Estudo de Redes Neurais: Perceptron e Backpropagation

Victor Souza Lima

26 de maio de 2025

Sumário

1 Contextualização **2**

2 Motivação do Problema **2**

3 Perceptron - Implementação e Avaliação **2**
3.1 Explicação da Implementação . . . 2
3.1.1 Funcionamento do Algoritmo . . . 2
3.1.2 Estratégias Implementadas . . . 3
3.2 Hiperparâmetros Utilizados . . . 3
3.3 Resultados na Última Época — 2 Entradas . . . 3
3.3.1 Função AND . . . 3
3.3.2 Função OR . . . 3
3.3.3 Função XOR — Limitação Evidente . . . 4
3.4 Resultados na Última Época — 3 Entradas . . . 5
3.4.1 Função AND . . . 5
3.4.2 Função OR . . . 5
3.4.3 Função XOR — Limitação Evidente . . . 6
3.4.4 Observações Adicionais . . . 7

4 Backpropagation - Implementação e Avaliação **7**
4.1 Explicação da Implementação . . . 7
4.1.1 Arquitetura . . . 7
4.1.2 Funcionamento . . . 7
4.1.3 Estratégias Implementadas . . . 8
4.2 Hiperparâmetros Utilizados . . . 8
4.3 Comparações e Benchmarks . . . 8
4.3.1 Curvas de Erro — Comparação Sigmoid vs. Tanh . . . 8
4.3.2 Decision Boundary — 2 Entradas (Backpropagation) . . . 11

5 Conclusão **13**

6 Código **13**

1 Contextualização

As redes neurais artificiais são modelos computacionais inspirados no funcionamento dos neurônios biológicos. Elas são compostas por unidades básicas (neurônios artificiais) que, interconectadas, possuem a capacidade de aprender padrões e realizar tarefas como classificação, regressão, predição e reconhecimento de padrões.

Este estudo tem como objetivo compreender profundamente os fundamentos de dois dos algoritmos mais clássicos na área de redes neurais:

- **Perceptron**: voltado para problemas linearmente separáveis.
- **Backpropagation**: algoritmo que permite o treinamento de redes neurais multicamadas, solucionando também problemas não linearmente separáveis.

Os testes foram conduzidos sobre funções booleanas clássicas (AND, OR e XOR) utilizando n entradas, permitindo avaliar a escalabilidade e a robustez dos algoritmos.

2 Motivação do Problema

O desenvolvimento deste projeto visa consolidar o entendimento dos princípios de funcionamento das redes neurais, bem como suas limitações e potencialidades. Ao implementar algoritmos do zero, é possível internalizar conceitos como:

- Ajuste de pesos;
- Propagação direta e retropropagação do erro;
- Influência de hiperparâmetros como taxa de aprendizado, bias e função de ativação;
- Compreender os limites do Perceptron em relação a problemas não linearmente separáveis, como o XOR.

3 Perceptron - Implementação e Avaliação

3.1 Explicação da Implementação

O Perceptron é o modelo mais simples de rede neural, composto por uma única camada. Sua lógica baseia-se na soma ponderada das entradas com pesos ajustáveis, acrescida de um bias, passando por uma função de ativação do tipo degrau.

3.1.1 Funcionamento do Algoritmo

1. Inicializa os pesos e o bias com valores aleatórios pequenos.
2. Para cada instância de treinamento:
   (a) Calcula a saída da função linear:
   $$z = \sum_{i=1}^{n} w_i x_i + b$$
   (b) Aplica a função de ativação:
   $$\hat{y} = \begin{cases} 1 & \text{se } z \geq 0 \\ 0 & \text{se } z < 0 \end{cases}$$
   (c) Atualiza os pesos e bias conforme a regra:
   $$w_j = w_j + \eta(y - \hat{y})x_j$$
   $$b = b + \eta(y - \hat{y})$$
3. Repete-se por um número pré-definido de épocas ou até a convergência.

3.1.2 Estratégias Implementadas

- Treinamento online (atualização dos pesos a cada amostra);
- Bias incluso para permitir deslocamento do hiperplano;
- Monitoramento dos erros por época para verificar a convergência;
- Plotagem da fronteira de decisão (decision boundary) durante o treinamento.

3.2 Hiperparâmetros Utilizados

- Taxa de aprendizado: 0.1
- Número máximo de épocas: 100
- Bias: Ativado
- Função de ativação: Degrau

3.3 Resultados na Última Época — 2 Entradas

3.3.1 Função AND

O Perceptron foi capaz de aprender perfeitamente a função AND com duas entradas. A reta de decisão após o treinamento está posicionada de forma adequada, separando os pontos da classe 0 e classe 1 de maneira correta.

Observa-se no gráfico que, mesmo com uma quantidade limitada de épocas (100), o modelo converge rapidamente para uma solução ótima, ajustando seus pesos de maneira eficaz para definir uma fronteira de decisão linear que divide corretamente os dados.

