Universidade de São Paulo
Escola de Engenharia de São Carlos
Departamento de Engenharia Elétrica

Alunos: Luisa Helena Bartocci Liboni
Rodrigo de Toledo Caropreso

Data de Entrega: 07/05/2012

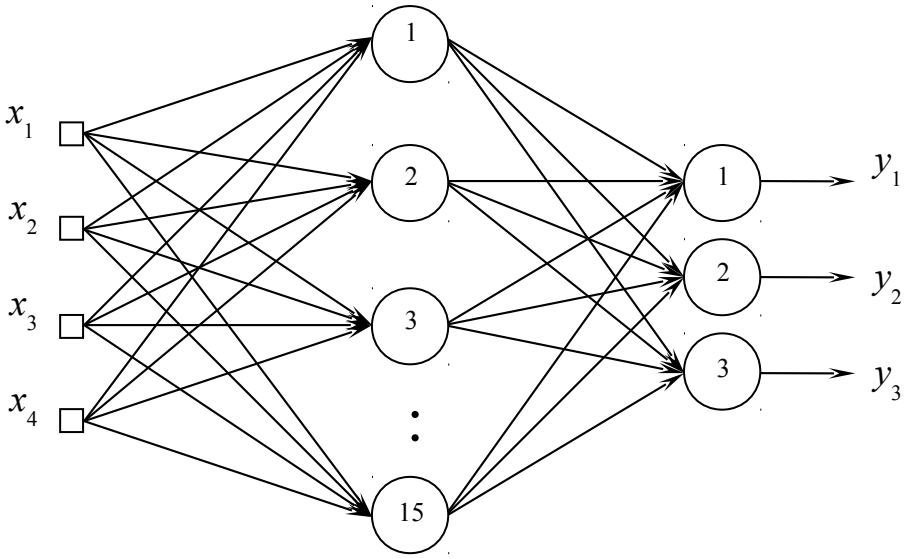
Redes Neurais Artificiais

(Prof. Ivan Nunes da Silva)

EPC-5

No processamento de bebidas, a aplicação de um determinado conservante é efetuada em função da combinação de 04 variáveis reais, definidas por x_1 (teor de água), x_2 (grau de acidez), x_3 (temperatura) e x_4 (tensão superficial). Sabe-se que existem apenas três tipos de conservantes que podem ser aplicados, os quais são categorizados por tipo A, B e C. A partir destas variáveis, realizam-se ensaios em laboratório para especificar que tipo de conservante deve ser aplicado em determinada bebida.

Por intermédio de 148 desses ensaios experimentais, a equipe de engenheiros e cientistas resolveu aplicar uma rede perceptron multicamadas como classificadora de padrões, a fim de que esta identifique qual conservante será aplicado em determinado lote de bebida. Por questões operacionais da própria linha de produção, utilizar-se-á aqui uma rede perceptron com três saídas, conforme apresentado na figura abaixo.



A padronização para a saída, representando o conservante a ser aplicado, ficou definida da seguinte forma:

Tipo de Conservante	y_1	y_2	y_3
Tipo A	1	0	0
Tipo B	0	1	0
Tipo C	0	0	1

Utilizando os dados de treinamento apresentados no Anexo, execute o treinamento de uma rede perceptron multicamadas (04 entradas, 15 neurônios na camada escondida e 03 saídas) que possa classificar, em função apenas dos valores medidos de x_1, x_2, x_3 e x_4 (já normalizados), qual o tipo de conservante que deverá ser aplicado em determinada bebida. Para tanto, faça as seguintes atividades:

1. Execute cinco treinamentos da rede perceptron multicamadas ilustrada na Figura 1, por meio do algoritmo de aprendizagem ***backpropagation***, inicializando-se todas as matrizes de pesos com valores aleatórios entre 0 e 1. Utilize a função de ativação logística para todos os neurônios, taxa de aprendizado $\eta = 0.1$ e precisão $\varepsilon = 10^{-6}$. Meça também o tempo de processamento envolvido com cada um desses treinamentos.

