**Uma imagem com texto

Descrição gerada automaticamente**

Elementos de Programação

Trabalho nº 2

Jodionísio Muachifi «97147»

Miguel Simões «118200»

Gustavo Reggio «118485»

**Grupo nº 1**

Mestrado em Engenharia de Computadores e Telemática

Mestrado em Engenharia de Automação Industrial

Professor: Armando J. Pinho

27 de novembro de 2023

[Link do repositório do Projeto 2](https://github.com/jmgitcloudua/ep_labwork2)

Conteúdo

[Abreviaturas ii](#_Toc151966504)

[1 Introdução 1](#_Toc151966505)

[2 Parte I 2](#_Toc151966506)

[2.1 Exercício nº 1 (a) 2](#_Toc151966507)

[2.2 Exercício nº 1 (b) 3](#_Toc151966508)

[2.3 Exercício nº 1 (c) 4](#_Toc151966509)

[2.4 Exercício nº 1 (d) 6](#_Toc151966510)

[2.5 Exercício nº 1 (e) 9](#_Toc151966511)

[3 Testes de redes neurais e análise de resultados 10](#_Toc151966512)

[3.1 Testes de redes neurais 10](#_Toc151966513)

[3.2 Análise de resultados 13](#_Toc151966514)

[4 Testes unitários e programação defensiva 17](#_Toc151966515)

[4.1 Testes unitários 17](#_Toc151966516)

[4.2 Programação defensiva 17](#_Toc151966517)

[5 Contribuição dos autores 18](#_Toc151966518)

[6 Conclusão 19](#_Toc151966519)

Figuras

[Figura 1 - Função de ativação (Sigmóide) 2](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793199)

[Figura 2 - Função de ativação (ficheiro **nn\_base.c**) 3](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793200)

[Figura 3 - Unidade bias adicionada na estrutura de dados **nn** 4](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793201)

[Figura 4 - Representação do grafo da Rede Neural com I=2, H=2, O=1 5](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793202)

[Figura 5 - Representação gráfica da função sigmóide e sua derivativa 6](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793203)

[Figura 6 - Comparação do XOR nn com e sem seed (n = 1000, h = 12) 10](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793204)

[Figura 7 - Comparação do XOR nn com e sem seed (n = 10000, h = 12) 11](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793205)

[Figura 8 - Comparação do XOR nn com e sem seed (n = 100000, h = 12) 11](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793206)

[Figura 9 - Comparação do XOR nn com e sem seed (n = 1000000, h = 12) 12](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793207)

[Figura 10 - Gráfico do MSE, Tempo em Relação a n, sem seed (n = 1000, h = 12, lr=0.1) 13](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793208)

[Figura 11 - Gráfico do MSE, Tempo em Relação a n, com seed (n = 1000, h = 12, lr=0.1, seed=123) 14](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793209)

[Figura 12 - Gráfico do MSE, Tempo em Relação a n, sem seed (n = 10000, h = 12, lr=0.5) 14](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793210)

[Figura 13 - Gráfico do MSE, Tempo em Relação a n, com seed (n = 10000, h = 12, lr=0.5, seed=123) 14](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793211)

[Figura 14 - Gráfico do MSE, Tempo em Relação a n, sem seed (n = 100000, h = 12, lr=0.5) 15](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793212)

[Figura 15 - Gráfico do MSE, Tempo em Relação a n, com seed (n = 100000, h = 12, lr=0.5, seed=123) 15](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793213)

[Figura 16 - Gráfico do MSE, Tempo em Relação a n, sem seed (n = 1000000, h = 12, lr=0.5) 15](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793214)

[Figura 17 - Gráfico do MSE, Tempo em Relação a n, com seed (n = 1000000, h = 12, lr=0.5, seed=123) 16](https://uapt33090-my.sharepoint.com/personal/jodionisiomuachifi_ua_pt/Documents/ua-mect/5ºAno-MECT/EP/projects/work2/ep-g1-report2.docx#_Toc151793215)

# Abreviaturas

MECT – Mestrado em Engenharia de Computadores e Telemática

MEAI – Mestrado em Engenharia de Automação Industrial

PD – Programação defensiva

TU – Teste Unitário

MSE – Mean Squared Error

NN – Neural Network

BP – Backpropagation

Capítulo 1

# 1 Introdução

Redes neurais representam modelos computacionais poderosos inspirados na complexa estrutura e funcionalidade do cérebro humano. Nos últimos anos, essa abordagem ganhou proeminência pela sua eficácia na resolução de problemas complexos numa variedade de domínios, desde reconhecimento de imagem e fala até processamento de linguagem natural e tomada de decisões. Este relatório concentra-se na implementação e avaliação de uma rede neural simples projetada para abordar a clássica função booleana ***XOR***.

Para garantir modularidade e flexibilidade, o código foi desenvolvido utilizando a linguagem de programação C, proporcionando uma implementação clara e concisa da rede neural. O processo de treino inclui estratégias de otimização, como a inicialização de pesos aleatórios, salvando o estado do modelo e introduzindo uma semente para garantir a reprodutibilidade dos resultados.

A arquitetura da rede neural é construída com uma abordagem modular, encapsulada nos arquivos ***nn\_base.h*** e ***nn\_base.c***. O modelo consiste em camadas interconectadas de unidades, cada uma incorporando uma função de ativação não linear. O treino da rede envolve a retropropagação (***backpropagation***), uma técnica fundamental que ajusta os pesos e o ***bias*** para minimizar a diferença entre as saídas previstas e os alvos (***targets***) desejados.

A implementação prática da rede neural é realizada no arquivo ***train\_nn.c***, que funciona como uma arena de treino e teste para a função ***XOR***. Os dados de treino, fornecidos no arquivo ***xor.train***, orientam a rede na aproximação da função booleana ***XOR***. O desempenho da rede é meticulosamente avaliado por meio de iterações, com o erro quadrático médio (***MSE***) atuando como métrica central de precisão.

Para assegurar robustez e integridade, foram incorporadas validações extensivas, inclusive por meio de testes unitários, para verificar cada parte do código individualmente. O código-fonte desenvolvido, com instruções detalhadas sobre a sua execução, encontra-se disponível em nosso repositório no GitHub, proporcionando um fácil acesso para utilização e referência. Os gráficos presentes neste relatório, os fizemos com a linguagem de programação Python e Matlab. O código do projeto também pode ser testado na versão Python que desenvolvemos.

Capítulo 2

# 2 Parte I

Os exercícios desenvolvidos nesta parte tinham como principal propósito aplicar e integrar a matéria que estudamos nas aulas teóricas, aplicando conceitos fundamentais sobre redes neurais e cálculo de retropropagação (***backpropagation***).