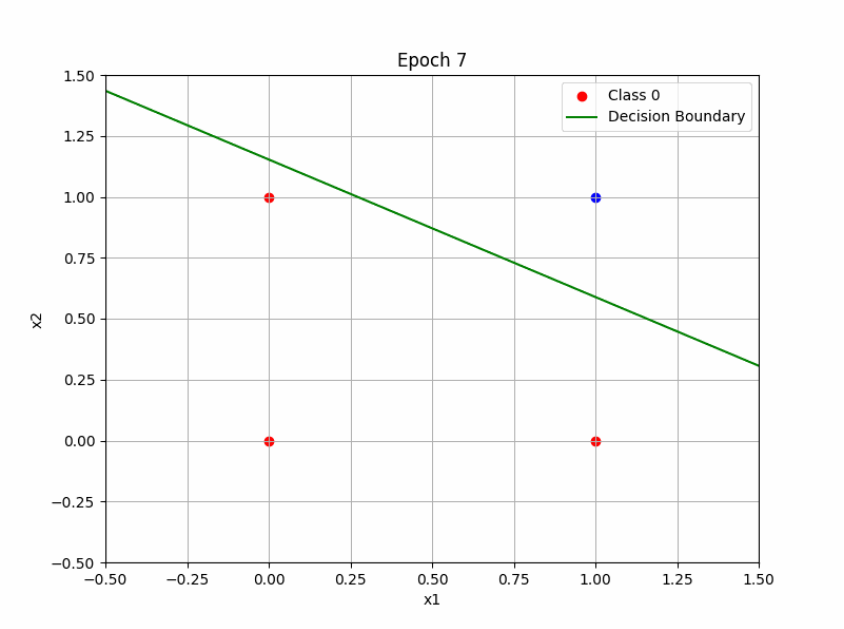

Figura 1: Fronteira de decisão após a última época para a função AND (2 entradas).

Clique aqui para visualizar o GIF completo da evolução do treinamento.

3.3.2 Função OR

Similar ao AND, o Perceptron também obteve sucesso na classificação da função OR. A fronteira de decisão final é capaz de separar corretamente os dados, demonstrando a eficácia do modelo em problemas linearmente separáveis.

O modelo converge rapidamente, posicionando a linha de decisão de maneira eficiente. O gráfico evidencia que o Perceptron não encontra dificuldades em separar os dados da função OR, visto que qualquer entrada com pelo menos um valor 1 pertence à classe positiva.

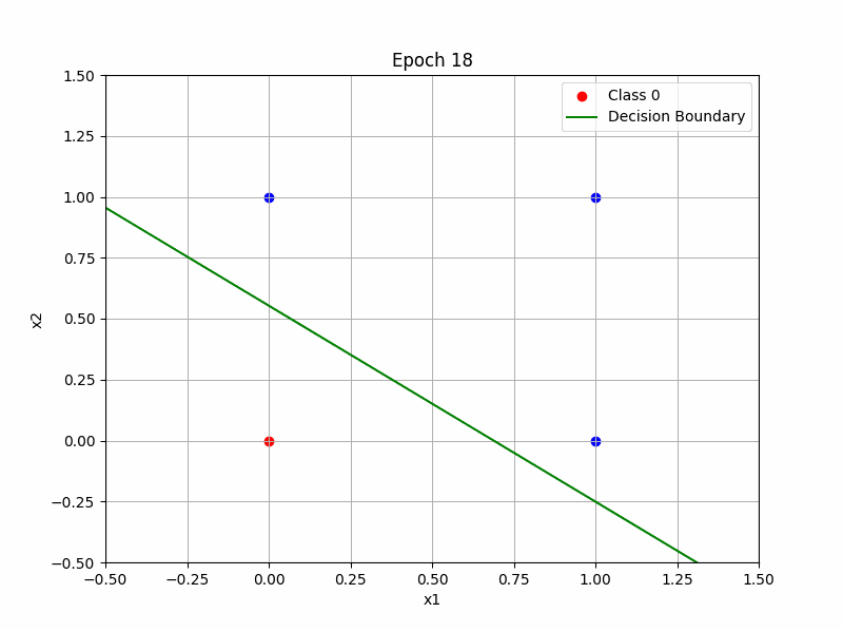

Figura 2: Fronteira de decisão após a última época para a função OR (2 entradas).

Clique aqui para visualizar o GIF completo da evolução do treinamento.

3.3.3 Função XOR — Limitação Evidente

A função XOR evidencia claramente a limitação do Perceptron. Observamos que, mesmo após 100 épocas, a linha de decisão permanece incapaz de separar corretamente os pontos. Isso ocorre porque a função XOR não é linearmente separável, e, portanto, **o Perceptron nunca irá convergir** para uma solução correta, independentemente do número de épocas.

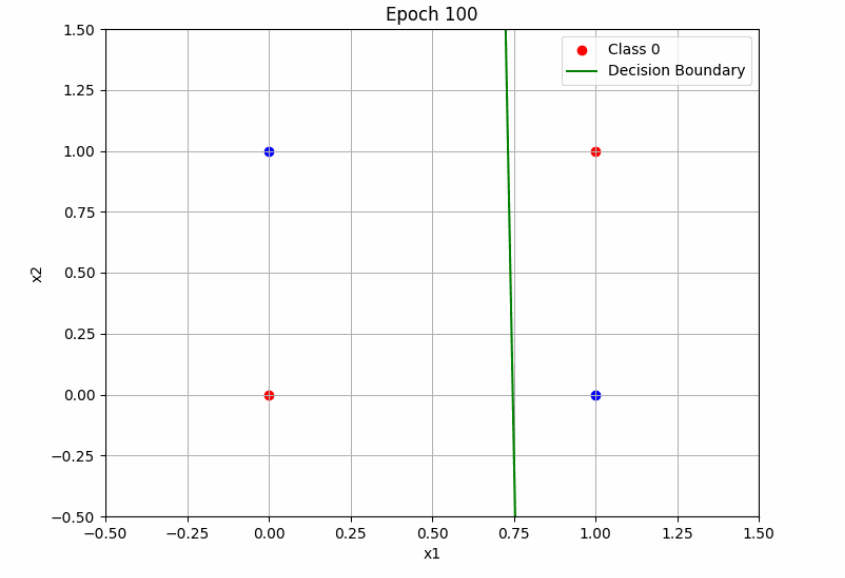

Figura 3: Tentativa de fronteira de decisão na última época para a função XOR (2 entradas).

