

Projeto 1 - EP

Universidade de Aveiro

João Torrinhas, Diogo Torrinhas



Projeto 1 - EP

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro

João Torrinhas, Diogo Torrinhas
(98435) joao.torrinhas@ua.pt, (98440) diogotorrinhas@ua.pt

October 30, 2023

Contents

1	Part I	2
1.1	Exercise 1	2
1.1.1	User Manual	2
1.1.2	Code Explanation	2
1.1.3	Results	2
1.2	Exercise 2	3
1.2.1	User Manual	3
1.2.2	Code Explanation	3
1.2.3	Results	3
1.3	Exercise 3	4
1.3.1	User Manual	4
1.3.2	Code Explanation	4
1.3.3	Results	4
1.4	Exercise 4	4
1.4.1	User Manual	4
1.4.2	Code Explanation	5
1.4.3	Results	5
2	Part II	6
2.1	User Manual	6
2.2	Code Explanation	6
2.2.1	nn_base.h	6
2.2.2	nn_base.c	7
2.2.2.1	Method nn_create	7
2.2.2.2	Method propagate	8
2.2.2.3	Method load_input_values	8
2.2.2.4	Method load_nn	9
2.2.2.5	Method write_nn	9
2.2.2.6	Method nn_destroy	10
2.2.2.7	Method main	10
2.3	Results	11
2.3.1	Example 1	11
2.3.2	Example 2	12
3	Author's Contribution	14

Chapter 1

Part I

1.1 Exercise 1

1.1.1 User Manual

Para compilar e correr este programa basta seguir os seguintes passos:

1. Abrir o terminal no diretório onde se encontra o programa `print_bit.c`.
2. Após estar no diretório correto, use o seguinte comando no terminal para compilar o programa:

```
gcc -o print_bit print_bit.c
```

Após correr este comando vai ser criado um executável com o nome `print_bit`.

3. Em seguida, no mesmo terminal, use o seguinte comando para correr o executável `print_bit`:

```
./print_bit
```

Este comando vai correr o programa compilado `print_bit` e vai ser possível ver no terminal o resultado do programa.

1.1.2 Code Explanation

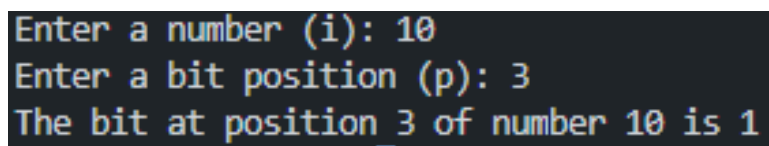
O programa `print_bit` é um programa que calcula e exibe o 'p'-ésimo bit (0 ou 1) na representação binária de um número inteiro. Isto é atinjido deslocando os bits (i) do número inteiro para as posições 'p' direitas e, em seguida, extraindo o bit menos significativo (LSB) através da operação AND com o valor 1.

Listing 1.1: `printbit`

```
printf("The bit at position %d of number %d is %d\n", p, i, (i >> p) & 1);
```

1.1.3 Results

Após corrermos o programa, obtemos o resultado na figura abaixo.



```
Enter a number (i): 10
Enter a bit position (p): 3
The bit at position 3 of number 10 is 1
```

Figure 1.1: Resultado

Tal como é possível ver pela imagem de cima, o programa está a funcionar corretamente. Foi introduzido o número 10, que corresponde ao valor binário `...0000 1010` (32 bits), e foi selecionada a posição 3 tendo como resultado o valor 1 porque na posição 3 da representação binário do número 10 encontra-se o bit 1.

1.2 Exercise 2

1.2.1 User Manual

Para compilar e correr este programa basta seguir os seguintes passos:

1. Abrir o terminal no diretório onde se encontra o programa `print_bits.c`.
2. Após estar no diretório correto, use o seguinte comando no terminal para compilar o programa:

```
gcc -o print_bits print_bits.c
```

Após correr este comando vai ser criado um executável com o nome `print_bits`.

3. Em seguida, no mesmo terminal, use o seguinte comando para correr o executável `print_bits`:

```
./print_bits
```

Este comando vai correr o programa compilado `print_bits` e vai ser possível ver no terminal o resultado do programa.

1.2.2 Code Explanation

O programa `print_bits.c` é um programa que converte um número inteiro na sua representação binária, e que a exibe bit a bit.

