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

Alunos: Luisa Helena Bartocci Liboni
Rodrigo de Toledo Caropreso

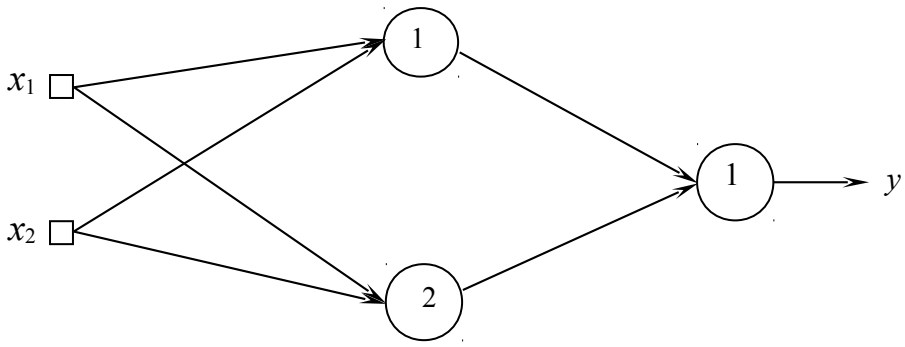
Data de Entrega: 21/05/2012

Redes Neurais Artificiais

(Prof. Ivan Nunes da Silva)

EPC-7

A verificação da presença de radiação em determinados compostos nucleares pode ser feita por intermédio da análise da concentração de duas variáveis definidas por x_1 e x_2. A partir de 50 situações conhecidas, resolveu-se então treinar uma ***RBF*** para a execução da tarefa de classificação de padrões neste processo, cuja topologia está ilustrada na figura seguinte.



A padronização para a saída, a qual representa a presença ou ausência de sinais de radiação, ficou definida da seguinte forma:

Status de Radiação	Saída (y)
Presença	1
Ausência	-1

Utilizando os dados de treinamento apresentados no Apêndice, execute o treinamento de uma ***RBF*** (2 entradas e 1 saída) que possa classificar, em função apenas dos valores medidos de x_1 e x_2, se determinado composto possui radiação. Para tanto, faça as seguintes atividades:

1. Execute o treinamento da camada escondida por meio do algoritmo de clusterização "*k-means*" (vizinhos mais próximos). Em se tratando de um problema de classificação de padrões, compute os centros dos dois clusters levando-se em consideração apenas aqueles padrões com presença de radiação. Após o treinamento, forneça os valores das coordenadas do centro de cada cluster e sua respectiva variância.

Cluster	Centro	Variância
1	0.5030, 0.7405	0.3306
2	0.4784, 0.1923	0.2922

2. Após o treinamento da camada intermediária execute o treinamento da camada de saída usando a regra delta generalizada. Utilize uma taxa de aprendizado $\eta = 0.01$ e precisão de $\varepsilon = 10^{-7}$. No final da convergência forneça os valores dos pesos referentes ao neurônio da camada de saída.

Peso	Valor
$W_{1,0}^{(2)}$	0.3201
$W_{1,1}^{(2)}$	-0.8340
$W_{1,2}^{(2)}$	1.7185

3. 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 a função sinal, ou seja:

$$y^{pos} = \begin{cases} 1, se\ y \geq 0 \\ -1, se\ y < 0 \end{cases}$$,função utilizada apenas no pós-processamento do conjunto de teste.

4. Faça a validação da rede 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.

Amostra	x_1	x_2	d	y	$y^{pós}$
1	0.8705	0.9329	-1	-0.6716	-1
2	0.0388	0.2703	1	0.1017	1
3	0.8236	0.4458	-1	-0.0836	-1
4	0.7075	0.1502	1	0.7906	1
5	0.9587	0.8663	-1	-0.5891	-1
6	0.6115	0.9365	-1	-0.9226	-1
7	0.3534	0.3646	1	0.6034	1
8	0.3268	0.2766	1	0.8504	1
9	0.6129	0.4518	-1	0.1827	1
10	0.9948	0.4962	-1	-0.3202	-1
Taxa de Acerto (%):90%					

5. Se for o caso, explique quais estratégias se pode adotar para tentar aumentar a taxa de acerto desta ***RBF***.

Uma possibilidade é aumentar a quantidade de neurons na camada intermediária a fim de melhorar a resolução do mapeamento dos clusters.

