UNIVERSIDADE FEDERAL DO CEARÁ

CURSO DE ENGENHARIA DE COMPUTAÇÃO

INTELIGÊNCIA COMPUTACIONAL APLICADA

PROFESSOR MÁRCIO AMORA

REDE PERCEPTRON MULTICAMADAS PARA APROXIMAÇÃO DE FUNÇÃO

ARTHUR SOUSA DE SENA MATRÍCULA: 345750

VICTÓRIA TOMÉ OLIVEIRA MATRÍCULA: 366333

VITORIA ALMEIDA BERTAIA MATRICULA: 356741

SOBRAL – CE

21/06/2016

SUMÁRIO

1. INTRODUÇÃO

Uma rede neural artificial (RNA) é um modelo computacional baseado no cérebro, inclusive no seu comportamento, ou seja, errando, aprendendo e fazendo descobertas. Inspirado em modelar e processar informações as redes neurais desta forma adquirem conhecimentos através de suas experiências. A rede neural refere-se a um conjunto de unidades de processamentos simples que se comunicam entre si, enviando sinais através de conexões ponderadas. Atualmente, as redes neurais são utilizadas para resolver uma grande variedade de problemas que são de difícil resolução.

Uma das características mais importantes de uma rede neural é sua capacidade de aprendizado e de generalização. Conforme sua experiência as redes neurais conseguem se adaptar e melhorar seu desempenho, essa adaptação ocorre através do processo de treinamento do seu algoritmo de aprendizagem, que se dá pelo meio do fornecimento do conjunto de regras, padrões e pelo seu teorema de convergência que garante que a rede será treinada e que seus números de interações sejam finitos.

Uma rede Perceptron consiste em uma rede neural composta por apenas uma camada de neurônios que processa informações baseadas em redes neurais e é construído com neurônios artificiais.

A propriedade mais importante de uma Perceptron é sua habilidade de aprender a partir de seu ambiente de funcionamento e a partir disso eleva o índice do seu desempenho. O aprendizado de uma rede Perceptron se dá pelo seu treinamento, elas aprendem padrões e as classificações desses padrões com aprendizado supervisionado. O Perceptron é formado por uma única camada de neurônios com pesos sinápticos e bias ajustáveis, como podemos observar na Figura 1.

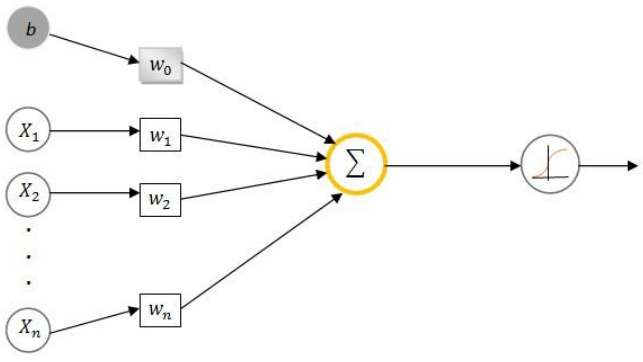

Figura 1. Perceptron

O Perceptron de Camada Simples foi à primeira rede Perceptron criada e é uma rede que pode ser usada com entradas binárias e bipolares. Uma das limitações da Perceptron de camada simples é que ela só pode resolver problemas linearmente separáveis, como mostrado na Figura 2.

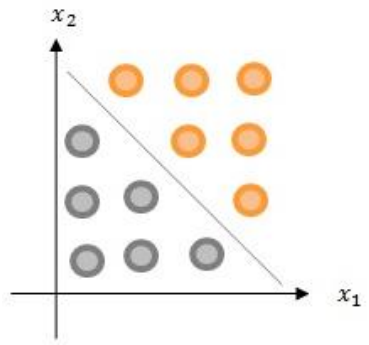

Figura 2. Gráfico Perceptron de Camada Simples

Minsky e Papert analisaram matematicamente o Perceptron e demonstraram que redes de uma camada não são capazes de solucionar problemas que não sejam linearmente separáveis. Como desacreditavam da possibilidade de implementação de um método de treinamento para redes de multiplas camadas, eles concluíram que as redes neurais sempre teria essa limitação.

Segundo Minsky e Papert:

> "O perceptron mostrou-se merecedor de estudo apesar de (e mesmo por causa de) suas severas limitações. Ele tem muitas características que atraem a atenção: sua linearidade; seu teorema de aprendizagem intrigante; sua clara simplicidade paradigmática como uma forma de computação paralela. Não há razão para se supor que qualquer uma dessas virtudes persista na versão de múltiplas camadas. Apesar disso, consideramos que é um importante problema a ser pesquisado para elucidar (ou rejeitar) nosso julgamento intuitivo de que a sua extensão para sistemas de múltiplas camadas é estéril." (Perceptrons de Misky e Papert (1969) – Seção 13.2)

Contudo, a observação de Misky e Papert se mostrou infundada, pois na atualidade as redes neurais avançaram de uma forma que conseguem resolver problemas não lineares por Perceptron de múltiplas camadas, treinados com o algoritmo de retropropagação.

O desenvolvimento do algoritmo de treinamento backpropagation, por Rumelhart, Hinton e Williams em 1986, mostrou que é possível treinar eficientemente redes de múltiplas

camadas, resultando no modelo de Redes Neurais Artificiais mais utilizado atualmente, às redes Perceptron Multi-Camadas (MLP), treinadas com o algoritmo backpropagation.