O programa começa por pedir ao utilizador que insira um número inteiro e em seguida o programa irá calcular o número de bits que irão ser necessário para representar esse número (Como se trata de um inteiro, irão ser 32 bits). O núcleo do programa é o ciclo for que irá iterar cada bit do inteiro uma vez que, o inteiro irá ser guardado na memória como uma sequência de bits. O programa faz isto deslocando os bits do número de entrada (**i** representa o número de entrada) e usando a operação AND, bit a bit, para extrair cada bit.

Listing 1.2: printbits

```
for (int j = numBits - 1; j >= 0; j--){
    printf("%d", (i >> j) & 1);
}
```

O bit extraído(0 ou 1) é então impresso no terminal até obtermos a representação binária do número pretendido.

1.2.3 Results

Após correremos o programa, obtemos o resultado na figura abaixo.

[illegible]

Figure 1.2: Resultado

Tal como é possível ver pela figura de cima, a transformação de decimal para binário foi um sucesso.

1.3 Exercise 3

1.3.1 User Manual

Para compilar e correr este programa basta seguir os seguintes passos:

1. Abrir o terminal no diretório onde se encontra o programa `bits_to_int`.
2. Após estar no diretório correto, use o seguinte comando no terminal para compilar o programa:

```
gcc -o bits_to_int bits_to_int.c -lm
```

Após correr este comando vai ser criado um executável com o nome `bits_to_int`.

3. Em seguida, no mesmo terminal, use o seguinte comando para correr o executável `bits_to_int`:

```
./bits_to_int
```

Este comando vai correr o programa compilado `bits_to_int` e vai ser possível ver no terminal o resultado do programa.

1.3.2 Code Explanation

O programa `bits_to_int` é um programa que converte uma string binária no seu respetivo número decimal. Começa por pedir ao utilizador que insira uma string de binária de 32 bits (inteiro). Em seguida, o programa usa um ciclo `for` para percorrer a string e depois converte a string binária em sua representação decimal. Ele faz isso multiplicando o valor de cada dígito binário (0 ou 1) pela potência de 2 correspondente nessa posição, seguindo a maneira convencional para transformar um número binário em um número decimal, e acumula os resultados na variável `num`. A subtração de 48 ('0' em ASCII) é usada para converter o dígito (0 ou 1) no seu respetivo inteiro, por exemplo, se for ('1'-48) o resultado é 1.

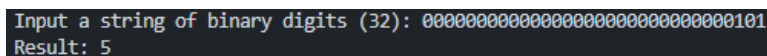
Listing 1.3: bitstoint

```
for(int i = NUMBER_BITS-1; i >= 0; i--){
    num += pow(2,31-i)*(bits[i]-48);
}
```

Por fim, imprime o valor decimal.

1.3.3 Results

Após corrermos o programa, obtemos o resultado na figura abaixo.



```
Input a string of binary digits (32): 00000000000000000000000000000101
Result: 5
```

Figure 1.3: Resultado

Tal como podemos ver pela figura de cima, o resultado de binário para decimal foi um sucesso.

1.4 Exercise 4

1.4.1 User Manual

1. Abrir o terminal no diretório onde se encontra o programa `bits.c`.
2. Após estar no diretório correto, use o seguinte comando no terminal para compilar o programa:

```
gcc -o bits bits.c -lm
```

Após correr este comando vai ser criado um executável com o nome `bits`.

Chapter 2

Part II

2.1 User Manual

Para compilar e correr este programa basta seguir os seguintes passos:

1. Abrir o terminal no diretório onde se encontra o programa `nn_base.c`.
2. Após estar no diretório correto, use o seguinte comando no terminal para compilar o programa:

```
gcc -o nn_base nn_base.c
```

Após correr este comando vai ser criado um executável com o nome `nn_base`.

3. Em seguida, no mesmo terminal, use o seguinte comando para correr o executável `nn_base`:

```
./nn_base
```

Este comando vai correr o programa compilado `nn_base` e vai ser possível ver no terminal o resultado do programa.

2.2 Code Explanation

Neste capítulo vai ser explicado em detalhe todo o código desenvolvido no exercício 5 da Parte II.

2.2.1 `nn_base.h`

O ficheiro `nn_base.h` é um *header file* e define a estrutura e as funções necessárias para criar, destruir, administrar e usar uma rede neural básica.

