UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO

Trabalho da Unidade 3

IMPLEMENTAÇÃO E UTILIZAÇÃO DE UMA MLP PARA APROXIMAR FUNÇÕES MATEMÁTICAS

VANESSA DANTAS DE SOUTO COSTA

Natal-RN
2018

VANESSA DANTAS DE SOUTO COSTA

Trabalho da Unidade 3

IMPLEMENTAÇÃO E UTILIZAÇÃO DE UMA MLP PARA APROXIMAR FUNÇÕES MATEMÁTICAS

Este relatório é referente ao trabalho desenvolvido na Unidade III da disciplina Controle Inteligente, correspondente à 40% da nota da 3º unidade do semestre 2018.2 da Universidade Federal do Rio Grande do Norte, sob orientação do **Prof. Fábio Meneghetti Ugulino de Araújo.**

Professor: Fábio Meneghetti Ugulino de Araújo.

Natal-RN
2018

RESUMO

Este trabalho tem como objetivo explicitar a implementação utilizada para desenvolvimento da rede neural MLP (Multilayer Perceptron) embasado na estratégia de algoritmo conhecido como **B**ackPropagation. Ainda, foi solicitado que a implementação desenvolvida fosse utilizada para aproximar 5 funções matemáticas distintas escolhidas pelo professor.

Palavras-chave: Multilayer Perceptron,Backpropagation, Feedforward, validação,treinamento, termo momentum

Sumário

1 INTRODUÇÃO

A inteligência artificial é um campo que está presente no nosso dia a dia, um exemplo disso, conforme abordado na disciplina, é aproximar funções matemáticas através de uma rede neural. Para tanto, podemos utilizar estratégias de validação e treinamento embasados na Multilayer Perceptron para obtenção do melhor resultado possível. Alguns exemplos disso são as funções descritas neste relatório.

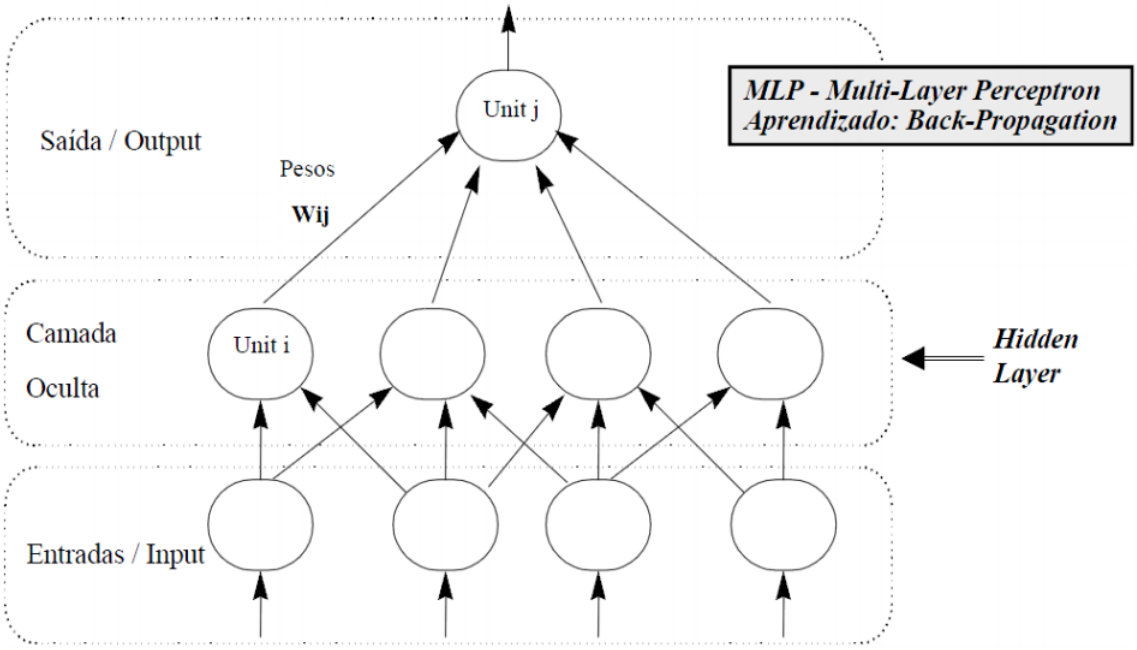
Multilayer Perceptron é uma rede neural semelhante à perceptron de Rosenblatt, mas com mais de uma camada de neurônios em alimentação direta. Tal tipo de rede é composta por camadas de neurônios ligadas entre si por sinapses com pesos. O aprendizado nesse tipo de rede é geralmente feito através do algoritmo de retro-propagação do erro conhecido como BackPropagation.

O foco deste trabalho é a implementação de uma rede neural multicamadas. Para testar seu funcionamento,utilizaremos o algoritmo da MLP implementado para aproximação de 5 funções matemáticas.

Com isso, esperamos gerar uma I.A. que possa aproximar com certa eficiência e precisão desejada determinado conjunto de dados.

2 FUNCIONAMENTO DO BACKPROPAGATION

Em primeiro momento, mostramos na imagem abaixo a arquitetura de uma rede MLP:



A função implementada possui os seguintes parâmetros:

Como Entradas:

in –> entrada matriz

ex.: Porta xor entradas saida esperada 1 0 1 1 1 0 0 0 0 0 1 1

out –> saída esperada

nce –> número de camadas escondidas

nuce –> número de neurônios na camada escondida.

Para a última camada (camada de saída), devemos ter apenas um neurônio e função de ativação linear.

TA = taxa de aprendizagem

face –> função de ativação da camada escondida

ini –> matriz de pesos iniciais(utilizada para agilizar treinamento). Se passada vazia, devemos iniciá-la aleatoriamente

epMax –> número de epocas para treinamento

emqTarget –> erro médio quadrático mínimo desejado

percentrein –> percentual de divisão do grupo de treinamento em “treinamento"e “validação", sugerido algo entre 70% e 80%

alphaMomento –> constante menor que 1, utilizada para cálculo do momento

Como saídas:

Pesos –> vetor de pesos

AtivacoesNos –> entradas de cada neurônio

saidas –> saida para plot

EMQ –> erro médio quadrático de cada época

Epoca –> número de iterações do BackPropagation

Uma MLP é uma rede neural feedforward que mapeia conjuntos de dados de entrada para conjunto de saídas apropriadas. Ela é composta por várias camadas de nós (vértices) em um grafo direcionado, cada camada é totalmente conectada na próxima. Exceto para os nós de entrada, cada nó é um neurônio com uma função de ativação.

MLP utiliza treinamento supervisionado. O processo de treinamento de redes MLP utiliza o algoritmo backpropagation conhecido também como regra delta generalizada.

A primeira fase do treinamento é o feedfoward, cujas amostras são inseridas nas entradas da rede e propagadas camada a camada até a produção das respectivas saídas, com intuito de obter as respostas da rede.

As respostas produzidas pelas saídas são comparadas com as respectivas respostas desejadas. São gerados desvios (erros), em seguida é aplicada a segunda fase do método backpropagation que é a backward (propagação reversa). Nessa fase as alterações dos pesos sinápticos e limiares de todos os neurônios da rede são executadas.

O treinamento das redes MLP com backpropagation pode demandar muitos passos no conjunto de treinamento, resultando um tempo de treinamento consideravelmente longo. Se for encontrado um mínimo local, o erro para o conjunto de treinamento satura em um valor maior que o aceitável.

Uma maneira de aumentar a taxa de aprendizado sem levar à oscilação é modificar a regra delta generalizada para incluir o termo momentum, uma constante que determina o efeito das mudanças passadas dos pesos na direção atual do movimento no espaço de pesos.

Desta forma, o termo momentum leva em consideração o efeito de mudanças anteriores de pesos na direção do movimento atual no espaço de pesos. O termo momentum se torna útil em espaços de erro que contenham longas gargantas, como curvas acentuadas ou vales com descidas suaves.

3 IMPLEMENTAÇÃO DA MLP

Modularizamos os passos necessários para o correto funcionamento da MLP em 5 funções distintas:

BackPropagation: responsável pelo treinamento utilizando momentum do grupo de treinamento

MLPnetwork: responsável pela validação utilizando os Pesos da rede treinada.

EvaluateNetwork: calcula a saída da rede.

FuncaoAtivacao: calcula a função de ativação podendo selecionar entre Tangente Sigmóide e Logarítmo Sigmóide.

FuncaoAtivacaoDerivada: calcula a derivada da função de ativação podendo selecionar entre Tangente Sigmóide e Logarítmo Sigmóide.

