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: 04/06/2012

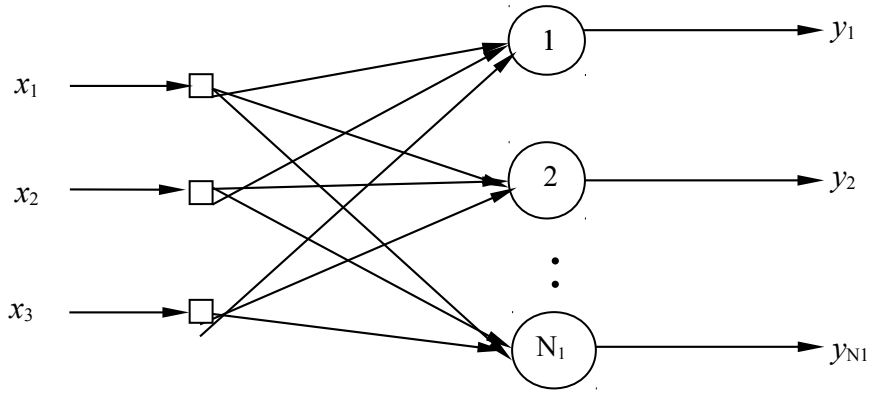
Redes Neurais Artificiais

(Prof. Ivan Nunes da Silva)

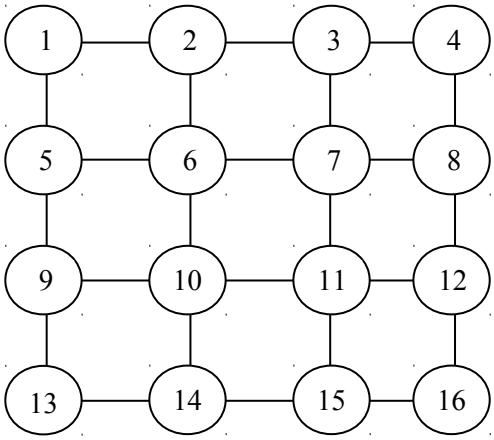
EPC-9

No processo industrial de fabricação de pneus sabe-se que o composto que forma a borracha pode apresentar imperfeições que impedem a sua utilização. Diversas amostras dessas imperfeições foram coletadas, sendo também realizadas as medidas referentes a três grandezas $\{x_1, x_2, x_3\}$ que participam do processo de fabricação das respectivas borrachas. Entretanto, a equipe de engenheiros e cientistas não tem sentimento de como essas variáveis podem estar relacionadas.

Assim, pretende-se aplicar uma Rede de Kohonen, conforme mostrado na figura abaixo, com o objetivo de detectar as eventuais similaridades e correlações entre essas variáveis, pois se tem como objetivo final o posterior agrupamento das amostras imperfeitas em classes.



Portanto, baseado nos dados fornecidos no apêndice, treine a rede de Kohonen, considerando N_1=16 e taxa de aprendizado η=0.001, sendo que o grid topológico é bidimensional (4x4), tendo raio de vizinhança entre os neurônios igual a 1. Logo, o diagrama esquemático do grid está como se segue:



De posse dos resultados advindos do treinamento da rede, efetuou-se uma análise neste conjunto e verificou-se que as amostras 1-20, 21-60 e 61-120 possuem particularidades em comum, podendo ser então consideradas três classes distintas, denominadas de classe A, B e C, respectivamente. Portanto, têm-se as seguintes questões:

1. Indique quem são os conjuntos de neurônios representados no grid que fornecem respostas relativas às classes A, B e C.

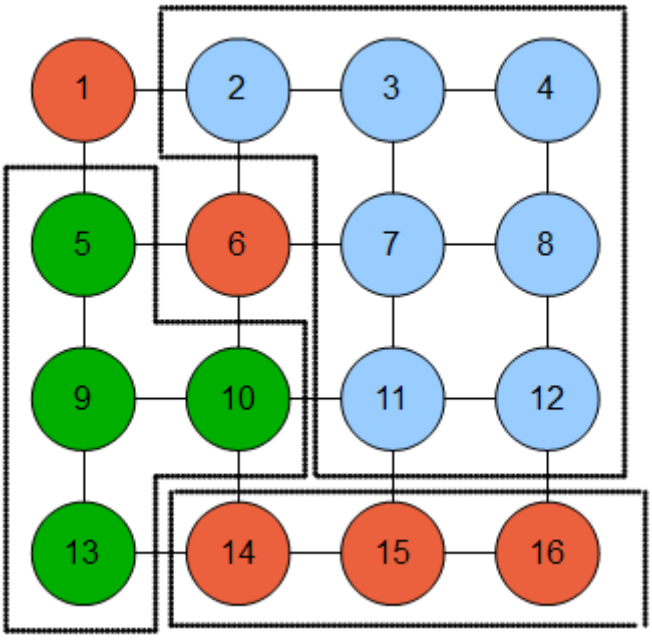
A partir do treinamento, pode-se dividir as classes da seguinte forma:

A = {1, 6, 14, 15, 16}

B = {5, 9, 10, 13}

C = {2, 3, 4, 7, 8, 11, 12}

Pode-se definir o seguinte mapa de contexto:



	Quantidade de amostras agrupadas por classe		
Neuronio Ativado	**CLASSE A**	**CLASSE B**	**CLASSE C**
1	**1**	0	0
2	1	0	**8**
3	0	0	**11**
4	3	0	**8**
5	0	**11**	0
6	**2**	0	0
7	1	0	**10**
8	1	0	**7**
9	0	**14**	0
10	0	**2**	0
11	2	0	**5**
12	0	0	**10**
13	0	**13**	0
14	**3**	0	0
15	**3**	0	0
16	**3**	0	1

Observa-se que alguns neuronios foram selecionados com poucas amostras (caso dos neurons 1, 6 e 10). No mapa bidimensional, estes neurons estão suficientemente próximos de forma que podemos inferir que o conjunto de amostras não contém elementos significativos naquela região do mapa de contexto.

Porém, o neuron 10 foi "mapeado" na classe B, juntamente com seus vizinhos {5, 9, 13} de forma que é mais confiável incluir o neuron 10 na classe B do que os neurons {1,6} na classe C, pois ficaram muito distantes dos demais elementos da classe C {14, 15, 16}.

Desta forma, seria interessante obter mais amostras para treinamento, para melhorar a precisão da rede (se o neuronio 10, por exemplo, mapear a classe C, forma-se um mapa bidimensional bem estruturado).

2. Para as amostras da tabela abaixo indique a que classes as mesmas pertencem.

Amostra	x_1	x_2	x_3	*Classe*
1	0.2471	0.1778	0.2905	A
2	0.8240	0.2223	0.7041	B
3	0.4960	0.7231	0.5866	C
4	0.2923	0.2041	0.2234	A
5	0.8118	0.2668	0.7484	B
6	0.4837	0.8200	0.4792	C
7	0.3248	0.2629	0.2375	A
8	0.7209	0.2116	0.7821	B
9	0.5259	0.6522	0.5957	C
10	0.2075	0.1669	0.1745	A
11	0.7830	0.3171	0.7888	B
12	0.5393	0.7510	0.5682	C

3. Demonstrar que a regra de alteração de pesos "Norma Euclidiana" para um padrão $\boldsymbol{x}$ é obtida a partir da minimização da função erro quadrático:

$$E = \frac{1}{2}\sum_i \left(w_i^{(j)} - x_i\right)^2$$

onde j é o índice do neurônio vencedor.

A minimização é feita com a utilização do vetor gradiente

$$\nabla E(w_{ji}) = \frac{\partial E(W_{ji})}{\partial W_{ji}}$$

$$\nabla E(w_{ji}) = \frac{2}{2}\sum (w_{ji} - x_i)$$

A adaptação dos pesos é feita no sentido oposto ao gradiente:

$$\Delta W_{ji} = -\eta \cdot \nabla E(W_{ji})$$
$$\Delta W_{ji} = \eta \sum (x_i - wji)$$
$$w_j(t+1) = w_j + \eta \sum (x_i - w_{ji}(t))$$
$$w_j = w_j + \eta(x_i - w_j)$$