Clique aqui para visualizar o GIF completo da evolução do treinamento.

No gráfico, nota-se que a linha de decisão fica oscilando, buscando uma solução, porém sem sucesso. Esta situação demonstra um conceito fundamental em redes neurais: **o Perceptron é incapaz de resolver problemas não linearmente separáveis**, o que reforça a necessidade do uso de modelos mais complexos, como redes multicamadas treinadas com backpropagation.

É importante ressaltar que a redução do número de épocas para 100 não altera o resultado, visto que o modelo nunca encontraria uma solução, independente do tempo de treinamento.

3.4 Resultados na Última Época — 3 Entradas

3.4.1 Função AND

O Perceptron foi capaz de resolver corretamente a função AND com três entradas. Observa-se que o modelo foi capaz de gerar um **hiperplano tridimensional (um plano em 3D)** capaz de separar adequadamente os pontos das classes.

O plano de decisão se ajustou progressivamente durante as épocas, até alcançar uma configuração que separa de forma correta os pontos correspondentes à saída 1 daqueles correspondentes à saída 0.

O resultado visual confirma que a função AND, sendo linearmente separável mesmo em três dimensões, é resolvida com sucesso pelo Perceptron.

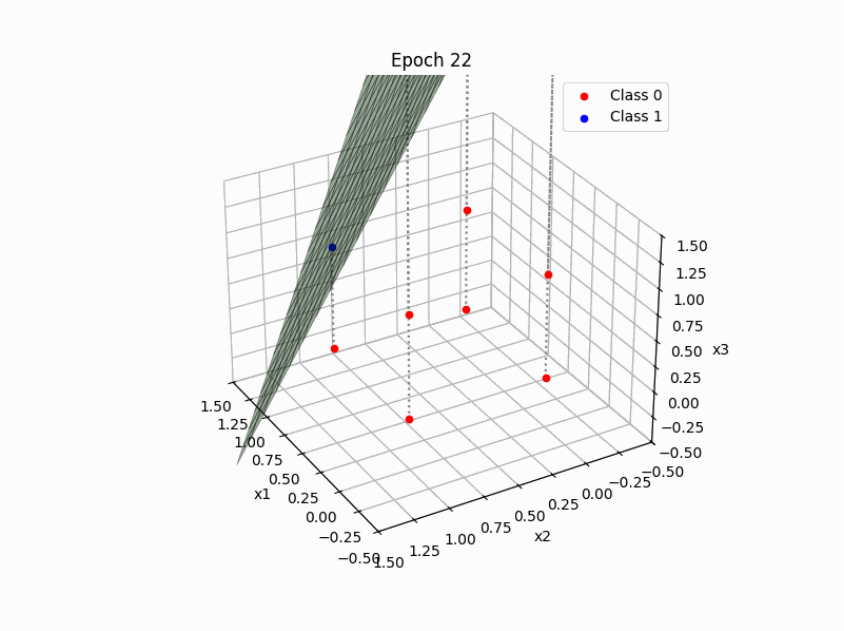

Figura 4: Fronteira de decisão após a última época para a função AND (3 entradas).

Clique aqui para visualizar o GIF completo da evolução do treinamento.

3.4.2 Função OR

Para a função OR com três entradas, o Perceptron também demonstrou excelente desempenho. O plano de decisão foi capaz de se posicionar corretamente no espaço tridimensional, realizando a separação entre os pontos da classe 1 (onde pelo menos uma entrada é 1) e da classe 0 (quando todas as entradas são 0).

O processo de convergência foi eficiente, com o hiperplano sendo ajustado de maneira suave e rápida durante as épocas.

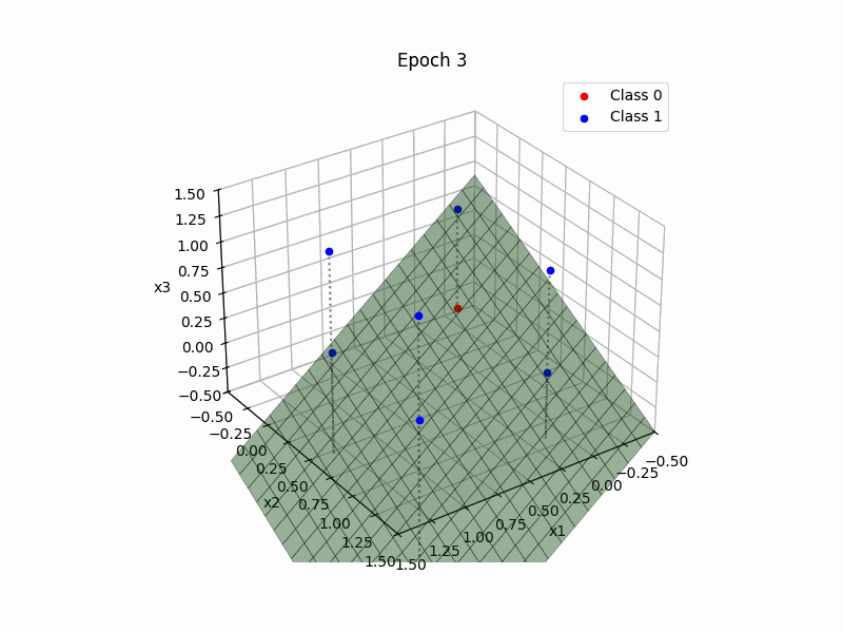

Figura 5: Fronteira de decisão após a última época para a função OR (3 entradas).

