Inteligência Artificial

Relatório do Problema 3
Perceptrons
2025/26

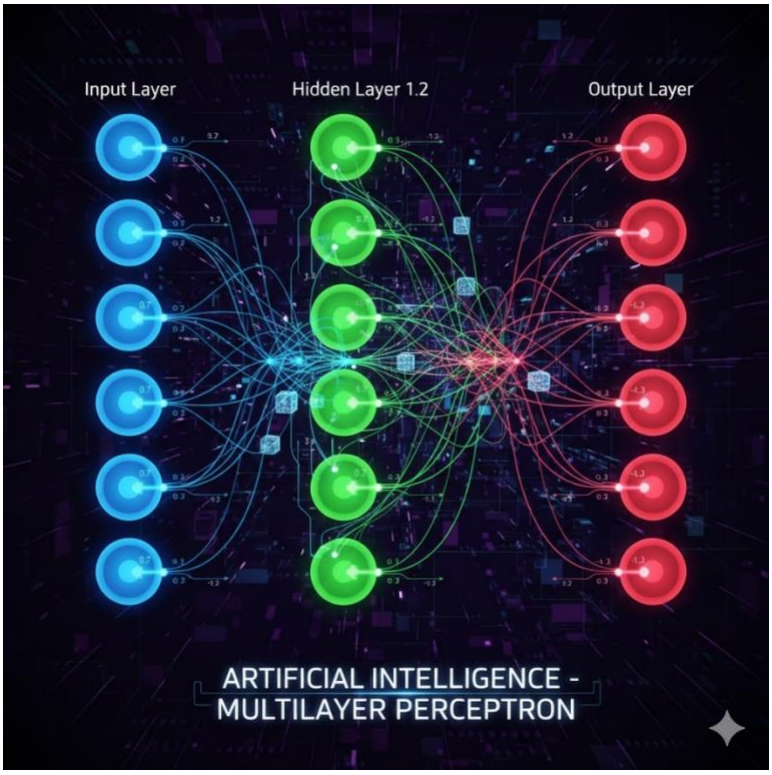


Imagem gerada por AI

Nome	Número
André Martins	35463
António Matoso	84100
Tomás Machado	84014

Índice

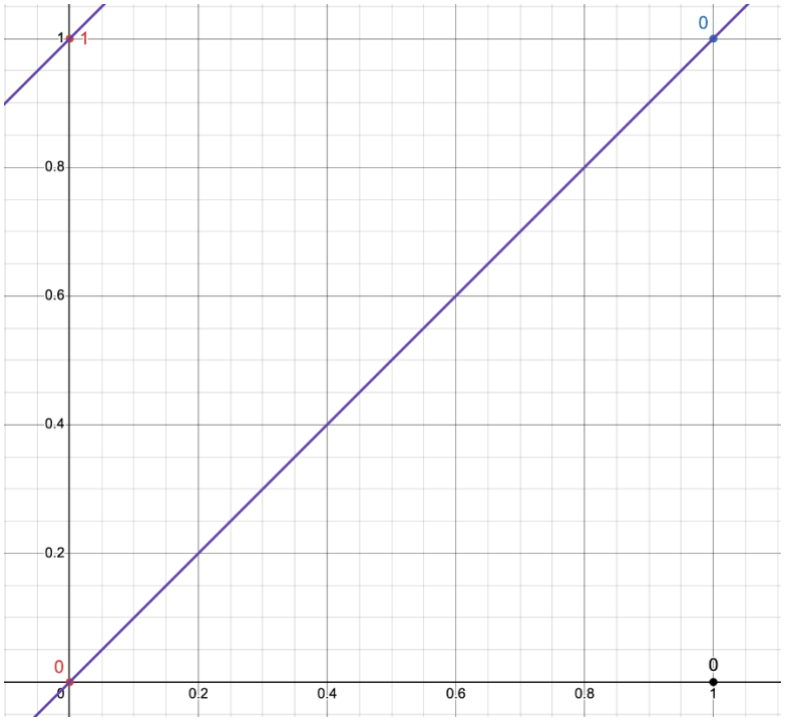
Introdução

O objetivo deste problema é estudar o funcionamento de neurónios artificiais, perceptrões multicamada (MLP), aplicados a funções lógicas binárias, nomeadamente a função NOT XOR (XNOR). Pretende-se propor conjuntos de dados, calcular pesos manualmente, configurar a rede e posteriormente treiná-la utilizando o algoritmo de *back propagation*.

Tarefa 1

Objetivo: Proponha um conjunto de dados $\{(x1^{(i)}, x2^{(i)}, y^{(i)})|\ i=1, \ldots, 4\}$ onde x1 e x2 são variáveis binárias independentes e y é uma variável binária dependente linearmente separável.