Treinamento	Número de Épocas	Tempo de Treinamento (s)
T1	1121	16.0683
T2	1161	15.9819
T3	1097	15.2847
T4	1238	16.9844
T5	1172	16.5351

2. Dado que o problema se configura como um típico processo de classificação de padrões, implemente a rotina que faz o pós-processamento das saídas fornecidas pela rede (números reais) para números inteiros. Utilize o critério do arredondamento simétrico, isto é:

$$y_i^{pós} = \begin{cases} 1, se\ y_i \geq 0.5 \\ 0, se\ y_i < 0.5 \end{cases}$$, utilizado apenas no pós-processamento do conjunto de teste.

```
total_acertos = 0;
for k=1: N_Amostras
        y(:, k) = MLP_Executa( W_1, W_2, [-1 DB_X1_Norm(k) DB_X2_Norm(k)
DB_X3_Norm(k) DB_X4_Norm(k)]' );

    %Pos processamento
    acertos_por_saida = 0;
    for i=1: N_Saidas
        if( y(i, k) >= 0.5 )
            yp(i, k) = 1;
        else
            yp(i, k) = 0;
        end;

        %Compara as saidas, computando acertos a cada neuronio
        if( yp(i,k) == d(i,k) )
            acertos_por_saida = acertos_por_saida + 1;
        end;
    end;

    %Se todas as saidas sao iguais às desejadas, temos um acerto geral
    if( acertos_por_saida == N_Saidas )
        total_acertos = total_acertos + 1;
    end;
end;
```

3. Para cada um dos cinco treinamentos, faça então a validação aplicando o conjunto de teste fornecido na tabela abaixo. Forneça a taxa de acerto (%) entre os valores desejados e os valores fornecidos pela rede (após o pós-processamento) em relação a todas as amostras de teste.

Treinamento: T1

Amostra	x_1	x_2	x_3	x_4	d_1	d_2	d_3	$y_1^{pós}$	$y_2^{pós}$	$y_3^{pós}$	y_1	y_2	y_3
1	0.8622	0.7101	0.6236	0.7894	0	0	1	0	0	1	0.0000	0.0043	0.9959
2	0.2741	0.1552	0.1333	0.1516	1	0	0	1	0	0	0.9991	0.0012	0.0000
3	0.6772	0.8516	0.6543	0.7573	0	0	1	0	0	1	0.0000	0.0042	0.9963
4	0.2178	0.5039	0.6415	0.5039	0	1	0	0	1	0	0.0037	0.9865	0.0011
5	0.7260	0.7500	0.7007	0.4953	0	0	1	0	0	1	0.0000	0.0873	0.9251
6	0.2473	0.2941	0.4248	0.3087	1	0	0	1	0	0	0.9397	0.0621	0.0000
7	0.5682	0.5683	0.5054	0.4426	0	1	0	0	1	0	0.0013	0.9876	0.0028
8	0.6566	0.6715	0.4952	0.3951	0	1	0	0	1	0	0.0004	0.9813	0.0098
9	0.0705	0.4717	0.2921	0.2954	1	0	0	1	0	0	0.9758	0.0298	0.0000
10	0.1187	0.2568	0.3140	0.3037	1	0	0	1	0	0	0.9948	0.0054	0.0000
11	0.5673	0.7011	0.4083	0.5552	0	1	0	0	1	0	0.0004	0.9809	0.0100
12	0.3164	0.2251	0.3526	0.2560	1	0	0	1	0	0	0.9875	0.0117	0.0000
13	0.7884	0.9568	0.6825	0.6398	0	0	1	0	0	1	0.0000	0.0016	0.9986
14	0.9633	0.7850	0.6777	0.6059	0	0	1	0	0	1	0.0000	0.0028	0.9973
15	0.7739	0.8505	0.7934	0.6626	0	0	1	0	0	1	0.0000	0.0012	0.9991
16	0.4219	0.4136	0.1408	0.0940	1	0	0	1	0	0	0.9938	0.0066	0.0000
17	0.6616	0.4365	0.6597	0.8129	0	0	1	0	0	1	0.0000	0.3087	0.7125
18	0.7325	0.4761	0.3888	0.5683	0	1	0	0	1	0	0.0015	0.9842	0.0026
Taxa de Acerto (%): 100%													

