UNIVERSIDADE
FEDERAL DO CEARÁ

ENGENHARIA DA COMPUTAÇÃO

INTELIGÊNCIA COMPUTACIONAL

STEFANE ADNA DOS SANTOS - 403249

04 DE OUTUBRO DE 2020
SOBRAL - CE

1. Introdução

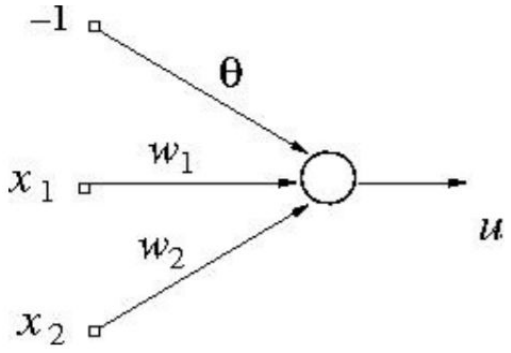
As redes neurais são modelos de computação que são inspirados no sistema nervoso de um animal. Essas redes são capazes de aprender e realizar tarefas como reconhecer determinados tipos de padrões e fazer modelagem de dados complexos. Existem vários tipos de redes neurais, para este trabalho, vamos estudar as redes Perceptron e RBF.

O Perceptron é uma rede que recebe vários valores de entrada e apenas uma única saída. É uma rede bem simples, com apenas uma camada, sendo muito utilizada para demonstrar como funcionam neurônios simples.

A rede RBF é mais complexa, possuindo duas camadas de neurônios, onde os neurônios da primeira camada utilizam funções de ativação não lineares e os neurônios da segunda camada utilizam funções de ativação lineares.

Um exemplo de neurônio artificial simples pode ser visualizado na Figura 01, onde os valores de x1 e x2 são as entradas, Θ é o limiar, w1 e w2 são os pesos sinápticos e u é a função de ativação.

Figura 01: Neurônio artificial simples



2. Rede Perceptron

Para este trabalho, será implementado um neurônio perceptron com valores de pesos inicialmente aleatórios que irão ter como entrada valores binários e deverão resolver o problema da porta AND de duas variáveis. A Tabela 01 exibe a tabela verdade referente a porta AND.

Tabela 01: Tabela verdade da porta AND

x1	x2	S
0	0	0
0	1	0
1	0	0
1	1	1

Inicialmente, deve-se declarar as variaveis de entrada e saída da tabela verdade no código. Como pode ser visualizado na Figura 02, onde a variável “entrada” possui três

colunas, sendo a primeira coluna o valor do bias, a segunda o valor de x1 e a terceira o valor de x2. A variável “saida” possui os valores de saída da tabela verdade.

Figura 02: Declarando as variaveis de “entrada” e “saida”

```
//Define os valores do bias e da tabela verdade
entrada = [-1 0 0
           -1 0 1
           -1 1 0
           -1 1 1];

//define os valores de saida da tabela verdade
saida = [0
         0
         0
         1];
```

Após isso, deve-se definir os pesos da rede de forma aleatória, para isso foi utilizada a função rand do scilab. A Figura 03 exibe o trecho de código, onde três pesos são escolhidos aleatoriamente e associados a um vetor wt.

Figura 03: Define os pesos aleatoriamente

```
//escolhe os pesos aleatoriamente
wt = [];
for i = 1:3
    wt(i)=rand();
end
```

Este algoritmo terá um funcionamento simples, possuindo um laço de repetição que deverá se repetir pelo total de épocas definido pelo usuário, ou até o erro de predição ter sido nulo durante oito vezes consecutivas. Em cada repetição desse laço, o algoritmo irá fazer a predição utilizando os dados de entrada e saída de uma linha da tabela verdade. Para realizar a predição serão utilizadas as equações passadas pelo professor durante as web-conferências.

No laço de repetição, a variável “xt” receberá os valores x1 e x2 de entrada de uma linha da tabela verdade, e a variável dt receberá o valor de saída referente às entradas da tabela verdade. Após isso, a variável “ut” receberá o produto escalar entre o vetor wt e xt, como pode ser demonstrado na Equação 01.

$$\mathbf{u(t) = w0x0 + w1x1 + ... + wpxp} \quad (01)$$

De acordo com o slide passado pelo professor, a saída “yt”, que terá o valor da predição da rede neural, terá o valor igual a 1 se u(t) for maior ou igual do que 0 e terá valor igual a 0 se a função de ativação u(t) for menor do que zero. A Figura 04 exibe o trecho de código referente ao cálculo da saída yt.

