Trabalho sobre Perceptron em Redes Neurais Artificiais (Até dupla)

Objetivo

Criar um material educacional em formato Jupyter Notebook (.ipynb) abordando os fundamentos do **Perceptron**, com explicações claras, fórmulas, gráficos, animações e códigos implementados manualmente (sem bibliotecas externas para o Perceptron).

---

1. Tópicos Obrigatórios

1.1 Introdução ao modelo de Perceptron (Mais detalhado no Anexo B ao final)

- **Explicação conceitual detalhada**
- Estrutura básica do Perceptron
- Fórmulas matemáticas detalhadas
- Explicação passo a passo das fórmulas e do funcionamento do algoritmo

1.2 Aplicação do Perceptron aos problemas lógicos OR e AND

- Criação manual dos datasets OR e AND
- Implementação do algoritmo do Perceptron para treinar e testar nesses datasets
- Gráficos demonstrando o treinamento e resultados
- Explicação detalhada das decisões gráficas (fronteiras de decisão)

1.3 Discussão do problema XOR e suas implicações para a evolução das redes neurais

- Demonstração gráfica e experimental da incapacidade do Perceptron em resolver o problema XOR
- Discussão conceitual sobre a limitação do modelo linear
- Contextualização histórica: como o problema XOR levou à criação de redes multicamadas (MLP)

---

2. Organização do Repositório Git

O repositório deverá seguir obrigatoriamente a seguinte estrutura:

G_NOME1_NOME2_ASSUNTO/ (pasta, onde nome é o nome do aluno)

│

├─ README.md (enfeite como desejar, mas é necessário por ao final o Anexo A)

└─ Codigo/ (pasta)

│ ├─ DATASET (base de dados utilizada) (opcional)

│ ├─ perceptron.ipynb (códigos-fontes)

│ ├─ enviromment.yml (arquivo que desceve o ambiente -ex: conda ou mamba)

│ ├─ README.md

│ └─ requirements.txt (com instruções para instalação/uso/rodar o código)

OBS: O arquivo README.md deve conter obrigatoriamente o Anexo A (reconhecimentos e licença de uso).

---

3. Instruções Específicas

3.1 Requisitos Gerais

- Todo código **deve ser implementado manualmente** (não é permitido usar bibliotecas externas para implementação do Perceptron, função de ativação, etc.).
- O notebook deve ser didático, explicativo e ilustrado com gráficos detalhados e animações claramente explicadas.
- Gráficos ou fórmulas sem explicações serão **desconsiderados**.

3.2 Commits (Boas práticas - Bônus)

Para incentivo à boas práticas, serão considerados **bônus extras** para o uso consistente e organizado dos tipos de commits abaixo:

- **feat**: nova funcionalidade ou implementação significativa
- **fix**: correção de erros ou bugs no código
- **chore**: alterações relacionadas à manutenção geral (configuração, organização de arquivos)
- **docs**: atualizações ou adições em documentações
- **style**: mudanças em formatação, espaços em branco, indentação
- **refactor**: melhorias de código sem adicionar novas funcionalidades
- **test**: adição de testes automatizados
- **perf**: melhorias de desempenho
- **ci**: alterações em configuração de integração contínua
- **build**: modificações que afetam o processo de build ou dependências

---

4. Submissão

- A submissão deverá ser feita através de **link do repositório git** diretamente no SIGAA.
- Trabalhos entregues fora do prazo ou através de outro método serão **desconsiderados**.
- Componente da dupla sem commits ou com commits insignificante será desconsiderado.
- **Qualquer tentativa de burlar a avaliação via commits é zero para a dupla, independente de quem fez a tentativa.**
- Apenas commits com histórico claro e coerente serão aceitos. O professor reserva-se o direito de analisar os commits para validar a participação dos integrantes do grupo.

