Pontifícia Universidade Católica de Minas Gerais
Instituto de Ciências Exatas e Informática
Disciplina: Inteligência Artificial
Atividade: Lista 8 - IA

Prof.: Cristiane Neri Nobre
Nome: Davi Cândido de Almeida _857859

Links de implementação e teste:

GitHub: https://github.com/DaviKandido/IA-Inteligencia-Artificial/tree/main/Listas/Lista%208

Estudos Iniciais
https://github.com/DaviKandido/IA-Inteligencia-Artificial/blob/main/Listas/Lista%208/Estudando_Com_Base_Teste.ipynb:

Modelo:
https://github.com/DaviKandido/IA-Inteligencia-Artificial/blob/main/Listas/Lista%208/ModeloPy/Modelopy.py

XOR:
https://github.com/DaviKandido/IA-Inteligencia-Artificial/blob/main/Listas/Lista%208/Problema_XOR.ipynb

7_Segmentos:
https://github.com/DaviKandido/IA-Inteligencia-Artificial/blob/main/Listas/Lista%208/Problema_Display_7_Segmentos.ipynb

I. Introdução e Objetivo

O objetivo desta atividade é implementar, a partir do zero (sem o uso de bibliotecas de *deep learning* de alto nível como Keras ou PyTorch), o algoritmo de **Backpropagation** para treinar uma Rede Neural Simples (RNS) com uma única camada oculta.

A RNS implementada possui uma arquitetura básica composta por uma camada de entrada, uma camada oculta e uma camada de saída.

II. Fundamentação Teórica e Implementação

2.1. Função de Ativação e Perda

As funções de ativação são utilizadas para introduzir não-linearidade na rede, permitindo o aprendizado de funções complexas como o XOR.

- **Função Sigmoide**: Utilizada na camada oculta e na camada de saída (para o problema binário). Sua fórmula é dada por:

$$\delta(x) = \frac{1}{1 + e^{-1}}$$

- **Funções Investigadas (Camada Oculta):** Para a camada oculta, a função tanh (tangente hiperbólica) foi utilizada, pois tende a acelerar a convergência do treinamento em alguns casos.

- **Funções de Saída e Perda:**
  - **Problema XOR (Binário):** Função de ativação **Sigmóide**.
  - **Problema 7 Segmentos (Multi-classe):** Função de ativação **Softmax** na saída, mais apropriada para medir a probabilidade de 10 classes.

III. Problema 1: O Problema do XOR

O problema do XOR é a tarefa não-linear mais simples, exigindo uma camada oculta para sua solução.

- **Configuração do Modelo:**
  - **Entrada (X):** 4 x 2 (Amostras vs. Features: [[0, 0], [0, 1], [1, 0], [1, 1]]).
  - **Rótulo (Y):** 4 x1 (Classes: [[0], [1], [1], [0]]).
  - **Estrutura da RN:** (2 neurônios de entrada, 3 neurônios ocultos, 1 neurônio de saída)
  - **Parâmetros:** taxa de aprendizagem = 0.1, 1000 épocas.

Resultados do Treinamento:

Métrica	**Valor**
Acurácia Final	**1.000 (100%)**
Perda Final (Loss)	Baixa (próxima de 0)

Saída de Console (Exemplo):

```
hidden_neurons = 3 # 3 neurónios na camada intermediaria/oculta
output_neurons = 1 # 2 classes, ou seja 2 neurónios na camada de Saida
random_seed = 8
learning_rate = 0.1 # taxa de aprendizado
epochs = 1000 # épocas de treinamento
```

✓ 0.0s

```
modelo = RnModel(



    x, y,
    hidden_neurons=hidden_neurons,
    output_neurons=output_neurons,
    random_seed=random_seed
)
result = modelo.fit(learning_rate=learning_rate, epochs=epochs)