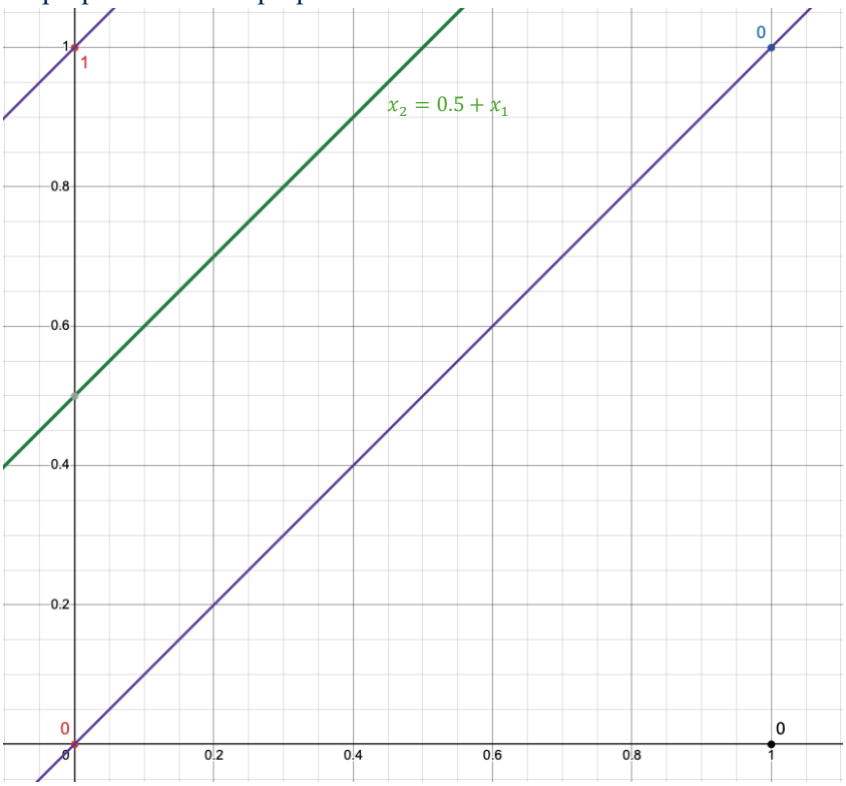
i	$x_1{}^{(i)}$	$x_2{}^{(i)}$	$y^{(i)} = \overline{x_1}x_2$
1	0	0	0
2	0	1	1
3	1	0	0
4	1	1	0



O conjunto de dados é linearmente separável, já que a fronteira de decisão não é única, qualquer linha situada entre as duas linhas assinaladas no gráfico consegue separar perfeitamente os 0's e o 1.

Tarefa 2

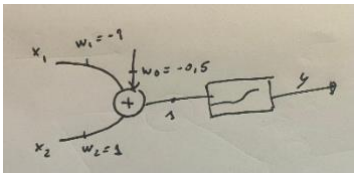
Objetivo: Calcule, sem treino, os pesos de um único neurónio para obter um erro de treino zero no conjunto de dados proposto em 1. Explique resumidamente.



Representação gráfica da separação linear das duas classes

A reta $x_2 = 0.5 + x_1$ foi escolhida porque separa visualmente o ponto da classe 1 dos restantes pontos da classe 0.

Reescrevendo a fronteira na forma do neurónio, obtemos:



$$-0.5 - x_1 + x_2 = 0 \Rightarrow w_0 = -0.5,\ w_1 = -1,\ w_2 = 1$$

O termo 0.5 representa uma margem entre as classes, garantindo que a linha passa entre elas.
Embora existam várias retas possíveis, esta posição intermédia torna o modelo mais estável.
Os sinais dos pesos refletem a influência das variáveis: x_2 tem peso positivo (favorece a classe 1) enquanto $x_1$tem peso negativo (favorece a classe 0).

$$z = \begin{cases} 1, & se\ w_0 + w_1 x_1 + w_2 x_2 > 0 \\ 0, & caso\ contrário \end{cases} \Rightarrow \begin{cases} 1, & se - 0.5\ - x_1 + x_2 > 0 \\ 0, & caso\ contrário \end{cases}$$

i	$x_1{}^{(i)}$	$x_2{}^{(i)}$	$-0.5 - x_1{}^{(i)} + x_2{}^{(i)}$	$z^{(i)}$	$y^{(i)} = \overline{x_1} x_2$
1	0	0	-0.5-0+0=-0.5	0	0
2	0	1	-0.5-0+1=0.5	1	1
3	1	0	-0.5-1+0=-1.5	0	0
4	1	1	-0.5-1+1=-0.5	0	0

Como podemos ver os nossos pesos estão a estimar bem o valor, ou seja $z \equiv y$, já que a função z classifica corretamente todos os exemplos.

Tarefa 3

Objetivo: Implementar e testar o neurónio configurado na tarefa 2.

Foi implementado um perceptrão simples com três componentes principais:

1. **Classe *Perceptron***: Representa o neurónio com os pesos w_0, w_1 e w_2 calculados manualmente na Tarefa 2. O método *predict()* calcula a saída aplicando a função Step.

2. **Classe *Step***: Implementa a função de ativação binária com que retorna 1 se $z > 0$ ou 0 caso contrário.