3.1 Código comentado do BackPropagation

```
%Author: Vanessa Dantas de Souto Costa
%email: vanessa.dantas796@gmail.com

%%%%%%%%%%%%%%%%%%%%%Entradas %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%in --> entrada matriz
%nAmostras = numero de entradas
%nSaidas = numero de saidas (exemplos de treinamento)

%ex.: Porta xor    entradas  saida esperada
%                   1 0         1
%                   1 1         0
%                   0 0         0
%                   0 1         1

%out --> s a da esperada
%nce --> n mero de camadas escondidas
%nuce --> n mero de neur nios na camada escondida.
%Obs.: para a ltima camada (camada de s a da), devemos ter apenas um
%neur nio e f u n  de a t i v a  linear
%TA = taxa de aprendizagem
%face --> f u n  de a t i v a  da camada escondida
%ini --> matriz de pesos iniciais(utilizada para agilizar treinamento). Se
%passada vazia, devemos inici -la aleatoriamente

%epMax --> n mero de epocas para treinamento
%emqTarget --> erro m dio quadr tico m nimo desejado
%percentrein --> percentual de divis o do grupo de treinamento em
%"treinamento" e "v a l i d a ", sugerido algo entre 70% e 80%

%alphaMomento --> constante menor que 1, utilizada para c lculo do momento

%%%%%%%%%%%%%%%%%%%%%S a das %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Pesos --> vetor de pesos
%AtivacoesNos --> entradas de cada neur nio
%saidas --> saida para plot
%EMQ --> erro m dio quadr tico de cada  oca
%Epoca --> n mero de i t e r a es do BackPropagation
```

```
38
39 %%%%%%%%%%%%%%% Fun  que executa o treinamento
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%
40
41 function [Pesos,AtivacoesNos,saidas,EMQ,Epoca]=BackPropagation(in,out,
   nce,nuce,TA,face,ini,epMax,emqTarget,percentrein, alphaMomento)
42
43 % Separar Grupo de treino do grupo de validacao
44      %embaralhar colunas
45      [N p]=size(in);
46      auxrand=randperm(N);
47      in=in(auxrand,:);%desse modo trocamos a ordem das linhas
48      out=out(auxrand,:);
49
50      %devemos pegar percentrein para treinamento e o restante para
51      %validacao
52      inTreino =in(1:floor(percentrein*N),:);
53      outTreino =out(1:floor(percentrein*N),:);
54      inValidacao =in((floor(percentrein*N)+1):end,:);
55      outValidacao =out((floor(percentrein*N)+1):end,:);
56
57      in=inTreino;
58      out=outTreino;
59
60 %inicializar saidas
61 saidas=zeros(length(in(:,1)),1);
62 saidasValidacaoTreinamento=zeros(length(inValidacao(:,1)),1);
63
64 %determina o nmero de neurnios de cada camada escondida
65 nNeuroniosCadaCamadaEscondida = ones(1,nce)*nuce;
66 %calcula o nAmostras e o nSaidas com base na resposta desejada
67 [nAmostras nSaidas] = size(out);
68
69 %calcula o nmero de ns  baseado na entrada
70 nNosEntrada = length(in(1,:));
71
72 %calcula nCamadas e o nNosPorCamada
73 nCamadas = 2 + length(nNeuroniosCadaCamadaEscondida);
74 nNosPorCamada = [nNosEntrada nNeuroniosCadaCamadaEscondida nSaidas];
75
76 %adicionar o Bias
77 nNosPorCamada(1:end-1) = nNosPorCamada(1:end-1) + 1;
78 in = [ones(length(in(:,1)),1) in];
```

```
79  inValidacao=[ones(length(inValidacao(:,1)),1) inValidacao];
80  %Pesos conectando nos de bias com camadas anteriores so
81  %desnecessariosare useless, mas para simplificar o codigo e torn-lo
        mais
82  %rpido consideramos PesosDelta = cell(1,nCamadas);
83  Pesos = cell(1, nCamadas);
84  PesosDelta = cell(1, nCamadas);
85
86  for i = 1:length(Pesos)-1
87      Pesos{i} = 2*rand(nNosPorCamada(i), nNosPorCamada(i+1))-1;
88      Pesos{i}(:,1) = 0; %Pesos nos do bias com camada anterior (
            redundante)
89      PesosDelta{i} = zeros(nNosPorCamada(i), nNosPorCamada(i+1));
90  end
91
92  %Pesos virtuais para nos de saida
93  Pesos{end} = ones(nNosPorCamada(end), 1);
94
95  %caso seja passado um parametro para iniciar os pesos
96  if ~isempty(ini)
97      Pesos=ini;
98  end;
99
100
101 AtivacoesNos = cell(1, nCamadas);
102 for i = 1:length(AtivacoesNos)
103     AtivacoesNos{i} = zeros(1, nNosPorCamada(i));
104 end
105 %  necessrios para o treinamento do Backpropagation de trs pra
        frente
106 NosErrosPropagados = AtivacoesNos;
107
108 emqTargetComprido = 0; %verificar se o erro emqTarget foi alcanado
109
110 %inicializa do vetor de erros por oca
111 EMQ = -1 * ones(1,epMax);
112
113 %Backpropagation e atualiza de pesos delta
114 PesosDeltaAntigosMomento = PesosDelta;
115 for Epoca = 1:epMax
116     for Amostra = 1:length(in(:,1))
117         AtivacoesNos{1} = in(Amostra,:);
118         for Camada = 2:nCamadas
```

```
119             AtivacoesNos{Camada} = AtivacoesNos{Camada-1}*Pesos{Camada
                    -1};
120             %AtivacoesNos{Camada} = FuncaoAtivacao(AtivacoesNos{Camada
                    },face);
121             %Porque os ns  do Bias no tem Pesos conectados com
                    camadas anteriores
122             if (Camada ~= nCamadas)
123                 AtivacoesNos{Camada}(1) = 1;
124                 AtivacoesNos{Camada} = FuncaoAtivacao(AtivacoesNos{
                        Camada},face);
125             end
126         end
127         % Armazenamento dos erros passados para trs
128         % (As gradiente of the bias nodes are zeros, they won't
                contribute to previous Camada errors nor PesosDelta)
129         NosErrosPropagados{nCamadas} =  out(Amostra,:)-AtivacoesNos{
                nCamadas};
130         for Camada = nCamadas-1:-1:1
131             if(Camada~=(nCamadas-1))
132                 gradiente=FuncaoAtivacaoDerivada(AtivacoesNos{Camada
                        +1},face);
133             else
134                 gradiente=1;
135             end
136             for node=1:length(NosErrosPropagados{Camada}) % For all
                    the Nodes in current Camada
137                 NosErrosPropagados{Camada}(node) =  sum(
                        NosErrosPropagados{Camada+1} .* gradiente .* Pesos{
                        Camada}(node,:) );
138             end
139         end
140         % Calculo dos pesos delta passados para trs (antes da
                multiplica  pela taxa de aprendizagem)
141         for Camada = nCamadas:-1:2
142             if(Camada~=nCamadas)
143                 derivative = FuncaoAtivacaoDerivada(AtivacoesNos{
                        Camada},face);
144             else
145                 derivative=1;
146             end
147             PesosDelta{Camada-1} = PesosDelta{Camada-1} + AtivacoesNos
                    {Camada-1}' * (NosErrosPropagados{Camada} .* derivative
                    );
```

```
148         end
149     end
150 
151 
152     %Aplicar Momento
153         for Camada = 1:nCamadas
154         PesosDelta{Camada} = TA*PesosDelta{Camada} + alphaMomento*
                PesosDeltaAntigosMomento{Camada};
155      end
156         PesosDeltaAntigosMomento = PesosDelta;
157 
158      % Atualiza  dos pesos
159     for Camada = 1:nCamadas-1
160         Pesos{Camada} = Pesos{Camada} + PesosDelta{Camada};
161     end
162 
163     % Resetar PesosDelta para Zeros
164     for Camada = 1:length(PesosDelta)
165         PesosDelta{Camada} = 0 * PesosDelta{Camada};
166     end
167 
168 
169     for Amostra = 1:length(in(:,1))
170         saidas(Amostra) = EvaluateNetwork(in(Amostra,:), AtivacoesNos,
                Pesos,face);
171     end
172 
173     %Calcular EMQ da epoca
174     %Validacao do Treinamento
175     for Amostra = 1:length(inValidacao(:,1))
176         saidasValidacaoTreinamento(Amostra)=EvaluateNetwork(
            inValidacao(Amostra,:), AtivacoesNos, Pesos,face);
177     end
178 
179 
180     EMQ(Epoca)= sum((saidasValidacaoTreinamento-outValidacao).^2)/(
        length(inValidacao(:,1)));
181     if (EMQ(Epoca) < emqTarget)
182         emqTargetComprido = 1;
183     end
184 
185 
186     display([int2str(Epoca) ' epocas de um total de ' int2str(epMax) '
```

```
        epocas  m ximas . EMQ = ' num2str(EMQ(Epoca)) ' Taxa de
       Aprendizagem = ' ...
187     num2str(TA) '.']);
188
189     %Caso tenhamos atingido o erro desejado, podemos parar o programa
190     if (emqTargetComprido)
191         EMQ=EMQ(1:Epoca);
192         break;
193     end
194
195     %salvar variaveis
196     ini=Pesos;
197     save('SaidasBackPropagation.mat','Pesos','AtivacoesNos','EMQ','
           saidas','Epoca');
198     save('ini.mat','ini');
199 end
200
201
202
203 end
```