## 2.1 Exercício nº 1 (a)

Esta parte do exercício é útil para compreender o contexto das redes neurais, sendo as funções de ativação um elemento crucial na introdução de não linearidade ao modelo. A função de ativação adotada neste projeto é a sigmóide, representada por uma função comumente utilizada que comprime os valores de entrada para um intervalo entre 0 e 1, como ilustrado na Figura 1

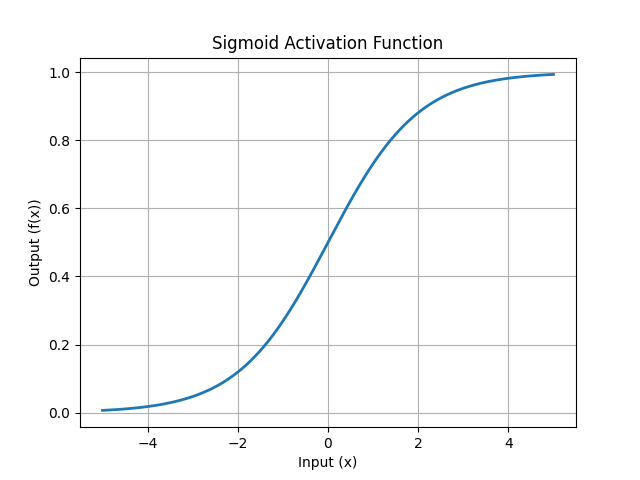


Figura 1 - Função de ativação (Sigmóide)

**Propriedades da função sigmóide**

**Não-linearidade**: A função sigmóide introduz não-linearidade à rede neural, permitindo que ela aprenda e represente relações complexas nos dados.

**Derivada**: A derivada da função sigmóide, , é utilizada no algoritmo de retropropagação durante a fase de treino. Ela ajuda a ajustar os pesos da rede para minimizar o erro.

**Faixa de saída**: A saída da função sigmóide é confinada à faixa (0, 1), o que é vantajoso em cenários em que a rede precisa produzir probabilidades.

Além disso, acrescentamos a função referida anteriormente no ficheiro de desenvolvimento dos protótipos (ficheiro ***nn\_base.c***), conforme mostra a Figura 2.

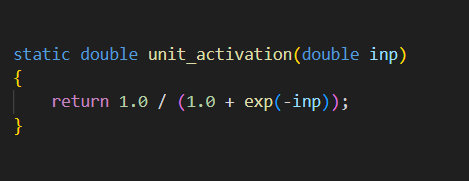


Figura 2 - Função de ativação (ficheiro **nn\_base.c**)

## 2.2 Exercício nº 1 (b)

O objetivo desta parte do exercício é acrescentar uma unidade á estrutura de dados ***nn*** (no ficheiro ***nn\_base.h)*** chamada ***bias***, que é um parâmetro essencial que permite que as unidades introduzam um offset à soma ponderada de entradas, proporcionando flexibilidade e auxiliando no processo de aprendizagem. A saída () de uma unidade (*j*) em camada (*l*) é atualizada do seguinte modo:

representa o peso que conecta a unidade *i* na camada *l - 1* à unidade *j* na camada *l*.

é a saída da unidade *i* na camada anterior (*l - 1*).

é o ***bias*** associado à unidade *j* na camada *l*.

A Figura 3 mostra a unidade acrescentada na estrutura de dados acima mencionada, i.e., adicionamos o ***bias*** nas duas camadas: ***hidden layer*** e ***output layer.***

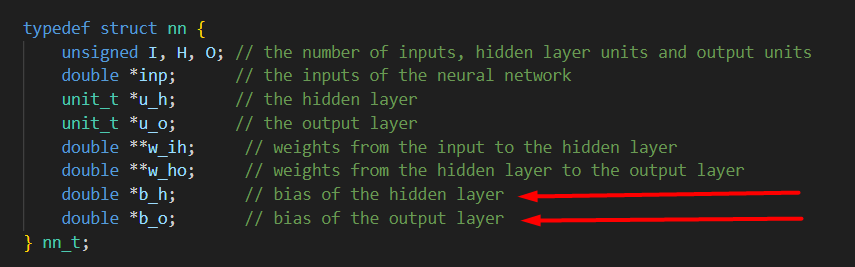


Figura 3 - Unidade bias adicionada na estrutura de dados **nn**

## 2.3 Exercício nº 1 (c)

O objetivo desta parte do exercício é implementar um código de teste para verificar se a rede neural está a implementar corretamente o que é esperado. Para tal definiu-se uma pequena rede neural com duas entradas, ambas com valor igual a um, duas *hidden layers*, e um output.

Nos cálculos abaixo, podemos ver como é efetuado o cálculo do ***Forwardpropagation***:

Para verificar o resultado calculado manualmente, criamos um programa denominado ***verify\_test\_design.c*** de modo a confirmar o resultado acima mencionado.

Para correr este programa, é necessário seguir as instruções que se segue e, assegurar que está no diretório partI:

*user@host: partI$* ***make***

*user@host: partI$* ***./verify\_test\_design***

Como se pode constatar, o valor final do resultado das contas manuais e o valor obtido pelo programa após a sua execução coincidem, demonstrando assim que a nossa ***forwardpropagtion*** produz os resultados esperados.

Com o objetivo de esclarecer e exemplificar a rede neural em questão, foi criada uma representação gráfica (ou seja, um grafo) utilizando a ferramenta ***draw.io*** para auxiliar na visualização.

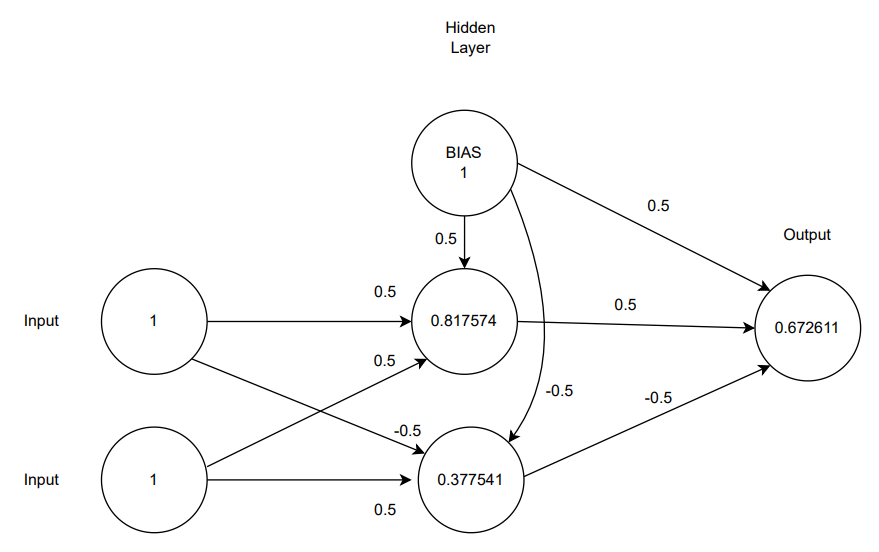


Figura 4 - Representação do grafo da Rede Neural com I=2, H=2, O=1

## 2.4 Exercício nº 1 (d)

O objetivo desta parte do exercício é aprender e implementar uma retropropagação. Esta é primordial para a aprendizagem da rede neural, e, como tal, não se aborda a retropropagação sem mencionar o cálculo de erro (***MSE***).

