nos valores iniciais dos pesos indicam como os pesos deverão ser modificados para melhorar o desempenho da rede. Os elementos computacionais (nós) usados numa rede neural são modelos não-lineares, tipicamente analógicos, que executam operações simples, tais como, calcular a soma ponderada de suas entradas e fornecer uma saída através de uma função não-linear denominada função de transferência. Um neurônio é caracterizado por um "threshold" ou "offset" θ e pelo seu tipo de função de transferência. Na figura 3.1 temos os 3 principais tipos de funções de transferência : hard limiters, threshold logic elements e sigmoidal nonlinearities.

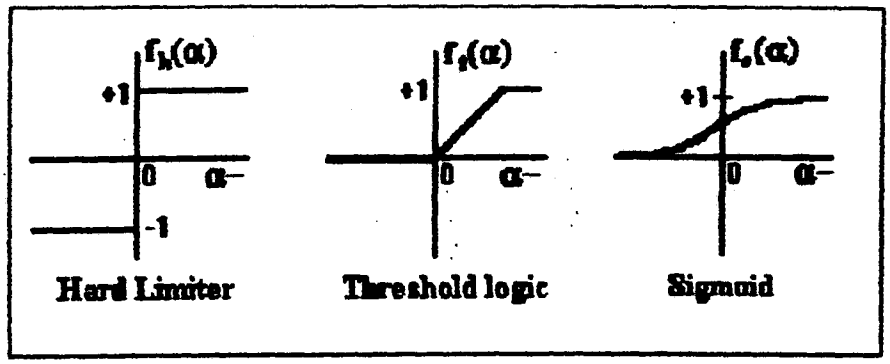


Figura 3.1 : 3 tipos de funções de transferência

3.1 PERCEPTRON

O percetron é o exemplo mais simples dentre as redes com entradas contínuas (redes cujos valores de entrada correspondem aos valores reais do problema). Esta rede causou muito interesse no início de seu desenvolvimento, pois era habilitada a reconhecer padrões simples. Um perceptron decide se uma entrada pertence a uma de duas classes (denotada por A e B, por exemplo).

Na figura 3.2 encontra-se o modelo de um Perceptron.

Trata-se de um único nó que recebe um vetor $X = (x_0, x_1, \ldots, x_{n-1})$ e fornece um saída Y, dada por:

$$Y = F\left(\sum_{i=1}^{n} w_i * x_i - \theta\right)$$

onde F é uma função de transferência "Hard Limit", w_i são os pesos e θ é um "threshold".

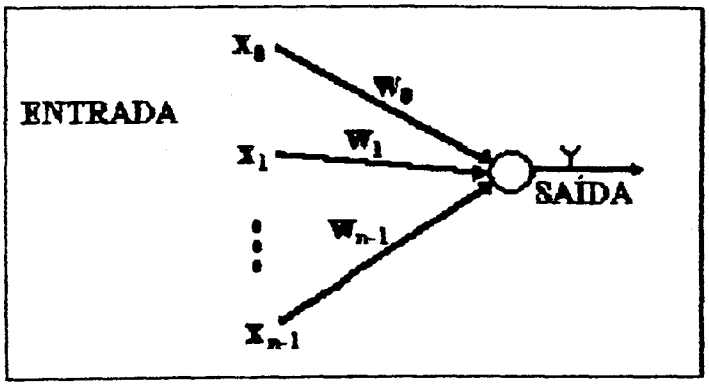


Figura 3.2 : Perceptron

Como a função de transferência é "Hard Limit", o valor de Y será 1 ou -1, que indicará se o vetor X pertence ou não a uma certa classe.

O estudo do perceptron serviu de base para o desenvolvimento das redes multi-camadas, pois sua estrutura é toda baseada no conceito inicial de perceptron.

3.2 Regra Delta

A Regra Delta, como outros algoritmos que seguem o gradiente de uma curva, é chamada uma regra de aprendizagem "gradient-descent".

Ela é utilizada para o ajuste dos pesos de uma rede neural da seguinte forma:

Primeiro cada neurônio deve fornecer uma saída através de uma função de transferência adequada. Então permite-se a comparação entre essa saída e a saída desejada, d, computando o valor do erro, E, para o padrão de entrada. E é computado subtraindo-se a saída atual, y, da desejada d : E = d - y.

Finalmente calcula-se a troca dos pesos usando-se a regra delta:

$$W' = W + (\beta * E * X) \mid X \mid^2$$

onde X e W são os vetores entrada e peso, respectivamente, e β é uma constante de aprendizagem, ou melhor, β é uma medida da velocidade de convergência do vetor peso para a posição do erro mínimo.

Note que a regra delta é uma equação vetorial. O erro E e a constante β são valores escalares; os outros elementos são vetores. Uma outra coisa importante para se notar sobre esta equação é que somente os pesos que estão nas entradas X são modificados.

3.3 Redes Multi-Camadas

Redes multi-camadas são redes "feed-forward" com uma ou mais camadas de neurônios entre os neurônios de entrada e saída (figura 3.3). Estas camadas adicionais contêm unidades escondidas ou neurônios que não estão diretamente conectados a ambos os neurônios de entrada e saída.

As camadas de uma rede multi-camadas têm diferentes nomes. Camadas as quais retornam as saídas da rede são chamadas camadas de saída. As outras camadas são chamadas escondidas ou ocultas.

Em geral, o número de neurônios nas camadas escondidas não é definido. Quanto mais neurônios, mais provável será de encontrarmos uma solução. Contudo, muitos neurônios significam aumento no tempo de treinamento e matrizes de pesos e vetores de "biases" muito grandes na solução.

A rede multi-camadas pode ser treinada com o algoritmo "Back-propagation". Este algoritmo é uma generalização do algoritmo da regra delta. Ele utiliza o método do gradiente para minimizar uma função erro, que é igual a diferença média entre as saídas desejadas e atuais da rede.

Esta generalização é dada pela fórmula :

$$W'' = W' + (\beta * E_p * X) \mid X \mid^2$$

$$E_p = (\Sigma (y_i - d_i)^2) / 2$$

onde d_i é o valor desejado, y_i o valor obtido pela rede, para cada padrão de entrada e E_p denota o erro relativo ao p-ésimo par: (x_i, d_i) apresentado a rede.

Um termo momentum pode ser acrescentado a fórmula acima, para acelerar a convergência da rede. Assim temos:

$$W'' = W' + (\beta * E_p * X) |X|^2 + \alpha * (W' - W)$$

O termo momentum é simplesmente a multiplicação de uma constante α pela mudança no vetor peso do neurônio de representação prévia do padrão de entrada. Então se o último peso mudado foi em uma direção, o termo momentum tenta fazer a próxima mudança de peso em mais ou menos a mesma direção. Dependendo dos valores dos parâmetros e da situação, o momentum pode ser suficiente para a atingir um mínimo local.

Considera-se que a rede atingiu a convergência quando o erro total, E_T, dado por :

$$E_T = (\Sigma E_p) < TOL$$

onde TOL é um valor arbitrário pequeno.

Redes multi-camadas treinadas com o algoritmo "Backpropagation" são comumente chamadas redes Backpropagation.

Redes Backpropagation treinadas tendem a dar respostas razoáveis quando são apresentadas entradas desconhecidas pela rede. Esta propriedade de generalização torna possível se treinar uma rede para um conjunto representativo de pares entrada/saída e conseguir bons resultados para novas entradas sem treinamento da rede para todos os possíveis pares entrada/saída.

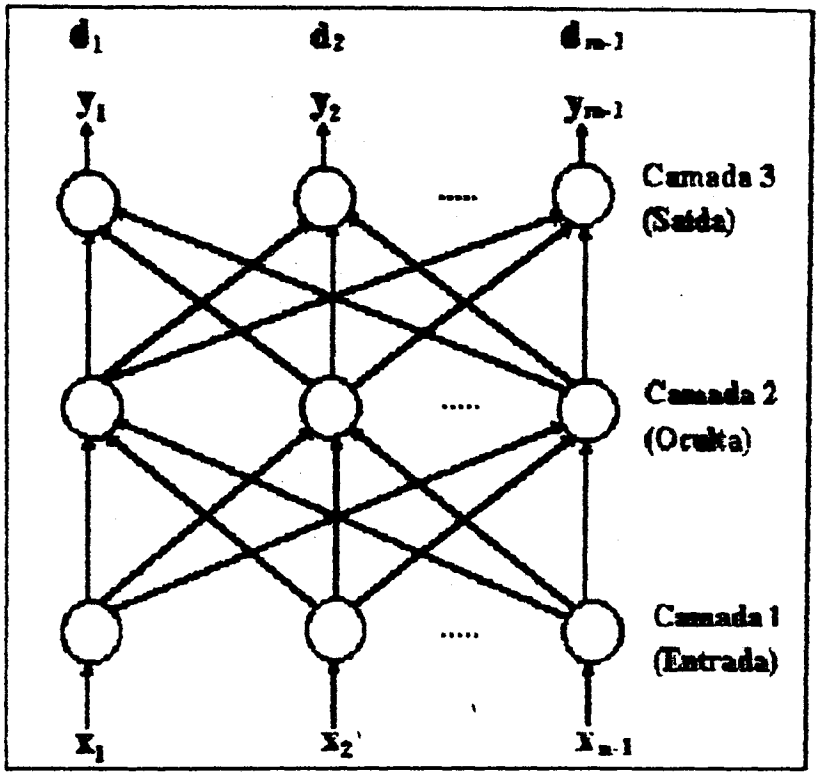


Figura 3.3 : Rede de 3 camadas

3.4 Algoritmo BACKPROPAGATION (ref [5])

1. Inicializar os pesos e "offsets" aleatoriamente com valores pequenos.

2. Apresentar o vetor de entrada $X = (x_0,x_1,...,x_{n-1})$ e especificar as saídas desejadas $D = (d_0,d_1,...,d_{m-1})$

3.Calcular as saídas $Y = (y_0, y_1, \ldots, y_{m-1})$ usando a função de transferência adotada.

4.Atualizar os pesos usando um algoritmo recursivo começando na última camada e voltando para a primeira camada escondida.

$$w_{ij}(t+1) = w_{ij}(t) + \beta * \delta_j * x_i$$

Nesta equação $W_{ij}(t)$ é o peso do nó i da camada escondida ou de uma entrada para o nó j no tempo t, X_i é ou uma saída do nó i ou uma entrada da rede, e δ_j é um termo erro para o nó j.

Se j está na camada de saída então:

$$\delta_j = y_j * (1 - y_j) * (d_j - y_j)$$

Se j está numa camada intermediária então:

$$\delta_j = x_j * (1 - x_j) * \sum_k (\delta_k * w_{jk})$$

A convergência é mais rápida, se um termo momentum é adicionado, e neste caso os pesos são dadas por:

$w_{ij}(t+1) = w_{ij}(t) + \beta * \delta_j * x_i + \alpha * (w_{ij}(t) - w_{ij}(t-1))$, onde $0<\alpha<1$.

5. Repetir os passos de 2) a 4) até a rede convergir.

A seguir temos um algoritmo Backpropagation detalhado usando uma rede de 3 camadas para o reconhecimento de letras. Sejam:

MAX1 - número de neurônios da primeira camada
MAX2 - número de neurônios da camada do meio
MAX3 - número de neurônios da última camada
W1 - vetor peso da camada de entrada para a do meio
W2 - vetor peso da camada do meio para a última
SAÍDA - vetor de saídas desejadas
KONE - vetor de entradas da rede
KTWO - vetor de saídas da camada do meio
KEND - vetor de saídas da última camada
ERRO_TOT - erro quadrático total
ERRO_KEND - vetor com o erro de cada neurônio da última camada
ERRO_KTWO - vetor com o erro de cada neurônio da camada do meio
W2_POST - nova matriz de pesos da camada do meio para a última
W1_POST - nova matriz de pesos da camada de entrada para a do meio
TOLERANCIA - valor do erro total desejado

Início

1. ---------- randomização dos pesos ----------

```
Para I ← 1 até MAX1 faça
    Para J ← 1 até MAX2 faç
        W1[I,J] ← (número aleatório entre "-1" e "1")
    Fim para
Fim para
Para I ← 1 até MAX2 faça
    Para J ← 1 até MAX3 faça
        W2[I,J] ← (número aleatório entre "-1" e "1")
    Fim para
Fim para
```

2.---------- inicializa saída padrão ----------

```
Para I ← 1 até MAX3 faça
   SAIDA[I] ← (saída padrão)
Fim para
```

3. ---------- inicializa entrada padrão ----------

```
Para I ← 1 até MAX1 faça
   KONE[I] ← (entrada padrão)
Fim para
```

4. ---------- leitura do modelo de entrada ----------

```
Para I ← 1 até MAX1 faça
   Leia KONE[I]
Fim para
```

5. ---------- cálculo da camada do meio ----------

```
      Para I ← 1 até MAX2 faça
    SOMA ← 0
   Para J ← 1 até MAX1 faça
      SOMA ← SOMA + (1. + EXP(-KONE[J])))*W1[J,I]
   Fim para
   KTWO[I] ← SOMA
Fim para
```

6. ---------- cálculo da última camada ----------

```
Para I ← 1 até MAX3 faça
   SOMA ← 0
   Para J ← 1 até MAX2 faça
      SOMA ← SOMA +(1. + EXP(-KTWO[J])))*W2[J,I]
   Fim para
   KEND[I] ← SOMA
Fim para
```

7. ---------- cálculo do erro total na última camada ----------

```
ERRO_TOT ← 0
Para I ← 1 até MAX3 faça
   ERRO1[I] ← SAIDA[I] - (1. + EXP(-KEND[I]))
   ERRO_TOT ← ERRO_TOT + ERRO1[I]*ERRO1[I]
Fim para
ERRO_TOT ← ERRO_TOT / 2
```

8. ---------- cálculo do erro de cada neurônio da última camada -------

```
Para I ← 1 até MAX3 faça
   ERRO_KEND[I] ← (1. / (1. + EXP(-KEND[I]))) *
                  ( 1 - (1. / (1. + EXP(-KEND[I])))) *
                  ERRO1[I]
Fim para
```

9. ---------- atualização dos pesos (camada do meio para a última) ------

```
Para I ← 1 até MAX2 faça
   Para J ← 1 até MAX3 faça
      W2_POST[I,J] ← W2[I,J] + (β * ERRO_KEND[J] *
                     (1. / (1.+ EXP(-KTWO[I])))) +
                     ( α * (W2[I,J] - W2_ANT[I,J]))
   Fim para
Fim para
```

{Obs : W2_ANT é o peso da iteração anterior}

10. ------ cálculo do erro de cada neurônio da camada do meio ----------

```
Para I ← 1 até MAX2 faça
   SOMA ← 0
   Para J ← 1 até MAX3 faça
      SOMA ← SOMA + W2[I,J] * ERRO_KEND[J]
   Fim para
   ERRO_KTWO[I] ← (1. / (1. + EXP(-KTWO[I]))) *
                  ( 1 - (1. + EXP(-KTWO[I]))) * SOMA
Fim para
```

11. -----Atualização dos pesos (primeira camada para a do meio)---------

```
Para I ← 1 até MAX1 faça
   Para J ← 1 até MAX2 faça
      W1_POST[I,J] ← W1[I,J] + (β * ERRO_KTWO[J] *
                     (1. / (1. + EXP(-KONE[I])))) +
                     (α * (W1[I,J] - W1_ANT[I,J]))
   Fim para
Fim para
```

{Obs : W!_ANT é o peso da iteração anterior}

12. ---------- ensino dos padrões para a rede ----------

```
LETRA ← FALSO
Repita
    Procedimento 5
    Procedimento 6
    Procedimento 7
    Se ERRO_TOT < TOLERANCIA então
       LETRA ← VERDADEIRO
   senão
          Procedimento 8
          Procedimento 10
          Procedimento 9
          Procedimento 11
     Fim se
Até LETRA
```

13. ---------- reconhecimento do modelo de entrada ----------

```
Procedimento 4
Procedimento 5
Procedimento 6
Se RECONHECEU então
    Imprime padrão reconhecido
Senão
    Imprime mensagem de não reconhecimento
Fim se
Fim
```

Inicialmente os pesos entre as camadas da rede são inicializados todos com valores aleatórios entre -1 e 1.

Em seguida, na fase de aprendizado, apresenta-se a rede o modelo a ser ensinado fornecendo seus valores como entrada na primeira camada, calculando a saída desta camada utilizando a função sigmoidal. As entradas da segunda camada são calculadas utilizando os pesos entre a primeira e segunda camadas e a saída da primeira. Novamente utiliza-se a função sigmoidal no cálculo da saída da segunda camada. Através, dos pesos entre a segunda e última camadas e também da saída da segunda, são calculadas as entradas da última camada e sua respectiva saída com a mesma função citada.

A saída resultante é comparada com a saída desejada do padrão a ser ensinado, e caso a rede tenha aprendido a reconhecê-lo, isto é, a rede tenha convergido, a rede passa para a fase de reconhecimento, senão, continuará a repetir o processo através da atualização dos pesos.

Para verificar a convergência, os módulos dos erros obtidos em cada neurônio da última camada, onde o erro é dado pela diferença entre o valor da saída desejada pela saída obtida em cada neurônio, são somados gerando um erro total. Se o erro total for menor que uma tolerância estipulada, a rede obteve convergência.

Caso o valor ainda seja menor que essa tolerância, o erro ocasionado na saída é projetado de volta para as camadas anteriores, e através destes erros, das saídas das camadas e dos pesos anteriores, os pesos são atualizados e os valores do padrão a ser ensinado é novamente apresentado à rede repetindo todo o processo até a sua convergência.

Ao terminar a fase de aprendizado, ou seja, quando todos os padrões já foram ensinados, passamos à fase de reconhecimento, cujo modelo a ser reconhecido é apresentado à rede. O que se busca nesta fase é a resposta da rede após a apresentação do modelo desconhecido (A rede agora, possui os pesos já adaptados resultantes do aprendizado). O padrão reconhecido será aquele que possui a sua saída parecida com a saída resultante da rede, que obtemos a partir do erro gerado entre as duas saídas (da rede e do padrão ensinado).

4 Redes Neurais no Ambiente MATLAB

Nesta seção apresentaremos como os modelos: PERCEPTRON e Redes Multi-Camadas são implementadas no ambiente MATLAB.

4.1 Modelo de Neurônio e Arquiteturas de Rede

O modelo de neurônio e as arquiteturas de redes neurais descritas a seguir são as adotadas no TOOLBOX/MATLAB.

4.1.1 Neurônio com uma entrada

Um neurônio com uma única entrada e sem "bias" é mostrado na figura 4.1(a). A entrada p é transmitida através de uma conexão que multiplica seu valor por um peso w, para formar o produto w*p. Aqui o produto w*p é o único argumento da função de transferência F, que produz a saída a. O neurônio na figura 4.1(b) tem um "bias" b, que é uma entrada adicional para a função de transferência F. O "bias" é mais ou menos como um peso, exceto que ele tem o valor constante 1.

"Biases" dão a rede mais graus de liberdade para aprender uma função desejada. Usando "biases" a rede tende a aumentar as chances de encontrar uma solução aceitável e também diminuir o número de ciclos de treinamento. Entende-se por ciclo a cada apresentação do conjunto de dados durante a fase de treinamento da rede.

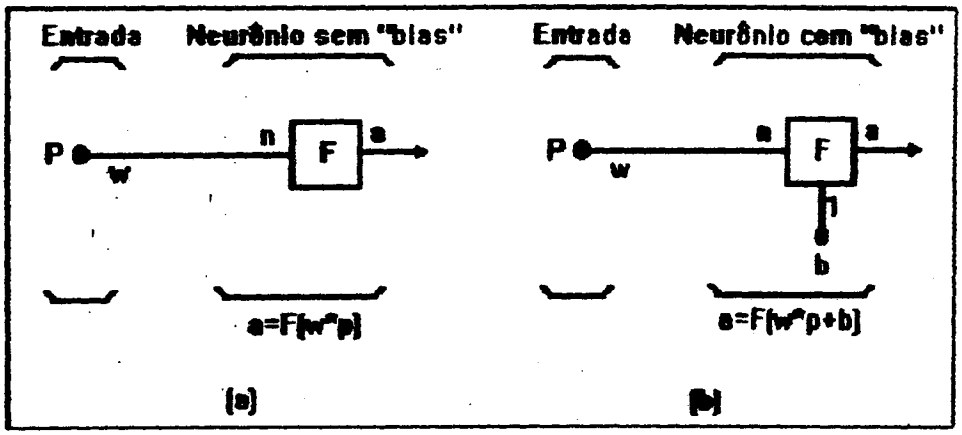


Figura 4.1

4.1.2 Funções de Transferência

Três das funções de transferência mais usadas são mostradas na figura 4.2.

A função de transferência "Hard Limit" usada no MATLAB limita a saída do neurônio em 0 ou 1. É frequentemente usada para criar neurônios que tomem decisões tais como classificação.

A função de transferência linear tem uma saída igual a sua entrada mais o "bias". Neurônios com esta função de transferência podem ser usadas como aproximações lineares.

A função de transferência "sigmoid" coloca entradas reais dentro do intervalo 0 a 1. Geralmente é usada em redes Backpropagation.