3.2 Código comentado da MLPnetwork

```
1  %Author: Vanessa Dantas de Souto Costa
2  %email: vanessa.dantas796@gmail.com
3
4
5  %in --> entrada matriz
6  %Pesos --> vetor de pesos
7  %AtivacoesNos --> entradas de cada neurnio
8  %face -->Funcao  a t i v a
9
10 % F u n   para validacao dado que a rede ja foi treinada
11
12 function [saidas]=MLPnetwork(in,AtivacoesNos,Pesos,face)
13     in = [ones(length(in(:,1)),1) in];
14     saidas=zeros(length(in(:,1)),1);
15     for Amostra = 1:length(in(:,1))
16             saidas(Amostra) = EvaluateNetwork(in(Amostra,:),
                  AtivacoesNos, Pesos,face);
17     end
18 end
```

3.3 Código comentado da EvaluateNetwork

```
%Author: Vanessa Dantas de Souto Costa
%email: vanessa.dantas796@gmail.com

function saidas = EvaluateNetwork(Amostra, AtivacoesNos, Pesos,face)

nCamadas = length(AtivacoesNos);

AtivacoesNos{1} = Amostra;
for Camada = 2:nCamadas
    AtivacoesNos{Camada} = AtivacoesNos{Camada-1}*Pesos{Camada-1};

    if (Camada ~= nCamadas) %Because bias nodes don't have Pesos
       connected to previous Camada
        AtivacoesNos{Camada} = FuncaoAtivacao(AtivacoesNos{Camada},
           face);
        AtivacoesNos{Camada}(1) = 1;
    end
end

saidas = AtivacoesNos{end};

end
```

3.4 Código comentado da FuncaoAtivacao

```
%Author: Vanessa Dantas de Souto Costa
%email: vanessa.dantas796@gmail.com

% Activation Function
function fx = FuncaoAtivacao(x,face)

%%%%%%%%%%%%%% definir a fun de ativa e sua derivada
   %%%%%%%%%%%%%%

    if strcmpi(face,'LOGSIGMOIDE')
        %usamos como fun de ativa :LOG sigmoide
        %FA(X) = 1./(1+exp(-X))
        fx=1./(1+exp(-x));
    else
        if strcmpi(face,'TANGENTESIGMOIDE')
```

```
            %usamos como fun de ativa :tangente sigmoide
            %FA(X) = 2./(1+exp(-2.*X)) - 1;
            fx=2./(1+exp(-2.*x)) - 1;
        else
            disp('ERRO: fun de ativa no especificada');
            return;
        end
    end

end
```

3.5 Código comentado da FuncaoAtivacaoDerivada

```
%Author: Vanessa Dantas de Souto Costa
%email: vanessa.dantas796@gmail.com

% Activation Function
function fx_drev = FuncaoAtivacaoDerivada(x,face)
   %%%%%%%%%%%%% definir a fun de ativa e sua derivada
      %%%%%%%%%%%%%%

    if strcmpi(face,'LOGSIGMOIDE')
        %usamos como fun de ativa :LOG sigmoide
        %FA(X) = 1./(1+exp(-X))
        fx_drev= exp(-x)./((exp(-x) + 1).^2);

    else
        if strcmpi(face,'TANGENTESIGMOIDE')
            %usamos como fun de ativa :tangente sigmoide
            %FA(X) = 2./(1+exp(-2.*X)) - 1;
            fx_drev=(4*exp(-2*x))./((exp(-2*x) + 1).^2);
        else
            disp('ERRO: fun de ativa no especificada');
            return;
        end
    end

end
```

4 APLICAÇÃO

Como dito, a MLP implementada foi utilizada para aproximar 5 funções matématicas escolhidas pelo conforme conforme mostradas abaixo:

4.1 função 1: f(x)=sin(x)./x

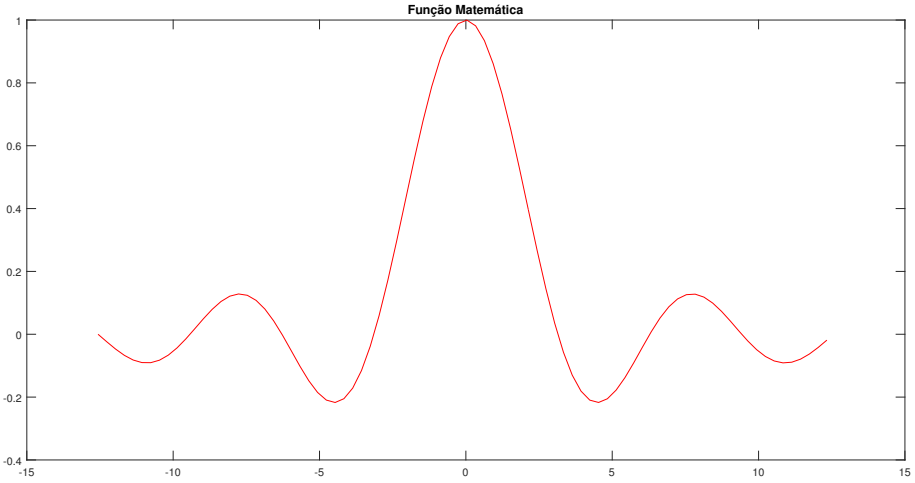
Script Feito para criar grupo de Treinamento e Validação, chamar as funções BackPropagation e MLPnetwork e plotar os resultados solicitados pelo professor:

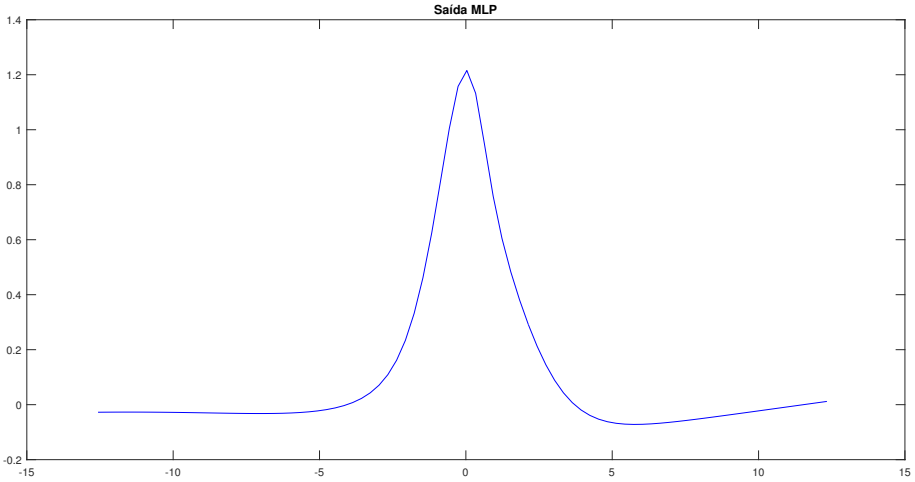
```
%Author: Vanessa Dantas de Souto Costa
%email: vanessa.dantas796@gmail.com

%Primeira fun f(x)=sin(x)./x

%treinamento
passoTreinamento=0.1;

x=-4*pi:passoTreinamento:4*pi;
y=sin(x)./x;
x=x';
y=y';
in=x./max(abs(x)); %normalizar
out=y;

nce=3;
nuce=10;
TA=0.0014;

epMax=1000;
emqTarget=0.01;
percentrein=0.7;
alphaMomento=0.9;
face='LOGSIGMOIDE'; %opes 'LOGSIGMOIDE' e 'TANGENTESIGMOIDE'
ini=[];

%carregar variveis
%load('ini.mat')
load('SaidasBackPropagation1.mat');

[Pesos,AtivacoesNos,saidas,EMQ,Epoca]=BackPropagation(in,out,nce,nuce,
   TA,face,ini,epMax,emqTarget,percentrein,alphaMomento);
EMQ=EMQ(1:Epoca);
```