Treinamento: T2

Amostra	x_1	x_2	x_3	x_4	d_1	d_2	d_3	$y_1^{pós}$	$y_2^{pós}$	$y_3^{pós}$	y_1	y_2	y_3
1	0.8622	0.7101	0.6236	0.7894	0	0	1	0	0	1	0.0000	0.0037	0.9968
2	0.2741	0.1552	0.1333	0.1516	1	0	0	1	0	0	0.9989	0.0029	0.0000
3	0.6772	0.8516	0.6543	0.7573	0	0	1	0	0	1	0.0000	0.0036	0.9972
4	0.2178	0.5039	0.6415	0.5039	0	1	0	0	1	0	0.0028	0.9874	0.0010
5	0.7260	0.7500	0.7007	0.4953	0	0	1	0	0	1	0.0000	0.0902	0.9261
6	0.2473	0.2941	0.4248	0.3087	1	0	0	1	0	0	0.9400	0.0622	0.0000
7	0.5682	0.5683	0.5054	0.4426	0	1	0	0	1	0	0.0009	0.9880	0.0024
8	0.6566	0.6715	0.4952	0.3951	0	1	0	0	1	0	0.0002	0.9808	0.0089
9	0.0705	0.4717	0.2921	0.2954	1	0	0	1	0	0	0.9747	0.0289	0.0000
10	0.1187	0.2568	0.3140	0.3037	1	0	0	1	0	0	0.9941	0.0075	0.0000
11	0.5673	0.7011	0.4083	0.5552	0	1	0	0	1	0	0.0002	0.9799	0.0094
12	0.3164	0.2251	0.3526	0.2560	1	0	0	1	0	0	0.9865	0.0151	0.0000
13	0.7884	0.9568	0.6825	0.6398	0	0	1	0	0	1	0.0000	0.0012	0.9991
14	0.9633	0.7850	0.6777	0.6059	0	0	1	0	0	1	0.0000	0.0024	0.9980
15	0.7739	0.8505	0.7934	0.6626	0	0	1	0	0	1	0.0000	0.0008	0.9994
16	0.4219	0.4136	0.1408	0.0940	1	0	0	1	0	0	0.9928	0.0104	0.0000
17	0.6616	0.4365	0.6597	0.8129	0	0	1	0	0	1	0.0000	0.3058	0.7133
18	0.7325	0.4761	0.3888	0.5683	0	1	0	0	1	0	0.0010	0.9847	0.0023
Taxa de Acerto (%): 100%													

