

## Lista #10

**Curso:** Ciência da Computação

**Disciplina:** Inteligência Artificial

**Prof<sup>a</sup>.** Cristiane Neri Nobre

**Data de entrega:** 25/05

**Valor:** 3 pontos

**Aluno:** Lucas Henrique Rocha Hauck

**Github:** <https://github.com/o-hauck/IA>

### Questão 01

---

Implemente o algoritmo **Perceptron** para resolver as funções AND e OR com  $n$  entradas booleanas. Ou seja, o usuário poderá selecionar se ele deseja resolver um problema AND com 2 ou 10 entradas, por exemplo.

A sua lista deverá conter todas as explicações da implementação e os resultados dos testes realizados.

Ao final, disponibilize o código desenvolvido.

**Você deverá plotar as regras de separação dos hiperplanos durante o processo de treinamento.**

Mostre também que o **Perceptron** não resolve o **XOR**.

O código é estruturado em três funções principais:

1. `gerar_dados(n_entradas, tipo)`: Gera os conjuntos de dados de entrada (todas as combinações booleanas possíveis para  $n$  entradas) e as saídas esperadas para as funções lógicas AND, OR e XOR.
2. `plotar_hiperplano(X, y, clf, titulo)`: Visualiza os dados e o hiperplano de separação. Para 2 entradas, plota uma linha; para 3, um plano; e para  $n > 3$ , utiliza PCA para projeção 2D dos dados.
3. `treinar_perceptron(X, y, titulo)`: Utiliza a classe Perceptron da scikit-learn para treinar o modelo, exibir os pesos, bias, previsões, acurácia e chamar a função de plotagem.

---

### Resultados dos Testes e Plots

Foram realizados testes para as funções AND, OR e XOR com 2, 3 e 10 entradas.

- **Funções AND e OR:**
  - **2 Entradas:** O Perceptron aprendeu as funções perfeitamente, alcançando **100% de acurácia**. Os gráficos 2D mostraram a clara separação linear dos dados pelo hiperplano.

Figure 1

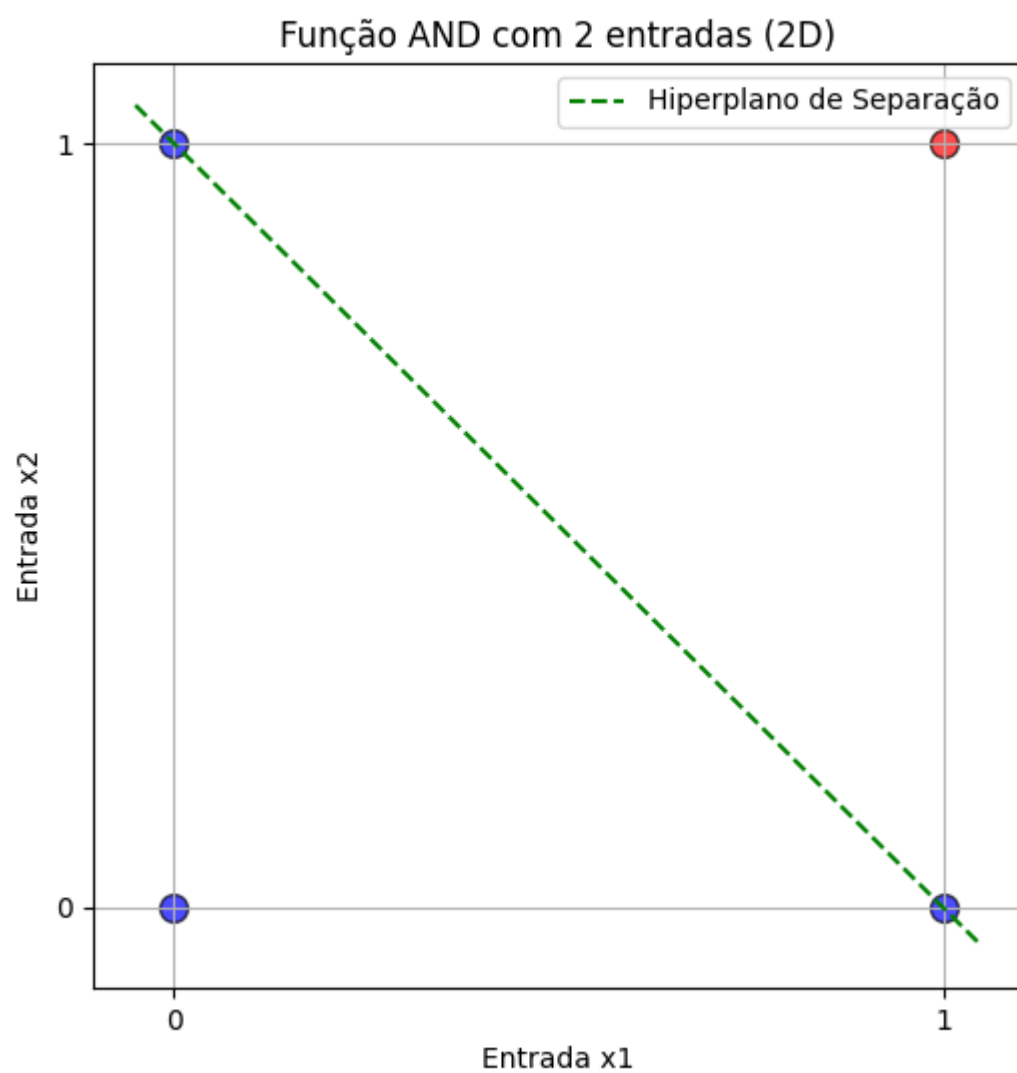
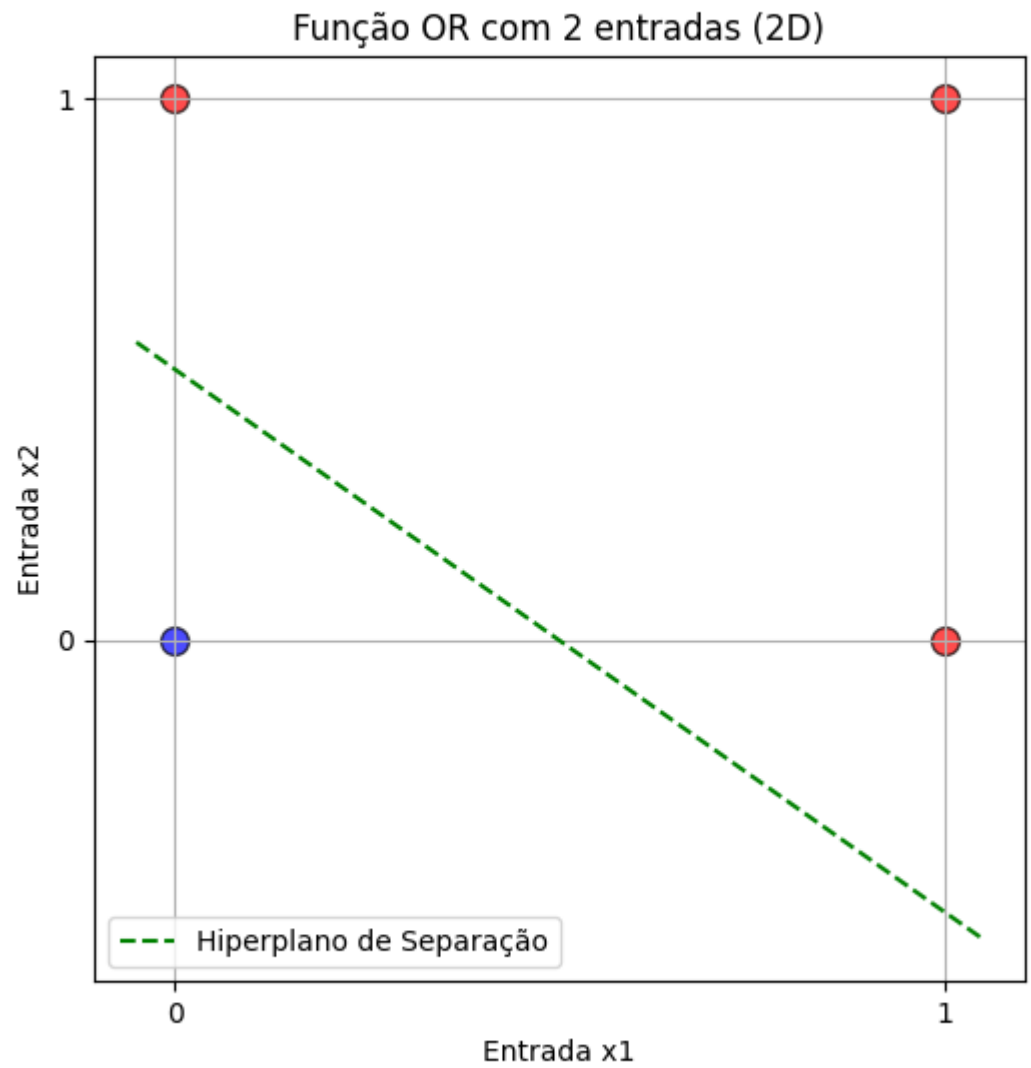