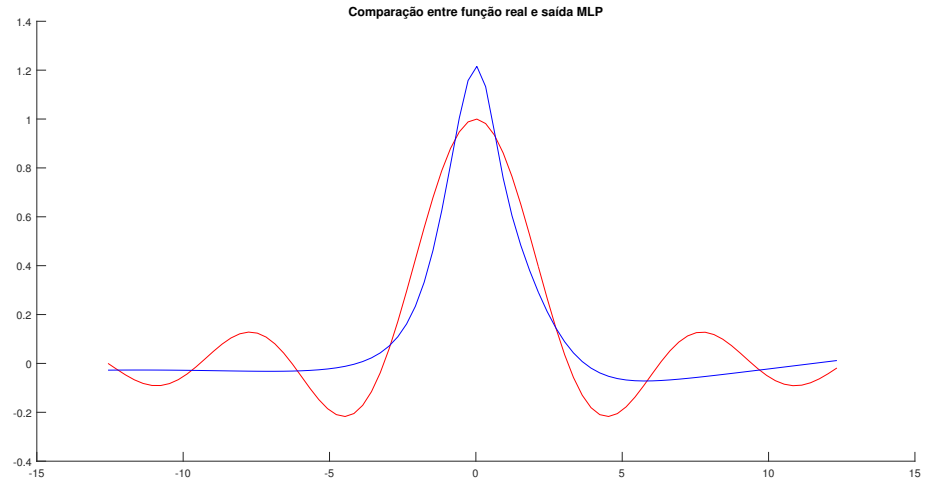
```
33
34 %salvar variaveis
35 save('SaidasBackPropagation1.mat','Pesos','AtivacoesNos','EMQ','saidas
   ','Epoca');
36
37 %plot da sada do treinamento
38 %figure();
39 %plot(saidas);
40
41 %validacao
42 passoValidacao=0.3;
43
44 xValidacao=-4*pi:passoValidacao:4*pi;
45 yValidacao=sin(xValidacao)./xValidacao;
46 xValidacao=xValidacao';
47 yValidacao=yValidacao';
48
49 inValidacao=xValidacao;
50 outValidacao=yValidacao;
51
52 [saidasValidacao]=MLPnetwork(inValidacao,AtivacoesNos,Pesos,face);
53
54
55 %plot das solicitaes feitas pelo professor
56
57 %ErroQuadrticoporca;
58 figure();
59 plot(EMQ);
60 title('Erro mdio quadrtico por Epoca');
61
62 %Errodecompara;
63 erroComp=saidasValidacao-outValidacao;
64 figure();
65 plot(erroComp);
66 title('Erro de compara entre a fun real e a sada da MLP');
67
68 % Grficodecompara ;
69 figure();
70 hold on
71 title('Compara entre fun real e sada MLP');
72 plot(xValidacao,yValidacao,'r');
73 plot(xValidacao,saidasValidacao,'b');
74 hold off
```

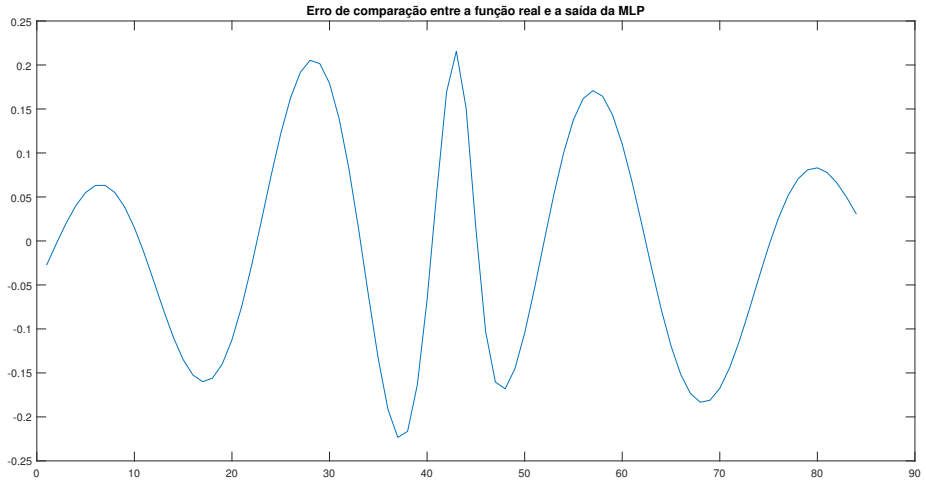
```
75
76 % S a dadaredeneural ;
77 figure();
78 plot(xValidacao,saidasValidacao,'b');
79 title('Sada MLP');
80
81 % S a dadafunmatemca ;
82 figure();
83 plot(xValidacao,yValidacao,'r');
84 title('Fun Matemtica');
```

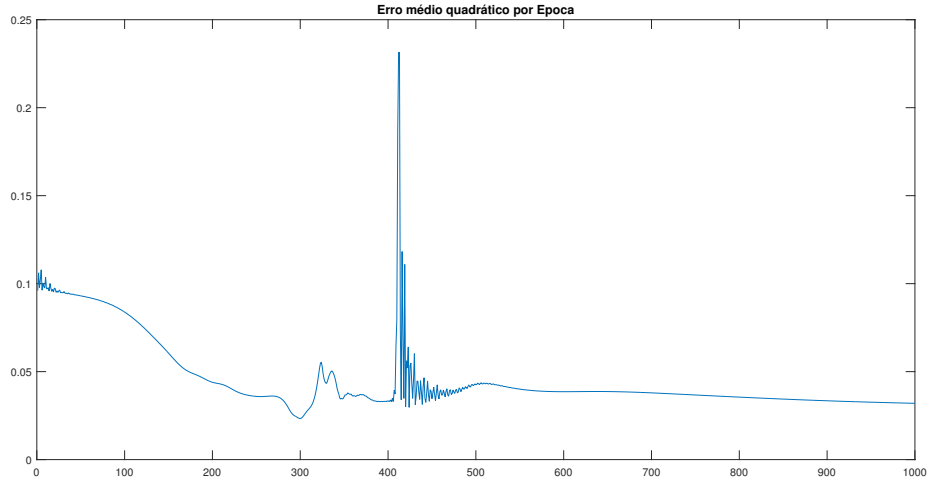
Plots Gerados pelo código:











Para essa função, a MLP aproximou razoavelmente, percebemos que a função tem uma leve inclinação como se fosse seguir as ondulações fora do impulso, mas ainda não o faz.

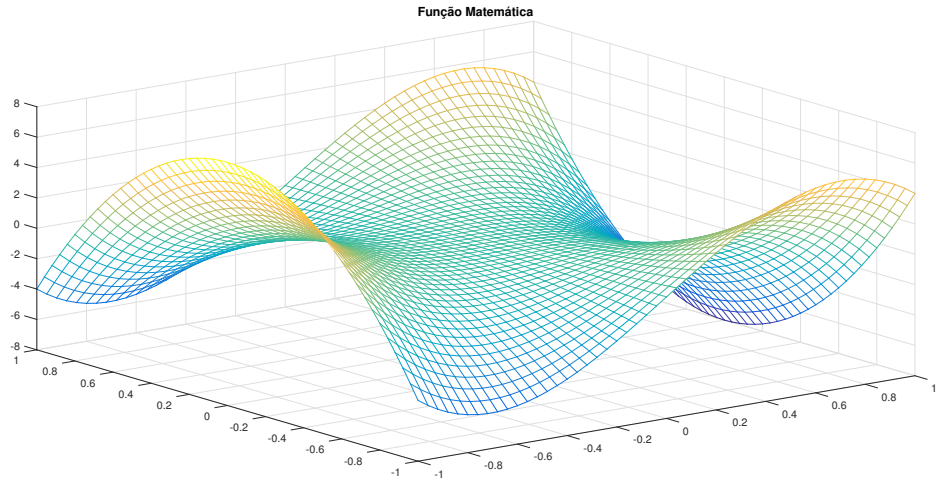
4.2 função 2: f(x,y)=12*x.*y.$^2 - 8 * x.^3$

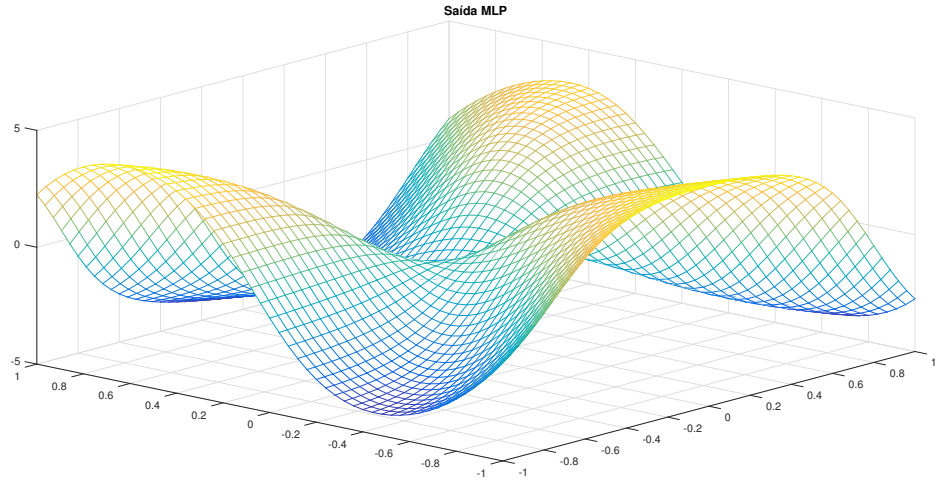
Script Feito para criar grupo de Treinamento e Validação, chamar as funções BackPropagation e MLPnetwork e plotar os resultados solicitados pelo professor:

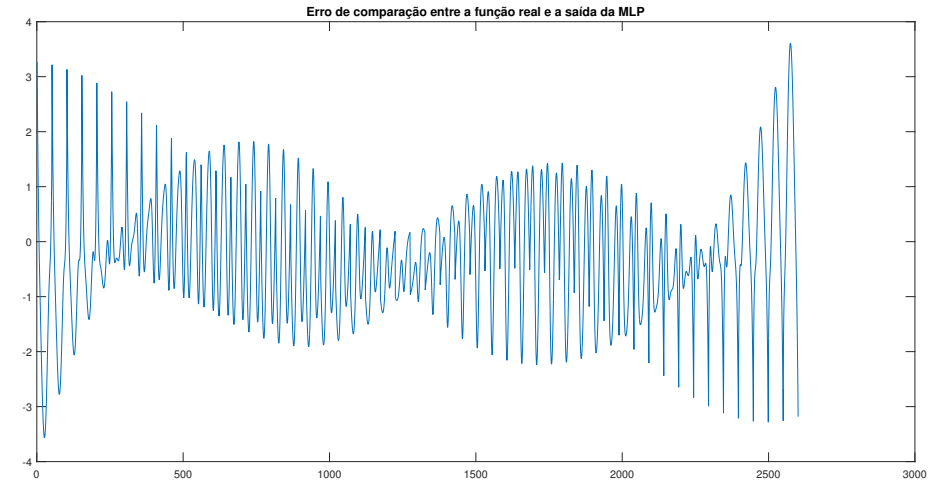
```
%Author: Vanessa Dantas de Souto Costa
%email: vanessa.dantas796@gmail.com

%Segunda fun f(x,y)=12*x.*y.^2-8*x.^3;

%treinamento
passoTreinamento=0.1;

[x,y]=meshgrid(-1:passoTreinamento:1);
z = 12*x.*y.^2-8*x.^3;
in=[x(:)./max(max(abs(x(:)))) y(:)./max(max(abs(y(:))))];
out=z(:);

nce=1;
nuce=20;
TA=0.0005;

epMax=1000;
emqTarget=0.01;
percentrein=0.7;
alphaMomento=0.9;
face='LOGSIGMOIDE'; %opes 'LOGSIGMOIDE' e 'TANGENTESIGMOIDE'
ini=[];

%carregar variveis
%load('ini.mat')
load('SaidasBackPropagation2.mat');

[Pesos,AtivacoesNos,saidas,EMQ,Epoca]=BackPropagation(in,out,nce,nuce,
   TA,face,ini,epMax,emqTarget,percentrein,alphaMomento);
EMQ=EMQ(1:Epoca);

%salvar variaveis
save('SaidasBackPropagation2.mat','Pesos','AtivacoesNos','EMQ','saidas
   ','Epoca');

%validacao
passoValidacao=0.04;

[xValidacao,yValidacao]=meshgrid(-1:passoValidacao:1);
zValidacao = 12*xValidacao.*yValidacao.^2-8*xValidacao.^3;

inValidacao=[xValidacao(:) yValidacao(:)];
```

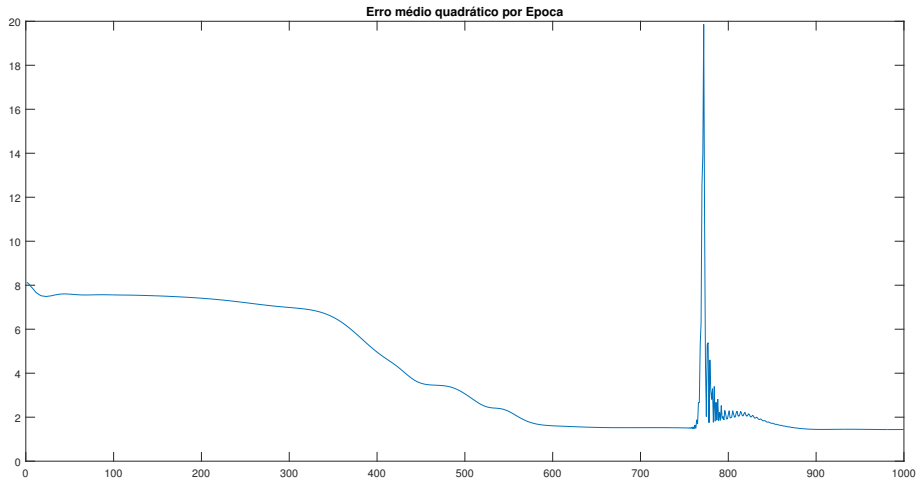
```
42 outValidacao=zValidacao(:);
43
44 [saidasValidacao]=MLPnetwork(inValidacao,AtivacoesNos,Pesos,face);
45 SaidaPlot=vec2mat(saidasValidacao,sqrt(length(saidasValidacao)));
46
47 %plot das solicitaes feitas pelo professor
48
49 %ErroQuadrticoporca;
50 figure();
51 plot(EMQ);
52 title('Erro mdio quadrtico por Epoca');
53
54
55 %Errodecompara;
56 erroComp=saidasValidacao-outValidacao;
57 figure();
58 plot(erroComp);
59 title('Erro de compara entre a fun real e a sada da MLP');
60
61 % G r ficodecompara ;
62 %figure();
63 %hold on
64 %mesh(xValidacao,yValidacao,zValidacao);
65 %mesh(xValidacao,yValidacao,SaidaPlot);
66 %title('Compara entre fun real e sada MLP');
67 %hold off
68
69 % S a dadaredeneural ;
70 figure();
71 mesh(xValidacao,yValidacao,SaidaPlot);
72 title('Sada MLP');
73
74 % S a dadafunmatemca ;
75 figure();
76 mesh(xValidacao,yValidacao,zValidacao);
77 title('Fun Matemtica');
```

Plots Gerados pelo código:


Função Matemática
8
6
4
2
0
-2
-4
-6
-8
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


Saída MLP
5
0
-5
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


Erro de comparação entre a função real e a saída da MLP
4
3
2
1
0
-1
-2
-3
-4
0
500
1000
1500
2000
2500
3000



Para essa função, a MLP aproximou bem, percebemos que a função gerada é bem parecida com a função real matemática.

4.3 função 3: f(x,y)=(y.*cos(x)+x.*sin(y))./(x.*y)

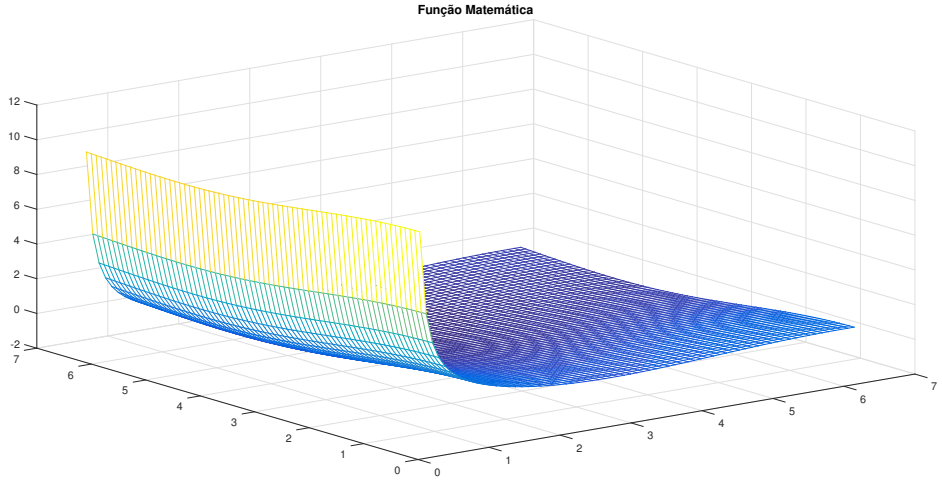
Script Feito para criar grupo de Treinamento e Validação, chamar as funções BackPropagation e MLPnetwork e plotar os resultados solicitados pelo professor:

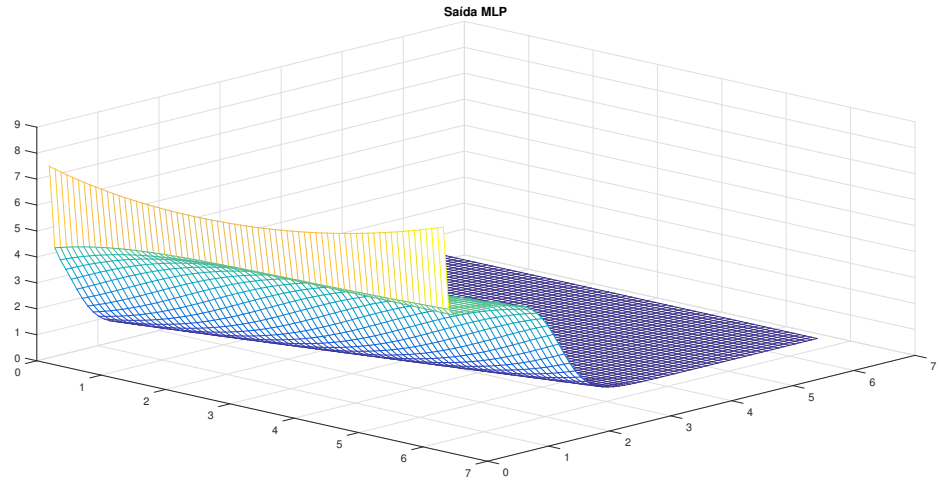
```
%Author: Vanessa Dantas de Souto Costa
%email: vanessa.dantas796@gmail.com

%Terceira fun f(x,y)=( y.*cos(x)+x.*sin(y))./(x.*y);

%treinamento
passoTreinamento=0.9;

[x,y]=meshgrid(0.1:passoTreinamento:2*pi);
z = (y.*cos(x)+x.*sin(y))./(x.*y);
in=[x(:)./max(max(abs(x(:)))) y(:)./max(max(abs(y(:))))];
out=z(:);

nce=1;
nuce=25;
TA=0.0005;

epMax=700;
emqTarget=0.01;
percentrein=0.7;
alphaMomento=0.9;
face='LOGSIGMOIDE'; % opes 'LOGSIGMOIDE' e 'TANGENTESIGMOIDE'
ini=[];
```