O Perceptron de Múltipas Camadas é uma rede que cada camada se conecta à próxima camada, porém não há caminho de volta. Os problemas que não são linearmente separáveis, como o exemplo mostrado na Figura 3, podem ser revolvidos através dessa rede. Essa conexão ocorre com um ou mais camadas entre os nós de entrada e de saída. Essas camadas intermediárias são chamadas de camadas ocultas, abrangem nós que estão diretamente conectados aos nós de entrada e saída. É a arquitetura de rede neural mais utilizada hoje em dia.

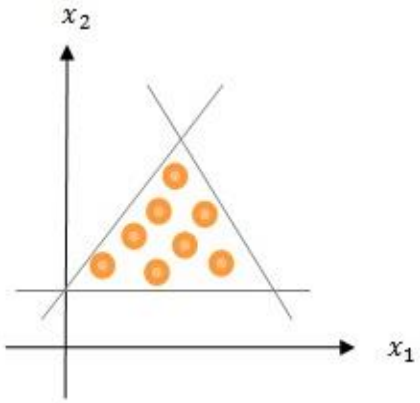

Figura 3. Gráfico Perceptron de Várias Camadas

Uma possível aplicação é a aproximação de funções não-lineares, que é o objetivo deste trabalho. O resultado esperado desse trabalho é obter uma aproximação da função $z = x^2 + y^2$. Para se chegar ao objetivo desejado, será utilizado o software MATLAB para implementar tal modelo.

2. REDE PERCEPTRON MULTICAMADAS

As perceptrons multicamadas (MLP) são as redes neurais mais utilizadas atualmente. Elas são semelhantes ao perceptron simples, tendo sua maior diferença na quantidade de camadas de neurônios. Elas pertencem a classe de redes neurais feedforward, que são capazes de aproximar classes genéricas de funções, incluindo funções contínuas e integráveis. Em uma rede MLP, os neurônios são agrupados em camadas. A primeira camada é chamada de camada de entrada, a última de camada de saída, e as remanescentes são as camada internas, ou intermediárias. Qualquer mapeamento pode ser realizado utilizando no máximo duas camadas intermediárias e apenas uma camada para aproximar qualquer função contínua.

As camadas intermediárias de uma MLP possuem função de ativação não linear, geralmente uma sigmóide definida por: $\frac{1}{1+e^{-t}}$. Além disso, é atribuido um peso w a cada sinal de entrada, somado a um bias b. A função de ativação da camada de saída geralmente é uma função linear. Um modelo genérico de rede MLP é mostrado na Figura 4.

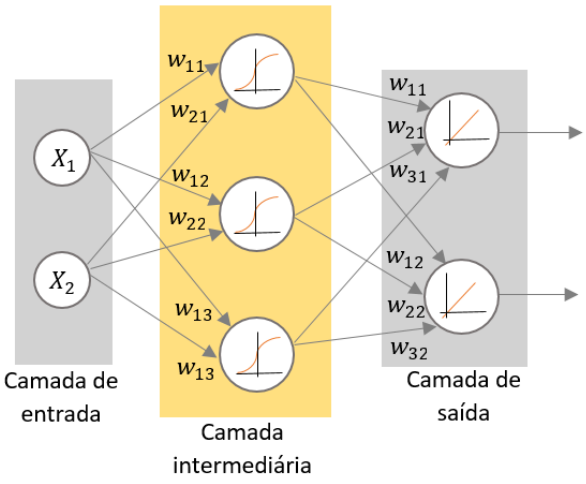

Figura 4. Modelo genérico de uma rede Perceptron Multicamadas

Apesar da MLP apresentar resolução para funções não linearmente separáveis, é necessário um algoritmo de treinamento capaz de definir de forma automática os pesos. O algoritmo é denominado a regra de delta de Windrow & Hoff para o treinamento da Adaline, e é denominado Backpropagation.

O Backpropagation é um método utilizado em redes neurais artificiais para treinar redes Perceptrons de múltiplas camadas. O Backpropagation utiliza o método de aprendizagem

supervisionada, ou seja, a aprendizagem de máquina que deduz uma função a partir de dados de treinamento marcados. Um algoritmo de aprendizagem supervisionada analisa os dados de treinamento e produz uma função que melhor mapeia o conjunto de entradas para a saída mais correta possível, que por sua vez pode ser utilizado para o mapeamento de novos exemplos.

O Backpropagation consiste em duas fases:

- Propagação Positiva do Sinal: Os pesos são mantidos fixos durante o processo;
- Retropropagação do erro: Os pesos são ajustados baseados na regra de correção do erro durante o processo.

O Backpropagation possui a programação do erro calculada no sentindo inverso do sinal de entrada, desta forma o algoritmo é denominado de retropropagação do erro.

O backpropagation procura o mínimo da função de erro utilizando o método do gradiente. A combinação de pesos que minimiza a função de erro é considerada a solução para o problema de aprendizagem. Como o cálculo do gradiente é realizado em cada iteração, a função de erro precisa ser contínua e diferenciável. Devido a esse motivo, a função de ativação mais popular utilizada nas camadas intermediárias é a sigmóide, que é contínua e diferenciável. Uma limitação encontrada na utilização do algoritmo backpropagation é o tempo de treinamento da rede, que dependendo da complexidade pode tornar o processo muito lento.

