UNIVERSIDADE
DE TRÁS-OS-MONTES
E ALTO DOURO

Trabalho Prático 3 – Redes Neuronais Artificiais

Engenharia Informática

Inteligência Artificial

José Paulo Barroso de Moura Oliveira

Eduardo José Solteiro Pires

Autores

João Leal - 68719

Vítor Neto - 68717

Vila Real, 23 de janeiro de 2021

Índice

1. Introdução

Neste trabalho pretende-se dar o exemplo de uma rede neuronal de alimentação direta (feedforward neural network).

As redes neuronais baseiam-se na estrutura dos neurónios, presentes nos seres vivos, cujos estão conectados por sinapses que produzem uma resposta química a uma entrada. O tamanho da resposta varia de acordo com a soma das reações da sinapse, se esse valor ultrapassar uma determinada threshold (valor limite), o neurónio dispara.

Uma rede neuronal, no fundo, é uma função avançada para aproximações, à qual se fornece uma matriz vetor de inputs, esta realiza uma séria de operações e devolve um vetor de outputs. No caso deste trabalho, a rede neuronal irá realizar uma operação lógica, o XOR (ou exclusivo). Para que uma rede neuronal forneça um vetor de outputs considerado correto, é necessário que esta passe por uma fase de treino, onde a mesma se irá basear no set de treino que será fornecido para que aprenda e generalize futuras inferências.

As redes neuronais são formadas por camadas, cada uma composta por estruturas denominadas de neurónio. Cada neurónio produz um resultado, uma ativação, baseado nos outputs fornecidos pelas camadas anteriores, que serão também manipuláveis por uma série de pesos calculados pela fase de treino. As redes neuronais feedforward, são assim denominadas porque os dados de uma camada são fornecidos à camada seguinte pelo método de mesmo nome, o processo de feedforward. Para determinar o quão bom o treino de uma rede neuronal é, existe uma métrica denominada de erro (no caso deste trabalho, sum_sq_error). Quanto menor for esta soma dos erros, melhor é considerado o resultado do treino da rede. A soma dos erros é baseada na soma dos erros quadráticos, que irá ser abordado mais intensivamente nos tópicos seguintes.

Serão abordados dois exemplos de redes neuronais, um com topologia 3x3x1 e outro com 3x5x3. Seguidamente será fornecida também uma matriz objetivo, para comparação com os resultados obtidos pela rede, para que sejam comparados e validados. O objetivo é que as redes neuronais sejam treinadas e, seguidamente, testadas. Nesta fase de testes, é pretendido que o output obtido seja igual ao da matriz objetivo fornecida.

2. Algoritmos das Redes Neuronais 3x3x1 e 3x5x3

2.1. Introdução aos Algoritmos das Redes Neuronais 3x3x1 e 3x5x3

Neste capítulo, vai ser feita uma breve explicação dos algoritmos desenvolvidos e utilizados para teste das redes neuronais pedidas. No capítulo 2.1.1 irá ser explicado o algoritmo da rede neuronal 3x3x1 e no capítulo seguinte, 2.1.2, o algoritmo da rede neuronal 3x5x3. Antes de começar esta explicação, é também fornecida a função de ativação sigmoidal, que irá ser útil no desenvolvimento de ambos os algoritmos:

```
function [s] = sig(x)
    s = 1./(1+exp(-x));
end
```

Figura 2.1.1 – Função de ativação sigmoidal

2.1.1 Introdução ao Algoritmo da Rede Neuronal 3x3x1

Para este algoritmo, começaram por serem estabelecidas as variáveis iniciais:

```
n_epochs = 2000; %Numero de épocas (iterações)
alpha = 0.9; %Fator de aprendizagem

%Vetor soma dos erros quadráticos
SSE = zeros(1, n_epochs);

N = 4; %Número de amostras

%Matriz com as amostras de entrada da função XOR
X = [0 0 1;
     0 1 1;
     1 0 1;
     1 1 1];

%Saídas da função XOR
T = [ 0
      1
      1
      0 ];
%Inicialização aleatória dos pesos [-1,1]
%W1-2x3 W2- 1x3
W1 = 2*rand(2,3) - 1
W2 = 2*rand(1,3) - 1

for epoch = 1:n_epochs
    sum_sq_error=0;
    for k = 1:N
        x = X(k,:)';
        t = T(k);

        %Soma da camada de entrada
        g1 = W1*x;
```

Figura 2.1.1.1 – Estabelecimento das variáveis iniciais

As variáveis estabelecidas são:

- Número de iterações: n_epochs (2000);
- Fator de aprendizagem: alpha (0.9);
- Vetor com a soma dos erros quadráticos: SSE, preenchido com 0s e com o número de colunas conforme o número de iterações, neste caso 2000;
- Número de amostras: N (4);
- Matriz com as amostras de entrada para a função XOR: X;
- Matriz com as saídas esperadas da função XOR: T;
- Inicialização aleatória dos pesos para a camada de saída e de entrada, W2 e W2, respetivamente, entre -1 e 1;
- Inicialização dos ciclos for conforme o número de iterações e inicialização da soma dos erros quadráticos: sum_sq_error (0);
- Inicialização das variáveis x e t, com os dados das matrizes dos inputs e outputs, respetivamente.

```
31          %Soma da camada de entrada
32 -        g1 = W1*x;
33          %Função de ativação sigmoidal
34 -        y1 = sig(g1);
35
36          %Adição à saída da camada escondida y1
37          %da entrada de bias com +1
38          %Resulta em y1_b
39 -        y1_b = [y1
40                  1];
41
42          %Soma da camada de saída
43 -        g2 = W2*y1_b;
44          %Função de ativação sigmoidal
45 -        y2 = sig(g2);
46
47          %Retropropagação
48          %Erro da camada de saída
49 -        e = t - y2;
50          %Cálculo do delta da camada de saída
51          %Sigmoide
52 -        delta2 = y2.*(1-y2).*e;
53
54          %Atualização da soma dos erros quadráticos
55 -        sum_sq_error = sum_sq_error + e^2;
56
57          %Erro da camada escondida
58 -        e1 = W2'*delta2;
59          %Erro sem o bias
60 -        e1_b = e1(1:2);
61
62          %Cálculo do delta da camada de saída
```