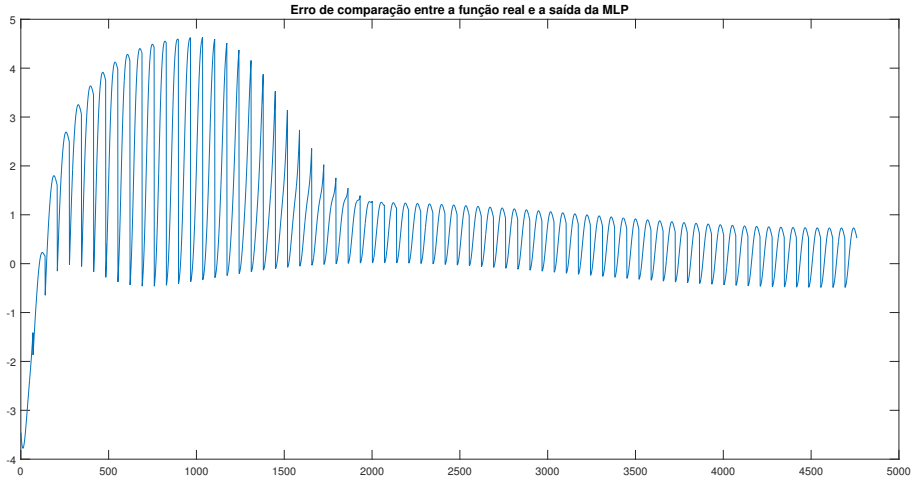
```
24
25 %carregar variveis
26 %load('ini.mat')
27 load('SaidasBackPropagation3.mat');
28
29 [Pesos,AtivacoesNos,saidas,EMQ,Epoca]=BackPropagation(in,out,nce,nuce,
   TA,face,ini,epMax,emqTarget,percentrein,alphaMomento);
30 EMQ=EMQ(1:Epoca);
31
32 %salvar variaveis
33 save('SaidasBackPropagation3.mat','Pesos','AtivacoesNos','EMQ','saidas
   ','Epoca');
34
35 %validacao
36 passoValidacao=0.09;
37
38 [xValidacao,yValidacao]=meshgrid(0.1:passoValidacao:2*pi);
39 zValidacao = ( yValidacao.*cos(xValidacao)+xValidacao.*sin(yValidacao)
   )./(xValidacao.*yValidacao);
40 inValidacao=[xValidacao(:) yValidacao(:)];
41 outValidacao=zValidacao(:);
42
43 [saidasValidacao]=MLPnetwork(inValidacao,AtivacoesNos,Pesos,face);
44 SaidaPlot=vec2mat(saidasValidacao,sqrt(length(saidasValidacao)));
45
46 %plot das solicitaes feitas pelo professor
47
48 %ErroQuadrticoporca;
49 figure();
50 plot(EMQ);
51 title('Erro mdio quadrtico por Epoca');
52
53 %Errodecompara;
54 erroComp=saidasValidacao-outValidacao;
55 figure();
56 plot(erroComp);
57 title('Erro de compara entre a fun real e a sada da MLP');
58
59 % Grficodecompara ;
60 %figure();
61 %hold on
62 %mesh(xValidacao,yValidacao,zValidacao);
63 %mesh(xValidacao,yValidacao,SaidaPlot);
```

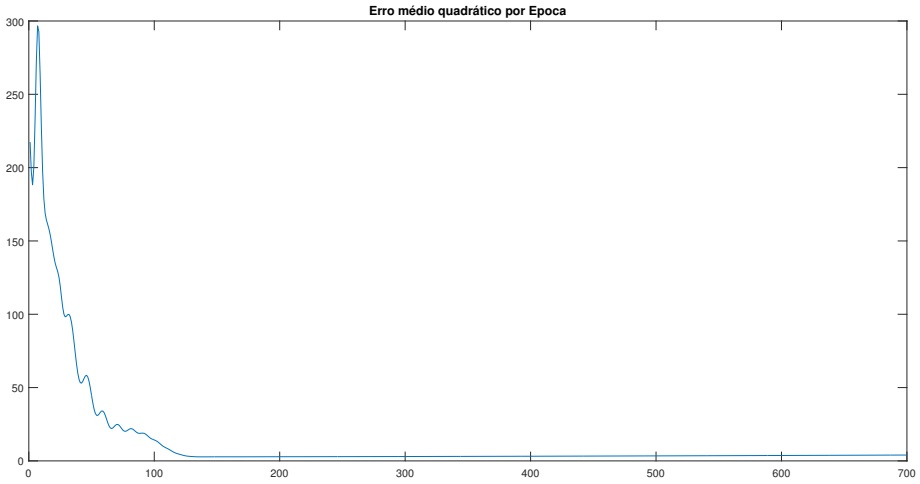
```
%title('Compara  entre  fun  real e sada MLP');
%hold off

% S a dadaredeneural ;
figure();
mesh(xValidacao,yValidacao,SaidaPlot);
title('Sada MLP');


%   S   a   dadafunmatemca   ;
figure();
mesh(xValidacao,yValidacao,zValidacao);
title('Fun  Matemtica');
```

Plots Gerados pelo código:









Para essa função, a MLP aproximou razoavelmente, percebemos que a função gerada lembra a função real matemática. No entanto, apresenta algumas ondulações em regiões indesejadas.

4.4 função 4: f(x,y)= -20*exp(-0.2*sqrt(0.5*x.$^2 + y.^2$)) $- exp(0.5 * cos(2 * x. * pi) + cos(2 * y. * pi)) + exp(1) + 20$

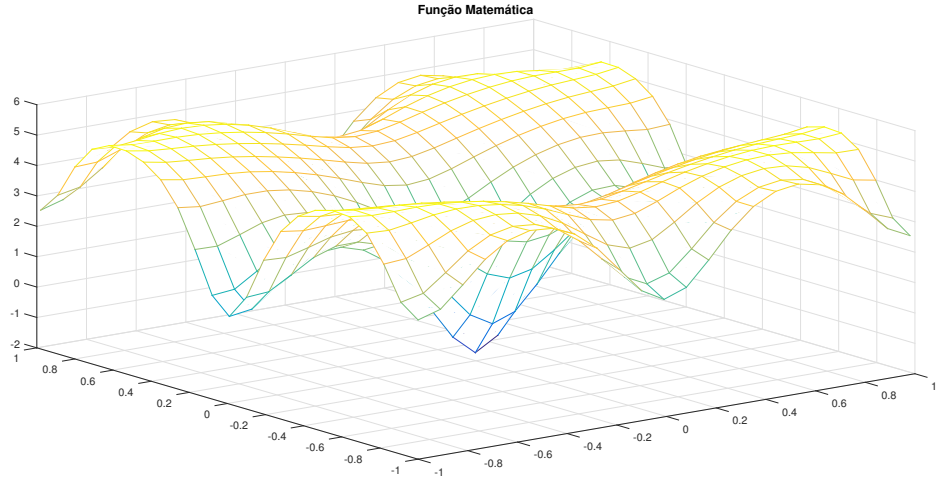
Script Feito para criar grupo de Treinamento e Validação, chamar as funções BackPropagation e MLPnetwork e plotar os resultados solicitados pelo professor:

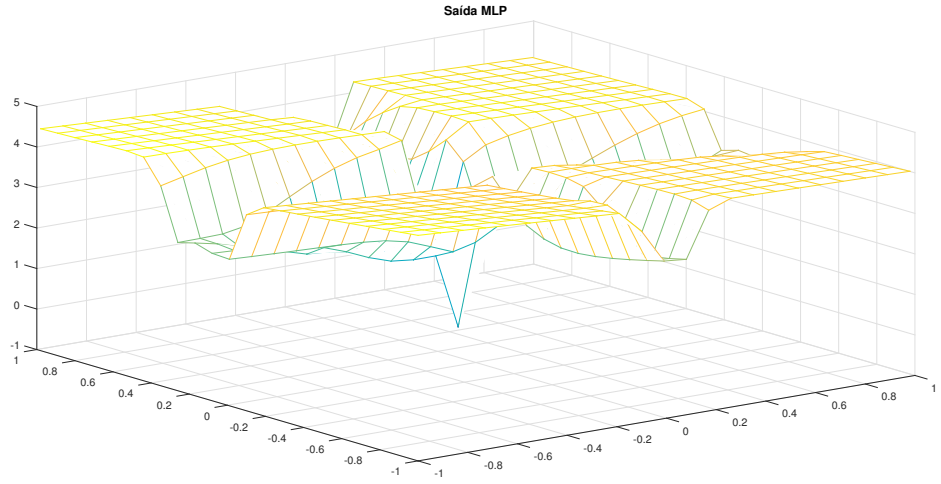
```
%Author: Vanessa Dantas de Souto Costa
%email: vanessa.dantas796@gmail.com

%Quarta fun f(x,y)= -20*exp(-0.2*sqrt(0.5*x.^2+y.^2))-exp(0.5*cos
   (2*x.*pi)+cos(2*y.*pi)) + exp(1) + 20;

%treinamento
passoTreinamento=0.1;