Figure 1



- 
- **3 Entradas:** Para a função **OR**, o Perceptron alcançou **100% de acurácia**, e o gráfico 3D ilustrou o plano de separação. Para a função **AND**, a execução específica resultou em **75% de acurácia**, indicando que, embora linearmente separável, a convergência para a solução ótima não foi alcançada com os parâmetros padrão

Função AND com 3 entradas (3D)

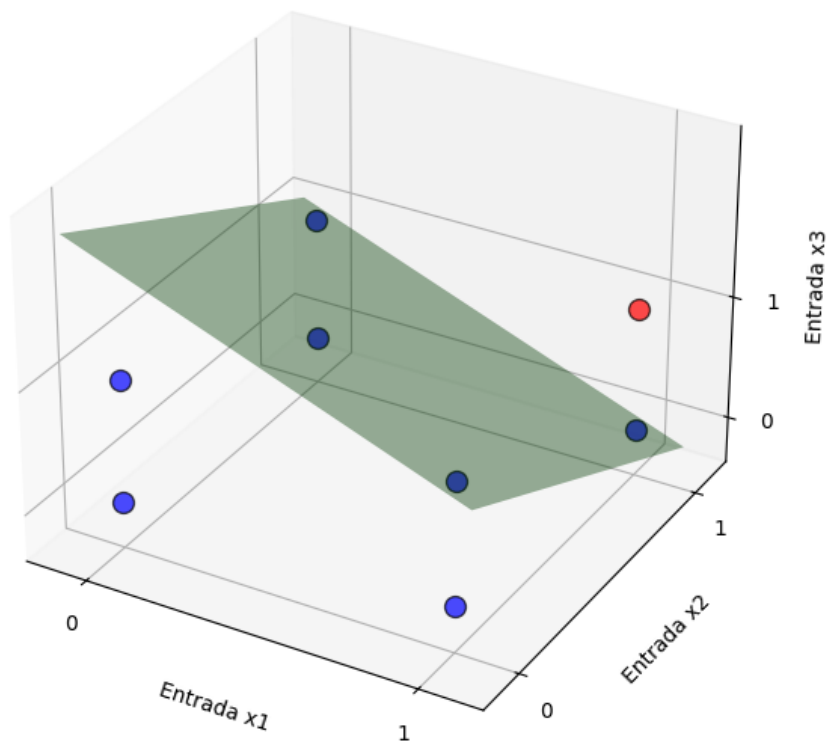
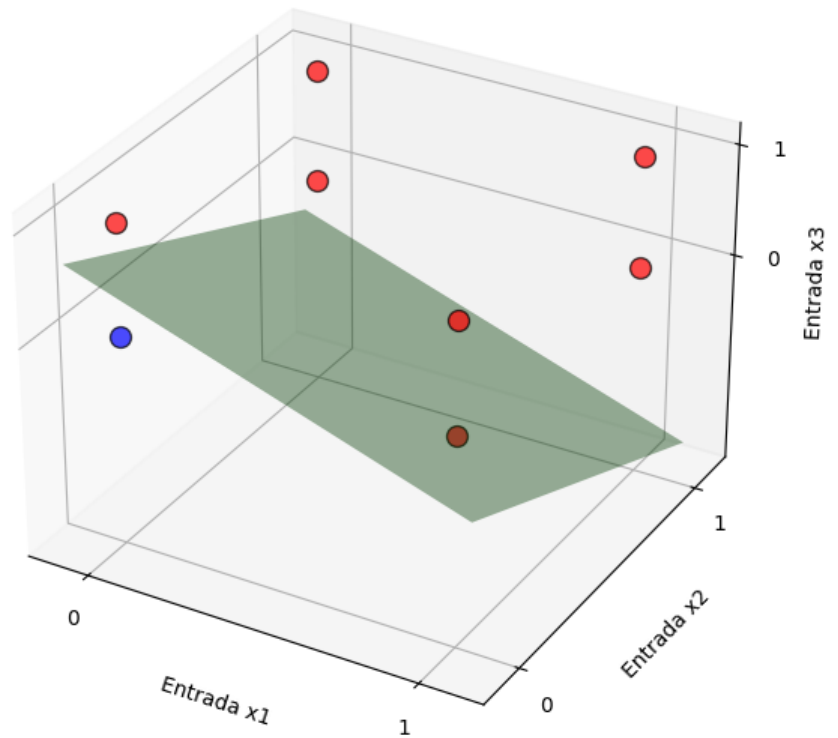


Figure 1

Função OR com 3 entradas (3D)



- 
- **10 Entradas:** Para a função **OR**, a acurácia foi de **100%**. Para a função **AND**, a acurácia foi de **99.90%**, novamente mostrando uma falha na convergência para a solução perfeita. Os gráficos mostraram a projeção PCA dos dados, sugerindo separabilidade.