Figura 2.1.1.2 – Dentro do ciclo for

Para a camada de entrada, é feita a soma da camada de entrada, elemento a elemento da matriz de entrada, com o peso da camada de entrada, W1:

$$g1 = W1 * x; (1)$$

Com esta soma calculada, passa-se à ativação de g1, com uso da função de ativação, a função sigmoidal:

$$y1 = sig(g1); (2)$$

À camada de saída escondida, adiciona-se a camada de bias, através da função (3):

$$y1\_b = [y1; 1]; (3)$$

Tal como foi feito para a camada de entrada, terá de ser efetuada a soma do peso à camada de saída, através da fórmula (4):

$$g2 = W2 * y1\_b; (4)$$

À soma da camada de saída com o peso, será também efetuada a função de ativação:

$$y2 = sig(g2); (5)$$

Com a adição da retropropagação, o algoritmo da rede neuronal vai ficar muito mais aceitável. Antes da adição deste algoritmo de retropropagação, uma maneira de tentar melhorar os resultados produzidos pelas redes neuronais, tentava-se usar algoritmos genéticos para melhorar a performance. Com o algoritmo de retropropagação, esta ajuda a que os resultados obtidos sejam significativamente melhorados. Para isto, é acrescentado o erro à "cauda" da rede neuronal. O erro é calculado pela seguinte fórmula:

$$e = t – y2; (6)$$

Com o erro calculado, atualiza-se a matriz das somas dos erros quadráticos, ou seja:

$$sum\_sq\_error = sum\_sq\_error + e^2; (7)$$

Para atualizar o delta da camada de saída, prossegue-se ao uso da seguinte fórmula:

$$delta2 = y2.*(1-y2).*e; (8)$$

Seguidamente, são também calculados os erros da camada escondida e o erro sem o bias, através das equações (8) e (9), respetivamente:

$$e1 = W2' * delta2; (9)$$

$$e1\_b = e1(1:2); (10)$$

No próximo passo, irão ser analisadas as atualizações dos pesos das camadas escondidas e de entrada.

```
61
62          %Cálculo do delta da camada de saída
63 -        deltal = yl.*(l-yl).*el_b;
64
65          %Atualização dos pesos da camada escondida
66 -        dW2 = alpha * delta2 * yl_b'; %Com bias
67 -        W2 = W2 + dW2;
68
69          %Atualização dos pesos da camada de entrada
70 -        dWl = alpha * deltal * x';
71 -        Wl = Wl + dWl;
72
73 -    end
74 -    SSE(epoch) = (sum_sq_error)/N;
75 - end
```

Figura 2.1.1.3 – Atualização dos pesos das camadas

Em primeiro lugar, calcula-se o delta da camada de saída, a partir da seguinte fórmula:

delta1 = y1.* (1-y1).*e1_b; (11)

Com os deltas calculados, passasse à atualização dos pesos das camadas, tanto da camada escondida, como da de entrada, com as fórmulas (12) e (13), para a camada escondida, e (14) e (15), para a camada de entrada:

dW2 = alpha * delta2 * y1_b'; (12)

W2 = W2 + dW2; (13)

dW1 = alpha * delta1 * x; (14)

W1 = W1 + dW1; (15)

No final de cada iteração, é também atualizada a matriz da soma dos erros quadráticos, dividindo cada uma das suas entradas pelo número total de amostras (N):

SSE(epoch) = (sum_sq_error)/N; (16)

Com todos estes passos concluídos, a rede está considerada treinada. Com a rede treinada, passa-se ao teste da mesma, através do algoritmo:

```
77      %Teste da rede
78 -    for k = 1:N
79
80 -        x = X(k,:)';
81 -        g1 = W1*x;
82
83          %Sigmóide
84 -        y1 = sig(g1);
85
86          %y1 mais uma entrada de bias
87 -        y1_b = [y1
88                  1];
89
90 -        g2 = W2*y1_b;
91
92          %Saída prevista XOR
93 -        y_plot(k) = sig(g2);
94 -    end
95 -    y_plot
96
97 -    It = 1 :1:n_epochs;
98 -    plot(It, SSE, 'r-', 'LineWidth', 2)
99 -    xlabel('Época')
100 -   ylabel('SSE')
101 -   title('Função de ativação: Sigmoide')
102
```

Figura 2.1.1.4 – Teste da rede

Para efetuar o teste da rede, recorre-se a um ciclo for conforme o número de amostras. Calcula-se os pesos novamente, com as fórmulas (1) e (4) e ativam-se novamente as funções, recorrendo à função de ativação sigmoidal. Com a função de ativação sigmoidal, calculam-se os resultados esperados e comparam-se com os esperados. Esta comparação irá ser analisada mais à frente, com erros diferentes (Capítulo 2.2).

No final dos ciclos, é efetuado um plot para expor a evolução do erro ao longo da iterações, que também irá ser analisado no capítulo 2.2.

2.1.2 Introdução ao Algoritmo da Rede Neuronal 3x5x3

Para a rede neuronal 3x5x3, a premissa é a mesma, porém alguns pontos chave serão modificados, como, por exemplo, as dimensões das matrizes.

```
n_epochs = 10000; %Numero de épocas (iterações)
alpha = 0.9; %Fator de aprendizagem

%Vetor soma dos erros quadráticos
SSE = zeros(3, n_epochs);

N=120; %Numero de amostras
%Amostras de entrada funçao XOR
Data_multi
X = [ X_A;
      X_B;
      X_C;
      ];

X=[X,ones(N,1)]; %X_Testes

%X=X_teste;

 %T 120x3
 %Primeiras 40 linhas 1º coluna a 1s
 %Segundas 40 linhas 2º coluna a 1s
 %Utlimas 40 linhas 3º coluna a 1s
 %Inicialização aleatoria dos pesos [-1,1]
 %Saidas da funçao XOR

T= [ones(40,1),zeros(40,2);
    zeros(40,1),ones(40,1),zeros(40,1);
    zeros(40,2),ones(40,1);
    ];
%W1-4X3 W2- 3X5
W1 = 2*rand(4,3) - 1;
W2 = 2*rand (3,5) - 1;
```

Figura 2.1.2.1 – Inicialização

A primeira mudança verificada, é no número de épocas. Serão realizadas 10000 épocas em vez de 2000. Como são fornecidas 3 classes diferentes no ficheiro Data_multi, que é um ficheiro externo ao algoritmo onde estão especificadas 3 classes diferentes de dados, então o vetor soma dos erros quadráticos terá de possuir também 3 linhas diferentes, 1 para os erros quadráticos para cada uma das classes. Outra diferença é a matriz de output, onde o resultado esperado é uma matriz de 120x3, onde as primeiras 40 entradas da primeira coluna representam

o resultado esperado para os dados da classe A, as entradas 41-80 da coluna 2 representam um número 1, que é o resultado esperado para os dados da classe B enquanto que as restantes colunas representam um 1 na coluna 3, resultado esperado para os dados da classe C. Os tamanhos das matrizes dos pesos, W1 e W2, foram também ajustados de forma a que ficassem adequados ao problema proposto.

```
63 -            equadrado = e.^2;
64
65              %Atualização da soma dos erros quadráticos
66 -            sum_sq_error = sum_sq_error + equadrado;
67
68              %Erro da camada escondida
69 -            e1 = W2'*delta2;
70              %Erro sem o bias
71 -            e1_b = e1(1:4);
72
73              %Cálculo do delta da camada de saída
74 -            delta1 = y1.*(1-y1).*e1_b;
75
76              %Atualização dos pesos da camada escondida
77 -            dW2 = alpha * delta2 * y1_b'; %Com bias
78 -            W2 = W2 + dW2;
79
80              %Atualização dos pesos da camada de entrada
81 -            dW1 = alpha * delta1 * x';
82 -            W1 = W1 + dW1;
83
84 -        end
85 -        A = (sum_sq_error)/N;
86 -        SSE(1,epoch) = A(1,1);
87 -        SSE(2,epoch) = A(2,1);
88 -        SSE(3,epoch) = A(3,1);
89 -    end
```

