PC.2. Implemente, utilizando a linguagem de programação Python ou JavaScript, uma rede neural que resolva o problema do AND lógico. Seu programa não precisa efetuar nenhum processo de aprendizagem e pode utilizar pesos definidos de modo arbitrário por você. Ilustre dois casos utilizando as seguintes funções de ativação: função de Heaviside e a função ReLU. Diga se os pesos utilizados foram os mesmos quando houve alteração da função de ativação.

O que é uma função de ativação?

É uma função matemática usada por um neurônio para decidir **se ativa (dispara)** ou não, dado um valor de entrada.

Função de Ativação Heaviside (Degrau):
Também chamada de **função degrau**, ativa o neurônio somente quando a entrada atinge ou ultrapassa um limite, ideal para: lógica booleana.
Fórmula:

$$f(x) = \begin{cases} 0 & \text{se } x < 0 \\ 1 & \text{se } x \geq 0 \end{cases}$$

Função de Ativação ReLU (Rectified Linear Unit):
A função ReLU só deixa passar valores positivos. Tudo que é menor que zero vira zero, ideal para: deep learning e regressão.
Fórmula:

$$f(x) = \max(0, x)$$

Tabela verdade do AND

Entrada x1	Entrada x2	Saída esperada
0	0	0
0	1	0
1	0	0
1	1	1

mesmos pesos arbitrários [1, 1, -1.5] em ambos os casos.

```
# Pesos arbitrários escolhidos (w1, w2, bias)
pesos = np.array([1, 1, -1.5])  # w1 = 1, w2 = 1, bias = -1.5
```

Função de ativação Heaviside:

```
### Função de ativação: Heaviside
def heaviside(x):
    return 1 if x >= 0 else 0
```

```
Usando função de ativação: heaviside
Entrada: [0 0] -> Soma: -1.50, Saída: 0, Esperado: 0
Entrada: [0 1] -> Soma: -0.50, Saída: 0, Esperado: 0
Entrada: [1 0] -> Soma: -0.50, Saída: 0, Esperado: 0
Entrada: [1 1] -> Soma: 0.50, Saída: 1, Esperado: 1
```

Função de ativação ReLu:

```
### Função de ativação: ReLU
def relu(x):
    return max(0, x)
```

```
Usando função de ativação: relu
Entrada: [0 0] -> Soma: -1.50, Saída: 0, Esperado: 0
Entrada: [0 1] -> Soma: -0.50, Saída: 0, Esperado: 0
Entrada: [1 0] -> Soma: -0.50, Saída: 0, Esperado: 0
Entrada: [1 1] -> Soma: 0.50, Saída: 0.5, Esperado: 1
```

```python
### Função que executa a rede com uma função de ativação qualquer
def rede_and(entradas, pesos, ativacao):
    print(f"Usando função de ativação: {ativacao.__name__}")
    for entrada, saida_esperada in zip(entradas, saidas_esperadas):
        soma = np.dot(entrada, pesos)
        saida = ativacao(soma)
        print(f"Entrada: {entrada[:2]} -> Soma: {soma:.2f}, Saída: {saida}, Esperado: {saida_esperada}")
    print()
```

A diferença nos resultados vêm da **função de ativação**:

- A **Heaviside** produz **valores binários** (0 ou 1), ideal para lógica booleana como o AND.
- A **ReLU** produz **valores contínuos ≥ 0**, o que pode não coincidir com as saídas esperadas sem um limiar adicional.

O que mudou foi apenas a função de ativação, e não os pesos. Isso demonstra que a saída final da rede depende não só dos pesos e entradas, mas também da função de ativação aplicada.