Universidade Federal do Rio Grande do Norte
Programa de Pós-Graduação em Engenharia Elétrica e de Computação
Redes Neurais (EEC1505)
Prof. Adrião Duarte Doria Neto
Alunos: José Lenival Gomes de França, Raphael Diego Comesanha e Silva, Danilo de Santana Pena.

Lista 3 Exercícios

1. A representação de uma determinada mensagem digital ternária, isto é formada por três bits, forma um cubo cujos vértices correspondem a mesma representação digital. Supondo que ao transmitirmos esta mensagem a mesma seja contaminada por ruído formando em torno de cada vértice uma nuvem esférica de valores aleatórios. O raio da esfera corresponde ao desvio padrão do sinal de ruído. Solucione o problema usando máquinas de vetor de suporte linear. Compare com a solução obtida na lista 2 onde foi usada uma rede de perceptron de Rosemblat com uma camada para atuar como classificador/decodificador. Para solução do problema defina antes um conjunto de treinamento e um conjunto de validação.

   RESOLUÇÃO:

   ...

2. Implemente a RBF considerando os algoritmos de treinamento para as três situações: (a) centros fixos e escolhidos aleatoriamente, (b) centros escolhidos através da seleção auto-supervisionada (algoritmo K-means) , (c) centros escolhidos através da seleção supervisionada, para as três questões abaixo:

   a) A função lógica $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$

   b)
   $$f(x) = \left[\frac{sin(\pi||x||)}{\pi||x||}\right], \; x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \; |x_1| \leq 10 \text{ e } |x_2| \leq 10$$

   c) $f(x) = x_1^2 + x_2^2 - 2x_1x_2 + x_1 + x_2 - 1$, $|x_1| \leq 10$, $|x_2| \leq 10$

   RESOLUÇÃO:

   As alternativas *a*) e *c*) possuiram erro igual a 0. Na alternativa *b*) não conseguiu-se bons resultados utilizando a *toolbox*, pois a saída da rede resulta em valor NaN.

   Segue o código utilizando a *toolbox* do MATLAB:

```
%% Questao 2
%% a)
clear

% Gerando dados
x1 = [0 1 0 1 0 1 0 1];
x2 = [0 0 1 1 0 0 1 1];
x3 = [0 0 0 0 1 1 1 1];
x = [x1; x2; x3];

f = xor(xor(x1,x2),x3);
padroes = [x; f]';
padroes = repmat(padroes,5,1);
ni = size(padroes,1);
padroes = padroes(randperm(ni),:);

% Treinando rede
net = newrbe(padroes(:,1:3)',padroes(:,4)');
%view(net)
% Simulando rede
y = sim(net,padroes(:,1:3)')';

%% b)
clear

x1 = -10:0.1:10;
x2 = -10:0.1:10;
x = zeros(1,length(x1));
for i=1:201
    x(i) = norm([x1(:,i) x2(:,i)]);
end

f = sin(x.*pi)./(x.*pi);

padroes = [x; f]';

net = newrbe(padroes(:,1)',padroes(:,2)');
y = sim(net,padroes(:,1)')';

%% c)
clear

x1 = -10:0.1:10;
x2 = -10:0.1:10;
x = [x1; x2];

f = x1.^2 + x2.^2 - 2.*x1.*x2 + x1 + x2 - 1;

padroes = [x; f]';

net = newrbe(padroes(:,1:2)',padroes(:,3)');

y = sim(net,padroes(:,1:2)')';
```

3. Considere o problema de classificação de padrões constituído neste caso de 12 padrões. A distribuição dos padrões tem como base um quadrado centrado no ponto (0.5,0.5) e lados iguais a 1. Os pontos (0.5,0.5), (1.0,0.5), (0.5,1.) e (0.0, 0.5) são centros de quatro semicírculos que se interceptam no interior do quadrado originando quatro classes e outras oito classes nas regiões de não interseção. Após gerar aleatoriamente dados que venham formar estas distribuições de dados, selecione um conjunto de treinamento e um conjunto de validação. Solucione o problema usando RBF, SVM e Máquina de Comitê. Verifique o desempenho do classificador usando o conjunto de validação e calculando a matriz de confusão e compare com o obtido na lista anterior usando MLP.