Clique aqui para visualizar o GIF completo da evolução do treinamento.

3.4.3 Função XOR — Limitação Evidente

O caso da função XOR com três entradas reforça a limitação estrutural do Perceptron. Mesmo com o acréscimo de uma dimensão, o problema permanece **não linearmente separável**, portanto, um único plano não é capaz de realizar a separação correta dos dados.

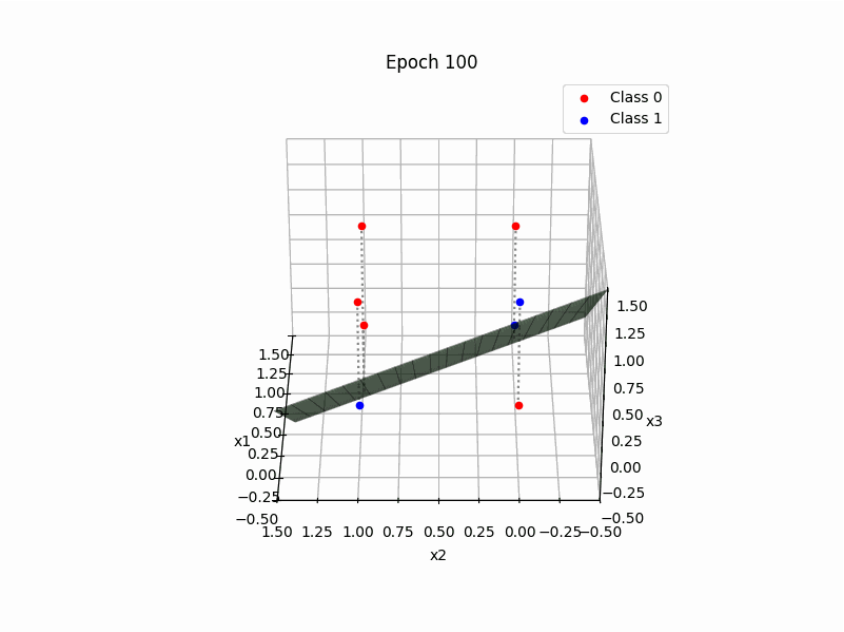

Figura 6: Tentativa de fronteira de decisão na última época para a função XOR (3 entradas).

Clique aqui para visualizar o GIF completo da evolução do treinamento.

Ao observar o plano de decisão gerado, percebe-se que ele falha completamente em encontrar uma configuração que consiga dividir as classes de maneira satisfatória. A fronteira oscila durante o treinamento, sem nunca conseguir posicionar-se de forma adequada.

Importante destacar que, mesmo que aumentássemos o número de épocas para valores muito superiores, o modelo **não conseguiria convergir**, pois a incapacidade está na natureza do problema, e não no tempo de treinamento.

3.4.4 Observações Adicionais

Assim como nos experimentos com duas entradas, o Perceptron demonstra competência para resolver problemas linearmente separáveis, independentemente do aumento de dimensionalidade, desde que a separabilidade linear seja mantida.

Contudo, quando a separabilidade linear não é possível, como é o caso do XOR, o Perceptron se mostra matematicamente incapaz de encontrar uma solução adequada.

Fica evidente também que, conforme o número de dimensões cresce, a visualização direta dos hiperplanos se torna mais complexa. Nos experimentos realizados, foi possível visualizar o plano de decisão em 3D. Entretanto, para problemas com 4 ou mais entradas, será necessária a aplicação de técnicas de **redução de dimensionalidade** — como PCA, t-SNE ou UMAP — para possibilitar uma análise visual eficiente e intuitiva.

4 Backpropagation - Implementação e Avaliação

4.1 Explicação da Implementação

O algoritmo Backpropagation permite o treinamento de redes neurais de múltiplas camadas (MLP), sendo fundamental para resolver problemas não linearmente separáveis.

4.1.1 Arquitetura

- Uma camada de entrada;
- Uma ou mais camadas ocultas;
- Uma camada de saída.

4.1.2 Funcionamento

1. **Forward Propagation**:
   - Calcula-se a saída de cada neurônio aplicando:

     $$a = f\left(\sum w_i x_i + b\right)$$

     onde f é a função de ativação.
2. **Cálculo do Erro**:
   - **Na camada de saída**:

     $$\delta = (y - \hat{y}) \cdot f'(z)$$

   - **Nas camadas ocultas**:

     $$\delta_{hidden} = f'(z) \cdot \sum (w_{kj} \cdot \delta_{next})$$
3. **Atualização dos Pesos**:
   - Os pesos são atualizados pela regra:

     $$w = w + \eta \cdot \delta \cdot a_{anterior}$$

4.1.3 Estratégias Implementadas

- Implementação de forward e backward de forma vetorizada;
- Inclusão de bias em todas as camadas;
- Suporte a diferentes funções de ativação:
  - Sigmoide;
  - Tangente Hiperbólica;
- Plotagem das curvas de erro por época;
- Geração dos decision boundaries para avaliar separabilidade.