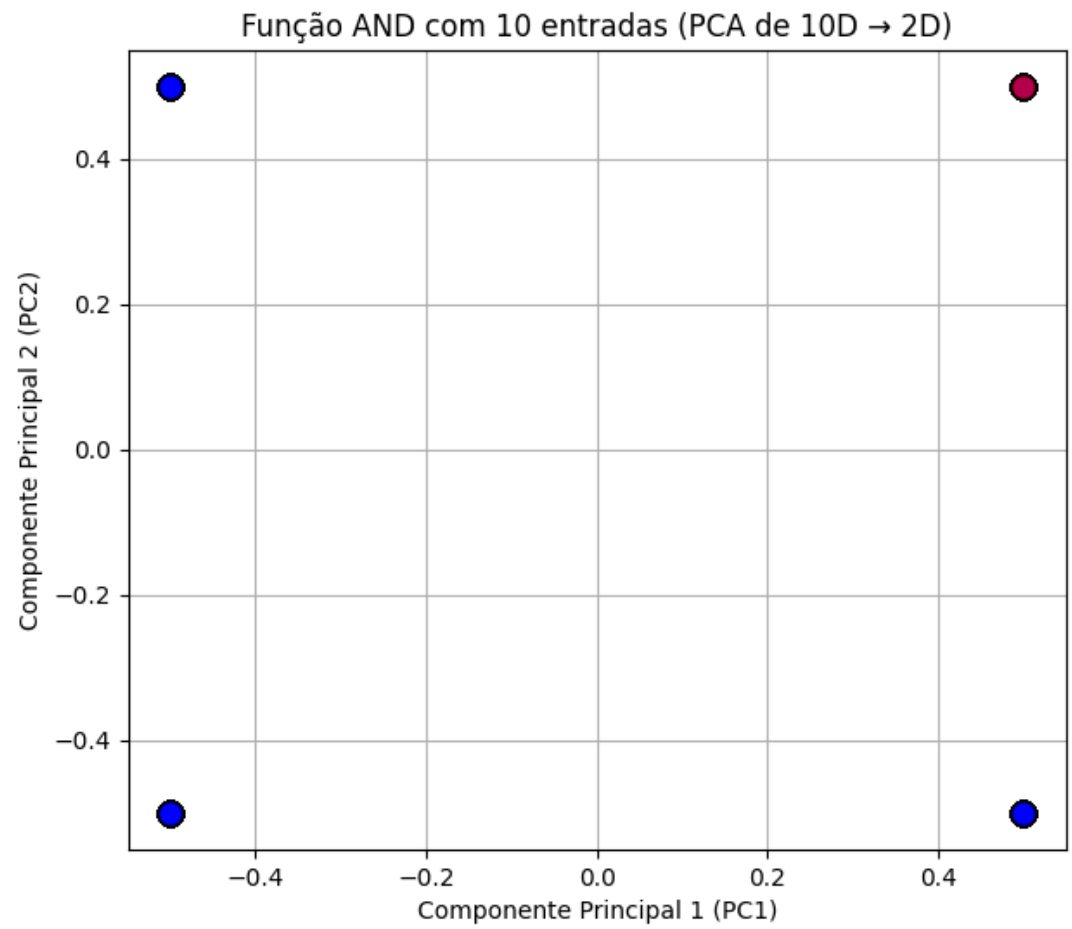
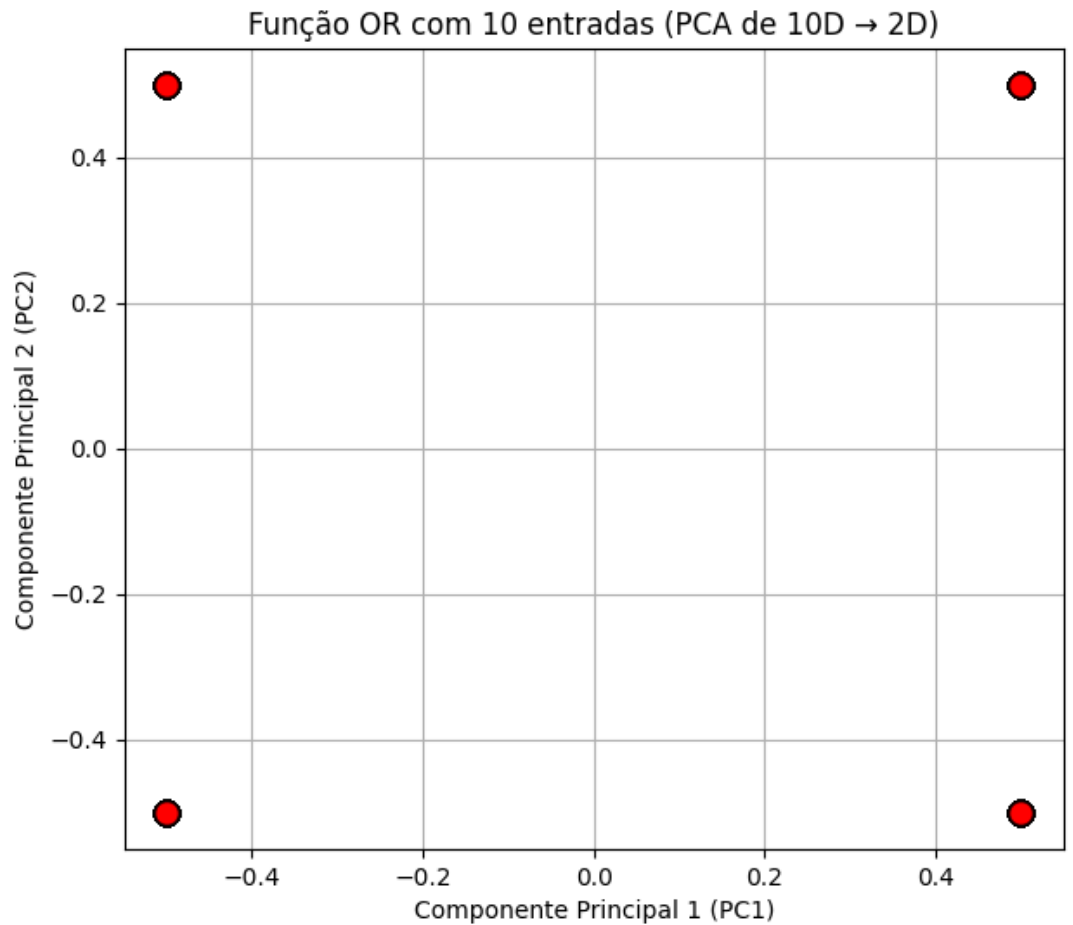