3. **Classe *DataSet***: Estrutura para armazenar pares de entrada-saída (x, y) do conjunto de dados.

```
package src;

public class Perceptron {
   private double w0;
   private double w1;
   private double w2;

   public Perceptron(double w0, double w1, double w2) {
      this.w0 = w0;
      this.w1 = w1;
      this.w2 = w2;
   }

   public double predict(double[] inputs) {
      return Step.f(this.w0 + this.w1 * inputs[0] + this.w2 * inputs[1]);
   }
}
```

```java
package src;

public class Step {
   private Step() {
      // Prevent instantiation
   }

   public static double f(double z) {
      return (z > 0.0) ? 1.0 : 0.0;
   }
}
```

```java
package src;

public class DataSet {
   private double[] x;
   private double y;

   public DataSet(double[] x, double y) {
      if (x == null || x.length == 0) {
         throw new IllegalArgumentException(
               "Input array cannot be null or empty");
      }
      this.x = x;
      this.y = y;
   }

   public double[] getX() {
      return this.x;
   }

   public double getY() {
      return this.y;
   }
}
```

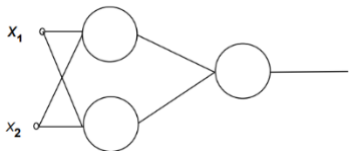
```java
package src;

public class Main {
   public static void main(String[] args) {
      Perceptron p = new Perceptron(-0.5, -1, 1);
      DataSet[] data = {
            new DataSet(new double[] { 0, 0 }, 0),
            new DataSet(new double[] { 0, 1 }, 1),
            new DataSet(new double[] { 1, 0 }, 0),
            new DataSet(new double[] { 1, 1 }, 0)
      };
      for (DataSet d : data) {
         double[] x = d.getX();
         System.out.printf(
               "Input:(%.0f, %.0f)->Predicted:%.0f,Expected:%.0f%n",
               x[0], x[1], p.predict(x), d.getY());
      }
   }
}
```

```
•→  Lab-4-raw git:(main) ✗ javac -d bin $(find src -name "*.java") && java -cp bin src.Main
Input:(0, 0)->Predicted:0,Expected:0
Input:(0, 1)->Predicted:1,Expected:1
Input:(1, 0)->Predicted:0,Expected:0
Input:(1, 1)->Predicted:0,Expected:0
```

Análise:

Os resultados mostram que o neurónio consegue classificar todos os exemplos corretamente, o que confirma que os pesos que calculámos manualmente estão corretos. A fronteira de decisão $x_2 = 0.5 + x_1$ separa perfeitamente as duas classes, levando a um erro de treino igual a zero. Isto demonstra que, quando os pesos são determinados de forma analítica, um único neurónio com função *Step* é suficiente para resolver problemas que são linearmente separáveis.

Tarefa 4

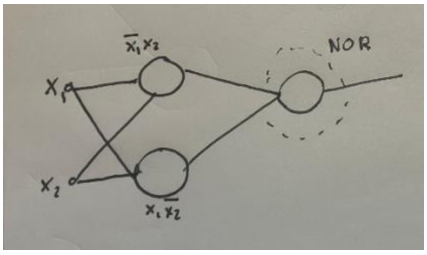


Objetivo: Configure, sem treino, o perceptron multicamadas acima para que funcione como uma função binária NOT XOR. Explique resumidamente.

i	${x_1}^{(i)}$	${x_2}^{(i)}$	$y^{(i)}(XOR) = \overline{x_1}x_2 + x_1\overline{x_2}$	$y^{(i)}(XNOR) = \overline{\overline{x_1}x_2 + x_1\overline{x_2}}$
1	0	0	0	1
2	0	1	1	0
3	1	0	1	0
4	1	1	0	1

A porta XOR não é linearmente separável, pois não existe uma única reta que separe os 0 dos 1 no plano (x_1, x_2). Como a XNOR é apenas a negação da XOR, também ela não é linearmente separável.

Primeira possível abordagem **(adotada neste relatório):**
Para resolvermos este problema, recorremos a dois neurónios na camada oculta para representar as duas partes da expressão da XOR ($a = \overline{x_1} \wedge x_2$ e $b = x_1 \wedge \overline{x_2}$).
O neurónio de saída combina estes dois sinais através de uma operação **NOR**, o que equivale a aplicar $\overline{a \vee b}$ e produz diretamente a função XNOR, como representado em baixo.