Basicamente, será construído um código para que, ao fim dos cálculos, haja uma validação da rede; esta será o cálculo de erro. A rede deve ser capaz de analisar o valor de erro e alterar os pesos a fim de minimizar esse valor de erro.

O termo ***“backpropagation”***deriva desta validação de erro e conforme o valor (entre 0 e 1) deve alterar os pesos consoante o rácio de aprendizagem.

**Caraterísticas essenciais do algoritmo *“Backpropagation”***

* **Função de ativação:** a escolha da função de ativação afeta o processo de treino. Funções de ativação comuns incluem o sigmóide, tangente hiperbólica (Tanh) e unidade linear retificada (Relu). Para o nosso caso, utilizamos a ***função sigmóide derivativa*** que mencionamos anteriormente na ***secção 2.1*** e ***2.3***, tal como ilustra o gráfico da Figura 5.

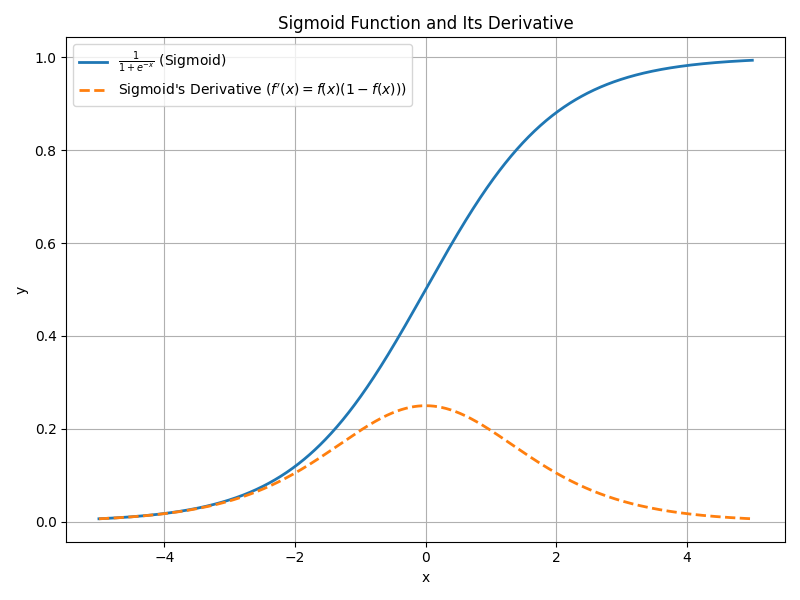
****

Figura 5 - Representação gráfica da função sigmóide e sua derivativa

* **Learning Rate (Taxa de Aprendizagem):** a taxa de aprendizagem controla o tamanho das actualizações de peso e de polarização. Uma taxa de aprendizagem demasiado pequena pode levar a uma convergência lenta, enquanto uma taxa de aprendizagem demasiado grande pode causar oscilações ou divergências.
* **Erro médio quadrático (MSE):** o MSE serve como uma métrica crucial para avaliar o desempenho da rede durante o treino. Orienta o processo de otimização ao quantificar a diferença média ao quadrado entre as saídas previstas e reais, refletindo a precisão geral do modelo.
* **Chain Rule (Regra da cadeia):** a retro propagação utiliza a regra da cadeia de cálculo para calcular gradientes de forma eficiente. Divide o erro global em contribuições de pesos e ***biases*** individuais. Para simplificar, vamos considerar uma rede neural simples com uma camada oculta apenas para ilustrar esta regra:

**Arquitetura da rede neural:**

* **Input Layer:** x1, x2, x3
* **Hidden Layer:** h1, h2​ com pesos e **bias**
* **Output Layer:** y com pesos ​ e bias b

**Forward Pass:**

1. **Input Layer to Hidden Layer**
2. **Hidden Layer to Output Layer:**

**Cálculo do Erro:**

**Derivada do erro em relação a y:**

**Derivada de** y **em relação a** :

**Derivada de** y **em relação a :**

**Derivada de em relação a :**

**Regra de atualização para** (gradiente decrescente) é:

, onde é a taxa de aprendizagem.

Para prosseguir com a implementação do código do algoritmo supracitado, reaproveitamos o código disponibilizado pelo Professor na plataforma e-learning, por este estar bem organizado e robusto. Portanto, criamos primeiramente no ficheiro ***nn\_base.c*** a função ***sigmoid\_derivative(double x)*** e só por conseguinte a função ***backpropagation(nn\_t \*nn, double \*targets, double learning\_rate)*** da qual realiza todo cálculo já explicado anteriormente.

Além disso, ainda no mesmo ficheiro, adicionamos uma *seed* (semente) na função ***create\_nn(unsigned I, unsigned H, unsigned O, unsigned seed)*** de modo a inicializar os pesos de forma aleatória com recurso a ***RAND\_MAX*** na primeira vez que o programa é executado. Também incluímos a função ***free\_nn(nn\_t \*nn),*** que permite libertar a memória após a sua utilização, e ***load\_input\_vector\_from\_array(nn\_t \*nn, double \*input\_array)*** que permite carregar os dados de entrada.

Em resumo, a retropropagação é um algoritmo fundamental para o treino de redes neurais, permitindo-lhes aprender com os dados e fazer previsões precisas. Sua natureza iterativa, uso eficiente de cálculo e adaptabilidade a várias arquiteturas de rede contribuem para seu uso generalizado no campo da aprendizagem automática. Entender o algoritmo de retropropagação é crucial para os profissionais que trabalham com redes neurais e forma a base para técnicas de otimização mais avançadas em ***deep learning***.