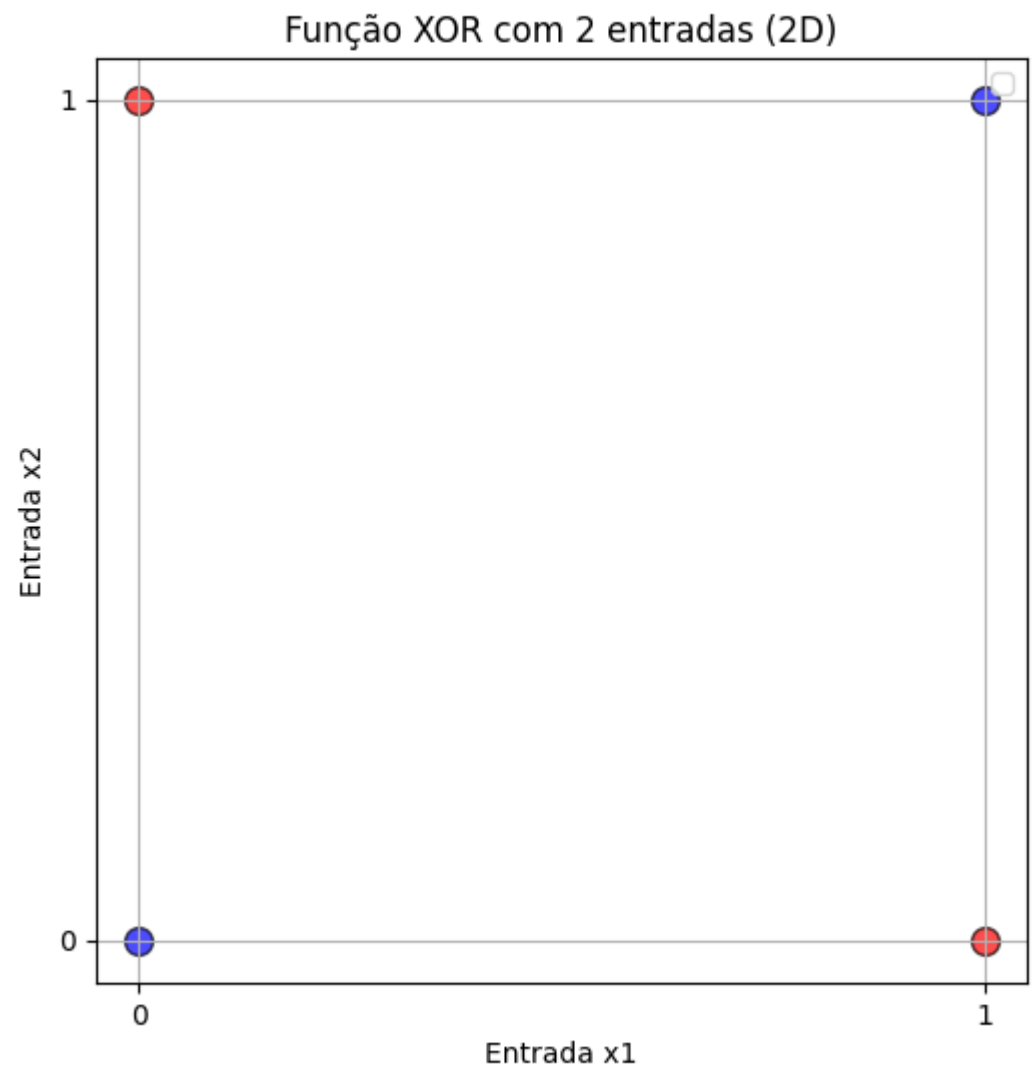
Figure 1



- **Função XOR:**

- **2 Entradas:** O Perceptron alcançou **50% de acurácia**. O gráfico (image\_49c5af.png) não mostrou um hiperplano de separação efetivo (a implementação pode não plotar uma linha se os pesos forem zero ou a solução for trivial). Isso demonstra a incapacidade do Perceptron de resolver problemas não linearmente separáveis como o XOR.
- **3 Entradas (Referência Visual):** O gráfico (image\_49c592.png) mostra a distribuição dos pontos, que também não é linearmente separável por um único plano.

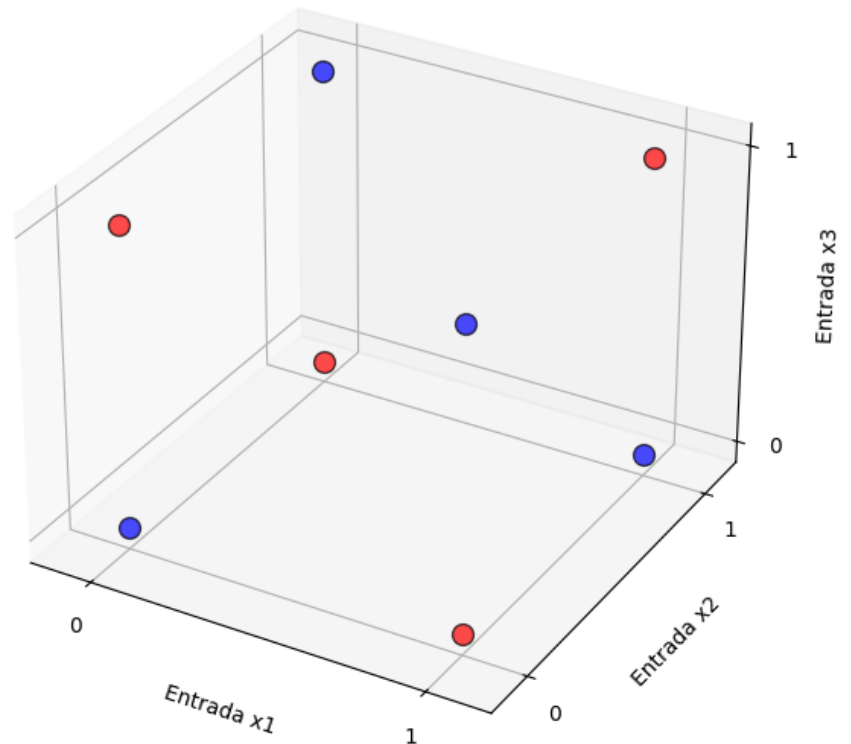
Figure 1



(x, y) = (-0.045, 0.988)