Treinamento: T3

Amostra	x_1	x_2	x_3	x_4	d_1	d_2	d_3	$y_1^{pós}$	$y_2^{pós}$	$y_3^{pós}$	y_1	y_2	y_3
1	0.8622	0.7101	0.6236	0.7894	0	0	1	0	0	1	0.0000	0.0035	0.9961
2	0.2741	0.1552	0.1333	0.1516	1	0	0	1	0	0	0.9994	0.0003	0.0000
3	0.6772	0.8516	0.6543	0.7573	0	0	1	0	0	1	0.0000	0.0034	0.9965
4	0.2178	0.5039	0.6415	0.5039	0	1	0	0	1	0	0.0040	0.9864	0.0014
5	0.7260	0.7500	0.7007	0.4953	0	0	1	0	0	1	0.0000	0.0887	0.9246
6	0.2473	0.2941	0.4248	0.3087	1	0	0	1	0	0	0.9391	0.0569	0.0000
7	0.5682	0.5683	0.5054	0.4426	0	1	0	0	1	0	0.0013	0.9873	0.0032
8	0.6566	0.6715	0.4952	0.3951	0	1	0	0	1	0	0.0004	0.9813	0.0109
9	0.0705	0.4717	0.2921	0.2954	1	0	0	1	0	0	0.9754	0.0211	0.0000
10	0.1187	0.2568	0.3140	0.3037	1	0	0	1	0	0	0.9956	0.0026	0.0000
11	0.5673	0.7011	0.4083	0.5552	0	1	0	0	1	0	0.0004	0.9803	0.0113
12	0.3164	0.2251	0.3526	0.2560	1	0	0	1	0	0	0.9889	0.0078	0.0000
13	0.7884	0.9568	0.6825	0.6398	0	0	1	0	0	1	0.0000	0.0012	0.9987
14	0.9633	0.7850	0.6777	0.6059	0	0	1	0	0	1	0.0000	0.0023	0.9975
15	0.7739	0.8505	0.7934	0.6626	0	0	1	0	0	1	0.0000	0.0008	0.9991
16	0.4219	0.4136	0.1408	0.0940	1	0	0	1	0	0	0.9950	0.0035	0.0000
17	0.6616	0.4365	0.6597	0.8129	0	0	1	0	0	1	0.0000	0.2991	0.6966
18	0.7325	0.4761	0.3888	0.5683	0	1	0	0	1	0	0.0015	0.9830	0.0030
Taxa de Acerto (%): 100%													

Treinamento:T4

Amostra	x_1	x_2	x_3	x_4	d_1	d_2	d_3	$y_1^{pós}$	$y_2^{pós}$	$y_3^{pós}$	y_1	y_2	y_3
1	0.8622	0.7101	0.6236	0.7894	0	0	1	0	0	1	0.0000	0.0040	0.9974
2	0.2741	0.1552	0.1333	0.1516	1	0	0	1	0	0	0.9973	0.0074	0.0000
3	0.6772	0.8516	0.6543	0.7573	0	0	1	0	0	1	0.0000	0.0039	0.9977
4	0.2178	0.5039	0.6415	0.5039	0	1	0	0	1	0	0.0028	0.9901	0.0009
5	0.7260	0.7500	0.7007	0.4953	0	0	1	0	0	1	0.0000	0.0932	0.9302
6	0.2473	0.2941	0.4248	0.3087	1	0	0	1	0	0	0.9347	0.0672	0.0000
7	0.5682	0.5683	0.5054	0.4426	0	1	0	0	1	0	0.0009	0.9898	0.0022
8	0.6566	0.6715	0.4952	0.3951	0	1	0	0	1	0	0.0003	0.9814	0.0084
9	0.0705	0.4717	0.2921	0.2954	1	0	0	1	0	0	0.9684	0.0398	0.0000
10	0.1187	0.2568	0.3140	0.3037	1	0	0	1	0	0	0.9898	0.0146	0.0000
11	0.5673	0.7011	0.4083	0.5552	0	1	0	0	1	0	0.0003	0.9805	0.0086
12	0.3164	0.2251	0.3526	0.2560	1	0	0	1	0	0	0.9804	0.0234	0.0000
13	0.7884	0.9568	0.6825	0.6398	0	0	1	0	0	1	0.0000	0.0013	0.9993
14	0.9633	0.7850	0.6777	0.6059	0	0	1	0	0	1	0.0000	0.0026	0.9984
15	0.7739	0.8505	0.7934	0.6626	0	0	1	0	0	1	0.0000	0.0008	0.9996
16	0.4219	0.4136	0.1408	0.0940	1	0	0	1	0	0	0.9872	0.0214	0.0000
17	0.6616	0.4365	0.6597	0.8129	0	0	1	0	0	1	0.0000	0.3022	0.7080
18	0.7325	0.4761	0.3888	0.5683	0	1	0	0	1	0	0.0010	0.9879	0.0021
Taxa de Acerto (%): 100%													

