Problem 3: Perceptrons

Brandon Mejia - 79261
Catarina Casanova - 72750
Miguel Correia - 71369

November 11, 2025

Contents

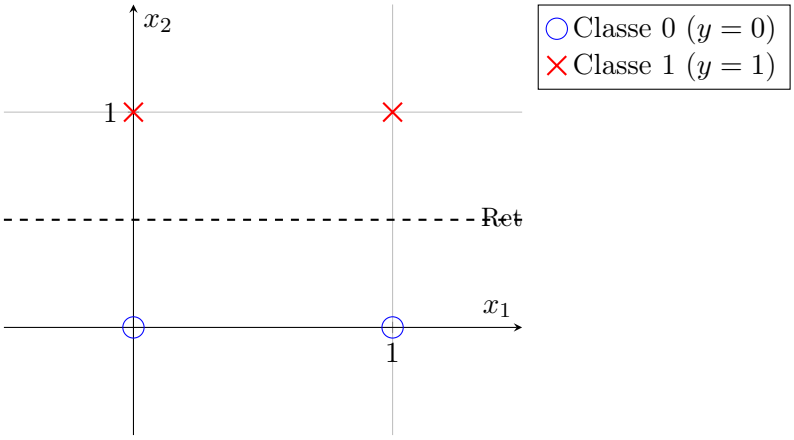
1 Tarefa 1: Propor um conjunto de dados

A tarefa pede um conjunto de dados com 4 exemplos que seja **linearmente separável**. Propomos a função lógica $\mathbf{y = x_2}$.

A tabela de verdade para esta função é:

x_1	x_2	y (Saída)
0	0	**0**
0	1	**1**
1	0	**0**
1	1	**1**

Como a função é $y = x_2$, os pontos da "Classe 0" ($y = 0$) são (0,0) e (1,0). Os pontos da "Classe 1" ($y = 1$) são (0,1) e (1,1). Este problema é **linearmente separável**, como se vê no plano cartesiano abaixo.



2 Tarefa 2: Calcular pesos para um único neurónio

A tarefa pede para calcular, sem treino, os pesos de um **único neurónio** que resolva o conjunto de dados da Tarefa 1 com erro zero.

Usaremos a **Função Degrau (Binary Step)** como função de ativação:

$$\text{Saída} = \begin{cases} 1 & \text{se } z \geq 0 \\ 0 & \text{se } z < 0 \end{cases} \tag{1}$$

A soma ponderada (input líquido) é $z = w_0 + w_1 x_1 + w_2 x_2$.

Para resolver $y = x_2$, precisamos de encontrar w_0, w_1, w_2 que satisfaçam:

- **(0, 0) → 0:** $z = w_0 < 0$
- **(0, 1) → 1:** $z = w_0 + w_2 \geq 0$
- **(1, 0) → 0:** $z = w_0 + w_1 < 0$
- **(1, 1) → 1:** $z = w_0 + w_1 + w_2 \geq 0$

Uma solução simples de pesos inteiros é:

- $w_0 = -1$ (satisfaz $w_0 < 0$)
- $w_2 = 1$ (satisfaz $w_0 + w_2 = -1 + 1 = 0 \geq 0$)
- $w_1 = 0$ (satisfaz $w_0 + w_1 = -1 + 0 = -1 < 0$)

Pesos do Neurónio Único: $w_0 = -1$, $w_1 = 0$, $w_2 = 1$.

3 Tarefa 3: Implementar e testar o neurónio

Testamos o neurónio único configurado na Tarefa 2 com os pesos $w_0 = -1, w_1 = 0, w_2 = 1$.

Entrada		Cálculo ($z = -1 + 0 \cdot x_1 + 1 \cdot x_2$)	Saída y	Esperada
x_1	x_2	z	(se $z \geq 0$)	
0	0	$z = -1 + 0 + 0 = -1$	**0**	**0**
0	1	$z = -1 + 0 + 1 = 0$	**1**	**1**
1	0	$z = -1 + 0 + 0 = -1$	**0**	**0**
1	1	$z = -1 + 0 + 1 = 0$	**1**	**1**

O neurónio único funciona perfeitamente para o problema linearmente separável.