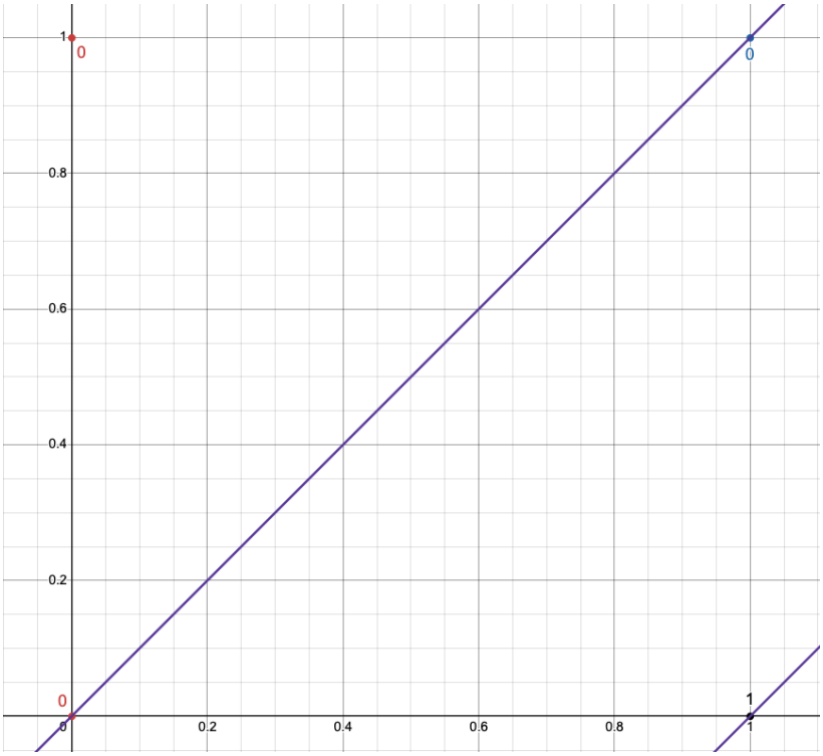
Uma abordagem alternativa consiste em construir diretamente a função XNOR usando dois neurónios AND na camada oculta e um neurónio OR na camada de saída.
Nesta configuração, o primeiro neurónio oculto calcula $x_1 \wedge x_2$e o segundo calcula $\overline{x_1} \wedge \overline{x_2}$.
O neurónio de saída realiza depois a operação OR entre estes dois valores, produzindo $(x_1 \wedge x_2) \vee (\overline{x_1} \wedge \overline{x_2})$, que corresponde exatamente à função XNOR.

i	${x_1}^{(i)}$	${x_2}^{(i)}$	$y^{(i)} = a = \overline{x_1}x_2$
1	0	0	0
2	0	1	1
3	1	0	0
4	1	1	0

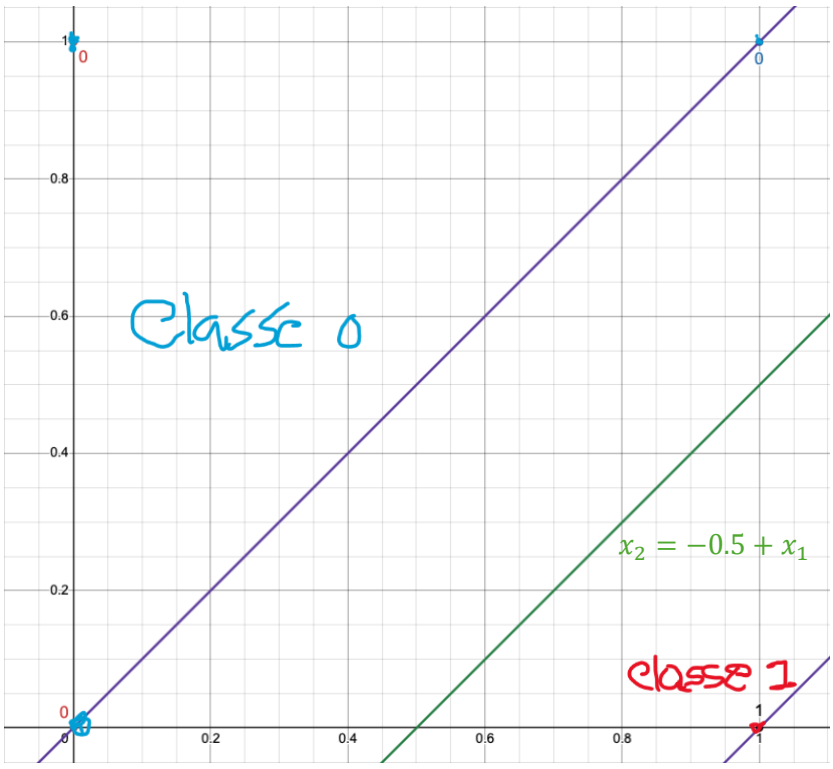
Como já tínhamos visto $y = \overline{x_1}x_2$ é linearmente separável.

i	${x_1}^{(i)}$	${x_2}^{(i)}$	$y^{(i)} = b = x_1\overline{x_2}$
1	0	0	0
2	0	1	0
3	1	0	1
4	1	1	0

Que tem como gráfico ($representação\ da\ função\ y = x_1\overline{x_2}$):



Podemos concluir que é linearmente separável, já que a fronteira de decisão não é única e qualquer reta situada entre as duas retas representadas no gráfico consegue separar perfeitamente os 0's do 1.