As funções de transferência na direita em cada um dos três casos mostrados, tem um "bias" e os da esquerda não. Você pode especificar se o "bias" é usado incluindo-o ou excluindo-o dos valores fornecidos pela função.

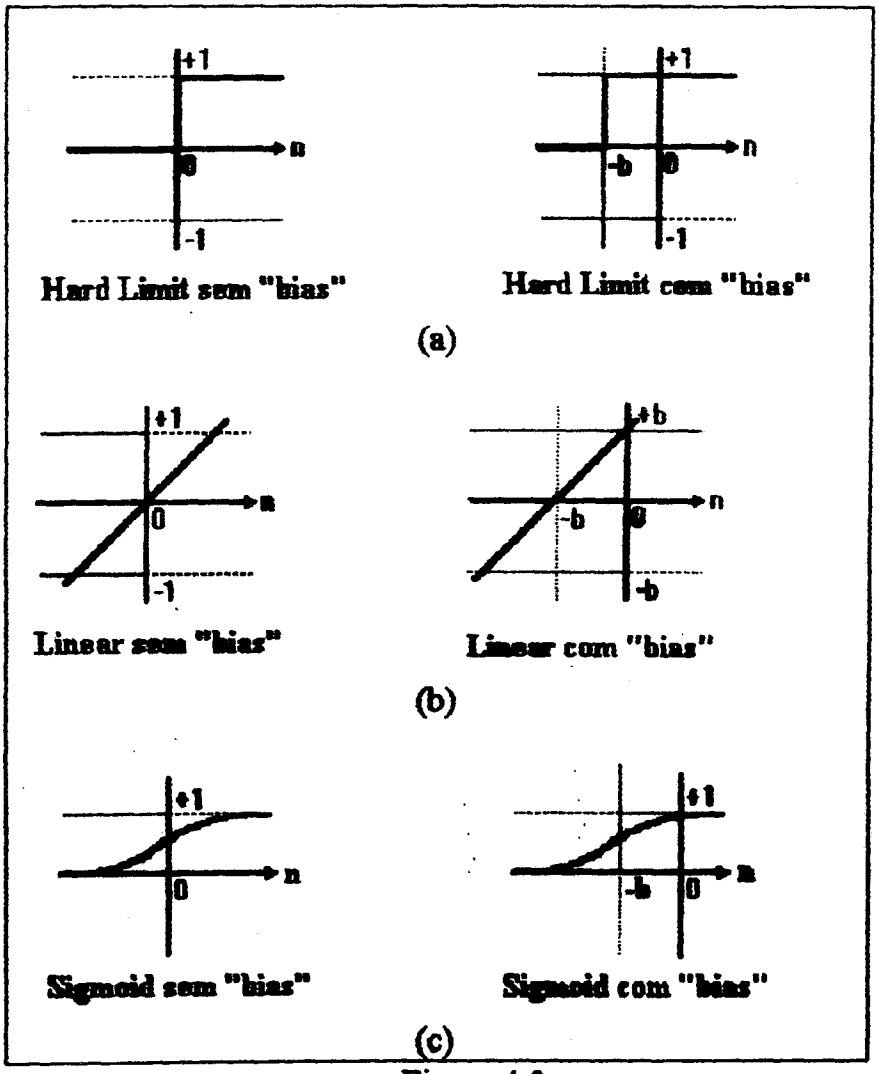


Figura 4.2

4.1.3 Neurônio com múltiplas entradas

Um neurônio com R entradas é mostrado na figura 4.3. Aqui as entradas individuais P(j), multiplicadas pelos elementos w(1,j) da matriz w, são somadas para formar as entradas para a função de transferência F. O neurônio tem um "bias" e uma saída a.

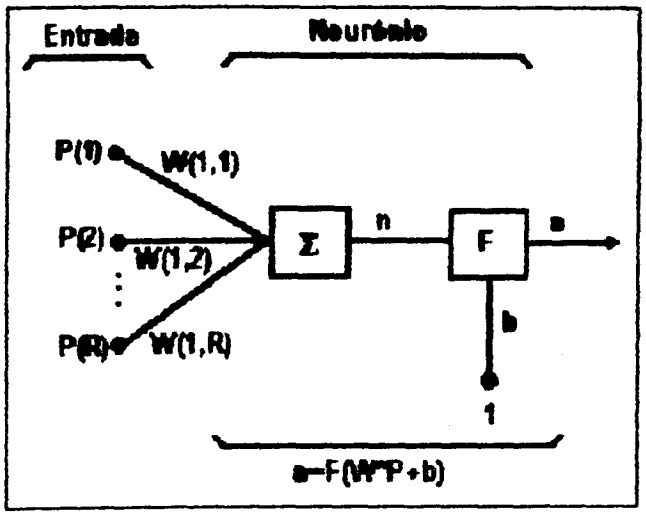


Figura 4.3

O vetor de pesos w(1,j) e o vetor de entrada P(j) podem ser representados como um vetor linha W e um vetor coluna P da seguinte maneira:

$$W = [\,W(1,1)\ W(1,2)\ \ldots\ W(1,R)\,]\ ;\quad P = \begin{bmatrix} P(1) \\ P(2) \\ \vdots \\ P(R) \end{bmatrix}\text{, respectivamente.}$$

4.1.4 Uma camada de neurônios

Uma rede de uma única camada com R entradas e S neurônios é mostrada na figura 4.4.

Neste caso, cada elemento da entrada do vetor de entrada P é conectado a cada neurônio através da matriz peso W. Cada um dos S neurônios tem uma somatória, e as saídas formam um vetor N de S elementos. A entrada da função de transferência da rede é a soma de suas entradas w*p apropriadas e o "bias" b. Finalmente, as saídas das camadas de neurônios formam um vetor A.

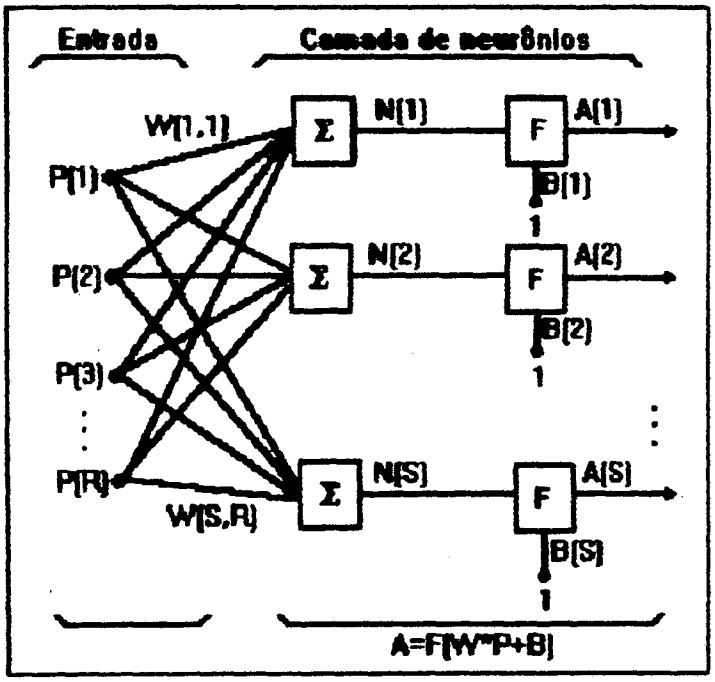


Figura 4.4

Obs : No ambiente MATLAB as entradas formam uma camada denominada camada de entrada e não uma camada de neurônios. No entanto, muitos autores adotam a camada de entrada como sendo uma camada de neurônios, como pode ser visto na figura 3.3. Neste caso as entradas são apresentadas diretamente aos neurônios da primeira camada.

4.1.5 Múltiplas camadas de neurônios

No MATLAB cada camada tem uma matriz peso W, um vetor "bias" B, um vetor N (w*p), e um vetor saída A. Para distinguir entre as matrizes peso, vetores saída, etc..., para cada destas camadas, colocaremos o número da camada nos nomes de cada uma destas variáveis. Por exemplo, a matriz peso e o vetor saída para a primeira camada são denotados como W1 e A1, para a segunda camada estas variáveis são denotadas como W2 e A2, etc...

Uma rede de 2 camadas é mostrada na figura 4.5. Ela tem uma camada de saída(camada 2) e 1 camada escondida (camada 1).

Redes múltiplas camadas são bem mais eficientes. Por exemplo, uma rede de duas camadas, a primeira com neurônios cuja função de transferência é "sigmoid" e a segunda linear, pode ser treinada para aproximar qualquer função prática arbitrariamente bem.

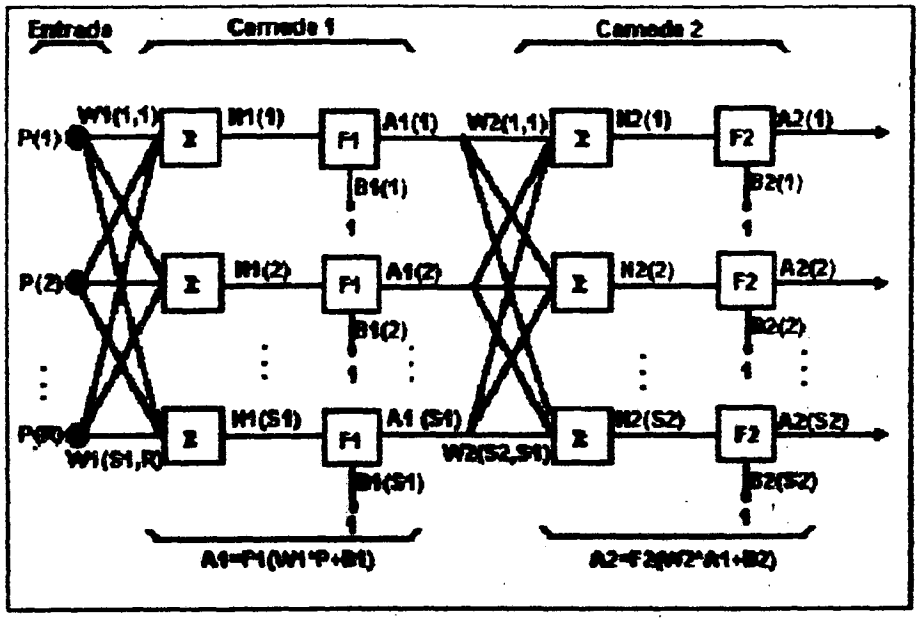


Figura 4.5

4.1.6 Batching : Múltiplos vetores de entrada.

Comumente um vetor de entrada é apresentado a rede, as saídas são calculadas e um algoritmo é aplicado para determinar as trocas dos pesos. Contudo você pode aplicar Q vetores de entrada simultaneamente, e conseguir a resposta da rede para cada um deles. Tal operação em "batching" é mais eficiente.

Um batch de Q vetores de entrada de R elementos aplicado para uma rede de S neurônios é representado da seguinte maneira:

$$P = \begin{bmatrix} P(1,1) & P(1,2) & \dots & P(1,Q) \\ P(2,1) & P(2,2) & \dots & P(2,Q) \\ \vdots & \vdots & & \vdots \\ P(R,1) & P(R,2) & \dots & P(R,Q) \end{bmatrix}$$

$$A = \begin{bmatrix} A(1,1) & A(1,2) & \dots & A(1,Q) \\ A(2,1) & A(2,2) & \dots & A(2,Q) \\ \vdots & \vdots & & \vdots \\ A(S,1) & A(S,2) & \dots & A(S,Q) \end{bmatrix}$$

A rede produzirá um vetor saída A(:,j) para cada vetor de entrada P(:,j). Assim a aplicação de Q vetores de entrada produzirão uma matriz saída A com S linhas e Q colunas.

Note que P(:,n) retorna a n-ésima coluna da matriz P e P(n,:) retorna a n-ésima linha da matriz P.

4.2 Implementando o PERCEPTRON no ambiente MATLAB

A seguir veremos todos os procedimentos necessários para implementação do perceptron no ambiente MATLAB.

4.2.1 Inicialização

A função **rands** do MATLAB é utilizada para inicializar os pesos e "biases" de uma rede com pequenos valores positivos ou negativos. Para uma rede com 4 entradas e 1 camada de 8 neurônios, o código de inicialização é:

```
R = 4;
S = 8;
[W,B] = rands(S,R)
```

R é o número de entradas da rede e S o número de neurônios na única camada da rede. W e B representam os pesos e "biases" da rede respectivamente.

A função **hardlim** aplica a função de transferência "Hard Limit" (figura 3.2(a)) para todos os n elementos da matriz N.

Quando um "bias" não é usado, o perceptron produz 1 se a entrada da rede é maior que 0, caso contrário, produz 0. Se um "bias" é usado, a função de transferência é deslocada uma quantidade b(bias) para a esquerda. Neste caso, o perceptron produz 1 se a entrada da rede é maior que b e 0 caso contrário. Normalmente "biases" são usados e a linha de código no MATLAB para calcular a saída de um perceptron é:

```
a = hardlim(W*P+b);
```

Quando um conjunto de vetores de entrada ("batch") são apresentados para o peceptron, o "bias" é passado como segundo argumento ao invés de ser adicionado ao primeiro argumento.

```
a = hardlim(W*P,b);
```

4.2.2 Regra de aprendizagem

A rede perceptron é treinada para responder a cada vetor de entrada com um correspondente vetor saída com valores 0 ou 1. A regra de aprendizagem do perceptron é aplicada a cada neurônio a fim de calcular as novas linhas da matriz peso W e os novos "biases".

As trocas dos elementos da matriz peso e "biases" podem ser calculados usando a função **learnp**, que retorna trocas para a matriz peso e para os "biases" dado o vetor de entrada P, o vetor saída A (calculado através da função de transferência) , e o vetor saída desejado T.

```
[dW,dB] = learnp(P,A,T);
```

Esta função também pode ser chamada com apenas um argumento de entrada dW para conseguir apenas novos pesos quando "biases" não são usados.

Outra característica da função de aprendizagem learnp é que P, A, e T, podem ser usados com treinamento em "batch" da rede, apresentando vários vetores de entrada simultaneamente.

O código seguinte apresenta um conjunto de vetores de entrada para um perceptron e atualiza os pesos e "biases" utilizando a regra de aprendizagem do perceptron e o vetor de saídas desejadas T.

```
A = hardlim(W*P,B);
[dW,dB] = learnp(P,A,T);
W = W + dW;
B = B + dB;
```

4.2.3 Treinamento

Vetores de um conjunto de treinamento são apresentados para a rede um após o outro. Se a saída da rede está correta nenhuma mudança é feita. Por outro lado, os pesos e "biases" são atualizados usando a regra de aprendizagem do perceptron. Uma passagem completa através de todos os vetores de treinamento será chamado um ciclo. Quando um ciclo completo tenha ocorrido sem erro, o treinamento termina. Depois disso qualquer vetor de entrada de treinamento poderá ser apresentado a rede e ela irá responder com o vetor saída correto. Se um vetor P_i fora do conjunto de treinamento é apresentado a rede, a rede tenderá a exibir um generalização respondendo com uma saída parecida à saída correspondente ao vetor de entrada parecido com o vetor P_i.

Os vetores de entrada podem ser apresentados simultaneamente ("batch"). A rede é considerada completamente treinada apenas se todas as saídas associadas com os vetores de entrada estão corretas. Por outro lado o treinamento continua.

O código para treinamento de perceptrons utilizando "batch" é mostrado a seguir. Muitos desses detalhes estão dentro da função **trainp** do MATLAB.

```
TP = [disp_freq max_epoch]
[W,B,epochs] = trainp(W,B,P,T,TP)
```

O argumento TP é um vetor linha de dois valores: o número de ciclos em que o progresso do treinamento da rede é mostrado, e o máximo número de ciclos para treinamento.

O seguinte código demonstra como trainp opera. Quando dado um conjunto de vetores de entrada P, o correspondente conjunto de vetores saída T, o número de ciclos em que o progresso do treinamento deverá ser mostrado (disp_freq), e o número máximo de ciclos para treinamento (max_epoch), trainp treina uma rede com a regra de aprendizagem do perceptron. Primeiro, é claro, as variáveis W e B devem ser inicializadas com valores randômicamente gerados.

```
% FASE DE APRESENTAÇÃO:
A = hardlim(W*P,B);
for epoch=1:max_epoch
        % FASE DE CHECAGEM
        if all(A = = T)
                epoch=epoch-1;
                break;
        end
        % FASE DE APRENDIZAGEM
        [dW,dB] = learnp(P,A,T);
        W = W + dW;
        B = B +dB;
        %FASE DE APRESENTAÇÃO
        A = hardlim(W*P,B);
end
```

O processo de treinamento pode ser visto como tendo 3 fases:

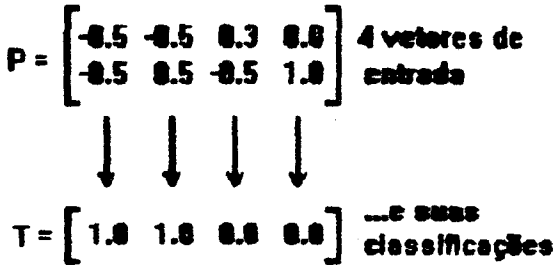
1. FASE DE APRESENTAÇÃO : Apresenta-se a matriz de vetores de entrada a rede e calcula-se a matriz de vetores saída através da função de transferência adotada.

2. FASE DE CHECAGEM : Verifica-se se cada vetor saída calculado A é igual ao vetor saída desejado T. Se todos os vetores entrada geraram os vetores saída corretos para um conjunto específico de pesos e "biases", ou após um número máximo de épocas , o treinamento pára.

3. FASE DE APRENDIZAGEM : Ajusta-se pesos e "biases" com a regra de aprendizagem perceptron e retorna-se a fase 1.

4.2.4 Exemplo

Considere um simples problema de classificação com quatro vetores de entrada divididos em duas classes. Dois dos vetores de entrada são associados com a saída 1 e os outros dois com a saída 0.



Aqui está o código necessário para definir o problema:

```
P = [-0.5  -0.5  +0.3  +0.0;
     -0.5  +0.5  -0.5  +1.0];
T = [1 1 0 0];
```

Os vetores de entrada para o problema são mostrados na figura 4.6. Os vetores entrada que tem saída 0 são marcados com um 'o' e os vetores entrada cuja saída é 1 são marcados com '+'.

Uma rede com duas entradas e um neurônio é suficiente para resolver o problema.

O número de entradas e neurônios da rede devem ser definidos e os pesos e "biases" iniciais devem ser encontrados antes do treinamento.

```
[R,Q] = size(P);
[S,Q] = size(T);
W = rands(S,R);
b = rands(S,1);
```

A seguir, o número de ciclos para mostrar o progresso da rede (disp_freq) o número máximo de ciclos (max_epoch) necessários para o treinamento devem ser inicializados. Abaixo temos 20 ciclos.

```
disp_freq = 1;
max_epoch = 20;
TP = [disp_freq max_epoch];
```

Agora a rede está pronta para ser treinada.

[W,b,epochs] = trainp(W,b,P,T,TP);

O gráfico na figura 4.7 mostra os vetores entrada rotulados com suas saídas. Uma linha tracejada mostra como uma rede divide inicialmente seu espaço de entradas. Como mostrado, os o's e +'s não estão divididos corretamente. Este gráfico mostra como a rede ajusta seus pesos e "biases" para classificar o espaço de entradas até encontrar uma solução. As linhas pontilhadas mostram os estados intermediários da rede, enquanto a linha sólida mostra a solução final.

O gráfico na figura 4.8 mostra como os dois pesos e "biases" da rede são trocados com o progresso do treinamento.

