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

Alunos: Luisa Helena Bartocci Liboni
Rodrigo de Toledo Caropreso

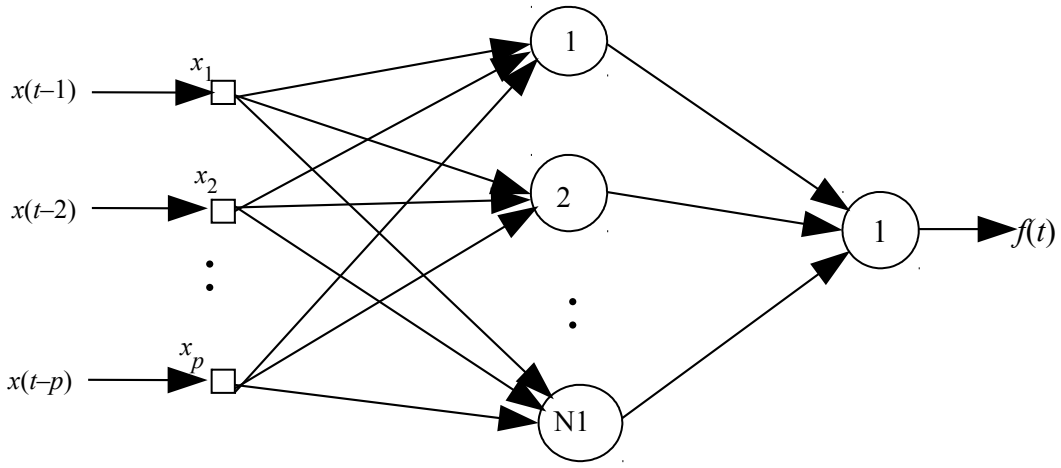
Data de Entrega: 14/05/2012

Redes Neurais Artificiais
(Prof. Ivan Nunes da Silva)

EPC-6

O preço de uma determinada mercadoria disposta para ser comercializada no mercado financeiro de ações possui um histórico de variação de valor conforme mostrado na tabela apresentada no Anexo.

Um pool de pesquisadores estará tentando aplicar redes neurais para tentar prever o comportamento futuro deste processo. Assim, pretende-se utilizar uma arquitetura perceptron multicamadas, com topologia "Time Delay" (TDNN), conforme mostrada na figura abaixo:



As topologias candidatas para serem adotadas no mapeamento do problema acima são especificadas como se segue:

Rede 1 → 05 entradas (p = 05) com N1 = 10
Rede 2 → 10 entradas (p = 10) com N1 = 15
Rede 3 → 15 entradas (p = 15) com N1 = 25

Utilizando o algoritmo de aprendizagem *backpropagation com momentum* e os dados de treinamento apresentados no Anexo, realize as seguintes atividades:

1. Execute 3 treinamentos para cada rede perceptron acima, inicializando-se as matrizes de pesos em cada treinamento com valores aleatórios entre 0 e 1. Se for o caso, reinicie o gerador de números aleatórios em cada treinamento de tal forma que os elementos das matrizes de pesos iniciais não sejam os mesmos. Utilize a função de ativação logística para todos os neurônios, taxa de aprendizado $\eta = 0.1$, fator de momentum $\alpha = 0.8$ e precisão $\varepsilon = 0.5 \times 10^{-6}$.