No exercício realizado, com 2 funções, temos o seguinte mapeamento de clusters, produzido pelo algoritmo K-Means:

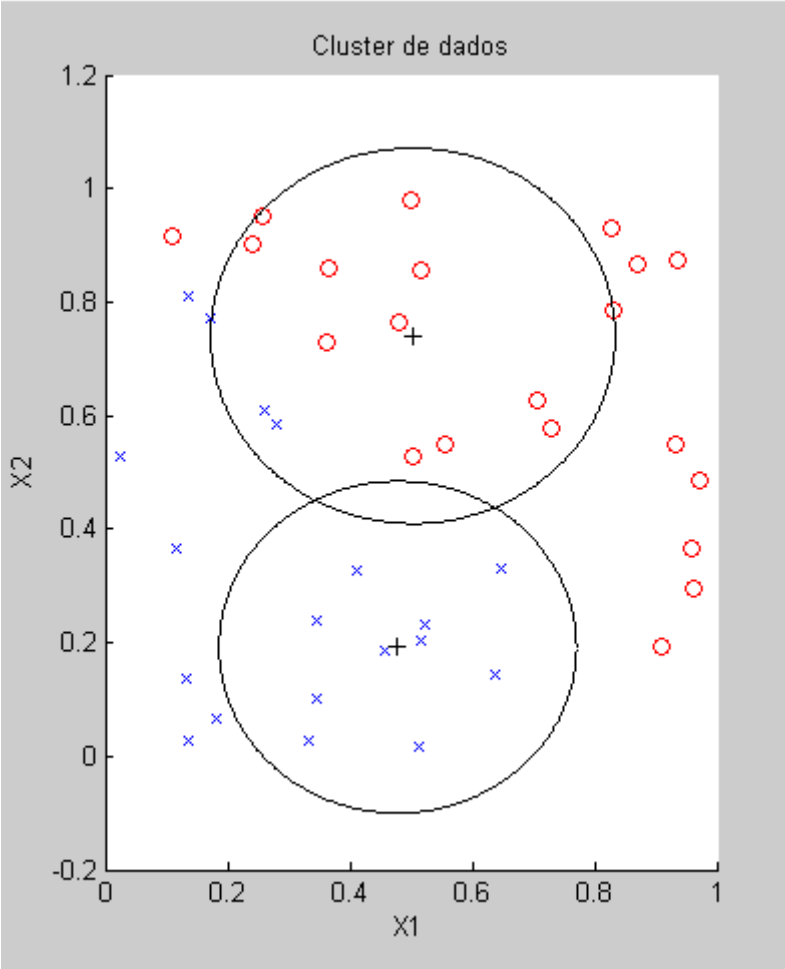


Figura1

Existem pontos fora do mapeamento, pontos na fronteira das regiões de receptividade, entre outros.

Se aumentarmos a camada intermediária, adicionando um neuron, o novo mapeamento passa a ser:

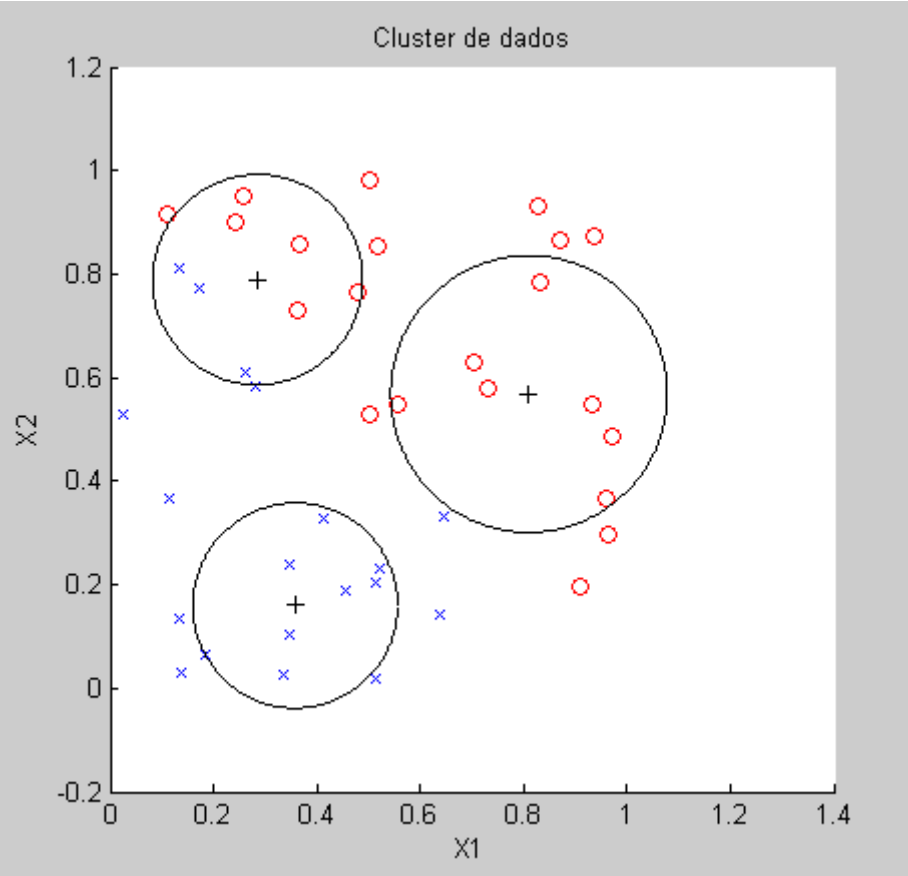


Figura2

Cluster	Centro	Variância
1	0.8101, 0.5686	0.2674
2	0.2851 0.7893	0.2031
3	0.3581, 0.1598	0.1981

A rede treinada obteve 100% de taxa de acerto, resultado melhor do que os 90% produzidos pela rede inicialmente usada no exercício.

Apesar disso, diversos pontos ficaram fora do mapeamento e existem regiões com pontos misturados das duas classes.

Uma resolução ainda melhor pode ser obtida com o acréscimo de mais neurons (tomando o cuidado de não evitar um overfitting), conforme ilustra a figura abaixo (8 neurons):

