# Relatório Simplificado da Lista de Exercícios #10
# Redes Neurais: Perceptron e Backpropagation

Samuel Horta de Faria
801528
https://github.com/SamuelHortadeFaria/IA_Lista10

25 de maio de 2025

# Sumário

# 1 Introdução

Este relatório detalha a implementação e análise dos algoritmos Perceptron e Backpropagation, conforme solicitado na Lista de Exercícios #10 da disciplina de Inteligência Artificial. O objetivo é explorar as capacidades e limitações desses modelos de redes neurais na resolução de problemas de classificação baseados em funções lógicas (AND, OR, XOR) com um número variável de entradas booleanas.

O Exercício 1 foca no Perceptron de camada única. O Exercício 2 aborda o algoritmo Backpropagation em redes Perceptron de Múltiplas Camadas (MLP).

# 2 Exercício 1: Implementação do Algoritmo Perceptron

## 2.1 Introdução ao Perceptron

O Perceptron é um modelo simples de rede neural com um único neurônio. Ele aprende a classificar padrões que são linearmente separáveis. O aprendizado é supervisionado, atualizando os pesos das conexões com base no erro entre a saída produzida e a desejada.

A regra de atualização de pesos do Perceptron envolve ajustar cada peso somando-se a ele um valor. Esse valor é o produto da taxa de aprendizado, do erro (saída desejada menos a saída atual do neurônio) e do valor da entrada correspondente àquele peso. O bias (ou limiar) é tratado como um peso conectado a uma entrada fixa de valor 1.

## 2.2 Explicação da Implementação

### 2.2.1 Geração de Dados (N Entradas)

Para um número $n$ de entradas, foram geradas todas as $2^n$ combinações de entradas booleanas (0 ou 1).

- **Função AND**: Saída 1 se todas as entradas são 1.
- **Função OR**: Saída 1 se alguma entrada é 1.
- **Função XOR**: Para $n = 2$, saída 1 se as entradas são diferentes.

### 2.2.2 Classe Perceptron

Uma classe Python foi criada para o Perceptron. Ela inicializa os pesos (aleatoriamente ou com zeros), incluindo o bias. Possui um método para predição, que calcula a soma ponderada das entradas e aplica uma função de ativação degrau. O método de treinamento itera sobre os dados, ajustando os pesos quando ocorrem erros.

### 2.2.3 Função de Ativação Degrau

A função de ativação é a degrau: se a soma ponderada das entradas (incluindo o bias) for maior ou igual a um limiar (geralmente zero), a saída do neurônio é 1; caso contrário, é 0.

### 2.2.4 Plotagem do Hiperplano de Separação

Para $n = 2$ entradas, o hiperplano de separação é uma reta. No notebook, essa reta é plotada para mostrar como o Perceptron divide o espaço de entradas. Para $n > 2$, essa visualização direta não é simples. As discussões sobre plotagem referem-se aos casos $n = 2$ visualizados no ambiente de desenvolvimento (notebook).

## 2.3 Resultados dos Testes

### 2.3.1 Função AND com $n$ Entradas

O Perceptron foi testado para a função AND com $n = 2$, $n = 3$ e $n = 5$ entradas.

- **AND com 2 entradas**: O Perceptron convergiu rapidamente. Uma visualização gráfica no notebook mostraria uma reta separando corretamente os pontos (0,0), (0,1), (1,0) da classe 0 do ponto (1,1) da classe 1.
- **AND com $n > 2$ entradas**: O Perceptron também aprendeu a função AND, atingindo 100% de precisão, pois ela continua linearmente separável.

As curvas de aprendizado (erro por época), geradas no notebook, mostrariam o erro diminuindo até zero.

### 2.3.2 Função OR com $n$ Entradas

O Perceptron foi testado para a função OR com $n = 2$, $n = 3$ e $n = 5$ entradas.

- **OR com 2 entradas**: O Perceptron convergiu. A visualização gráfica no notebook exibiria uma reta separando o ponto (0,0) da classe 0 dos pontos (0,1), (1,0), (1,1) da classe 1.
- **OR com $n > 2$ entradas**: A função OR também foi aprendida com 100% de precisão.

Similarmente ao AND, as curvas de erro no notebook indicariam a convergência.

### 2.3.3 Por que o Perceptron NÃO resolve o XOR?

A função XOR para duas entradas não é linearmente separável. Seus pontos são (0,0)→0, (0,1)→1, (1,0)→1, (1,1)→0. Não existe uma única reta que possa separar os pontos da classe 0 dos pontos da classe 1 no plano bidimensional. O Perceptron, por gerar apenas um hiperplano, falha. O treinamento não converge para erro zero; a acurácia não atinge 100%. Uma representação gráfica dos pontos do XOR no notebook ilustraria essa não separabilidade.