   RESOLUÇÃO:

4. Utilize uma rede NARX, no caso uma rede neural perceptron de múltiplas camadas com realimentação global, para fazer a predição de um passo, até predição de três passos da série temporal $x(n) = 1 + cos(n + cos2(n))$. Avalie o desempenho mostrando para cada caso os erros de predição.

   RESOLUÇÃO:

5. Implemente uma rede de Hopfield, para reconhecer as letras AFC. (Para cada letra forme uma matriz binária de pixel). Verifique o desempenho com as letras sendo apresentadas de forma ruidosa.

   RESOLUÇÃO:

6. Dado o modelo não linear de espaço de estado abaixo, obtenha o modelo de espaço de estados linearizado para ser utilizado no algoritmo EKF.

$$x(n+1) = f(n, x(n)) + v_1(n)$$

$$y(n) = c(n, x(n)) + v_2(n)$$

$$f(n, x(n)) = \begin{bmatrix} x_1(n) + x_2^2(n) \\ nx_1(n) - x_1(n)x_2(n) \end{bmatrix}$$

$$c(n, x(n)) = x_1(n)x_2^2(n) + v_2(n)$$

   RESOLUÇÃO:

7. Um problema interessante para testar a capacidade de uma rede neural atuar como classificado de padrões é o problema das duas espirais intercaladas. Gere os exemplos de treinamento usando as seguintes equações:
para espiral 1 $x = \frac{\theta}{4}cos(\theta)$, $y = \frac{\theta}{4}sen(\theta)$, $\theta \geq 0$
para espiral 2 $x = (\frac{\theta}{4} + 0.8)cos(\theta)$, $y = (\frac{\theta}{4} + 0.8)sen(\theta)$, $\theta \geq 0$
fazendo θ assumir 51 igualmente espaçados valores entre 0 e 20 radianos. Utilize uma rede competitiva e em seguida uma rede SOM para atuar como classificador auto-supervisionado, isto é, a espiral 1 sendo uma classe e espiral 2 sendo outra classe. Para comparar as regiões de decisões formadas pela rede , gere uma grade uniforme com 100 x 100 exemplos de teste em um quadrado [-5,5]. Esboce os pontos classificados pela rede.

RESOLUÇÃO:

Foram implementados o algoritmo competitivo e o algoritmo SOM, como segue abaixo junto com o resultado:

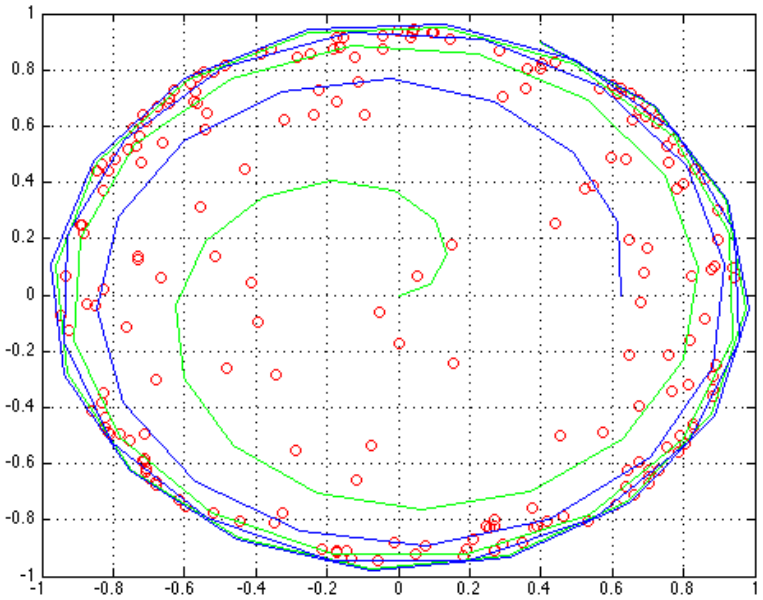

Figura 1: Resultado da rede SOM para a espiral.

```
classdef ALG_COMPET < handle
    properties
        xi % matriz de amostras de treinamento
        xin % vetor de entrada do k-ésimo padrão
        %xnk % vetro normalizado de entrada do k-ésimo padrão
        w % matriz do pesos
        wn % vetro normalizado de pesos
        eta % taxa de aprendizagem
        lx % nº de linhas das matizes de pesos e de treinamento (nº de entrdadas)
        cx % nº de colunas da matriz de amostras de treinamneto (nº de amostras)
        cw % nº de colunas da matriz de pesos (nº de neurônios)
    end
    methods
        function obj = ALG_COMPET(n_ent,n_amost, n_class)
            % inicialização da matriz de pesos e de amostaras
            obj.xi = -1*ones(n_ent+1, n_amost);% Sem bias
            %obj.w = 2*rand(n_ent+1,n_class)-1;
%             nw = ceil(n_amost);
%             sxi = 0;
%              for j = 1:n_class
%                  for i = 1:n_ent
%                       for k = 1:nw
%                            %sxi = sxi + obj.xi(i,k);
%                       end
%                      %obj.w(i,j) = (sxi/nw)*j;
%                  end
%              end
            obj.lx = n_ent+1;% nº de entradas sem bias
            obj.cx = n_amost;% nº de amostras
            obj.cw = n_class;% nº de neurônios
        end
        function nor(obj,entradat)
            obj.xi(2:obj.lx,:) = entradat;
            obj.w = 6*rand(obj.lx,obj.cw)-3;% Inicialização aleatória da matriz de↙
pesos
            obj.w(1,:) = -1;
            % Normalização dos vetores de pesos e de treinamento
             for xc = 1:obj.cx
                 obj.xin(:,xc) = obj.xi(:,xc)/norm(obj.xi(:,xc));
             end
             for wc = 1:obj.cw
                 obj.wn(:,wc) = obj.w(:,wc)/norm(obj.w(:,wc));
             end
            plot(obj.wn(2,:),obj.wn(3,:),'or')
            hold on
            plot(obj.xin(2,1:51),obj.xin(3,1:51),'b')
            plot(obj.xin(2,52:102),obj.xin(3,52:102),'y')
            %sphere
            grid
        end
        function [mdist, W] = trei(obj,txap,nep)
            obj.eta = txap; %Taxa de aprendizagem
            a=0;
            dist = [];
            mdist = [];
```

```
            ind = 0;
            epoca = 1;
            while a==0
                a=0;
                for k = 1:obj.cx
                    for j = 1:obj.cw
                        somatxw = 0;
                        for i = 1:obj.lx
                            somatxw = somatxw + (obj.xin(i,k)-obj.wn(i,j))^2;
                        end
                        dist(j) = sqrt(somatxw);
                    end
                    [mdist, ind] = min(dist);
                    obj.wn(:,ind) = obj.wn(:,ind)+ obj.eta*(obj.xin(:,k)-obj.wn(:,↙
ind));
                end
                if epoca == nep
                    a=1;
                end
                epoca=epoca+1;
            end
            plot(obj.wn(2,:),obj.wn(3,:),'or')
            hold on
            plot(obj.xin(2,1:51),obj.xin(3,1:51),'b')
            plot(obj.xin(2,52:102),obj.xin(3,52:102),'y')
            grid
            W = obj.wn;
        end
%         function ind = valid(obj, entradav, pesost)
%             obj.xi = entradav;
%             % Normalização dos vetores de pesos e de treinamento
%             for xc = 1:obj.cx
%                 obj.xin(:,xc) = obj.xi(:,xc)/norm(obj.xi(:,xc));
%             end
%             obj.wn = pesost;
%             plot(obj.wn(1,:),obj.wn(2,:),'*r')
%             hold on
%             plot(obj.xin(1,1:51),obj.xin(2,1:51),'*b')
%             plot(obj.xin(1,52:102),obj.xin(2,52:102),'*y')
%             grid
%             figure
%             for k = 1:obj.cx
%                     for j = 1:obj.cw
%                         somatxw = 0;
%                         for i = 1:obj.lx
%                             somatxw = somatxw + (obj.xin(i,k)-obj.wn(i,j))^2;
%                         end
%                         dist(j) = sqrt(somatxw);
%                     end
%                     [mdist ind(k)] = min(dist);
%                     if ind == 1
%                          plot(obj.xin(1,k),obj.xin(2,k),'*k')
%                     else
%                         plot(obj.xin(1,k),obj.xin(2,k),'*r')
%                     end
```

```
%                       hold on
%                   end
%                   plot3(obj.wn(1,:),obj.wn(2,:),'*y')
%                   grid
%               end
    end

end
```

```
classdef ALG_SOM < handle
    properties
        xi % matriz de amostras de treinamento
        xin % vetor de entrada do k-ésimo padrão
        w % matriz do pesos
        wn % vetro normalizado de pesos
        eta % taxa de aprendizagem
        gau % Função de vizinhança lateral dos neurônios no espaço de saída
        lx % nº de linhas das matizes de pesos e de treinamento (nº de entrdadas)
        cx % nº de colunas da matriz de amostras de treinamneto (nº de amostras)