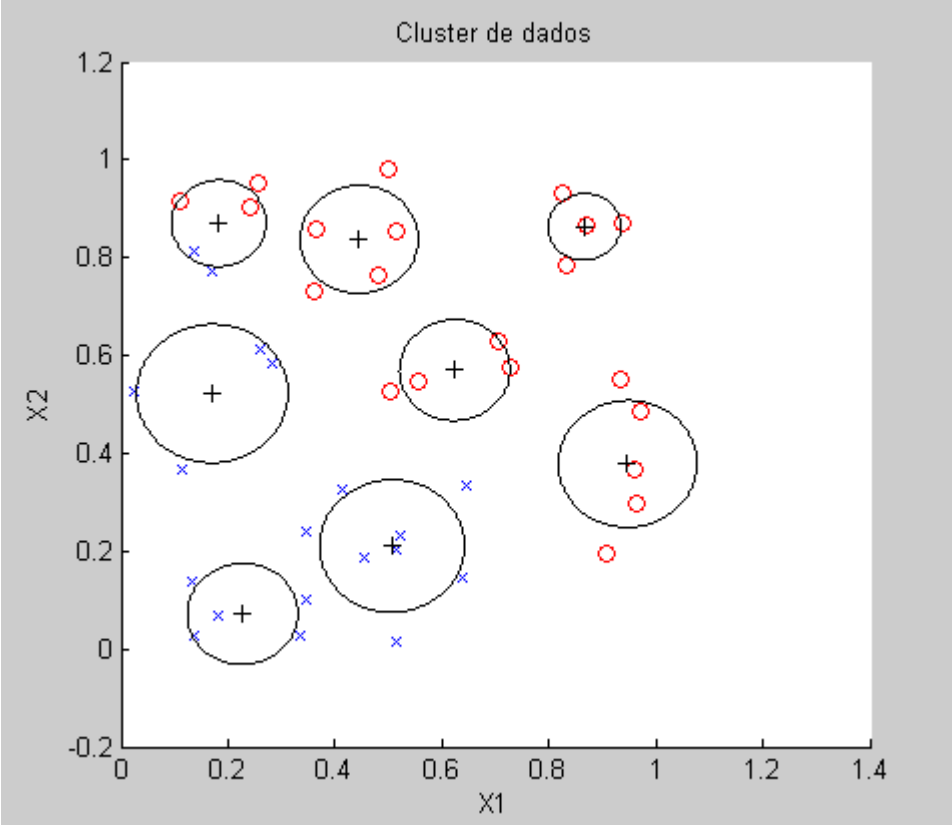


Figura3

Cluster	Centro	Variância
1	0.4452, 0.8375	0.1103
2	0.1826, 0.8704	0.0885
3	0.2269, 0.0718	0.1034
4	0.5065, 0.2105	0.1355
5	0.6236, 0.5703	0.1033
6	0.1705, 0.5223	0.1417
7	0.8661, 0.8634	0.0679
8	0.9470, 0.3786	0.1297

O treinamento desta rede não foi excessivamente mais oneroso do que as anteriores, produziu 100% de classificação também. No entanto, o melhor mapeamento das classes (que pode ser visto comparando as figuras 2 e 3) torna esta última topologia mais adequada e robusta para classificar futuras amostras

CÓDIGO FONTE UTILIZADO

CARREGAMENTO DE DADOS

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

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

EXECUÇÃO DO EPC

```
clear;
clc;

%Carrega os dados
Carrega_Tabela_Treino;

%Monta vetores de amostras
eta = 0.01; %coeficiente de
treinamento
epson = 1e-07; % margem do erro

N_Entradas = 2; %entradas do RBF
n_camadas = 2;
size_Camadas = [2 1];
N_Saidas = size_Camadas(2);
N_Camada_Oculta = size_Camadas(1);
N_Amostras = length(DB_X1);

%PRIMEIRO ESTAGIO DE TREINAMENTO -
INICIO
X = [DB_X1 DB_X2];
[W_1 sigma] =
RBF_Estagio1_Treino(X, N_Entradas,
N_Camada_Oculta);

disp 'Centroides'
W_1

disp 'Variancias'
sigma'

pause
%PRIMEIRO ESTAGIO DE TREINAMENTO -
FIM

%SAIDAS DA CAMADA NEURAL
INTERMEDIARIA - INICIO
[ Y_1 ] = RBF_Estagio1_Operacao( X,
W_1, sigma );
%SAIDAS DA CAMADA NEURAL
INTERMEDIARIA - FIM

%SEGUNDO ESTAGIO DE TREINAMENTO -
INICIO
%monta matriz de entradas
x = [];
x = [ (-1)*ones(N_Amostras, 1)
Y_1 ]';

d =  [DB_D]';
max_epocas = 2000;

%Treinamento
tic
[W_2, eqm, epoca ] =
RBF_Estagio2_Treino( eta, epson, x,
d, max_epocas, 1, N_Saidas );
t = toc
%SEGUNDO ESTAGIO DE TREINAMENTO -
FIM

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

pause

%OPERACAO - BEGIN
%Carrega os dados
Carrega_Tabela_Operacao;

N_Amostras = length(DB_X1);

%Estagio 1
X = [DB_X1 DB_X2];
[ Y_1 ] = RBF_Estagio1_Operacao( X,
W_1, sigma );

%Estagio 2
%monta matriz de entradas
x = [];
x = [ (-1)*ones(N_Amostras, 1)
Y_1 ];

d =  [DB_D]';

y  = []; %saida real
yp = []; %saida pós processada

%Executa Rede
total_acertos = 0;
for k=1: N_Amostras

    y(:, k) =
RBF_Estagio2_Operacao( W_2,
x(k, :)' );
```

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

total_acertos
N_Amostras
pause

disp 'Saidas Reais do RBF';
y'

%Pos processamento
disp 'Saidas Pós-Processadas do
RBF';
yp'

disp 'Saidas Desejadas';
d'

%Verifica taxa de acerto
disp 'Taxa de acerto';
(total_acertos / N_Amostras) * 100
```

CÁLCULO DO EQM

```
function EM = EQM( X, d, 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_2 = W_2 * X;
Y_2 = 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;
```

ROTINAS DO RBF