Figura 2.1.2.2 – Atualização dos pesos

Outra mudança importante é no cálculo dos erros, onde a soma do erro irá precisar de ser calculada em dois passos. Primeiro é efetuado o erro ao quadrado e apenas depois é efetuado o incremento deste erro calculado à soma total dos erros. No final do for interno é também efetuada a inserção dos erros linha a linha, visto que a matriz das somas dos erros quadráticos irá agora possuir 3 linhas, 1 para cada classe.

```
91      %Teste da rede
92 -    for k = 1:N
93
94 -        x = X(k,:)';
95 -        g1 = W1*x;
96
97          %Sigmóide
98 -        y1 = sig(g1);
99
100         %y1 mais uma entrada de bias
101 -       y1_b = [y1
102                 1];
103
104 -       g2 = W2*y1_b;
105
106         %Saída prevista XOR
107         %y_plot(k) = sig(g2);
108
109 -       y_plot(k,:) = sig(g2)';
110
111
112 -   end
```

Figura 2.1.2.3 – Teste da rede

O teste da rede irá ser também semelhante ao já visto para a rede 3x3x1, exceto no cálculo do y_plot, visto que terão que ser percorridas todas as colunas diferentes do mesmo, já que este possui mais colunas que o anterior.

```
115 -    It = 1 :1:n_epochs;
116 -    plot(It, SSE, 'r-', 'LineWidth', 2)
117 -    xlabel('Época')
118 -    ylabel('SSE')
119 -    title('Função de ativação: Sigmoide')
120 -    hold off
121
122 -    figure
123 -    hold on
124 -    grid on
125 -    Data_multi
126 -     plot(X_A(:,1), X_A(:,2), 'r*')
127 -     plot(X_B(:,1), X_B(:,2), 'b*')
128 -     plot(X_C(:,1), X_C(:,2), 'k*')
129 -     plot(X_teste(:,1), X_teste(:,2), 'go')
130 -    hold off
131
132 -    figure
133 -    hold on
134 -    grid on
135 -    for j = 1:120
136 -        if j > 0 && j <= 40
137 -        plot(y_plot(j,1), y_plot(j,2), 'r*')
138 -        end
139 -        if j > 40 && j <= 80
140 -        plot(y_plot(j,1), y_plot(j,2), 'b*')
141 -        end
142 -        if j > 80 && j <= 120
143 -        plot(y_plot(j,1), y_plot(j,2), 'k*')
144 -        end
145 -    end
```

Figura 2.1.2.4 – Análise de resultados

Para terminar, são criados 3 gráficos, que serão analisados mais pormenorizadamente no capítulo 2.2. O primeiro é o gráfico que possui a evolução do erro ao longo das iterações. O segundo possui os dados fornecidos no ficheiro Data_multi, onde se pode ver uma clara separação dos dados nas três classes definidas, as classes A, B e C. Finalmente, são apresentados os dados relativamente à matriz de output obtida no treino da rede adequadamente conforme cada uma das classes especificadas anteriormente.

2.2. Resolução do Problema com os algoritmos das Redes Neuronais Dados

Para os primeiros testes, foram inicializados os pesos no intervalo [-1; 1]. Com esta inicialização é pretendido que a matriz de output (y_plot) obtida seja semelhante à matriz esperada (T). No caso destes atributos que foram fornecidos, foi obtido o seguinte gráfico da evolução da soma dos erros quadráticos:

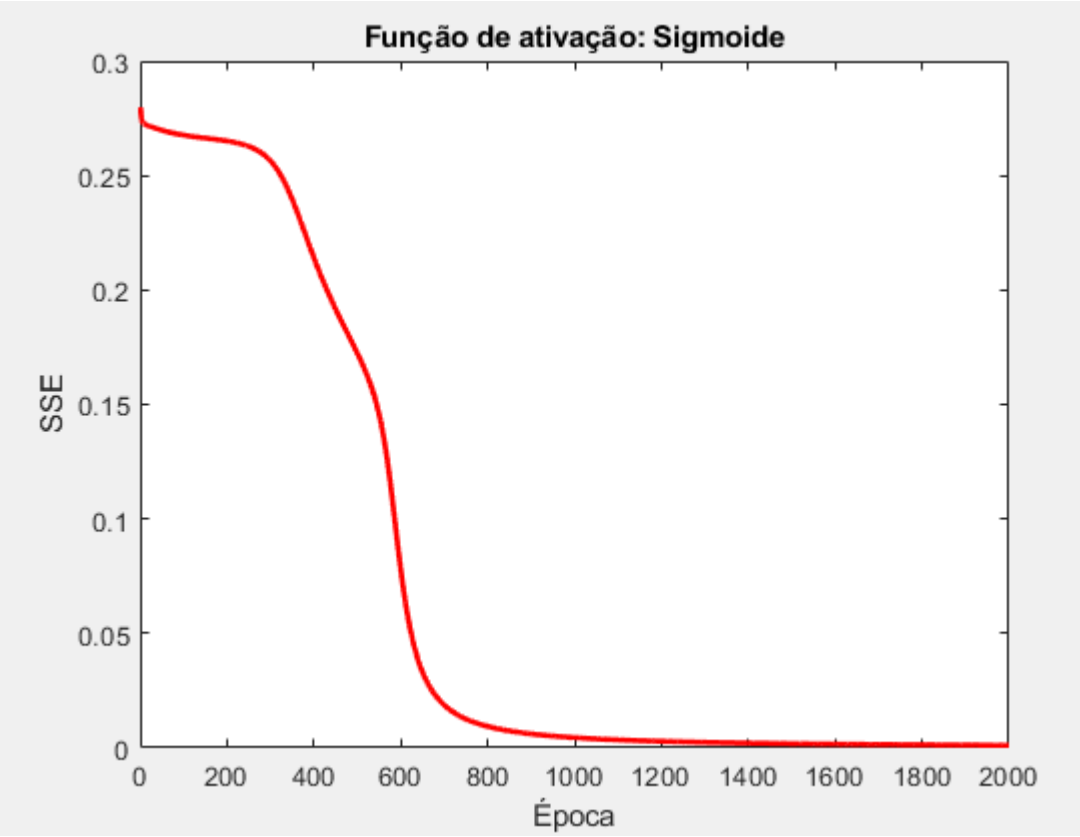

Figura 2.2.1 – Evolução da Soma dos Erros Quadráticos

Neste gráfico, pode ser observado que o erro começa muito alto, o que indica que a rede ainda está destreinada, e vai diminuindo ao longo das iterações, indicado uma óbvia evolução da eficácia da rede resultante do treino da mesma, até que o erro atinge, no exemplo fornecido, um erro de 0,0011, o que indica um erro extremamente baixo, que é o pretendido. Seguidamente, o mais importante é comparar os resultados pretendidos, que, neste caso, são a comparação entre a matriz de output e a matriz obtida com o algoritmo, onde o objetivo pretendido é que a primeira se assemelhe o máximo possível da última.

```
T =

     0
     1   y_plot =
     1
     0       0.0336    0.9607    0.9692    0.0303
```

Figura 2.2.2 – Comparação dos Resultados Obtidos

Como pode ser verificado, os resultados da matriz y_plot foram muito semelhantes aos pretendidos, apenas com um pequeno desvio do que era pretendido, como era esperado devido ao erro quadrático calculado de 0,0046.

O mesmo teste irá decorrer, mas com inicialiações de peso diferentes, neste caso, dentro de intervalos maiores. O resultado do teste que se segue foi obtido com uma inicialização dos pesos dentro do intervalo [-2; 2], o que significa que o algoritmo tem uma margem de erro maior, isto é, resultados mais extravasados do que aqueles pretendidos:

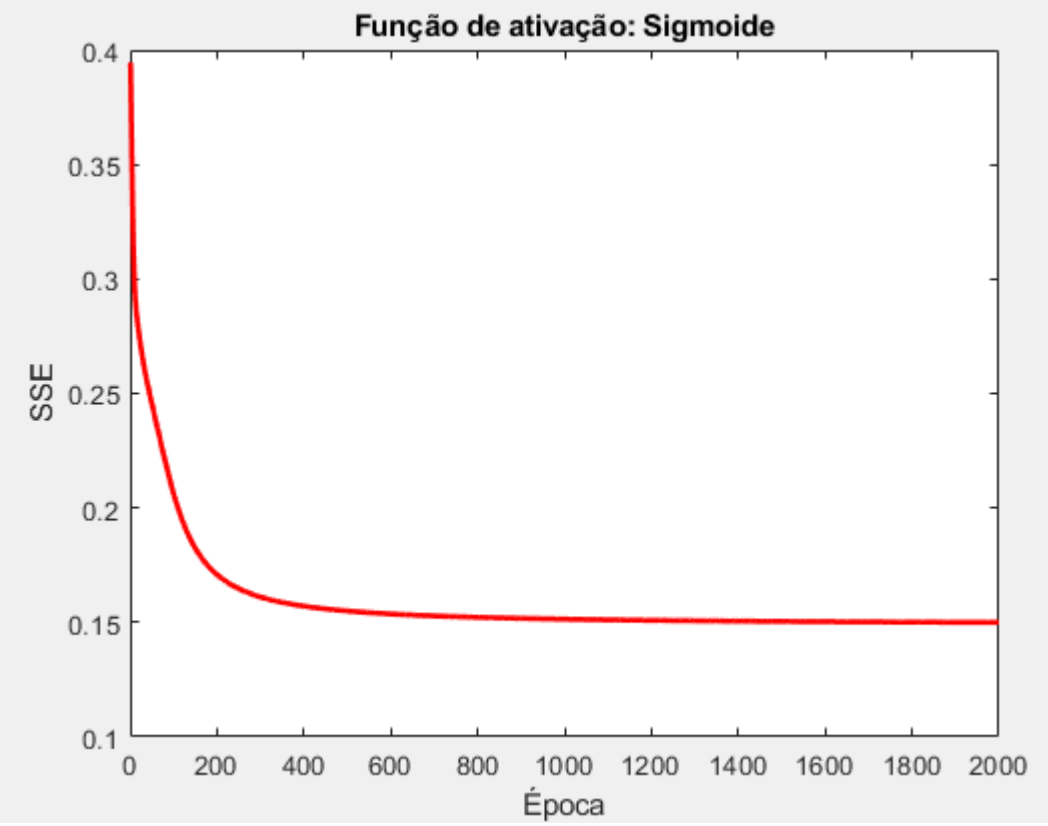

Figura 2.2.3 – Evolução do SSE com pesos no intervalo [-2; 2]

A diferença no gráfico é obvia, não atingindo este um resultado tão próximo do 0 como o exemplo fornecido anteriormente, atingindo um erro mínimo de 0,1532.

Comparando agora os resultados obtidos:

```
T =

     0
     1   y_plot =
     1
     0       0.0276     0.9696     0.4534     0.4560
```

Figura 2.2.4 – Comparação dos resultados obtidos para o intervalo [-2; 2]

O resultado obtido foi exatamente o esperado, uma aproximação ao pretendido notável, mas não tão notável e com muitos mais desvios que a inicialização dos pesos no intervalo [-1; 1]. Nestes resultados pode ser observado que as duas primeiras entradas da matriz obtida foram muito próximas do esperado enquanto que as duas consequentes ficaram aquém das expectativas.

Para uma inicialização dentro do intervalo [-5; 5], uma diferença ainda maior foi observada:

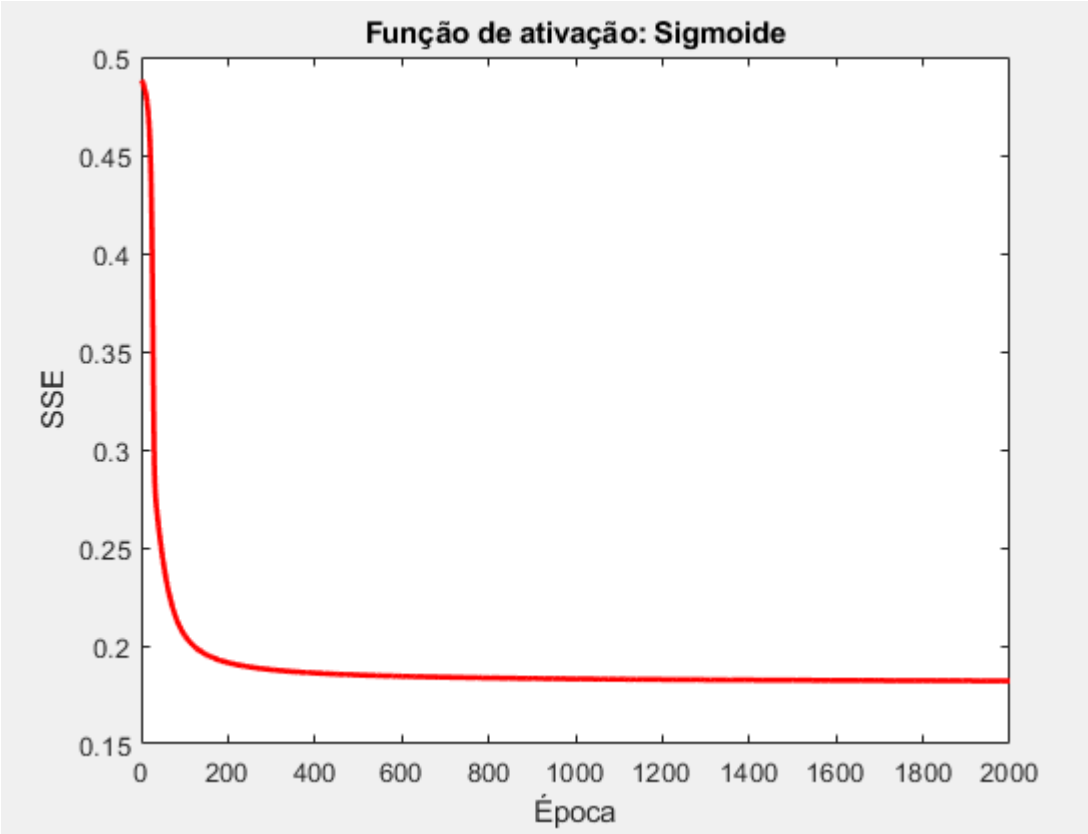

Figura 2.2.5 – Evolução do SSE com pesos no intervalo [-5; 5]

Onde o erro mal baixou do 0,2 e a soma dos erros quadráticos foi de 0,7284, um valor muito mais alto que aqueles verificados em iterações prévias.

```
T =

     0
     1
     1
     0

y_plot =

    0.0317    0.6528    0.6533    0.6552
```

Figura 2.2.6 - Comparação dos resultados obtidos para o intervalo [-5; 5]

Tal como era expectável, como o erro foi muito maior, o desvio dos resultados obtidos foi também muito superior ao que foi observado anteriormente, com as entradas dois, três e quatro da matriz a não se aproximarem sequer do resultado pretendido.

Com os resultados dos três testes realizados, pode-se concluir que, com um intervalo dos pesos mais estreito, melhor serão os resultados obtidos, ou seja, mais próxima será a matriz obtida da matriz de output fornecida no início da fase de testes.

Procede-se agora à fase de testes da rede neuronal 3x5x3, onde três classes de dados são fornecidas e observáveis no seguinte gráfico:

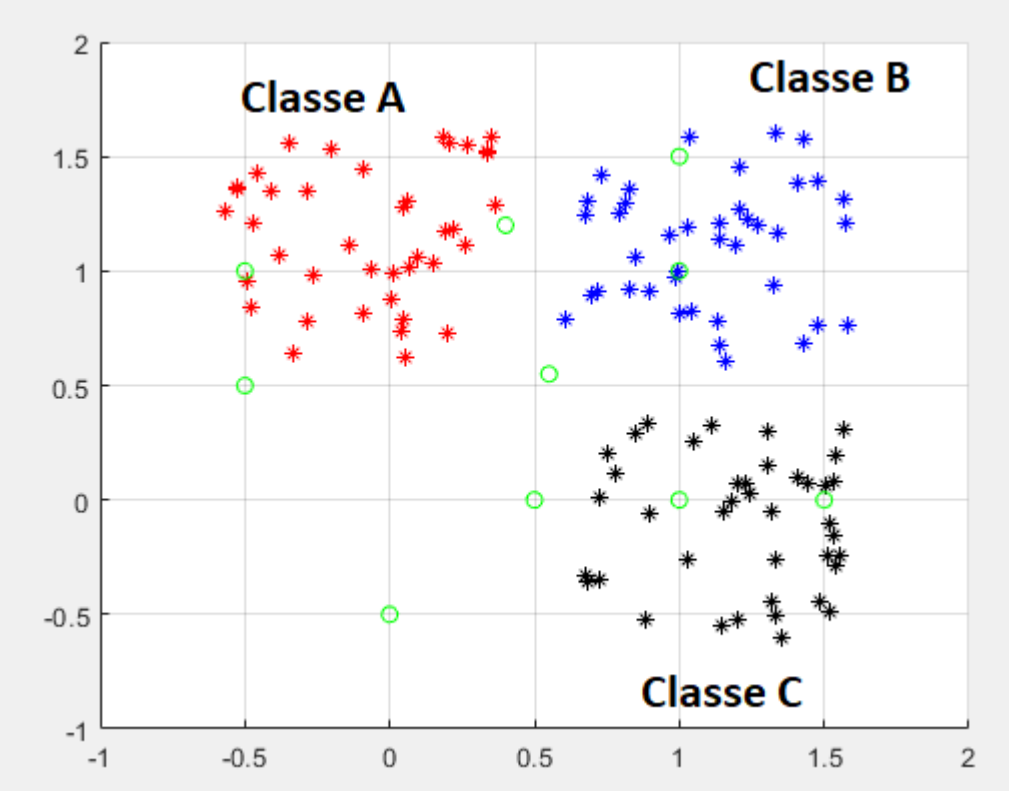

Figura 2.2.7 – Classes da rede neuronal 3x5x3

Pode também ser observada uma quarta “classe”, representada pelos círculos verdes, que representa uma matriz de testes.