## 2.4 Considerações Finais do Exercício 1

O Perceptron é eficaz para problemas linearmente separáveis como AND e OR. Sua limitação com o XOR destaca a necessidade de redes mais complexas.

# 3 Exercício 2: Implementação do Algoritmo Backpropagation

## 3.1 Introdução ao Backpropagation e MLP

Redes Neurais de Múltiplas Camadas (MLP) usam camadas ocultas de neurônios entre a entrada e a saída, permitindo resolver problemas não linearmente separáveis. O algoritmo Backpropagation treina essas redes ajustando os pesos. Ele consiste em uma fase forward (cálculo da saída e do erro) e uma fase backward (propagação do erro para trás para ajustar os pesos).

## 3.2 Explicação da Implementação

### 3.2.1 Estrutura da Rede Neural

Usou-se TensorFlow/Keras para construir MLPs. A rede típica continha: uma camada de entrada com $n$ neurônios, uma camada oculta com um número variável de neurônios (ex: $2n$), e uma camada de saída com 1 neurônio (para classificação binária). O bias foi incluído nos neurônios.

### 3.2.2 Geração de Dados Booleanos

A mesma função de geração de dados do Exercício 1 foi usada para AND, OR e XOR. Para XOR com $n$ entradas, a saída é 1 se o número de entradas '1' for ímpar.

### 3.2.3 Treinamento da Rede

O treinamento utilizou otimizadores como Adam ou SGD, função de perda 'binary$_c rossentropy$'$emétrica$

## 3.3 Resultados dos Testes e Investigações

A MLP aprendeu AND, OR e XOR com $n$ entradas, atingindo 100% de acurácia com treinamento adequado.

### 3.3.1 A Importância da Taxa de Aprendizado

A taxa de aprendizado ($\eta$) controla o tamanho do ajuste nos pesos.

- **Muito baixa**: Convergência lenta.
- **Adequada**: Bom equilíbrio entre velocidade e estabilidade.
- **Muito alta**: Pode causar instabilidade e divergência (erro aumenta).

Testes com XOR ($n = 2$) variando a taxa de aprendizado (ex: 0.001, 0.01, 0.1) foram realizados. Gráficos de erro/acurácia por época (gerados no notebook) ilustrariam que taxas como 0.01 ou 0.1 geralmente funcionam bem.

### 3.3.2 A Importância do Bias

O bias permite que a função de ativação do neurônio seja deslocada, o que é crucial para que a rede aprenda fronteiras de decisão que não passam pela origem. Sem bias, a capacidade de aprendizado da rede é muito limitada. Nas implementações com Keras, o bias é usado por padrão. Experimentos (realizados no notebook) comparando redes com e sem bias para o XOR demonstrariam que a ausência de bias impede ou dificulta severamente a convergência para uma solução correta.

### 3.3.3 A Importância da Função de Ativação

Funções de ativação não lineares são essenciais nas camadas ocultas. Foram investigadas:

- **Sigmoide**: Saída entre 0 e 1. Pode sofrer de "vanishing gradients"(gradientes que diminuem muito, dificultando o treino).
- **Tangente Hiperbólica (tanh)**: Saída entre -1 e 1. Frequentemente melhor que sigmoide em camadas ocultas por ser centrada em zero, mas também pode ter vanishing gradients.
- **ReLU (Rectified Linear Unit)**: Saída é a entrada se positiva, ou 0 se negativa. Eficiente e ajuda com vanishing gradients, mas pode ter "dying ReLUs"(neurônios que sempre saem 0).

Testes com XOR ($n = 2$) usando estas funções (resultados e gráficos de convergência gerados no notebook) mostrariam que todas podem resolver o problema, mas Tanh e ReLU frequentemente convergem mais rápido ou de forma mais estável.

### 3.3.4 Testes com Diferentes Números de Entradas ($n$)

A MLP foi testada com AND, OR e XOR para $n = 2, 3, 4$.

- **AND e OR com $n$ entradas**: Aprendidas facilmente.
- **XOR com $n$ entradas (Paridade)**: Aprendida com sucesso, mas exigiu mais neurônios na camada oculta e/ou mais épocas à medida que $n$ aumentava, devido à maior complexidade.

## 3.4 Considerações Finais do Exercício 2

MLPs com Backpropagation resolvem problemas não lineares como o XOR. Taxa de aprendizado, bias e função de ativação são hiperparâmetros críticos que afetam o desempenho e a convergência.

# 4 Conclusão Geral

O Perceptron é simples e bom para dados linearmente separáveis. A MLP com Backpropagation é mais poderosa, lidando com não linearidade, mas exige ajuste cuidadoso de seus componentes e parâmetros.