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

**Alunos:** Luisa Helena Bartocci Liboni
Rodrigo de Toledo Caropreso

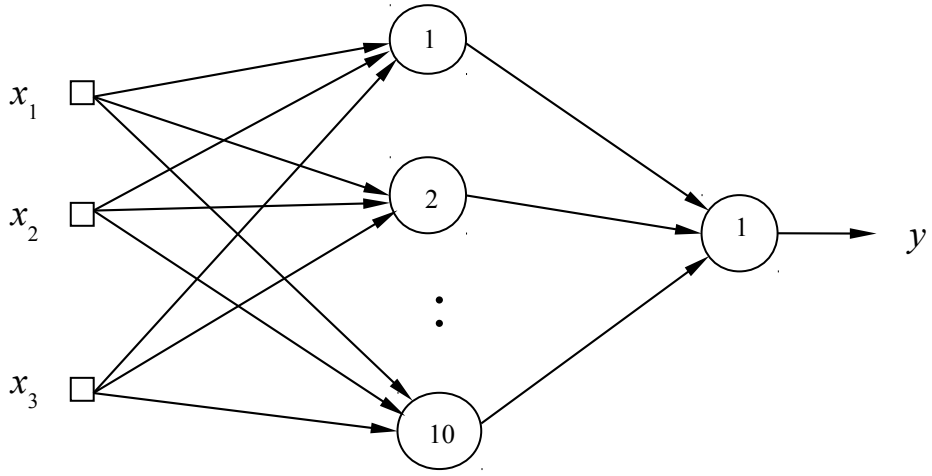
# Redes Neurais Artificiais

**(Prof. Ivan Nunes da Silva)**

## EPC-3

Para a confecção de um sistema de ressonância magnética, observou-se que é de extrema importância para o bom desempenho do processador de imagens de que a variável $\{y\}$, que mede a energia absorvida do sistema, possa ser estimada a partir da medição de três outras grandezas $\{x_1 , x_2 , x_3\}$. Entretanto, em função da complexidade do sistema, sabe-se que este mapeamento é de difícil obtenção por técnicas convencionais, sendo que o modelo matemático disponível para representação do mesmo não fornece resultados satisfatórios.

Assim, a equipe de engenheiros e cientistas pretende utilizar uma rede perceptron multicamadas como um aproximador universal de funções, tendo-se como objetivo final de que, dado como entrada os valores de $\{x_1 , x_2 , x_3\}$, a mesma possa estimar (após o treinamento) o respectivo valor da variável $\{y\}$ que representa a energia absorvida. A topologia da rede perceptron constituída de duas camadas neurais está ilustrada na figura abaixo.



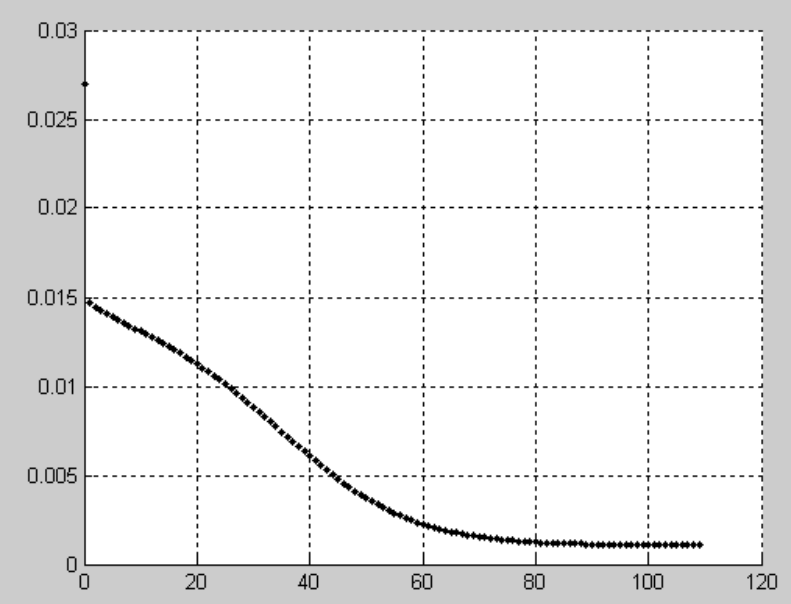

Utilizando o algoritmo de aprendizagem *backpropagation* (Regra Delta Generalizada) e os dados de treinamento apresentados no Anexo, sendo que as variáveis de entrada $\{x_1 , x_2 , x_3\}$ já estão todas normalizadas, realize as seguintes atividades:

1. Execute 5 treinamentos para a rede perceptron, inicializando-se as suas 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 $\eta = 0.1$ e precisão $\varepsilon = 10^{-6}$.