A regra Delta é utilizada para ajustar os pesos e bias da rede, assim minimizando o erro entre a saída da rede simulada e a saída da rede real desejada. É calculado a gradiente da função de erro em relação aos pesos e bias, no algoritmo os pesos e o bias são atualizados na direção oposta, ou seja, em retropropagação para minimizar o erro. Contudo o erro somente é calculado para a camada de saída, então o Backpropagation se responsabiliza em determinar a influência do erro nas camadas intermediárias da rede. A atualização dos pesos e bias obedece as equações abaixo:

$$w_{ij}^{m}(t+1) = w_{ij}^{m}(t) - \alpha \frac{\partial \Im(t)}{\partial w_{ij}^{m}}, \qquad b_{i}^{m}(t+1) = b_{i}^{m}(t) - \alpha \frac{\partial \Im(t)}{\partial b_{i}^{m}}$$

3. DESCRIÇÃO DO PROBLEMA

A Perceptron de múltiplas camadas tem como uma das suas principais aplicações a aproximação de funções não lineares. As redes Perceptron utilizam algoritmo de retro-propagação do erro, o back-propagation, que é baseado na regra de aprendizado de correção de erro.

O objetivo do trabalho é implementar esse algorítimo de treinamento para aproximar a função real não-linear, dada por: $z = x^2 + y^2$. Sua representação gráfica pode ser vista na Figura 5.

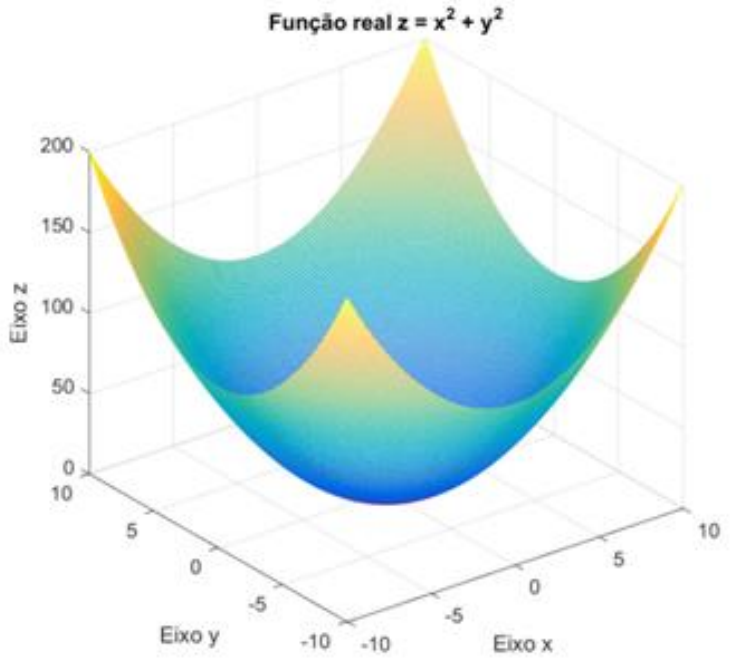

Figura 5. Função real a ser aproximada.

Para se chegar ao objetivo almejado, simplesmente será utilizada uma rede neural com duas camadas: intermediária e saída. A função de ativação da camada intermediária será uma função não-linear e da camada de saída uma função linear. A topologia da rede utilizada para resolver tal problema é mostrado na Figura 6.

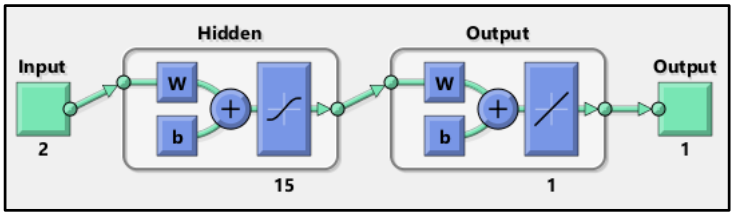

Figura 6. Diagrama da rede MLP utilizada para a resolução do problema.

Os seguintes parâmetros serão utilizados no algorítimo de treinamento:

- Número de amostras para treinamento = 200;
- Função de ativação da camada intermediária = tansig = $\frac{2}{1+e^{-2n}} - 1$;
- Função de ativação da camada de saída = purelin (Função linear);
- Número de neurôneos da camada interna = 15;
- Número máximo de iterações = 3000;
- Tolerância de erro = 1 e -8;
- Tempo máximo de treinamento: 15s;
- Número máximo de iterações com aumento do erro de validação: 10.