Estes valores finais são obtidos após 4 ciclos de treinamento:

W = [-2.6161 -0.6922]; B = [-0.1680];

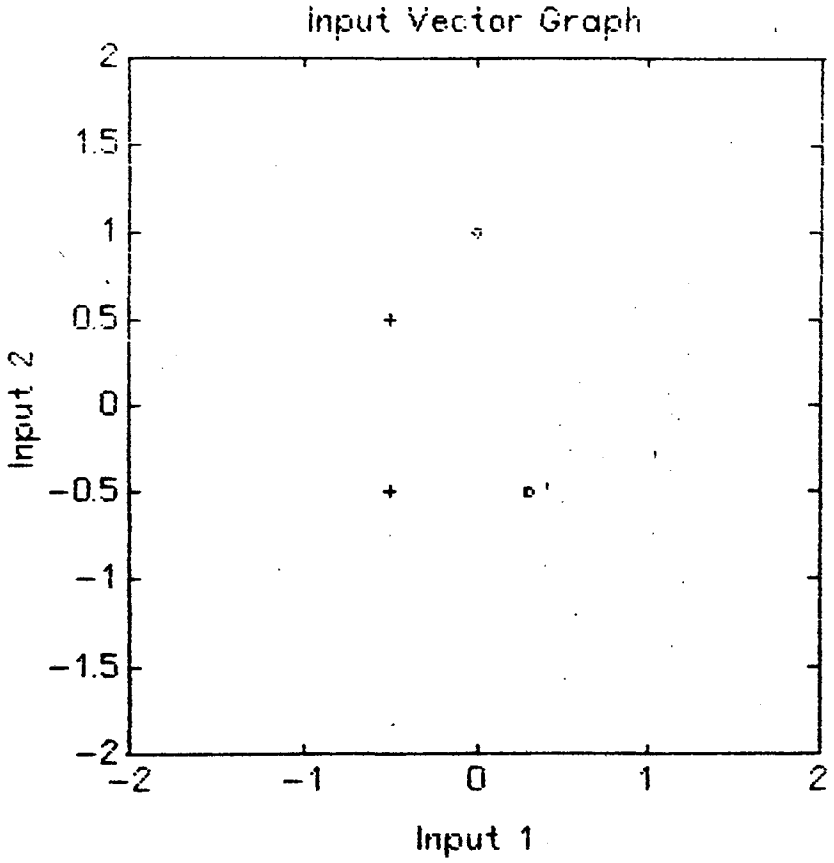


Figura 4.6

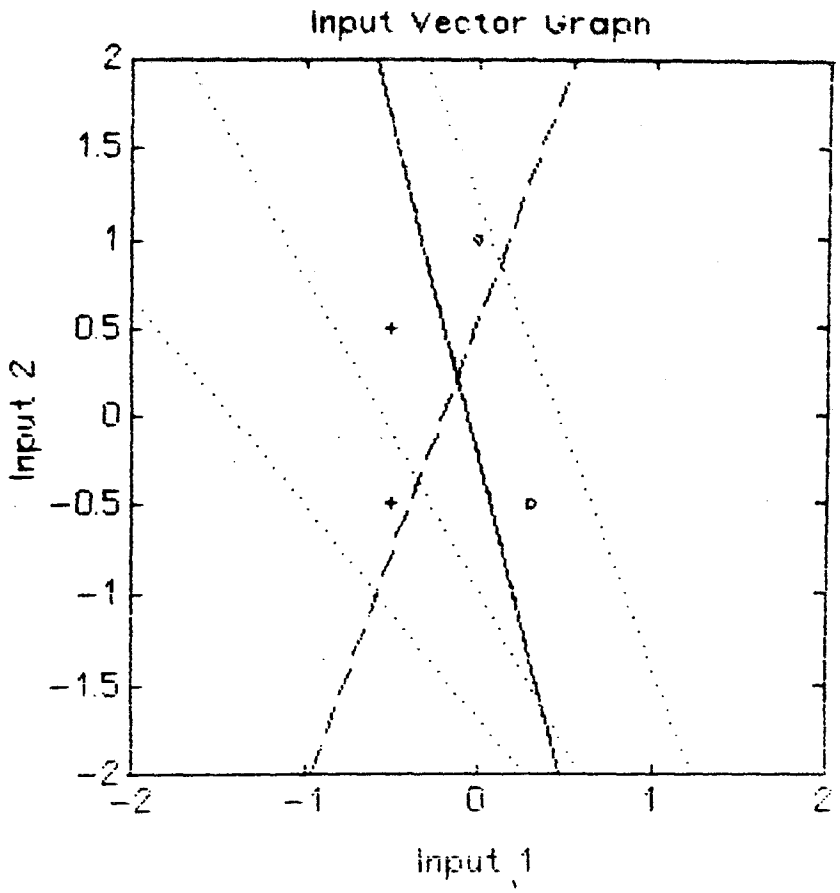


Figura 4.7

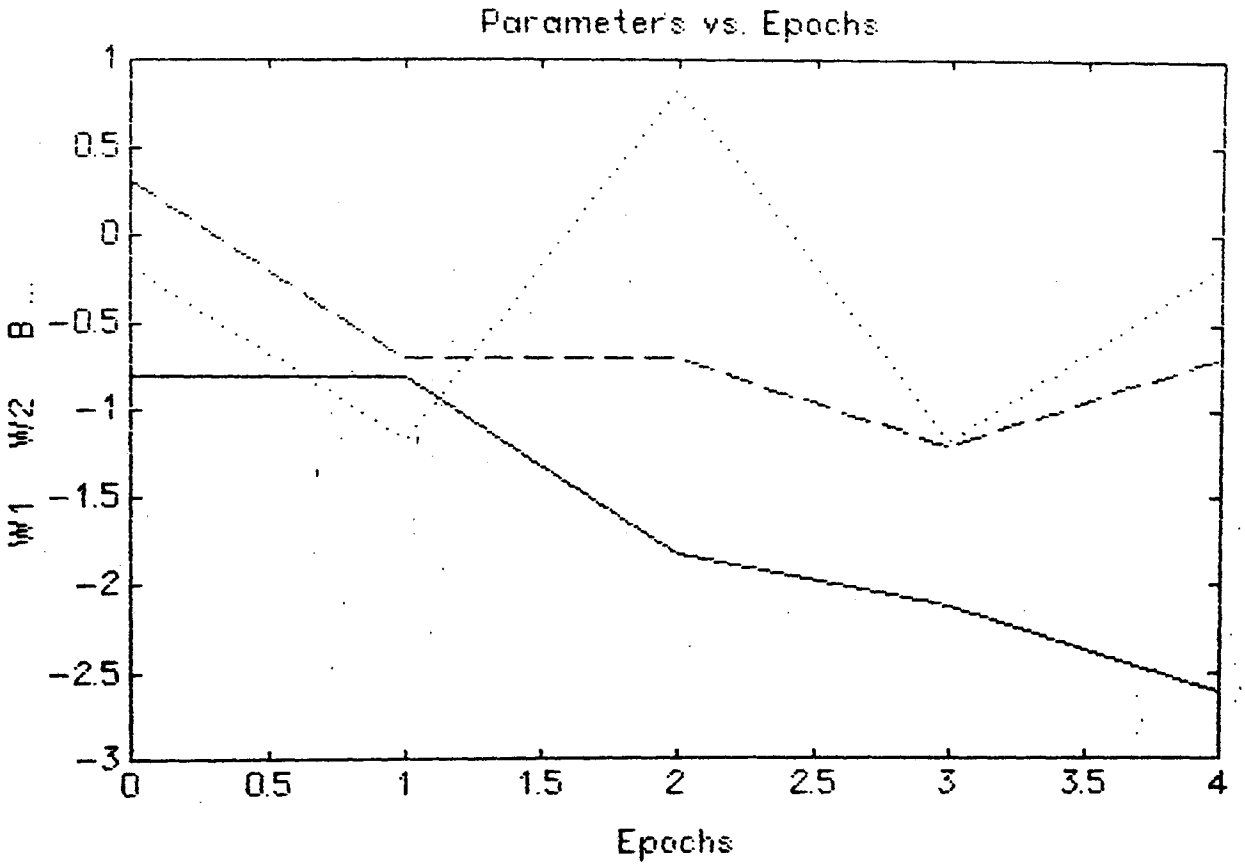


Figura 4.8

Perceptrons podem encontrar uma solução diferente se eles começam com diferentes condições iniciais. Por exemplo, a rede foi treinada novamente com diferentes condições iniciais. A rede resolveu novamente o problema, mas obteve diferentes pesos e "biases":

W = [-2.1642 0.0744];
B = [-0.6433];

A segunda solução divide o espaço de entradas em espaços diferentes.

4.3 Implementando BACKPROPAGATION no ambiente MATLAB

A seguir descrevemos como implementar o algoritmo backpropagation utilizando o TOOLBOX / MATLAB.

4.3.1 Inicialização

A matriz de pesos iniciais para a rede backpropagation é criada com elementos randômicos entre -1 e 1. Tais elementos podem ser obtidos usando a função **rands** do MATLAB.

A seguir temos o código de inicialização para uma rede de duas camadas (figura 4.5) com 2 entradas, 4 neurônios na camada oculta e 1 neurônio na camada de saída. Para o treinamento serão utilizados os seguintes vetores entrada/saída:

$(0\ 0) \rightarrow 0$
$(0\ 1) \rightarrow 1$
$(1\ 0) \rightarrow 1$
$(1\ 1) \rightarrow 0$

Na fase de inicialização as matrizes P (matriz de entradas da rede) e T (matriz de saídas) devem ser definidas. Neste caso:

$$P = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$ 4 vetores de entrada

$$T = \begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}$$ ...e suas saídas

A primeira coluna de P corresponde ao primeiro vetor de entrada apresentado à rede, a segunda coluna corresponde ao segundo vetor de entrada e assim por diante. Do mesmo modo, cada elemento de T corresponde a uma saída desejada da rede. As linhas de código para definir P e T são:

```
P = [0 0 1 1;
     0 1 0 1];
T = [0 1 1 0];
```

O restante do código de inicialização é:

```
[R,Q] = size(P);
S1 = 4;
[S2,Q] = size(T);
[W1,B1] = rands(S1,R);
[W2,B2] = rands(S2,S1);
```

A função **size** retorna o número de linhas e o número de colunas de uma determinada matriz.

R e Q recebem o número de linhas e o número de colunas da matriz P, ou seja, R recebe o número de entradas da rede (2 entradas) e Q recebe o número de vetores de entrada da rede (4 vetores). S1 contém o número de neurônios na camada escondida e S2 o número de neurônios na última camada (1 neurônio). O número de neurônios na saída da rede é obtido pelo número de saídas da rede.

Backpropagation frequentemente usa a função de transferência "sigmoid" (figura 4.2(c)).

A função **logsig** disponível no MATLAB aplica a função de transferência "sigmoid" para todos os n elementos da matriz N.

Se a última camada da backpropagation tem neurônios cuja função de transferência é "sigmoid", então as saídas da rede são limitadas para uma pequena extensão (entre 0 e 1). Se os neurônios têm função de transferência linear as saídas da rede podem assumir quaisquer valores.

Para simplificação chamaremos neurônios sigmóides, os neurônios que têm como função transferência a função "sigmoid". Analogamente, chamaremos neurônios lineares e neurônios tan-sigmóides, os neurônios cujas funções de tranferência são as funções linear e tan-sigmoid respectivamente.

Por exemplo:

```
A = logsig(W*P,B);
```

Esta linha de código calcula as saídas A geradas pela matriz P, composta por vários vetores de entrada, para neurônios "sigmoides" com peso W e "bias" B.

Alternativamente, redes Backpropagation podem usar as funções de tranferência "tan-sigmoid" (**tansig**) e linear (**purelin**).

4.3.2 Regra de aprendizagem

A regra de aprendizagem backpropagation é usada para treinar redes multi-camadas, não lineares, para realizar aproximação de funções, associação e classificação de padrões.

Derivadas do erro quadrático total da rede com respeito a uma entrada da rede são usadas por backpropagation para ajustar os pesos e "biases" a fim de diminuir o erro quadrático total. As derivadas de erro, ou vetores delta, de uma camada de saída são calculados do vetor erro da rede E. Os vetores delta das camadas escondidas são calculados dos vetores deltas das camadas seguintes. Esta retropropagação de vetores delta originou a técnica backpropagation.

Estes vetores são calculadados com funções **deltalin, deltatan** e **deltalog**, cujas descrições são apresentadas a seguir.

deltalog(A,E) retorna o vetor delta para uma camada de saída de neurônios cuja função de transferência é "sigmoid". A é o vetor saída e E é o vetor erro. **deltalog(A,D,W)** retorna o vetor delta para uma camada escondida de neurônios sigmóides. D é o vetor delta da camada seguinte e W é a matriz peso. Calcular um vetor delta do vetor delta da camada seguinte é referido como uma "backpropagation" de deltas.

deltalin(A,E) retorna o vetor delta para uma camada de saída de neurônios lineares .

deltalin(A,D,W) retorna o vetor delta para uma camada oculta de neurônios lineares .

deltatan(A,E) retorna o vetor delta para uma camada de saída de neurônios tan-sigmóides .

deltatan(A,D,W) retorna o vetor delta para uma camada oculta de neurônios tan-sigmóides .

As trocas a serem feitas nos pesos e "biases" das camadas são calculadas usando estes vetores deltas D, e o vetor P da camada de entrada. Isto é encontrado na função **learnbp**.

learnbp(P,D,lr) retorna uma matriz de trocas de pesos para uma camada dado o vetor entrada P, vetores delta D, e uma taxa de aprendizagem lr. A taxa de aprendizagem lr corresponde ao β do item 3.2.

[dW,dB] = learnbp(P,D,lr) retorna uma matriz de trocas de pesos e um um vetor de trocas de "bias" para uma camada.

4.3.3 Treinamento

Para treinar uma rede, vetores de entrada e correspondentes vetores de saída são apresentados a rede e os vetores erro são calculados. O erro quadrático total é então encontrado. Se o erro quadrático total para todos os vetores de treinamento é menor que o erro desejado, o treinamento pára. Por outro lado o vetor delta é calculado para a camada saída e outros vetores deltas são retropropagados através da rede.

Finalmente os pesos são atualizados usando a regra de aprendizado backpropagation e o processo se repete. Se os erros da rede são pequenos o bastante, vetores podem ser apresentados a rede e ela irá responder com o vetor saída correto. Se um vetor fora do conjunto de vetores de entrada de treinamento é apresentado para uma rede bem treinada, a rede tentará exibir uma generalização no seu resultado saída.

O processo de treinamento pode ser visto como tendo 4 fases:

1. FASE DE APRESENTAÇÃO : Apresenta-se um vetor de treinamento de entrada a rede e calcula-se o vetor saída através da função de transferência utilizada.

2. FASE DE CHECAGEM : Os vetores erro da rede e o erro quadrático total para o vetor entrada são calculados. Se o erro quadrático total para todos os vetores de treinamento é menor que o erro desejado ou se seu número máximo de ciclos foi alcançado o processo pára.

3. FASE BACKPROPAGATION : Calcula-se o vetor delta para a camada saída usando o vetor de saídas desejado. Então retropropaga-se o vetor delta para as camadas precedentes.

4. FASE DE APRENDIZAGEM : Calcula-se a nova matriz peso e o novo vetor bias para cada camada. Então retorna-se a fase 1.

Todos esses detalhes são realizados pela função de treinamento **trainbp**, a qual é usada para treinar uma rede backpropagation multi-camadas. Primeiro o problema deve ser definido criando uma matriz de vetores de entrada P e suas correspondentes saídas T, e os pesos W e "biases" B inicializados randômicamente. Uma string F contendo o nome da função de tranferência (**logsig, purelim, tansig**) da camada deve também ser fornecida. Então uma rede com uma única camada pode ser treinada para um número desejado de ciclos max_epoch, um nível de erro aceitável err_goal, usando a taxa de aprendizagem desejada. O erro quadrático total e o número de ciclos correntes são mostrados através de um gráfico a cada disp_freq ciclos para nos manter atualizados do progresso.

```
TP = [disp_freq max_epoch err_goal lr];
[W,B,epochs,TR] = trainbp(W,B,'F',P,T,TP)
```

O argumento TP da função trainbp é um vetor linha com 4 valores :

- disp_freq - o número de ciclos em que o erro quadrático total é mostrado.
- max_epoch - o número máximo de ciclos de treinamento.
- err_goal - o nível de erro aceitável para o erro quadrático total.
- lr - a taxa de aprendizagem. Tipicamente 0.01, 0.001, ou menor (lr corresponde ao β do item 3.2).

O treinamento continua até que o número máximo de ciclos é alcançado ou o erro máximo tenha sido passado. Então os novos pesos e "biases" são retornados junto com o número de ciclos de treinamento e um vetor linha TR. TR fornece um registro de treinamento. Neste caso, ele contém o erro quadrático total da rede através do treinamento.

A função de treinamento trainbp também pode ser usada para treinar redes "feed-forward" com duas ou três camadas. Para uma rede com duas camadas trainbp é chamado como se segue:

```
[W1,B1,W2,B2,epochs,TR] =
      trainbp(W1,B1,'F1',W2,B2,'F2',P,T,TP)
```

Aqui W1, B1, e F1 especificam os pesos, biases e funções tranferências da primeira camada. As variáveis W2, B2, and F2 definem a segunda camada. O treinamento ocorre como uma camada simples exceto que pesos e biases para duas camadas são retornados.

Aqui está o código para o treinamento de uma rede de três camadas com trainbp:

```
[W1,B1,W2,B2,W3,B3,te,TR]
      = trainbp(W1,B1,'F1',W2,B2,'F2',W3,B3,'F3',P,T,TP)
```

A taxa de aprendizagem especifica o tamanho de trocas que são feitas nos pesos e "biases" a cada ciclo. Taxas de aprendizagem pequenas resultam em longos tempos de treinamento mas garantem que os valores da rede não saltem sobre "vales" na superfície de erro que levariam a um valor aceitável de erro.

4.3.4 Exemplos

Exemplo 1:

Temos aqui um problema simples com dois vetores de entrada e dois vetores de saída:

```
P = [-3.0 2.0];
T = [ 0.4 0.8];
```

A rede terá uma entrada porque cada vetor de entrada tem apenas um elemento. Do mesmo modo, como os vetores de saída tem apenas um elemento, a camada de saída da rede terá apenas um neurônio. Para manter esse exemplo simples a camada de saída da rede será a única camada. A função de transferência usada será a função "sigmoid".

Para estabelecer o número de entradas e o número de neurônios na camada de saída da rede no MATLAB, a linha de código deve ser a seguinte:

```
[R,Q] = size(P);
[S,Q] = size(T);
```

Neste caso, R (número de entradas da rede) e S (número de neurônios na última camada) são iguais a 1 e Q (número de vetores de entrada/saída) é igual a 2. Variando os valores do "bias" de -4 a 4 e variando W através do mesmo intervalo, o erro quadrático total para cada combinação dos pesos e "biases" pode ser calculado para os vetores de entrada definidos e saídas correspondentes. Assim que todos os erros tenham sido calculados, gráficos como os da figura 4.9 e 4.10 podem ser gerados. O trabalho de treinamento da rede começa inicializando os pesos e "biases" da rede com pequenos valores randômicos. Os valores iniciais usados neste exemplo são representados pelo '+' na figura 4.10. Eles são gerados com:

```
[W,B] = rands(S,R);
```

Parâmetros de treinamento devem ser fornecidos para mostrar o progresso do treinamento (disp_freq), limitar o número de ciclos de treinamento (max_epoch), definir um erro aceitável (err_goal), e estabelecer uma taxa de aprendizagem (lr).

```
disp_freq = 1;
max_epoch = 100;
err_goal = 0.001;
lr = 1;
```

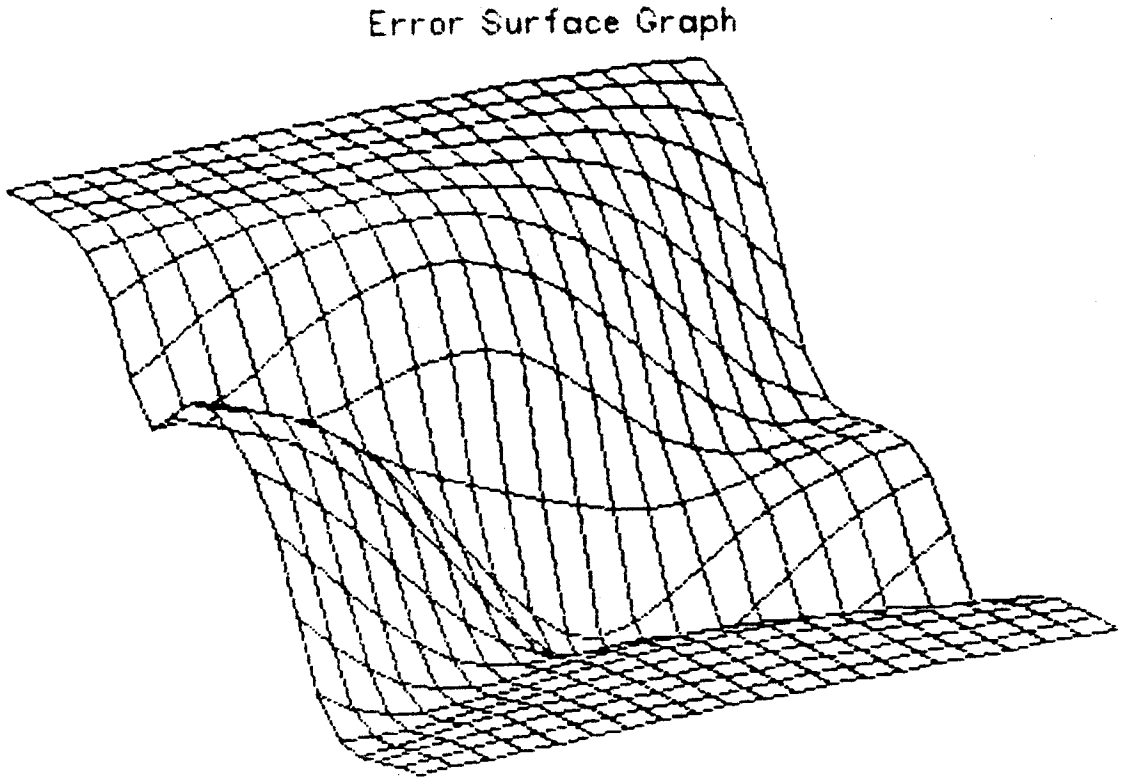


Figura 4.9

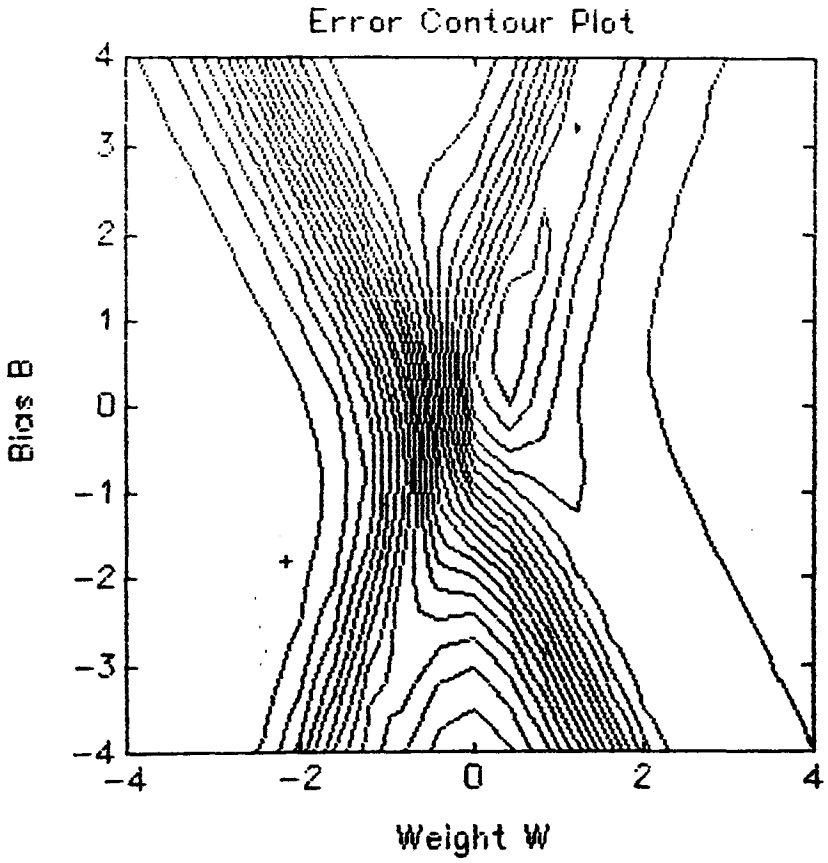


Figura 4.10

Agora a rede está pronta para ser treinada.

```
TP = [disp_freq max_epoch err_goal lr];
[W,B,epochs,TR] = trainbp(W,B,'logsig',P,T,TP);
```

O gráfico da figura 4.11 mostra como a rede muda de seus valores iniciais para uma solução.