O elemento central do ficheiro `nn_base.h` é a estrutura de dados **NeuralNetwork** que vai armazenar os atributos necessários para a mesma, nomeadamente o número de *input*, *hidden*, *output units* que existem na *NeuralNetwork*, o valor das arestas que ligam cada uma das *layers* e o valor de cada *unit* tal como é mostrado no excerto de código abaixo.

Listing 2.1: NeuralNetwork Structure

```
typedef struct
{
    int number_I; // Number of Input Units
    int number_H; // Number of Hidden Units
    int number_O; // Number of Output Units
    double** weights_to_hidden; // I->H
    double** weights_to_output; // H->O
    double* I; // Array to store input values
    double* H; // Array to store hidden layer values
    double* O; // Array to store output value
} NeuralNetwork;
```


Neste ficheiro também é feita a declaração de um conjunto de funções/métodos, tal como foi dito mais acima.

No próximo capítulo vão ser explicados em pormenor cada um destes métodos.

Listing 2.2: Declared Functions

```
NeuralNetwork* nn_create(int num_inputs, int num_hidden, int num_outputs);
void propagate(double input_values[], NeuralNetwork *nn);
void load_input_values(double input_values[], NeuralNetwork *nn);
void load_nn(NeuralNetwork *nn);
void write_nn(NeuralNetwork *nn);
void nn_destroy(NeuralNetwork *nn);
```

2.2.2 nn_base.c

Neste ficheiro vão ser implementadas as funções declaradas no ficheiro **nn_base.h** e é onde vai ser implementada a função *main* para executar o que é pedido no enunciado.

2.2.2.1 Method nn_create

A função **nn_create** é principalmente responsável por criar uma nova rede neural alocando memória para a matriz que vai guardar os valores das arestas e para o array com os valores de cada *unit* inicializando também as arestas a 0. Além disso, ainda vai ler a primeira linha do ficheiro de entrada para atribuir os valores do número de unidades aos respetivos atributos (**number_I**, **number_H** e **number_O**) da estrutura da rede neural. Por fim, retorna um ponteiro para a rede neural criada.

Listing 2.3: Read first line and close file

```
FILE *fp = fopen("NeuralNetwork1.txt", "r");
int number_inputs, number_hidden, number_outputs;

// Read the first line (I H O)
if (fscanf(fp, "%d_%d_%d", &number_inputs, &number_hidden, &number_outputs) == 3) {
    nn->number_I = number_inputs;
    nn->number_H = number_hidden;
    nn->number_O = number_outputs;
    fclose(fp);
}
```

A alocação de memória necessária para os arrays com os valores de cada unidade vai ter em conta o tipo de dados, no caso `double`, e o número das mesmas, ou seja, vai ser determinado pela multiplicação do tamanho de um `double` e pelo número de unidades dadas pelos atributos **number_I**, **number_H** e **number_O** da estrutura da rede.

Listing 2.4: Allocation for input, hidden, and output layers

```
// Allocate memory for input, hidden, and output layers
nn->I = malloc(sizeof(double) * nn->number_I);
nn->H = malloc(sizeof(double) * nn->number_H);
nn->O = malloc(sizeof(double) * nn->number_O);
```

No caso das matrizes que contêm os valores das arestas entre unidades, o processo é muito semelhante. No exemplo do atributo **weights_to_hidden** (variável que contêm as arestas que ligam a *layer* inicial à *layer* do meio), o tamanho para a alocação de memória é determinado multiplicando **nn->number_I** (o número de unidades de entrada) pelo tamanho de um ponteiro para um `double` (**sizeof(double*)**). Este processo vai alocar memória para uma matriz de ponteiros, onde cada ponteiro apontará para uma linha da matriz. Para a outra variável que une as arestas entre a *layer* do meio e a *layer* de saída o processo é semelhante.

Posto isto, vai ter de ser alocada também memória para essas linhas da matriz e, por fim, vão ser inicializadas cada aresta a 0.

