**UNIVERSIDADE FEDERAL DO CEARÁ - CAMPUS SOBRAL**

**CURSO ENGENHARIA DA COMPUTAÇÃO**

**DISCIPLINA: INTELIGÊNCIA COMPUTACIONAL**

**PROFESSOR: JARBAS JOACI DE MESQUITA SÁ JUNIOR**

# RELATÓRIO

**Andressa Gomes Moreira – 402305**

**Sobral – CE**
**2020**

## 1. Questão 01

As soluções referentes à questão 01 encontram-se no arquivo questao01.sce, na qual foi-se utilizado o software Scilab versão 6.1.0. Dessa forma, implementou-se um neurônio Perceptron, a fim de resolver o problema do AND e plotar os pontos da tabela-verdade e a reta de separação das classes desses pontos. A priori, definiu-se um passo de aprendizagem igual a 0.5 e criou-se um vetor de pesos com valores aleatórios.

```
//1.1 - Definir valor do passo de aprendizagem (n) entre 0 e 1
n = 0.5       //Passo de aprendizagem(n)

//1.2 - Criar o vetor de pesos e limiar w(0) com valores aleatórios.
w = rand(1,3)  //Vetor de pesos
```

Figura 1 – Passo de aprendizagem e vetor de pesos

Assim sendo, para resolver o problema da função AND deve-se determinar uma matriz contendo os valores de entrada da tabela verdade e o bias igual a -1, como também uma saída contendo os valores de saída da tabela-verdade. Em seguida, selecionou-se o vetor de entrada (x), contendo todas as possibilidades para os valores de entrada da função AND.

```
entrada=[-1 0 0; -1 0 1; -1 1 0; -1 1 1];//Valores de entrada para função AND
saida=[0; 0; 0; 1]                       //Valores esperados como saída
x=entrada(i, :)                          //Vetor de entrada x(t)
```

Figura 2 – Valores de entrada e saída

Ademais, foi-se calculado a ativação u(t), definida como o produto escalar entre o vetor de entrada (x) e o vetor de pesos (w) e a partir da função de ativação determinou-se a saída gerada pela rede y(t), uma vez que se a função de ativação for maior do que zero a saída y(t) é igual a 1, caso contrário, y(t) é igual a zero.

```
  u = w * x'      //Função de ativação u(t)
  if u > 0 then
      y = 1;      //saída gerada pela rede y(t) = 1
  else y = 0;     //saída gerada pela rede y(t) = 0
  end
```

Figura 3 – Função de ativação e saída gerada pela rede

Em seguida, iniciando o treinamento, calculou-se o erro, que é a diferença entre a saída esperada e a saída encontrada pela rede.

```
d = saida(i) //Verificar a saída desejada para a função AND
e = d - y;   //Erro
```

Figura 4 – Erro

Seguindo com o treinamento, o próximo passo foi ajustar os pesos via regra de aprendizagem e definir o critério de parada. Dessa forma, se o erro for diferente de zero, ajusta os pesos, caso contrário, continua os mesmos valores para o vetor de pesos. Ademais,

como critério de parada determinou-se que para encerrar o treinamento o erro tem que ser igual a zero n vezes seguidas, sendo “n” o tamanho do vetor que armazena os valores esperados como saída. Logo, para esse exemplo, o erro ser igual a zero durante quatro vezes para finalizar o treinamento, se a condição não for atendida deve-se voltar para a etapa de selecionar o vetor de entrada, calcular a ativação e a saída gerada pela rede.

```
if e ~= 0 then
    w = (w + (n * e * x));    // Ajustar pesos via regra de aprendizagem.
    comp = 0                  // Zera a variável que irá definir o critério de parada
else
    comp = comp + 1 // Atualiza a variável que irá definir o critério de parada
end

if comp == length(saida) then
    break
end
```