3.1 Implementação em Java

Abaixo está o código-fonte em Java que implementa o neurónio único (usando a classe MLP como um neurónio único) e o testa. O código usa os pesos $w_1 = 0$, $w_2 = 1$, $w_0(bias) = -0.5$, que também são uma solução válida.

```java
package apps;

import math.Matrix;
import neural.MLP;
import neural.activation.IDifferentiableFunction;
import neural.activation.Step;

import java.util.Scanner;

public class SingleNeuron0101 {

    public static void main(String[] args) {
        // 1. Define the dataset for the logic y = x2 (pattern 0101)
        double[][] inputs = {
                {0, 0},
                {0, 1},
                {1, 0},
                {1, 1}
        };
        double[] expectedOutputs = {0, 1, 0, 1};

        // 2. Define the weights and bias calculated manually
        // For y = x2, we can use weights that focus on x2 and ignore
           x1.
        Matrix weights = new Matrix(new double[][]{{0.0}, {1.0}}); //
           Shape: 2x1
        double bias = -0.5;

        // 3. Create and configure an MLP to act as a single neuron
        int[] topology = {2, 1}; // 2 inputs, 1 output neuron
        IDifferentiableFunction[] activations = {new Step()};
        MLP mlp = new MLP(topology, activations, -1);

        // Use setters to apply the manually calculated weights and
           bias
        mlp.setWeights(new Matrix[]{weights});
        mlp.setBiases(new double[]{bias});
```

```
        System.out.println("Testing an MLP as a single neuron for the
            logic y = x2 (pattern 0101):");
        System.out.println("Weights: w1=0.0, w2=1.0, Bias: -0.5");
        System.out.println("----------------------------------------")
            ;

        // Test with the predefined dataset
        Matrix testMatrix = new Matrix(inputs);
        Matrix predictions = mlp.predict(testMatrix);

        for (int i = 0; i < inputs.length; i++) {
            double prediction = predictions.get(i, 0);
            System.out.printf("Input: [%.0f, %.0f] -> Predicted: %.0f,
                Expected: %.0f\n",
                    inputs[i][0], inputs[i][1], prediction,
                        expectedOutputs[i]);
        }

        // --- Interactive Testing Part ---
        System.out.println("\n--- Interactive Test Mode ---");
        System.out.println("Enter the number of pairs to test, followed
            by the pairs (e.g., 1.0 0.9)");

        Scanner sc = new Scanner(System.in);
        int n;
        try {
            n = sc.nextInt();
        } catch (java.util.InputMismatchException e) {
            System.err.println("\nError: Invalid input. Please enter an
                integer for the number of pairs first.");
            sc.close();
            return; // Exit the program gracefully
        }

        for (int i = 0; i < n; i++) {
            // Create an array for the two input values
            double[][] testInput = new double[1][2];
            testInput[0][0] = Double.parseDouble(sc.next().replace(',',
                '.'));
            testInput[0][1] = Double.parseDouble(sc.next().replace(',',
                '.'));

            // Get the neuron's prediction
            double prediction = mlp.predict(new Matrix(testInput)).get
                (0, 0);

            System.out.printf("Input: [%.1f, %.1f] -> Predicted: %d\n",
                testInput[0][0], testInput[0][1], (int)prediction);
        }
        sc.close();
    }
}
```

Listing 1: SingleNeuron0101.java

4 Tarefa 4: Configurar a Rede Multicamada (MLP) para NOT XOR

A tarefa pede para configurar a rede **multilayer perceptron** para que funcione como uma função **NOT XOR** (também conhecida como XNOR). Este é um problema clássico **não linearmente separável**.