Listing 2.5: Allocation for matrix's

```
//Initialize weights
nn->weights_to_hidden = (double**) malloc (nn->number_I * sizeof(double*));
nn->weights_to_output = (double**) malloc (nn->number_H * sizeof(double*));

//Initialize weight matrices and layers to zero
for (int i = 0; i < nn->number_I; i++) {
    nn->weights_to_hidden[i] = (double*) malloc (nn->number_H * sizeof(double));
    for (int j = 0; j < nn->number_H; j++) {
        nn->weights_to_hidden[i][j] = 0.0;
    }
}

for (int i = 0; i < nn->number_H; i++) {
    nn->weights_to_output[i] = (double*) malloc (nn->number_O * sizeof(double));
    for (int j = 0; j < nn->number_O; j++) {
        nn->weights_to_output[i][j] = 0.0;
    }
}
```

2.2.2.2 Method propagate

A função **propagate** vai receber como parâmetros um vetor de entrada e um ponteiro para a rede neural. Através desse vetor de entrada e dos valores presentes nas arestas da rede neural vai ser calculado o produto interno entre a camada de entrada e a camada do meio (Propagar *input* para *hidden layer*) e, em seguida, entre a camada do meio e a camada de saída (Propagar *hidden* para *output layer*) com o objetivo de se determinar o valor das saídas.

Em baixo, encontra-se o código referente a esta função.

Listing 2.6: Propagate function

```
void propagate(double input_values[], NeuralNetwork *nn){
    //Propagate input to hidden layer
    for(int i = 0; i < nn->number_H; i++){
        double aux = 0.0;
        for(int j = 0; j < nn->number_I; j++){
            aux += nn->I[j] * nn->weights_to_hidden[j][i];
        }
        nn->H[i] = aux;
    }

    //Propagate hidden to output layer
    for(int i = 0; i < nn->number_O; i++){
        double aux = 0.0;
        for(int j = 0; j < nn->number_H; j++){
            aux += nn->H[j] * nn->weights_to_output[j][i];
        }
        nn->O[i] = aux;
    }
}
```

2.2.2.3 Method load_input_values

Esta função tem como objetivo guardar os valores presentes no vetor de entrada (vetor de entrada é um parâmetro da função) no array **I** da estrutura de dados, que guardará os valores das unidades entrada.

2.2.2.4 Method `load_nn`

Em primeiro lugar, a função começa por tentar abrir um ficheiro (neste exemplo chama-se `NeuralNetwork1.txt`) em modo leitura usando a função `fopen()` e caso o mesmo não possa ser aberto, será exibida no terminal uma mensagem de erro.

Listing 2.7: Open file in read mode

```
FILE *fp = fopen("NeuralNetwork1.txt", "r");

if (fp == NULL) {
    perror("Error_opening_file");
    return;
}
```

Posto isto, vai ser lido o texto presente no ficheiro. A primeira linha, que contém a informação sobre o número de unidades de cada camada, é pulada porque na função `nn_create` essa linha já é lida para atribuir o número de unidades às respetivas *layers* na estrutura de dados da rede. Em seguida, vão ser lidas as linhas seguintes que seguem um formato específico, sendo que cada linha possui a estrutura: **layer:unit layer:unit peso**.

Enquanto as linhas estiverem neste formato, vão ser lidos os pesos das arestas que unem as respetivas unidades entre as *layers* e vão ser atribuídos esses pesos às respetivas variáveis da rede. Quando a próxima linha não estiver no formato específico, é terminada a leitura e fechado o ficheiro usando a função `fclose()`.

Listing 2.8: Read lines from the file

```
// Skip the first line because it has already been readed in the method nn_create(I H O)
char line[256];
fgets(line, sizeof(line), fp);

// Read the remaining lines with weight values
while(fscanf(fp, "%d:%d:%d:%lf", &fl, &input_unit, &sl, &hidden_unit, &weight) == 5){

    if (fl == 1 && sl == 2) {
        nn->weights_to_hidden[input_unit - 1][hidden_unit - 1] = weight;
    } else if (fl == 2 && sl == 3){
        nn->weights_to_output[input_unit - 1][hidden_unit - 1] = weight;
    }
}

fclose(fp);
```

2.2.2.5 Method `write_nn`

Esta função tem como objetivo escrever num ficheiro de dados a estrutura da rede neural. Vai ser aberto um ficheiro em *write mode* que vai ser representado por um ponteiro e, em seguida, vai ser escrito nesse ficheiro a configuração da rede neural, seguindo a estrutura do ficheiro de entrada, através dos valores presentes nas variáveis que contêm os pesos das arestas entre as *layers*. O código apresentado em baixo mostra como é feita a abertura do ficheiro para escrita e a escrita da configuração da rede neural no ficheiro.