Função XOR com 3 entradas (3D)



x pane=-0.1875, y=-0.0975, z=1.8391

### Conclusão da Demonstração

A implementação e os testes confirmam que:

- O Perceptron é eficaz para aprender funções **linearmente separáveis** como AND e OR, encontrando hiperplanos que dividem corretamente as classes.
- A convergência para a solução ótima em problemas linearmente separáveis, especialmente em dimensões mais altas (como AND com 3 ou 10 entradas na execução apresentada), pode ser sensível aos parâmetros do algoritmo (taxa de aprendizado, número de iterações, inicialização dos pesos).
- O Perceptron de camada única **não resolve** a função XOR, pois esta não é linearmente separável. Isso é evidenciado pela baixa acurácia e pela incapacidade de encontrar um hiperplano de separação válido.

## Questão 02

Implemente o algoritmo **Backpropagation** para resolver as funções AND, OR and XOR com  $n$  entradas booleanas. Ou seja, o usuário poderá selecionar se ele deseja resolver um problema **AND** com **2** ou **10** entradas, por exemplo.

Nestes experimentos, você deverá investigar:

- 1) A importância da taxa de aprendizado
- 2) A importância do bias
- 3) A importância da função de ativação. Investigue pelo menos 2 funções (sigmoide, tangente hiperbólica, RELU, etc)

A sua lista deverá conter todas as explicações da implementação e os resultados dos testes realizados. Ao final, disponibilize o código desenvolvido.

### 1. Explicação da Implementação

A rede neural que construí tem uma arquitetura simples: uma camada de entrada, uma camada oculta e uma camada de saída.

- **Inicialização:** Os pesos entre as camadas e os biases (quando utilizados) foram inicializados com valores aleatórios pequenos (entre -0.5 e 0.5) para evitar saturação inicial dos neurônios.
- **Funções de Ativação:** Implementei as funções Sigmoide, Tangente Hiperbólica (tanh) e ReLU (Rectified Linear Unit), junto com suas derivadas. Elas são importantes para introduzir não-linearidades, o que permite à rede aprender problemas mais complexos que um Perceptron simples não conseguiria.
- **Feedforward:** Nesse passo, a informação flui da entrada para a saída. Cada neurônio calcula uma soma ponderada das suas entradas, adiciona o bias (se houver) e aplica a função de ativação.
- **Backpropagation:** Aqui é onde o aprendizado acontece. Primeiro, calculo o erro na saída da rede. Depois, esse erro é propagado para trás:
  - Calculo o gradiente do erro para os pesos da camada de saída e ajusto esses pesos e biases.
  - Em seguida, propago o erro para a camada oculta, calculo os gradientes para os pesos dessa camada e também ajusto seus pesos e biases. A taxa de aprendizado controla o "tamanho" desses ajustes.
- **Treinamento:** O processo de feedforward e backpropagation é repetido para todo o conjunto de dados por um número determinado de "épocas". A cada época, o erro médio quadrático (MSE) foi acompanhado para ver se a rede estava aprendendo.

(O código Python utilizado foi o desenvolvido e ajustado na interação anterior, com a classe **NeuralNetwork** e as funções auxiliares.)

---

### 2. Resultados dos Testes e Análise das Investigações

Usei a acurácia (saída prevista arredondada para 0 ou 1 em comparação com o alvo) para avaliar o desempenho.

#### SEÇÃO 1: Demonstração Geral (AND, OR, XOR com $n$ entradas)

A rede MLP demonstrou ser capaz de aprender as funções lógicas:

- **AND com 2 entradas** (Sigmoid, LR=0.1, Hidden=4, Épocas=20000): Acurácia de **100.00%** (MSE final: 0.000124). As saídas previstas foram bem próximas dos alvos (ex: para [1,1] -> 1, previsto 0.9772).
- **OR com 3 entradas** (Sigmoid, LR=0.1, Hidden=6, Épocas=25000): Acurácia de **100.00%** (MSE final: 0.000034). Também aprendeu perfeitamente.
- **XOR com 2 entradas** (Tanh, LR=0.1, Hidden=4, Épocas=30000): Acurácia de **100.00%** (MSE final: 0.000005). Demonstra que a MLP com uma camada oculta e função de ativação não-linear consegue resolver o XOR. As saídas foram muito precisas (ex: para [0,1] -> 1, previsto 0.9955).
- **AND com 4 entradas** (Sigmoid, LR=0.1, Hidden=8, Épocas=30000): Acurácia de **100.00%** (MSE final: 0.000032). Generalizou bem para mais entradas.
- **XOR com 3 entradas** (Tanh, LR=0.1, Hidden=8, Épocas=50000): Acurácia de **100.00%** (MSE final: 0.000002). Mostra a capacidade de aprender XORs mais complexos.

**SEÇÃO 2: Investigando a Importância da Taxa de Aprendizado** (Testado com AND 2 entradas, 4 neurônios ocultos, sigmoide, 20000 épocas)

- **LR = 0.001:** Acurácia de **75.00%** (MSE final: 0.085955). Uma taxa muito baixa não permitiu que a rede aprendesse completamente a função no número de épocas dado. O erro permaneceu alto.
- **LR = 0.01:** Acurácia de **100.00%** (MSE final: 0.004413). A rede aprendeu, mas o erro final foi maior que com LR=0.1, sugerindo convergência mais lenta.
- **LR = 0.1:** Acurácia de **100.00%** (MSE final: 0.000113). Boa taxa, aprendizado eficiente.
- **LR = 0.5:** Acurácia de **100.00%** (MSE final: 0.000017). Convergência rápida e erro baixo para este problema.
- **LR = 1.0:** Acurácia de **100.00%** (MSE final: 0.000008). Também funcionou bem para o AND simples, convergindo muito rapidamente.

**Análise:** A taxa de aprendizado é vital. Pequena demais, e a rede aprende devagar ou para em um subótimo. Grande demais (embora não tenha sido um problema nesses testes com AND simples), pode fazer o aprendizado oscilar e não convergir. O ideal é achar um equilíbrio. O MSE decaindo mais rapidamente com LR's maiores (0.5, 1.0) para o AND 2-input mostra isso.