```
function y = RBF_Estagio2_Operacao(
W_2, X )
%RBF_Estagio2_Operacao
%   X     -> matriz com entradas
%   y     -> vetor com saidas
produzidas pela rede
%   W_2   -> matriz de pesos da
rede treinada

I_2 = W_2 * X;
y = I_2; %usando função linear na
saida


function  [W_2,  eqm,  epoca  ]  =
RBF_Estagio2_Treino(  eta,  epson,
entradas,    saidas,    max_epocas,
n_camadas, size_Camadas )
%RBF_Estagio2 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

N_Entradas = size(entradas, 1);
N_Amostras = size(entradas, 2);

W_2 = rand(size_Camadas,
N_Entradas); %Matriz de pesos da
camada 2 - 1 neuron = 1 linha; cada
coluna é uma sinapse e tem que
incluir o bias da camada anterior
(3 colunas)

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

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

%TREINAMENTO MLP 1 camada - BEGIN
difEQM = 1;
EQM_Atual = EQM( entradas, saidas,
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
        %FORWARD -INICIO
        X = entradas( :, k );
        d = saidas  ( :, k );

        I_2 = W_2 * X;
        Y_2 = I_2; %usando função
linear na saida
        %FORWARD - FIM

        %BACKWARD - INICIO
        %Camada 2
        delta_2 =
GradienteLocalDeSaida( 1, 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) * X(i);
            end;
        end;
        %BACKWARD - FIM
    end;

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

    eqm(epoca) = EQM_Atual;

    difEQM = abs (EQM_Atual -
EQM_Anterior);

    epoca = epoca + 1;
end;

%TREINAMENTO MLP 1 camada - 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       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) ) );
    delta_saida(j) = ( d(j) -
Y(j) ); %usando função linear
(aproximador de funções)
end;


function     [     Y_1     ]     =
RBF_Estagio1_Operacao(    X,    W_1,
sigma )
%RBF_Estagio1_Operacao Executa o
primeiro estagio da RBF
%N_Amostras      -> quantidade de
amostras de entradas
%N_Camada_OCulta -> quantidade de
funcoes da camada intermediaria
%X               -> vetor de
entradas
%W_1             -> matriz de pesos
(sinapses - cada linha = 1 neuron,
cada
%coluna = 1 sinapse), basicamente
as coordenadas do centroide de cada
%funcao radial
%sigma           -> vetor de
variancias (cada linha = 1 neuron)

N_Amostras = size(X, 1); %qdte de
linhas  = amostras
N_Entradas = size(X, 2); %qdte de
colunas = entradas da rede
N_Camada_Oculta = size(W_1, 1);
%qdte de linhas  = neurons

for k=1:N_Amostras
    for j=1:N_Camada_Oculta
        soma = 0;
        for i=1:N_Entradas
            soma = soma + ( X( k, i
) - W_1( j, i ) )^2;
            Y_1(k, j) = exp(-soma/
(2*sigma(j)^2));
        end;
    end;
end;
```

```
function     [W_1,     var]     =
RBF_Estagio1_Treino(X,  N_Entradas,
N_Camada_Oculta)
%RBF_Estagio1 Executa o Primeiro
Estagio da RBF
%Calcula os centroides (pesos) e
variancias

%Vetor de inicialização do k-means
(alocaçao do "chute inicial de cada
%centroide")
start = [];
for i=1:N_Camada_Oculta
    start = [start; X(i,:)];
end;

%Etapa1: Clusterização por K-Means
[idx, c] = kmeans( X,
N_Camada_Oculta, 'Start', start );
% [idx, c] = kmeans( X,
N_Camada_Oculta );

for j=1:size(c,1) %'j' -> cada
linha corresponde a um centroide
    soma = 0;
    for k=1:length(idx)
        if( idx(k) == j ) %verifica
se aquela amostra pertence ao
cluster 'j'
            for i=1:N_Entradas
                soma = soma + (X(k,
i) - c(j, i))^2;
            end;
        end;
    end;
    sigma(j) = soma /
length(find(idx==j)); %verifica
quantos elementos em idx pertencem
ao cluster 'j'
end;

W_1 = c;
var = sigma;
```