Listing 2.9: Open file in write mode

```
FILE *fp = fopen("NeuralNetworkOutput.txt", "w");

if (fp == NULL) {
    perror("Error_opening_file");
    return;
}
```

Listing 2.10: Write to the file

```

fprintf(fp, "%d_%d_%d\n", nn->number_I, nn->number_H, nn->number_O);

for (int i = 0; i < nn->number_I; i++) {
    for (int j = 0; j < nn->number_H; j++) {
        fprintf(fp, "1:%d_2:%d_%lf\n", i + 1, j + 1, nn->weights_to_hidden[i][j]);
    }
}

for (int i = 0; i < nn->number_H; i++) {
    for (int j = 0; j < nn->number_O; j++) {
        fprintf(fp, "2:%d_3:%d_%lf\n", i + 1, j + 1, nn->weights_to_output[i][j]);
    }
}

```

2.2.2.6 Method `nn_destroy`

Na função `nn_destroy` é realizada a limpeza da memória libertando a memória alocada dinamicamente para a rede neural, ou seja, desaloca os arrays e matrizes associados à rede.

2.2.2.7 Method `main`

Na função `main` é onde está implementada a resolução do enunciado da pergunta 5 com a ajuda de um conjunto de funções que estão explicadas em cima.

Em primeiro lugar, vai ser criada a rede neural com duas *input units*, duas *hidden units* e uma *output unit* (valores no ficheiro de entrada para este exemplo), usando a função `nn_create`. Em seguida, vão ser lidos pesos/valores pré-treinados das arestas entre as camadas para a rede.

Listing 2.11: Create Neural Network and load file

```

NeuralNetwork *nn = nn_create();
load_nn(nn);

```

Posto isto, os valores de entrada vão ser carregados para a rede neural para serem propagados (produto interno) com o objetivo de determinar os valores de saída.

Listing 2.12: Load input values to network and propagate

```

double input_values[] = {2.0, 1.0};
load_input_values(input_values, nn);
propagate(input_values, nn);

```

Por fim, depois da propagação, vão ser "printados" o resultado dos valores de saída no terminal, vai ser escrita a estrutura da rede neural num ficheiro de saída (pesos das arestas entre camadas) seguindo a estrutura do ficheiro de entrada e vai ser limpa/desalocada a memória usada pela rede neural terminando assim o programa principal.

Listing 2.13: Print output values, write to the file and free memory

```

for (int i = 0; i < nn->number_O; i++) {
    printf("Output[%d]: %lf\n", i, nn->O[i]);
}
write_nn(nn);
nn_destroy(nn);
return 0;

```

2.3 Results

2.3.1 Example 1

A Figura 2.1, mostra o exemplo de um esboço da rede neural que vai ser lida do ficheiro da Figura 2.2.

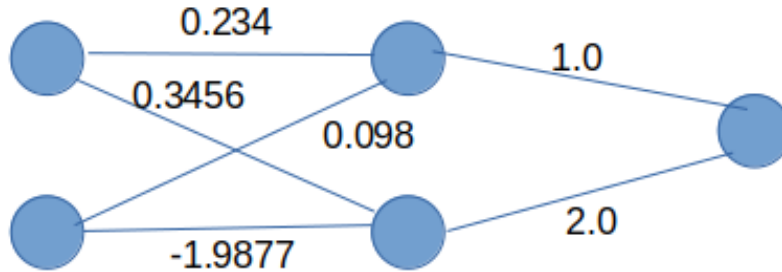


Figure 2.1: Esboço da Neural Network

```
1 2 2 1
2 1:1 2:1 0.234
3 1:1 2:2 0.3456
4 1:2 2:1 0.098
5 1:2 2:2 -1.9877
6 2:1 3:1 1.0
7 2:2 3:1 2.0
```

Figure 2.2: Ficheiro de entrada com as configurações da rede neural

Após correr o programa, é criado um ficheiro de saída e é mostrado no terminal o resultado das unidades de saída.

```
1 2 2 1
2 1:1 2:1 0.234000
3 1:1 2:2 0.345600
4 1:2 2:1 0.098000
5 1:2 2:2 -1.987700
6 2:1 3:1 1.000000
7 2:2 3:1 2.000000
```

Figure 2.3: Ficheiro de saída

Assumindo que o vetor de entrada é $\{2.0, 1.0\}$, os cálculos a serem feitos na propagação da rede neural são os seguintes:

$$\begin{aligned}
H[0] &= (2 \cdot 0.234) + 1 \cdot (0.098) = 0.566 \\
H[1] &= 2 \cdot (0.3456) + 1 \cdot (-1.9877) = -1.2965 \\
O[0] &= (0.566 \cdot 1) + (-1.2965 \cdot 2) = -2.027
\end{aligned}$$