Figura 5 – Ajuste de pesos e critério de parada

Desse modo, após o treinamento encontrou-se uma solução na seguinte forma:

$$w = \begin{bmatrix} \theta \\ w_1 \\ w_2 \end{bmatrix} \tag{1}$$

Ademais, fez-se a plotagem do gráfico, na qual obteve-se uma reta para separar os pontos da tabela-verdade, dividindo-os em duas classes, na qual a classe 1 contém os pontos em que a saída (y) é igual a 1 e a classe 2 agrupa os pontos em que a saída (y) é igual a zero. Assim sendo, a reta de separação foi obtida através da equação da reta no plano (x1, x2):

$$x_2 = -\left(\frac{w_1}{w_2} * x_1\right) + \left(\frac{\theta}{w_2}\right) \tag{2}$$

A plotagem do gráfico:

```
    // Pontos que representam a classe 1:
    plot(1, 1, 'X');

    // Pontos que representam a classe 2:
    class2_x1 = [0 0 1];
    class2_x2 = [0 1 0];
    plot(class2_x1, class2_x2, 'o');

    // A equação da reta:
    x1 = linspace(-2, 3);
    x2 = -((w1/w2)*x1) + (teta/w2);
    plot(x1, x2, '--r');
```

Figura 6 – Plotagem do gráfico

Portanto, como o vetor de pesos é criado com valores aleatórios pode-se obter soluções diferentes a cada treinamento. Logo, algumas soluções encontradas e seus respectivos gráficos são exibidos a seguir:

| Treinamento t = 22 | Vetor de pesos w (0) | Solução Encontrada |
|---|---|---|
| $\boldsymbol{\theta}$ | 0.3873779 | 1.8873779 |
| $\boldsymbol{w_1}$ | 0.9222899 | 1.4222899 |
| $\boldsymbol{w_2}$ | 0.9488184 | 0.9488184 |

Tabela 1 – Resultados

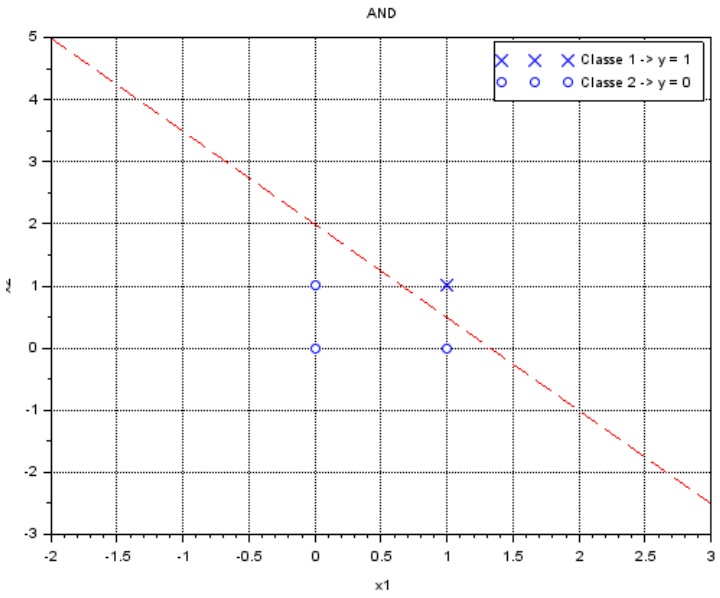


Figura 7 – Gráfico gerado a partir dos valores da tabela 1

| Treinamento t = 18 | Vetor de pesos w (0) | Solução Encontrada |
|---|---|---|
| $\boldsymbol{\theta}$ | 0.2615761 | 1.2615761 |
| $\boldsymbol{w_1}$ | 0.4993494 | 0.9993494 |
| $\boldsymbol{w_2}$ | 0.2638578 | 0.2638578 |