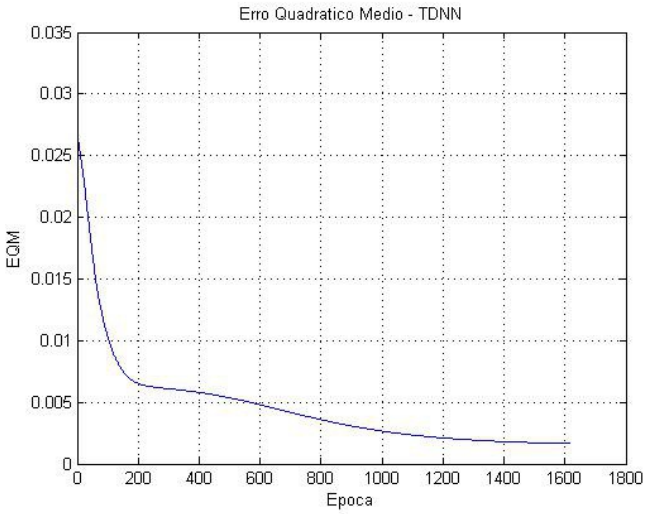
2.Registre os resultados finais desses 3 treinamentos, considerando-se cada uma dessas três topologias de rede, na tabela a seguir:

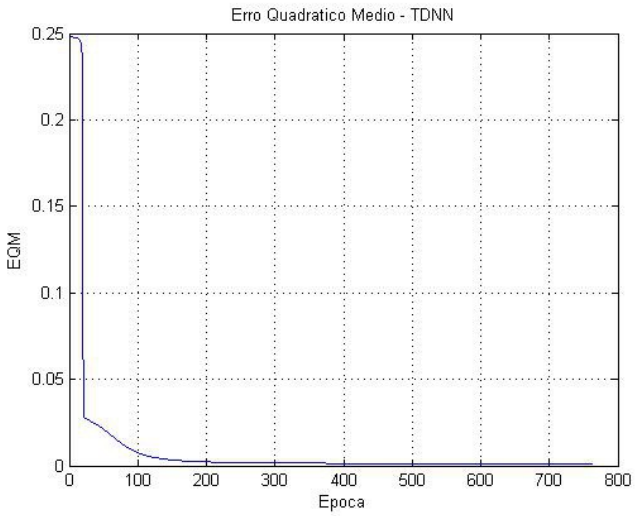
Treinamento	**Rede 1**		**Rede 2**		**Rede 3**	
	EQM	Épocas	EQM	Épocas	EQM	Épocas
1º (T1)	0.0017	1912	0.0010	808	0.0011	888
2º (T2)	0.0017	1615	0.0011	764	0.0009	841
3º (T3)	0.0018	1901	0.0010	770	0.0009	849

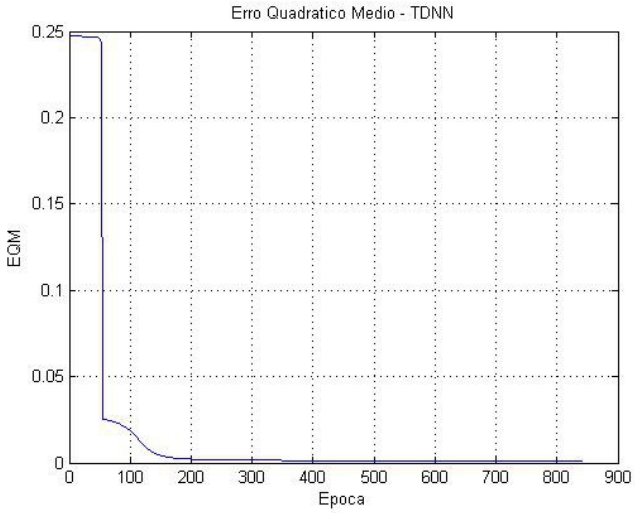
3.Para todos os treinamentos efetuados no item 2, faça então a validação da rede em relação aos valores desejados apresentados na tabela abaixo. Forneça para cada treinamento o erro relativo médio entre os valores desejados e os valores fornecidos pela rede em relação a todas as amostras de teste. Obtenha também a respectiva variância.