```

✓ 0.1s

```
Epoch: [100 / 1000] Accuracy: 0.500 Loss: 0.6709 Correct: 2 Total: 4
Epoch: [200 / 1000] Accuracy: 1.000 Loss: 0.5944 Correct: 4 Total: 4
Epoch: [300 / 1000] Accuracy: 1.000 Loss: 0.4204 Correct: 4 Total: 4
Epoch: [400 / 1000] Accuracy: 1.000 Loss: 0.2482 Correct: 4 Total: 4
Epoch: [500 / 1000] Accuracy: 1.000 Loss: 0.1531 Correct: 4 Total: 4
Epoch: [600 / 1000] Accuracy: 1.000 Loss: 0.1046 Correct: 4 Total: 4
Epoch: [700 / 1000] Accuracy: 1.000 Loss: 0.0775 Correct: 4 Total: 4
Epoch: [800 / 1000] Accuracy: 1.000 Loss: 0.0608 Correct: 4 Total: 4
Epoch: [900 / 1000] Accuracy: 1.000 Loss: 0.0496 Correct: 4 Total: 4
Epoch: [1000 / 1000] Accuracy: 1.000 Loss: 0.0418 Correct: 4 Total: 4
```

Gráfico do Histórico de Perda:

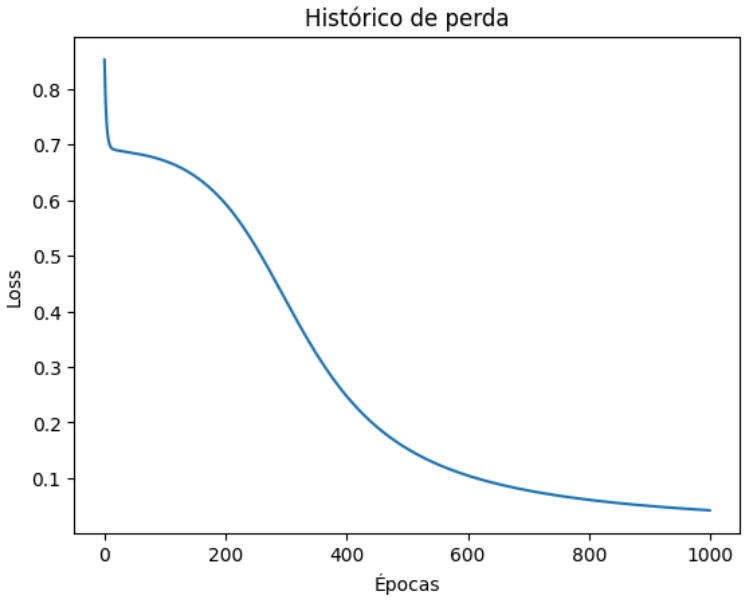
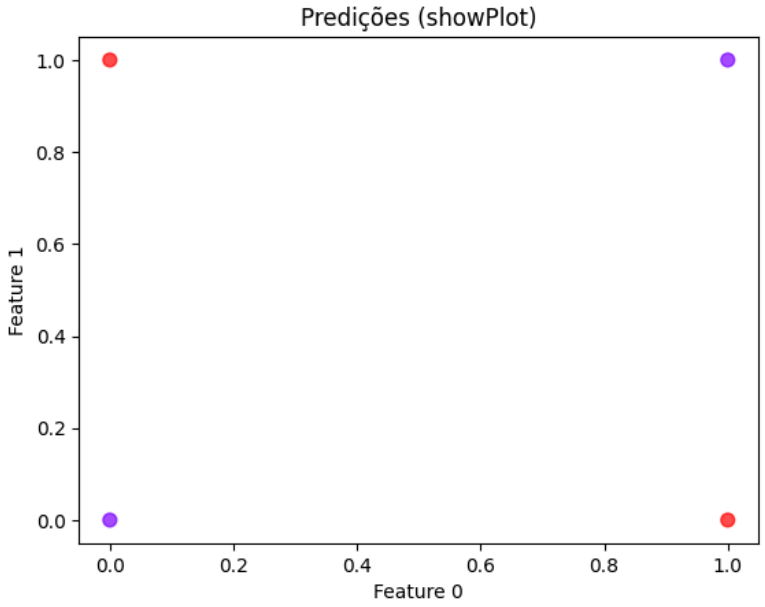


Gráfico de Predição (Visualização 2D):



IV. Problema 2: Dígitos Binários de um Display de 7 Segmentos

Este problema é de classificação multi-classe, exigindo 10 saídas e Softmax.