O registro de treinamento TR contem os erros da rede através do treinamento. Estes erros podem ser mostrados através da seguinte linha de código.

```
plotem(TR)
```

O erro associado com cada vetor saída desejado pode ser mostrado:

```
A = logsig(W*P,B);
barrer(A,T)
```

Os dois gráficos são mostrados nas figuras 4.12 e 4.13. O gráfico da figura 4.12 mostra o erro da rede através do treinamento e o gráfico da figura 4.13 é um gráfico de barras dos valores finais do erro quadrático total associado com cada par de vetores entrada/saída.

O treinamento pára em 36 ciclos quando o erro quadrático total é 0.00092, que é menor que o erro aceitável de 0.001. Aqui estão os valores finais:

```
W = 0.3350;
B = 0.5497;
```

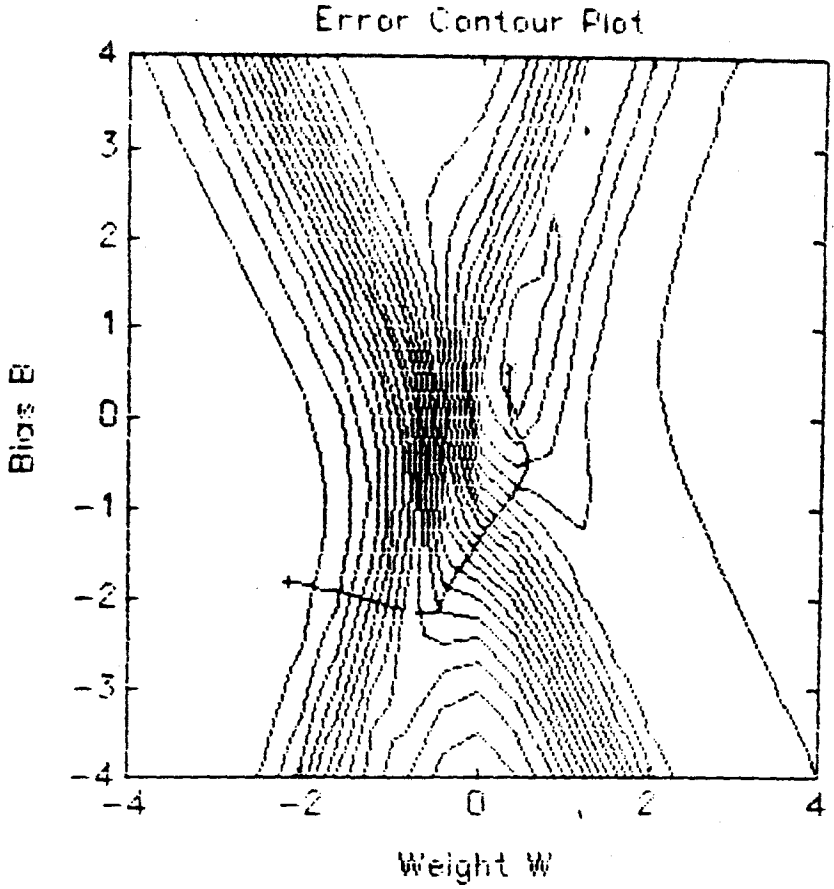


Figura 4.11

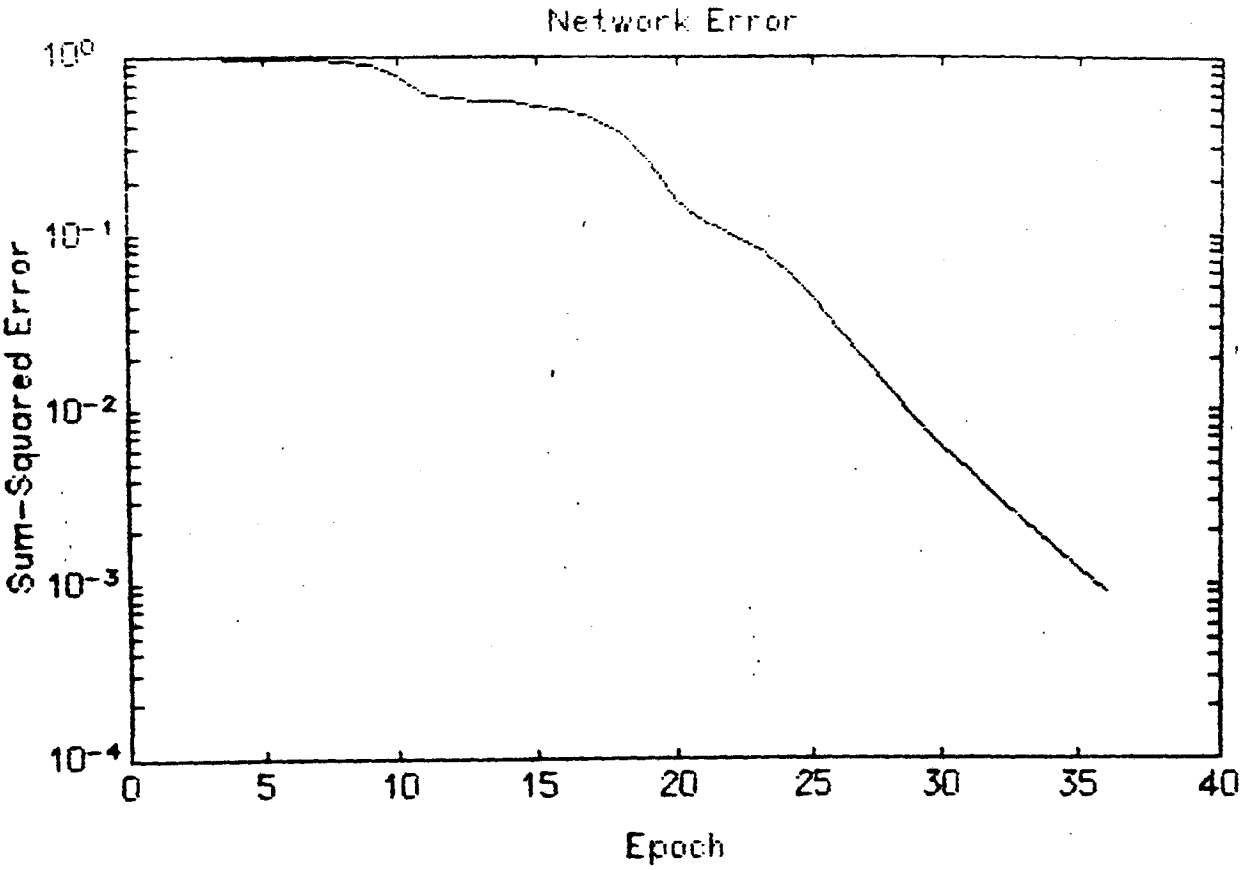


Figura 4.12

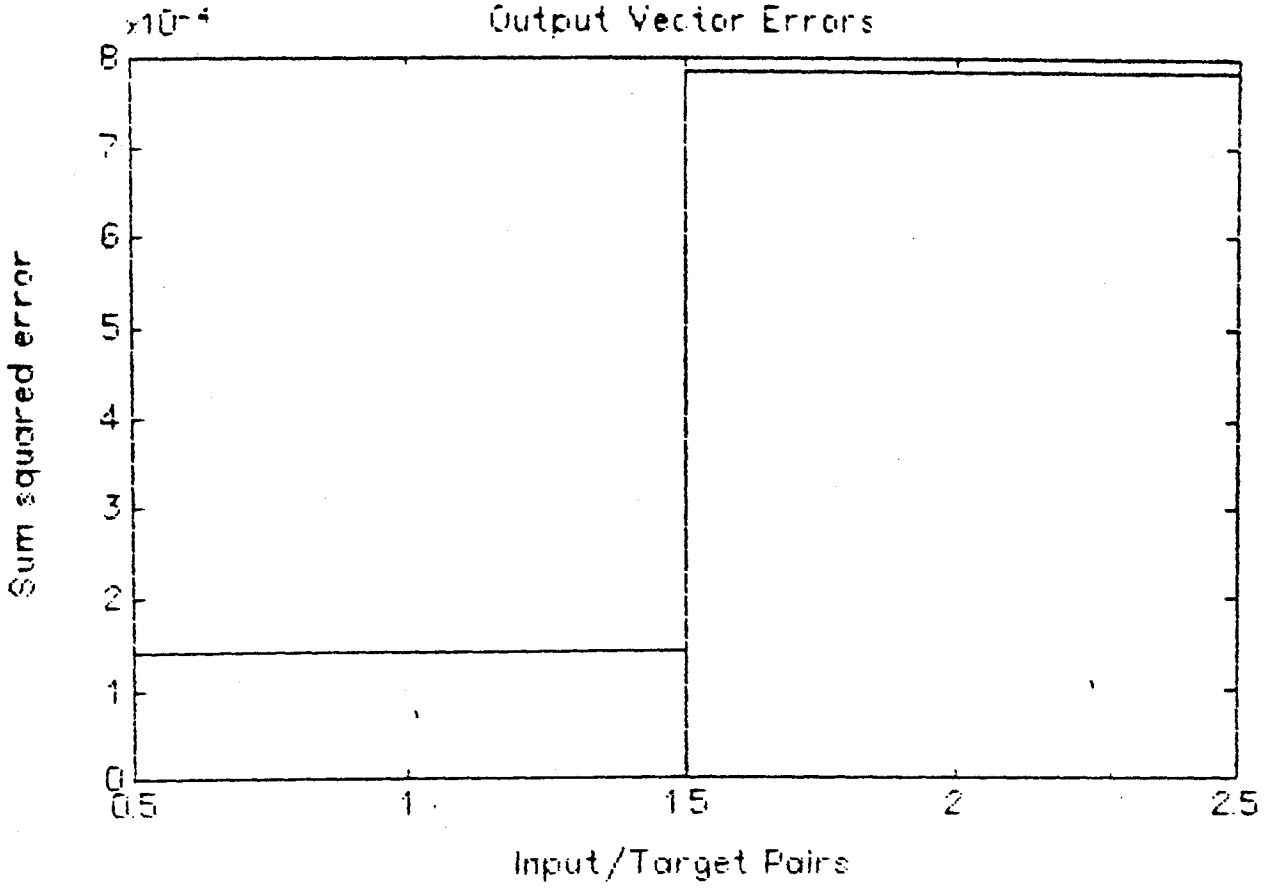

Output Vector Errors
Sum squared error
Input/Target Pairs
0
1
2
3
4
5
6
7
8
0.5
1
1.5
2
2.5

Figura 4.13

Exemplo 2:

Aqui está um conjunto de 21 vetores de entrada de 1 elemento e as saídas correspondentes para treinar uma rede de 2 camadas para aproximação de uma função.

```
P = -1:0.1:1;
T = [-0.96  0.577  -0.0729  0.377  0.641   0.66   0.461 ...
     0.1336  -0.201  -0.434  -0.5   -0.393  -0.1647  0.0988 ...
     0.3072   0.396  0.3449   0.1816  -0.0312  -0.2189 -0.3201];
```

O código acima inicializa P com 21 valores: -1, -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.

O gráfico da figura 4.14 mostra os valores dos vetores entrada/saída.

Cada vetor de entrada e saída tem um elemento, assim a rede tem uma entrada e um neurônio com função de transferência linear na última camada. Cinco neurônios com função de transferência tan-sigmoid serão usados na primeira camada oculta.

Este código define a arquitetura da rede e inicializa pesos e "biases":

```
[R,Q] = size(P);
S1 = 5;
[S2,Q] = size(T);
[W1,B1] = rands(S1,R);
[W2,B2] = rands(S2,S1);
```

As saídas da rede são calculadas para os vetores de entrada:

```
A1 = tansig(W1*P2,B1);
A2 = purelin(W2*A1,B2);
```

O comportamento inicial da rede é mostrado pela linha sólida no gráfico da figura 4.15.

Agora os parâmetros da rede podem ser definidos e a rede treinada:

```
disp_freq = 10;
max_epoch = 8000;
err_goal = 0.02;
lr = 0.01;
TP = [disp_freq max_epoch err_goal lr];
[W1,B1,W2,B2,epochs,TR] =
        trainbp(W1,B1,'tansig',W2,B2,'purelin',P,T,TP)
```

O gráfico da figura 4.16 mostra o final do treinamento, após 6801 ciclos, quando o erro foi menor que o erro aceitável

Embora a rede foi treinada apenas para os valores de entrada de -1.0, -0.9, -0.8, ..., 0, ..., 0.8, 0.9, 1, outras entradas podem ser apresentadas a rede. Por exemplo, se um vetor P = 0.33 é apresentado a rede, a saída A2 pode ser calculada:

```
A1 = tansig(W1*0.33,B1);
A2 = purelin(W2*A1,B2)
A2 = 0.1659
```