        cw % nº de colunas da matriz de pesos (nº de neurônios)
    end
    methods
        function obj = ALG_SOM(n_ent,n_amost, n_neu)
            % Criação da matriz de pesos e de amostaras
            obj.xi = -1*ones(n_ent+1, n_amost);% O +1 é para o bias -1
            obj.w = ones(n_ent+1,n_neu);% matriz de pesos
%             nw = ceil(0.2*n_amost);
%             sxi = 0;
%             for j = 1:n_neu
%                 for i = 1:n_ent
%                     for k = 1:nw
%                         sxi = sxi + obj.xi(i,k);
%                     end
%                     obj.w(i,j) = (sxi/nw)*5*j;
%                 end
%             end
            obj.lx = n_ent+1;% nº de entradas
            obj.cx = n_amost;% nº de amostras
            obj.cw = n_neu;% nº de neurônios
        end

        function nors(obj,entradat,iniw)
            obj.xi(2:obj.lx,:) = entradat;
            if iniw == 'aleat'
            obj.w = 6*rand(obj.lx,obj.cw)-3;% Inicialização aleatória da matriz de↙
pesos
            obj.w(1,:) = -1;
            end
%             if iniw == 'aleat'
%             obj.w = 6*rand(obj.lx,obj.cw)-3;% Inicialização grade quadrada da matriz↙
de pesos
%             obj.w(1,:) = -1;
%             end
%             if iniw == 'aleat'
%             obj.w = 6*rand(obj.lx,obj.cw)-3;% Inicialização grade exagonal da matriz↙
de pesos
%             obj.w(1,:) = -1;
%             end
            % Inicialiazação e Normalização dos vetores de pesos e de treinamento
             for xc = 1:obj.cx
                 obj.xin(:,xc) = obj.xi(:,xc)/norm(obj.xi(:,xc));
             end
             for wc = 1:obj.cw
                 obj.wn(:,wc) = obj.w(:,wc)/norm(obj.w(:,wc));
```

```
            end
            plot(obj.wn(2,:),obj.wn(3,:),'or')
            hold on
            plot(obj.xin(2,1:51),obj.xin(3,1:51),'-g')
            plot(obj.xin(2,52:102),obj.xin(3,52:102),'-b')
            grid
        end
        function treis(obj,txap,nep,r_gau)
            obj.eta = txap; %Taxa de aprendizagem fixa
%             obj.eta = txap; %Taxa de aprendizagem variável
            a=0;
            ind = 0;
            epoca = 1;
            while a==0
                a=0;
                % Cálculo da distância entre a amostra atual e todos os vetores w
                for k = 1:obj.cx
                    for j = 1:obj.cw
                        somatxw = 0;
                        for i = 1:obj.lx
                            somatxw = somatxw + (obj.xin(i,k)-obj.wn(i,j))^2;
                        end
                        dist(j) = sqrt(somatxw);
                    end
                    [mdist, ind] = min(dist); %determinação do neurônio vencedor
                    % Cálculo da distância da vizinhança e atualização
                    for j = 1:obj.cw
                        somatww = 0;
                        for i = 1:obj.lx
                            somatww = somatww + (obj.wn(i,ind)-obj.wn(i,j))^2;
                        end
                        distr = sqrt(somatww);
                        %dp = r_gau*exp-()% Desvio padrão variavel
                        obj.gau = exp(-(distr^2/(2*(r_gau)^2)));
                        obj.wn(:,j) = obj.wn(:,j)+ obj.eta*obj.gau*(obj.xin(:,k)-obj.wn↙
(:,j));
                    end

                end
                if epoca == nep
                    a=1;
                end
                epoca=epoca+1;
            end
            plot(obj.wn(2,:),obj.wn(3,:),'or')
            hold on
            plot(obj.xin(2,1:51),obj.xin(3,1:51),'-g')
            plot(obj.xin(2,52:102),obj.xin(3,52:102),'-b')
            grid
            W = obj.wn;
        end

%         function ind = valid(obj, entradav, pesost)
%             obj.xi(2:obj.lx,:) = entradav;
%             % Normalização dos vetores de pesos e de treinamento
```

```
%             for xc = 1:obj.cx
%                 obj.xin(:,xc) = obj.xi(:,xc)/norm(obj.xi(:,xc));
%             end
%             obj.wn = pesost;
%             plot3(obj.wn(1,:),obj.wn(2,:),obj.wn(3,:),'*r')
%             hold on
%             plot3(obj.xin(1,:),obj.xin(2,:),obj.xin(3,:),'*')
%             grid
%             figure
%             for k = 1:obj.cx
%                     for j = 1:obj.cw
%                         somatxw = 0;
%                         for i = 1:obj.lx
%                             somatxw = somatxw + (obj.xin(i,k)-obj.wn(i,j))^2;
%                         end
%                         dist(j) = sqrt(somatxw);
%                     end
%                     [mdist ind(k)] = min(dist);
%                     if ind == 1
%                          plot3(obj.xin(1,k),obj.xin(2,k),obj.xin(3,k),'*k')
%                     else
%                         plot3(obj.xin(1,k),obj.xin(2,k),obj.xin(3,k),'*r')
%                     end
%                     hold on
%             end
%             plot3(obj.wn(1,:),obj.wn(2,:),obj.wn(3,:),'*y')
%             grid
%         end
    end

end
```