Treinamento: T5

Amostra	x_1	x_2	x_3	x_4	d_1	d_2	d_3	$y_1^{pós}$	$y_2^{pós}$	$y_3^{pós}$	y_1	y_2	y_3
1	0.8622	0.7101	0.6236	0.7894	0	0	1	0	0	1	0.0000	0.0043	0.9965
2	0.2741	0.1552	0.1333	0.1516	1	0	0	1	0	0	0.9984	0.0041	0.0000
3	0.6772	0.8516	0.6543	0.7573	0	0	1	0	0	1	0.0000	0.0042	0.9971
4	0.2178	0.5039	0.6415	0.5039	0	1	0	0	1	0	0.0030	0.9869	0.0010
5	0.7260	0.7500	0.7007	0.4953	0	0	1	0	0	1	0.0000	0.0870	0.9284
6	0.2473	0.2941	0.4248	0.3087	1	0	0	1	0	0	0.9364	0.0645	0.0000
7	0.5682	0.5683	0.5054	0.4426	0	1	0	0	1	0	0.0010	0.9881	0.0025
8	0.6566	0.6715	0.4952	0.3951	0	1	0	0	1	0	0.0003	0.9803	0.0090
9	0.0705	0.4717	0.2921	0.2954	1	0	0	1	0	0	0.9704	0.0343	0.0000
10	0.1187	0.2568	0.3140	0.3037	1	0	0	1	0	0	0.9922	0.0101	0.0000
11	0.5673	0.7011	0.4083	0.5552	0	1	0	0	1	0	0.0003	0.9795	0.0094
12	0.3164	0.2251	0.3526	0.2560	1	0	0	1	0	0	0.9844	0.0177	0.0000
13	0.7884	0.9568	0.6825	0.6398	0	0	1	0	0	1	0.0000	0.0015	0.9990
14	0.9633	0.7850	0.6777	0.6059	0	0	1	0	0	1	0.0000	0.0028	0.9978
15	0.7739	0.8505	0.7934	0.6626	0	0	1	0	0	1	0.0000	0.0011	0.9994
16	0.4219	0.4136	0.1408	0.0940	1	0	0	1	0	0	0.9916	0.0123	0.0000
17	0.6616	0.4365	0.6597	0.8129	0	0	1	0	0	1	0.0000	0.2963	0.7064
18	0.7325	0.4761	0.3888	0.5683	0	1	0	0	1	0	0.0012	0.9855	0.0022
Taxa de Acerto (%): 100%													

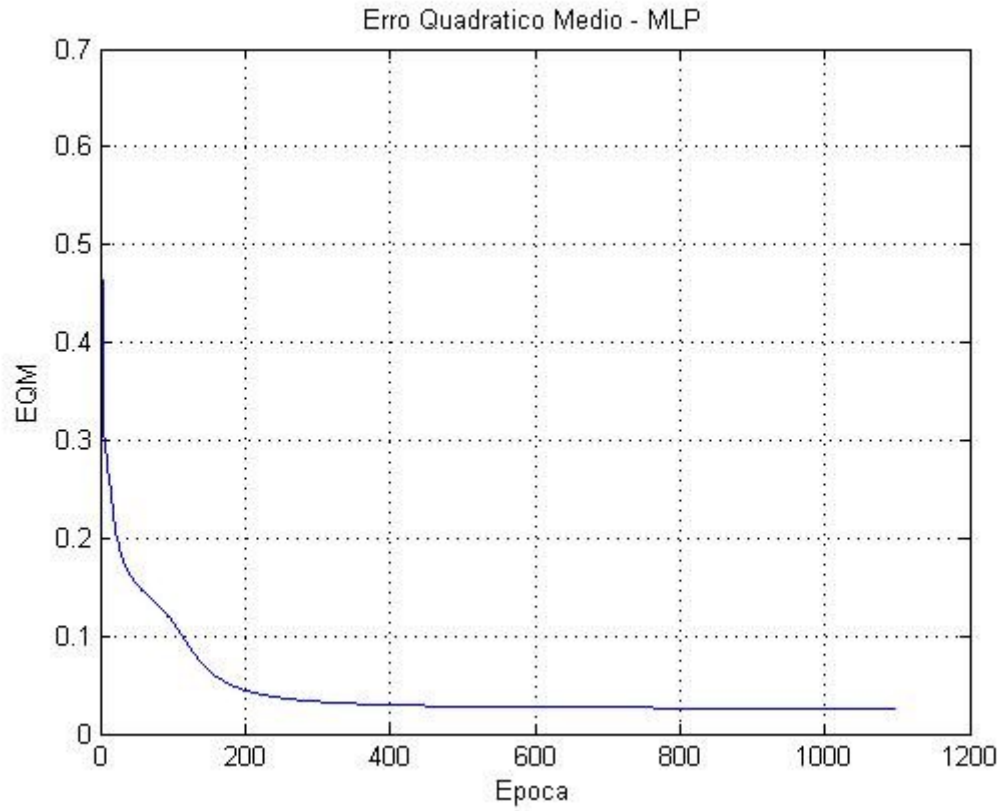
4. Explique qual foi o motivo de se realizar cinco treinamentos para uma mesma configuração topológica de rede perceptron multicamadas.