## 2.5 Exercício nº 1 (e)

O objetivo desta parte do exercício é treinar a rede neural utilizando a função XOR, que serve como um valioso caso de teste para redes neurais, destacando sua capacidade de aprender e representar relações não-lineares. Este relatório demonstra a importância das funções de ativação não lineares e das camadas ocultas para aumentar o poder de expressão da rede neural, utilizando a função XOR, cujos valores lógicos estão ilustrados na Tabela 1.

|  |  |  |
| --- | --- | --- |
| **Input** | | **Output** |
| A | B | A XOR B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Tabela 1 - Valores lógicos da função XOR (input, output)

Para obter bons resultados no treino de uma rede neural, é crucial fazer escolhas adequadas quanto à arquitetura da rede, aos (hiper)parâmetros e ao conjunto de dados de treino. No nosso caso, os parâmetros relevantes para alcançar resultados satisfatórios são: taxa de aprendizagem, número de iterações, número de camadas ocultas e semente (seed). Todos estes parâmetros e as demais características de inicialização da rede neural estão definidas no ficheiro ***train\_nn.c***.

Para correr o exercício deste programa, é necessário seguir as instruções que se segue e, assegurar que está no diretório partI:

*user@host: partI$* ***make***

*user@host: partI$* ***./train\_nn train -h 12 -n 10000 -lr 0.5 xor.train***

Onde:

* ***h*** representa o número de camadas ocultas;
* ***n*** representa o número de iterações e;
* ***lr*** representa o learning rate.

No capítulo seguinte, apresentaremos diversos testes, variando os parâmetros mencionados acima, e discutiremos os resultados obtidos.

Capítulo 3

# 3 Testes de redes neurais e análise de resultados

## 3.1 Testes de redes neurais

O foco desta secção é a realização de testes completos ou mais detalhados de redes neurais para avaliar seu desempenho e capacidade de propagação utilizando a função XOR. O teste de redes neurais envolve a exposição dos modelos treinados a dados novos para avaliar a capacidade de fazer previsões e guardar o estado anterior do treino.

* **Teste nº 1**

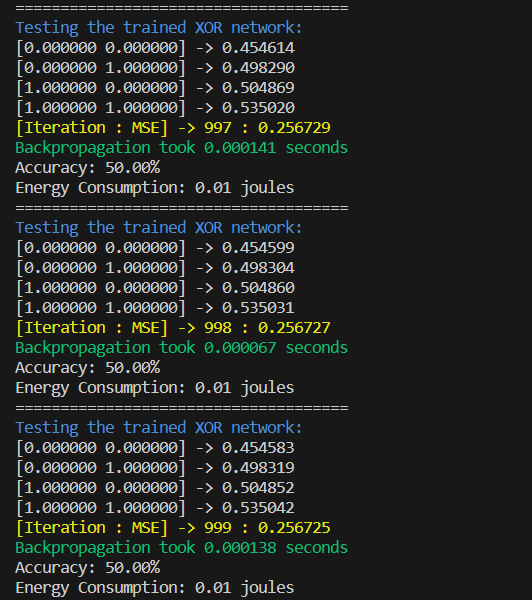
*user@host: partI$* ***make***

*user@host: partI$* ***./train\_nn train -h 12 -n 1000 -lr 0.1 xor.train***

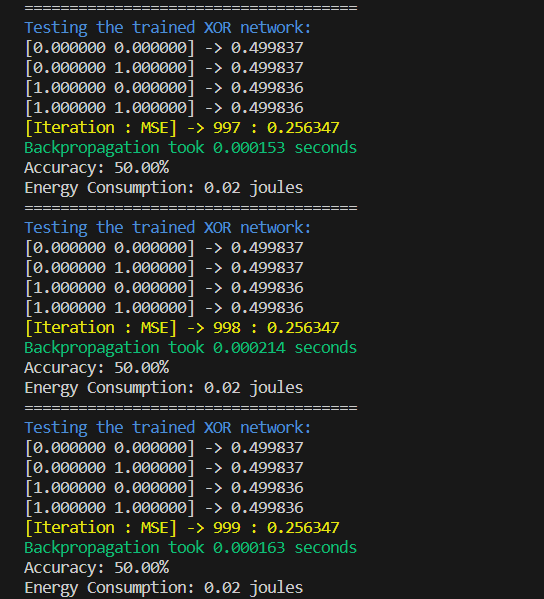
*user@host: partI$* ***make***

*user@host: partI$* ***./train\_nn train -h 12 -n 1000 -lr 0.1 -s 123 xor.train***

Figura 6 - Comparação do XOR nn com e sem seed (n = 1000, h = 12)



**A – Teste XOR nn sem seed**



**B – Teste XOR nn com seed**

* **Teste nº 2**

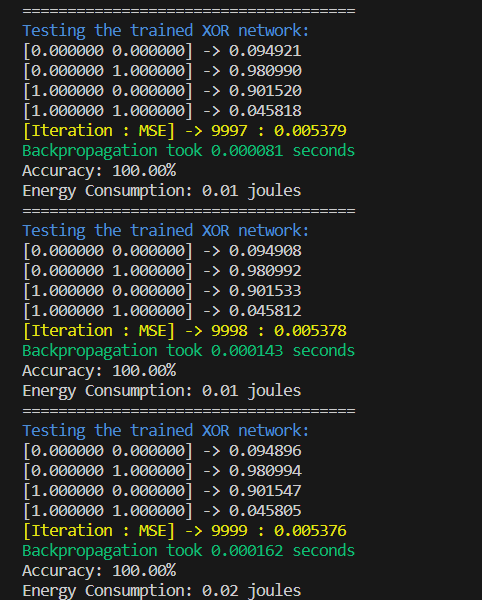
*user@host: partI$* ***make***

*user@host: partI$* ***./train\_nn train -h 12 -n 10000 -lr 0.5 xor.train***

*user@host: partI$* ***make***

*user@host: partI$* ***./train\_nn train -h 12 -n 10000 -lr 0.5 -s 123 xor.train***