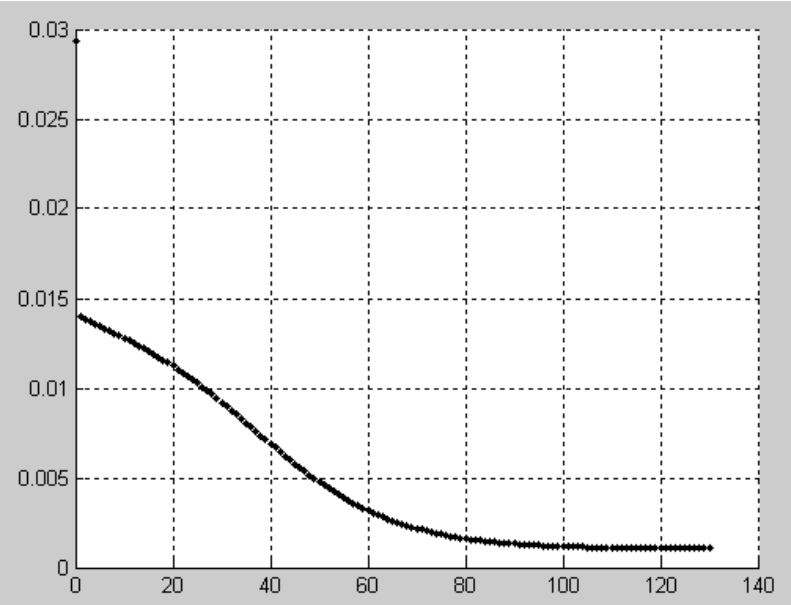
2. Registre os resultados finais desses 5 treinamentos na tabela abaixo:

| Treinamento | Erro Quadrático Médio | Número de Épocas |
|---|---|---|
| 1º (T1) | 0.0010 | 115 |
| 2º (T2) | 0.0011 | 133 |
| 3º (T3) | 0.0012 | 111 |
| 4º (T4) | 0.0017 | 89 |
| 5º (T5) | 0.0011 | 108 |

3. Para os dois treinamentos acima, com maiores números de épocas, trace os respectivos gráficos dos valores de erro quadrático médio (EQM) em função de cada época de treinamento. Imprima os dois gráficos numa mesma folha de modo não superpostos.




$T_1$



$T_2$

4. Baseado na tabela do item 2, explique de forma detalhada por que tanto o erro quadrático médio quanto o número de épocas variam de treinamento para treinamento.

A variação do número de épocas e do erro quadrático médio entre diferentes treinamentos ocorre em razão dos valores aleatórios dos pesos sinápticos definidos nas matrizes de pesos, inicializados no algoritmo de treinamento, uma para cada camada da rede.

Isso coloca a RNA em posições diferentes dentro da superfície de erro da função a ser aproximada, o que modifica a quantidade necessária de épocas para levar a rede até o ponto de mínimo local (ou global).

Para a variação do erro quadrático médio um valor diretamente ligado é a da precisão, dada no EPC por $\varepsilon = 10^{-6}$. Se essa precisão fosse muito menor, como por exemplo, $\varepsilon = 10^{-20}$, com certeza teríamos para todos os treinamentos valores de erro quadrático médio final muito mais próximas, já que a precisão no treinamento tem que ser menor que uma diferença entre dois EQM´s sucessivos como critério de parada do *loop* de treinamento.