		Rede 1			**Rede 2**			**Rede 3**		
Amostra	*f*(*t*)	(T1)	(T2)	(T3)	(T1)	(T2)	(T3)	(T1)	(T2)	(T3)
t = 101	0.4173	0.4730	0.4797	0.4817	0.4782	0.4921	0.4793	0.4406	0.4481	0.4560
t = 102	0.0062	0.0261	0.0297	0.0281	0.0233	0.0256	0.0221	0.0270	0.0249	0.0254
t = 103	0.3387	0.3974	0.3983	0.3954	0.3672	0.3631	0.3711	0.3851	0.3982	0.3901
t = 104	0.1886	0.2195	0.2263	0.2298	0.1515	0.1475	0.1465	0.1524	0.1482	0.1373
t = 105	0.7418	0.7022	0.6948	0.6956	0.7347	0.7382	0.7449	0.7649	0.7630	0.7467
t = 106	0.3138	0.1898	0.1837	0.1875	0.2132	0.2134	0.2249	0.2049	0.2136	0.2112
t = 107	0.4466	0.4145	0.4096	0.4103	0.4471	0.4476	0.4475	0.4661	0.4563	0.4512
t = 108	0.0835	0.1033	0.0965	0.0996	0.1334	0.1310	0.1268	0.1071	0.0947	0.0915
t = 109	0.1930	0.2319	0.2306	0.2330	0.1810	0.1762	0.1895	0.1888	0.1859	0.1905
t = 110	0.3807	0.2639	0.2655	0.2577	0.4233	0.4190	0.4279	0.4130	0.4162	0.4207
t = 111	0.5438	0.5489	0.5486	0.5474	0.5410	0.5449	0.5338	0.4961	0.4991	0.5046
t = 112	0.5897	0.6436	0.6398	0.6303	0.5861	0.5849	0.5835	0.6068	0.6130	0.6130
t = 113	0.3536	0.3350	0.3435	0.3349	0.2785	0.2709	0.2706	0.2764	0.2888	0.2845
t = 114	0.2210	0.2077	0.2142	0.2145	0.2297	0.2273	0.2268	0.2267	0.2191	0.2145
t = 115	0.0631	0.1790	0.1715	0.1755	0.0534	0.0538	0.0570	0.0586	0.0555	0.0570
t = 116	0.4499	0.3844	0.3902	0.3828	0.3980	0.4020	0.3981	0.4030	0.4088	0.4111
t = 117	0.2564	0.2786	0.2801	0.2860	0.2886	0.2905	0.2841	0.2643	0.2591	0.2638
t = 118	0.7642	0.7897	0.7874	0.7910	0.7713	0.7635	0.7715	0.7592	0.7443	0.7527
t = 119	0.1411	0.1310	0.1288	0.1360	0.1570	0.1568	0.1593	0.1724	0.1715	0.1723
t = 120	0.3626	0.3290	0.3285	0.3314	0.3632	0.3632	0.3562	0.3400	0.3490	0.3575
Erro Relativo Médio:		0.0004	0.0004	0.0004	0.0018	0.0022	0.0017	-0.0051	-0.0049	-0.0052
Variância:		4.0969	4.0734	4.1057	3.8233	3.8116	3.8282	3.8575	3.7856	3.7713

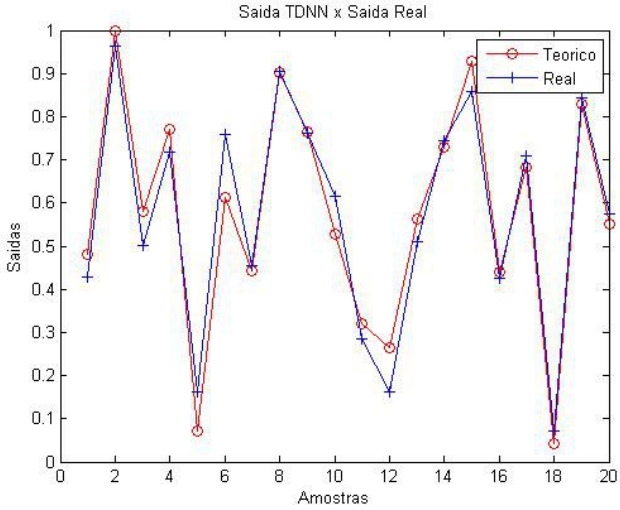
4.Para cada uma das topologias apresentadas na tabela acima, considerando-se ainda o melhor treinamento {T1, T2 ou T3} realizado em cada uma delas, trace então o gráfico dos valores de erro quadrático médio (EQM) em função de cada época de treinamento. Imprima os três gráficos numa mesma folha de modo não superpostos.