A reta $x_2 = -0.5 + x_1$ foi escolhida porque separa visualmente o ponto da classe 1 dos restantes pontos da classe 0.
Reescrevendo a fronteira na forma do neurónio, obtemos:

$$x_1 - x_2 - 0.5 = 0 \Rightarrow w_0 = -0.5,\ w_1 = 1,\ w_2 = -1$$

$$z_b = \begin{cases} 1, & se\ w_0 + w_1 x_1 + w_2 x_2 > 0 \\ 0, & caso\ contrário \end{cases} \Rightarrow \begin{cases} 1, & se\ -0.5 + x_1 - x_2 > 0 \\ 0, & caso\ contrário \end{cases}$$

i	${x_1}^{(i)}$	${x_2}^{(i)}$	$-0.5 + {x_1}^{(i)} - {x_2}^{(i)}$	${z_b}^{(i)}$	$y^{(i)} = x_1\overline{x_2}$
1	0	0	-0.5+0-0=-0.5	0	0
2	0	1	-0.5+0-1=-1.5	0	0
3	1	0	-0.5+1-0=0.5	1	1
4	1	1	-0.5+1-1=-0.5	0	0

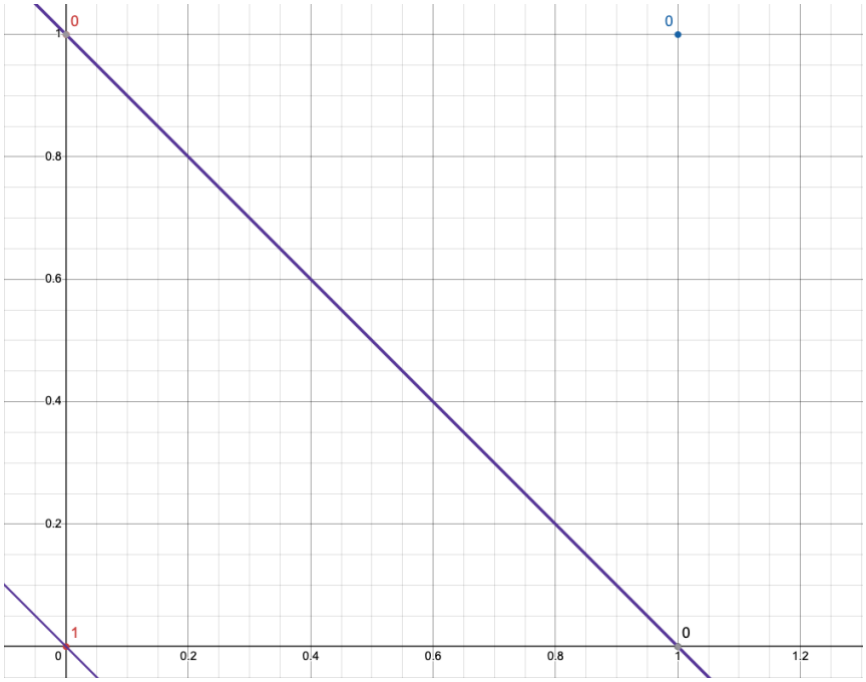
Podemos observar que os pesos escolhidos permitem que o neurónio estime corretamente a saída desejada, ou seja $z_b \equiv y$, já que a função z_b classifica corretamente todos os exemplos.

Partindo das duas funções parciais, ambas linearmente separáveis, que representam os casos em que apenas uma das entradas é igual a 1 ($\overline{x_1}x_2$ *e* $x_1\overline{x_2}$), combiná-las através de um OR permitiria obter a operação XOR, uma vez que esta só produz saída 1 quando as entradas são diferentes. No entanto, como o objetivo é implementar a função NOT XOR (XNOR), o neurónio de saída é configurado para realizar uma operação NOR, aplicando a negação ao resultado do OR dessas duas funções. Desta forma, a rede devolve o valor 1 apenas quando as duas entradas são iguais, correspondendo exatamente ao comportamento da XNOR.

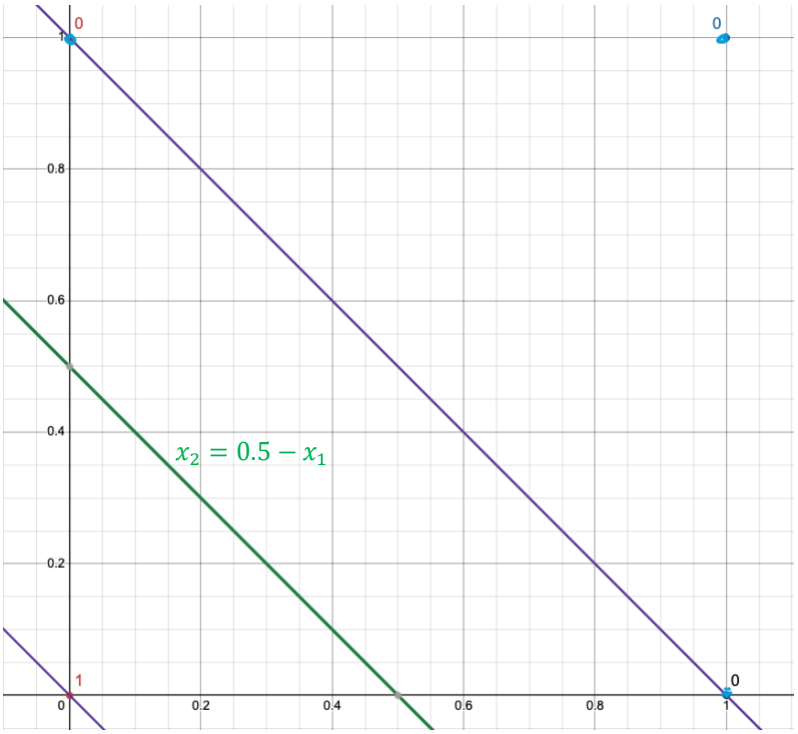
Então temos: $z_a = \begin{cases} 1, & se\ -0.5 - x_1 + x_2 > 0 \\ 0, & caso\ contrário \end{cases}$ *e* $z_b = \begin{cases} 1, & se\ -0.5 + x_1 - x_2 > 0 \\ 0, & caso\ contrário \end{cases}$