**A – Teste XOR nn sem seed**



**B – Teste XOR nn com seed**

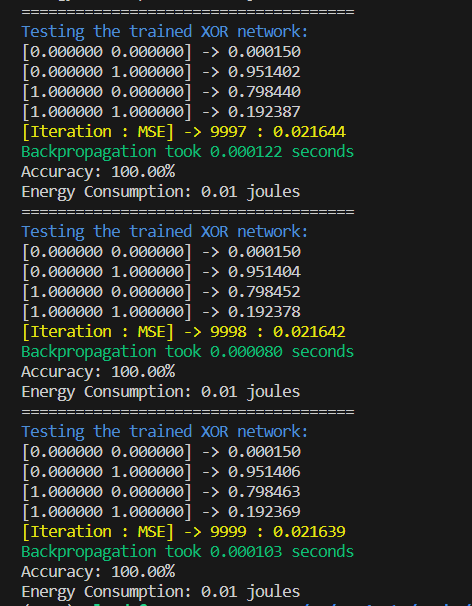
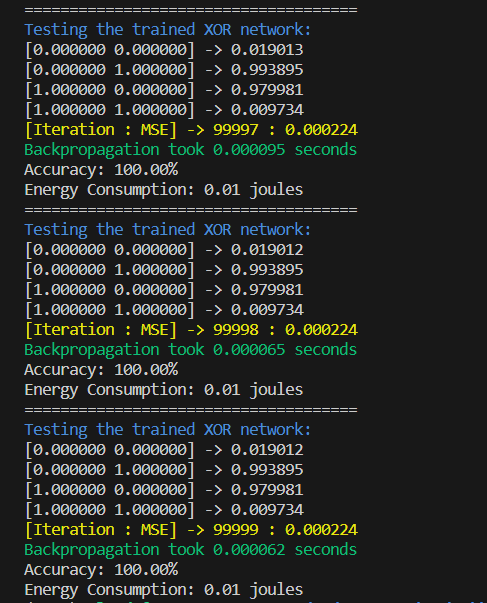
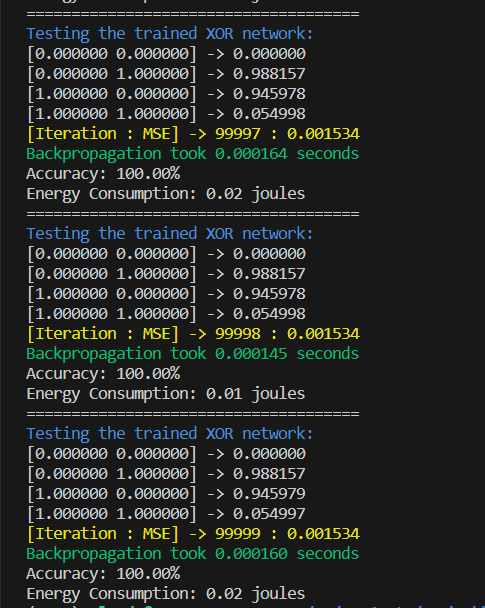


Figura 7 - Comparação do XOR nn com e sem seed (n = 10000, h = 12)



**A – Teste XOR nn sem seed**



**B – Teste XOR nn com seed**

Figura 8 - Comparação do XOR nn com e sem seed (n = 100000, h = 12)

Para testar os resultados obtidos da Figura 8, pode-se utilizar a CLI testados na Figura 7, alterando apenas o número de iterações.

* **Teste nº 3**

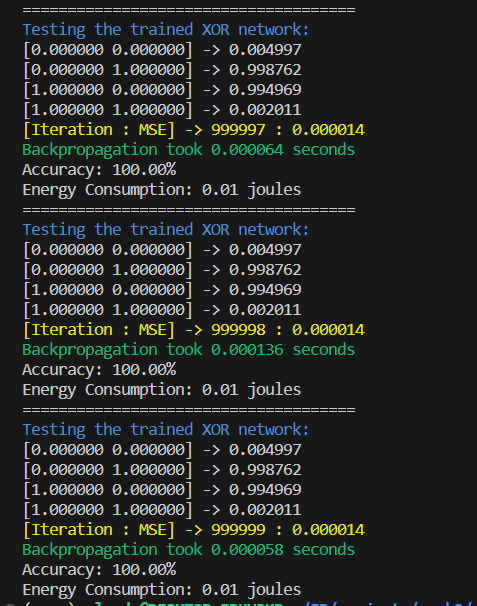
*user@host: partI$* ***make***

*user@host: partI$* ***./train\_nn train -h 12 -n 1000000 -lr 0.5 xor.train***

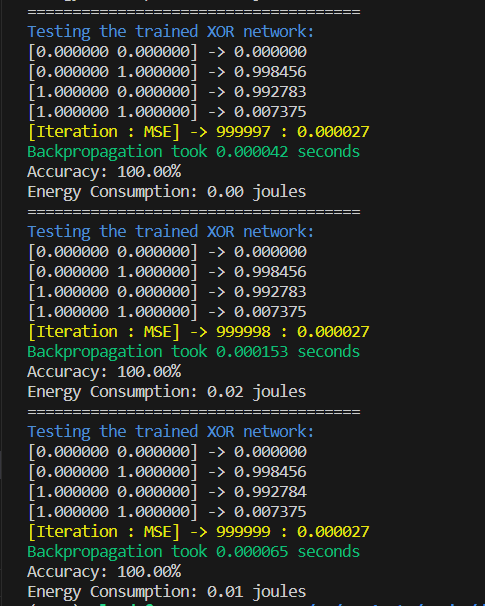
*user@host: partI$* ***make***

*user@host: partI$* ***./train\_nn train -h 12 -n 1000000 -lr 0.5 -s 123 xor.train***

Figura 9 - Comparação do XOR nn com e sem seed (n = 1000000, h = 12)



**A – Teste XOR nn sem seed**



**B – Teste XOR nn com seed**

Em resumo, podemos constatar que quanto maior for o número de iterações e o *learning rate* ajustável (0,5 para o nosso caso), maiores são as probabilidades de produzir resultados ótimos. Isso pode ser verificado na Figura 9, onde a ***accuracy*** atinge o valor de 100%, com ***MSE*** próximo de zero. No entanto, é importante notar que valores de l*earning rate* superiores a 1 podem resultar em comportamentos ou resultados instáveis, sendo necessário um ajuste cuidadoso.

Vale ressaltar que realizámos vários testes e, neste relatório, apenas incluímos os mais importantes, de modo a não estender demasiado o relatório.