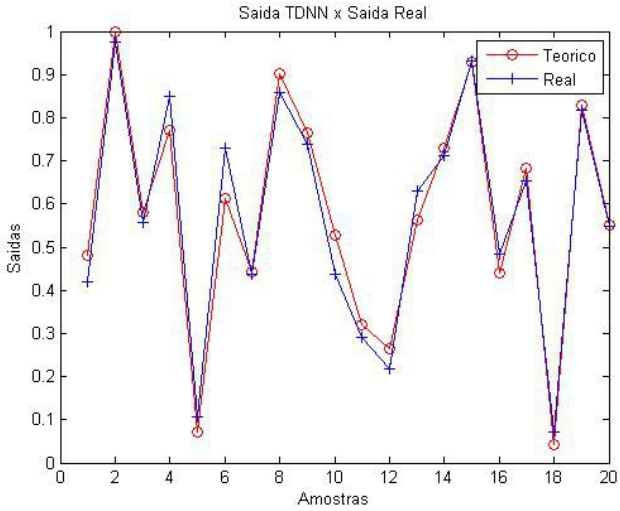

Erro Quadratico Medio - TDNN
EQM
Epoca
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
0
200
400
600
800
1000
1200
1400
1600
1800

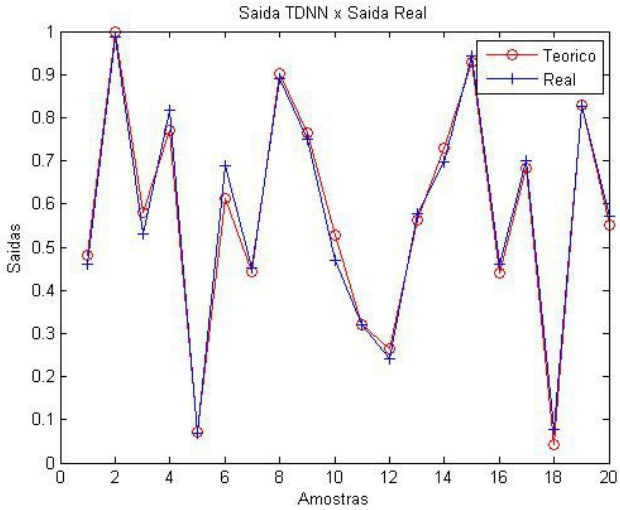

Erro Quadratico Medio - TDNN
EQM
Epoca
0.25
0.2
0.15
0.1
0.05
0
0
100
200
300
400
500
600
700
800


Erro Quadratico Medio - TDNN
EQM
Epoca
0.25
0.2
0.15
0.1
0.05
0
0
100
200
300
400
500
600
700
800
900

5.Para cada uma das topologias apresentadas na tabela acima, considerando-se ainda o melhor treinamento {T1, T2 ou T3} realizado em cada uma delas, trace então o gráfico dos valores desejados e dos valores estimados pela respectiva rede em função do domínio de operação assumido (*t*=101..120). Imprima os três gráficos numa mesma folha de modo não superpostos.







6. Baseado nas análises dos itens acima, indique então qual das topologias candidatas {Rede 1, Rede 2 ou Rede 3}, e que com qual configuração final de treinamento {T1, T2 ou T3}, seria a mais adequada para realização de previsões neste processo.

A topologia 3 é a mais adequada pois foi a que mais se aproximou da saída estimada em todos os treinamentos e obteve as menores variâncias se comparadas as das outras topologias. Em relação à configuração final de treinamento da topologia 3, a mais indicada seria o treinamento 2, pois possui o menor erro relativo médio e variância aproximada se comparada aos outros treinamentos da topologia 3, além do menor número de épocas, ou seja, um treinamento mais rápido.