Começamos por mostrar que a operação NOR é linearmente separável:

i	${x_1}^{(i)}$	${x_2}^{(i)}$	$y^{(i)}(NOR) = \overline{{x_1}^{(i)} + {x_2}^{(i)}}$
1	0	0	1
2	0	1	0
3	1	0	0
4	1	1	0



Podemos concluir que é linearmente separável, já que a fronteira de decisão não é única e qualquer linha situada entre as duas linhas assinaladas no gráfico consegue separar perfeitamente os 0's e o 1.



A linha $x_2 = 0.5 - x_1$ foi escolhida porque separa visualmente o ponto da classe 1 dos restantes pontos da classe 0.
Reescrevendo a fronteira na forma do neurónio, obtemos:

$$-x_1 - x_2 + 0.5 = 0 \Rightarrow w_0 = 0.5,\ w_1 = -1,\ w_2 = -1$$

$$z = \begin{cases} 1, & se\ w_0 + w_1 x_1 + w_2 x_2 > 0 \\ 0, & caso\ contrário \end{cases} \Rightarrow \begin{cases} 1, & se\ 0.5 - x_1 - x_2 > 0 \\ 0, & caso\ contrário \end{cases}$$

i	${x_1}^{(i)}$	${x_2}^{(i)}$	$0.5 - {x_1}^{(i)} - {x_2}^{(i)}$	$z^{(i)}$	$y^{(i)} = \overline{{x_1}^{(i)} + {x_2}^{(i)}}$
1	0	0	0.5-0-0=0.5	1	1
2	0	1	0.5-0-1=-0.5	0	0
3	1	0	0.5-1-0=-0.5	0	0
4	1	1	0.5-1-1=-1.5	0	0

Podemos observar que os pesos escolhidos permitem que o neurónio estime corretamente a saída desejada, ou seja $z \equiv y$, já que a função z implementada classifica corretamente todos os exemplos.

Como cada neurónio da camada escondida representa uma das partes da expressão da XOR, isto é:

$$z_a = \overline{x_1} x_2 \ e \ z_b = x_1 \overline{x_2}$$

as suas saídas tornam-se as novas entradas da camada seguinte.
Assim, na camada de saída, x_1e $x_2$passam a ser substituídos por z_ae z_b, respetivamente, pois o neurónio final opera sobre os resultados produzidos pelos dois primeiros da camada escondida.

$$Então,\ z_f = \begin{cases} 1, & se\ w_0 + w_1 x_1 + w_2 x_2 > 0 \\ 0, & caso\ contrário \end{cases} \Rightarrow \begin{cases} 1, & se\ 0.5 - z_a - z_b > 0 \\ 0, & caso\ contrário \end{cases}$$

i	${x_1}^{(i)}$	${x_2}^{(i)}$	${z_a}^{(i)}$	${z_b}^{(i)}$	$-0.5 - {z_a}^{(i)} - {z_b}^{(i)}$	${z_f}^{(i)}$	$y^{(i)}(NXOR) = \overline{\overline{x_1} x_2 + x_1 \overline{x_2}}$
1	0	0	0	0	0.5-0-0=0.5	1	1
2	0	1	1	0	0.5-1-0=-0.5	0	0
3	1	0	0	1	0.5-0-1=-0.5	0	0
4	1	1	0	0	0.5-0-0=0.5	1	1

Podemos observar que os pesos escolhidos permitem que o neurónio estime corretamente a saída desejada, ou seja $z_f \equiv y$, já que a função z_f implementada classifica corretamente todos os exemplos.