Tabela 2 – Resultados

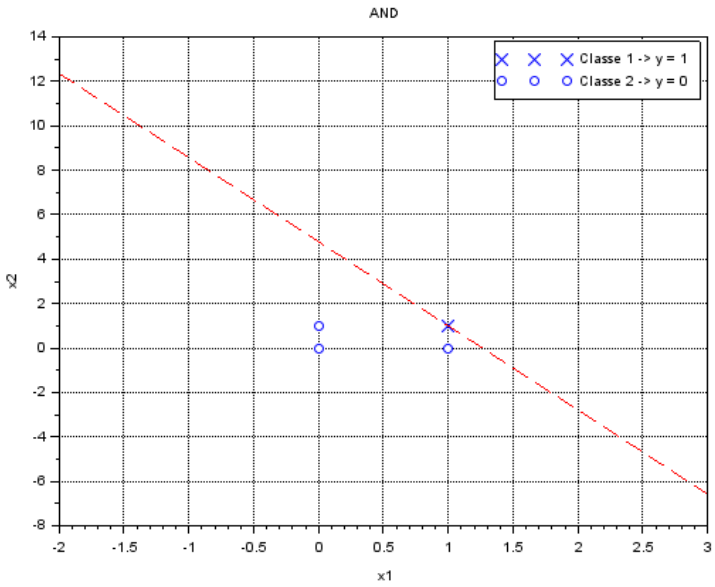


Figura 8 – Gráfico gerado a partir dos valores da tabela 2

| Treinamento t = 10 | Vetor de pesos w (0) | Solução Encontrada |
|---|---|---|
| $\boldsymbol{\theta}$ | 0.068374 | 0.568374 |
| $\boldsymbol{w_1}$ | 0.5608486 | 0.5608486 |
| $\boldsymbol{w_2}$ | 0.6623569 | 0.1623569 |

Tabela 3 – Resultados

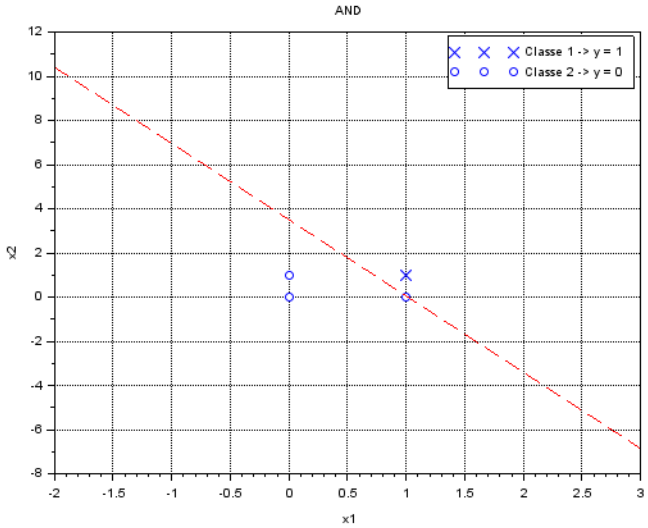


Figura 9 – Gráfico gerado a partir dos valores da tabela 3

Portanto, pode-se observar que é possível encontrar infinitas retas que separam as duas classes, uma vez que os pesos são determinados de forma aleatória. Entretanto, algumas soluções não são tão eficientes, como visto na figura 08, pois a reta de separação passa exatamente em cima do ponto (1,1).

## 2. Questão 02

As soluções referentes à questão 02 encontram-se no arquivo questao02.sce. A priori, a base de dados “aerogerador.dat” foi carregada, a primeira coluna está relacionada aos valores da velocidade do vento (x) e a segunda exibe os valores para a potência gerada (y). Dessa forma, implementou-se uma rede neural RBF para realizar a criação de uma curva que se ajuste aos dados do aerogerador. Logo, trata-se de uma rede que apresenta uma única camada oculta, na qual os neurônios possuem uma função de ativação de base radial.