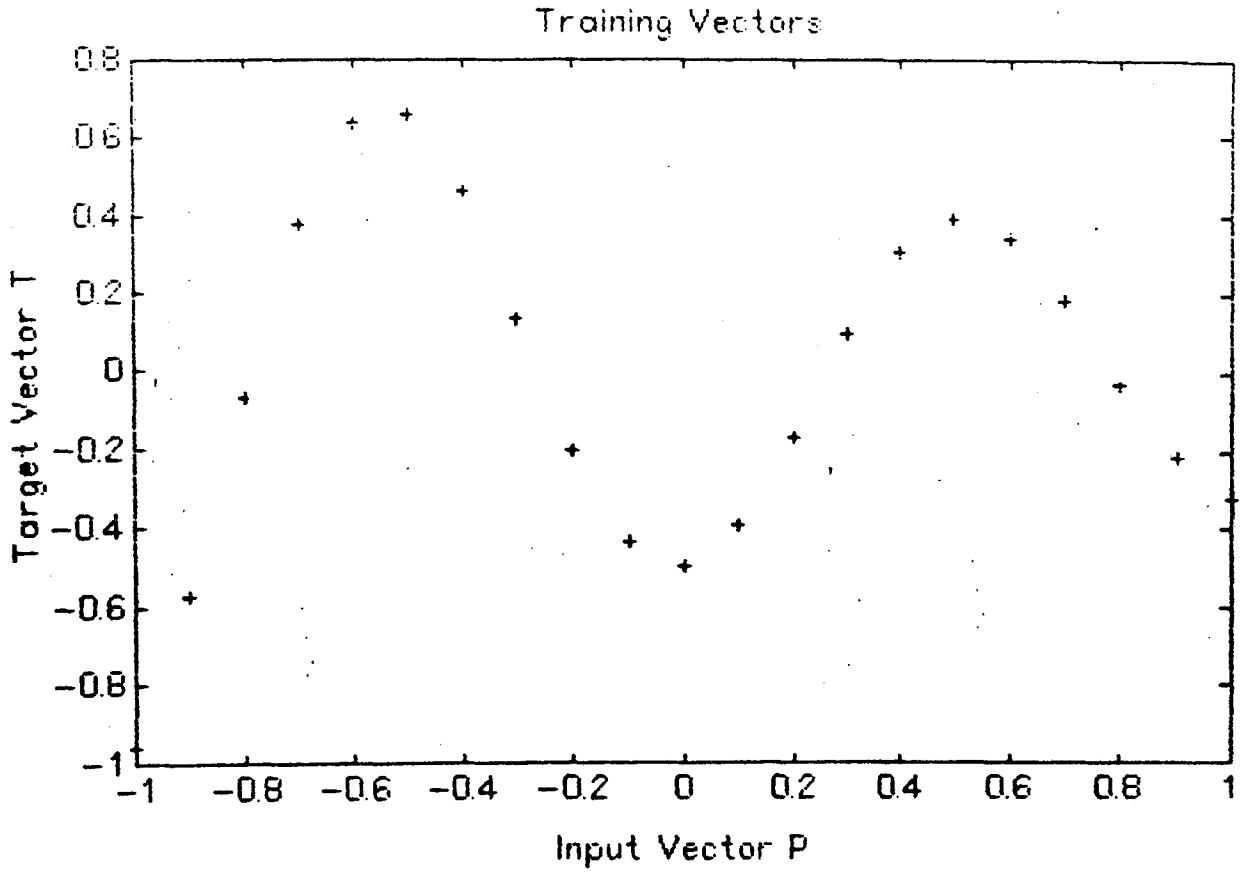


Figura 4.14

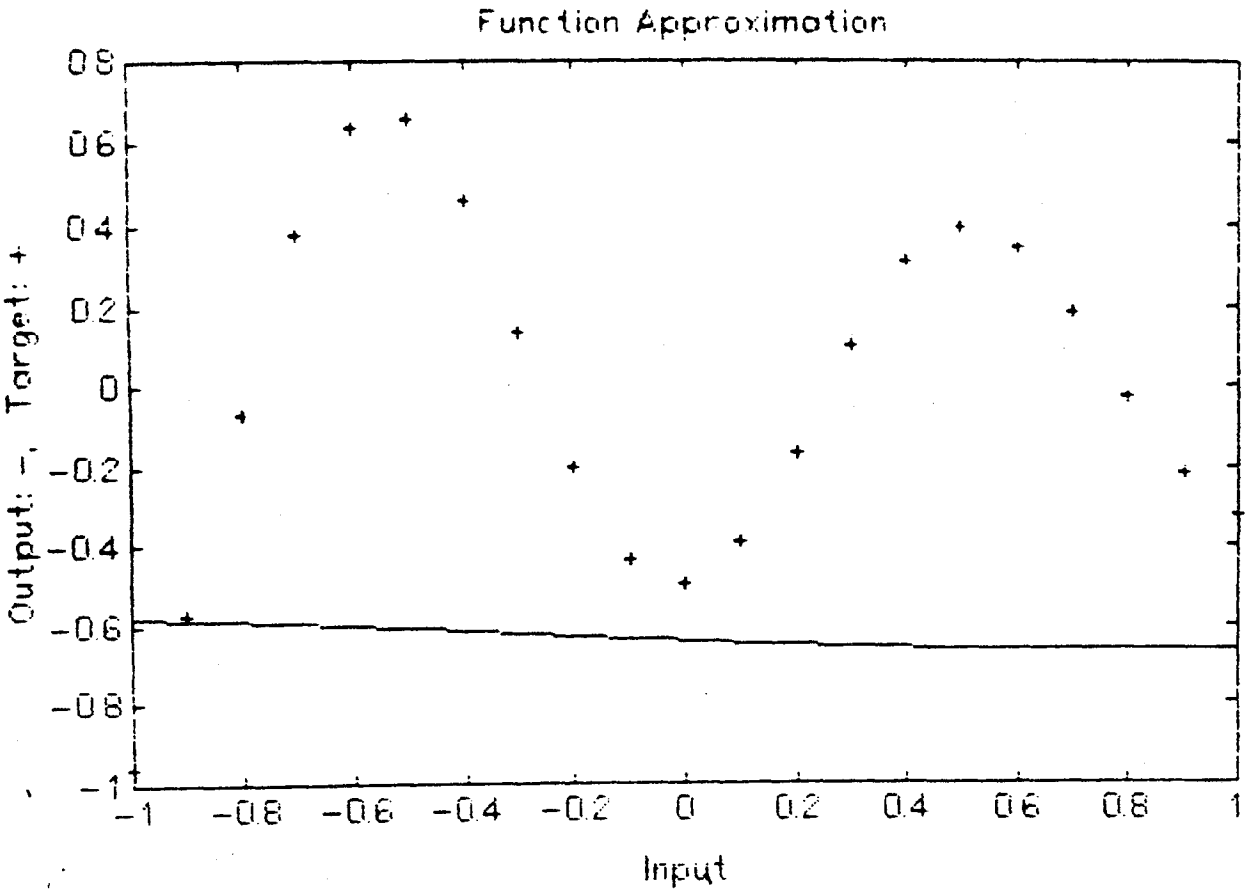


Figura 4.15

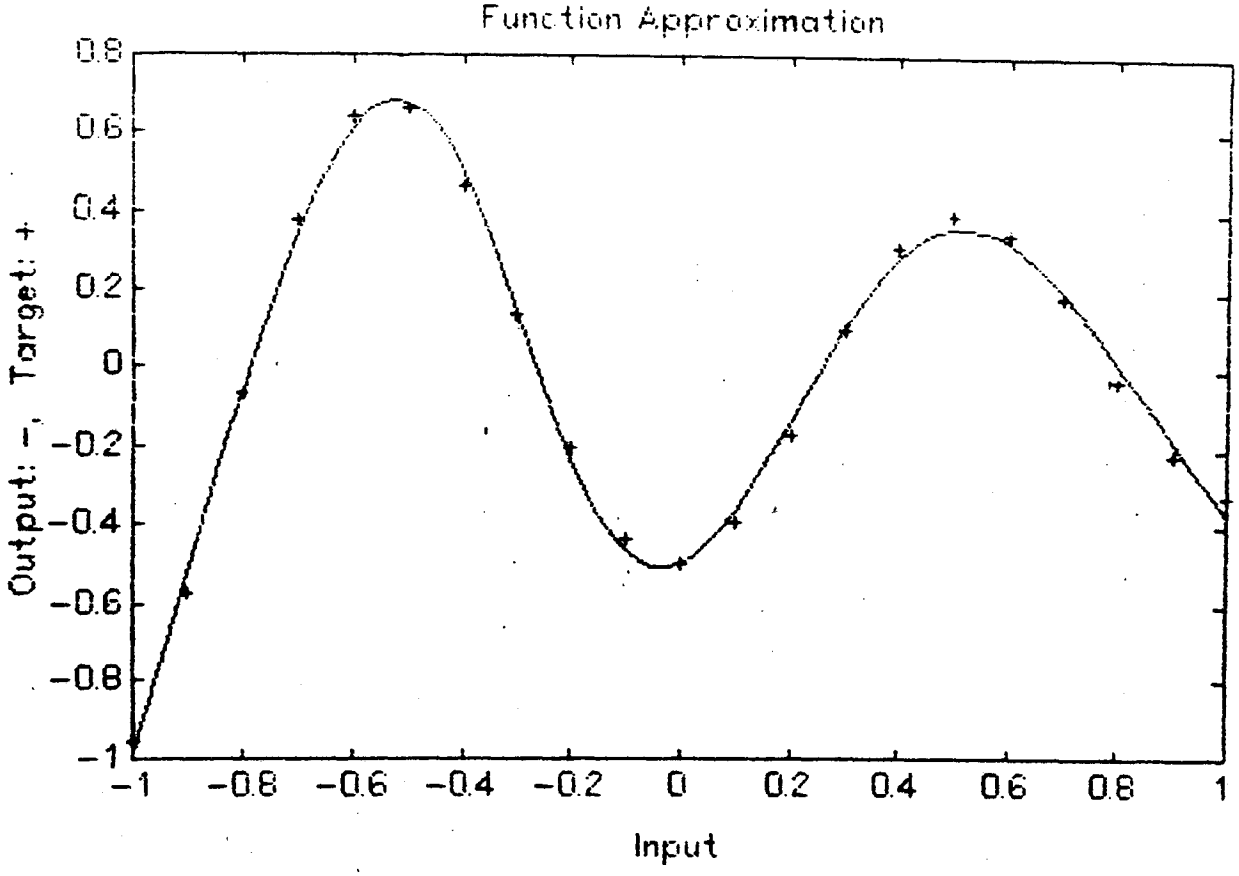


Figura 4.16

4.4 Melhorando o algoritmo BACKPROPAGATION

Existem três maneiras simples de tornar o processo de aprendizatgem mais rápido :

4.4.1 Aprendizagem com momentum

Momentum permite a rede ignorar pequenas características na superfície de erro. Sem momentum uma rede pode "estacionar" num erro local. Com momentum uma rede pode deslizar através de tais mínimos e procurar um mínimo global.

Momentum pode ser adicionado a aprendizagem backpropagation da seguinte forma (ver item 3.3):

$$W'' = W' + (lr^* E_p * P) |P|^2 + momentum * (W' - W)$$

Quando a constante momentum é zero, uma troca de peso é baseado somente no gradiente. Quando a constante momentum é 1 a nova troca de peso é tomada igual a última troca de peso e o gradiente é simplesmente ignorado. Tipicamente a constante momentum é colocada igual a 0.95.

learnbpm implementa backpropagation com momentum.

Antes de usar learnbpm, a matriz de troca de pesos e o vetor de troca de "bias" devem ser inicalizados com zero.

```
dW = 0 * W;
dB = 0 * B;
```

Então trocas de pesos e bias de uma camada pode ser encontrada da entrada atual P, deltas D, taxa de aprendizagem lr, e constante momentum mc.

```
[dW,dB] = learnbpm(P,D,lr,mc,dW,dB);
```

A função learnbpm retorna uma nova matriz de pesos e um novo vetor de "biases".

Estas trocas de pesos e "biases" serão rejeitados se elas resultarem num aumento muito grande do erro. A rejeição de pesos ocorre se a razão do novo erro para o velho exceder uma máxima razão de erro. A máxima razão de erro pode ter qualquer valor maior do que 1. É tipicamente setada para 1.04.

trainbpm treina uma rede com mais de três camadas usando a regra de aprendizagem backpropagation com momentum. Duas camadas são usadas comumente.

```
TP = [disp_freq max_epoch err_goal lr momentum err_ratio];
[W,B,TE,TR] = trainbpm(W,B,'logsig',P,T,TP]
```

4.4.2 Melhores condições iniciais

Nguyen e Widrow encontraram uma maneira de selecionar pesos e "biases" iniciais que são melhores do que puramente valores randômicos. Eles fizeram isto analizando como uma rede de duas camadas executa uma função de aproximação. O uso de seus métodos levou a uma rede satisfatória em poucos ciclos de treinamento.

A função nwlog implementa o método Nguyen-Widrow para neurônios com função de transferência "sigmoid". Ele aceita o número de linhas e colunas da matriz peso requisitada e retorna uma matriz peso inicial e um vetor "bias" inicial. (Outra função, tanlog, retorna uma matriz peso inicial e "bias" para uma camada tan-sigmóide.)

```
[W1,B1] = nwlog(S1,R);
[W2,B2] = nwlog(S2,S1);
```

4.4.3 Taxa de aprendizagem adaptativa

Uma taxa de aprendizagem adaptativa requer algumas mudanças no procedimento de treinamento. Primeiro as saídas iniciais da rede e erro são calculados. A cada ciclo, novos pesos e "biases" são calculados usando a corrente taxa de aprendizagem. Novas saídas e novos erros são então calculados.

Da mesma forma que para o momentum, se o novo erro excede o velho erro por mais que uma razão pré-definida (tipicamente 1.04), os novos pesos, "biases", saídas e erros são descartados. Em adição, a taxa de aprendizagem é diminuída (tipicamente multiplicada por 0.7). Por outro lado os novos pesos, "biases", etc..., são mantidos. Se o novo erro é menos que o velho erro, a taxa de aprendizagem é aumentada(tipicamente multiplicada por 1.05).

Este procedimento aumenta a taxa de aprendizagem, mas apenas para a extensão que a rede pode aprender sem aumentos grandes nos erros. Quando uma taxa de aprendizagem grande poderia resultar em aprendizagem estável, a taxa de aprendizagem é aumentada. Quando a taxa de aprendizagem é muito alta para garantir uma diminuição no erro, ela é diminuida até conseguir aprendizagem estável.

O treinamento de uma rede com uma taxa de aprendizagem adaptativa é implementado com a função trainbpa, que é chamada como trainbp exceto com os parâmetros adicionais de treinamento em TP. Como trainbp, trainbpa pode treinar redes tendo três camadas.

Aqui está a chamada para treinar uma rede com uma simples camada:

```
[W,B,TE,TR] = trainbpa(W,B,'F',P,T,TP)
```

Os parâmetros no vetor TP são: disp_freq, max_epoch, err_goal, learning_rate, lr_increase_multiplier, lr_decrease_multiplier, e o error_ratio. A função retorna novos pesos e "biases", o número de ciclos TE que a rede foi treinada, e um registro de treinamento, TR, o qual tem duas linhas de valores. A primeira linha contém o erro da rede através do treinamento. A segunda linha contém a taxa de aprendizagem.

4.4.4 Adotando o momentum e taxa de aprendizagem adaptativa

Note que momentum pode ser usado junto com uma taxa de aprendizagem para obter o benefício de ambas técnicas. A função **trainbpx** aplica ambas as técnicas para backpropagation. Esta função é chamada como trainbp, trainbpm e trainbpa, exceto que requer mais parâmetros de treinamento.

```
TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec momentum err_ratio];
[W1,B1,W2,B2,epochs,TR] =
        trainbpx(W1,B1,'logsig',W2,B2,'logsig',P,T,TP)
```

4.5 Outras Redes Neurais disponíveis no MATLAB

Outras Redes Neurais Artificiais que são disponíveis no ambiente MATLAB são apresentadas a seguir:

- ADALINE / Widrow-Hoff (Delta) Rule
- Associative Learning Rules
  - Hebbian Learning Rule
  - Instar Learning Rule
  - Kohonen Learning Rule
  - Outstar Learning Rule
- Competitive Learning
- Feature Map
- Adaptative Resonance Theory - ART1 Network
- Hopfield Network

Maiores detalhes podem ser encontrados no manual "Neural Network TOOLBOX for use with MATLAB".

5 Aplicação da Backpropagation no Reconhecimento de Doenças do Coração

Baseados numa tabela armazenada num arquivo "doencas.cor" a respeito do diagnóstico de doenças do coração realizamos testes com Backpropagation. Todos os atributos dessa tabela (Apêndice) são valores numéricos.

Principal responsável pela coleta dos dados:

V. A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

A tabela possui 303 instâncias e 76 atributos.

Apenas 14 atributos são realmente usados:

1. idade
2. sexo (1=masculino , 0=feminino)
3. cp (tipo de dor no tórax
   - Valor 1: angina típica
   - Valor 2: angina atípica
   - Valor 3: dor não anginal
   - Valor 4: não sintomático
4. trestbps (pressão do sangue em repouso (em mm Hg na entrada para o hospital)
5. chol - colesterol em mg/dl
6. fbs - açúcar no sangue em jejum (1 = verdadeiro; 0 = falso)
7. restecg - resultados eletrocardigráfico em repouso
   - Valor 0: normal
   - Valor 1: tendo onda ST-T anormal (inversões da onda T e/ou elevação ou depressão ST de 0.05 mV)
   - Valor 2: mostrando provável ou definida hipertrofia ventricular esquerda pelo "Estes' criteria"
8. thalach - taxa máxima do coração atingida.
9. exang - exercício de angina induzida (1 = sim; 0 = não)
10. oldpeak - depressão ST induzida pelo exercício relativo ao repouso.
11. slope - a inclinação do segmento ST do exercício de pico
    - Valor 1: para cima
    - Valor 2: plano
    - Valor 3: para baixo
12. ca - número de vasos maiores (0-3) coloridos pelo "flourosopy"
13. thal - 3 = normal; 6 = defeito fixado; 7 = defeito reversível
14. num - diagnóstico de doença do coração
    - -- Valor 0: < 50% ausência de doença
    - -- Valor 1: > 50% presença de doença

5.1 Testes

Os testes realizados foram feitos utilizando diretamente 200 dados da tabela de doenças do coração para treinamento e 103 para teste (aproximadamente 1/3 dos dados). Como dispunhamos de 13 atributos e 200 dados, a matriz P foi definida com 13 linhas e 200 colunas (ver item 4.2.1). O vetor saída T foi definido com 200 valores indicando a presença ou ausência de doença (0 = ausência e 1 = presença).

Fizemos um programa em linguagem C para transformar a tabela "doencas.cor" na matriz P e no vetor T e armazená-los no arquivo doencas.mat (ver item 2.5.1).

Inicialmente, considerando os dados na forma original, verificamos que em nenhum dos testes realizados a rede convergia para o erro quadrático total desejado. Decidimos então normalizar os dados de entrada, ou seja, colocamos todos os valores de entrada entre 0 e 1. Para isso acrescentamos ao programa em C o código para normalização dos dados. O programa completo é mostrado a seguir.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "c:\util\matlab\loadsave\savemat.c"

#define A 13   /* numero de linhas da matriz P */
#define C 1    /* numero de linhas da matriz T */
#define N 200  /* numero de colunas da matriz P e T */

double x[N][A]; /* matriz contendo as 13 primeiras colunas da tabela de
                   doencas do coracao */
double dv[A+1];

Ler_Vetor(arq,vet,n)
   FILE *arq;
   double *vet;
   int n;
{
   register int i;
   float aux;

   for (i=0;i<n;i++)
   {
      fscanf(arq,"%f",&aux);
      ftell(arq);
      if (feof(arq)) return 0;
      *(vet+i)=aux;
   }
   return 1;
}
```

```
Converte()
{
  double y[N][C]; /* matriz contendo a ultima coluna da matriz de doencas */
  FILE *arq;
  int n;
  register int i;

  arq=fopen("doencas.cor","rb");
  if (!arq) {
    printf("Nao consegui abrir o arquivo de dados !\n");
    return 0;
  }
  for (n=0;!feof(arq) && (n<N);n++)
  {
     x[n][0]=1;
     y[n][0]=0;
     if(!Ler_Vetor(arq,x[n],A) || !Ler_Vetor(arq,y[n],C))
     {
        printf("Erro ao ler dados \n\n");
        fclose(arq);
        return 0;
     }
     fscanf(arq,"\r\n");
  }
  fclose(arq);


// Normaliza os dados da tabela de doencas

  for (i=0;i<=A;i++) dv[i]=1;
  for (n=0;n<N;n++) for(i=0;i<A;i++)
    if (x[n][i]/dv[i]>1)
    dv[i]=pow10(1+(int)log10(x[n][i]));
  for (n=0;n<N;n++)
    for(i=0;i<A;i++)
     x[n][i]=x[n][i]/dv[i];

  arq=fopen("doencas.mat","wb");
  savemat(arq,0,"P",A,N,0,x,(double *)0);
  savemat(arq,0,"T",C,N,0,y,(double *)0);
  fclose(arq);

}

main()
{
   Converte();
}
```

Em todos os testes os valores da taxa de aprendizagem, do momentum e do erro quadrático desejado foram considerados 0.02, 0.95 e 0.01 respectivamente.

Realizamos testes para duas topologias de redes diferentes:

1.Utilizando 2 camadas (1 camada oculta):

	Teste 1	Teste 2	Teste 3
Nro. de entradas	13	13	13
Nro. de neurônios na camada oculta	20	100	200
Nro. de neurônios na última camada	1	1	1

A seguir temos o código do programa feito no MATLAB que utilizamos para realização desses testes e os gráficos resultantes do teste 1(figuras 5.1 e 5.2) e do teste 2 (figuras 5.3 e 5.4):

```
load doencas

[R,Q] = size(P);
S1 = 20;
[S2,Q] = size(T);

[W1,B1] = nwlog(S1,R);
[W2,B2] = nwlog(S2,S1)

A1 = logsig(W1*P,B1);
A2 = logsig(W2*A1,B2);

disp_freq = 50;
max_epoch = 15000;
err_goal = 0.01;
lr = 0.02;
lr_inc = 1.05;
lr_dec = 0.7;
err_ratio = 1.04;
momentum = 0.95;

TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec momentum err_ratio];
[W1,B1,W2,B2,epochs,TR] = trainbpx(W1,B1,'logsig',W2,B2,'logsig',P,T,TP);

A = logsig(W1*P,B1);
FIM = logsig(W2*A,B2);