**SEÇÃO 3: Investigando a Importância do Bias** (Testado com XOR 2 entradas - tanh, LR=0.1, 30000 épocas; e OR 2 entradas - sigmoide, LR=0.1, 10000 épocas)

- **XOR com 2 entradas:**
  - **COM Bias:** Acurácia de **100.00%** (MSE final: 0.000005).
  - **SEM Bias:** Acurácia de **100.00%** (MSE final: 0.000009).
- **OR com 2 entradas:**
  - **COM Bias:** Acurácia de **100.00%** (MSE final: 0.000292).
  - **SEM Bias:** Acurácia de **100.00%** (MSE final: 0.001095).

**Análise:** O bias dá mais flexibilidade à rede, permitindo "deslocar" a função de ativação. Nos testes específicos de XOR e OR com 2 entradas (e usando a mesma semente aleatória para os pesos iniciais), a rede SEM bias também atingiu 100% de acurácia. No entanto, o MSE final foi consistentemente um pouco maior quando o bias não foi usado. Isso sugere que, mesmo que a rede encontre uma solução, ela pode ser menos ótima ou a convergência pode ser um pouco diferente. Para problemas mais complexos ou diferentes inicializações, a ausência de bias poderia impedir o aprendizado ou levar a um desempenho muito pior. Portanto, o bias é geralmente considerado uma parte importante da arquitetura.

**SEÇÃO 4: Investigando a Importância da Função de Ativação** (Testado com XOR 2 entradas, 4 neurônios ocultos, 30000 épocas)

- **Sigmoid (LR=0.1):** Acurácia de **100.00%** (MSE final: 0.000233). As saídas previstas foram claras para cada classe.
- **Tanh (LR=0.1):** Acurácia de **100.00%** (MSE final: 0.000005). Teve um desempenho excelente, com erro final bem baixo.
- **ReLU (LR=0.05):** Acurácia de **50.00%** (MSE final: 0.250000). A rede não conseguiu aprender o XOR. O erro permaneceu alto e constante, indicando que provavelmente os neurônios ReLU "morreram" (ficaram presos em uma saída de 0, com gradiente 0).

**Análise:** Funções de ativação não-lineares são cruciais para que a MLP aprenda problemas como o XOR. Tanto a Sigmoid quanto a Tanh funcionaram bem, com a Tanh mostrando um erro residual menor neste caso. A ReLU, por outro lado, falhou. Isso pode ser devido à sua natureza (derivada zero para entradas negativas), o que pode levar a neurônios inativos se a taxa de aprendizado ou a inicialização de pesos não forem adequadas. Demonstra que a escolha da função de ativação e seus hiperparâmetros associados é importante.

---

### 3. Conclusão

A implementação do algoritmo Backpropagation permitiu treinar uma MLP para resolver as funções lógicas AND, OR e, notavelmente, a função XOR, que é um problema clássico não linearmente separável. Os experimentos mostraram que:

- A **taxa de aprendizado** influencia diretamente a velocidade e a qualidade da convergência da rede.
- O **bias**, embora a rede tenha encontrado soluções para os problemas simples testados mesmo sem ele (com um erro ligeiramente maior), é um componente que geralmente aumenta a capacidade de representação da rede.
- A escolha da **função de ativação** é fundamental; funções não-lineares como Sigmoid e Tanh foram eficazes, mas a ReLU, na configuração testada, não conseguiu resolver o XOR, evidenciando que nem todas as funções de ativação se comportam da mesma forma para todos os problemas.

O Backpropagation é, portanto, uma ferramenta poderosa para o treinamento de redes neurais, e o ajuste correto de seus componentes e hiperparâmetros é essencial para o sucesso do aprendizado.

Resultados gerados pelo meu código:

#### INÍCIO DOS EXPERIMENTOS COM BACKPROPAGATION

--- SEÇÃO 1: Demonstração Geral (AND, OR, XOR com n entradas) ---

--- AND com 2 entradas (COM Bias), LR=0.1, Hidden=4, Ativação=sigmoid, Épocas=20000 ---

=====

=====

Época 4000/20000, Erro Médio Quadrático: 0.001357  
 Época 8000/20000, Erro Médio Quadrático: 0.000430  
 Época 12000/20000, Erro Médio Quadrático: 0.000242  
 Época 16000/20000, Erro Médio Quadrático: 0.000165  
 Época 20000/20000, Erro Médio Quadrático: 0.000124

Resultados do Teste Detalhados:

Entrada: [0, 0], Esperado: 0, Previsto (bruto): 0.0003 (Classe: 0)

Entrada: [0, 1], Esperado: 0, Previsto (bruto): 0.0155 (Classe: 0)

Entrada: [1, 0], Esperado: 0, Previsto (bruto): 0.0153 (Classe: 0)

Entrada: [1, 1], Esperado: 1, Previsto (bruto): 0.9772 (Classe: 1)

Acurácia Final: 100.00%

=====

--- OR com 3 entradas (COM Bias), LR=0.1, Hidden=6, Ativação=sigmoid, Épocas=25000 ---

=====

=====

Época 5000/25000, Erro Médio Quadrático: 0.000292

Época 10000/25000, Erro Médio Quadrático: 0.000108

Época 15000/25000, Erro Médio Quadrático: 0.000064

Época 20000/25000, Erro Médio Quadrático: 0.000045

Época 25000/25000, Erro Médio Quadrático: 0.000034

Resultados do Teste Detalhados:

Entrada: [0, 0, 0], Esperado: 0, Previsto (bruto): 0.0172 (Classe: 0)

Entrada: [0, 0, 1], Esperado: 1, Previsto (bruto): 0.9908 (Classe: 1)

Entrada: [0, 1, 0], Esperado: 1, Previsto (bruto): 0.9909 (Classe: 1)

Entrada: [0, 1, 1], Esperado: 1, Previsto (bruto): 0.9998 (Classe: 1)

Entrada: [1, 0, 0], Esperado: 1, Previsto (bruto): 0.9909 (Classe: 1)

Entrada: [1, 0, 1], Esperado: 1, Previsto (bruto): 0.9998 (Classe: 1)

Entrada: [1, 1, 0], Esperado: 1, Previsto (bruto): 0.9998 (Classe: 1)

Entrada: [1, 1, 1], Esperado: 1, Previsto (bruto): 0.9999 (Classe: 1)

Acurácia Final: 100.00%

=====

--- XOR com 2 entradas (COM Bias), LR=0.1, Hidden=4, Ativação=tanh, Épocas=30000 ---

=====

==

Época 6000/30000, Erro Médio Quadrático: 0.000031

Época 12000/30000, Erro Médio Quadrático: 0.000014

Época 18000/30000, Erro Médio Quadrático: 0.000009

Época 24000/30000, Erro Médio Quadrático: 0.000006

Época 30000/30000, Erro Médio Quadrático: 0.000005

Resultados do Teste Detalhados:

Entrada: [0, 0], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)