4.2 Hiperparâmetros Utilizados

- Taxas de aprendizado: 0.01;
- Número máximo de épocas: 100.000;
- Critério de parada: erro médio menor que 0.0001 ou até atingir 100.000 épocas;
- Bias: Ativado;
- Funções de ativação:
  - Sigmoide;
  - Tangente Hiperbólica (Tanh).
- Arquitetura da rede neural:
  - 1 camada de entrada com 2 neurônios;
  - 1 camada oculta com 1 neurônio;
  - 1 camada de saída com 1 neurônio.

4.3 Comparações e Benchmarks

4.3.1 Curvas de Erro — Comparação Sigmoid vs. Tanh

Nesta seção, são analisadas as curvas de erro para os modelos treinados com funções de ativação Sigmoid e Tanh, aplicados às funções AND, OR e XOR com 2 e 3 entradas.

AND — 2 bits A função **Tanh** converge de maneira significativamente mais rápida (em menos de 2000 épocas), enquanto a **Sigmoid** necessita de aproximadamente 70.000 épocas para estabilizar próximo do erro mínimo. Este comportamento é atribuído ao problema do desaparecimento do gradiente característico da Sigmoid.

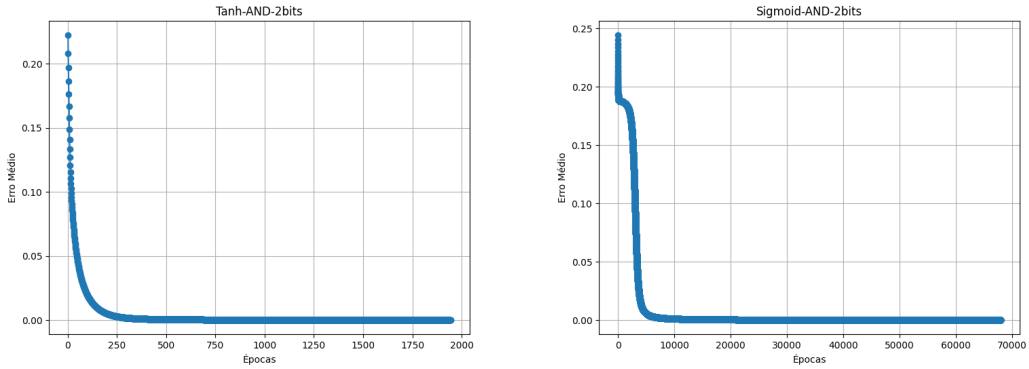

Figura 7: Curva de erro para a função AND com 2 bits — Tanh (esquerda) vs. Sigmoid (direita).

AND — 3 bits O padrão se repete: **Tanh** converge em cerca de 2500 épocas, enquanto a **Sigmoid** leva mais de 25.000 épocas para atingir o mesmo patamar de erro.

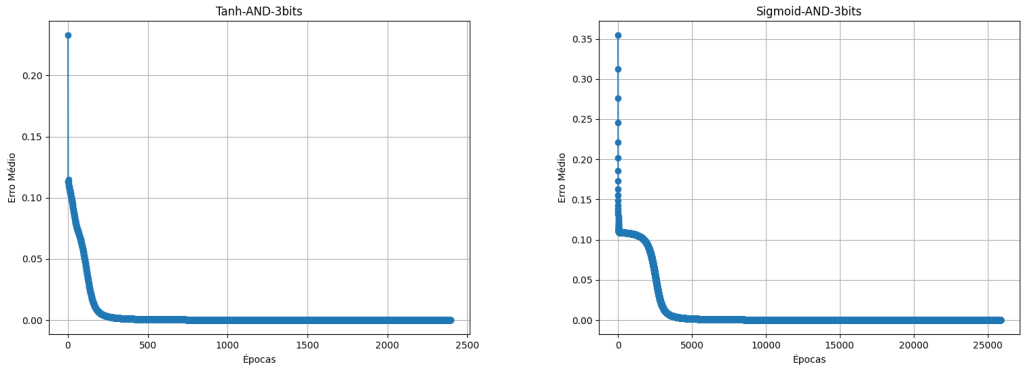

Figura 8: Curva de erro para a função AND com 3 bits — Tanh (esquerda) vs. Sigmoid (direita).

OR — 2 bits A função OR confirma o comportamento já observado: a **Tanh** converge de forma eficiente (em menos de 1200 épocas), enquanto a **Sigmoid** demora mais de 60.000 épocas para atingir o erro mínimo.

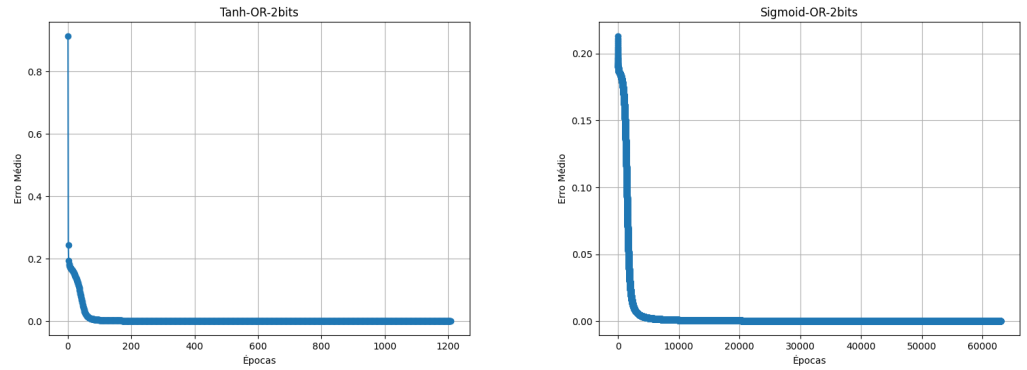

Figura 9: Curva de erro para a função OR com 2 bits — Tanh (esquerda) vs. Sigmoid (direita).