A princípio, foi-se calculada a distância (norma) do vetor de entrada ao centro de um cluster. Ademais, as saídas de cada neurônio da camada oculta foram definidas por meio de uma função de ativação de base radial e ao resultado foi adicionado um bias igual a -1.

```
//n = quantidade de neurônios ocultos

//Definir o centroide
aleat_x = grand(1,'prm', x) //Realiza uma pertubação nos valores da entrada
centroide = aleat_x(1:n)    //Seleciona os n (número de neurônios ocultos)
                            //primeiros valores para compor o centroide.

for i = 1:amostra

    //Definir a norma u = ||x - centroide||
    vet = [x(i)*ones(1,n)] - centroide
    norma = abs(vet)'

    //Implementar a função ativação dos neurônios ocultos
    beta = (1/(2*sigma))
    z(:, i)= exp(-(norma.^2)*beta)
end

//Adicionar o bias = -1
Z = [-1*ones(1,amostra); z]
```

Figura 10 – Norma, função de ativação e bias.

Outrossim, para definir os pesos da camada de saída, foi-se utilizada a equação de solução do problema dos quadrados mínimos, conhecida como equação normal.

$$Z^T * Z * w = Z^T * y \Rightarrow w = (Z^T * Z)^{-1} * (Z^T * y) \quad (3)$$

Entretanto, para melhorar os resultados aplicou-se a regularização de Thikonov, utilizada para reduzir os efeitos da multicolinearidade, que ocorre quando as linhas da matriz $Z^T * Z$ não são linearmente independentes. Assim a equação (3) foi reescrita da seguinte forma:

$$w = (Z^T * Z + \lambda * I)^{-1} * (Z^T * y) \quad (4)$$

```
    lambda = 0.000000001;
    I = eye(n+1,n+1);
    w = y*Z'*((Z*Z')+(lambda*I))^(-1);
```

Figura 11 – Pesos utilizando a regularização de Thikonov.

Em seguida, definiu-se o coeficiente de determinação (R2) usado para definir a adequação de um modelo.

```
/* Coeficiente de Determinação */
function [R2, y_preditor]=coef_determ(Z, w)
    //Definir o modelo de regressão ajustado (preditor) y^
    y_preditor = w * Z;

    //Definir o Coeficiente de determinação R2
    media = mean(y);
    somat1 = sum((y - y_preditor).^2);
    somat2 = sum((y-media).^2);
    R2 = 1-(somat1/somat2);
endfunction
```

Figura 12 – Coeficiente de Determinação

Por fim, plotou-se os gráficos com as curvas que se ajustam ao conjunto de dados do aerogerador para diversas quantidades de neurônios ocultos.

```
/* PLOTAGEM DOS GRÁFICOS */
function []= plotar(y_preditor, R2, n)
    clf;
    //Passo 09: Plotagem do gráfico
    plot(x,y,'.');
    plot(x,y_preditor,'r-');

    title('Rede RBF com ' + string(n) + ' neurônios ocultos. R2 =' + string(R2))
    xlabel('Velocidade do Vento x');
    ylabel('Potência Gerada y');
endfunction
```

Figura 13 – Plotagem dos gráficos

Portanto, a tabela 4 apresenta os resultados dos coeficientes de determinação para diferentes quantidades de neurônios oculto.

| **Quantidades de neurônios ocultos.** | ***R2*** |
|---|---|
| **2** | 0.9532539 |
| **5** | 0.9613314 |
| **10** | 0.9736762 |
| **20** | 0.9736848 |
| **50** | 0.9736925 |

Tabela 4 – Resultados dos coeficientes de determinação

Portanto, o algoritmo gerou gráficos para exibir a criação de uma curva que se ajuste aos dados do aerogerador para diferentes valores dos neurônios ocultos, como é mostrado a seguir:

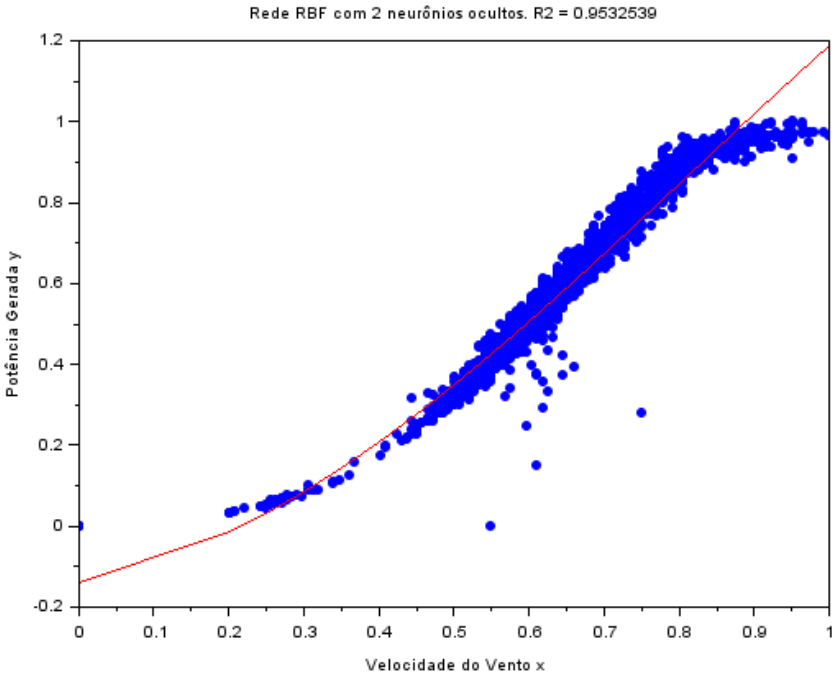


Figura 14 – RBF com 2 neurônios ocultos

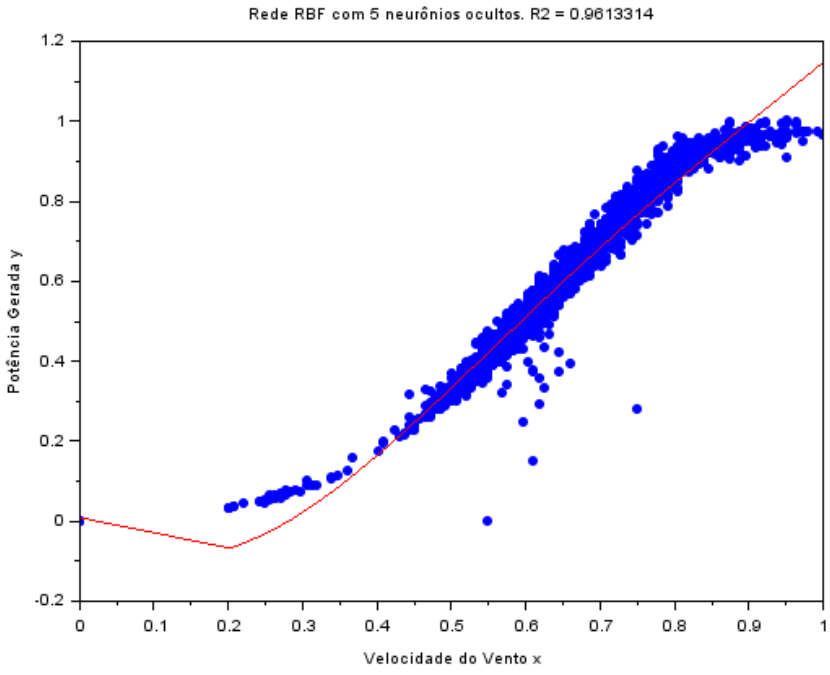


Figura 15 – RBF com 5 neurônios ocultos

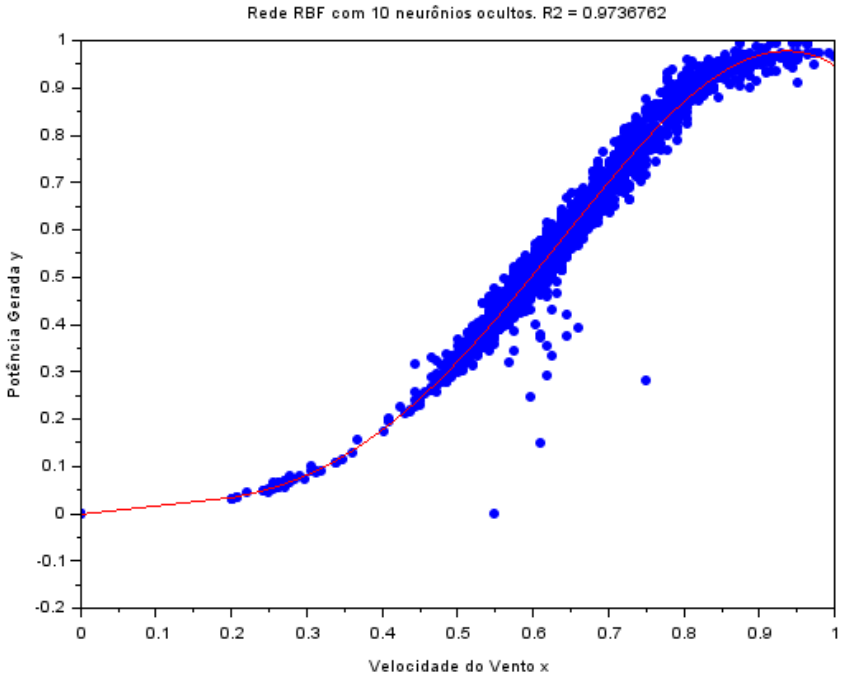


Figura 16 – RBF com 10 neurônios ocultos

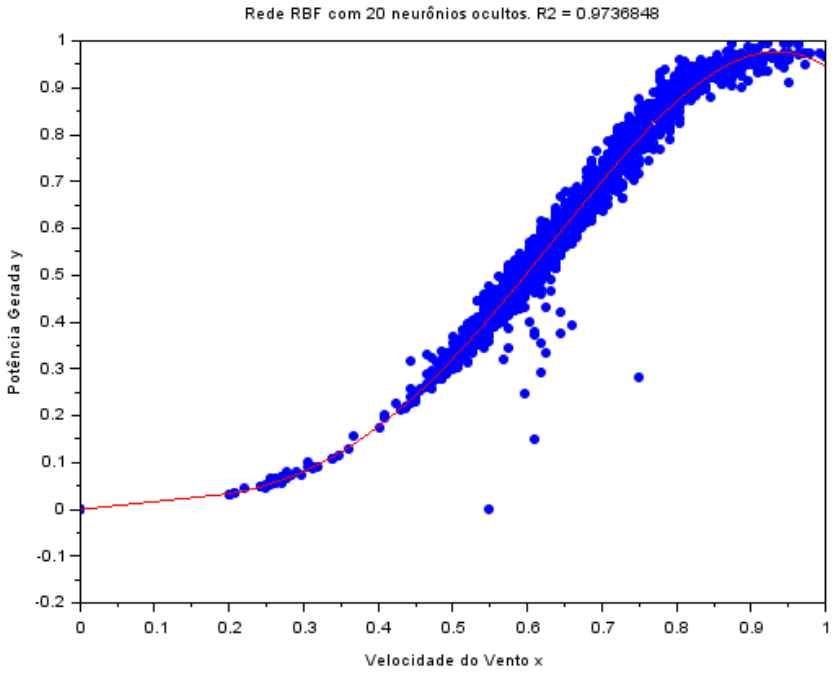


Figura 17 – RBF com 20 neurônios ocultos