plottr(TR);
meta graf1;
barerr(FIM,T);
meta graf2;
```

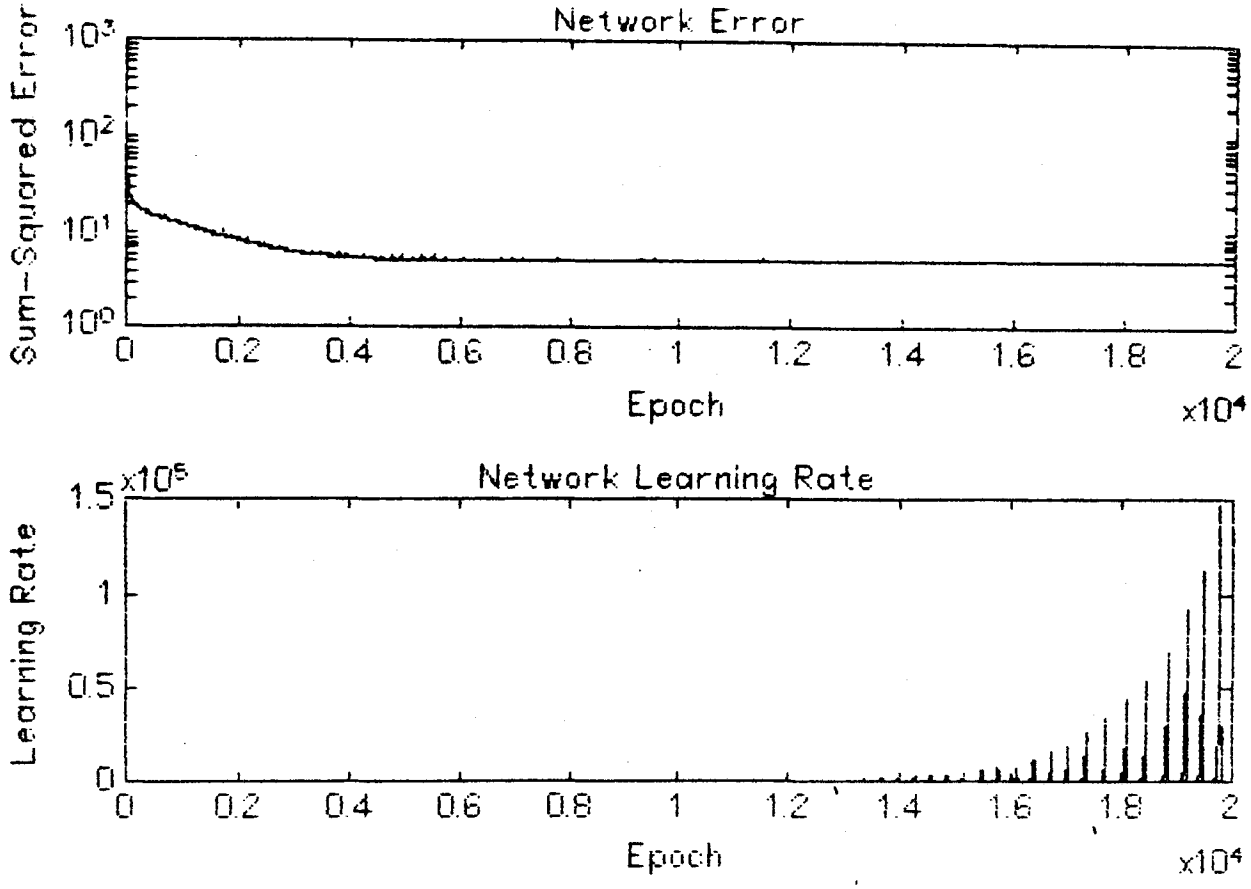


Figura 5.1

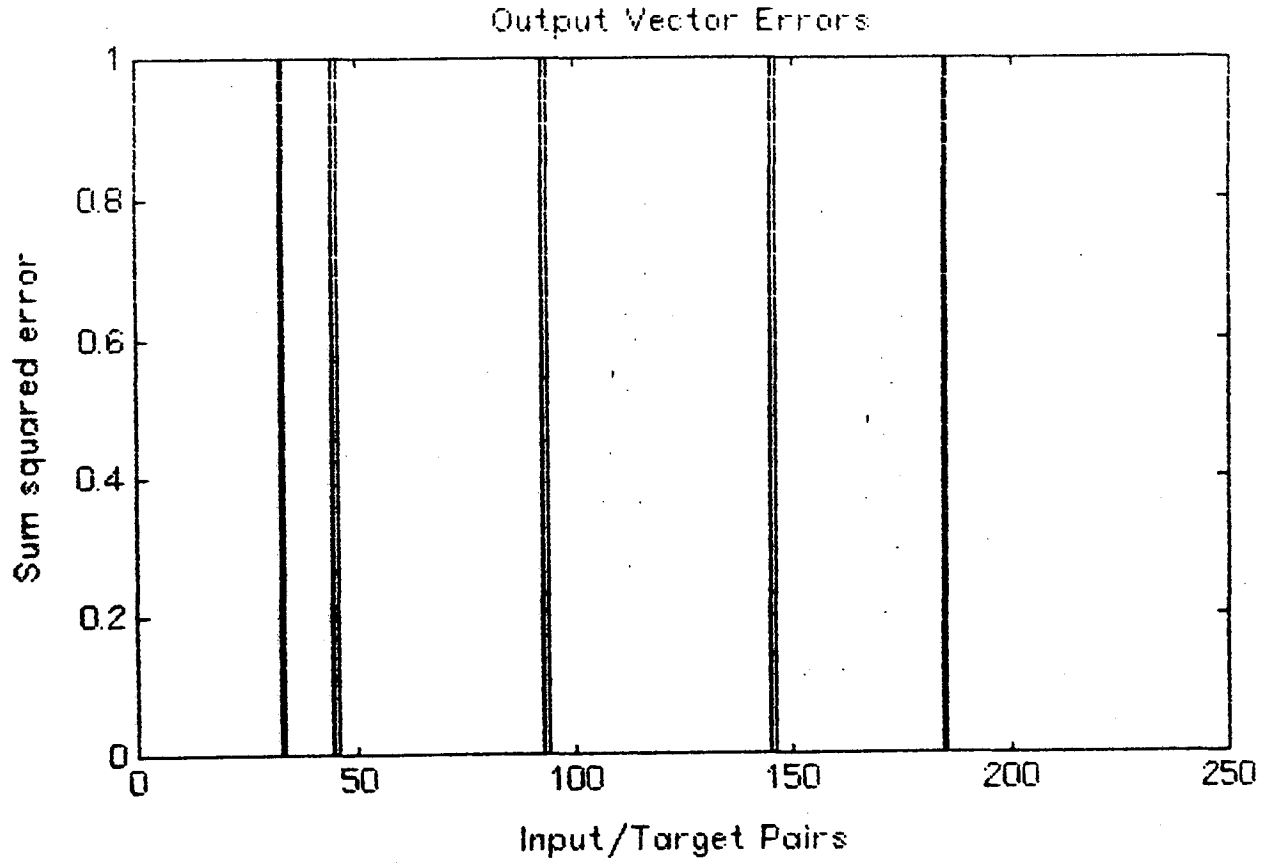


Figura 5.2

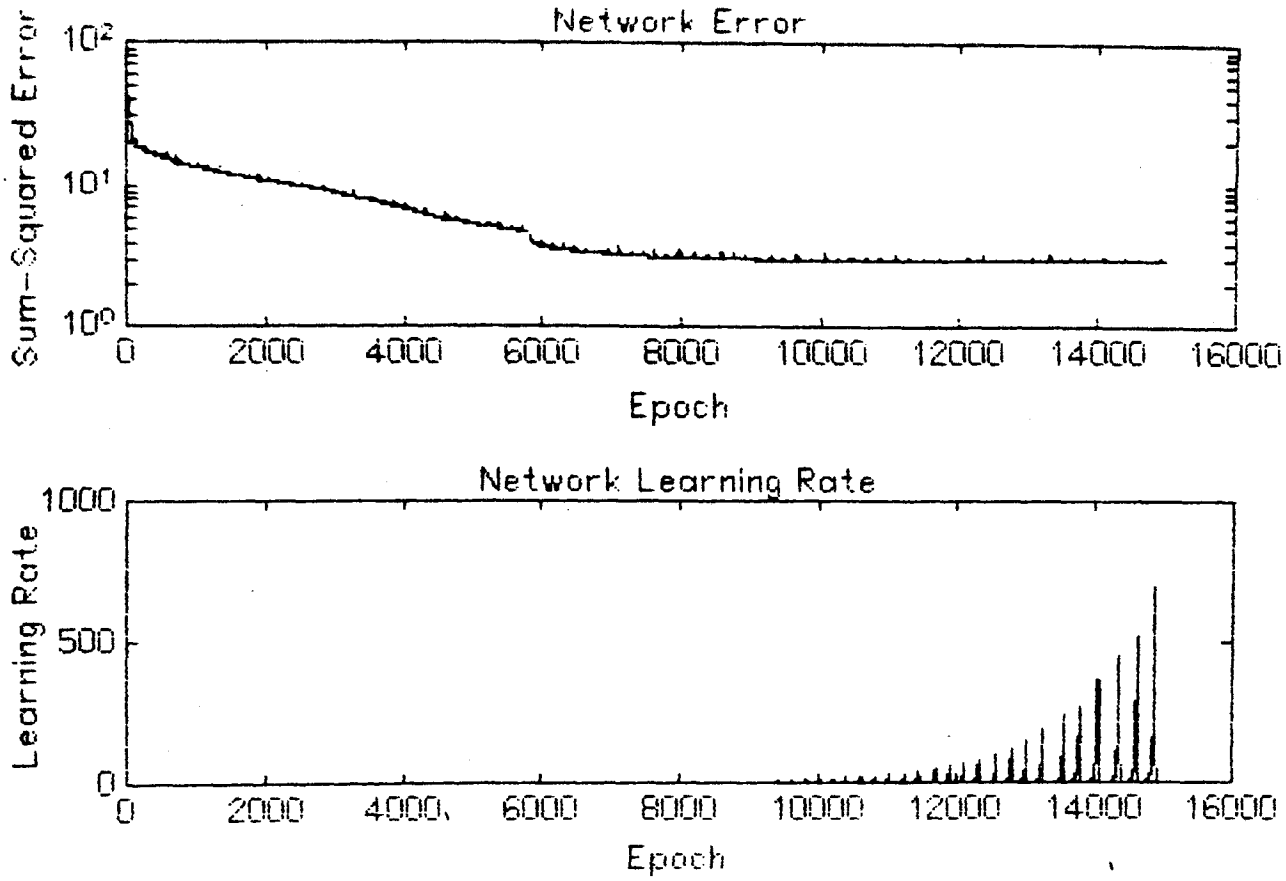


Figura 5.3

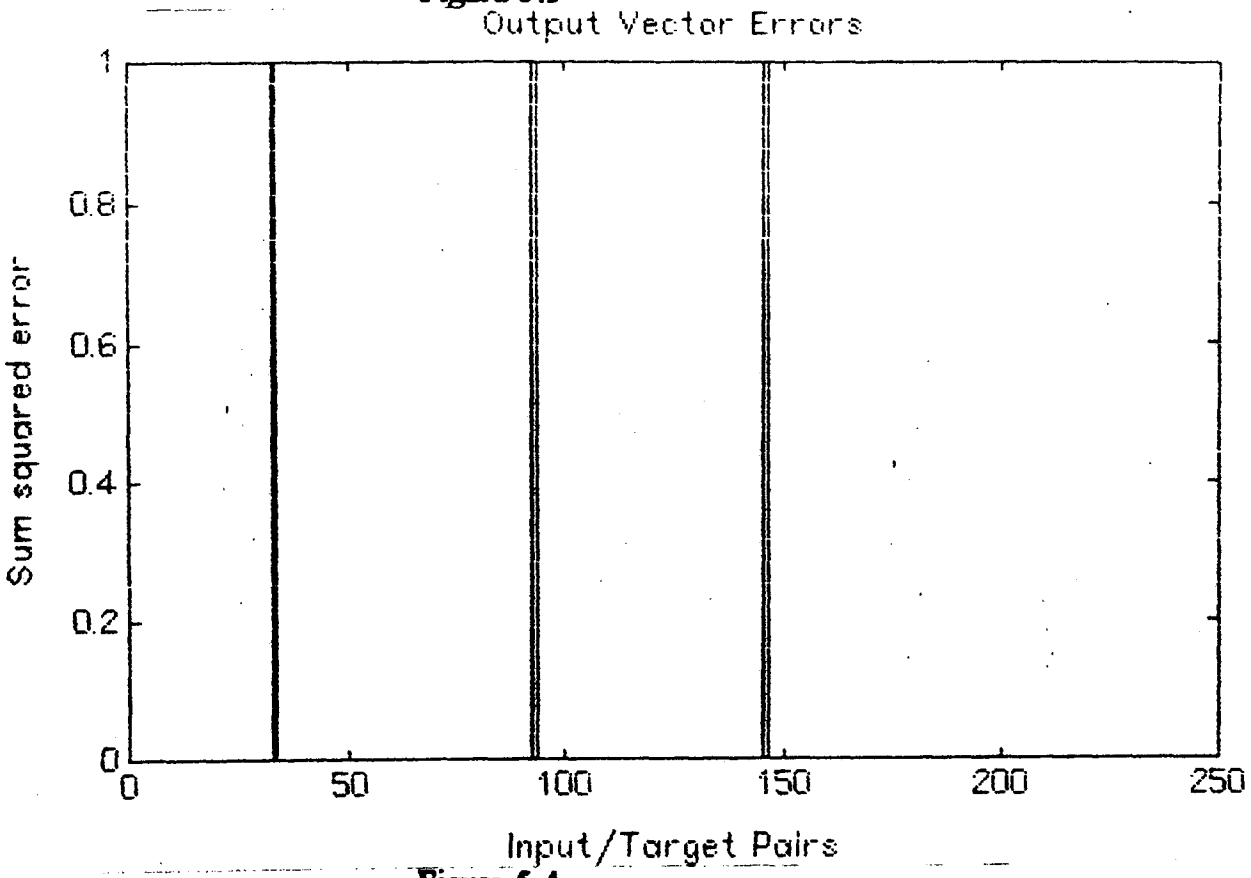


Figura 5.4

2. Utilizando 3 camadas (2 camadas ocultas):

	Teste 4	Teste 5
Nro. de entradas	13	13
Nro. de neurônios na primeira camada oculta	20	100
Nro. de neurônios na segunda camada oculta	20	100
Nro. de neurônios na última camada	1	1

A seguir temos o código do programa feitos no MATLAB que utilizamos para realização dos testes 4 e 5 e os gráficos resultantes do teste 4(figuras 5.5 e 5.6):

```
load doencas

[R,Q] = size(P);
S1 = 20;
S2 = 20;
[S3,Q] = size(T);

[W1,B1] = nwlog(S1,R);
[W2,B2] = nwlog(S2,S1);
[W3,B3] = nwlog(S3,S2)

A1 = logsig(W1*P,B1);
A2 = logsig(W2*A1,B2);
A3 = logsig(W3*A2,B3);

disp_freq = 100;
max_epoch = 15000;
err_goal = 0.01;
lr = 0.02;
lr_inc = 1.05;
lr_dec = 0.7;
err_ratio = 1.04;
momentum = 0.95;

TP = [disp_freq max_epoch err_goal lr lr_inc lr_dec momentum err_ratio];
[W1,B1,W2,B2,W3,B3,epochs,TR] = trainbpx(W1,B1,'logsig',W2,B2,'logsig',W3,B3,'logsig',P,T,TP);

A1 = logsig(W1*P,B1);
A2 = logsig(W2*A1,B2);
FIM = logsig(W3*A2,B3);