OR — 3 bits Neste caso, a **Tanh** apresenta uma curva suave e rápida, convergindo em cerca de 900 épocas, enquanto a **Sigmoid** precisa de aproximadamente 19.000 épocas.

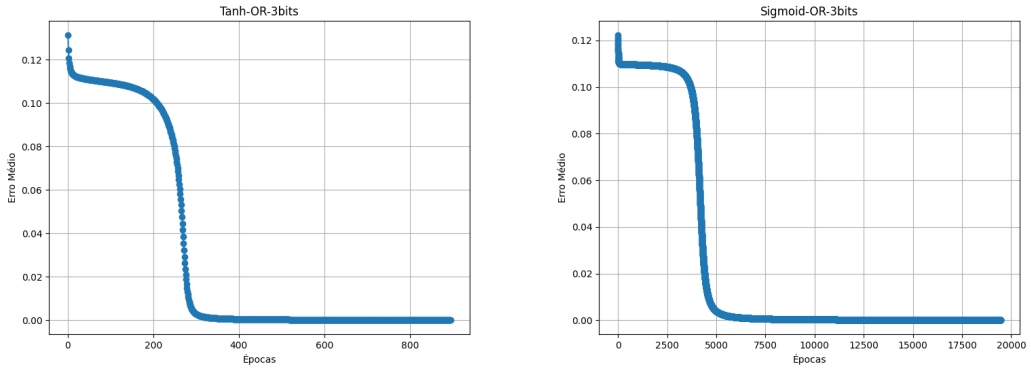

Figura 10: Curva de erro para a função OR com 3 bits — Tanh (esquerda) vs. Sigmoid (direita).

XOR — 2 bits Observa-se que, apesar de ambas as funções conseguirem reduzir o erro, a função **Sigmoid** não conseguiu atingir o critério de erro mínimo de 0.0001 dentro do limite de 100.000 épocas. A curva apresenta uma longa região de estagnação (platô) antes de começar a reduzir gradualmente o erro, o que evidencia o problema do gradiente desaparecer. A função **Tanh** mostra desempenho superior, convergindo rapidamente em menos de 3000 épocas.

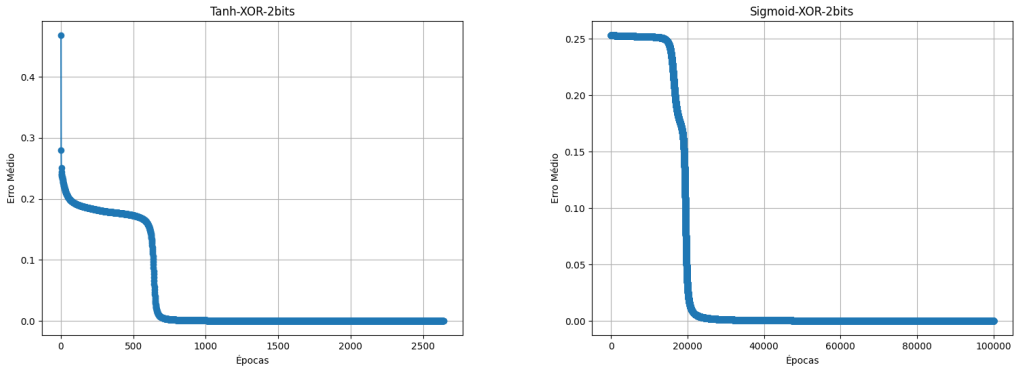

Figura 11: Curva de erro para a função XOR com 2 bits — Tanh (esquerda) vs. Sigmoid (direita).

XOR — 3 bits O comportamento se intensifica no cenário de 3 bits. A função **Sigmoid** falhou novamente em atingir o critério de erro mínimo de 0.0001, mesmo após 100.000 épocas, que era o limite definido. Por outro lado, a função **Tanh** apresenta uma curva de erro suave e eficiente, convergindo rapidamente para valores aceitáveis de erro em menos de 3000 épocas.

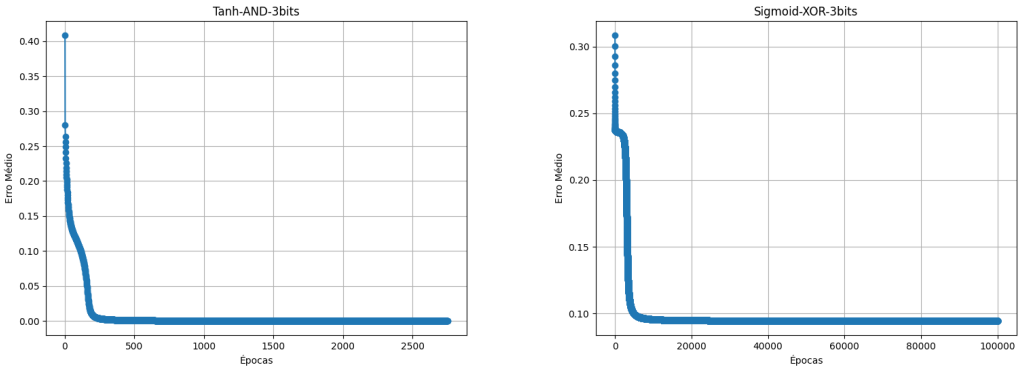

Figura 12: Curva de erro para a função XOR com 3 bits — Tanh (esquerda) vs. Sigmoid (direita).