---

5. Critérios de Avaliação

Critério	**Pontuação**
Clareza e organização do material (.ipynb)	0.1 pontos
Corretude das Implementações	0.1 pontos
Explicação detalhada dos tópicos com gráficos e fórmulas bem explicadas	0.1 pontos
Animações com explicações didáticas	0.1 pontos
Estrutura correta do repositório git e boas práticas de commits (bônus, apenas para quem completou os ítens anteriores, ou seja, fez todo o trabalho)	0.35 pontos
Implementar 10 funções custos ao final (pode gerar os inputs de forma aleatória (somente válido para quem fez todos os ítens anteriores) gerando seus gráficos e explicações de forma organizada	0.35 pontos

Total possível: **0.4 pontos** + até **0.7 ponto de bônus**

Professor Responsável:
Dr. Thales Levi Azevedo Valente
Universidade Federal do Maranhão (UFMA)

Boa sorte a todos!

ANEXO A

Texto para o arquivo "README.md"

Reconhecimentos e Direitos Autorais

@autor: [Seus Nomes]

@contato: [Seus Emails - se quiserem]

@data última versão: [Data de Hoje]

@versão: 1.0

@outros repositórios: [URLs - apontem para os seus Gits, se quiserem]

@Agradecimentos: Universidade Federal do Maranhão (UFMA), Professor Doutor Thales Levi Azevedo Valente, e colegas de curso.

Copyright/License

Anexo B

Detalhamento Obrigatório para o Trabalho sobre Perceptron

1. Estrutura Básica do Perceptron

- Explicação clara da estrutura:
    - **Entradas:** xi
    - **Pesos:** wi
    - **Bias (viés):** b
    - **Somatório e função de ativação**
- Representação gráfica simplificada do Perceptron indicando entradas, pesos, somatório, função de ativação e saída.
- **Implementação obrigatória**:
  Código manual que implementa a operação básica do Perceptron:

Onde:

- u é o valor intermediário antes da ativação.
- wi são os pesos.
- xi são as entradas.
- b é o bias (viés).

Observação:
Esta operação deve ser implementada explicitamente em forma de **multiplicação matricial/vetorial** (não use funções prontas como np.dot(), faça o código manual explicitamente com loops for, para reforçar o aprendizado).

---

2. Fórmulas Matemáticas Detalhadas

A seguir estão as fórmulas matemáticas que devem ser claramente detalhadas e implementadas passo a passo no notebook:

a. Multiplicação Matricial/Vetorial

- Demonstre e explique o funcionamento matemático da multiplicação entre os vetores de entrada e pesos.

$$u = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n + b$$

b. Função de Ativação

- Implementação detalhada da **função degrau unitário**:
- Explique claramente por que essa função é utilizada no Perceptron e seu impacto na decisão do neurônio.

c. Função de Custo (Erro)

- Explicação detalhada e implementação do cálculo do erro (função custo) usado no Perceptron (Erro absoluto):

$$E = y_{desejado} - y_{predito}$$

Onde:

- $y_{desejado}$ é o valor esperado.
- $y_{predito}$ é o valor calculado pelo perceptron.

d. Regra de Atualização dos Pesos e Bias

- Fórmula detalhada e implementação manual do ajuste dos pesos e bias:

Onde:

- n (taxa de aprendizado) deve ser explicada claramente no notebook.
- Demonstre claramente como esses ajustes afetam o treinamento e a convergência do modelo.

---

4. Explicação Passo a Passo do Funcionamento do Algoritmo

Obrigatório explicar detalhadamente:

- Inicialização dos pesos (aleatória ou zerados, com explicação).Fluxo completo do algoritmo:
    1. Calcular a multiplicação entre entradas e pesos.
    2. Aplicar a função de ativação.
    3. Avaliar o erro (função custo).
    4. Atualizar pesos e bias.

- Critério de parada (quando o Perceptron considera que aprendeu) ou um número x de épocas.
- Demonstrar visualmente com gráficos a convergência do algoritmo durante o treinamento (plotando os valores dos pesos e do erro em cada época).

🔹 **Implementação Manual Obrigatória**

Todos os passos acima devem ser explicitamente implementados **sem bibliotecas externas** para o funcionamento do Perceptron. É permitido o uso de bibliotecas básicas apenas para plotagem (matplotlib) e manipulação básica de listas (numpy) desde que não realize as operações matemáticas fundamentais automaticamente.

- **Permitido:** Plotagem com matplotlib, uso básico do numpy para criação e visualização de arrays simples.
- **Proibido:** np.dot(), funções pré-construídas para multiplicação vetorial/matricial, implementação automática do algoritmo de aprendizado.