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

| **Amostra** | $x_1$ | $x_2$ | $x_3$ | $d$ | $y_{rede}$ **(T1)** | $y_{rede}$ **(T2)** | $y_{rede}$ **(T3)** | $y_{rede}$ **(T4)** | $y_{rede}$ **(T5)** |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 0.0611 | 0.2860 | 0.7464 | 0.4831 | 0.4874 | 0.4763 | 0.4891 | 0.5395 | 0.4866 |
| **2** | 0.5102 | 0.7464 | 0.0860 | 0.5965 | 0.5996 | 0.5708 | 0.6063 | 0.6203 | 0.5918 |
| **3** | 0.0004 | 0.6916 | 0.5006 | 0.5318 | 0.5306 | 0.5131 | 0.5359 | 0.5723 | 0.5253 |
| **4** | 0.9430 | 0.4476 | 0.2648 | 0.6843 | 0.7084 | 0.6925 | 0.7144 | 0.7225 | 0.7096 |
| **5** | 0.1399 | 0.1610 | 0.2477 | 0.2872 | 0.2786 | 0.3524 | 0.2744 | 0.2458 | 0.2883 |
| **6** | 0.6423 | 0.3229 | 0.8567 | 0.7663 | 0.7513 | 0.7514 | 0.7574 | 0.7501 | 0.7535 |
| **7** | 0.6492 | 0.0007 | 0.6422 | 0.5666 | 0.5772 | 0.5452 | 0.5830 | 0.6187 | 0.5766 |
| **8** | 0.1818 | 0.5078 | 0.9046 | 0.6601 | 0.6850 | 0.6685 | 0.6892 | 0.7018 | 0.6824 |
| **9** | 0.7382 | 0.2647 | 0.1916 | 0.5427 | 0.5421 | 0.5108 | 0.5467 | 0.5602 | 0.5383 |
| **10** | 0.3879 | 0.1307 | 0.8656 | 0.5836 | 0.6098 | 0.5823 | 0.6159 | 0.6539 | 0.6083 |
| **11** | 0.1903 | 0.6523 | 0.7820 | 0.6950 | 0.6959 | 0.6814 | 0.6991 | 0.7068 | 0.6926 |
| **12** | 0.8401 | 0.4490 | 0.2719 | 0.6790 | 0.6792 | 0.6554 | 0.6854 | 0.6980 | 0.6778 |
| **13** | 0.0029 | 0.3264 | 0.2476 | 0.2956 | 0.2874 | 0.3317 | 0.2822 | 0.2590 | 0.2945 |
| **14** | 0.7088 | 0.9342 | 0.2763 | 0.7742 | 0.7784 | 0.7890 | 0.7738 | 0.7641 | 0.7794 |
| **15** | 0.1283 | 0.1882 | 0.7253 | 0.4662 | 0.4661 | 0.4581 | 0.4668 | 0.5145 | 0.4667 |
| **16** | 0.8882 | 0.3077 | 0.8931 | 0.8093 | 0.8067 | 0.8341 | 0.8121 | 0.7851 | 0.8141 |
| **17** | 0.2225 | 0.9182 | 0.7820 | 0.7581 | 0.7792 | 0.7901 | 0.7741 | 0.7504 | 0.7782 |
| **18** | 0.1957 | 0.8423 | 0.3085 | 0.5826 | 0.5969 | 0.5726 | 0.6038 | 0.6262 | 0.5894 |
| **19** | 0.9991 | 0.5914 | 0.3933 | 0.7938 | 0.7891 | 0.8067 | 0.7899 | 0.7786 | 0.7957 |
| **20** | 0.2299 | 0.1524 | 0.7353 | 0.5012 | 0.4997 | 0.4832 | 0.5017 | 0.5497 | 0.4993 |
| | | | **Erro Relativo Médio (%)** | | -0.4565 | 0.0930 | -0.7198 | -1.8019 | -0.4553 |
| | | | **Variância (%)** | | 1.7073 | 1.8123 | 1.6943 | 1.6371 | 1.7262 |

6. Baseado nas análises da tabela acima, indique qual das configurações finais de treinamento {T1 , T2 , T3 , T4 ou T5} seria a mais adequada para o sistema de ressonância magnética, ou seja, qual delas está oferecendo a melhor generalização.

O treinamento T2 apresentou o menor erro relativo médio dentre todos os treinamentos efetuados, sendo portanto mais recomendado para utilização prática.