4.1 Lógica (Álgebra Booleana)

A tabela de verdade para o XNOR é:

x_1	x_2	y (Saída)
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**1**

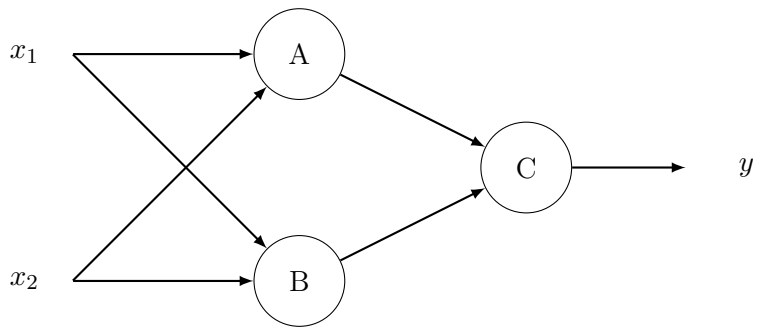
Decompomos a lógica em duas partes que são linearmente separáveis, e depois combinamolas:

$$y = (\neg x_1 \cdot \neg x_2) + (x_1 \cdot x_2) \tag{2}$$

Isto significa que o Neurónio A irá implementar $(\neg x_1 \cdot \neg x_2)$ (Porta NOR), o Neurónio B irá implementar $(x_1 \cdot x_2)$ (Porta AND), e o Neurónio C irá combinar os resultados (Porta OR).

4.2 Diagrama da Rede Neuronal

Usamos a mesma arquitetura 2-2-1:



4.3 Condição (Binary Step)

Para todos os neurónios (A, B, e C), usamos a mesma Função Degrau:

$$\text{Saída} = \begin{cases} 1 & \text{se } z \geq 0 \\ 0 & \text{se } z < 0 \end{cases} \tag{3}$$

4.4 Resolução (Configurar os Pesos da MLP)

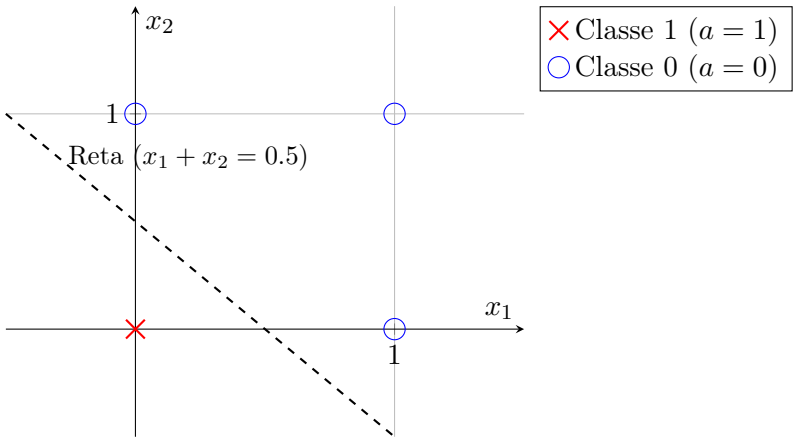
Configuramos os pesos de cada neurónio com base nos cálculos das notas.

4.4.1 Neurónio A $= \neg x_1 \cdot \neg x_2$ (Porta NOR)

Deve dar 1 **apenas** para (0, 0).

- $z_A = w_{0,A} + w_{1,A} \cdot x_1 + w_{2,A} \cdot x_2$
- **Pesos (das notas):** $w_0 = 0.5$, $w_1 = -1$, $w_2 = -1$.
- **Verificação:**
  - (0,0): $z = 0.5 + (-1) \cdot 0 + (-1) \cdot 0 = 0.5 \geq 0 \rightarrow 1$ (Correto)
  - (0,1): $z = 0.5 + (-1) \cdot 0 + (-1) \cdot 1 = -0.5 < 0 \rightarrow 0$ (Correto)
  - (1,0): $z = 0.5 + (-1) \cdot 1 + (-1) \cdot 0 = -0.5 < 0 \rightarrow 0$ (Correto)
  - (1,1): $z = 0.5 + (-1) \cdot 1 + (-1) \cdot 1 = -1.5 < 0 \rightarrow 0$ (Correto)

O plano de separação deste neurónio é:



4.4.2 Neurónio B $= x_1 \cdot x_2$ (Porta AND)

Deve dar 1 apenas para (1, 1).