4. IMPLEMENTAÇÃO EM MATLAB

A implementação do problema se deu através da toolbox feedforwardnet do Matlab. Para o treinamento da rede são utilizados dois vetores, cada um com 200 números aleatórios no intervalo [-10,10]. Foi criada uma rede neural com 15 neurôneos na camada intermediária, tal número foi o que proporcionou melhores resultados nos testes realizados. Abaixo pode-se ver o código em Matlab da implementação do problema:

```
%% PROBLEMA ================================================================
%   CRIAR UMA REDE MLP PARA APROXIMAR A FUNCAO z=x^2+y^2
%==========================================================================

%% Inicialização da rede ================================================

clc;

limI = -10; % Limite inferior do vetor de valores de treinamento
limS = 10; % Limite superior do vetor de valores de treinamento

% 200 valores aleatórios dentro do intervalo [-10,10] são gerados para treinar a rede
x = (limI + (limS-limI)*rand(200,1))'; % Vetor de treinamento x
y = (limI + (limS-limI)*rand(200,1))'; % Vetor de treinamento y
z = x.^2 + y.^2; % Funcao real

numNeuros = 15; % Número de neurôneos da camada interna da rede
zMLP = feedforwardnet(numNeuros); % Rede MLP para aproximacao de funcao é criada
zMLP.layers{1}.transferFcn = 'tansig'; % A função de ativação da primeira camada é definida como tangente hiperbólica sigmóide
zMLP.layers{2}.transferFcn = 'purelin'; % A função de ativação da camada de saída é definida como linear.
zMLP.trainParam.lr = 10e5; % Indice de aprendizagem

% Critérios de parada
zMLP.trainParam.epochs = 3000;  % Numero maximo de iteracoes.
zMLP.trainParam.goal = 1e-8; % Tolerancia de erro.
zMLP.trainParam.time = 15; % Tempo máximo de treinamento.
zMLP.trainParam.max_fail = 10; % Número máximo de erros de validação, utilizado para parada antecipada.

inputs = [x;y]; % Vetores x e y sao definidos como entrada da rede MLP
targets = z; % z é definida como a funcao alvo (desejada).

% Treinamento da rede
[zMLP,tr] = train(zMLP, inputs, targets);


%% Trecho responsável por plotar os gráficos ============================

d=limI:0.1:limS; % Intervalo utilizado para plotar a função
[X, Y]=meshgrid(d,d); % Faz um meshgrid do intervalo

% Mesh da função real ------------------------------------------------------
f = figure('Name','Funções real e aproximada','NumberTitle','off','position',[0 0 1200 480]);
movegui(f,'center');
subplot(1,2,1);
Z = griddata(x,y,z,X,Y,'v4');
mesh(X,Y,Z)
xlabel('Eixo x');
ylabel('Eixo y');
zlabel('Eixo z');
title('Função real z = x^2 + y^2');
zlim([0 200]);

% Mesh da função aproximada ------------------------------------------------
subplot(1,2,2);
a= sim(zMLP,inputs);
```

```
A = griddata(x,y,a,X,Y,'v4');
mesh(X,Y,A)
xlabel('Eixo x');
ylabel('Eixo y');
zlabel('Eixo z');
title('Função aproximada zMLP');
zlim([0 200]);


% Mesh de erro ------------------------------------------------------------
f = figure('Name','Gráfico de Erro','NumberTitle','off','position',[0 0 800 480]);
movegui(f,'center');
AZ = griddata(x,y,z-a,X,Y,'v4');
mesh(X,Y,AZ)
colormap hsv
colorbar
xlabel('Eixo x');
ylabel('Eixo y');
zlabel('Erro');
title('Superfície de erro')

% Evolução do erro quadrático médio durante o treinamento -----------------

figure, plotperform(tr);
ylabel('Erro Quadrático Médio');

% Gráfico de regressão linear ---------------------------------------------
figure, plotregression(targets,a);
title('Regressão');
xlabel('Alvo');

% Mostra o diagrama da rede -----------------------------------------------
view(zMLP);


%% Trecho mostra relatório com resultado dos pesos e teste da rede ========

display(sprintf('---------RESULTADOS---------\n\n'));

% Pesos e bias ------------------------------------------------------------
display(sprintf('Pesos da camada interna\n\n    w_k1      w_k2       b_k'));
display(sprintf('%10f %10f %10f\n',zMLP.IW{1}(:,1),zMLP.IW{1}(:,2),zMLP.b{1}));
display(sprintf('\nPesos do neurôneo de saída\n\n    w_k'));
display(sprintf('%10f\n',zMLP.LW{2,1}'));
display(sprintf('\n     b'));
display(sprintf('%10f\n',zMLP.b{2}));

% Teste da rede -----------------------------------------------------------
fprintf('\n\nTeste da rede para valores não treinados\n');
zr = @(a,b) a^2 + b^2; % Funcao real

% Dois vetores com 10 valores aleatórios são criados para testar
% a rede depois de treinada.
x1 = (limI + (limS-limI)*rand(20,1))'; % Vetor de teste x1
x2 = (limI + (limS-limI)*rand(20,1))'; % Vetor de teste x2

table(20,5)=ones();

%Imprime os resultados em formato de tabela
for i=1:20
   table(i,1)=x1(i);
   table(i,2)=x2(i);
   table(i,3)=zr(x1(i),x2(i));
   table(i,4)=zMLP([x1(i);x2(i)]);
   table(i,5)=abs(table(i,3)-table(i,4))*100;
end

fprintf('\n   x       y      z(x,y)    zMLP(x,y)  Erro');
fprintf('\n -----    -----    --------  ----------  -----');
```

```
for i=1:20
   fprintf('\n');
   for j=1:5
      if j>2&&j<5
         fprintf('%8.3f   ',table(i,j));
      else if j==5
            fprintf('%5.3f%%   ',table(i,j));
         else
            fprintf('%6.3f   ',table(i,j));
         end

      end

   end
end
fprintf('\n -----    -----    --------  ----------  -----\n\n');
```

5. RESULTADOS

A fins de comparação, serão exibidos resultados de dois treinamentos. No primeiro treinamento, o critério de parada atingido foi o limite máximo de iterações, no caso 3000 iterações. Já no segundo experimento, o critério de parada obedecido foi a tolerancia de erro, atingindo um valor menor que 1×10^{-8}. No final do processo de cada treinamento é exibido um relátorio com os valores finais dos pesos e bias de cada camada. Para verificar a eficácia da rede treinada, é exibido uma comparação entre os resultados da função real e os obtidos pela MLP para vinte valores aleatórios diferentes daqueles utilizados no treinamento. Nas Figuras 7 e 8, são exibidos os relatórios do treinamento 1 e 2 respectivamente.