ploterr(TR);
meta graf1;
barerr(FIM,T);
meta graf2;
```

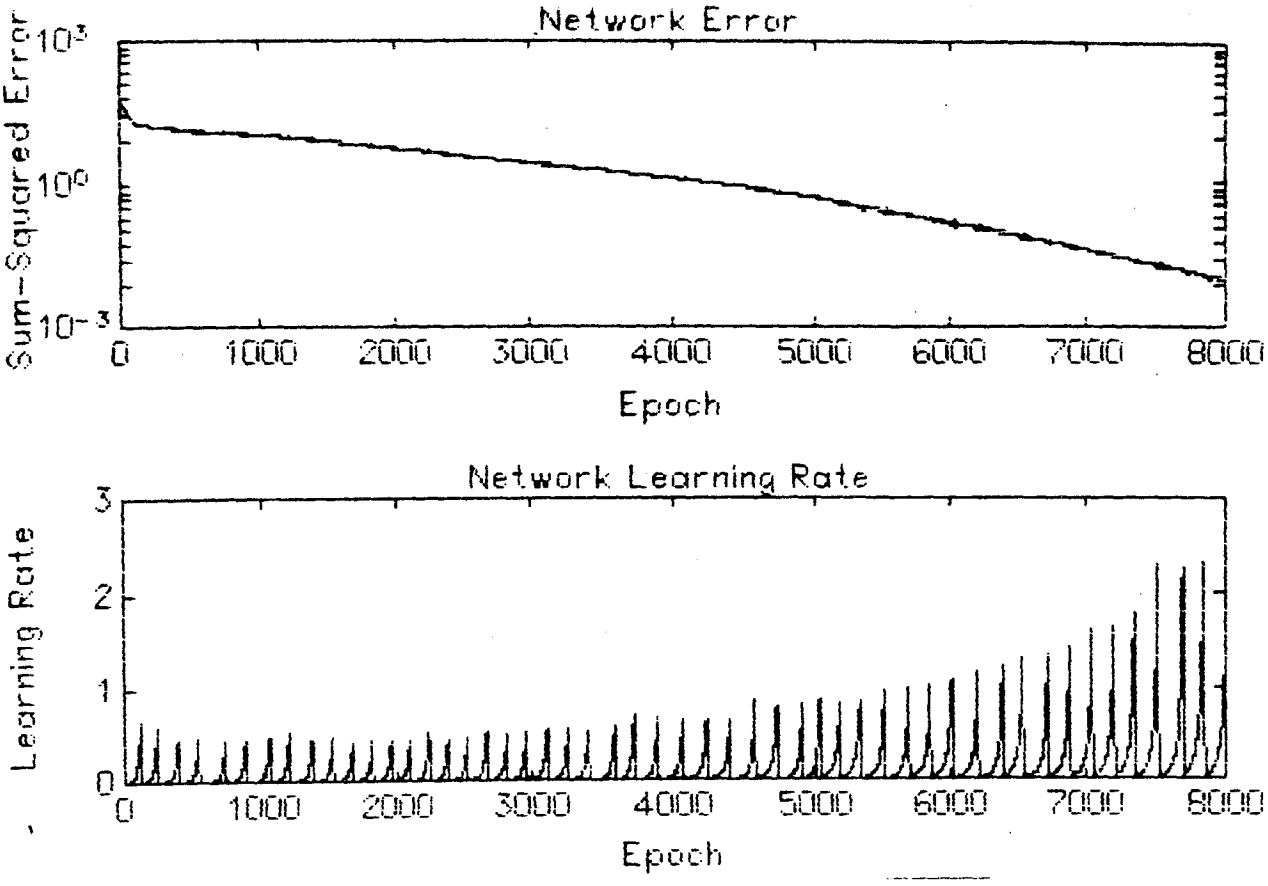


Figura 5.5

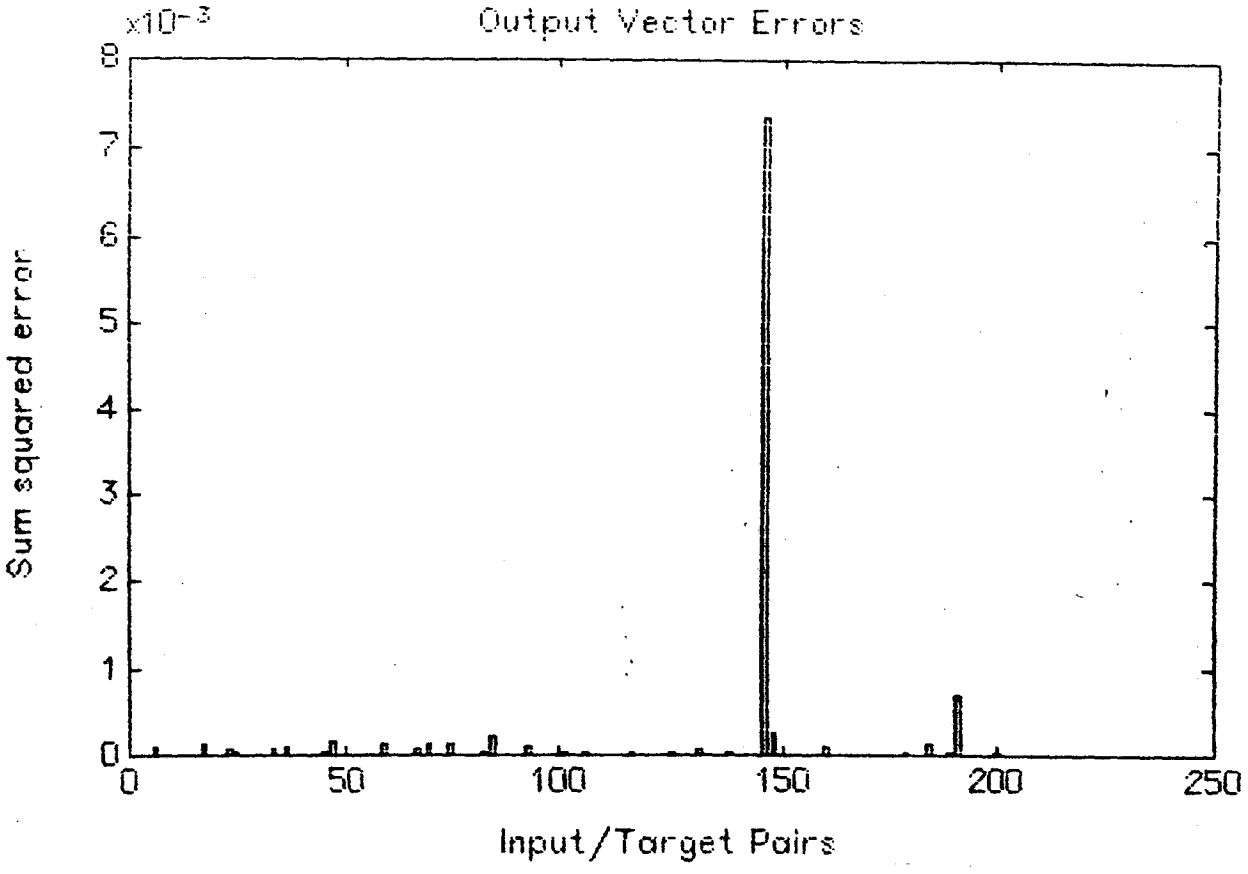


Figura 5.6

5.2 Resultados

Realizamos inicialmente o teste 1 com 20 neurônios na camada oculta e verificamos que a rede não convergiu para o erro desejado em 20000 ciclos de treinamento. Decidimos aumentar o número de neurônios na camada oculta, pois esta é uma das maneiras de se melhorar o desempenho de uma rede. Observamos que mesmo com 100 ou 200 neurônios na camada oculta o erro desejado não era atingido e o tempo de treinamento era muito longo.

Decidimos então aumentar o número de camadas da rede, definindo uma rede com duas camadas ocultas.

Observamos a partir do teste 3 uma aumento significativo no desempenho da rede, que atingiu um erro quadrático total menor do que o desejado em 7994 ciclos de treinamento. Ao apresentarmos os 103 dados de teste à rede, esta respondeu com uma porcentagem de erro de 20 %.

Fizemos ainda mais um teste aumentando o número de neurônios nas duas camadas ocultas para 50. A rede convergiu, mas o tempo de treinamento foi muito longo devido ao grande número de neurônios nas camadas.

Percebemos através dos testes que, tanto para a rede de duas camadas quanto para a rede de três camadas, com mais de 20 neurônios nas camadas ocultas o desempenho da rede não era alterado significativamente e muitos neurônios levavam a tempos de treinamento excessivamente longos (mais de 24 horas utilizando um micro da linha AT 486 33MHz). A rede atingiu seu melhor desempenho aumentando-se apenas uma camada oculta com 20 neurônios.

6 Conclusão

Este trabalho apresentou conceitos básicos de utilização do Neural Network TOOLBOX do ambiente MATLAB, que é um software para implementação dos principais modelos de Redes Neurais Artificiais.

Dois modelos de Redes Neurais Artificiais denominados Perceptron e Redes Multi-Camadas foram estudados, implementados neste ambiente e testados em várias aplicações.

Uma particular aplicação: Reconhecimento de Doenças do Coração foi realizada, utilizando-se uma Rede Multi-Camadas. Topologias diferentes desta rede foram consideradas para mostrar o seu desempenho.

Acreditamos que o presente trabalho será de grande utilização para usuários iniciantes do MATLAB, pois, o mesmo contém um capítulo dedicado aos comandos básicos do MATLAB e a construção e impressão de gráficos neste ambiente.

7 Bibliografia

[1] CAUDILL, M., "Neural Networks Primer_part I",AI EXPERT, DEZ/87.

[2] CAUDILL, M., "Neural Networks Primer_part II",AI EXPERT, FEV/88.

[3] CAUDILL, M., "Neural Networks Primer_part III",AI EXPERT, JUN/88.

[4] CAUDILL, M., "Neural Networks Primer_part IV",AI EXPERT, AUG/88.

[5] LIPPMANN, R.P., "An introduction to Computing with Neural Nets" IEEE-ASSP Magazine, vol. 4, nro 2, abril, 1987.

[6] HUSH, D.R and HORNE, B., "An introduction to Neural Networks" UNM Technical Report Number EECE 90-005, november, 1990.

[7] F.ROSENBLATT, "The perceptron: A probabilistic model for information storage and organization in the brain," Psychological Review, vol.65, pp.386-408, 1958.

[8] MATLAB™ for Sun Workstations - User's Guide

[9] BEALE M.H. & DEMUTH H.B., Neural Network TOOLBOX For Use with MATLAB.

8 Apêndice

Neste apêndice consta a tabela de doenças do coração "doenças.cor".

As 14 colunas da tabela doenças.cor correspondem aos 14 atributos descritos na página 63.

63.0 1.0 1.0 145.0 233.0 1.0 2.0 150.0 0.0 2.3 3.0 0.0 6.0 0.0
67.0 1.0 4.0 160.0 286.0 0.0 2.0 108.0 1.0 1.5 2.0 3.0 3.0 1.0
67.0 1.0 4.0 120.0 229.0 0.0 2.0 129.0 1.0 2.6 2.0 2.0 7.0 1.0
37.0 1.0 3.0 130.0 250.0 0.0 0.0 187.0 0.0 3.5 3.0 0.0 3.0 0.0
41.0 0.0 2.0 130.0 204.0 0.0 2.0 172.0 0.0 1.4 1.0 0.0 3.0 0.0
56.0 1.0 2.0 120.0 236.0 0.0 0.0 178.0 0.0 0.8 1.0 0.0 3.0 0.0
62.0 0.0 4.0 140.0 268.0 0.0 2.0 160.0 0.0 3.6 3.0 2.0 3.0 1.0
57.0 0.0 4.0 120.0 354.0 0.0 0.0 163.0 1.0 0.6 1.0 0.0 3.0 0.0
63.0 1.0 4.0 130.0 254.0 0.0 2.0 147.0 0.0 1.4 2.0 1.0 7.0 1.0
53.0 1.0 4.0 140.0 203.0 1.0 2.0 155.0 1.0 3.1 3.0 0.0 7.0 1.0
57.0 1.0 4.0 140.0 192.0 0.0 0.0 148.0 0.0 0.4 2.0 0.0 6.0 0.0
56.0 0.0 2.0 140.0 294.0 0.0 2.0 153.0 0.0 1.3 2.0 0.0 3.0 0.0
56.0 1.0 3.0 130.0 256.0 1.0 2.0 142.0 1.0 0.6 2.0 1.0 6.0 1.0
44.0 1.0 2.0 120.0 263.0 0.0 0.0 173.0 0.0 0.0 1.0 0.0 7.0 0.0
52.0 1.0 3.0 172.0 199.0 1.0 0.0 162.0 0.0 0.5 1.0 0.0 7.0 0.0
57.0 1.0 3.0 150.0 168.0 0.0 0.0 174.0 0.0 1.6 1.0 0.0 3.0 0.0
48.0 1.0 2.0 110.0 229.0 0.0 0.0 168.0 0.0 1.0 3.0 0.0 7.0 1.0
54.0 1.0 4.0 140.0 239.0 0.0 0.0 160.0 0.0 1.2 1.0 0.0 3.0 0.0
48.0 0.0 3.0 130.0 275.0 0.0 0.0 139.0 0.0 0.2 1.0 0.0 3.0 0.0
49.0 1.0 2.0 130.0 266.0 0.0 0.0 171.0 0.0 0.6 1.0 0.0 3.0 0.0
64.0 1.0 1.0 110.0 211.0 0.0 2.0 144.0 1.0 1.8 2.0 0.0 3.0 0.0
58.0 0.0 1.0 150.0 283.0 1.0 2.0 162.0 0.0 1.0 1.0 0.0 3.0 0.0
58.0 1.0 2.0 120.0 284.0 0.0 2.0 160.0 0.0 1.8 2.0 0.0 3.0 1.0
58.0 1.0 3.0 132.0 224.0 0.0 2.0 173.0 0.0 3.2 1.0 2.0 7.0 1.0
60.0 1.0 4.0 130.0 206.0 0.0 2.0 132.0 1.0 2.4 2.0 2.0 7.0 1.0
50.0 0.0 3.0 120.0 219.0 0.0 0.0 158.0 0.0 1.6 2.0 0.0 3.0 0.0
58.0 0.0 3.0 120.0 340.0 0.0 0.0 172.0 0.0 0.0 1.0 0.0 3.0 0.0
66.0 0.0 1.0 150.0 226.0 0.0 0.0 114.0 0.0 2.6 3.0 0.0 3.0 0.0
43.0 1.0 4.0 150.0 247.0 0.0 0.0 171.0 0.0 1.5 1.0 0.0 3.0 0.0
40.0 1.0 4.0 110.0 167.0 0.0 2.0 114.0 1.0 2.0 2.0 0.0 7.0 1.0
69.0 0.0 1.0 140.0 239.0 0.0 0.0 151.0 0.0 1.8 1.0 2.0 3.0 0.0
60.0 1.0 4.0 117.0 230.0 1.0 0.0 160.0 1.0 1.4 1.0 2.0 7.0 1.0
64.0 1.0 3.0 140.0 335.0 0.0 0.0 158.0 0.0 0.0 1.0 0.0 3.0 1.0
59.0 1.0 4.0 135.0 234.0 0.0 0.0 161.0 0.0 0.5 2.0 0.0 7.0 0.0
44.0 1.0 3.0 130.0 233.0 0.0 0.0 179.0 1.0 0.4 1.0 0.0 3.0 0.0
42.0 1.0 4.0 140.0 226.0 0.0 0.0 178.0 0.0 0.0 1.0 0.0 3.0 0.0
43.0 1.0 4.0 120.0 177.0 0.0 2.0 120.0 1.0 2.5 2.0 0.0 7.0 1.0

57.0 1.0 4.0 150.0 276.0 0.0 2.0 112.0 1.0 0.6 2.0 1.0 6.0 1.0
55.0 1.0 4.0 132.0 353.0 0.0 0.0 132.0 1.0 1.2 2.0 1.0 7.0 1.0
61.0 1.0 3.0 150.0 243.0 1.0 0.0 137.0 1.0 1.0 2.0 0.0 3.0 0.0
65.0 0.0 4.0 150.0 225.0 0.0 2.0 114.0 0.0 1.0 2.0 3.0 7.0 1.0
40.0 1.0 1.0 140.0 199.0 0.0 0.0 178.0 1.0 1.4 1.0 0.0 7.0 0.0
71.0 0.0 2.0 160.0 302.0 0.0 0.0 162.0 0.0 0.4 1.0 2.0 3.0 0.0
59.0 1.0 3.0 150.0 212.0 1.0 0.0 157.0 0.0 1.6 1.0 0.0 3.0 0.0
61.0 0.0 4.0 130.0 330.0 0.0 2.0 169.0 0.0 0.0 1.0 0.0 3.0 1.0
58.0 1.0 3.0 112.0 230.0 0.0 2.0 165.0 0.0 2.5 2.0 1.0 7.0 1.0
51.0 1.0 3.0 110.0 175.0 0.0 0.0 123.0 0.0 0.6 1.0 0.0 3.0 0.0
50.0 1.0 4.0 150.0 243.0 0.0 2.0 128.0 0.0 2.6 2.0 0.0 7.0 1.0
65.0 0.0 3.0 140.0 417.0 1.0 2.0 157.0 0.0 0.8 1.0 1.0 3.0 0.0
53.0 1.0 3.0 130.0 197.0 1.0 2.0 152.0 0.0 1.2 3.0 0.0 3.0 0.0
41.0 0.0 2.0 105.0 198.0 0.0 0.0 168.0 0.0 0.0 1.0 1.0 3.0 0.0
65.0 1.0 4.0 120.0 177.0 0.0 0.0 140.0 0.0 0.4 1.0 0.0 7.0 0.0
44.0 1.0 4.0 112.0 290.0 0.0 2.0 153.0 0.0 0.0 1.0 1.0 3.0 1.0
44.0 1.0 2.0 130.0 219.0 0.0 2.0 188.0 0.0 0.0 1.0 0.0 3.0 0.0
60.0 1.0 4.0 130.0 253.0 0.0 0.0 144.0 1.0 1.4 1.0 1.0 7.0 1.0
54.0 1.0 4.0 124.0 266.0 0.0 2.0 109.0 1.0 2.2 2.0 1.0 7.0 1.0
50.0 1.0 3.0 140.0 233.0 0.0 0.0 163.0 0.0 0.6 2.0 1.0 7.0 1.0
41.0 1.0 4.0 110.0 172.0 0.0 2.0 158.0 0.0 0.0 1.0 0.0 7.0 1.0
54.0 1.0 3.0 125.0 273.0 0.0 2.0 152.0 0.0 0.5 3.0 1.0 3.0 0.0
51.0 1.0 1.0 125.0 213.0 0.0 2.0 125.0 1.0 1.4 1.0 1.0 3.0 0.0
51.0 0.0 4.0 130.0 305.0 0.0 0.0 142.0 1.0 1.2 2.0 0.0 7.0 1.0
46.0 0.0 3.0 142.0 177.0 0.0 2.0 160.0 1.0 1.4 3.0 0.0 3.0 0.0
58.0 1.0 4.0 128.0 216.0 0.0 2.0 131.0 1.0 2.2 2.0 3.0 7.0 1.0
54.0 0.0 3.0 135.0 304.0 1.0 0.0 170.0 0.0 0.0 1.0 0.0 3.0 0.0
54.0 1.0 4.0 120.0 188.0 0.0 0.0 113.0 0.0 1.4 2.0 1.0 7.0 1.0
60.0 1.0 4.0 145.0 282.0 0.0 2.0 142.0 1.0 2.8 2.0 2.0 7.0 1.0
60.0 1.0 3.0 140.0 185.0 0.0 2.0 155.0 0.0 3.0 2.0 0.0 3.0 1.0
54.0 1.0 3.0 150.0 232.0 0.0 2.0 165.0 0.0 1.6 1.0 0.0 7.0 0.0
59.0 1.0 4.0 170.0 326.0 0.0 2.0 140.0 1.0 3.4 3.0 0.0 7.0 1.0
46.0 1.0 3.0 150.0 231.0 0.0 0.0 147.0 0.0 3.6 2.0 0.0 3.0 1.0
65.0 0.0 3.0 155.0 269.0 0.0 0.0 148.0 0.0 0.8 1.0 0.0 3.0 0.0
67.0 1.0 4.0 125.0 254.0 1.0 0.0 163.0 0.0 0.2 2.0 2.0 7.0 1.0
62.0 1.0 4.0 120.0 267.0 0.0 0.0 99.0 1.0 1.8 2.0 2.0 7.0 1.0
65.0 1.0 4.0 110.0 248.0 0.0 2.0 158.0 0.0 0.6 1.0 2.0 6.0 1.0
44.0 1.0 4.0 110.0 197.0 0.0 2.0 177.0 0.0 0.0 1.0 1.0 3.0 1.0
65.0 0.0 3.0 160.0 360.0 0.0 2.0 151.0 0.0 0.8 1.0 0.0 3.0 0.0
60.0 1.0 4.0 125.0 258.0 0.0 2.0 141.0 1.0 2.8 2.0 1.0 7.0 1.0
51.0 0.0 3.0 140.0 308.0 0.0 2.0 142.0 0.0 1.5 1.0 1.0 3.0 0.0
48.0 1.0 2.0 130.0 245.0 0.0 2.0 180.0 0.0 0.2 2.0 0.0 3.0 0.0
58.0 1.0 4.0 150.0 270.0 0.0 2.0 111.0 1.0 0.8 1.0 0.0 7.0 1.0

45.0 1.0 4.0 104.0 208.0 0.0 2.0 148.0 1.0 3.0 2.0 0.0 3.0 0.0
53.0 0.0 4.0 130.0 264.0 0.0 2.0 143.0 0.0 0.4 2.0 0.0 3.0 0.0
39.0 1.0 3.0 140.0 321.0 0.0 2.0 182.0 0.0 0.0 1.0 0.0 3.0 0.0
68.0 1.0 3.0 180.0 274.0 1.0 2.0 150.0 1.0 1.6 2.0 0.0 7.0 1.0
52.0 1.0 2.0 120.0 325.0 0.0 0.0 172.0 0.0 0.2 1.0 0.0 3.0 0.0
44.0 1.0 3.0 140.0 235.0 0.0 2.0 180.0 0.0 0.0 1.0 0.0 3.0 0.0
47.0 1.0 3.0 138.0 257.0 0.0 2.0 156.0 0.0 0.0 1.0 0.0 3.0 0.0
53.0 0.0 3.0 128.0 216.0 0.0 2.0 115.0 0.0 0.0 1.0 0.0 -9.0 0.0
53.0 0.0 4.0 138.0 234.0 0.0 2.0 160.0 0.0 0.0 1.0 0.0 3.0 0.0
51.0 0.0 3.0 130.0 256.0 0.0 2.0 149.0 0.0 0.5 1.0 0.0 3.0 0.0
66.0 1.0 4.0 120.0 302.0 0.0 2.0 151.0 0.0 0.4 2.0 0.0 3.0 0.0
62.0 0.0 4.0 160.0 164.0 0.0 2.0 145.0 0.0 6.2 3.0 3.0 7.0 1.0
62.0 1.0 3.0 130.0 231.0 0.0 0.0 146.0 0.0 1.8 2.0 3.0 7.0 0.0
44.0 0.0 3.0 108.0 141.0 0.0 0.0 175.0 0.0 0.6 2.0 0.0 3.0 0.0
63.0 0.0 3.0 135.0 252.0 0.0 2.0 172.0 0.0 0.0 1.0 0.0 3.0 0.0
52.0 1.0 4.0 128.0 255.0 0.0 0.0 161.0 1.0 0.0 1.0 1.0 7.0 1.0
59.0 1.0 4.0 110.0 239.0 0.0 2.0 142.0 1.0 1.2 2.0 1.0 7.0 1.0
60.0 0.0 4.0 150.0 258.0 0.0 2.0 157.0 0.0 2.6 2.0 2.0 7.0 1.0
52.0 1.0 2.0 134.0 201.0 0.0 0.0 158.0 0.0 0.8 1.0 1.0 3.0 0.0
48.0 1.0 4.0 122.0 222.0 0.0 2.0 186.0 0.0 0.0 1.0 0.0 3.0 0.0
45.0 1.0 4.0 115.0 260.0 0.0 2.0 185.0 0.0 0.0 1.0 0.0 3.0 0.0
34.0 1.0 1.0 118.0 182.0 0.0 2.0 174.0 0.0 0.0 1.0 0.0 3.0 0.0
57.0 0.0 4.0 128.0 303.0 0.0 2.0 159.0 0.0 0.0 1.0 1.0 3.0 0.0
71.0 0.0 3.0 110.0 265.0 1.0 2.0 130.0 0.0 0.0 1.0 1.0 3.0 0.0
49.0 1.0 3.0 120.0 188.0 0.0 0.0 139.0 0.0 2.0 2.0 3.0 7.0 1.0
54.0 1.0 2.0 108.0 309.0 0.0 0.0 156.0 0.0 0.0 1.0 0.0 7.0 0.0
59.0 1.0 4.0 140.0 177.0 0.0 0.0 162.0 1.0 0.0 1.0 1.0 7.0 1.0
57.0 1.0 3.0 128.0 229.0 0.0 2.0 150.0 0.0 0.4 2.0 1.0 7.0 1.0
61.0 1.0 4.0 120.0 260.0 0.0 0.0 140.0 1.0 3.6 2.0 1.0 7.0 1.0
39.0 1.0 4.0 118.0 219.0 0.0 0.0 140.0 0.0 1.2 2.0 0.0 7.0 1.0
61.0 0.0 4.0 145.0 307.0 0.0 2.0 146.0 1.0 1.0 2.0 0.0 7.0 1.0
56.0 1.0 4.0 125.0 249.0 1.0 2.0 144.0 1.0 1.2 2.0 1.0 3.0 1.0
52.0 1.0 1.0 118.0 186.0 0.0 2.0 190.0 0.0 0.0 2.0 0.0 6.0 0.0
43.0 0.0 4.0 132.0 341.0 1.0 2.0 136.0 1.0 3.0 2.0 0.0 7.0 1.0
62.0 0.0 3.0 130.0 263.0 0.0 0.0 97.0 0.0 1.2 2.0 1.0 7.0 1.0
41.0 1.0 2.0 135.0 203.0 0.0 0.0 132.0 0.0 0.0 2.0 0.0 6.0 0.0
58.0 1.0 3.0 140.0 211.0 1.0 2.0 165.0 0.0 0.0 1.0 0.0 3.0 0.0
35.0 0.0 4.0 138.0 183.0 0.0 0.0 182.0 0.0 1.4 1.0 0.0 3.0 0.0
63.0 1.0 4.0 130.0 330.0 1.0 2.0 132.0 1.0 1.8 1.0 3.0 7.0 1.0
65.0 1.0 4.0 135.0 254.0 0.0 2.0 127.0 0.0 2.8 2.0 1.0 7.0 1.0
48.0 1.0 4.0 130.0 256.0 1.0 2.0 150.0 1.0 0.0 1.0 2.0 7.0 1.0
63.0 0.0 4.0 150.0 407.0 0.0 2.0 154.0 0.0 4.0 2.0 3.0 7.0 1.0
51.0 1.0 3.0 100.0 222.0 0.0 0.0 143.0 1.0 1.2 2.0 0.0 3.0 0.0

55.0 1.0 4.0 140.0 217.0 0.0 0.0 111.0 1.0 5.6 3.0 0.0 7.0 1.0