Tarefa 5

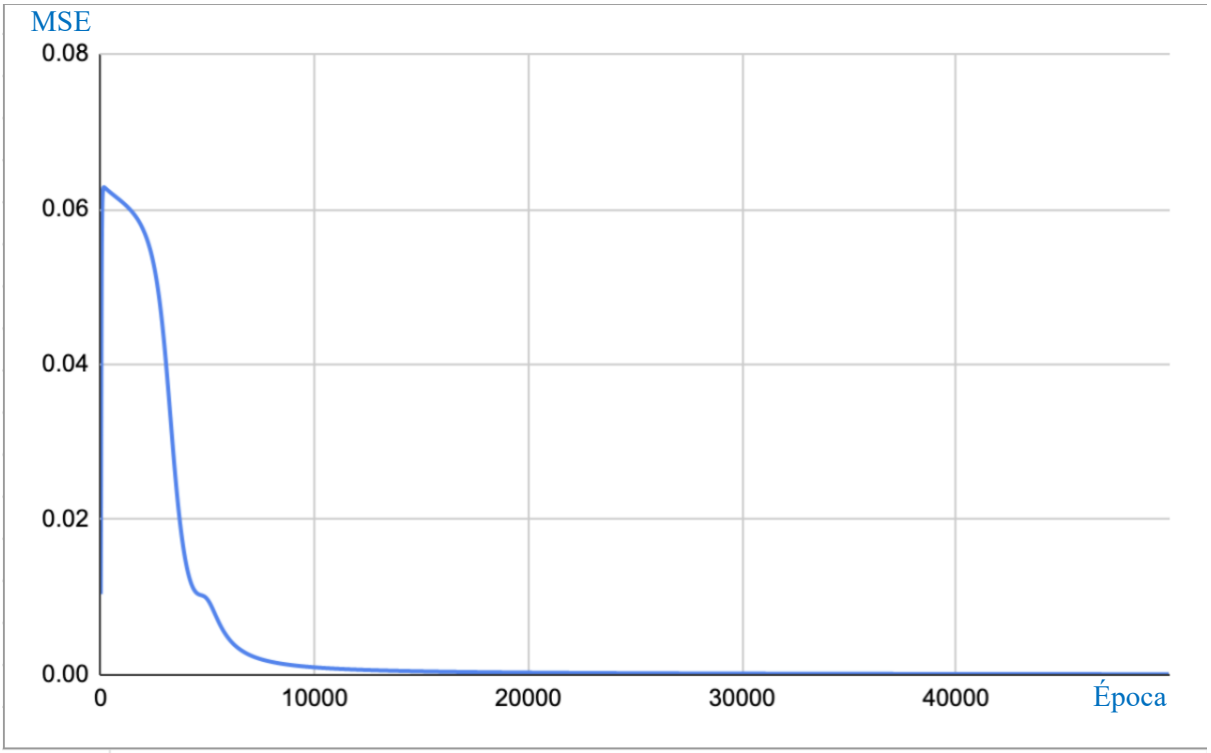
Objetivo: Implemente o treino da rede com o algoritmo de *back propagation* utilizando o conjunto de dados da função NOT XOR. Trace o MSE em cada iteração do processo de treino. Quão semelhantes são os pesos obtidos aos calculados em 4.? Explique resumidamente.

Configuração

- Learning rate: 0.1
- Épocas: 50000
- Função de ativação: *Sigmoid* em todas as camadas

- Topologia: 2 → 2 → 1
- Pesos iniciais: aleatórios

Gráfico do MSE



Análise do gráfico

O MSE começa por subir um pouco e depois entra numa descida contínua, o que é exatamente o que costuma acontecer neste tipo de treino. Parte de cerca de 0.010 e, nas primeiras 150 épocas, ainda aumenta até perto de 0.063. Esse pico basicamente mostra que o gradiente descendente anda a "procurar o caminho" e a afastar-se de uma solução que não seria a melhor. A partir daí, o erro começa finalmente a baixar de forma constante e vai diminuindo ao longo das 50000 épocas até ficar praticamente a zero (por volta de 0.0001). Esta queda lenta mas estável faz sentido tendo em conta a taxa de aprendizagem de 0.1, já que esta é segura e evita oscilações, mas também torna o processo mais demorado.

No fim, o erro tão baixo mostra que a rede aprendeu mesmo a XNOR. Como o conjunto de dados são só os quatro casos da tabela de verdade, não dá para tirar conclusões sobre *overfitting,* já que não é possível haverem dados de validação.

Pesos Obtidos Após o Treino

```
Weights:
Layer 1:
6.469146765638291 4.5825119875331275
6.4746355552447445 4.583843521868441
Layer 2:
-9.583892870982845
10.283769003739407
Biases:
Layer 1:
-2.875291980352411 -7.033502501509305
Layer 2:
4.432402793676509
```

Camada 1 (input → hidden)

$w_0 \equiv b$

$$W^{(1)} = \begin{bmatrix} 6.4691 & 4.5825 \\ 6.4746 & 4.5838 \end{bmatrix}$$

$$b^{(1)} = [-2.8753 \quad -7.0335]$$

Camada 2 (hidden → output)

$$W^{(2)} = \begin{bmatrix} -9.5839 \\ 10.2838 \end{bmatrix}$$

$$b^{(2)} = [4.4324]$$

Pesos calculados manualmente (Tarefa 4)

Camada escondida (input → hidden)

- Neurónio $z_a = \overline{x_1} x_2$:

$$w_0 = -0.5, w_1 = -1, w_2 = 1$$

- Neurónio $z_b = x_1 \overline{x_2}$:

$$w_0 = -0.5, w_1 = 1, w_2 = -1$$

Camada de saída (hidden → output)

- Neurónio final (NOR de z_a e z_b para obter XNOR):

$$w_0 = 0.5, w_1 = -1, w_2 = -1$$

Pesos obtidos pelo treino (Tarefa 5)