Pesos da camada interna

w_k1	w_k2	b_k
-0.974810	7.627722	9.947337
0.007658	0.707132	0.005790
-20.800211	0.002937	-6.925088
0.003253	-14.894469	-15.217539
0.507243	22.839565	-0.006215
0.003305	2.776740	-2.823289
1.627081	-0.002427	1.679545
-9.044171	0.905285	-3.616276
0.957073	-0.177304	12.714739
-0.001727	-33.297715	-1.822036
1.880728	-7.333013	-6.526862
0.359480	-0.622964	-2.733514
26.327536	-1.034205	-2.480402
1.070849	-7.139922	-9.704534
1.307442	-12.782066	-2.899406

Pesos do neurôneo de saída

w_k	b
-3.002825	16.091871
-0.000009	
0.000001	
-0.024349	
1.975241	
0.966182	
-0.000029	
2.075959	
-0.000002	
-2.015855	
0.000034	
0.000002	
-9.403313	
-0.000001	
0.862819	

Teste da rede para valores não treinados

x	y	z(x,y)	zMLP(x,y)	Erro
-7.592	7.685	116.697	116.696	0.050%
3.210	3.034	19.509	19.509	0.015%
-3.241	4.370	29.600	29.600	0.008%
6.328	-6.023	76.316	76.317	0.067%
5.886	-2.454	40.671	40.671	0.045%
-6.894	-6.749	93.070	93.070	0.072%
7.642	4.720	80.683	80.683	0.009%
-6.708	3.136	54.839	54.838	0.039%
8.055	8.703	140.624	140.624	0.043%
-0.389	1.626	2.796	2.796	0.023%
9.385	0.396	88.230	88.230	0.018%
5.440	2.663	36.684	36.684	0.023%
0.922	8.694	76.429	76.429	0.016%
-9.535	1.056	92.039	92.039	0.011%
-1.873	-5.831	37.513	37.514	0.057%
9.352	-0.840	88.169	88.170	0.055%
7.853	-2.650	68.699	68.698	0.023%
-2.292	0.889	6.042	6.042	0.021%
-6.332	-1.167	41.454	41.454	0.048%
-8.198	-1.277	68.836	68.836	0.014%

Figura 7. Resultados do primeiro treinamento.

Pesos da camada interna

w_k1	w_k2	b_k
5.878699	12.258238	0.053125
-0.032496	-0.020096	-4.659934
-5.835705	-0.036617	-3.359720
-2.495438	-1.237283	-0.843717
0.140312	3.626786	-0.580257
3.131241	-13.611527	-1.588006
0.971266	0.600859	-0.146068
-4.007532	1.094583	-0.109482
-0.082436	-0.040860	-0.027847
-3.695521	-7.633866	-0.019169
-8.035889	-13.765606	-2.645971
1.040070	-1.380103	2.040925
0.718523	0.199812	0.655044
-0.237914	-2.180638	-0.762844
2.256584	7.801323	1.354393

Pesos do neurôneo de saída

w_k	b
-0.000007	16.119832
0.000000	
1.127851	
-1.643871	
6.486944	
0.000308	
0.000000	
-0.215764	
0.001571	
0.004065	
1.869438	
1.635192	
-0.000331	
0.000000	
-6.966791	

Teste da rede para valores não treinados

x	y	z(x,y)	zMLP(x,y)	Erro
-6.160	-4.457	57.811	57.811	0.003%
-2.524	7.469	62.156	62.156	0.029%
-4.817	2.513	29.523	29.523	0.002%
0.999	0.231	1.051	1.051	0.004%
9.590	7.427	147.133	147.132	0.059%
-3.325	-2.677	18.224	18.225	0.016%
-4.689	8.141	88.271	88.271	0.045%
-2.283	-9.754	100.343	100.343	0.011%
-8.270	-8.437	139.589	139.589	0.012%
3.149	-2.114	14.384	14.384	0.016%
3.352	1.782	14.413	14.413	0.019%
-7.125	1.572	53.240	53.240	0.004%
8.050	9.474	154.552	154.553	0.064%
-0.205	4.067	16.586	16.586	0.027%
-4.601	7.947	84.326	84.325	0.050%
6.745	1.633	48.159	48.159	0.010%
-4.730	-7.722	82.005	82.005	0.007%
-2.451	4.907	30.087	30.087	0.006%
-7.037	8.408	120.224	120.224	0.020%
-0.330	5.619	31.686	31.686	0.011%

Figura 8. Resultados do segundo treinamento.

Verifica-se através dos relatórios que os resultados obtidos pela rede MLP são muito próximos aos resultados da função real, com erro máximo de 0,072% no primeiro treinamento e 0,064% no segundo. Percebe-se que os erros do segundo treinamento são em geral menores, o que realmente era esperado, já que o erro quadrático médio do segundo é bem menor que o primeiro. Através destes testes, verifica-se o poder de generalização da perceptron multicamadas, pois mesmo para valores não treinados, a rede exibe um desempenho muito satisfatório, proporcionando erros muito baixos. Além dos relatórios mostrados, é plotado uma série de gráficos para se ter uma melhor vizualização da eficácia do processo de treinamento da rede.

Os gráficos da função real e da rede MLP do primeiro treinamento são exibidos na Figura 9. Através da análise dos gráficos, percebe-se com clareza que o processo de aprendizagem foi eficaz e gerou uma aproximação bastante fiel à função original.