Pelos cálculos de cima, é possível verificar que o resultado da única unidade de saída existente é de -2.027 estando de acordo com o resultado que aparece no terminal após correr o programa **nn_base**.

```

joao@joaoT-pc:~/Desktop/Mestrado/EP/Project1$ ./nn_base
First input: 1, Second input: 1
First input: 1, Second input: 2
First input: 2, Second input: 1
First input: 2, Second input: 2
First input: 1, Second input: 1
First input: 2, Second input: 1
Weights from Input to Hidden Layer:
Weight I1 to H1: 0.234000
Weight I1 to H2: 0.345600
Weight I2 to H1: 0.098000
Weight I2 to H2: -1.987700
Weights from Hidden to Output Layer:
Weight H1 to O1: 1.000000
Weight H2 to O1: 2.000000
Output[0]: -2.027000

```

Figure 2.4: Resultados após execução do programa

2.3.2 Example 2

Na Figura de abaixo, encontra-se outro exemplo de um esboço da rede neural, usada para testar este programa, que vai ser lida pelo ficheiro de entrada representado na Figura 2.6.

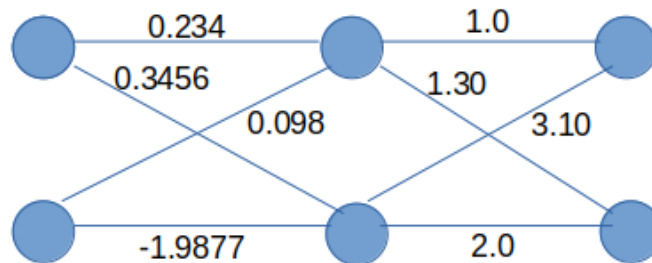


Figure 2.5: Esboço 2 da Neural Network

```

1 2 2 2
2 1:1 2:1 0.234
3 1:1 2:2 0.3456
4 1:2 2:1 0.098
5 1:2 2:2 -1.9877
6 2:1 3:1 1.0
7 2:1 3:2 1.30
8 2:2 3:1 3.10
9 2:2 3:2 2.0

```

Figure 2.6: Ficheiro de entrada 2 com as configurações da rede neural

Após correr o programa, é criado um ficheiro de saída e é mostrado no terminal o resultado das unidades de saída semelhante ao exemplo 1.

```
1 2 2 2
2 1:1 2:1 0.234000
3 1:1 2:2 0.345600
4 1:2 2:1 0.098000
5 1:2 2:2 -1.987700
6 2:1 3:1 1.000000
7 2:1 3:2 1.300000
8 2:2 3:1 3.100000
9 2:2 3:2 2.000000
```

Figure 2.7: Ficheiro de saída 2

Assumindo que o vetor de entrada é igual ao exemplo 1 ($\{2.0, 1.0\}$), os cálculos a serem feitos na propagação desta rede neural, neste exemplo, são os seguintes:

$$\begin{aligned}H[0] &= (2 \cdot 0.234) + 1 \cdot (0.098) = 0.566 \\H[1] &= 2 \cdot (0.3456) + 1 \cdot (-1.9877) = -1.2965 \\O[0] &= (0.566 \cdot 1) + (-1.2965 \cdot 3.10) = -3.45315 \\O[1] &= (-1.2965 \cdot 2) + (0.566 \cdot 1.30) = -1.8572\end{aligned}$$

Pelos cálculos de cima, é possível verificar novamente que os resultados das unidades de saída estão de acordo com os resultados que aparecem no terminal, após a execução do programa.

```
Weights from Input to Hidden Layer:
Weight I1 to H1: 0.234000
Weight I1 to H2: 0.345600
Weight I2 to H1: 0.098000
Weight I2 to H2: -1.987700
Weights from Hidden to Output Layer:
Weight H1 to O1: 1.000000
Weight H1 to O2: 1.300000
Weight H2 to O1: 3.100000
Weight H2 to O2: 2.000000
Output[0]: -3.453150
Output[1]: -1.857200
joao@joaoT-pc:~/Desktop/Mestrado/EP/Project1$
```

Figure 2.8: Resultados após execução do programa

Chapter 3

Author's Contribution

Todos participaram de forma igual na divisão e elaboração deste projeto, pelo que a percentagem de contribuição de cada aluno fica:

- João Torrinhas - 50%
- Diogo Torrinhas - 50%