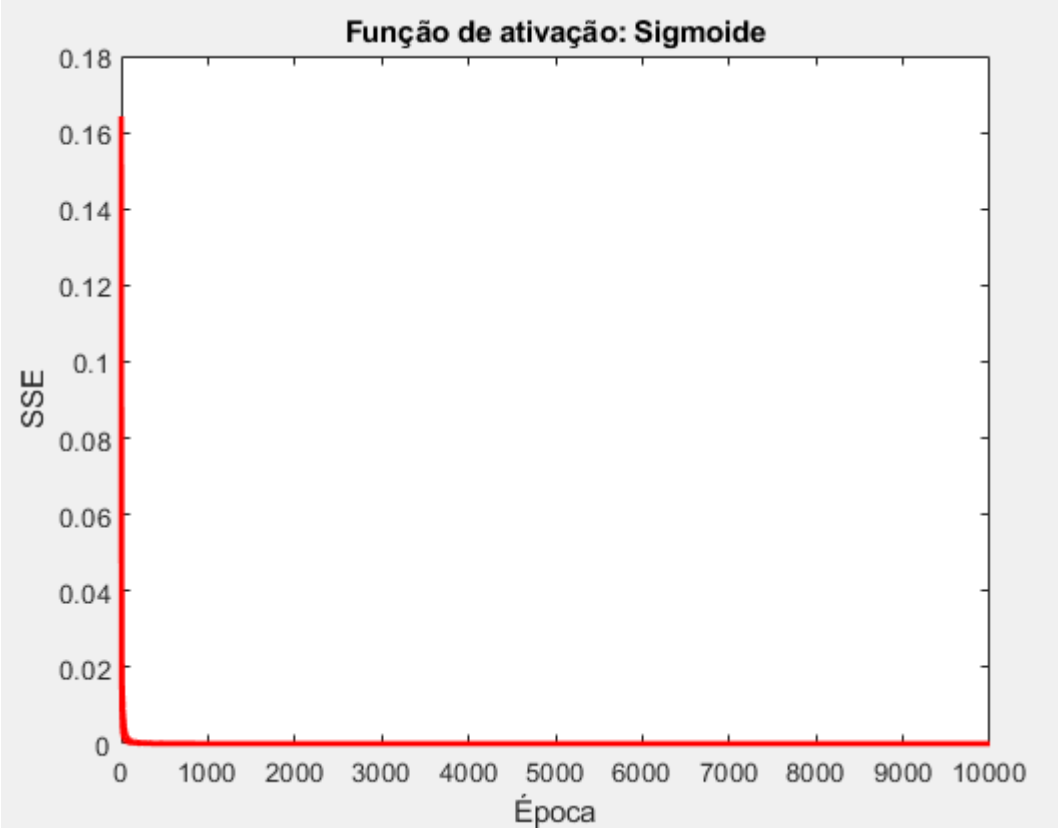

Figura 2.2.8 – Evolução da Soma dos Erros Quadráticos para o intervalo [-1; 1]

Na figura 2.2.8, pode ser observada a evolução da SSE, que, como pode ser visto, apenas demorou algumas épocas de treino até que atingisse um resultado extremamente próximo do ideal, 0.

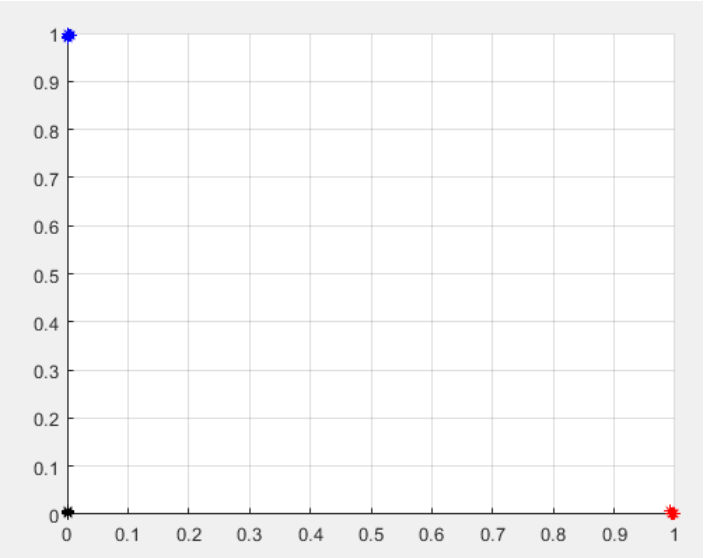

Figura 2.2.9 – Resultados Obtidos

Na figura acima observa,-se os resultados obtidos para as duas primeiras colunas da matriz dos dados obtidos. Verifica-se uma clara divisão por classes dos mesmos.

No caso da inicialização dos pesos no intervalo [-1; 1], verificou-se que a matriz dos resultados obtidos foi muito próxima dos resultados expectáveis, visto que a soma dos erros para cada uma das classes foi considerado muito baixo, o que é ótimo. Porém, houve uma diferença entre as duas redes neuronais quando sujeitas a intervalos maiores de peso. No caso da rede neuronal 3x3x1, como já foi visto previamente, quando esta foi sujeita a um intervalo mais amplo de pesos, houve um desfasamento maior nos resultados obtidos e uma diferença maior entre os resultados obtidos e os esperados, enquanto que, na rede 3x5x3, mesmo quando sujeitos a um intervalo de pesos maiores, esta conseguiu na mesma apresentar resultados relativamente próximos dos esperados:

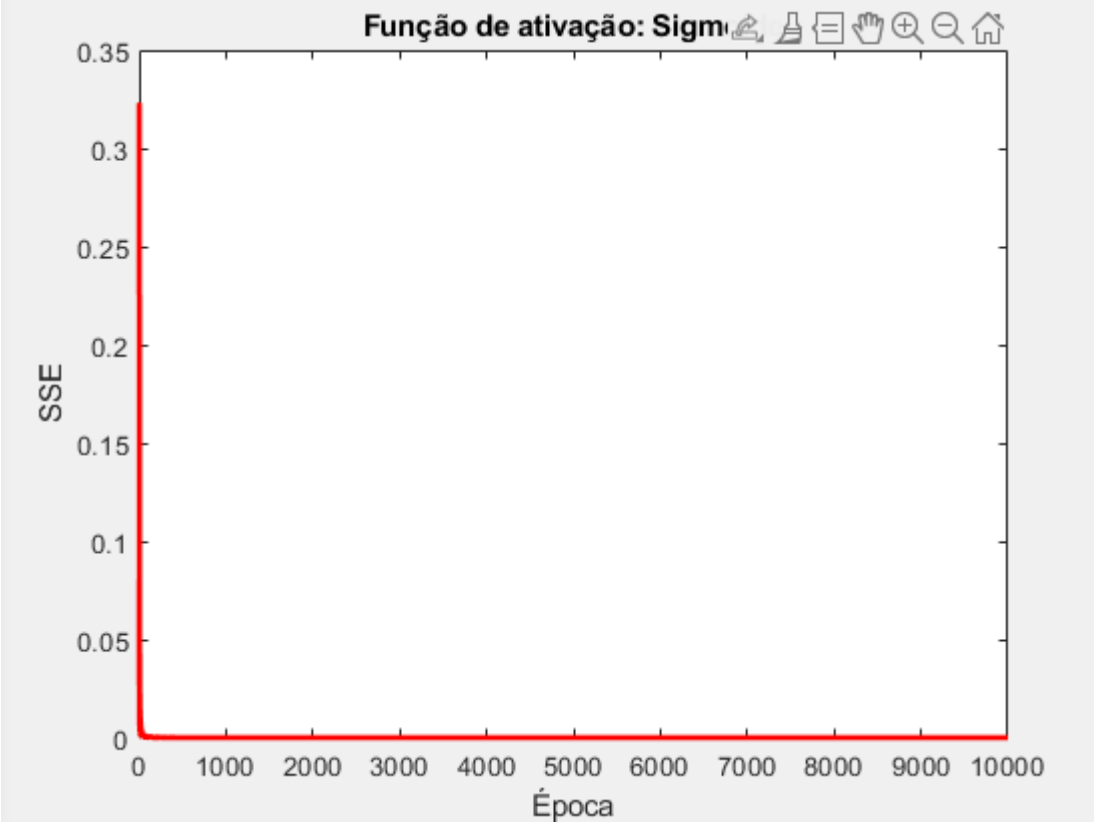

Figura 2.2.10 - Evolução da Soma dos Erros Quadráticos para o intervalo [-5; 5]

Como pode ser analisado no gráfico da evolução da soma dos erros quadráticos, este sofre apenas minúsculas, quase irrelevantes, alterações, tendo atingido na mesma um erro suficientemente pequeno para ser considerada aceitável.