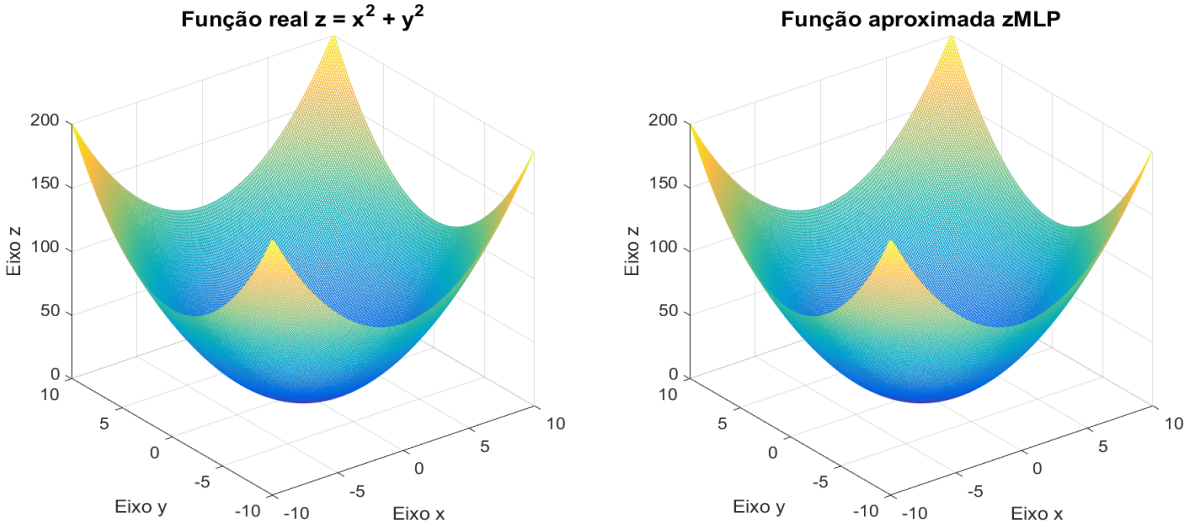

Figura 9. Função real e função aproximada.

Para se ter uma melhor noção do quão próxima a função aproximada ficou da função real, na Figura 10 é mostrado o gráfico de regressão linear da rede. Como pode-se ver, a linha de tendência da função aproximada, representada pela linha azul é muito próxima da função real, em pontilhado. Como percebe-se, a distância entre as retas é de aproximadamente 0,00005.

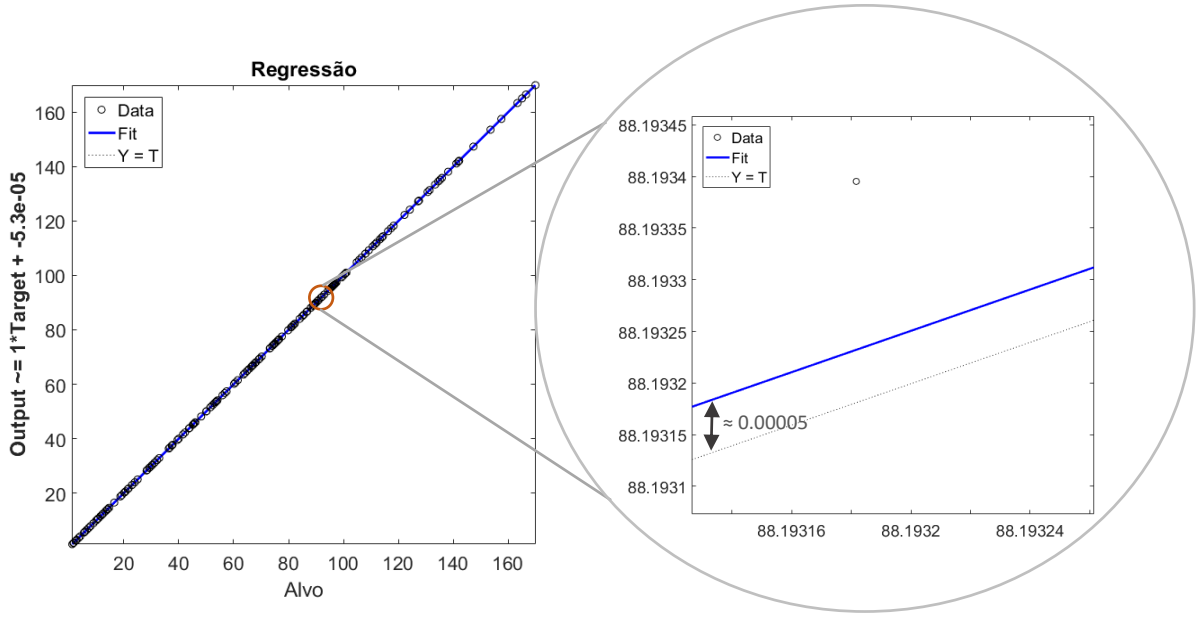

Figura 10. Regressão linear da rede MLP.

Nas Figuras 11 e 12, encontram-se os gráficos de evolução do erro quadrático médio durante o processo dos dois treinamentos. O Matlab utiliza a técnica de parada antecipada para

evitar o overfitting da rede. Para implementar tal técnica o software separa os dados de treinamento em três subconjuntos, como é visto no gráfico. O subconjunto de validação, curva em verde, é o subconjunto utilizado para realizar a parada antecipada. Geralmente, quando está acontecendo overfitting, o erro quadrático médio do conjunto de validação tende a aumentar, assim, se o erro aumentar durante 10 iterações, no caso dessa implementação, a parada antecipada acontece.

Como percebe-se, nos dois treinamentos, o erro começa muito alto, mas já nas primeiras iterações, ele cai bruscamente e continua diminuindo ao longo do processo, até que algum dos critérios de parada sejam atendidos. No primeiro treinamento é atingido o limite de 3000 iterações e no segundo treinamento o erro chega a $9{,}984 \times 10^{-9}$, parando com 2385 iterações.