onde i = 1, 2 ....., número de amostras

CÓDIGO-FONTE

```
clc;
clear;

Carrega_Tabela_Treino;

eta = 0.001;
x0 = [DB_X1 DB_X2 DB_X3]';
N1 = 16; %neuronios
N  = 3; %entradas
epson = 1e-4;
max_epocas = 500;
N_Amostras = length(DB_X1);

%Mapa dos vizinhos
omega = {};
omega{1}  =[2 5];
omega{2}  =[1 3 6];
omega{3}  =[2 4 7];
omega{4}  =[3 8];
omega{5}  =[1 6 9];
omega{6}  =[2 5 7 10];
omega{7}  =[3 6 8 11];
omega{8}  =[4 7 12];
omega{9}  =[5 10 13];
omega{10} =[6 9 11 14];
omega{11} =[7 10 12 15];
omega{12} =[8 11 16];
omega{13} =[9 14];
omega{14} =[10 13 15];
omega{15} =[11 14 16];
omega{16} =[12 15];

%Normaliza o vetor de entrada
for k=1:N_Amostras
       x( :, k ) = x0( :, k ) /
norm(x0( :, k ));
end;

%Inicializacao
for i=1:N1
    w( :, i ) = x( :, i );
end;

%Treinamento
epoca = 0;
stop = 0;

w_anterior = w;
maior_mudanca_anterior = 0;

while(~stop)
    for k=1:N_Amostras
        for j = 1: N1
                            D(j,  k)  =
sqrt( sum( ( x( :, k ) - w( :,
j ) ).^2 ) );
        end;

           [value, winner] = min(D(:,
k)); %menor distancia, determina o
vencedor

        %Atualiza pesos do vencedor
            w( :, winner ) = w( :,
winner ) + eta * ( x( :, k ) -  w( :,
winner ) );

        %Normaliza
         w( :, winner ) = w( :, winner
) / norm( w( :, winner ) );

        %Atualiza vizinhos do vencedor
        vizinhos = omega{winner};
        for j=1: length(vizinhos)
            index = vizinhos(j);
               w( :, index ) = w( :,
index ) + (eta/2) * ( x( :, k ) -
w( :, index ) );

            %Normaliza
                w( :, index ) = w( :,
index ) / norm( w( :, index ) );
        end;
    end;
    epoca = epoca + 1;

    %Criterio de parada
%              maior_mudanca_atual   =
max(max(abs(w-w_anterior)));

    w_change = w-w_anterior;
    for hh=1:size(w_change,2)
        norma(hh) = norm( w_change( :,
hh ) ); %calcula a norma de cada
coluna
    end;

                maior_mudanca_atual   =
max(  norma  );     %norm(max(abs((w-
w_anterior)), [],2));
     maior_mudanca_atual
     if( maior_mudanca_atual < epson )
%||     (     maior_mudanca_atual     -
maior_mudanca_anterior ) < epson )
         stop = 1;
     end;
%
%            maior_mudanca_anterior  =
maior_mudanca_atual;

    w_anterior = w;

end;
```

```
disp 'Fim do Mapeamento: Numero de Epocas'
epoca
pause

%Identifica as classes
A = [];
B = [];
C = [];
for k=1:N_Amostras
    for j = 1: N1
        D(j, k) = sqrt( sum( ( x( :, k ) - w( :, j ) ).^2 ) );
    end;

      [value, winner] = min(D(:, k));
%menor distancia, determina o vencedor

    %Define a classe
    if(k <= 20)
%         if( isempty(find(A==winner)) )
            A = [A winner];
%         end;
    end;

    if(k >= 21 && k <= 60 )
%         if( isempty(find(B==winner)) )
            B = [B winner];
%         end;
    end;

    if(k >= 61 && k <= 120 )
%         if( isempty(find(C==winner)) )
            C = [C winner];
%         end;
    end;

    %     disp(sprintf( 'Amostra: %3d - Neuron:%d', k, winner ));
end;

for i=1:16
     disp(sprintf('A - %d = %d', i, length(find(A==i))));
     disp(sprintf('B - %d = %d', i, length(find(B==i))));
     disp(sprintf('C - %d = %d\n', i, length(find(C==i))));
end;

disp 'Classes'
A
B
C
pause

%OPERACAO

%Define as classes
A = [1 6 14 15 16];
B = [5 9 10 13];
C = [2 3 4 7 8 11 12];

Carrega_Tabela_Operacao;
xOp = [DB_X1 DB_X2 DB_X3]';

N_Amostras = length(DB_X1);

%Normaliza o vetor de entrada
for k=1:N_Amostras
      x( :, k ) = xOp( :, k ) / norm(xOp( :, k ));
end;

%Verificação
for k=1:N_Amostras
    for j = 1: N1
        D(j, k) = sqrt( sum( ( x( :, k ) - w( :, j ) ).^2 ) );
    end;

      [value, winner] = min(D(:, k));
%menor distancia, determina o vencedor

    disp( sprintf( 'Neuronio vencedor: %d', winner ) );

    %Decide a classe
    if( ~isempty(find(A==winner)) )
         disp(sprintf( 'Amostra: %2d - Classe: A', k));
    end;
    if( ~isempty(find(B==winner)) )
         disp(sprintf( 'Amostra: %2d - Classe: B', k));
    end;
    if( ~isempty(find(C==winner)) )
         disp(sprintf( 'Amostra: %2d - Classe: C', k));
    end;
end;
```