Figura 04: Cálculo da saída yt

```
xt = entrada(linha, 1:3);
dt = saida(linha,1);

//somatorio
ut = xt*wt;
if ut<0 then
    yt = 0;
else
    yt = 1;
end
```

O erro da rede neural deve ser calculado de acordo com a Equação 02, onde d(t) é a saída esperada e y(t) é a saída da rede neural.

$$e(t) = d(t) - y(t) \quad (02)$$

Para o cálculo de ajustes dos pesos foi utilizada a Equação 03, onde 'n' é o valor do passo de aprendizagem, que deve ter valores entre o intervalo de 0 e 1.

$$w(t+1) = w(t) + n*e(t)x(t) \quad (03)$$

A Figura 05 exibe o cálculo do erro e dos pesos da próxima interação.

Figura 05: Cálculo do erro e ajuste dos pesos

```
//define o erro
erro = dt - yt;

//define os pesos da proxima interação
for j= 1:3
    wt(j) = wt(j)+0.4*erro*xt(j);
end
```

A Figura 06 exibe um critério de parada que foi implementado no código, se a rede fizer 8 predições corretas consecutivas, então significará que a rede já aprendeu, logo o treinamento será encerrado.

Figura 06: Critério de Parada

```
if dt==yt then
    contador = contador + 1;
    if contador==8 then
        break
    end
end
```

O problema do AND possui duas classes, sendo uma classe a que possui valor igual a 0 e a segunda classe com valor igual a 1. Para plotar os valores no gráfico, será necessário

definir os valores x1 e x2 das classes um e dois. Para fazer o plot da reta deverá ser utilizada a Equação 04, onde w1 e w2 são os pesos da rede.

$$x2 = -(w1/w2)*x1 + \Theta/w2 \quad (04)$$

A Figura 07 exibe o código de plotagem.

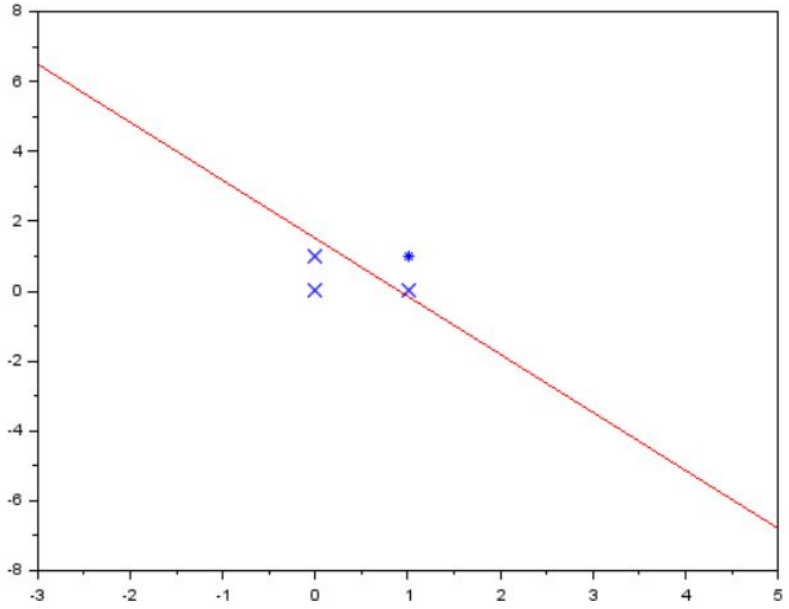
Figura 07: código de plotagem

```
//fazer a plotagem
classe1x1 = 1;
classe1x2 = 1;
classe2x1 = [0 0 1];
classe2x2 = [0 1 0];
....
x1 = linspace(-3,5);
x2 = -((wt(2)/wt(3))*x1) + ((wt(1)/wt(3))); //equação da reta

//fazer a plotagem
plot(classe1x1,classe1x2,'*');
plot(classe2x1, classe2x2, 'x');
plot(x1,x2,'r-');
```