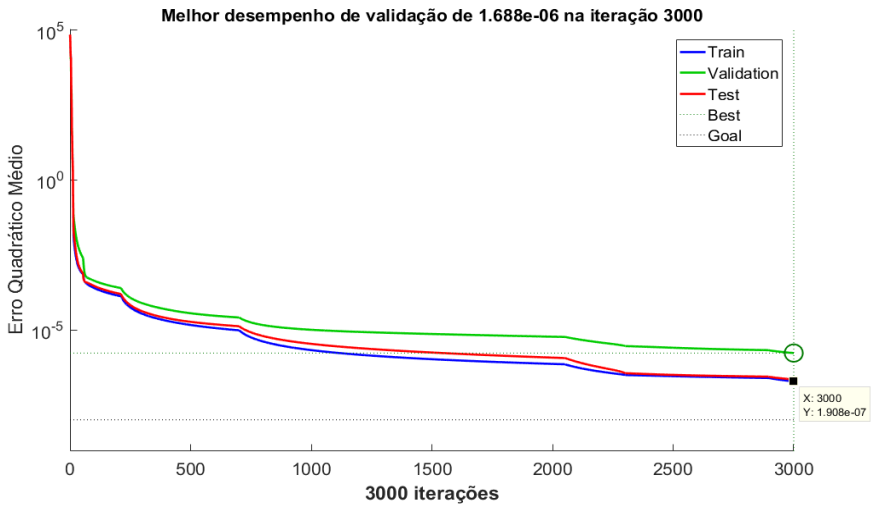

Figura 11. Erro quadrático médio durante o treinamento 1.

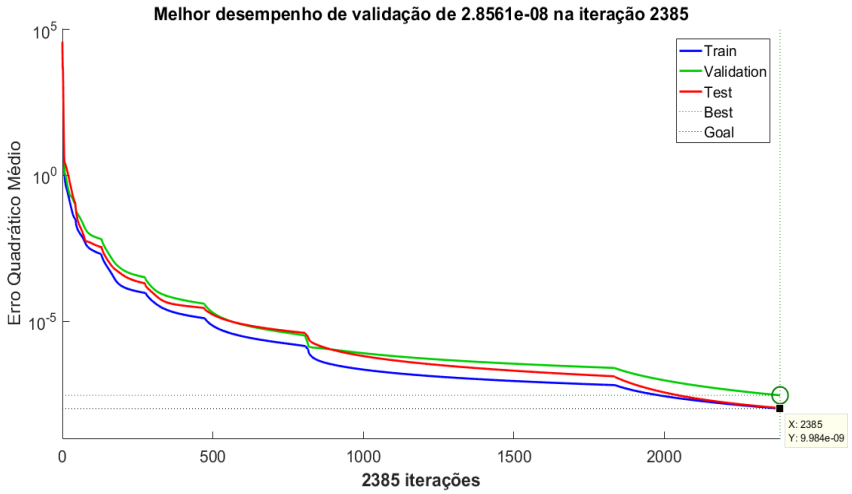

Figura 12. Erro quadrático médio durante o treinamento 2.

Por último, as superfícies de erro da rede MLP treinada são mostradas nas Figuras 13 e 14. Para uma melhor interpretação, é mostrado duas vistas do gráfico em cada figura. Através da análise de tais vistas é possível ter uma visão geral do desempenho da rede. Percebe-se que a escala de erro do segundo treinamento é menor do que o primeiro, o que já se sabia, pois o erro quadrático médio final do segundo treinamento é bem menor.

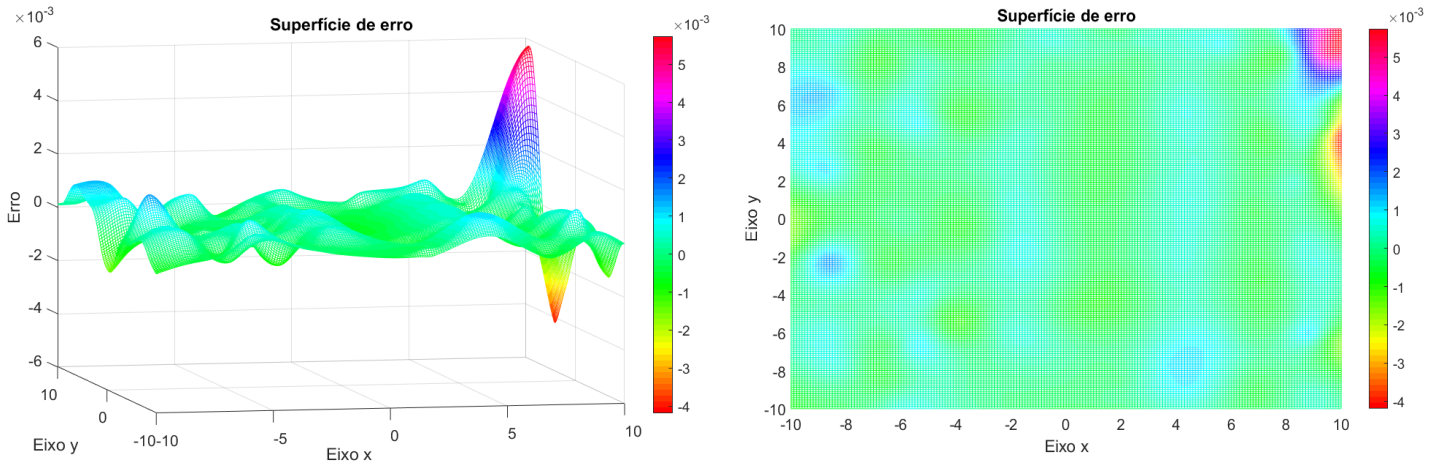

Figura 13. Superfície de erro da rede MLP do treinamento 1.