7. Em relação aos métodos de treinamento que são variantes do algoritmo *backpropagation*, investigue e comente sobre as principais características e vantagens dos seguintes algoritmos:
   a. Algoritmo de treinamento Resilient-Propagation (RProp).
   b. Algoritmo de treinamento Levenberg-Marquardt (LM).

Algoritmo de treinamento Resilient Propagation (Rprop)

A principal diferença entre este algoritmo e as outras heurísticas baseadas em variações do "*backpropagation*" é que os ajuste dos pesos (ω) dos neurônios da rede e da taxa de aprendizado (η) depende apenas dos sinais dos gradientes da função erro $E(\omega)$, não dependendo portanto de sua magnitude. A função $E(\omega)$ é responsável pela especificação de um critério de desempenho que está associado à rede.

No algoritmo "Rprop", os pesos e a taxa de aprendizado são alterados apenas uma única vez em cada época de treinamento. Cada peso ω_{ji} possui sua própria taxa de variação (Δ_{ji}), a qual varia em função do tempo t da seguinte forma:

$$\Delta_{ji}(t) = \begin{cases} \eta^{+}.\Delta_{ji}(t-1), \text{ se } \frac{\partial E}{\partial w_{ji}}(t-1).\frac{\partial E}{\partial w_{ji}} > 0 \\ \eta^{-}.\Delta_{ji}(t-1), \text{ se } \frac{\partial E}{\partial w_{ji}}(t-1).\frac{\partial E}{\partial w_{ji}} < 0 \\ \Delta_{ji}(t-1), \text{ caso contrário} \end{cases} \quad (1)$$

onde $0 < \eta^{-} < 1 < \eta^{+}$. Uma mudança no sinal das derivadas parciais correspondentes ao peso ω_{ji} indica que a última mudança foi grande suficiente para que o sistema saltasse sobre um ponto de mínimo da função $E(\omega)$, o que implica então numa diminuição do valor de Δ_{ji} proporcional ao fator η^{-}. Já as derivadas consecutivas com o mesmo sinal indicam que o sistema está movendo

permanentemente em uma única direção, o que implica assim num aumento sensível de Δ_{ji} proporcional ao fator η^+.

Os pesos da rede são então alterados através das seguintes equações:

$$\Delta w_{ji}(t) = \begin{cases} -\Delta_{ji}(t), \text{ se } \frac{\partial E}{\partial w_{ji}}(t) > 0 \\ +\Delta_{ji}(t), \text{ se } \frac{\partial E}{\partial w_{ji}}(t) < 0 \\ 0 \text{ , caso contrário} \end{cases} \qquad (2)$$

É importante notar que a mudança nos pesos da rede depende apenas do sinal das derivadas parciais, independendo de sua magnitude. Se a derivada for positiva, o peso é decrementado por $\Delta_{ji}(t)$; se a derivada for negativa, o peso será incrementado por $\Delta_{ji}(t)$.

Algoritmo de treinamento Levenberg-Marquardt (LM)

Esse método trabalha com uma variação suave entre os extremos dos métodos de Newton e do Gradiente, sendo que se utiliza este último quando se está longe do mínimo, visto que o método de Newton pode não convergir nesta situação. Por outro lado, quando se está distante da solução usa-se o método do Gradiente. Esse método combina os métodos do Gradiente e de Newton através da alteração dos elementos da diagonal principal da matriz Hessiana.

O método de Levenberg-Marquardt na prática tem características de convergência global (converge para o mínimo local a partir de qualquer valor aproximado), o limite do tamanho do passo é feito através do uso da região de confiança. O método fica caracterizado como:

min || V(xi)+J(xi)(xi+1- xi)||2

sujeito a: ||xi+1-xi||≤δc

sendo δc um escalar que define a região de confiança. A solução do problema acima (DENNIS e SCHNABEL 1983) é:

xi+1=xi-[J(xi)TJ(xi)+μc I]-1J(xi)TV(xi)

onde:μc=0 se δc ≥ ||[J(xi)TJ(xi)]-1J(xi)TV(xi)|| ; e μc>0 caso contrário.