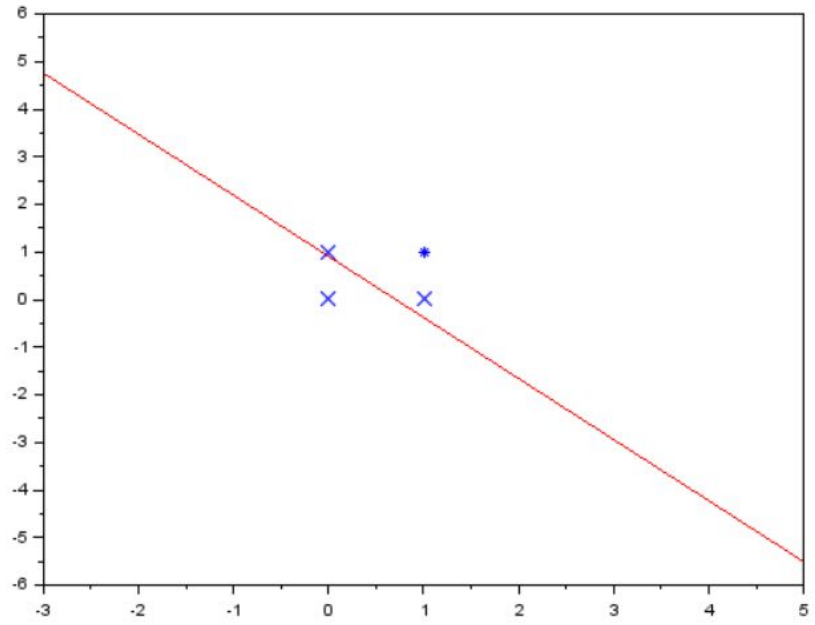
O passo de aprendizagem foi escolhido como 0.4, pois foi o valor que gerou melhores resultados. A Figura 08 exibe o gráfico de saída quando o passo de aprendizagem é 0.9 e com 20 épocas.

Figura 08: Passo de aprendizagem igual a 0.9



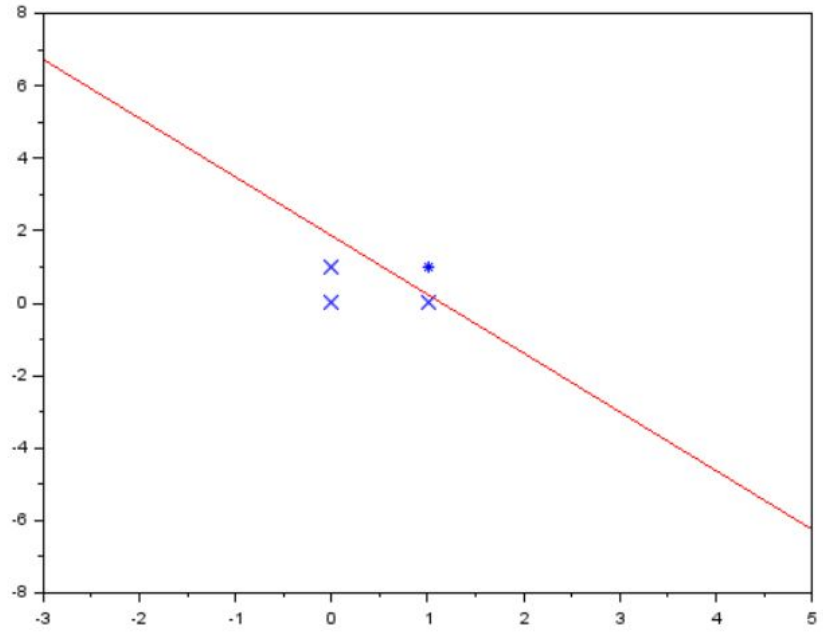
A Figura 09 exibe o gráfico de resultado quando o passo de aprendizagem é 0.6 e com 20 épocas.

Figura 09: Passo de aprendizagem igual a 0.6



A Figura 10 exibe o gráfico de resultado quando o passo de aprendizagem é 0.4 e com 20 épocas.

Figura 10: Passo de aprendizagem igual a 0.4



No terminal do Scilab é exibido os valores de entrada, saída, a predição e o erro gerado. A Figura 11 exibe o terminal do scilab.

Figura 11: Terminal Scilab.

```
"EPOCA"

 7.

"entrada"

-1.   1.   0.

"Saida esperada"

 0.

"Saida da network"

 0.

"Erro"

 0.
```

3. Rede RBF

As redes RBF utilizam funções de base radial como funções de ativação. Com isso, as saídas desse tipo de rede são uma combinação linear de funções de base radial e as entradas e parâmetros dos neurônios. Para este trabalho, será utilizado uma rede RBF para criar uma curva que se ajuste ao conjunto de dados do aerogerador. A Figura 12 exibe os dados do aerogerador sendo importados e o comando para que o usuário insira a quantidade de neurônios desejada no teste.

Figura 12: Importação do dataset

```
dataset = fscanfMat('C:\Users\STEFA\Desktop\ic\aerogerador.dat'); //

num_neuronios = input("Digite a quantidade de neuronios desejada");
```

Após isso, os dados do Dataset são processados, separando os valores de x e y e normalizando os valores para que eles fiquem dentro do intervalo de 0 e 1. A normalização é utilizada como uma forma de melhorar o desempenho da rede. A Figura 13 exibe o código do processo de normalização dos dados.