8. Considere a distribuição dos padrões que tem como base em um círculo com raio igual a 0.25 centrado origem. Os pontos +1 e -1 de cada eixo são centros de quatro semicírculos que se interceptam no interior a as regiões que excluem o círculo de raio igual a 0.25 do quadrado originando quatro classes. Gere aleatoriamente os dados que venham formar estas distribuições de dados. Utilize a rede SOM de modo a quantizar através da distribuição de neurônios a distribuição dos dados.

   RESOLUÇÃO:

   Foi feito a geração dos dados através da equação do círculo:

$$x = \sqrt{r^2 - (y - y_0)^2} + x_0$$

$$y = \sqrt{r^2 - (x - x_0)^2} + y_0$$

   A classificação dos dados foi feito através de um vetor $[QXQYHV]$, em que para cada ponto gerado QX representa um dos dois quadrantes no eixo X, podendo assim assumir o valor 0 ou 1, o equivalente ocorre para QY no eixo Y, H representa se o ponto gerado encontra-se dentro de um semi-círculo horizontal e V para um semi-círculo vertical. Com este vetor é possível classificar o ponto gerado, visto que se o ponto estiver com valores de H e V iguais a 1 significa que o ponto encontra-se dentro de dois semi-círculos simultaneamente, ou seja, na região de interesse. Os valores de QX e QY descrevem em quais das quatro regiões os pontos se encontram.

   A rede foi treinada utilizando a *toolbox* do MATLAB, com o método *newsom*, com uma arquitetura 10x10 uniformimente distribuida. Com 500 pontos gerados e 200 iterações, tem-se:

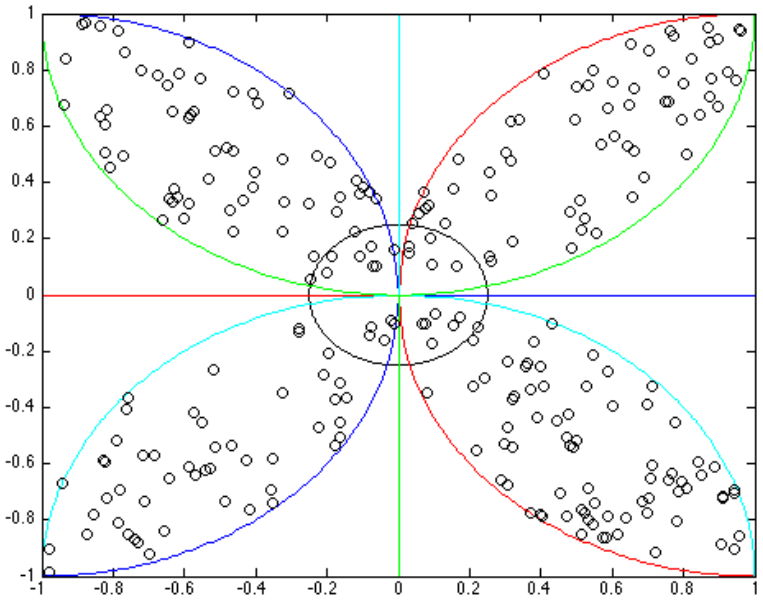

Figura 2: Desenho das regioes de interesse.