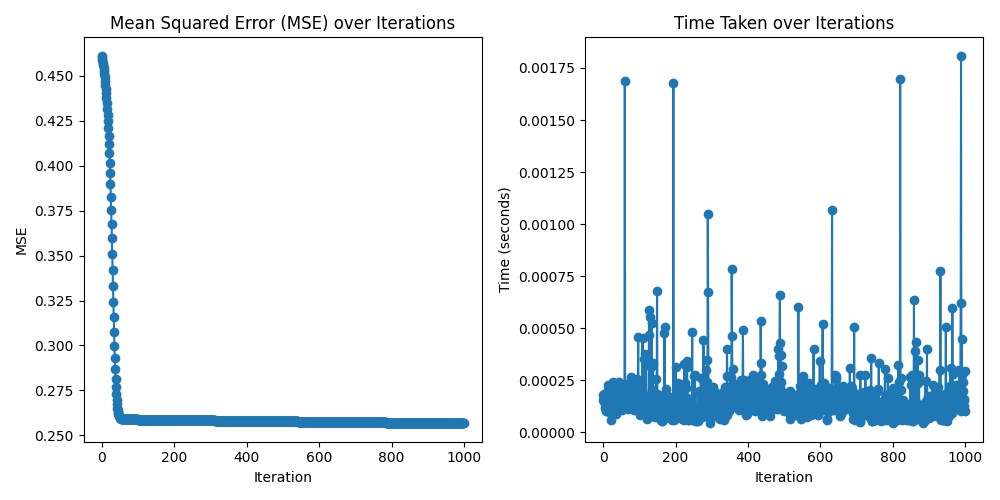
## 3.2 Análise de resultados

Uma vez testada a rede neural, a análise dos resultados implica uma análise pormenorizado de vários aspectos relacionados com o seu desempenho e comportamento. Os principais componentes da análise incluem:

**Comparação de desempenho:** a comparação do desempenho de diferentes testes efetuados anteriormente de redes neurais mostrou a avaliação do grau de propagação do modelo para diferentes conjuntos de dados ou variações nos parâmetros em diferentes situações no que concerne a ***n, h e lr***.

**Análise de erros:** consistiu em investigar os casos em que o modelo não consegue fazer previsões precisas. A compreensão do ***MSE*** pode orientar melhorias na arquitetura da rede neural ou no processo de formação. Pudemos averiguar que este erro se aproximou de zero quando ***n*** fosse maior e ***lr*** ajustável a 0.5.

**Visualizações:** visualizar as previsões do modelo, como o ***MSE*** em função do número de iterações e em relação ao tempo que o algoritmo de retropropagação leva para executar seus cálculos, permitiu constatar o quão importante é compreender a dinâmica do processo de aprendizagem da rede neural. Essa análise detalhada revelou padrões significativos, destacando a influência da taxa de aprendizagem na convergência do modelo e fornecendo insights valiosos para otimizar o desempenho do algoritmo tal como ilustras as Figuras 10, 11, 12, 13, 14, 15, 16 e 17.



.

Figura 10 - Gráfico do MSE, Tempo em Relação a n, sem seed (n = 1000, h = 12, lr=0.1)

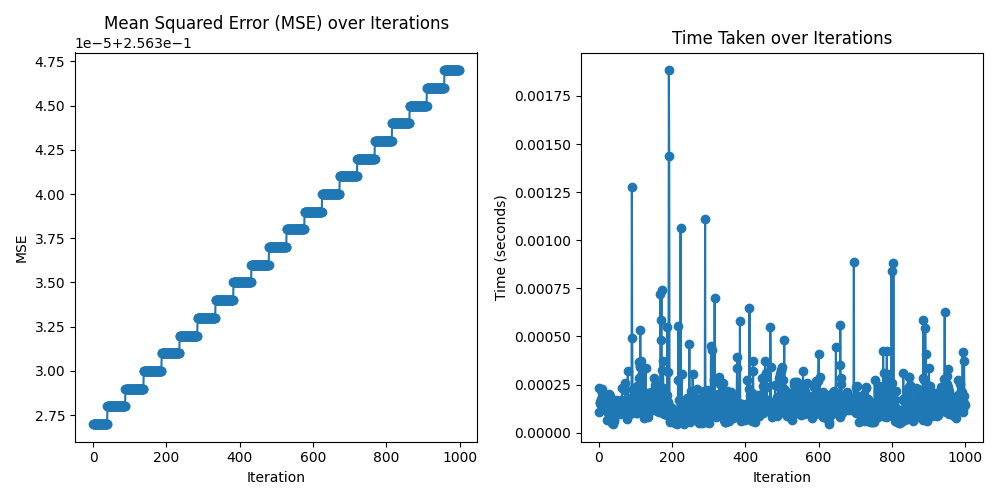


Figura 11 - Gráfico do MSE, Tempo em Relação a n, com seed (n = 1000, h = 12, lr=0.1, seed=123)

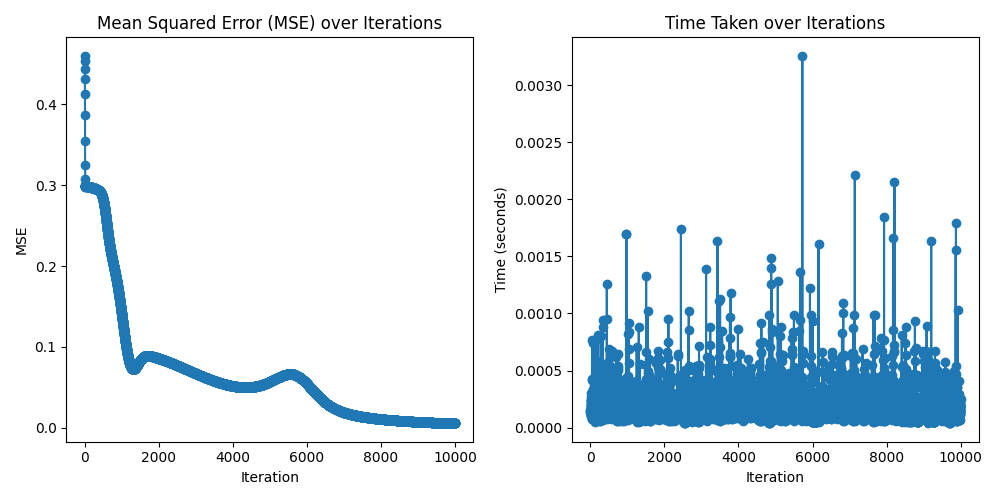


Figura 12 - Gráfico do MSE, Tempo em Relação a n, sem seed (n = 10000, h = 12, lr=0.5)