Figura 13: Normalização dos dados

```
x = dataset(:,1)';
x = x./max(x); //normaliza os valores de x para ficarem entre 0 e 1

y = dataset(:,2)';
y = y./max(y); //normaliza os valores de y para ficarem entre 0 e 1
```

É de suma importância a definição dos valores da quantidade de amostras da rede. Ademais, também é necessário o cálculo do centróide. Para isso são escolhidos N valores aleatórios de amostras do dataset para compor o valor do centróide, sendo N o número de

neurônios da rede. A Figura 14 exibe o código do cálculo do centróide e da quantidade de neurônios da rede.

Figura 14: Definição dos valores dos centróides

```
quat_amostras = length(y); //quantidade

//calculo dos centroides
index = grand(1,"prm",1:quat_amostras);
t = x(index(1:num_neuronios)); //recebe
```

Após isso foi realizado o cálculo da função de ativação, utilizando a função gaussiana, exibida na Equação 05. A Figura 15 exibe o código da função de ativação gaussiana.

$$\varphi(v) = \exp(-u^2/2\sigma^2) \quad (05)$$

Figura 15: Código da função de ativação Gaussiana

```
sig = 1;
pv = ones(num_neuronios,quat_amostras); //reto
for i=1:quat_amostras
    u = abs(repmat(x(i),1,num_neuronios) - t);
    pv(:,i) = exp(-(u.^2/(2*sig)));
end
```

A regularização de Thikonov é utilizada para minimizar os efeitos da multicolinearidade, ela pode ser exibida na Equação 06.

$$B = (pv'*pv + \lambda*I)*(-1)*(pv*y) \quad (06)$$

Sendo I dada pela matriz identidade de dimensão (k+1)x(k+1) onde K é o número de neurônios da rede. A Figura 16 exibe o código para o cálculo da regularização de Thikonov.

Figura 16: Regularização de Thikonov

```
//adicionando o valor do -1 do bias no vetor
pv = [-1*ones(1,quat_amostras);pv];

lambda = 1*10.^(-9); //define o valor do lambda
B = y*pv'*(pv*pv'+lambda*eye(num_neuronios+1,num_neuronios+1))^(-1);
y_ = B*pv;
```

O coeficiente de determinação R2 foi utilizado para definir a adequação do modelo da rede. A Figura 17 exibe o código para o cálculo do R2 e da plotagem do gráfico.

Figura 17: Cálculo R2

```
R2 = 1 - sum((y-y_).^2)/sum((y-mean(y)).^2);
disp("VALOR DE R2");
disp(R2);

title("Valor de R2 = " + string(R2));
plot(x,y,".");
plot(x,y_,"k--");
```

A Tabela 02 exibe os valores de R2 de acordo com o número de neurônios da rede.

Tabela 02: Valor de R2 para diferentes quantidades de neurônios

Quantidade de Neurônios	R2
5	0.968435
10	0.9735723
30	0.9737013

Figura 18: Gráfico da curva de ajuste para 10 neurônios

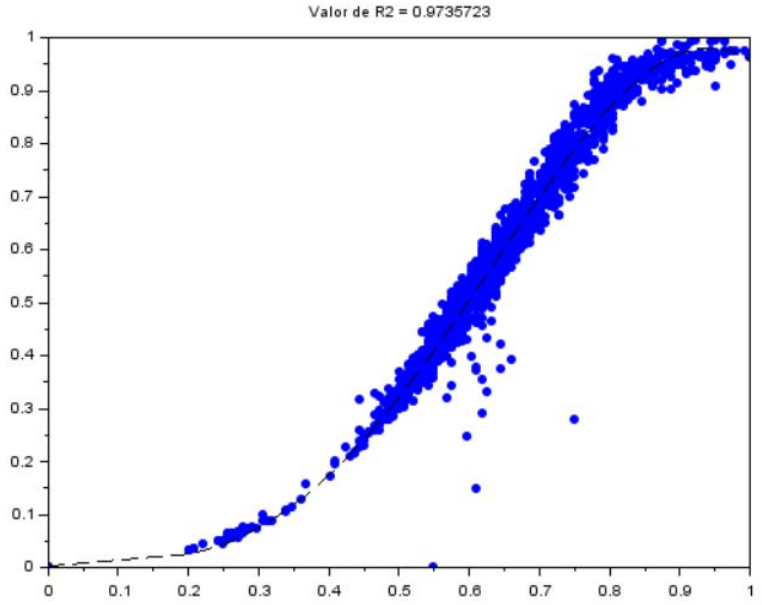


Figura 18: Gráfico da curva de ajuste para 30 neurônios