ARQUITETURA DE COMPUTADORES

2020-2021

Laboratório 2 – Rede Neuronal Artificial em Assembly

Este laboratório destina-se a consolidar conhecimentos de introdução à programação na linguagem Assembly da arquitetura RISC-V, utilizando o simulador Ripes[1].

O trabalho deve ser realizado fora do horário de laboratório, destinando-se este à demonstração e avaliação do trabalho realizado. No final da aula de laboratório deverá submeter o código Assembly no Fénix.

Para garantir a correção da solução, deverá validar a sua solução do laboratório no emulador (Ripes) e confirmar os resultados, observando os valores finais dos registos ou o conteúdo da memória. Pode igualmente colocar pontos de paragem no código (de tal forma que o emulador parará a execução sempre que atingir um destes pontos) clicando nos números das linhas correspondentes.

Introdução

Uma rede neuronal artificial é um sistema computacional que imita as redes de neurónios que formam o sistema nervoso central dos animais. Estas são formadas por unidades elementares, designadas por neurónios. Um neurónio artificial calcula uma saída y a partir de um conjunto de valores de entrada x_i usando a expressão:

$$y = f\left(\sum_{i=1}^{N} w_i x_i + b\right),$$

em que w_i são os pesos das entradas, b é um termo constante de *bias* e $f(x)$ é uma função que depende do tipo de neurónio e é, em geral, não linear. Esta função pode ser, por exemplo, a função sigmoide, $h_S(x) = 1/(1 + e^{-\alpha x})$, a função de retificação, $h_R(x) = max(0, x)$, entre outras. Neste laboratório, em particular, vamos utilizar a função degrau, que corresponde a uma sigmoide quando $\alpha \to +\infty$,

$$f(x) = \lim_{\alpha \to +\infty} h_S(x) = \begin{cases} 1, if\ x \geq 0 \\ 0, if\ x < 0 \end{cases}$$

Na figura 1 apresenta-se o diagrama de blocos correspondente a um neurónio com duas entradas.

[1] https://github.com/mortbopet/Ripes

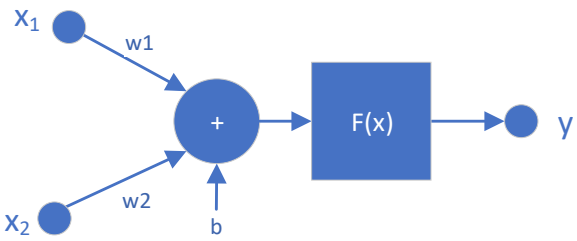

Figura 1 - Neurónio artificial com duas entradas.

Uma rede neuronal pode ser configurada para implementar funções arbitrárias das entradas, dada um número suficiente de neurónios e uma arquitetura adequada. No laboratório, pretende-se implementar uma pequena rede de 2 entradas que implementa a função XOR. Esta rede está representada na Figura 2. Deve-se considerar que as entradas A e B tomam apenas os valores 0 ou 1.

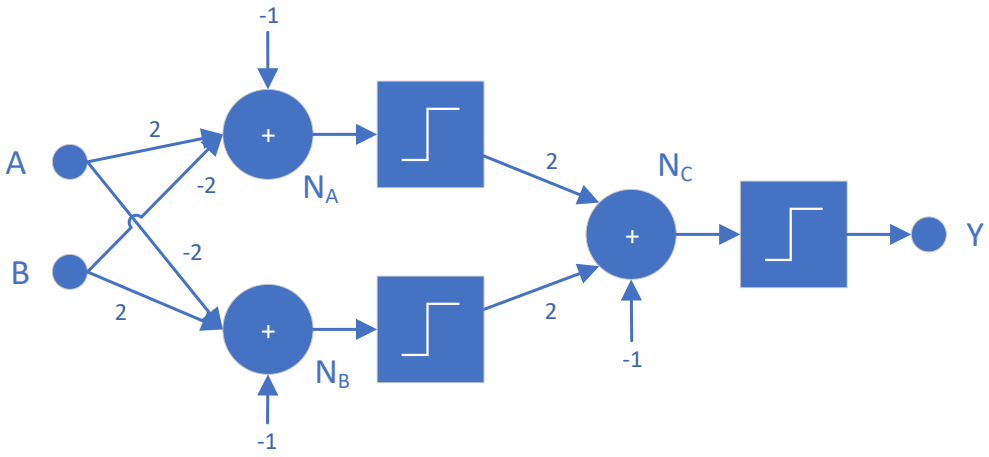

Figura 2 - Pequena rede neuronal que implementa o operador lógico XOR.

O neurónio N_A implementa a função lógica $A\overline{B}$. O neurónio N_B implementa a função lógica $\overline{A}B$. O neurónio N_C implementa a função lógica OR. Daqui resulta que o sinal na saída será $y = A\overline{B} + \overline{A}B = A \oplus B$. Esta pequena rede neuronal é implementada pelo seguinte código em C:

```
int neuronio(int x1, int x2, int w1, int w2, int b) {
	int s;

	s = x1 * w1 + x2 * w2 + b;

	if (s >= 0) return 1;
	else return 0;
}

int rede_neuronal_xor(int a, int b) {
	int c, d, y;
	c = neuronio(a, b, 2, -2, -1);
	d = neuronio(a, b, -2, 2, -1);
	y = neuronio(c, d, 2, 2, -1);

	return y;
}
```

Exercício 1

a) Pretende-se implementar, em assembly, a rede neuronal da Figura 2, baseando-se no código C apresentado. Contudo, assume-se a utilização de uma arquitetura simplificada do processador RISC-V, onde **não foi disponibilizada a unidade de execução que realiza a operação de multiplicação**. Por conseguinte, será necessário implementar todas as operações de multiplicação através de outras operações elementares (ex: somas sucessivas). Para isso devem ser implementadas 3 funções em assembly:

1. Função **`multiplica`**:
   Recebe dois valores pela pilha (multiplicando e multiplicador) e retorna o resultado do produto também pela pilha. O resultado deve ser calculado por somas sucessivas do multiplicando utilizando um ciclo que itera tantas vezes quanto o valor absoluto do multiplicador. O sinal do resultado deve ser determinado utilizando as regras operatórias habituais.
2. Função **`neuronio`**:
   Deve corresponder à função do código em C apresentado, de forma a implementar um neurónio **genérico**, chamando a função `multiplica`. A passagem de parâmetros deverá ser realizada por registo, seguindo a convenção de compilador, tal como indicado no *RISC-V Reference Card*.
3. Função **`rede_neuronal_xor`**:
   Deve corresponder à função em C e chamar a função `neuronio`. Tal como no caso anterior, a passagem de parâmetros deverá ser realizada por registo, seguindo a convenção de compilador, tal como indicado no *RISC-V Reference Card*.

Não serão consideradas soluções que não obedeçam a esta estrutura.

b) Escreva o código assembly para testar a rede implementada, invocando a função **`rede_neuronal_xor`** para os quatro valores possíveis da tabela de verdade: 00, 01, 10, e 11. Execute o programa e chame o docente para verificar o resultado.

Exercício 2

Este é um exercício surpresa que será divulgado pelo docente durante a aula.