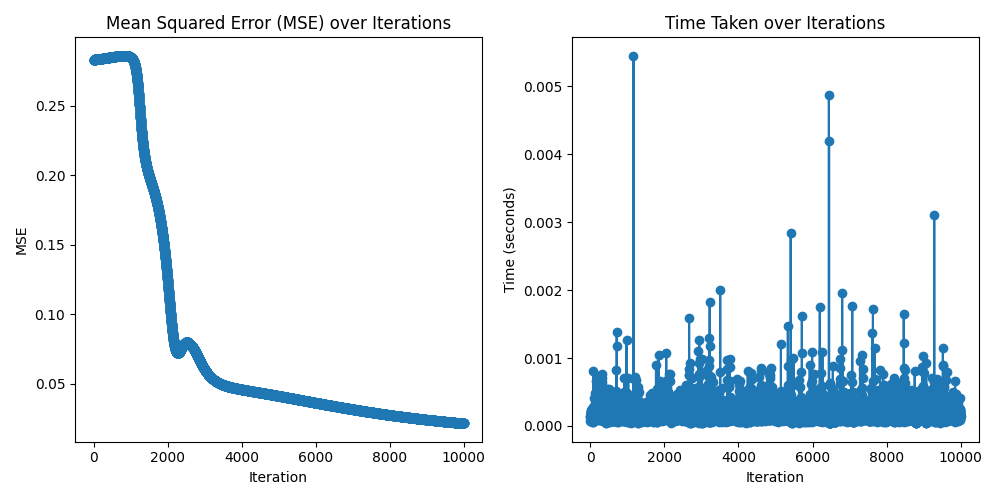


Figura 13 - Gráfico do MSE, Tempo em Relação a n, com seed (n = 10000, h = 12, lr=0.5, seed=123)

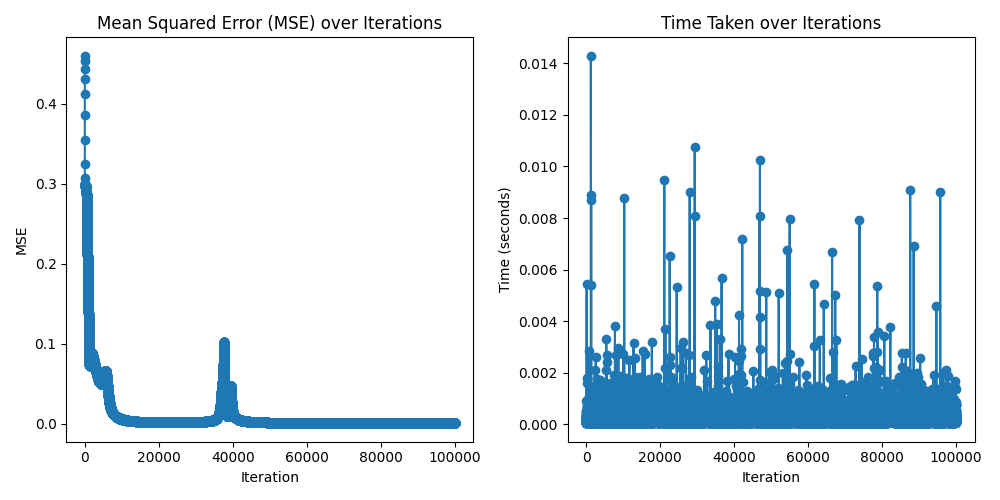


Figura 14 - Gráfico do MSE, Tempo em Relação a n, sem seed (n = 100000, h = 12, lr=0.5)

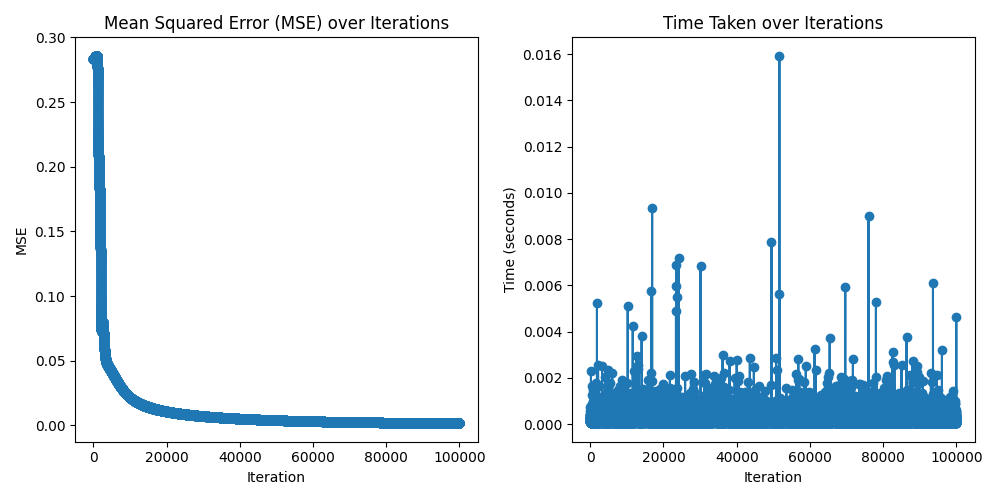


Figura 15 - Gráfico do MSE, Tempo em Relação a n, com seed (n = 100000, h = 12, lr=0.5, seed=123)

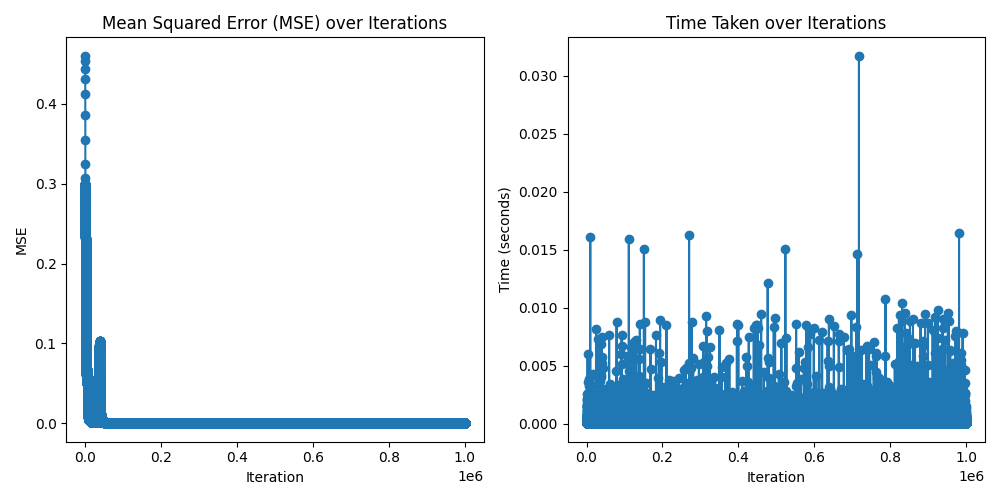


Figura 16 - Gráfico do MSE, Tempo em Relação a n, sem seed (n = 1000000, h = 12, lr=0.5)