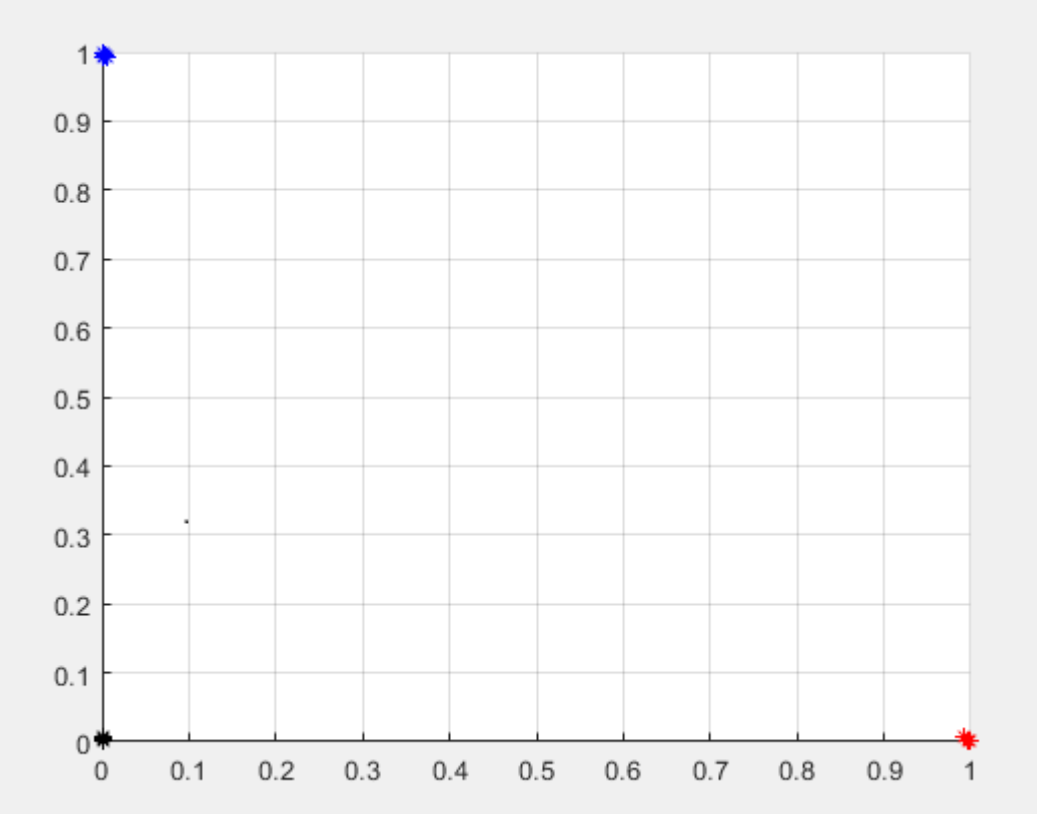

Figura 2.2.11 – Resultados Obtidos para o Intervalo [-5; 5]

Quanto aos resultados obtidos, foram virtualmente os mesmos, não havendo uma alteração significante o suficiente para ser documentada. A matriz da soma dos erros foi a seguinte: Classe A: 0,0004; Classe B: 0,0005; Classe C: 0,0003. Estes erros são considerados muito bons em termos de significância, ou seja, são erros muito pequenos, o que é muito bom pois significa que os resultados obtidos estão muito perto dos que eram esperados.

Ao analisar estes dois algoritmos, foi observado que a rede de topologia 3x3x1 é mais sensível a alterações de pesos enquanto que a rede de topologia 3x5x3 não observa alterações significantes nos resultados obtidos. No geral os resultados obtidos foram os esperados teoricamente.

3. Conclusão

Com base nas redes treinadas e testadas, é possível concluir que as redes neuronais artificiais são um avanço muito grande tanto na área da matemática como das engenharias em geral. Estas servem para fazer uma aproximação de uma função e comparar o resultado obtido com o esperado.

No caso deste trabalho, pode-se concluir que, dependendo das topologias das redes neuronais, o valor dos pesos vai também afetar o output das redes de formas diferentes. No caso da rede de topologia 3x3x1, o valor dos pesos irá ser muito mais significativo do que na rede 3x5x3, pois irá influenciar muito mais o resultado obtido do que nesta. Pode-se também observar que, nos resultados obtidos, a soma dos erros quadráticos foi diminuindo conforme o número de iterações em ambos os casos. Para a rede neuronal 3x5x3, o método foi um pouco diferente do método utilizado para desenvolver o algoritmo para a rede 3x3x1, visto que foram fornecidas classes diferentes, ou seja, em vez de apenas uma linha de erros quadráticos, por exemplo, tiveram de ser mantidos os resultados de cada uma das somas dos erros para os resultados dos treinos de cada uma das classes diferentes. No geral os resultados obtidos foram os esperados, visto que a matriz de output foi semelhante à matriz esperada no início do algoritmo.