Conclusão das Curvas de Erro

- A função **Tanh** apresenta desempenho superior em todos os cenários, com uma convergência mais rápida, suave e estável.
- A função **Sigmoid** sofre severamente com o problema do desaparecimento do gradiente, especialmente à medida que o número de entradas cresce.
- Nos testes com a função XOR (tanto 2 quanto 3 bits), a **Sigmoid não conseguiu atingir o erro mínimo de 0.0001 dentro do limite de 100.000 épocas**, demonstrando sua limitação em cenários que exigem maior complexidade na separação dos dados.
- Ambas as funções são matematicamente capazes de resolver problemas linear e não linearmente separáveis quando se utilizam redes com camadas ocultas, porém a **Tanh** é claramente mais eficiente e robusta.

4.3.2 Decision Boundary — 2 Entradas (Backpropagation)

Nesta seção, analisamos as fronteiras de decisão (decision boundaries) geradas pela rede neural para as funções AND, OR e XOR com duas entradas. As fronteiras demonstram visualmente como a rede realiza a separação dos dados no espaço bidimensional.

AND — 2 bits A função AND é linearmente separável. A rede neural foi capaz de gerar uma fronteira de decisão aproximadamente linear, como esperado. Observa-se que o plano de separação divide corretamente a região onde ambas as entradas são iguais a 1, atribuindo-a à classe positiva, enquanto os demais pontos pertencem à classe negativa.

A fronteira é bem definida, mostrando que o modelo aprendeu eficientemente a relação lógica da função AND.

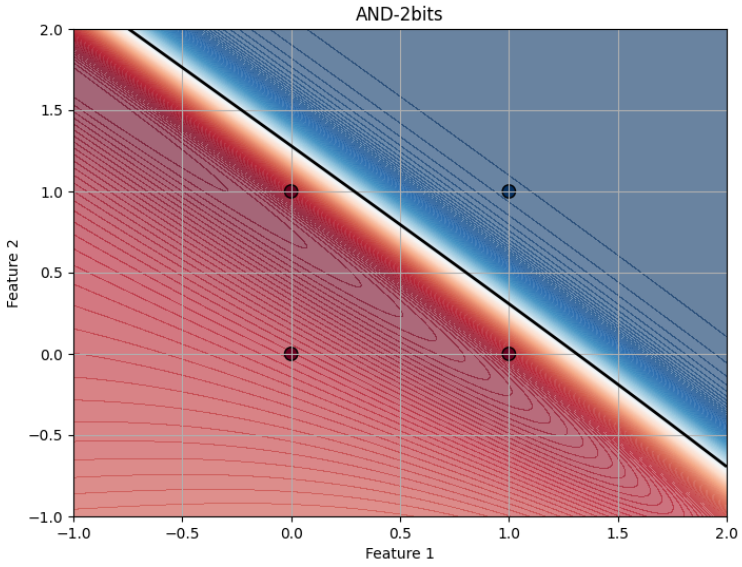

Figura 13: Fronteira de decisão para a função AND com 2 entradas.

OR — 2 bits Assim como no AND, a função OR também é linearmente separável. A rede neural gerou uma fronteira linear, que separa corretamente os pontos onde pelo menos uma das entradas é 1 (classe positiva) do ponto (0, 0) que pertence à classe negativa.

A fronteira é bem ajustada e confirma a eficácia do modelo em resolver problemas simples e linearmente separáveis.

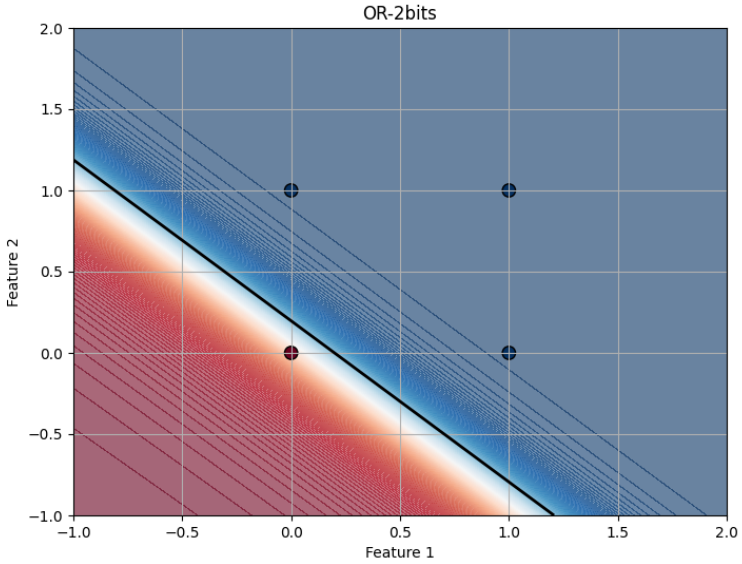

Figura 14: Fronteira de decisão para a função OR com 2 entradas.