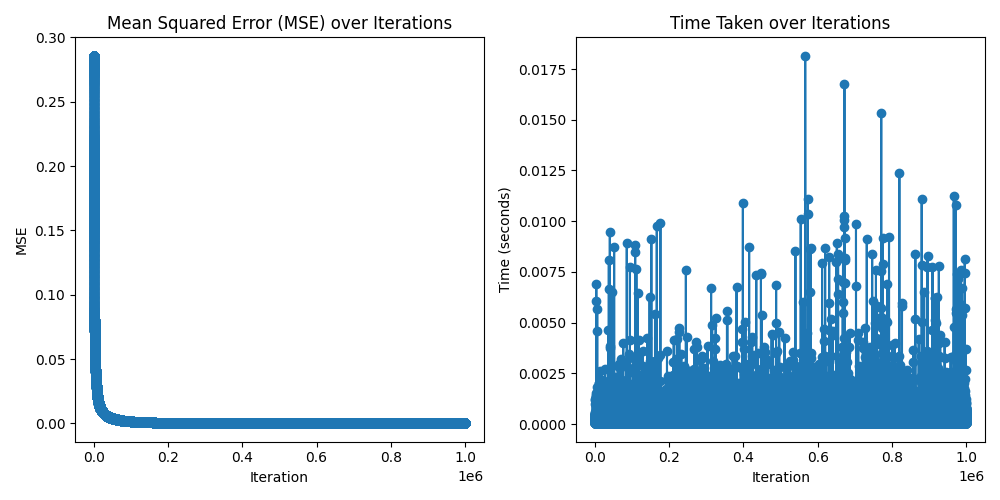


Figura 17 - Gráfico do MSE, Tempo em Relação a n, com seed (n = 1000000, h = 12, lr=0.5, seed=123)

**Discussão de resultados dos gráficos**

Como esperado, um aumento no tamanho do conjunto de dados (número de iterações), tal como ilustrado nas Figuras 16 e 17, resulta geralmente numa convergência mais suave e num MSE mais baixo (aproximado de zero), refletindo a capacidade melhorada da rede para aprender com mais dados.

As Figuras 12 e 17 mostram a sensibilidade da rede à taxa de aprendizagem. Embora uma taxa de aprendizagem mais elevada (***0.5 > 0.1***) possa conduzir a uma convergência mais rápida, pode também introduzir oscilações ou ultrapassagens no processo de formação, afetando o MSE final.

A consistência dos resultados em diferentes conjuntos de dados sugere que o número escolhido de camadas ocultas (12) é adequado para a tarefa/testes em causa. No entanto, é crucial notar que o número ótimo de camadas ocultas pode variar com base na complexidade do problema.

As Figuras 11, 13, 15 e 17 com sementes demonstram a importância da inicialização das sementes para a reprodutibilidade. As mesmas condições iniciais permitem uma comparação justa do desempenho do modelo.

Também verificamos que ocorreu instabilidade/discrepância no tempo de execução do algoritmo de retropropagação devido ao uso da função da biblioteca C - ***clock()***, que mede o tempo que a CPU leva para executar uma determinada tarefa. Além disso, é importante salientar que a máquina utilizada nos testes foi uma VM (VirtualBox).

Capítulo 4

# 4 Testes unitários e programação defensiva

Testes unitários e programação defensiva são práticas cruciais na engenharia de software que desempenham papéis fundamentais na criação de software (programa) robusto e confiável.

## 4.1 Testes unitários

Os testes unitários (TU) são projetados para verificar o funcionamento de unidades individuais de código, como funções ou métodos. Eles ajudam a identificar erros e problemas de lógica no código logo no início do desenvolvimento, o que é crucial para corrigir problemas antes que se tornem mais complexos e difíceis de resolver.

No projeto, criamos testes unitários para identificar erros e garantir que os programas criados em cada trecho do código funcionem corretamente.

## 4.2 Programação defensiva

A programação defensiva (PD) envolve a escrita de código para prevenir erros e lidar com entradas inesperadas ou situações de erro. Isso inclui a validação das entradas do utilizador e a verificação de pré-condições e pós-condições. Além disso, a PD ajuda a tornar o software mais robusto, reduzindo a probabilidade de falhas e crashes.

No projeto, não aprofundámos demasiado a programação defensiva, mas utilizámos conceitos introdutórios sobre este assunto, como a verificação de pré-condições.

Para depurar os programas, utilizamos duas ferramentas importantes: o GNU Debugger (gdb) e Valgrind.

Para testar os testes unitários deste projeto, é necessário seguir as instruções que se segue e, assegurar que está no diretório partI :

*user@host: partI$* ***make***

*user@host: partI$* ***./unit\_tests***

Para limpar os executáveis, basta fazer ***make clean*** no diretório correspondente.

Capítulo 5

# 5 Contribuição dos autores

A participação de cada autor foi excelente no que diz respeito à pesquisa, discussão e escrita deste trabalho, destacando-se o esforço de cada membro do grupo em ralação ao desenvolvimento do código. A percentagem de contribuição para cada estudante fica como segue:

* Jodionísio Muachifi – 40%
* Miguel Simões – 32%
* Gustavo Reggio – 28%

Capítulo 6

# 6 Conclusão

Com este trabalho, foi possível compreender na prática o funcionamento da construção de uma estrutura básica de rede neural em linguagem de programação C, implementada com o auxílio do algoritmo de retropropagação e testada através da função XOR. Este estudo evidencia o quão poderosa uma rede neural pode ser na prática, especialmente quando se ajustam cuidadosamente os (hiper)parâmetros para obter resultados ótimos. Observamos o melhor desempenho da nossa rede ao ajustar o número de iterações para 1.000.000, a taxa de aprendizagem para 0,5 e a semente (*seed*) mais baixa.

Em suma, as características discutidas contribuíram coletivamente para a eficácia e adaptabilidade da rede neural. A interação entre as funções de ativação, o algoritmo de treino proposto e o ajuste dos (hiper)parâmetros permitiu que a rede neural se comportasse de forma eficiente, alcançando um ***MSE*** aproximado de zero.

Referências bibliográficas

* <https://www.w3schools.com/c/c_structs.php>
* https://en.wikipedia.org/wiki/Pointer\_(computer\_programming)
* <https://www.youtube.com/watch?v=Ilg3gGewQ5U>
* <https://elearning.ua.pt/mod/url/view.php?id=1336755>
* <https://en.wikipedia.org/wiki/Backpropagation>
* <https://en.wikipedia.org/wiki/Neural_network>
* <https://en.wikipedia.org/wiki/GNU_Debugger>
* https://en.wikipedia.org/wiki/Valgrind