65.0 1.0 1.0 138.0 282.0 1.0 2.0 174.0 0.0 1.4 2.0 1.0 3.0 1.0
45.0 0.0 2.0 130.0 234.0 0.0 2.0 175.0 0.0 0.6 2.0 0.0 3.0 0.0
56.0 0.0 4.0 200.0 288.0 1.0 2.0 133.0 1.0 4.0 3.0 2.0 7.0 1.0
54.0 1.0 4.0 110.0 239.0 0.0 0.0 126.0 1.0 2.8 2.0 1.0 7.0 1.0
44.0 1.0 2.0 120.0 220.0 0.0 0.0 170.0 0.0 0.0 1.0 0.0 3.0 0.0
62.0 0.0 4.0 124.0 209.0 0.0 0.0 163.0 0.0 0.0 1.0 0.0 3.0 0.0
54.0 1.0 3.0 120.0 258.0 0.0 2.0 147.0 0.0 0.4 2.0 0.0 7.0 0.0
51.0 1.0 3.0 94.0 227.0 0.0 0.0 154.0 1.0 0.0 1.0 1.0 7.0 0.0
29.0 1.0 2.0 130.0 204.0 0.0 2.0 202.0 0.0 0.0 1.0 0.0 3.0 0.0
51.0 1.0 4.0 140.0 261.0 0.0 2.0 186.0 1.0 0.0 1.0 0.0 3.0 0.0
43.0 0.0 3.0 122.0 213.0 0.0 0.0 165.0 0.0 0.2 2.0 0.0 3.0 0.0
55.0 0.0 2.0 135.0 250.0 0.0 2.0 161.0 0.0 1.4 2.0 0.0 3.0 0.0
70.0 1.0 4.0 145.0 174.0 0.0 0.0 125.0 1.0 2.6 3.0 0.0 7.0 1.0
62.0 1.0 2.0 120.0 281.0 0.0 2.0 103.0 0.0 1.4 2.0 1.0 7.0 1.0
35.0 1.0 4.0 120.0 198.0 0.0 0.0 130.0 1.0 1.6 2.0 0.0 7.0 1.0
51.0 1.0 3.0 125.0 245.0 1.0 2.0 166.0 0.0 2.4 2.0 0.0 3.0 0.0
59.0 1.0 2.0 140.0 221.0 0.0 0.0 164.0 1.0 0.0 1.0 0.0 3.0 0.0
59.0 1.0 1.0 170.0 288.0 0.0 2.0 159.0 0.0 0.2 2.0 0.0 7.0 1.0
52.0 1.0 2.0 128.0 205.0 1.0 0.0 184.0 0.0 0.0 1.0 0.0 3.0 0.0
64.0 1.0 3.0 125.0 309.0 0.0 0.0 131.0 1.0 1.8 2.0 0.0 7.0 1.0
58.0 1.0 3.0 105.0 240.0 0.0 2.0 154.0 1.0 0.6 2.0 0.0 7.0 0.0
47.0 1.0 3.0 108.0 243.0 0.0 0.0 152.0 0.0 0.0 1.0 0.0 3.0 1.0
57.0 1.0 4.0 165.0 289.0 1.0 2.0 124.0 0.0 1.0 2.0 3.0 7.0 1.0
41.0 1.0 3.0 112.0 250.0 0.0 0.0 179.0 0.0 0.0 1.0 0.0 3.0 0.0
45.0 1.0 2.0 128.0 308.0 0.0 2.0 170.0 0.0 0.0 1.0 0.0 3.0 0.0
60.0 0.0 3.0 102.0 318.0 0.0 0.0 160.0 0.0 0.0 1.0 1.0 3.0 0.0
52.0 1.0 1.0 152.0 298.0 1.0 0.0 178.0 0.0 1.2 2.0 0.0 7.0 0.0
42.0 0.0 4.0 102.0 265.0 0.0 2.0 122.0 0.0 0.6 2.0 0.0 3.0 0.0
67.0 0.0 3.0 115.0 564.0 0.0 2.0 160.0 0.0 1.6 2.0 0.0 7.0 0.0
55.0 1.0 4.0 160.0 289.0 0.0 2.0 145.0 1.0 0.8 2.0 1.0 7.0 1.0
64.0 1.0 4.0 120.0 246.0 0.0 2.0 96.0 1.0 2.2 3.0 1.0 3.0 1.0
70.0 1.0 4.0 130.0 322.0 0.0 2.0 109.0 0.0 2.4 2.0 3.0 3.0 1.0
51.0 1.0 4.0 140.0 299.0 0.0 0.0 173.0 1.0 1.6 1.0 0.0 7.0 1.0
58.0 1.0 4.0 125.0 300.0 0.0 2.0 171.0 0.0 0.0 1.0 2.0 7.0 1.0
60.0 1.0 4.0 140.0 293.0 0.0 2.0 170.0 0.0 1.2 2.0 2.0 7.0 1.0
68.0 1.0 3.0 118.0 277.0 0.0 0.0 151.0 0.0 1.0 1.0 1.0 7.0 0.0
46.0 1.0 2.0 101.0 197.0 1.0 0.0 156.0 0.0 0.0 1.0 0.0 7.0 0.0
77.0 1.0 4.0 125.0 304.0 0.0 2.0 162.0 1.0 0.0 1.0 3.0 3.0 1.0
54.0 0.0 3.0 110.0 214.0 0.0 0.0 158.0 0.0 1.6 2.0 0.0 3.0 0.0
58.0 0.0 4.0 100.0 248.0 0.0 2.0 122.0 0.0 1.0 2.0 0.0 3.0 0.0
48.0 1.0 3.0 124.0 255.0 1.0 0.0 175.0 0.0 0.0 1.0 2.0 3.0 0.0
57.0 1.0 4.0 132.0 207.0 0.0 0.0 168.0 1.0 0.0 1.0 0.0 7.0 0.0

52.0 1.0 3.0 138.0 223.0 0.0 0.0 169.0 0.0 0.0 1.0 -9.0 3.0 0.0
54.0 0.0 2.0 132.0 288.0 1.0 2.0 159.0 1.0 0.0 1.0 1.0 3.0 0.0
35.0 1.0 4.0 126.0 282.0 0.0 2.0 156.0 1.0 0.0 1.0 0.0 7.0 1.0
45.0 0.0 2.0 112.0 160.0 0.0 0.0 138.0 0.0 0.0 2.0 0.0 3.0 0.0
70.0 1.0 3.0 160.0 269.0 0.0 0.0 112.0 1.0 2.9 2.0 1.0 7.0 1.0
53.0 1.0 4.0 142.0 226.0 0.0 2.0 111.0 1.0 0.0 1.0 0.0 7.0 0.0
59.0 0.0 4.0 174.0 249.0 0.0 0.0 143.0 1.0 0.0 2.0 0.0 3.0 1.0
62.0 0.0 4.0 140.0 394.0 0.0 2.0 157.0 0.0 1.2 2.0 0.0 3.0 0.0
64.0 1.0 4.0 145.0 212.0 0.0 2.0 132.0 0.0 2.0 2.0 2.0 6.0 1.0
57.0 1.0 4.0 152.0 274.0 0.0 0.0 88.0 1.0 1.2 2.0 1.0 7.0 1.0
52.0 1.0 4.0 108.0 233.0 1.0 0.0 147.0 0.0 0.1 1.0 3.0 7.0 0.0
56.0 1.0 4.0 132.0 184.0 0.0 2.0 105.0 1.0 2.1 2.0 1.0 6.0 1.0
43.0 1.0 3.0 130.0 315.0 0.0 0.0 162.0 0.0 1.9 1.0 1.0 3.0 0.0
53.0 1.0 3.0 130.0 246.0 1.0 2.0 173.0 0.0 0.0 1.0 3.0 3.0 0.0
48.0 1.0 4.0 124.0 274.0 0.0 2.0 166.0 0.0 0.5 2.0 0.0 7.0 1.0
56.0 0.0 4.0 134.0 409.0 0.0 2.0 150.0 1.0 1.9 2.0 2.0 7.0 1.0
42.0 1.0 1.0 148.0 244.0 0.0 2.0 178.0 0.0 0.8 1.0 2.0 3.0 0.0
59.0 1.0 1.0 178.0 270.0 0.0 2.0 145.0 0.0 4.2 3.0 0.0 7.0 0.0
60.0 0.0 4.0 158.0 305.0 0.0 2.0 161.0 0.0 0.0 1.0 0.0 3.0 1.0
63.0 0.0 2.0 140.0 195.0 0.0 0.0 179.0 0.0 0.0 1.0 2.0 3.0 0.0
42.0 1.0 3.0 120.0 240.0 1.0 0.0 194.0 0.0 0.8 3.0 0.0 7.0 0.0
66.0 1.0 2.0 160.0 246.0 0.0 0.0 120.0 1.0 0.0 2.0 3.0 6.0 1.0
54.0 1.0 2.0 192.0 283.0 0.0 2.0 195.0 0.0 0.0 1.0 1.0 7.0 1.0
69.0 1.0 3.0 140.0 254.0 0.0 2.0 146.0 0.0 2.0 2.0 3.0 7.0 1.0
50.0 1.0 3.0 129.0 196.0 0.0 0.0 163.0 0.0 0.0 1.0 0.0 3.0 0.0
51.0 1.0 4.0 140.0 298.0 0.0 0.0 122.0 1.0 4.2 2.0 3.0 7.0 1.0
43.0 1.0 4.0 132.0 247.0 1.0 2.0 143.0 1.0 0.1 2.0 -9.0 7.0 1.0
62.0 0.0 4.0 138.0 294.0 1.0 0.0 106.0 0.0 1.9 2.0 3.0 3.0 1.0
68.0 0.0 3.0 120.0 211.0 0.0 2.0 115.0 0.0 1.5 2.0 0.0 3.0 0.0
67.0 1.0 4.0 100.0 299.0 0.0 2.0 125.0 1.0 0.9 2.0 2.0 3.0 1.0
69.0 1.0 1.0 160.0 234.0 1.0 2.0 131.0 0.0 0.1 2.0 1.0 3.0 0.0
45.0 0.0 4.0 138.0 236.0 0.0 2.0 152.0 1.0 0.2 2.0 0.0 3.0 0.0
50.0 0.0 2.0 120.0 244.0 0.0 0.0 162.0 0.0 1.1 1.0 0.0 3.0 0.0
59.0 1.0 1.0 160.0 273.0 0.0 2.0 125.0 0.0 0.0 1.0 0.0 3.0 1.0
50.0 0.0 4.0 110.0 254.0 0.0 2.0 159.0 0.0 0.0 1.0 0.0 3.0 0.0
64.0 0.0 4.0 180.0 325.0 0.0 0.0 154.0 1.0 0.0 1.0 0.0 3.0 0.0
57.0 1.0 3.0 150.0 126.0 1.0 0.0 173.0 0.0 0.2 1.0 1.0 7.0 0.0
64.0 0.0 3.0 140.0 313.0 0.0 0.0 133.0 0.0 0.2 1.0 0.0 7.0 0.0
43.0 1.0 4.0 110.0 211.0 0.0 0.0 161.0 0.0 0.0 1.0 0.0 7.0 0.0
45.0 1.0 4.0 142.0 309.0 0.0 2.0 147.0 1.0 0.0 2.0 3.0 7.0 1.0
58.0 1.0 4.0 128.0 259.0 0.0 2.0 130.0 1.0 3.0 2.0 2.0 7.0 1.0
50.0 1.0 4.0 144.0 200.0 0.0 2.0 126.0 1.0 0.9 2.0 0.0 7.0 1.0
55.0 1.0 2.0 130.0 262.0 0.0 0.0 155.0 0.0 0.0 1.0 0.0 3.0 0.0

62.0 0.0 4.0 150.0 244.0 0.0 0.0 154.0 1.0 1.4 2.0 0.0 3.0 1.0
37.0 0.0 3.0 120.0 215.0 0.0 0.0 170.0 0.0 0.0 1.0 0.0 3.0 0.0
38.0 1.0 1.0 120.0 231.0 0.0 0.0 182.0 1.0 3.8 2.0 0.0 7.0 1.0
41.0 1.0 3.0 130.0 214.0 0.0 2.0 168.0 0.0 2.0 2.0 0.0 3.0 0.0
66.0 0.0 4.0 178.0 228.0 1.0 0.0 165.0 1.0 1.0 2.0 2.0 7.0 1.0
52.0 1.0 4.0 112.0 230.0 0.0 0.0 160.0 0.0 0.0 1.0 1.0 3.0 1.0
56.0 1.0 1.0 120.0 193.0 0.0 2.0 162.0 0.0 1.9 2.0 0.0 7.0 0.0
46.0 0.0 2.0 105.0 204.0 0.0 0.0 172.0 0.0 0.0 1.0 0.0 3.0 0.0
46.0 0.0 4.0 138.0 243.0 0.0 2.0 152.0 1.0 0.0 2.0 0.0 3.0 0.0
64.0 0.0 4.0 130.0 303.0 0.0 0.0 122.0 0.0 2.0 2.0 2.0 3.0 0.0
59.0 1.0 4.0 138.0 271.0 0.0 2.0 182.0 0.0 0.0 1.0 0.0 3.0 0.0
41.0 0.0 3.0 112.0 268.0 0.0 2.0 172.0 1.0 0.0 1.0 0.0 3.0 0.0
54.0 0.0 3.0 108.0 267.0 0.0 2.0 167.0 0.0 0.0 1.0 0.0 3.0 0.0
39.0 0.0 3.0 94.0 199.0 0.0 0.0 179.0 0.0 0.0 1.0 0.0 3.0 0.0
53.0 1.0 4.0 123.0 282.0 0.0 0.0 95.0 1.0 2.0 2.0 2.0 7.0 1.0
63.0 0.0 4.0 108.0 269.0 0.0 0.0 169.0 1.0 1.8 2.0 2.0 3.0 1.0
34.0 0.0 2.0 118.0 210.0 0.0 0.0 192.0 0.0 0.7 1.0 0.0 3.0 0.0
47.0 1.0 4.0 112.0 204.0 0.0 0.0 143.0 0.0 0.1 1.0 0.0 3.0 0.0
67.0 0.0 3.0 152.0 277.0 0.0 0.0 172.0 0.0 0.0 1.0 1.0 3.0 0.0
54.0 1.0 4.0 110.0 206.0 0.0 2.0 108.0 1.0 0.0 2.0 1.0 3.0 1.0
66.0 1.0 4.0 112.0 212.0 0.0 2.0 132.0 1.0 0.1 1.0 1.0 3.0 1.0
52.0 0.0 3.0 136.0 196.0 0.0 2.0 169.0 0.0 0.1 2.0 0.0 3.0 0.0
55.0 0.0 4.0 180.0 327.0 0.0 1.0 117.0 1.0 3.4 2.0 0.0 3.0 1.0
49.0 1.0 3.0 118.0 149.0 0.0 2.0 126.0 0.0 0.8 1.0 3.0 3.0 1.0
74.0 0.0 2.0 120.0 269.0 0.0 2.0 121.0 1.0 0.2 1.0 1.0 3.0 0.0
54.0 0.0 3.0 160.0 201.0 0.0 0.0 163.0 0.0 0.0 1.0 1.0 3.0 0.0
54.0 1.0 4.0 122.0 286.0 0.0 2.0 116.0 1.0 3.2 2.0 2.0 3.0 1.0
56.0 1.0 4.0 130.0 283.0 1.0 2.0 103.0 1.0 1.6 3.0 0.0 7.0 1.0
46.0 1.0 4.0 120.0 249.0 0.0 2.0 144.0 0.0 0.8 1.0 0.0 7.0 1.0
49.0 0.0 2.0 134.0 271.0 0.0 0.0 162.0 0.0 0.0 2.0 0.0 3.0 0.0
42.0 1.0 2.0 120.0 295.0 0.0 0.0 162.0 0.0 0.0 1.0 0.0 3.0 0.0
41.0 1.0 2.0 110.0 235.0 0.0 0.0 153.0 0.0 0.0 1.0 0.0 3.0 0.0
41.0 0.0 2.0 126.0 306.0 0.0 0.0 163.0 0.0 0.0 1.0 0.0 3.0 0.0
49.0 0.0 4.0 130.0 269.0 0.0 0.0 163.0 0.0 0.0 1.0 0.0 3.0 0.0
61.0 1.0 1.0 134.0 234.0 0.0 0.0 145.0 0.0 2.6 2.0 2.0 3.0 1.0
60.0 0.0 3.0 120.0 178.0 1.0 0.0 96.0 0.0 0.0 1.0 0.0 3.0 0.0
67.0 1.0 4.0 120.0 237.0 0.0 0.0 71.0 0.0 1.0 2.0 0.0 3.0 1.0
58.0 1.0 4.0 100.0 234.0 0.0 0.0 156.0 0.0 0.1 1.0 1.0 7.0 1.0
47.0 1.0 4.0 110.0 275.0 0.0 2.0 118.0 1.0 1.0 2.0 1.0 3.0 1.0
52.0 1.0 4.0 125.0 212.0 0.0 0.0 168.0 0.0 1.0 1.0 2.0 7.0 1.0
62.0 1.0 2.0 128.0 208.0 1.0 2.0 140.0 0.0 0.0 1.0 0.0 3.0 0.0
57.0 1.0 4.0 110.0 201.0 0.0 0.0 126.0 1.0 1.5 2.0 0.0 6.0 0.0
58.0 1.0 4.0 146.0 218.0 0.0 0.0 105.0 0.0 2.0 2.0 1.0 7.0 1.0

64.0	1.0	4.0	128.0	263.0	0.0	0.0	105.0	1.0	0.2	2.0	1.0	7.0	0.0
51.0	0.0	3.0	120.0	295.0	0.0	2.0	157.0	0.0	0.6	1.0	0.0	3.0	0.0
43.0	1.0	4.0	115.0	303.0	0.0	0.0	181.0	0.0	1.2	2.0	0.0	3.0	0.0
42.0	0.0	3.0	120.0	209.0	0.0	0.0	173.0	0.0	0.0	2.0	0.0	3.0	0.0
67.0	0.0	4.0	106.0	223.0	0.0	0.0	142.0	0.0	0.3	1.0	2.0	3.0	0.0
76.0	0.0	3.0	140.0	197.0	0.0	1.0	116.0	0.0	1.1	2.0	0.0	3.0	0.0
70.0	1.0	2.0	156.0	245.0	0.0	2.0	143.0	0.0	0.0	1.0	0.0	3.0	0.0
57.0	1.0	2.0	124.0	261.0	0.0	0.0	141.0	0.0	0.3	1.0	0.0	7.0	1.0
44.0	0.0	3.0	118.0	242.0	0.0	0.0	149.0	0.0	0.3	2.0	1.0	3.0	0.0
58.0	0.0	2.0	136.0	319.0	1.0	2.0	152.0	0.0	0.0	1.0	2.0	3.0	1.0
60.0	0.0	1.0	150.0	240.0	0.0	0.0	171.0	0.0	0.9	1.0	0.0	3.0	0.0
44.0	1.0	3.0	120.0	226.0	0.0	0.0	169.0	0.0	0.0	1.0	0.0	3.0	0.0
61.0	1.0	4.0	138.0	166.0	0.0	2.0	125.0	1.0	3.6	2.0	1.0	3.0	1.0
42.0	1.0	4.0	136.0	315.0	0.0	0.0	125.0	1.0	1.8	2.0	0.0	6.0	1.0
52.0	1.0	4.0	128.0	204.0	1.0	0.0	156.0	1.0	1.0	2.0	0.0	-9.0	1.0
59.0	1.0	3.0	126.0	218.0	1.0	0.0	134.0	0.0	2.2	2.0	1.0	6.0	1.0
40.0	1.0	4.0	152.0	223.0	0.0	0.0	181.0	0.0	0.0	1.0	0.0	7.0	1.0
42.0	1.0	3.0	130.0	180.0	0.0	0.0	150.0	0.0	0.0	1.0	0.0	3.0	0.0
61.0	1.0	4.0	140.0	207.0	0.0	2.0	138.0	1.0	1.9	1.0	1.0	7.0	1.0
66.0	1.0	4.0	160.0	228.0	0.0	2.0	138.0	0.0	2.3	1.0	0.0	6.0	0.0
46.0	1.0	4.0	140.0	311.0	0.0	0.0	120.0	1.0	1.8	2.0	2.0	7.0	1.0
71.0	0.0	4.0	112.0	149.0	0.0	0.0	125.0	0.0	1.6	2.0	0.0	3.0	0.0
59.0	1.0	1.0	134.0	204.0	0.0	0.0	162.0	0.0	0.8	1.0	2.0	3.0	1.0
64.0	1.0	1.0	170.0	227.0	0.0	2.0	155.0	0.0	0.6	2.0	0.0	7.0	0.0
66.0	0.0	3.0	146.0	278.0	0.0	2.0	152.0	0.0	0.0	2.0	1.0	3.0	0.0
39.0	0.0	3.0	138.0	220.0	0.0	0.0	152.0	0.0	0.0	2.0	0.0	3.0	0.0
57.0	1.0	2.0	154.0	232.0	0.0	2.0	164.0	0.0	0.0	1.0	1.0	3.0	1.0
58.0	0.0	4.0	130.0	197.0	0.0	0.0	131.0	0.0	0.6	2.0	0.0	3.0	0.0
57.0	1.0	4.0	110.0	335.0	0.0	0.0	143.0	1.0	3.0	2.0	1.0	7.0	1.0
47.0	1.0	3.0	130.0	253.0	0.0	0.0	179.0	0.0	0.0	1.0	0.0	3.0	0.0
55.0	0.0	4.0	128.0	205.0	0.0	1.0	130.0	1.0	2.0	2.0	1.0	7.0	1.0
35.0	1.0	2.0	122.0	192.0	0.0	0.0	174.0	0.0	0.0	1.0	0.0	3.0	0.0
61.0	1.0	4.0	148.0	203.0	0.0	0.0	161.0	0.0	0.0	1.0	1.0	7.0	1.0
58.0	1.0	4.0	114.0	318.0	0.0	1.0	140.0	0.0	4.4	3.0	3.0	6.0	1.0
58.0	0.0	4.0	170.0	225.0	1.0	2.0	146.0	1.0	2.8	2.0	2.0	6.0	1.0
58.0	1.0	2.0	125.0	220.0	0.0	0.0	144.0	0.0	0.4	2.0	-9.0	7.0	0.0
56.0	1.0	2.0	130.0	221.0	0.0	2.0	163.0	0.0	0.0	1.0	0.0	7.0	0.0
56.0	1.0	2.0	120.0	240.0	0.0	0.0	169.0	0.0	0.0	3.0	0.0	3.0	0.0
67.0	1.0	3.0	152.0	212.0	0.0	2.0	150.0	0.0	0.8	2.0	0.0	7.0	1.0
55.0	0.0	2.0	132.0	342.0	0.0	0.0	166.0	0.0	1.2	1.0	0.0	3.0	0.0
44.0	1.0	4.0	120.0	169.0	0.0	0.0	144.0	1.0	2.8	3.0	0.0	6.0	1.0
63.0	1.0	4.0	140.0	187.0	0.0	2.0	144.0	1.0	4.0	1.0	2.0	7.0	1.0
63.0	0.0	4.0	124.0	197.0	0.0	0.0	136.0	1.0	0.0	2.0	0.0	3.0	1.0

41.0 1.0 2.0 120.0 157.0 0.0 0.0 182.0 0.0 0.0 1.0 0.0 3.0 0.0
59.0 1.0 4.0 164.0 176.0 1.0 2.0 90.0 0.0 1.0 2.0 2.0 6.0 1.0
57.0 0.0 4.0 140.0 241.0 0.0 0.0 123.0 1.0 0.2 2.0 0.0 7.0 1.0
45.0 1.0 1.0 110.0 264.0 0.0 0.0 132.0 0.0 1.2 2.0 0.0 7.0 1.0
68.0 1.0 4.0 144.0 193.0 1.0 0.0 141.0 0.0 3.4 2.0 2.0 7.0 1.0
57.0 1.0 4.0 130.0 131.0 0.0 0.0 115.0 1.0 1.2 2.0 1.0 7.0 1.0
57.0 0.0 2.0 130.0 236.0 0.0 2.0 174.0 0.0 0.0 2.0 1.0 3.0 1.0
38.0 1.0 3.0 138.0 175.0 0.0 0.0 173.0 0.0 0.0 1.0 -9.0 3.0 0.0