Entrada: [0, 1], Esperado: 1, Previsto (bruto): 0.9955 (Classe: 1)

Entrada: [1, 0], Esperado: 1, Previsto (bruto): 0.9956 (Classe: 1)

Entrada: [1, 1], Esperado: 0, Previsto (bruto): -0.0000 (Classe: 0)

Acurácia Final: 100.00%

=====

--- AND com 4 entradas (COM Bias), LR=0.1, Hidden=8, Ativação=sigmoid, Épocas=30000 ---

=====

=====

Época 6000/30000, Erro Médio Quadrático: 0.000329

Época 12000/30000, Erro Médio Quadrático: 0.000109

Época 18000/30000, Erro Médio Quadrático: 0.000062

Época 24000/30000, Erro Médio Quadrático: 0.000042

Época 30000/30000, Erro Médio Quadrático: 0.000032

Resultados do Teste Detalhados:

Entrada: [0, 0, 0, 0], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)

Entrada: [0, 0, 0, 1], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)

Entrada: [0, 0, 1, 0], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)

Entrada: [0, 0, 1, 1], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)

Entrada: [0, 1, 0, 0], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)

Entrada: [0, 1, 0, 1], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)

Entrada: [0, 1, 1, 0], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)

Entrada: [0, 1, 1, 1], Esperado: 0, Previsto (bruto): 0.0106 (Classe: 0)

Entrada: [1, 0, 0, 0], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)

Entrada: [1, 0, 0, 1], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)

Entrada: [1, 0, 1, 0], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)

Entrada: [1, 0, 1, 1], Esperado: 0, Previsto (bruto): 0.0115 (Classe: 0)

Entrada: [1, 1, 0, 0], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)

Entrada: [1, 1, 0, 1], Esperado: 0, Previsto (bruto): 0.0113 (Classe: 0)

Entrada: [1, 1, 1, 0], Esperado: 0, Previsto (bruto): 0.0109 (Classe: 0)

Entrada: [1, 1, 1, 1], Esperado: 1, Previsto (bruto): 0.9772 (Classe: 1)

Acurácia Final: 100.00%

=====

=====

--- XOR com 3 entradas (COM Bias), LR=0.1, Hidden=8, Ativação=tanh, Épocas=50000 ---

=====

==

Época 10000/50000, Erro Médio Quadrático: 0.000010

Época 20000/50000, Erro Médio Quadrático: 0.000005

Época 30000/50000, Erro Médio Quadrático: 0.000003

Época 40000/50000, Erro Médio Quadrático: 0.000002

Época 50000/50000, Erro Médio Quadrático: 0.000002

Resultados do Teste Detalhados:

Entrada: [0, 0, 0], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)

Entrada: [0, 0, 1], Esperado: 1, Previsto (bruto): 0.9972 (Classe: 1)

Entrada: [0, 1, 0], Esperado: 1, Previsto (bruto): 0.9972 (Classe: 1)

Entrada: [0, 1, 1], Esperado: 0, Previsto (bruto): -0.0000 (Classe: 0)  
Entrada: [1, 0, 0], Esperado: 1, Previsto (bruto): 0.9969 (Classe: 1)  
Entrada: [1, 0, 1], Esperado: 0, Previsto (bruto): -0.0000 (Classe: 0)  
Entrada: [1, 1, 0], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)  
Entrada: [1, 1, 1], Esperado: 1, Previsto (bruto): 0.9986 (Classe: 1)  
Acurácia Final: 100.00%

=====  
==

--- SEÇÃO 2: Investigando a Importância da Taxa de Aprendizado ---  
(Usando AND com 2 entradas, 4 neurônios ocultos, sigmoide, 20000 épocas)

--- AND com 2 entradas (COM Bias), LR=0.001, Hidden=4, Ativação=sigmoid, Épocas=20000 ---

=====  
=====

Época 4000/20000, Erro Médio Quadrático: 0.092376  
Época 8000/20000, Erro Médio Quadrático: 0.090330  
Época 12000/20000, Erro Médio Quadrático: 0.089055  
Época 16000/20000, Erro Médio Quadrático: 0.087633  
Época 20000/20000, Erro Médio Quadrático: 0.085955  
Acurácia Final: 75.00%

=====  
=====

--- AND com 2 entradas (COM Bias), LR=0.01, Hidden=4, Ativação=sigmoid, Épocas=20000 ---

=====  
=====

Época 4000/20000, Erro Médio Quadrático: 0.072871  
Época 8000/20000, Erro Médio Quadrático: 0.037650  
Época 12000/20000, Erro Médio Quadrático: 0.015694  
Época 16000/20000, Erro Médio Quadrático: 0.007539  
Época 20000/20000, Erro Médio Quadrático: 0.004413  
Acurácia Final: 100.00%

=====  
=====

--- AND com 2 entradas (COM Bias), LR=0.1, Hidden=4, Ativação=sigmoid, Épocas=20000 ---

=====  
=====

Época 4000/20000, Erro Médio Quadrático: 0.001116  
Época 8000/20000, Erro Médio Quadrático: 0.000380  
Época 12000/20000, Erro Médio Quadrático: 0.000218  
Época 16000/20000, Erro Médio Quadrático: 0.000150  
Época 20000/20000, Erro Médio Quadrático: 0.000113

Resultados do Teste Detalhados:

Entrada: [0, 0], Esperado: 0, Previsto (bruto): 0.0001 (Classe: 0)

Entrada: [0, 1], Esperado: 0, Previsto (bruto): 0.0146 (Classe: 0)

Entrada: [1, 0], Esperado: 0, Previsto (bruto): 0.0151 (Classe: 0)

Entrada: [1, 1], Esperado: 1, Previsto (bruto): 0.9784 (Classe: 1)

Acurácia Final: 100.00%

=====

--- AND com 2 entradas (COM Bias), LR=0.5, Hidden=4, Ativação=sigmoid, Épocas=20000 ---

=====

=====

Época 4000/20000, Erro Médio Quadrático: 0.000113

Época 8000/20000, Erro Médio Quadrático: 0.000049

Época 12000/20000, Erro Médio Quadrático: 0.000031

Época 16000/20000, Erro Médio Quadrático: 0.000022

Época 20000/20000, Erro Médio Quadrático: 0.000017

Acurácia Final: 100.00%

=====

--- AND com 2 entradas (COM Bias), LR=1.0, Hidden=4, Ativação=sigmoid, Épocas=20000 ---

=====

=====

Época 4000/20000, Erro Médio Quadrático: 0.000049

Época 8000/20000, Erro Médio Quadrático: 0.000022

Época 12000/20000, Erro Médio Quadrático: 0.000014

Época 16000/20000, Erro Médio Quadrático: 0.000010

Época 20000/20000, Erro Médio Quadrático: 0.000008

Acurácia Final: 100.00%

=====

--- SEÇÃO 3: Investigando a Importância do Bias ---

(Usando XOR com 2 entradas, 4 neurônios ocultos, tanh, LR=0.1, 30000 épocas)

Nota: A importância do bias é crucial. Sem ele, a rede pode não conseguir aprender funções que não são separáveis pela origem.