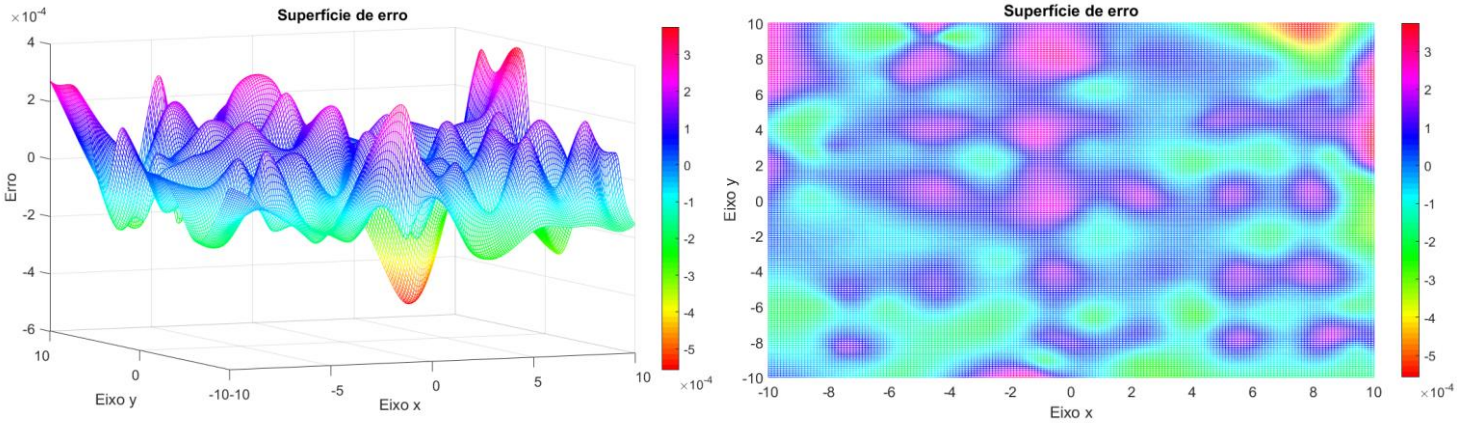

Figura 14. Superfície de erro da rede MLP do treinamento 2.

No gráfico de erro do primeiro treinamento percebe-se que o erro é muito próximo a zero na maioria da região mostrada, variando de -1×10^{-3} a 1×10^{-3}. O erro é levemente maior em apenas dois pontos do gráfico, onde há picos, próximo as coordenadas (10,10) e (10,3). Já no segundo treinamento, o gráfico é um pouco mais irregular, contudo, a escala de erro é menor, variando de -2×10^{-4} a 2×10^{-4} na maior parte da área plotada, apresentando apenas um pico onde o erro chega a 5×10^{-4}, próximo à coordenada (8,10).

6. CONCLUSÃO

Através do trabalho implementado, vimos a eficiência das redes Perceptrons Multicamadas para a aproximação de funções. Comprovamos que apenas uma camada intermediária foi necessário paraaproximar a função dada, o que obedece a teoria. Nos dois treinamentos realizados, os erros obtidos foram muito baixos, chegando ao máximo em 0,072%. Cada treinamento utilizou um critério de parada distinto, no primeiro, o erro alvo não foi atingido, mas mesmo dessa forma, no fim das 3000 iterações o erro obtido foi satisfatório. No segundo treinamento, os resultados foram ainda melhores, mostrando um desenpenho melhor que o primeiro, pois neste caso, a tolerância de erro foi satisfeita.

Através dos relatórios exibitos, e dos gráficos plotados pudemos ter uma visão detalhada do comportamento da rede MLP. Pôde-se verificar os pesos e bias resultantes no final de cada treinamento e os testes da rede para amostras em que não foi treinada. O gráfico do erro quadrático médio mostrou o comportamento da rede ao longo do treinamento, mostrando a rede aumentar seu desenpenho ao passo que as iterações aumentam. A regressão linear mostrou quão bem a função foi aproximada, mostrando que a MLP difere estremamente pouco da função real. E através da superfície de erro, foi possível mapear a função treinada e detectar as regiões mais eficientes. De maneira geral, a implementação foi realizada com sucesso. Sendo assim, podemos concluir que todos os resultados foram satisfatórios e atenderam os objetivos trabalho.

REFERÊNCIAS

KROSE, Ben; SMAGT, Patrick. *An Introduction to Neural Networks*. 8. ed. Amsterdam: University of Amsterdam, 1996.

ROJAS, Raul. *Neural Networks:* A Systematic Introduction. Berlin: Springer-Verlag, 1996.

HAYKIN, Simon. *Neural Networks:* a comprehensive foundation. 2. ed. Singapore: Pearson Education, 2001.

MINSKY, Marvin; PAPERT, Seymour. *Perceptrons: an introduction to computational geometry*. Expanded Edition. 1987.

Neural Networks Structures. Disponível em <http://www.ieee.cz/knihovna/Zhang/Zhang100-ch03.pdf>. Acesso em 12 de Junho de 2016.

Learning in Multi-Layer Perceptrons: Back-Propagation. Disponível em <http://www.cs.bham.ac.uk/~jxb/INC/l7.pdf>. Acesso em 18 de Junho de 2016.

The Backpropagation Algorithm. Disponível em < https://page.mi.fuberlin.de/rojas/ neural/chapter/K7.pdf>. Acesso em 18 de Junho de 2016.

.