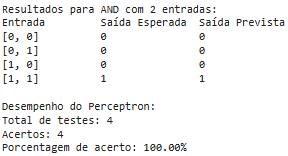
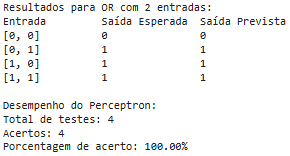
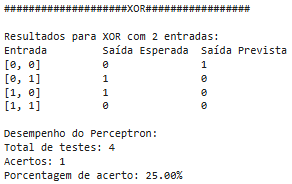
**Lista 7**

Nome: Diego Polanski de Freitas Vieira

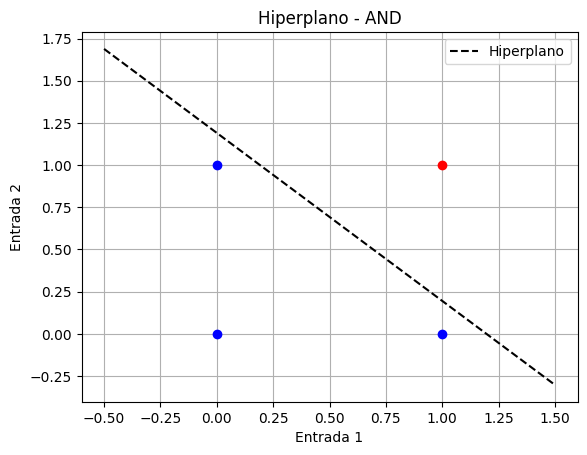
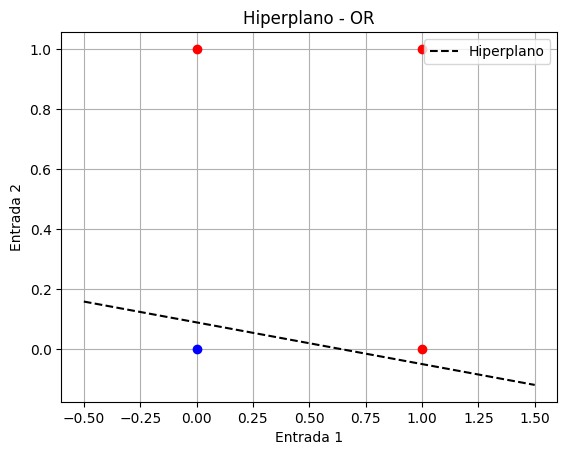
Link para os códigos:

1. Link do código: <https://colab.research.google.com/drive/1g-ps5pX47xCTXdrWZlqdvsGQ336gn9PK#scrollTo=nm2J3AO9UT4X>

Desempenho do AND (2 entradas) Desempenho do OR (2 entradas)

 Desempenho do OR (2 entradas) “falhou”

Hiperplanos:

* **Definição da Classe Perceptron:**
  + A classe Perceptron foi criada para simular o funcionamento de um neurônio simples, com métodos para ativação, treinamento e resposta. Os pesos são inicializados aleatoriamente, e a função de ativação utilizada é a função degrau (limiar).
* **Funções Auxiliares:**
  + Foram implementadas funções para gerar tabelas-verdade de operadores lógicos AND, OR e XOR. Cada tabela associa as combinações possíveis de entradas com as saídas esperadas.
* **Treinamento:**
  + O treinamento do perceptron é realizado ajustando os pesos de acordo com os erros entre a saída prevista e a saída esperada. Esse processo ocorre durante o número definido de épocas.
* **Teste e Desempenho:**
  + Após o treinamento, o perceptron é testado com as combinações da tabela-verdade para avaliar a precisão. O desempenho é medido pelo número de acertos e a porcentagem de acerto.

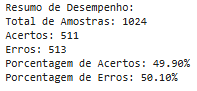
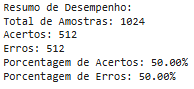
1. Link do Código: <https://colab.research.google.com/drive/1vCwlB4hUp4FuAvFyFlZznDCGk_RvNuMC#scrollTo=qjRF_yXpjqBF>

O Backpropagation é um algoritmo usado para treinar redes neurais ajustando os pesos com base no erro das previsões. Ele funciona em duas etapas: primeiro, os dados passam pela rede para gerar uma saída; depois, o erro é calculado e propagado de volta, ajustando os pesos para melhorar o desempenho. Esse processo é repetido várias vezes, permitindo que a rede aprenda a fazer previsões mais precisas. É uma técnica essencial no aprendizado de máquinas, especialmente para problemas complexos como reconhecimento de imagens e processamento de linguagem natural.

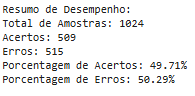
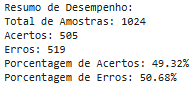
* **A importância da taxa de aprendizado:**

Os testes foram feitos com a porta XOR; 4 entradas; função de ativação “Sigmoid”; 100 número de épocas.