XOR — 2 bits A análise da função XOR é particularmente interessante, pois este problema não é linearmente separável. Observa-se que a rede neural conseguiu gerar uma fronteira de decisão claramente não linear, formando duas regiões bem definidas que separam corretamente os pontos das classes.

Essa fronteira curva evidencia a capacidade do algoritmo Backpropagation, em conjunto com uma camada oculta, de resolver problemas não linearmente separáveis, algo impossível para um Perceptron simples.

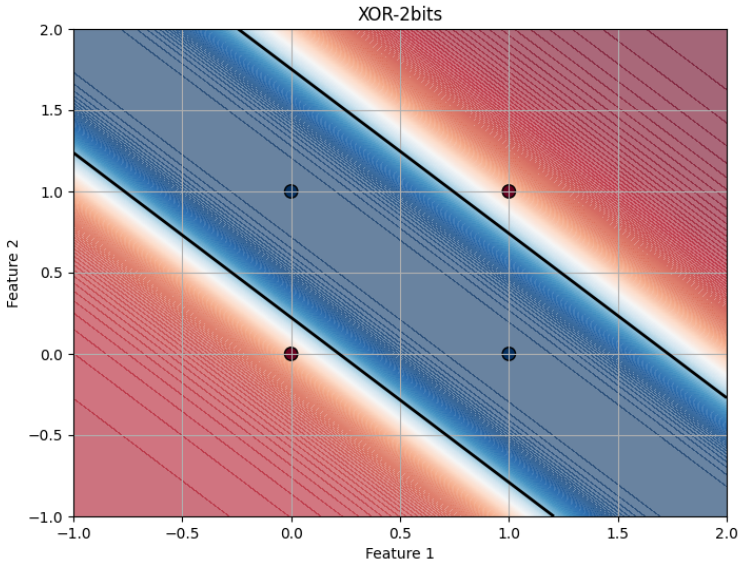

Figura 15: Fronteira de decisão para a função XOR com 2 entradas.

Conclusão sobre Decision Boundary

- Para funções linearmente separáveis (AND e OR), a rede gera fronteiras aproximadamente lineares, confirmando sua capacidade de generalização.

- Para funções não linearmente separáveis como o XOR, a rede gera uma **fronteira de decisão não linear**, evidenciando o papel crítico da camada oculta no aprendizado de padrões complexos.
- As fronteiras geradas ilustram visualmente a diferença de capacidade entre um Perceptron simples e uma rede treinada com Backpropagation.

5 Conclusão

O presente estudo teve como objetivo explorar, entender e comparar dois dos algoritmos mais fundamentais no campo das redes neurais: o Perceptron e o Backpropagation. Por meio da implementação prática desses modelos aplicados às funções booleanas AND, OR e XOR, foi possível observar de forma clara suas capacidades, limitações e comportamentos em diferentes cenários.

O **Perceptron** demonstrou ser um modelo eficaz para problemas **linearmente separáveis**, como as funções AND e OR, apresentando convergência rápida e estabilidade durante o treinamento. Contudo, sua limitação estrutural torna-o **incapaz de resolver problemas não linearmente separáveis**, como o XOR, independentemente da quantidade de épocas ou ajustes nos hiperparâmetros. Este resultado reforça um dos conceitos teóricos mais importantes em redes neurais: a necessidade de camadas ocultas para a modelagem de funções complexas.

Por outro lado, o algoritmo **Backpropagation**, aplicado em redes multicamadas, foi capaz de superar esta limitação, resolvendo não apenas problemas linearmente separáveis, mas também aqueles **não linearmente separáveis**, como a função XOR. A presença da camada oculta permite que o modelo aprenda representações mais complexas dos dados, ajustando fronteiras de decisão não lineares de maneira eficiente.

Durante os experimentos, ficou evidente que a **taxa de aprendizado** exerce um papel crítico no desempenho do treinamento:

- Taxas muito elevadas podem provocar **instabilidades**, com oscilações e até divergência no erro.
- Taxas muito baixas resultam em **convergência extremamente lenta**, aumentando o custo computacional.

O uso do **bias** também se mostrou essencial, proporcionando maior flexibilidade no ajuste das fronteiras de decisão, especialmente em cenários onde a separação dos dados não passa necessariamente pela origem.

Adicionalmente, foi possível comparar o desempenho de diferentes funções de ativação. As funções não lineares, especialmente a **Tangente Hiperbólica (Tanh)**, apresentaram desempenho superior em relação à **Sigmoide**, principalmente em termos de velocidade de convergência e estabilidade. A Sigmoide, por sua vez, sofreu notoriamente com o problema do **desvanecimento do gradiente**, especialmente em problemas mais complexos como o XOR.

Por fim, destaca-se que, à medida que o número de entradas cresce, a visualização dos hiperplanos de decisão se torna inviável diretamente. Assim, para análises em espaços com quatro ou mais dimensões, torna-se indispensável a aplicação de técnicas de **redução de dimensionalidade**, como **PCA**, **t-SNE** e **UMAP**, permitindo a compreensão visual dos padrões aprendidos.

Este estudo reforça conceitos teóricos fundamentais, valida suas implicações práticas e abre espaço para investigações futuras, como a exploração de arquiteturas mais profundas, diferentes funções de ativação, regularização, otimização de hiperparâmetros e análise de desempenho em conjuntos de dados mais complexos e reais.

6 Código

O código desenvolvido pode ser acessado no repositório: GitHub - Neural Networks From Scratch