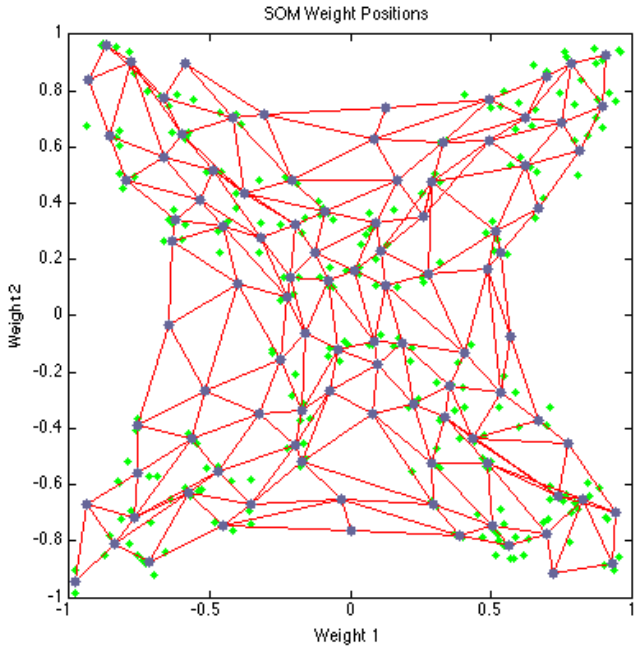

Figura 3: Resultado da rede.

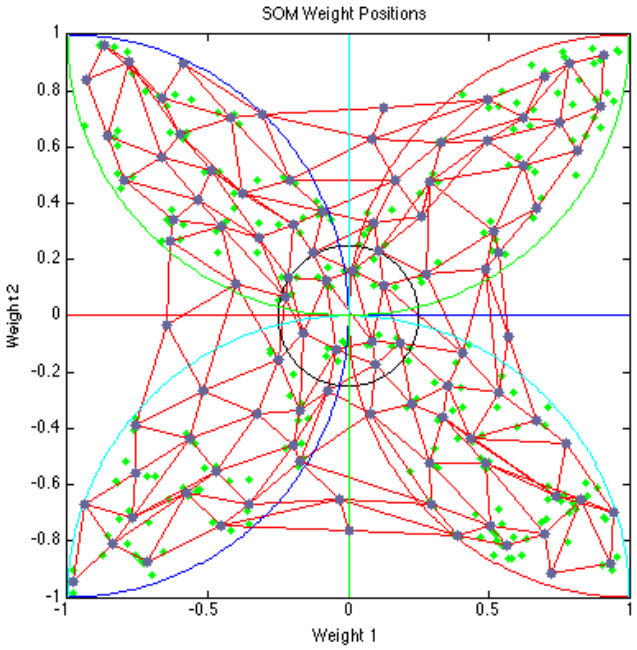

Figura 4: Resultado da rede com as regioes.

9. Pesquise e apresente o formalismo do algoritmo K-means por lote.

   RESOLUÇÃO:

10. Pesquise e apresente o formalismo do algoritmo SOM por lote.

    RESOLUÇÃO:

Trabalhos

1. Pesquise e apresente um trabalho sobre a reconstrução tridimensional usando a rede SOM e a rede Neuro-GAS.

2. Pesquise e apresente um trabalho sobre Neurofuzzy.

Data de entrega: 23/05/2013

A entrega e apresentação dos trabalhos correspondem a um processo de avaliação. Portanto a presença é obrigatório.

Os trabalhos e a lista podem ser feito em grupo de até três componentes.

Na apresentação os componentes serão submetidos a questionamentos sobre a solução da lista e o desenvolvimento dos trabalhos.

Desenvolvimento da Pesquisa

...