CÓDIGO FONTE UTILIZADO

Carregamento dos Dados

```
function DB_X = Carrega_Tabela_operacao( filename )

[DB_X] = textread( filename, '%f', 'headerlines', 1);

[DB_X] = textread( 'Dados_Treino.dat', '%f', 'headerlines', 1);

clear;
clc;

%Dados do TDNN
OpFile = 'Dados_Operacao_R3.dat';
p     = 15; %entradas da TDNN
eta   = 0.1; %coeficiente de treinamento
alfa  = 0.8 %coeficiente de momentum
epson = 0.5e-06; % margem do erro

n_camadas = 2;
size_Camadas = [25 1];
max_epocas = 10000;

%Carrega os dados
Carrega_Tabela_Treino;

%Pre-Processamento: normalização
maximo = 1%max(DB_X);
minimo = 0%min(DB_X);
DB_X_Norm = Normaliza( minimo, maximo, DB_X );

%Pre-Processamento: janelamento
dados = janelamento( p, DB_X_Norm );

%Vetor de saida
d = dados(:, p+1);

sizeD = size(dados);
linhas = sizeD(1);

x = [ (-1) * ones( linhas, 1 ) dados ]; %adiciona uma coluna de -1 à matriz de dados
x( :, p+2 ) = []; %remove ultima coluna (dados de saida), o p+2 é porque já inserimos o -1 na linha de cima

x = x';
d = d';

%TREINAMENTO
epoca = 1;
while( epoca < 10 )
    tic
    [W_1, W_2, eqm, epoca] = TDNN_Treino( alfa, eta, epson, x, d, max_epocas, n_camadas, size_Camadas  );
    toc
end;

%Grafico do EQM
plot( 1: length(eqm), eqm );
grid;
title( 'Erro Quadratico Medio - TDNN');
xlabel( 'Epoca' );
ylabel( 'EQM' );

disp 'PAROU'
pause

%OPERACAO
%Carrega os dados
DB_X = Carrega_Tabela_Operacao(OpFile);

%Pre-Processamento: normalização
DB_X_Norm = Normaliza( 0, 1, DB_X );

%Pre-Processamento: janelamento
dados = janelamento( p, DB_X_Norm );
```

```
%Vetor de saida
d = dados(:, p+1);

sizeD = size(dados);
linhas = sizeD(1);

x = [ (-1) * ones( linhas, 1 )
dados ]; %adiciona uma coluna de
-1 à matriz de dados
x( :, p+2 ) = []; %remove ultima
coluna (dados de saida), o p+2 é
porque já inserimos o -1 na linha
de cima

x = x';
d = d';

N_Amostras = linhas;

y  = []; %saida real
E = 0;
for k=1: N_Amostras
    y(:, k) = TDNN_Executa( W_1,
W_2, x(:,k) );

    %Erro
    E = E + (d(k) - y(k));
end;

E = E / N_Amostras;

%Variancia
variancia = var(d,y)*100;

disp 'Erro Relativo Medio';
E
disp 'Variancia'
variancia
pause

figure;
x = [1:N_Amostras];
%Grafico da comparação de saidas
% plot( x,d,x,y );
plot(d, 'ro-');
hold on;
plot(y, 'b+-');
% grid;
title( 'Saida TDNN x Saida Real');
xlabel( 'Amostras' );
ylabel( 'Saidas' );
legend('Teorico', 'Real',
'Location', 'NorthEast');
disp 'Saidas Fornecidas pela Rede:'
y'

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 );

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

EM  = sum( E_k ) / N_amostras;


function delta_saida =
GradienteLocalDeSaida( size_camada,
d, Y )
%UNTITLED1 Summary of this function
goes here
%  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) ) );
end;

function delta_saida =
GradienteLocalIntermediario( size_
camada, size_camada_posterior,
delta_posterior, W, Y )
%UNTITLED1 Summary of this
function goes here
% 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)
)

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 *
( Y(j+1) * ( 1 - Y(j+1) ) );
end;

function saida = Janelamento( p,
DB_X )
