

Lista 7 - IA (extra)

Sophia Carrazza Ventorim de Sousa

PUC Minas - 2024

1- Perceptron:

Código utilizado:

<https://colab.research.google.com/drive/1vf5RSVYIWMukyfxHlBdcd8Fk6hR-9fB1?usp=sharing>

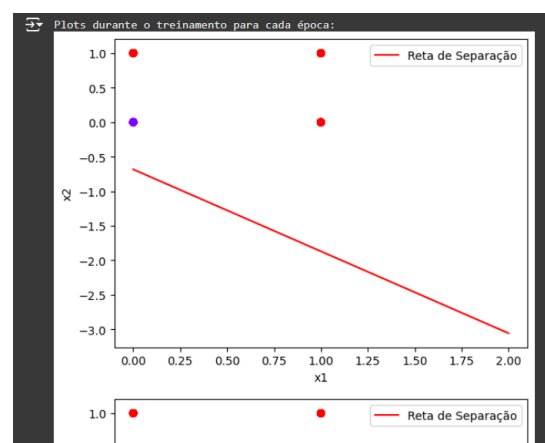
```
Interface de Usuário

[159] # Mini Interface
problema = "OR" #param ["AND", "OR", "XOR"]
quantidade_de_entradas = 100 #param [type:
num_entradas = quantidade_de_entradas

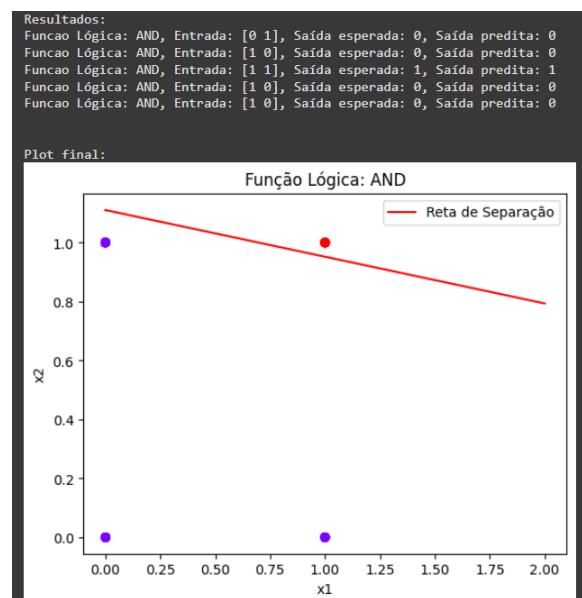
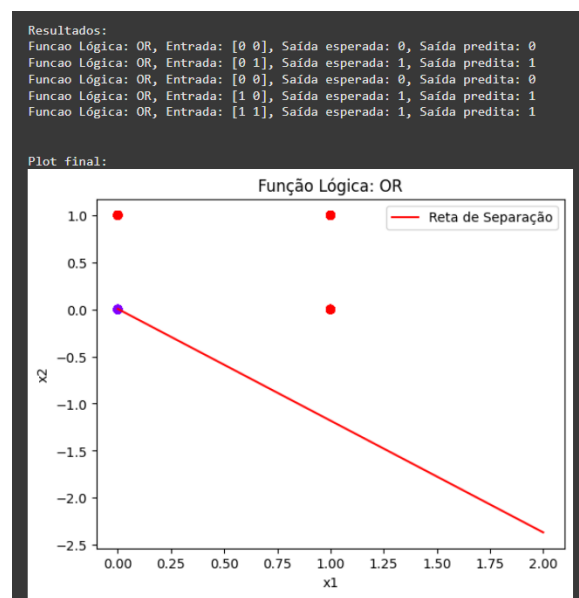
entradas, saidas = dados(problema, num_entr
perceptron = Perceptron(entradas.shape[1])
print(f"Plots durante o treinamento para c
perceptron.fit(entradas, saidas)

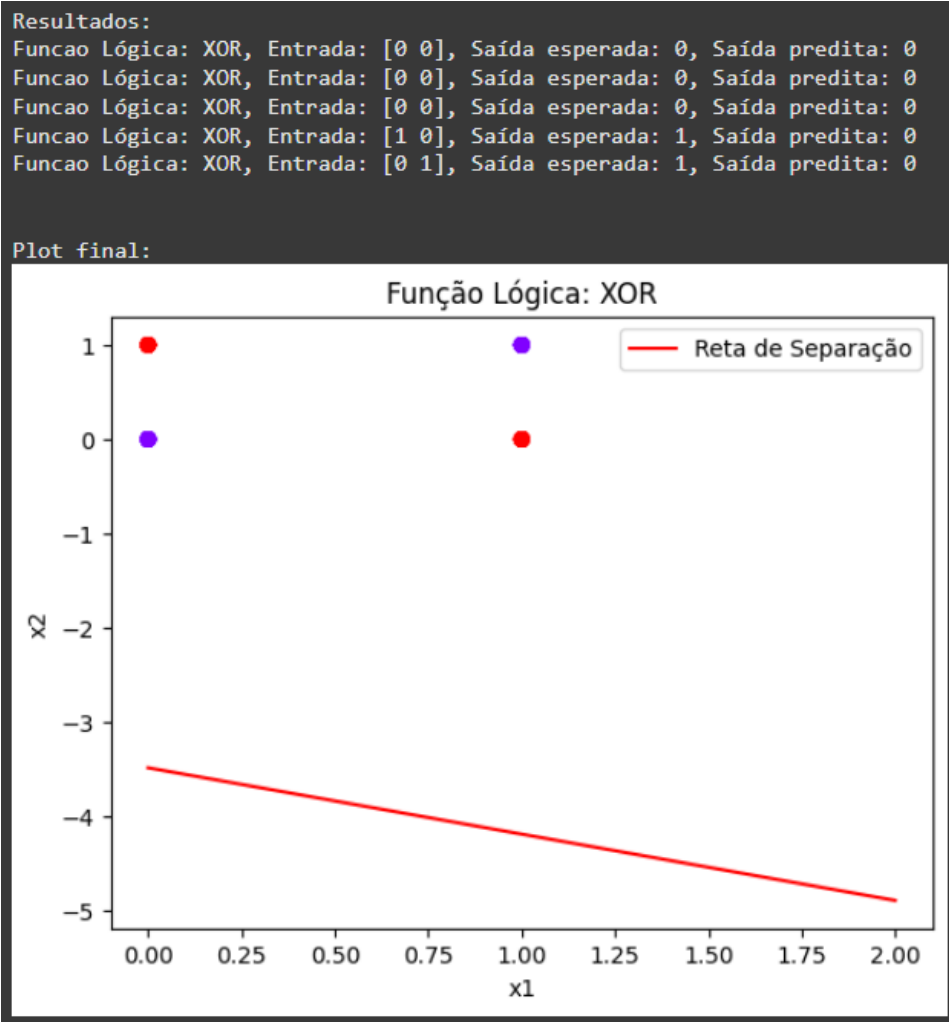
print("\n")
print("Resultados:")
for entrada, saida_esperada in zip(entradas
saida_predita = perceptron.predicao(ent
print(f"Funcao Lógica: {problema}, Entr

print("\n")
# Plot da reta de separação
x1 = np.linspace(0, 2, 100) #valores para
x2 = -(perceptron.pesos[0] * x1 + perceptrc
print("Plot final:")
plt.figure()
plt.title(f"Função Lógica: {problema}")
plt.xlabel("x1")
plt.ylabel("x2")
plt.scatter(entradas[:, 0], entradas[:, 1])
plt.plot(x1, x2, color="red", label="Reta d
plt.legend()
plt.show()
```