# 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(size_Camadas(1),
N_entradas); %Matriz de pesos da
camada 1 (entrada -> Neurons Ocultos)
W_2 = rand(size_Camadas(2),
size_Camadas(1) + 1); %Matriz de pesos
da camada 2 (Neurons Ocultos -> Camada
de Saida)

% 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)');
W_1
pause
W_2
pause

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

while( epoca <= max_epocas && ~stop )
    erro_parcial = 0;

    for k=1:N_amostras

        %FeedForward
        %1a Camada: Entrada -> Neurons
da camada oculta
        camada_L = 1;
        I_1 = Get_IJL( size_Camadas,
camada_L, entradas( :, k ), W_1 );
        Y_1 = Get_YJL( size_Camadas,
camada_L, I_1 );

        %Ajeita o vetor de saida,
adicionando o bias
        Y_1 = [ -1 Y_1 ]';

        %2a Camada: Neurons Camada
Oculta -> Neurons de Saida
        camada_L = 2;
        I_2 = Get_IJL( size_Camadas,
camada_L, Y_1, W_2 );
        Y_2 = Get_YJL( size_Camadas,
camada_L, I_2 );

        %Ajeita o vetor de saida, aqui
nao tem bias
        Y_2 = Y_2';

        %Backpropagation
        %Calcula o gradiente local da
camada de saida
        camada_L = 2;
        delta_2 =
GradienteLocalDeSaida( size_Camadas,
camada_L, saidas(k), 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;

        %Calcula o gradiente local da
camada de entrada
        camada_L = 1;
        delta_1 =
GradienteLocalIntermediario( size_Cama
das, camada_L, 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) * entradas(i, k);
            end;
        end;

        %Repete o FeedForward para
calcular o ERRO
        %FeedForward
        %1a Camada: Entrada -> Neurons
da camada oculta
        camada_L = 1;
        I_1 = Get_IJL( size_Camadas,
camada_L, entradas( :, k ), W_1 );
        Y_1 = Get_YJL( size_Camadas,
camada_L, I_1 );

        %Ajeita o vetor de saida,
adicionando o bias
        Y_1 = [ -1 Y_1 ]';

        %2a Camada: Neurons Camada
Oculta -> Neurons de Saida
        camada_L = 2;
        I_2 = Get_IJL( size_Camadas,
camada_L, Y_1, W_2 );
        Y_2 = Get_YJL( size_Camadas,
camada_L, I_2 );

        Y_2 = Y_2';

        % Calcula o erro para as
amostras atuais
        erro_parcial(k) = 0.5 * sum
( (saidas(k) - Y_2) .^2 );
    end;

    %Calcula erro Quadratico Medio
    eqm( epoca ) =
sum( erro_parcial ) / N_amostras;

    if( epoca > 1 ) %precisamos de 2
epocas para poder comparar
        if( abs(eqm(epoca) -
eqm(epoca-1)) < epson )
            stop = 1;
        end;
    end;
    epoca = epoca + 1;
end;

epoca = epoca - 1;

if( stop == 1 )
    disp( sprintf( 'Rede treinada.
Numero de epocas: %d', epoca) );
else
    disp( sprintf( 'Limite de epocas
atingido (%d), rede nao treinada.',
epoca) );
end;


function delta_saida =
GradienteLocalIntermediario( size_Cama
das, camada_L, delta_local, pesos,
saidas_MLP )
%UNTITLED1 Summary of this function
goes here
%  saidas_MLP        -> vetor com a
saida de cada neuronio na camada dada
%  por 'camada_L'
% delta_local        -> gradiente
local da camada posterior (ou seja,
% camada_L + 1)
% pesos              -> 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 )

alfa = 1;

for j=1:size_Camadas(camada_L)
    delta_saida(j) = 0;
    for k=1:size_Camadas(camada_L + 1)
        delta_saida(j) =
delta_saida(j) + delta_local(k) *
pesos(k, j);
    end;
    delta_saida(j) = (-1) *
delta_saida(j) * ( alfa *
saidas_MLP(j) * ( 1 -
saidas_MLP(j) ) ); end;
```