É necessário realizar uma quantidade maior de treinamentos a fim de identificar o treinamento com melhor desempenho, uma vez que a inicialização aleatória dos pesos faz com que a rede inicie o treinamento em pontos diferentes da superfície de erro.

Realizando vários treinamentos, é possível evitar a utilização de uma rede mal treinada (por ter atingido um mínimo local) e selecionar um conjunto de pesos que garanta uma taxa de acerto dentro de padrões aceitáveis.

5. Para o melhor dos cinco treinamentos realizados acima, trace o respectivo gráfico dos valores de erro quadrático médio (EQM) em função de cada época de treinamento.

Dentre os treinamentos realizados, todos obtiveram 100% de acerto. Dentre eles, o treinamento T3 foi o que convergiu mais rápido (atingiu o valor definido como critério de parada em menos épocas). Segue abaixo o gráfico do erro para T3:



CÓDIGO FONTE UTILIZADO

Carregamento dos Dados

```
[N_Amostra DB_X1 DB_X2 DB_X3] = textread( 'Dados_Operacao.dat', '%d %f %f %f %f', 'headerlines', 1);

[N_Amostra DB_X1 DB_X2 DB_X3 DB_D] = textread( 'Dados_Treino.dat', '%d %f %f %f %f', 'headerlines', 1);
```