Resultados:





Conclusões:

O código implementa um Perceptron para classificar dados binários random com base em uma função lógica (AND, OR e XOR) de escolha do usuário. O Perceptron usa uma função de ativação limiar para a saída (0 ou 1) e também visualiza a evolução da linha de separação em cada época no hiperplano.

O Perceptron foi capaz de ajustar seus pesos para aprender as funções lógicas AND e OR, já que são problemas linearmente separáveis, mas não conseguiu aprender a função XOR, pois não há uma reta de separação válida para esse problema: o XOR não é linearmente separável.

*O código está propriamente documentado com comentários.

2- Backpropagation:

Código utilizado:

<https://colab.research.google.com/drive/1OC4FNzbwdyc2wELaVHB5KNEz0TQxZFqj?usp=sharing>

```
# testando a rede
print("Resultados:")
for entrada, alvo in zip(entradas, alvos):
    saida_continua = rede_neural.propagacao_frente(entrada) # saída da rede
    saida_binaria = int(round(saida_continua[0])) # converte para binário (0 ou 1)
    print(f"Entrada: {entrada}, Saída esperada: {int(alvo[0])}, Saída prevista: {saida_binaria}")
```

Resultados:
Entrada: [0 0], Saída esperada: 0, Saída prevista: 0
Entrada: [0 1], Saída esperada: 0, Saída prevista: 0
Entrada: [1 0], Saída esperada: 0, Saída prevista: 0
Entrada: [1 1], Saída esperada: 1, Saída prevista: 1

Conclusões e Investigações:

O código implementa uma rede neural simples com uma camada escondida, demonstrando a implementação com a função AND. A rede usa feedforward para calcular as saídas e backpropagation para ajustar os pesos com base no erro calculado.

Para a função AND, a rede conseguiu aprender a relação corretamente, e previu saídas binárias de acordo com a tabela verdade correta.

*O código está propriamente documentado com comentários.