Camada escondida (input → hidden)

- Neurónio z_1:

$$w_0 = -2.8753, w_1 = 6.4691, w_2 = 4.5825$$

- Neurónio z_2:

$$w_0 = -7.0335, w_1 = 6.4746, w_2 = 4.5838$$

Camada de saída (hidden → output)

- Neurónio final (OR de z_1 e z_2 para obter XNOR):

$$w_0 = 4.4324, w_1 = -9.5839, w_2 = 10.2838$$

Tendo em conta a função sigmoide:

$$g(z) = \frac{1}{1 + e^{-z}}$$

Comparação entre pesos manuais e treinados

Os pesos obtidos pelo treino não coincidem numericamente com os que tínhamos calculado manualmente na Tarefa 4 eles são bem maiores e até têm sinais diferentes. Isso **não significa que houve algum erro**, em redes com função sigmoide, os pesos tendem a crescer em magnitude para aproximar o comportamento da função *step*.

$$z_1 = -2.8753 + 6.4691x_1 + 4.5825x_2$$

i	${x_1}^{(i)}$	${x_2}^{(i)}$	${z_1}^{(i)}$	$g({z_1}^{(i)})$	$y^{(i)}(OR)$
1	0	0	$-2.8753 + 0 + 0 = -2.8753$	$\frac{1}{1 + e^{2.8753}} = 0.053388 < 0.5$	0
2	0	1	$-2.8753 + 0 + 4.5825 = 1.7072$	$\frac{1}{1 + e^{-1.7072}} = 0.84647 > 0.5$	1
3	1	0	$-2.8753 + 6.4691 + 0 = 3.5938$	$\frac{1}{1 + e^{-3.5938}} = 0.97324 > 0.5$	1
4	1	1	$-2.8753 + 6.4691 + 4.5825 = 8.1763$	$\frac{1}{1 + e^{-8.1763}} = 0.99972 > 0.5$	1

$$z_2 = -7.0335 + 6.4746x_1 + 4.5838x_2$$

i	${x_1}^{(i)}$	${x_2}^{(i)}$	${z_2}^{(i)}$	$g({z_2}^{(i)})$	$y^{(i)}(AND)$
1	0	0	$-7.0335 + 0 + 0 = -7.0335$	$\frac{1}{1+e^{7.0335}} = 0.00088106 < 0.5$	0
2	0	1	$-7.0335 + 0 + 4.5838 = -2.4497$	$\frac{1}{1+e^{2.4497}} = 0.079460 < 0.5$	0
3	1	0	$-7.0335 + 6.4746 + 0 = -0.55890$	$\frac{1}{1+e^{0.55890}} = 0.36380 < 0.5$	0
4	1	1	$-7.0335 + 6.4746 + 4.5838 = 4.0249$	$\frac{1}{1+e^{-4.0249}} = 0.98245 > 0.5$	1

$$z = 4.4324 - 9.5839\ z_1 + 10.2838\ z_2$$

i	${x_1}^{(i)}$	${x_2}^{(i)}$	${z_1}^{(i)}$	${z_2}^{(i)}$	$z^{(i)}$	$g(z^{(i)})$	$y^{(i)}(NXOR)$
1	0	0	0	0	$4.4324 + 0 + 0 = 4.4324$	$\frac{1}{1+e^{-4.4324}} = 0.98825 > 0.5$	1
2	0	1	1	0	$4.4324 - 9.5839 + 0 = -5.1515$	$\frac{1}{1+e^{5.1515}} = 0.0057574 < 0.5$	0
3	1	0	1	0	$4.4324 - 9.5839 + 0 = -5.1515$	$\frac{1}{1+e^{5.1515}} = 0.0057574 < 0.5$	0
4	1	1	1	1	$4.4324 - 9.5839 + 10.2838 = 5.1323$	$\frac{1}{1+e^{-5.1323}} = 0.99413 > 0.5$	1

Mesmo que os pesos treinados não coincidam com os que calculámos manualmente e até pareçam "estranhos" por serem muito maiores, a rede acabou por aprender exatamente o comportamento da XNOR. E esta não é obrigada a seguir a mesma lógica que usamos na solução à mão, e neste caso encontrou o seu próprio caminho. No fim, o resultado é o mesmo, a rede reproduz corretamente a XNOR, mostrando que há várias estratégias diferentes de chegar à mesma função.

Conclusão

Neste relatório vimos como perceptrões e redes multicamada conseguem aprender funções lógicas. Começámos com exemplos simples, resolvidos à mão, e depois passámos para a XNOR, que é mais complexa, configurámos a rede manualmente e também treinámo-la. Mesmo que os pesos do treino não coincidam com os calculados à mão, a rede aprendeu perfeitamente a XNOR. O MSE mostra como o erro primeiro explora diferentes soluções e depois desce de forma estável até quase zero. Isto mostra que a rede encontra várias maneiras de chegar ao mesmo resultado.
No fim, o mais importante é que, independentemente do caminho, a rede reproduziu corretamente o comportamento esperado. Aprendemos que redes neurais são flexíveis e capazes de descobrir soluções eficazes por conta própria.