- $z_B = w_{0,B} + w_{1,B} \cdot x_1 + w_{2,B} \cdot x_2$
- **Pesos (das notas):** $w_0 = -1.5$, $w_1 = 1$, $w_2 = 1$
- **Verificação:**
  - (0,0): $z = -1.5 + 1 \cdot 0 + 1 \cdot 0 = -1.5 < 0 \rightarrow 0$ (Correto)
  - (0,1): $z = -1.5 + 1 \cdot 0 + 1 \cdot 1 = -0.5 < 0 \rightarrow 0$ (Correto)
  - (1,0): $z = -1.5 + 1 \cdot 1 + 1 \cdot 0 = -0.5 < 0 \rightarrow 0$ (Correto)
  - (1,1): $z = -1.5 + 1 \cdot 1 + 1 \cdot 1 = 0.5 \geq 0 \rightarrow 1$ (Correto)

4.4.3 Neurónio C $= A + B$ (Porta OR)

Recebe as saídas a (de A) e b (de B). Deve dar 1 se $a = 1$ ou $b = 1$.

- $z_C = w_{0,C} + w_{A,C} \cdot a + w_{B,C} \cdot b$
- **Pesos (das notas):** $w_0 = -0.5$, $w_A = 1$, $w_B = 1$
- **Verificação:**
  - (a=0, b=0): $z = -0.5 + 1 \cdot 0 + 1 \cdot 0 = -0.5 < 0 \rightarrow 0$ (Correto)
  - (a=1, b=0): $z = -0.5 + 1 \cdot 1 + 1 \cdot 0 = 0.5 \geq 0 \rightarrow 1$ (Correto)
  - (a=0, b=1): $z = -0.5 + 1 \cdot 0 + 1 \cdot 1 = 0.5 \geq 0 \rightarrow 1$ (Correto)

4.5 Verificação Final da Rede Completa

Testamos a rede inteira, passo a passo, com todas as entradas originais x_1, x_2.

Entrada		Neurónio A (NOR)		Neurónio B (AND)		Neurónio C (OR)		Saída
x_1	x_2	z_A	Saída a	z_B	Saída b	z_C	Saída y	Esperada
0	0	0.5	**1**	-1.5	**0**	0.5	**1**	**1**
0	1	-0.5	**0**	-0.5	**0**	-0.5	**0**	**0**
1	0	-0.5	**0**	-0.5	**0**	-0.5	**0**	**0**
1	1	-1.5	**0**	0.5	**1**	0.5	**1**	**1**

A rede (MLP) funciona perfeitamente para a função NOT XOR.

5 Tarefa 5: Treino com Backpropagation (NOT XOR)

Esta secção analisa os resultados do treino da rede para a função NOT XOR, utilizando o algoritmo de *backpropagation*.

5.1 Comparação dos Pesos (Manuais vs. Treinados)

Comparamos os pesos calculados manualmente na Tarefa 4 com os pesos médios obtidos após 15 execuções de treino.

5.1.1 Pesos Manuais (Baseados em Lógica)

Estes pesos foram desenhados para serem logicamente interpretáveis (NOR, AND, OR).

	Neurónio A (NOR)	**Neurónio B (AND)**
w_0 (Bias)	0.5	-1.5
w_1 (de x_1)	-1	1
w_2 (de x_2)	-1	1

	Neurónio C (OR)
w_0 (Bias)	-0.5
w_A (de A)	1
w_B (de B)	1

5.1.2 Pesos Treinados (Média de 15 Amostras)

Estes pesos são o resultado da otimização matemática do *backpropagation*.

	Neurónio A (H0)	**Neurónio B (H1)**
w_0 (Bias)	-2.9534	-2.2084
w_1 (de x_1)	3.7568	4.6566
w_2 (de x_2)	5.2512	4.6123

	Neurónio C (Out0)
w_0 (Bias)	1.8347
w_A (de H0)	0.6166
w_B (de H1)	0.5152

5.1.3 Breve Explicação

Como se pode observar ao comparar as tabelas, os pesos obtidos através do treino **não são de todo similares** aos pesos que calculámos manualmente.