- 1)** A taxa de aprendizado é essencial nos algoritmos de redes neurais, já que ela controla o tamanho dos ajustes feitos nos pesos durante a retropropagação. Uma taxa muito alta pode levar a ajustes muito grandes e "pulos", perdendo o resultado ótimo.
- 2)** O Bias faz com que a rede possa aprender melhor mesmo quando a saída desejada não é diretamente proporcional às entradas com ponderação.
- 3)** Foram testadas as funções de ativação ReLu, Tanh e Sigmoid. A derivada dessa função é usada no cálculo do gradiente durante o backpropagation. Quanto mais próximo esse valor estiver de zero, mais lento será o ajuste dos pesos (causando o problema do vanishing gradient). Logo, essa derivada deve ser escolhida de forma a estar bem personalizada aos dados treinados.
- 4)** O Backpropagation usa funções não-lineares pois isso permite a formação de relações complexas entre dados, resolvendo problemas difíceis de serem resolvidos com uma única transformação linear. Assim, no backpropagation, os pesos podem ser devidamente ajustados para entender padrões dos dados de treinamento.

3- Reconhecimento de Imagens:

*Este código foi rodado no VScode, pois o colab não suportou a quantidade de imagens do dataset.

Código utilizado:

```
https://drive.google.com/file/d/1jwNjLnjEb2kP7lzzZhRBGzcr-1A9f92_/view?usp=sharing
```

O que foi feito:

- Aumento do número de camadas (3 camadas com 64, 128, e 128 filtros);
- Adição do dropout (50%) na camada densa para evitar overfitting;
- Usei "image_dataset_from_directory" em vez de "ImageDataGenerator";
- Otimização feita com Adam com taxa de aprendizado de 0.001.

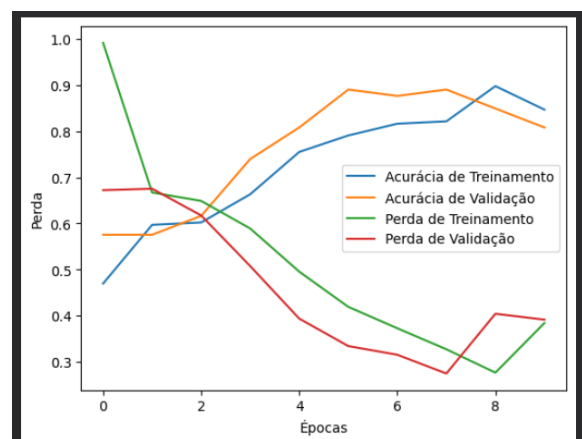
Modelo:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 64)	1,792
max_pooling2d (MaxPooling2D)	(None, 63, 63, 64)	0
conv2d_1 (Conv2D)	(None, 61, 61, 128)	73,856
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 128)	0
conv2d_2 (Conv2D)	(None, 28, 28, 128)	147,584
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 256)	6,422,784
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 1)	256

Total params: 6,646,273 (25.35 MB)

Trainable params: 6,646,273 (25.35 MB)

Non-trainable params: 0 (0.00 B)



Resultados:

```
# Previsão
prediction = model.predict(img_array)
print("Predição:", "Homer" if prediction[0][0] > 0.5 else "Bart")

[51] ✓ 0.1s

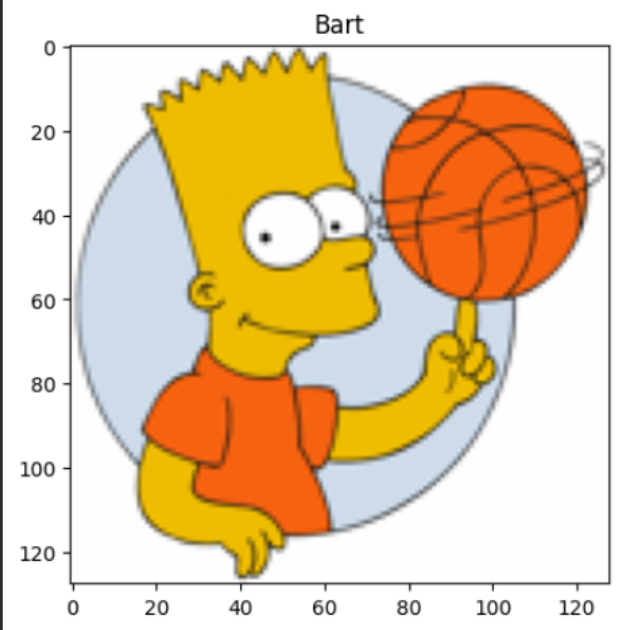
... 1/1 ————— 0s 75ms/step
Predição: Bart

▶ ▾

plt.imshow(img)
plt.title("Homer" if prediction[0][0] > 0.5 else "Bart")
plt.show()

[53] ✓ 0.1s

...


```

- Acurácia no treinamento: 89% após 10 épocas.
- Acurácia na validação: 83,5%.
- O aumento no número de filtros e a inclusão do dropout auxiliaram na redução do overfitting.