4. Bibliografia

[1] Paulo Moura Oliveira, Eduardo Solteiro Pires (2020) Protocolo de Trabalho Prático: Redes Neuronais Artificiais

[2] Paulo Moura Oliveira, Eduardo Solteiro Pires (2020) Modelo para o Relatório dos Trabalhos

[3] Paulo Moura Oliveira, (2020), Redes Neuronais Artificiais – Part I

[4] Paulo Moura Oliveira, (2020), Redes Neuronais Artificiais – Part II

[5] Paulo Moura Oliveira, (2020), Redes Neuronais Artificiais – Part III

[6] Paulo Moura Oliveira, (2020), Redes Neuronais Artificiais – Part IV

[7] Paulo Moura Oliveira, (2020), Redes Neuronais Artificiais – Função Ou-Exclusivo (XOR)

[8] Rohan Kapur, (2016) – "Rohan & Lenny #1: Neural Networks & The Backpropagation Algorithm, Explained" https://ayearofai.com/rohan-lenny-1-neural-networks-the-backpropagation-algorithm-explained-abf4609d4f9d Acedido em 26-1-2021

Anexo A – Código da Rede Neuronal 3x3x1

```
n_epochs = 2000; %Numero de épocas (iterações)
alpha = 0.9; %Fator de aprendizagem

%Vetor soma dos erros quadráticos
SSE = zeros(1, n_epochs);

N = 4; %Número de amostras

%Matriz com as amostras de entrada da função XOR
X = [0 0 1;
     0 1 1;
     1 0 1;
     1 1 1];

%Saídas da função XOR
T = [ 0
      1
      1
      0 ];
%Inicialização aleatória dos pesos [-1,1]
%W1-2x3 W2- 1x3
W1 = 2*rand(2,3) - 1
W2 = 2*rand(1,3) - 1

for epoch = 1:n_epochs
    sum_sq_error=0;
    for k = 1:N
        x = X(k,:)';
        t = T(k);

        %Soma da camada de entrada
        g1 = W1*x;
        %Função de ativação sigmoidal
        y1 = sig(g1);

        %Adição à saída da camada escondida y1
        %da entrada de bias com +1
        %Resulta em y1_b
        y1_b = [y1
                1];

        %Soma da camada de saída
        g2 = W2*y1_b;
        %Função de ativação sigmoidal
        y2 = sig(g2);

        %Retropropagação
        %Erro da camada de saída
        e = t - y2;
        %Cálculo do delta da camada de saída
        %Sigmoide
        delta2 = y2.*(1-y2).*e;

        %Atualização da soma dos erros quadráticos
        sum_sq_error = sum_sq_error + e^2;
```

```
        %Erro da camada escondida
        e1 = W2'*delta2;
        %Erro sem o bias
        e1_b = e1(1:2);

        %Cálculo do delta da camada de saída
        delta1 = y1.*(1-y1).*e1_b;

        %Atualização dos pesos da camada escondida
        dW2 = alpha * delta2 * y1_b'; %Com bias
        W2 = W2 + dW2;

        %Atualização dos pesos da camada de entrada
        dW1 = alpha * delta1 * x';
        W1 = W1 + dW1;

    end
    SSE(epoch) = (sum_sq_error)/N;
end

%Teste da rede
for k = 1:N

    x = X(k,:)';
    g1 = W1*x;

    %Sigmóide
    y1 = sig(g1);

    %y1 mais uma entrada de bias
    y1_b = [y1
            1];

    g2 = W2*y1_b;

    %Saída prevista XOR
    y_plot(k) = sig(g2);
end
y_plot

It = 1 :1:n_epochs;
plot(It, SSE, 'r-', 'LineWidth', 2)
xlabel('Época')
ylabel('SSE')
title('Função de ativação: Sigmoide')
```

Anexo B – Código da Rede Neuronal 3x5x3

```
n_epochs = 10000; %Numero de épocas (iterações)
alpha = 0.9; %Fator de aprendizagem

%Vetor soma dos erros quadráticos
SSE = zeros(3, n_epochs);

N=120; %Numero de amostras
%Amostras de entrada funçao XOR
Data_multi
X = [ X_A;
      X_B;
      X_C;
      ];

X=[X,ones(N,1)]; %X_Testes

%X=X_teste;

 %T 120x3
 %Primeiras 40 linhas 1° coluna a 1s
 %Segundas 40 linhas 2° coluna a 1s
 %Utlimas 40 linhas 3° coluna a 1s
 %Inicialização aleatoria dos pesos [-1,1]
 %Saidas da funçao XOR

T= [ones(40,1),zeros(40,2);
    zeros(40,1),ones(40,1),zeros(40,1);
    zeros(40,2),ones(40,1);
    ];
%W1-4X3 W2- 3X5
W1 = 2*rand(4,3) - 1;
W2 = 2*rand (3,5) - 1;

for epoch = 1:n_epochs
    sum_sq_error=0;
    for k = 1:N
        x = X(k,:)';
        t = T(k,:)';

        %Soma da camada de entrada
        g1 = W1*x;
        %Função de ativação sigmoidal
        y1 = sig(g1);

        %Adição à saída da camada escondida y1
        %da entrada de bias com +1
        %Resulta em y1_b
        y1_b = [y1
                1];

        %Soma da camada de saída
        g2 = W2*y1_b;
        %Função de ativação sigmoidal
        y2 = sig(g2);
```

```
        %Retropropagação
        %Erro da camada de saída
        e = t - y2;
        %Cálculo do delta da camada de saída
        %Sigmoide
        delta2 = y2.*(1-y2).*e;

        equadrado = e.^2;

        %Atualização da soma dos erros quadráticos
        sum_sq_error = sum_sq_error + equadrado;

        %Erro da camada escondida
        e1 = W2'*delta2;
        %Erro sem o bias
        e1_b = e1(1:4);

        %Cálculo do delta da camada de saída
        delta1 = y1.*(1-y1).*e1_b;

        %Atualização dos pesos da camada escondida
        dW2 = alpha * delta2 * y1_b'; %Com bias
        W2 = W2 + dW2;

        %Atualização dos pesos da camada de entrada
        dW1 = alpha * delta1 * x';
        W1 = W1 + dW1;

    end
    A = (sum_sq_error)/N;
    SSE(1,epoch) = A(1,1);
    SSE(2,epoch) = A(2,1);
    SSE(3,epoch) = A(3,1);
end

%Teste da rede
for k = 1:N

    x = X(k,:)';
    g1 = W1*x;

    %Sigmóide
    y1 = sig(g1);

    %y1 mais uma entrada de bias
    y1_b = [y1
            1];

    g2 = W2*y1_b;

    %Saída prevista XOR
    %y_plot(k) = sig(g2);

    y_plot(k,:) = sig(g2)';


end
```

```
y_plot

It = 1 :1:n_epochs;
plot(It, SSE, 'r-', 'LineWidth', 2)
xlabel('Época')
ylabel('SSE')
title('Função de ativação: Sigmoide')
hold off

figure
hold on
grid on
Data_multi
 plot(X_A(:,1), X_A(:,2), 'r*')
 plot(X_B(:,1), X_B(:,2), 'b*')
 plot(X_C(:,1), X_C(:,2), 'k*')
 plot(X_teste(:,1), X_teste(:,2), 'go')
hold off

figure
hold on
grid on
for j = 1:120
    if j > 0 && j <= 40
    plot(y_plot(j,1), y_plot(j,2), 'r*')
    end
    if j > 40 && j <= 80
    plot(y_plot(j,1), y_plot(j,2), 'b*')
    end
    if j > 80 && j <= 120
    plot(y_plot(j,1), y_plot(j,2), 'k*')
    end
end
```