- **Configuração do Modelo:**
  - **Entrada (X):** 10 x 7 (Segmentos ‘a’ a ‘g’).
  - **Rótulo (Y):** 10 x 1 (Índices de 0 a 9).
  - **Estrutura da RN:** (7 neurônios de entrada , 5 neurônios ocultos, **10 neurônios de saída**).
    - **Nota:** Embora o documento sugira 4 neurônios de saída , 10 neurônios são essenciais para codificar 10 classes usando Softmax/One-Hot.
  - **Parâmetros:** taxa de aprendizagem = 0.1, 5000 épocas.

Resultados do Treinamento:

Métrica	Valor
Acurácia Final	**1.000 (100%)**
Perda Final (Loss)	Baixa (próxima de 0)

Saída de Console (Exemplo):

```python
#Importando o modelo da Rede Neural

from ModeloPy.Modelopy import RnModel
```

0.0s

```python
hidden_neurons = 5 # 1 neurónios na camada intermediaria/oculta
output_neurons = 10 # 2 classes, ou seja 2 neurónios na camada de Saida
random_seed = 8
learning_rate = 0.1 # taxa de aprendizado
epochs = 5000 # épocas de treinamento
```

0.0s

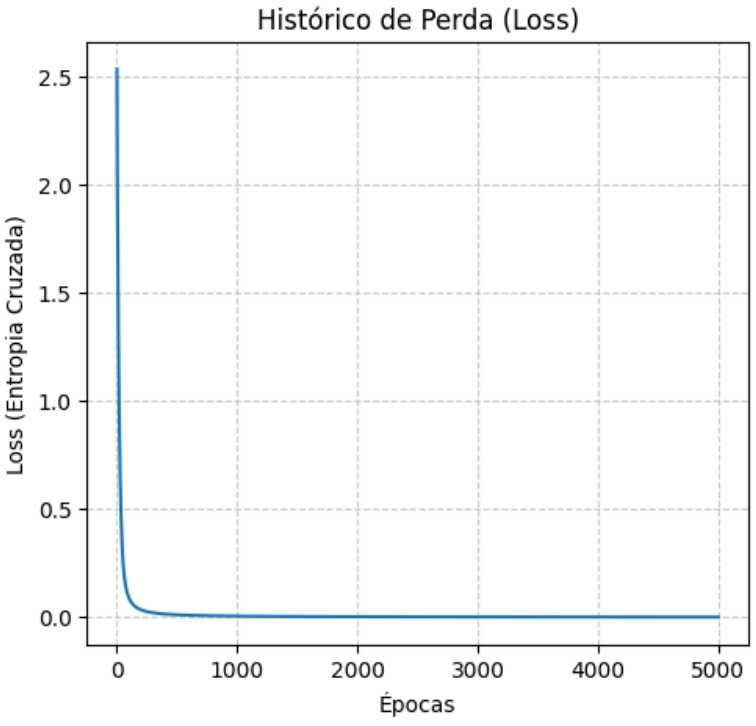
```python
modelo = RnModel(
    x, y,
    hidden_neurons=hidden_neurons,
    output_neurons=output_neurons,
    random_seed=random_seed,
    activation="tanh"
)
result = modelo.fit(learning_rate=learning_rate, epochs=epochs)