Taxa de aprendizado: 0.1 Taxa de aprendizado: 0.3

 ****

Taxa de aprendizado: 0.5 Taxa de aprendizado: 1.0

 ****

**OBS:** % de acerto baixo devido ao baixo número de épocas.

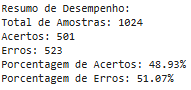
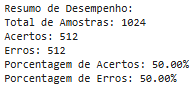
* **A importância do bias:**

O bias em redes neurais é como um ajuste extra que ajuda a definir quando um neurônio deve ativar. Ele permite que a rede aprenda não só a inclinação dos pesos, mas também onde a função começa tornando possível representar coisas que não começam na origem. Sem o bias, a rede seria bem mais limitada no que consegue aprender.

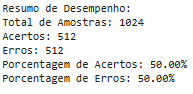
* **Funções de ativação:**

Os testes foram feitos com a porta XOR; 4 entradas; 1 de taxa de aprendizado; 100 número de épocas.

Sigmoid ReLU

** **

Tangente Hiperbólica



**Sigmoid:** Boa para saídas probabilísticas, mas pode sofrer com gradientes pequenos.

**ReLu:** Simples e eficiente, mas pode gerar neurônios mortos.

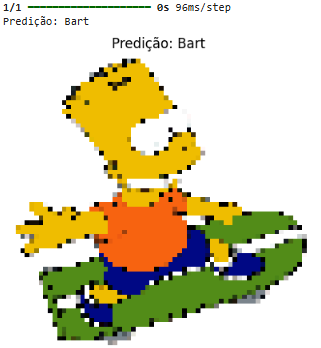
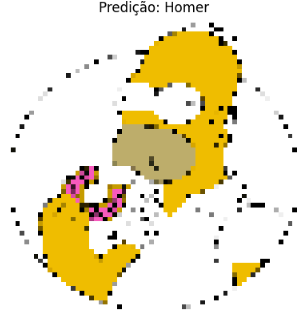
**Tangente Hiperbólica:** Similar à sigmoide, mas centrada em zero, o que ajuda no aprendizado em algumas redes.

* **Razão do backpropagation utilizar uma função de ativação não linear:**

O backpropagation usa funções de ativação não lineares porque elas permitem à rede aprender padrões mais complexos. Se fossem lineares, mesmo com várias camadas, a rede só conseguiria resolver problemas simples, como uma regressão linear. Funções como ReLU ou sigmoide dão à rede a capacidade de representar relações não lineares, essenciais para lidar com problemas mais difíceis, como o XOR, que um perceptron comum não consegue resolver.

1. Link do código: <https://colab.research.google.com/drive/1dnOpmVnpFa-D1vLa-wcag4T8Wox1VLZT#scrollTo=EVUKZkuyN8I_>

**Resultados Obtidos:**

** **

**Respostas**:

**Alterações na Estrutura da CNN e Impacto no Desempenho:**

Camadas de Convolução: Adicionei uma terceira camada de convolução com 64 filtros (3×33\times33×3) para capturar padrões mais complexos.

Kernel e Filtros: Alterei o kernel da primeira camada para 5×55\times55×5 e aumentamos para 64 filtros nas camadas seguintes, com maior detalhamento na extração de características.

Pooling: Substitui a última camada de pooling por Global Average Pooling, compactando os dados e reduzindo o risco de overfitting.

**Explicação do algoritmo:**

Este código implementa uma Rede Neural Convolucional (CNN) para classificar imagens entre duas classes: "Bart" e "Homer", personagens do desenho Os Simpsons. Aqui está uma visão geral dos principais passos:

**Preparação de Dados:**

O código descompacta um arquivo ZIP contendo as imagens para um diretório específico.

Usa a biblioteca ImageDataGenerator para realizar aumentação de dados, como escalonamento, cortes aleatórios e inversão horizontal.

**Construção da CNN:**

Utiliza o Keras com TensorFlow para construir a arquitetura da CNN.

**Contém:**

Três camadas de convolução com 64 filtros e ReLU como função de ativação.

Camadas de Pooling (MaxPooling) para reduzir dimensionalidade.

Uma camada de Flatten para transformar as imagens em um vetor unidimensional.

Camadas densas (Dense), incluindo a final com uma função de ativação sigmoid para prever probabilidades binárias.

**Compilação e Treinamento:**

Compila a rede usando o otimizador Adam, perda binária (binary\_crossentropy), e mede a precisão.

Treina o modelo com um conjunto de treinamento e validação, especificando 5 épocas.

**Testes e Previsões:**

Carrega imagens individuais para teste.

Usa a rede treinada para prever se a imagem pertence à classe "Bart" ou "Homer".

Exibe o resultado da previsão junto com a imagem usando Matplotlib.