```

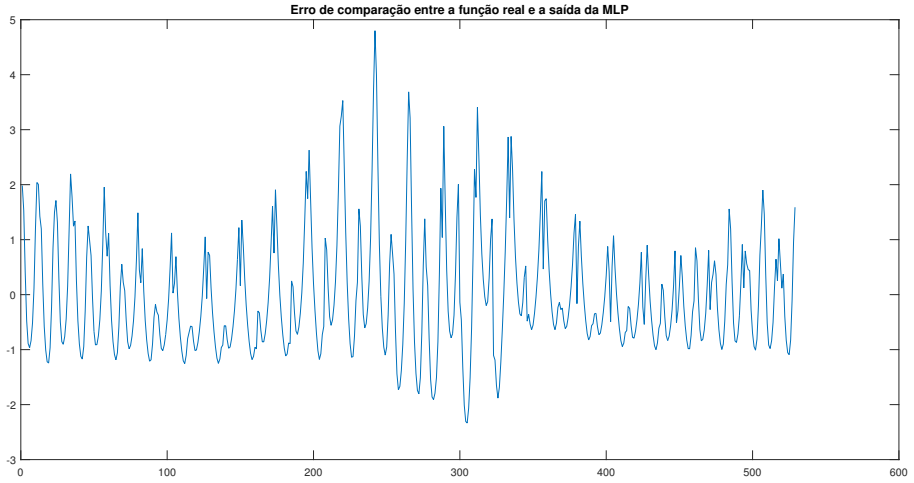
```
[x,y]=meshgrid(-1:passoTreinamento:1);
z = -20*exp(-0.2*sqrt(0.5*x.^2+y.^2))-exp(0.5*cos(2*x.*pi)+cos(2*y.*pi
   )) + exp(1) + 20;
in=[x(:)./max(max(abs(x(:)))) y(:)./max(max(abs(y(:))))];
out=z(:);

nce=2;
nuce=26;
TA=0.0005;

epMax=1000;
emqTarget=0.01;
percentrein=0.7;
alphaMomento=0.9;
face='LOGSIGMOIDE'; % o p es  'LOGSIGMOIDE' e 'TANGENTESIGMOIDE'
ini=[];

%carregar  v a r i veis
%load('ini.mat')
load('SaidasBackPropagation4.mat');

[Pesos,AtivacoesNos,saidas,EMQ,Epoca]=BackPropagation(in,out,nce,nuce,
   TA,face,ini,epMax,emqTarget,percentrein,alphaMomento);
EMQ=EMQ(1:Epoca);

%salvar  variaveis
save('SaidasBackPropagation4.mat','Pesos','AtivacoesNos','EMQ','saidas
   ','Epoca');

%plot  da  s a da  do  treinamento
%figure();
%plot(saidas);

%validacao
passoValidacao=0.09;

[xValidacao,yValidacao]=meshgrid(-1:passoValidacao:1);
zValidacao = -20*exp(-0.2*sqrt(0.5*xValidacao.^2+yValidacao.^2))-exp
   (0.5*cos(2*xValidacao.*pi)+cos(2*yValidacao.*pi)) + exp(1) + 20;
inValidacao=[xValidacao(:) yValidacao(:)];
outValidacao=zValidacao(:);

[saidasValidacao]=MLPnetwork(inValidacao,AtivacoesNos,Pesos,face);
```

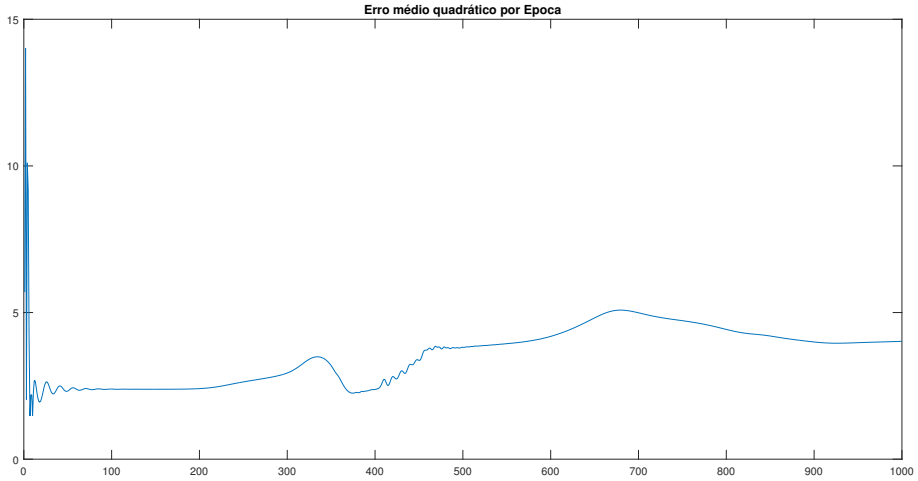
```
48 SaidaPlot=vec2mat(saidasValidacao,sqrt(length(saidasValidacao)));
49
50 %plot das solicitaes feitas pelo professor
51
52 %Erro Quadrticoporca;
53 figure();
54 plot(EMQ);
55 title('Erro mdio quadrtico por Epoca');
56
57
58 %Errodecompara;
59 erroComp=saidasValidacao-outValidacao;
60 figure();
61 plot(erroComp);
62 title('Erro de compara entre a fun real e a sada da MLP');
63
64 % G ficodecompara ;
65 %figure();
66 %hold on
67 %mesh(x,y,z);
68 %mesh(xValidacao,yValidacao,SaidaPlot);
69 %title('Compara entre fun real e sada MLP');
70 %hold off
71
72 % S a dadaredeneural ;
73 figure();
74 mesh(xValidacao,yValidacao,SaidaPlot);
75 title('Sada MLP');
76
77 % S a dadafunmatemca ;
78 figure();
79 mesh(xValidacao,yValidacao,zValidacao);
80 title('Fun Matemtica');
```

Plots Gerados pelo código:


Função Matemática


Saída MLP


Erro de comparação entre a função real e a saída da MLP



Para essa função, a MLP aproximou bem, percebemos que a função gerada é parecida com a função real matemática, com apenas algumas regiões que apresentam curvas menos “suaves".

4.5 função 5: f(x,y)= -(y+47).*sin(sqrt(abs(x./2+ y + 47))) - x.*sin(sqrt(abs(x- y - 47)))

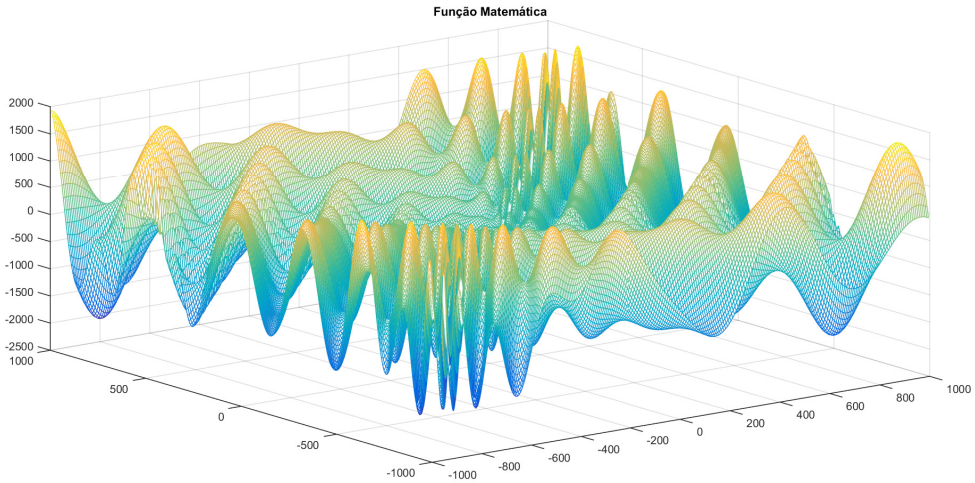
Script Feito para criar grupo de Treinamento e Validação, chamar as funções BackPropagation e MLPnetwork e plotar os resultados solicitados pelo professor:

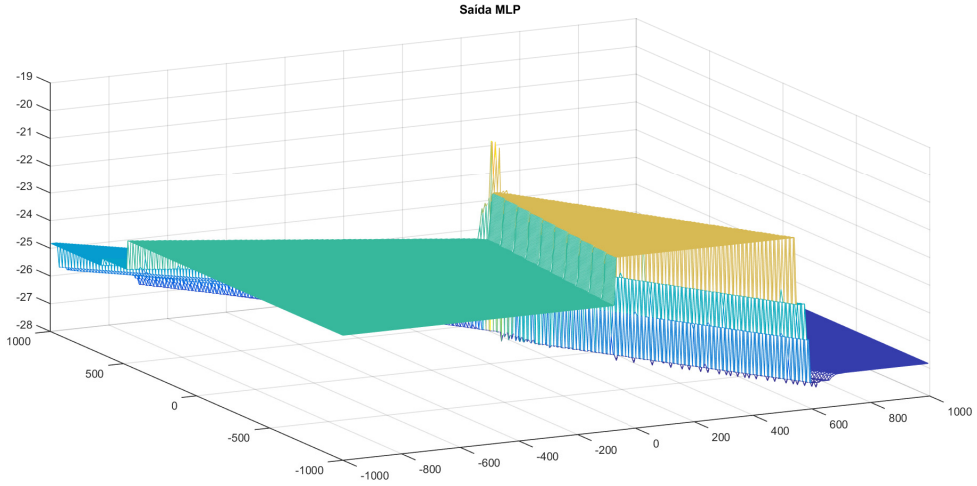
```
%Author: Vanessa Dantas de Souto Costa
%email: vanessa.dantas796@gmail.com

%Quinta  f u n  f(x,y)= -(y+47).*sin( sqrt(abs(x./2+ y + 47 )) ) - x.*
   sin(sqrt(abs(x- y - 47)));

%treinamento
passoTreinamento=100;

[x,y]=meshgrid(-1000:passoTreinamento:1000);
z = -(y+47).*sin( sqrt(abs(x./2+ y + 47 )) ) - x.*sin(sqrt(abs(x- y -
   47)));
in=[x(:)./max(max(abs(x(:)))) y(:)./max(max(abs(y(:))))];
out=z(:);

nce=4;
nuce=25;
TA=0.00005;

epMax=500;
emqTarget=1.7;
```

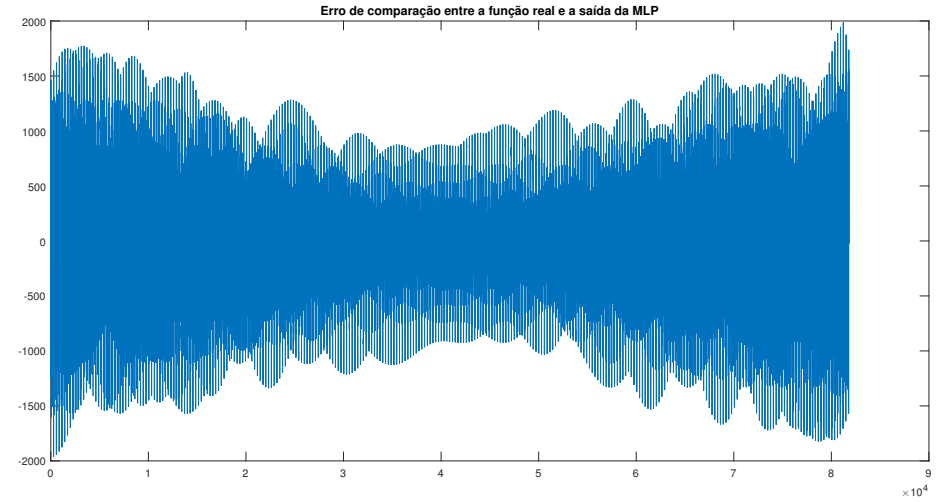
```
20 percentrein=0.7;
21 alphaMomento=0.9;
22 face='LOGSIGMOIDE'; % o p es  'LOGSIGMOIDE' e 'TANGENTESIGMOIDE'
23 ini=[];
24
25 %carregar  v a r i veis
26 %load('ini.mat')
27 load('SaidasBackPropagation5.mat');
28
29 [Pesos,AtivacoesNos,saidas,EMQ,Epoca]=BackPropagation(in,out,nce,nuce,
      TA,face,ini,epMax,emqTarget,percentrein,alphaMomento);
30 EMQ=EMQ(1:Epoca);
31
32 %salvar variaveis
33 save('SaidasBackPropagation5.mat','Pesos','AtivacoesNos','EMQ','saidas
      ','Epoca');
34
35 %plot da  s a da  do  treinamento
36 %figure();
37 %plot(saidas);
38
39 %validacao
40 passoValidacao=7;
41
42 [xValidacao,yValidacao]=meshgrid(-1000:passoValidacao:1000);
43 zValidacao=-(yValidacao+47).*sin( sqrt(abs(xValidacao./2+ yValidacao +
       47 )) ) - xValidacao.*sin(sqrt(abs(xValidacao- yValidacao - 47)));
44 inValidacao=[xValidacao(:) yValidacao(:)];
45 outValidacao=zValidacao(:);
46
47 [saidasValidacao]=MLPnetwork(inValidacao,AtivacoesNos,Pesos,face);
48 SaidaPlot=vec2mat(saidasValidacao,sqrt(length(saidasValidacao)));
49
50 %plot das  s o l i c i t a e s  feitas pelo professor
51
52 % E r r o Q u a d r ticoporca ;
53 figure();
54 plot(EMQ);
55 title('Erro  m dio  quadrtico  por Epoca');
56
57 % E r r o d e c o m p a r a ;
58 erroComp=saidasValidacao-outValidacao;
59 figure();
```

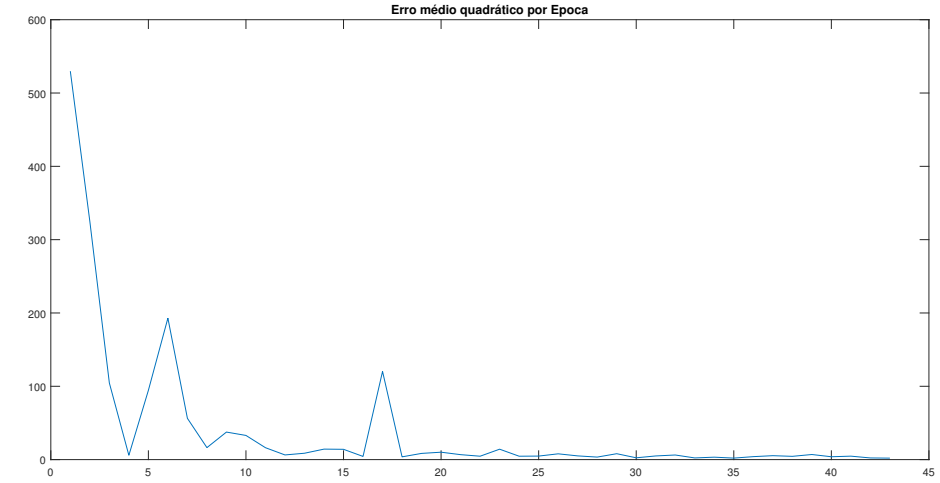
```
60 plot(erroComp);
61 title('Erro de compara  entre a fun  real e a sada da MLP');
62
63 % G r ficodecompara ;
64 %figure();
65 %hold on
66 %mesh(xValidacao,yValidacao,zValidacao);
67 %mesh(xValidacao,yValidacao,SaidaPlot);
68 %title('Compara  entre fun  real e sada MLP');
69 %hold off
70
71
72 % S a dadaredeneural ;
73 figure();
74 mesh(xValidacao,yValidacao,SaidaPlot);
75 title('Sada MLP');
76
77 % S a dadafunmatemca ;
78 figure();
79 mesh(xValidacao,yValidacao,zValidacao);
80 title('Fun  Matemtica');
```

Plots Gerados pelo código:









Para essa função, a MLP não obteve bom resultado, percebemos que a função gerada distoa bastante da função real matemática.

5 Conclusão

Perceptrons multicamadas usando um algoritmo backpropagation é o algoritmo padrão para qualquer aprendizado supervisionado, processo de reconhecimento de padrões e objeto de investigação em cursos de neurociência computacional e processamento distribuído em paralelo . Eles são úteis na pesquisa em termos de sua capacidade de resolver problemas estocásticos, que muitas vezes permite obter soluções aproximadas para extremamente complexos problemas como regular a tensão de entrada para o correto funcionamento de uma bomba d'água.

As dificuldades encontradas nesta atividade consistiram em criar um algoritmo rápido e eficiente, bem como em encontrar os melhores parâmetros para o treinamento. O algoritmo criado apresentou um desempenho aceitável para aproximar a maioria das funções matemáticas escolhidas.

Em virtude da importância da MLP, esperamos que a apresentação do código juntamente com as funções aproximadas tenha estimulado o aprendizado da mesma.

6 Referências

http://www.joinville.udesc.br/portal/professores/andretavares/materiais/RP_Aula12_MLP.pdf

http://conteudo.icmc.usp.br/pessoas/andre/research/neural/MLP.htm