```

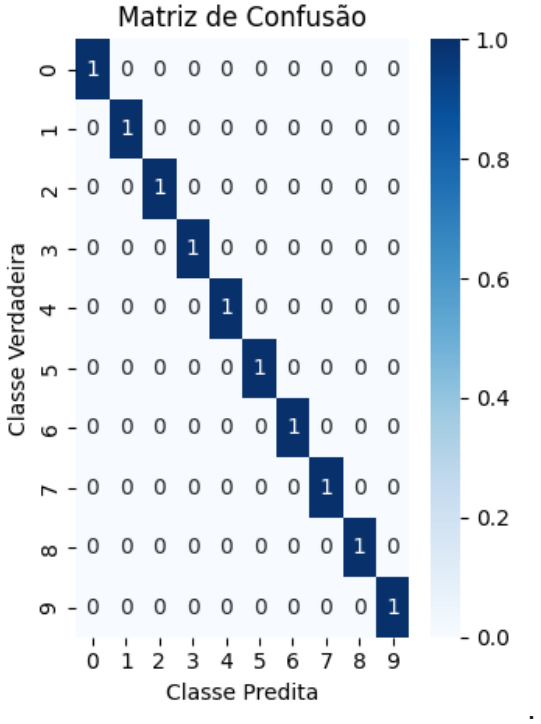
0.5s

```
Epoch: [500 / 5000] Accuracy: 1.000 Loss: 0.0115 Correct: 10 Total: 10
Epoch: [1000 / 5000] Accuracy: 1.000 Loss: 0.0054 Correct: 10 Total: 10
Epoch: [1500 / 5000] Accuracy: 1.000 Loss: 0.0035 Correct: 10 Total: 10
Epoch: [2000 / 5000] Accuracy: 1.000 Loss: 0.0026 Correct: 10 Total: 10
Epoch: [2500 / 5000] Accuracy: 1.000 Loss: 0.0020 Correct: 10 Total: 10
Epoch: [3000 / 5000] Accuracy: 1.000 Loss: 0.0017 Correct: 10 Total: 10
Epoch: [3500 / 5000] Accuracy: 1.000 Loss: 0.0014 Correct: 10 Total: 10
Epoch: [4000 / 5000] Accuracy: 1.000 Loss: 0.0013 Correct: 10 Total: 10
Epoch: [4500 / 5000] Accuracy: 1.000 Loss: 0.0011 Correct: 10 Total: 10
Epoch: [5000 / 5000] Accuracy: 1.000 Loss: 0.0010 Correct: 10 Total: 10
```

Gráfico do Histórico de Perda:



Matriz de Confusão (Distribuição dos Acertos):



4.1.Robustez do Modelo (Adicionando Ruído)

Para testar a robustez, foram introduzidas entradas com ruído

Teste de Ruído: Foi simulada a falha de um segmento ativo (trocando 1 por 0) em cada dígito.

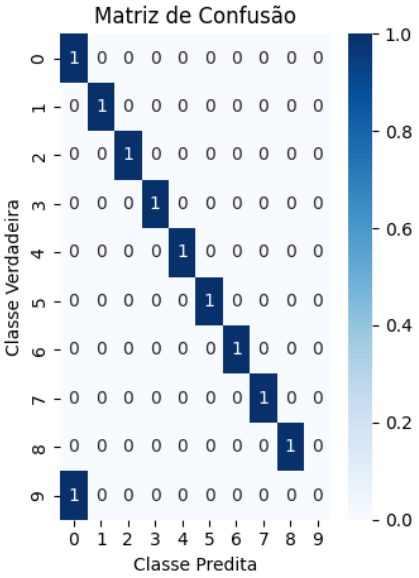
Exemplo de Resultados com Ruído:

Dígito Original	Entrada com Ruído (Exemplo)	Acerto?	Número que ela concluiu
8	(1, 1, 1, 1, 1, 1, **0**) - Sem o segmento G	**SIM**	**8**
6	(1, **1**, 1, 1, 1, 1, 1) - Com segmento B ativo	**SIM**	**6**
9	(0, 1, 1, 0, **0**, 1, 1) - Sem o segmento E	**NÃO**	**0**

Observação: Pressupõe que o modelo apesar de um número limitado de amostras de treinamento, apresentando poucas falhas ao receber entradas com ruído, mas

acredita-se que alterações/falhas mais bruscas nos dados resultam em ruídos de maior graus entre os erros podendo demonstrar uma problema de robustez do modelo.

4.2. Matriz de Confusão (Distribuição dos Acertos):



V. Conclusão

A atividade de implementação do Backpropagation demonstrou sucesso na solução de tarefas de classificação binária (XOR) e multi-classe (7 Segmentos). A arquitetura de Rede Neural Simples, combinada com funções de ativação adequadas (tanh/Sigmoid e Softmax), permitiu atingir acurácia de **100%** nos dados de treinamento originais. O principal desafio técnico residiu em garantir a correta propagação dos deltas de erro (Backpropagation) e o mapeamento adequado entre as perdas e suas respectivas funções de ativação. A etapa de testes com ruído reforçou que, para a robustez em cenários reais, o conjunto de dados de treinamento deve ser ampliado para incluir variações e falhas.