--- XOR com 2 entradas (COM Bias), LR=0.1, Hidden=4, Ativação=tanh, Épocas=30000 ---

=====

==

Época 6000/30000, Erro Médio Quadrático: 0.000030

Época 12000/30000, Erro Médio Quadrático: 0.000013



Época 18000/30000, Erro Médio Quadrático: 0.000009  
Época 24000/30000, Erro Médio Quadrático: 0.000006  
Época 30000/30000, Erro Médio Quadrático: 0.000005

Resultados do Teste Detalhados:

Entrada: [0, 0], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)  
Entrada: [0, 1], Esperado: 1, Previsto (bruto): 0.9948 (Classe: 1)  
Entrada: [1, 0], Esperado: 1, Previsto (bruto): 0.9966 (Classe: 1)  
Entrada: [1, 1], Esperado: 0, Previsto (bruto): -0.0000 (Classe: 0)  
Acurácia Final: 100.00%

=====  
==

--- XOR com 2 entradas (SEM Bias), LR=0.1, Hidden=4, Ativação=tanh, Épocas=30000 ---

=====  
==

Época 6000/30000, Erro Médio Quadrático: 0.000053  
Época 12000/30000, Erro Médio Quadrático: 0.000024  
Época 18000/30000, Erro Médio Quadrático: 0.000016  
Época 24000/30000, Erro Médio Quadrático: 0.000012  
Época 30000/30000, Erro Médio Quadrático: 0.000009

Resultados do Teste Detalhados:

Entrada: [0, 0], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)  
Entrada: [0, 1], Esperado: 1, Previsto (bruto): 0.9939 (Classe: 1)  
Entrada: [1, 0], Esperado: 1, Previsto (bruto): 0.9940 (Classe: 1)  
Entrada: [1, 1], Esperado: 0, Previsto (bruto): 0.0000 (Classe: 0)  
Acurácia Final: 100.00%

=====  
==

--- OR com 2 entradas (COM Bias), LR=0.1, Hidden=2, Ativação=sigmoid, Épocas=10000 ---

=====  
====

Época 2000/10000, Erro Médio Quadrático: 0.005831  
Época 4000/10000, Erro Médio Quadrático: 0.001208  
Época 6000/10000, Erro Médio Quadrático: 0.000613  
Época 8000/10000, Erro Médio Quadrático: 0.000399  
Época 10000/10000, Erro Médio Quadrático: 0.000292

Resultados do Teste Detalhados:

Entrada: [0, 0], Esperado: 0, Previsto (bruto): 0.0352 (Classe: 0)  
Entrada: [0, 1], Esperado: 1, Previsto (bruto): 0.9767 (Classe: 1)  
Entrada: [1, 0], Esperado: 1, Previsto (bruto): 0.9770 (Classe: 1)  
Entrada: [1, 1], Esperado: 1, Previsto (bruto): 0.9945 (Classe: 1)  
Acurácia Final: 100.00%

=====

--- OR com 2 entradas (SEM Bias), LR=0.1, Hidden=2, Ativação=sigmoid, Épocas=10000 ---

=====

=====

Época 2000/10000, Erro Médio Quadrático: 0.069654

Época 4000/10000, Erro Médio Quadrático: 0.007964

Época 6000/10000, Erro Médio Quadrático: 0.002708

Época 8000/10000, Erro Médio Quadrático: 0.001571

Época 10000/10000, Erro Médio Quadrático: 0.001095

Resultados do Teste Detalhados:

Entrada: [0, 0], Esperado: 0, Previsto (bruto): 0.0733 (Classe: 0)

Entrada: [0, 1], Esperado: 1, Previsto (bruto): 0.9620 (Classe: 1)

Entrada: [1, 0], Esperado: 1, Previsto (bruto): 0.9621 (Classe: 1)

Entrada: [1, 1], Esperado: 1, Previsto (bruto): 0.9774 (Classe: 1)

Acurácia Final: 100.00%

=====

=====

--- SEÇÃO 4: Investigando a Importância da Função de Ativação ---

(Usando XOR com 2 entradas, 4 neurônios ocultos, LR=0.1, 30000 épocas)

--- XOR com 2 entradas (COM Bias), LR=0.1, Hidden=4, Ativação=sigmoid, Épocas=30000 ---

=====

=====

Época 6000/30000, Erro Médio Quadrático: 0.012235

Época 12000/30000, Erro Médio Quadrático: 0.001028

Época 18000/30000, Erro Médio Quadrático: 0.000493

Época 24000/30000, Erro Médio Quadrático: 0.000318

Época 30000/30000, Erro Médio Quadrático: 0.000233

Resultados do Teste Detalhados:

Entrada: [0, 0], Esperado: 0, Previsto (bruto): 0.0227 (Classe: 0)

Entrada: [0, 1], Esperado: 1, Previsto (bruto): 0.9798 (Classe: 1)

Entrada: [1, 0], Esperado: 1, Previsto (bruto): 0.9776 (Classe: 1)

Entrada: [1, 1], Esperado: 0, Previsto (bruto): 0.0209 (Classe: 0)

Acurácia Final: 100.00%

=====

=====

--- XOR com 2 entradas (COM Bias), LR=0.1, Hidden=4, Ativação=tanh, Épocas=30000 ---

```
=====
==
Época 6000/30000, Erro Médio Quadrático: 0.000030
Época 12000/30000, Erro Médio Quadrático: 0.000013
Época 18000/30000, Erro Médio Quadrático: 0.000009
Época 24000/30000, Erro Médio Quadrático: 0.000006
Época 30000/30000, Erro Médio Quadrático: 0.000005
Acurácia Final: 100.00%
=====
==
```

--- XOR com 2 entradas (COM Bias), LR=0.05, Hidden=4, Ativação=relu, Épocas=30000 ---

```
=====
===
Época 6000/30000, Erro Médio Quadrático: 0.250000
Época 12000/30000, Erro Médio Quadrático: 0.250000
Época 18000/30000, Erro Médio Quadrático: 0.250000
Época 24000/30000, Erro Médio Quadrático: 0.250000
Época 30000/30000, Erro Médio Quadrático: 0.250000
Acurácia Final: 50.00%
=====
===
```

FIM DOS EXPERIMENTOS