Treinamento da Rede MLP

```
function [W_1, W_2, eqm] = MLP_Treino( eta, epson, entradas, saidas, max_epocas, n_camadas, size_Camadas  )
%MLP_Treino Treinamento de MLP
%   eta          -> coeficiente de treino
%   epson        -> margem de erro
%   entradas     -> matriz com entradas
%   saidas       -> vetor com saidas desejadas
%   max_epocas   -> limite de epocas de treinamento
%   n_camadas    -> numero de camadas neurais da rede MLP
%   size_camadas -> vetor-linha com a quantidade de neuronios em cada
%   camada

sizeW = size(entradas);
N_entradas = sizeW(1);
N_amostras = sizeW(2);

% pesos = rand(N_entradas, n_camadas);
```

```
% Wji = j-esimo neuron de uma cada
ao i-esimo sinal da camada de
entrada (na primeira matriz de
pesos)
%assim, todos os sinais de entrada
de um neuronio ficam na linha, cada
%linha contem 1 neuronio

W_1 = rand(15, 5); %15 neurons = 15
linhas; (4 entradas + 1 bias) = 5
colunas
W_2 = rand(3, 16); %Matriz de pesos
da camada 2 - 1 neuron = 1 linha (+1
bias);

% entradas'
% pause


%
% amostrasTreinamento = [entradas;
saidas]';

% 11 entradas (10 neurons da camada
anterior + 1 bias) = 11 colunas

% I_1 -> vetor que possui, em cada
posição, a entrada ponderada em
relação
% ao j-esimo neuronio da camada 1
(entrada) -> soma ponderada das
entradas no
% neuronio 'j'

disp('Inicialização da Rede MLP -
Pesos (Pressione uma tecla para
continuar)');

%inicio do treinamento
epoca = 1;
eqm(epoca) = 1 + epson;
stop = 0;

nCamadaOculta = size_Camadas(1);
nCamadaSaida  = size_Camadas(2);

%TESTE GERAL
difEQM = 1;
EQM_Atual = EQM( entradas, saidas,
W_1, W_2 );
eqm(epoca) = EQM_Atual;

while (epson < difEQM && epoca <
max_epocas)
    EQM_Anterior = EQM_Atual;

    %1a Camada: Entrada -> Neurons
da camada oculta
    for k=1:N_amostras
        %TESTE - FORWARD
        X = entradas( :, k );
        d = saidas  ( :, k );

        I_1 = W_1 * X;
        Y_1 = logsig( I_1 );

        %Ajeita o vetor de saida,
adicionando o bias
         sizeY = size(Y_1);
         Y_1 = [ (-1) * ones( 1,
sizeY(2) ); Y_1 ]; %adiciona uma
linha de -1 à matriz Y_1

        %2a Camada: Neurons Camada
Oculta -> Neurons de Saida
        I_2 = W_2 * Y_1;
        Y_2 = logsig(I_2);

%         Y_2 = I_2; %usnado função
linear na saida
        %TESTE - FORWARD - FIM

%         %Remove a 1a linha de Y_1
(bias nao entra no bakcpropag)
%         Y_1( 1, : ) = [];

        %TESTE - BACKWARD
        %Camada 2
        size_Camada_Saida = 3;
%saida
        delta_2 =
GradienteLocalDeSaida( size_Camada_S
aida, d, Y_2 );

        %Atualiza pesos da camada de
saida
        sizeW = size(W_2);
        for j=1:sizeW(1) %cada linha
é um neuron
            for i=1:sizeW(2) %cada
coluna é uma sinapse
                W_2(j,i) = W_2(j,i)
+ eta * delta_2(j) * Y_1(i);
            end;
        end;

        %Camada 1
        size_Camada_Oculta = 15;
        delta_1 =
GradienteLocalIntermediario( size_Ca
```

```
mada_Oculta, size_Camada_Saida,
delta_2, W_2, Y_1 );

        %Atualiza pesos de uma
camada de entrada
        sizeW = size(W_1);
        for j=1:sizeW(1) %cada linha
é um neuron
            for i=1:sizeW(2) %cada
coluna é uma sinapse
                W_1(j,i) = W_1(j,i)
+ eta * delta_1(j) * X(i);
            end;
        end;
        %TESTE - BACKWARD - FIM

    end;

    %Calcula o Erro
    EQM_Atual = EQM( entradas,
saidas, W_1, W_2 );
    EQM_Atual
%     if( EQM_Atual ~=
EQM_Atual_Mossim )