- **Não existe uma solução única:** A "superfície de erro" de uma rede neuronal é complexa e não-convexa. Isto significa que existem múltiplas (virtualmente infinitas) combinações de pesos que resolvem o problema XNOR perfeitamente. O treino, especialmente com 15 inicializações aleatórias diferentes, irá encontrar 15 soluções válidas, mas diferentes. A média desses pesos representa apenas uma solução matemática, não a única.
- **Inicialização Aleatória:** O *backpropagation* inicia com pesos aleatórios. A solução específica que o algoritmo encontra depende muito desse ponto de partida. A nossa solução manual (Tarefa 4) foi um "ponto de partida" lógico e deliberado.
- **Lógica vs. Otimização Matemática:** A nossa solução manual foi baseada na **lógica humana**, onde desenhámos portas (NOR, AND, OR). A solução do *backpropagation* é puramente **matemática**, focada apenas em minimizar a função de custo (MSE). Os neurónios 'A' e 'B' na rede treinada provavelmente não representam funções lógicas "limpas" como NOR ou AND, mas sim alguma combinação matemática não-intuitiva que, quando combinada por 'C', resulta na saída XNOR correta.

5.2 Evolução do Erro de Treino (MSE)

O gráfico seguinte demonstra a evolução do Erro Quadrático Médio (MSE) ao longo de 10.000 épocas de treino, mostrando como o erro converge para perto de zero.

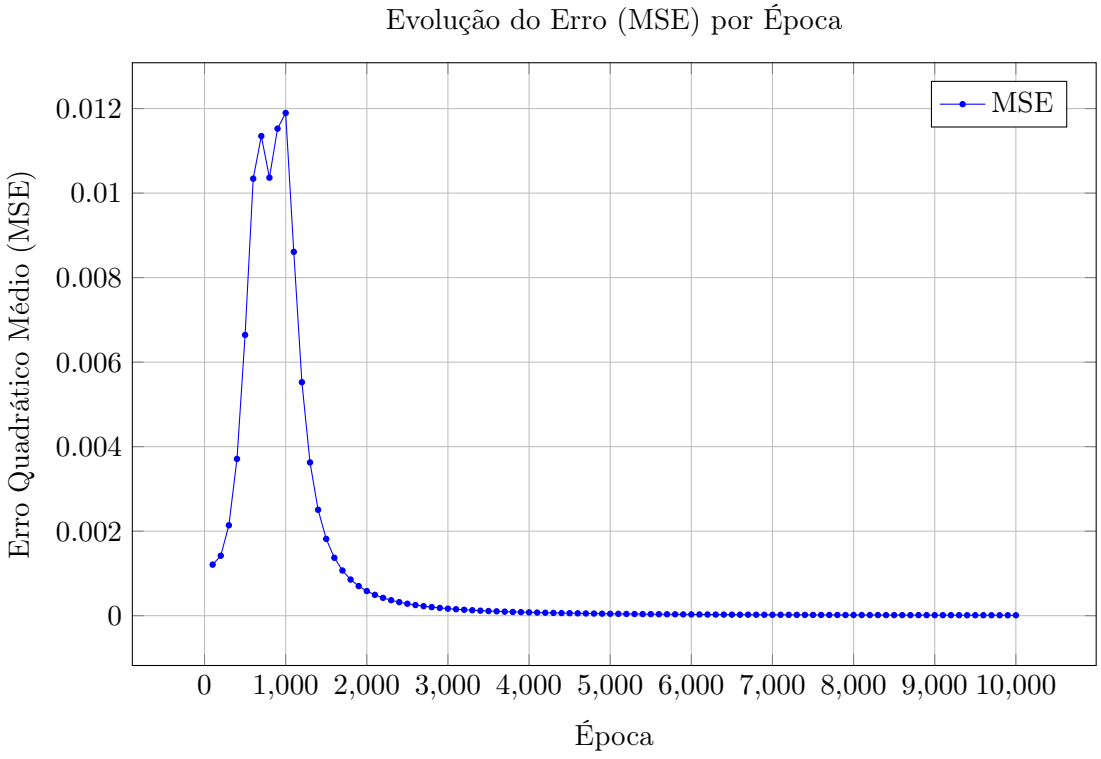


Figure 1: Gráfico da evolução do MSE ao longo das épocas de treino.