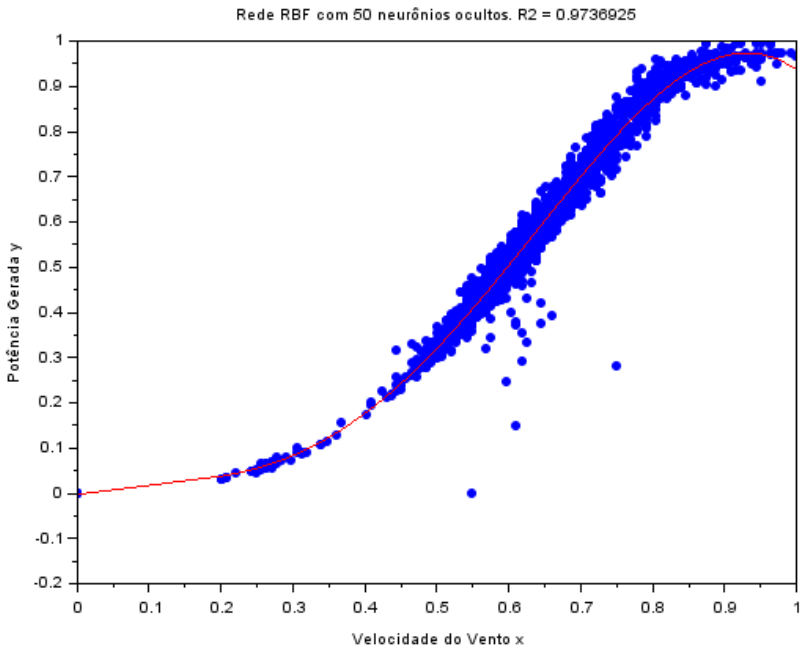


Figura 18 – RBF com 50 neurônios ocultos

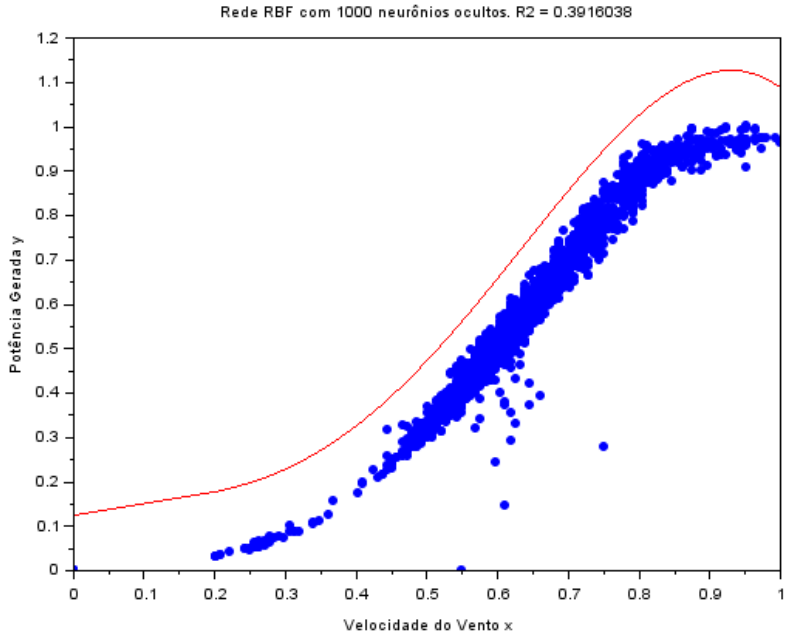


Figura 19 - RBF com 1000 neurônios ocultos

Portanto, é possível observar que o comportamento da rede RBF é similar ao das regressões polinomiais, uma vez que a curva se adequa melhor ao comportamento dos dados à medida que a quantidade de neurônios ocultos é aumentada, e consequentemente, o resultado para o coeficiente de determinação também melhora. Entretanto, é possível observar que esse aumento na quantidade de neurônios ocultos é limitado, pois ao adicionar uma quantidade exacerbada de neurônios a curva não apresenta um resultado interessante.