%         disp 'Erro EQM'
%     end;

    eqm(epoca) = EQM_Atual;

    difEQM = abs (EQM_Atual -
EQM_Anterior);

    epoca = epoca + 1;
end;

%TESTE GERAL - FIM

if( epoca < max_epocas )
    disp( sprintf( 'Rede treinada.
Numero de epocas: %d', epoca) );
else
    disp( sprintf( 'Limite de epocas
atingido (%d), rede nao treinada.',
epoca) );
end;

function EM = EQM( X, d, W_1, W_2 )
%EQM Calcula o erro quadratico medio
% X  -> entradas do MLP
% d  -> saidas desejadas

sizeW = size(X);
N_entradas = sizeW(1);
N_amostras = sizeW(2);

%Repete o FeedForward para calcular
o ERRO
% for k=1:N_amostras
%FeedForward
%1a Camada: Entrada -> Neurons da
camada oculta
I_1 = W_1 * X;
Y_1 = logsig( I_1 );

%Ajeita o vetor de saida,
adicionando o bias
 sizeY = size(Y_1);
 Y_1 = [ (-1) * ones( 1, sizeY(2) );
Y_1 ]; %adiciona uma linha de -1 à
matriz Y_1

%2a Camada: Neurons Camada Oculta ->
Neurons de Saida
I_2 = W_2 * Y_1;
Y_2 = logsig( I_2 );
% Y_2 = I_2; %usando função linear
na saida

% Calcula o erro para as amostras
atuais
E_k = ( (d - Y_2) .^2 ) / 2;
E_k = sum( E_k, 1 ); %soma os
elementos de cada coluna entre si

EM  = sum( E_k ) / N_amostras;


function delta_saida =
GradienteLocalDeSaida( size_camada,
d, Y )
%  saidas_desejadas -> vetor com
amostras de saida para a epoca atual
%  saidas_MLP       -> vetor com a
saida de cada neuronio
%  entradas         -> vetor com o
valor de entrada em cada neuron,
antes
%  da função de ativação


%a derivada da função de ativação
logistica (chamada g, por exemplo)
é:
% g' = g * (1 - g)
for j=1:size_camada
    delta_saida(j) = ( d(j) - Y(j) )
* ( Y(j) * ( 1 - Y(j) ) );
%     delta_saida(j) = ( d(j) -
Y(j) ); %usando função linear
(aproximador de funções)
```

```
end;


function delta_saida =
GradienteLocalIntermediario( size_ca
mada, size_camada_posterior,
delta_posterior, W, Y )
% Y                -> vetor com a
saida de cada neuronio na camada
dada por 'camada_L'
% delta_posterior -> gradiente local
da camada posterior (ou seja,
camada_L + 1)
% W                -> matriz de pesos
da camada posterior (ou seja, camada
L + 1 )

%a derivada da função de ativação
logistica (chamada g, por exemplo)
é:
% g' = alfa * g * (1 - g)
% como g está calculada para o ponto
em questão g' = a * y ( 1- y )

%a primeira coluna da matriz de
pesos contem as sinapses do bias,
que nao devem ser levadas em
%conta na correção do erro
(portanto, usa-se W( k, j+1 )

%a primeira linha da matriz Y contém
o bias, que nao deve ser usado na
%correção (portanto, usa-se Y(j+1) )

soma = delta_posterior * W; %o 1o
elemento conterá resultado do bias,
nao levar em conta

for j=1:size_camada
    delta_saida(j) = 0;
%     soma = 0;
%     for k=1:size_camada_posterior
%         soma = soma +
delta_posterior(k) * W(k, j+1);
%     end;
    delta_saida(j) = soma(j+1) *
( Y(j+1) * ( 1 - Y(j+1) ) );
end;


function y = MLP_Executa( W_1, W_2,
X )
%Perceptron_Executa Operacao de
Perceptron multi camada
%   entradas    -> vetor com uma
entrada
%   pesos       -> matriz de pesos
do treinamento

sizeW1 = size(W_1);
sizeW2 = size(W_2);

size_Camadas = [ sizeW1(1)
sizeW2(1)];

I_1 = W_1 * X;
Y_1 = logsig( I_1 );

%Ajeita o vetor de saida,
adicionando o bias
sizeY = size(Y_1);
Y_1 = [ (-1) * ones( 1, sizeY(2) );
Y_1 ]; %adiciona uma linha de -1 à
matriz Y_1

%2a Camada: Neurons Camada Oculta ->
Neurons de Saida
I_2 = W_2 * Y_1;
Y_2 = logsig( I_2 );
% Y_2 = I_2; %usando função linear
na saida

y = Y_2;
```