%Janelamento gera o arquivo de
entradas e saidas para a rede
%   de acordo com o valor de p
(numero de entradas)

saida = [];

linha =1;
for t=p+1:length(DB_X)
    for j=1:p
        saida(linha,j) = DB_X(t-j);
    end;
    saida(linha, p+1) = DB_X(t); %
coluna de saida
    linha = linha + 1;
end;

function NormArray =
Normaliza( max_scale, min_scale,
Vetor_Amostras )
%Normaliza Faz a normalizacao do
vetor de amostras
% max_scale -> valor maximo da
escala (para funcao sgn = 1)
% min_scale -> valor minimo da
escala (para funcao sgn = -1)
% Vetor_Amostras -> vetor com
amostras

max_amostras = max(Vetor_Amostras);
min_amostras = min(Vetor_Amostras);

for k=1:length(Vetor_Amostras)
    NormArray(k) = (max_scale -
min_scale) * ( Vetor_Amostras(k) -
min_amostras ) / (max_amostras -
min_amostras) + min_scale;
end;


function [W_1, W_2, eqm, epoca] =
TDNN_Treino( alfa, eta, epson,
entradas, saidas, max_epocas,
n_camadas, size_Camadas  )
%TDNN_Treino Treinamento de TDNN
```

```
%   alfa          -> taxa de
momentum
%   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

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

W_1 = rand(nCamadaOculta,
N_entradas);   %1 neuron por
linha; a quantidade de colunas
corresponde ao tamanho do vetor de
entrada que já inclui o bias
W_2 = rand(nCamadaSaida,
nCamadaOculta+1); %Matriz de pesos
da camada 2 - 1 neuron = 1 linha;
cada coluna é uma sinapse e tem
que incluir o bias da camada
anterior N_Camada_Oculta + 1

%Matrizes de peso da iteração
anterior (para calculo de momentum)
W_1_Anterior =
W_1;%zeros(nCamadaOculta,
N_entradas);
W_2_Anterior =
W_2;%zeros(nCamadaSaida,
nCamadaOculta+1);

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

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

%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
         Y_1 = [ -1; Y_1 ];

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

        %TESTE - BACKWARD
        %Camada 2
        delta_2 =
GradienteLocalDeSaida( nCamadaSaida
, 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) + alfa * ( W_2(j,i) -
W_2_Anterior(j,i) ) + eta *
delta_2(j) * Y_1(i); %correção com
momentum
            end;
        end;

        %Ja atualizou os pesos,
armazena na matriz para a proxima
iteração,
        %com uso de momentum
        W_2_Anterior = W_2;

        %Camada 1
        delta_1 =
GradienteLocalIntermediario( nCama
daOculta, nCamadaSaida, 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) + alfa * ( W_1(j,i) -
W_1_Anterior(j,i) ) + eta *
delta_1(j) * X(i); %correção com
momentum
            end;
        end;

        %Ja atualizou os pesos,
armazena na matriz para a proxima
iteração,
        %com uso de momentum
        W_1_Anterior = W_1;

        %TESTE - BACKWARD - FIM
    end;

    %Calcula o Erro
    EQM_Atual = EQM( entradas,
saidas, W_1, W_2 );
    EQM_Atual

    eqm(epoca) = EQM_Atual;

    difEQM = abs (EQM_Atual -
EQM_Anterior);

    epoca = epoca + 1;
end;

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 y = TDNN_Executa( W_1,
W_2, X )
%TDNN_Executa Operacao de TDNN
%   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;
```