UNIVERSITÀ DI PISA

Computer Engineering

Electronic and Communication Systems

Perceptron

Project Report

TEAM MEMBERS:
Olgerti Xhanej

Academic Year: 2020/2021

Contents

1 — Introduction

1.1 Problem Description

The main goal of the activity described in this report is the following: realizing a network implementing a **perceptron** with a **sigmoid activation function**.
Before describing the whole design and implementation process a very little introduction about the architecture must be done.

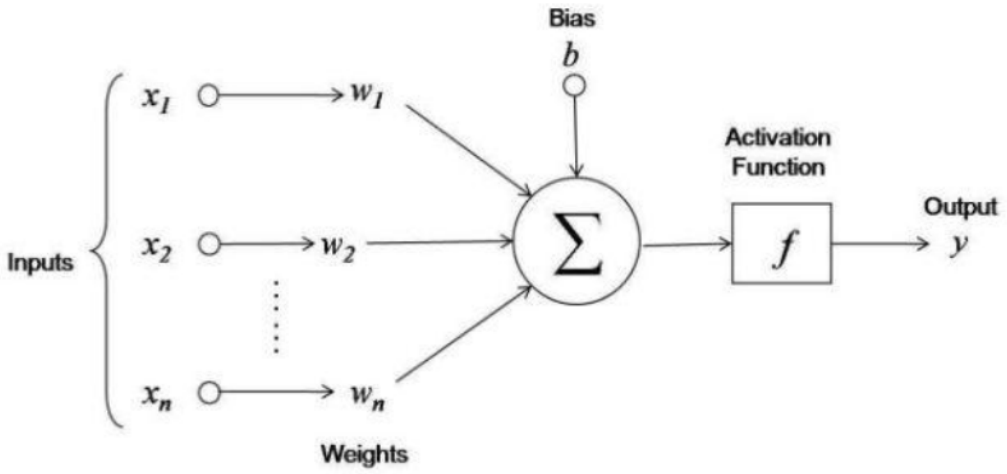


Figure 1: Perceptron Architecture

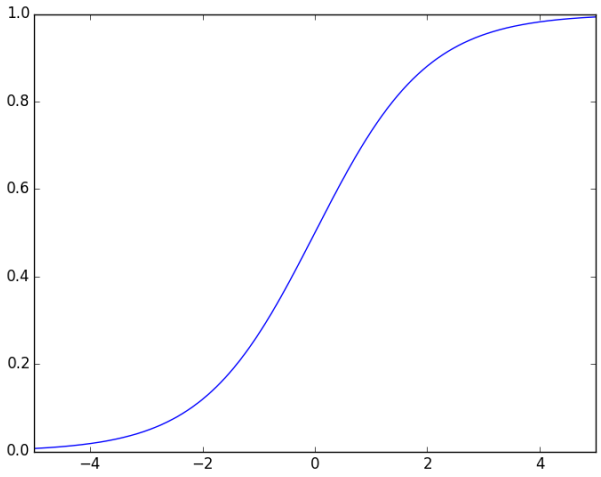
A **Perceptron** is a *binary classifier that maps his inputs to a specific output y = f(z), where f() is the* ***activation function*** *of the perceptron.* The inputs are real numbers and the input z of the activation function is obtained as:

$$z = b + \sum_{i=0}^{N_L - 1} w_i x_i \tag{1}$$

Every input x_i, every weight w_i and the bias b are real numbers in the range of $[-1, 1]$.
The **activation function**, in our case, will be a **sigmoid function**, described as follows:

$$y = \frac{1}{1 + e^{-z}} \tag{2}$$

Figure 2: Sigmoid Function Plot

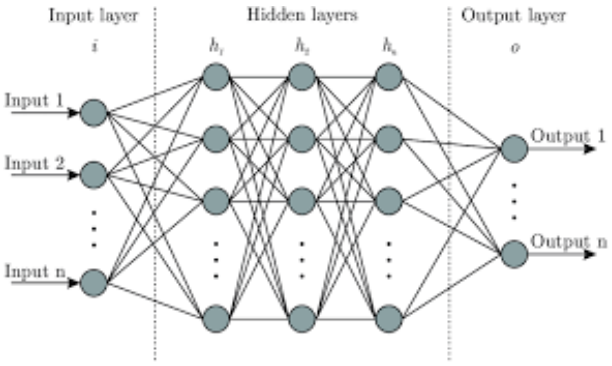


Where z is the result of the equation (1.1).

1.2 Applications

A single perceptron is the building block of *artificial neural networks*, in which different layers of perceptrons are connected. The output of the neural network is a real number and could be use to classify *complex objects*: patterns, human faces, handwritings, medical diagnosis, e-mail spams.

Figure 3: Neural network example



In the image above there is a simple schema of a neural network, in which the circles represent the perceptrons.

1.3 Possible Architectures

The main architecture will be made up by three main logical parts, from an higher-lever point of view:

- **Multiplication Circuit**: implementation of the multiplication operation between each input x_i and each weight w_i.
- **Adder Circuit**: implementation of the addition between the results of the former phase and the bias b.
- **Activation Function Circuit**: implementation of the computation of the sigmoid function.

In the next chapter the architecture will be documented with more precision. Different project choices could be made for each logical part of the architecture:

- **Multiplication Circuit**: could be implemented through a **ROM-based solution** in which every possible result is stored and the two inputs represent the addresses for getting the result. This solution is good only with a very low number of bits, which is not our case: in fact the the ROM will be composed by $2^{(n_{w_i}+n_{b_i})}$ memory cells. In order to implement the multiplication circuit will be implemented through a **Paraller Multiplier**.
- **Adder Circuit**: different choice could be made to implement the adder circuit. Starting from the simplest to the more complex solution we can exploit the **Serial Adder**, the **Parallel Adder** or the **Parallel Adder with Pipeline** . The first one needs less logic but requires n clock cycles for computing an n bits result. The second solution improves the first one by computing one result in **one clock cycle**, on the other hand it could add some problems due to long logic chains between two register. The third solution is the best from the perspective of the number of clock cycles required and the **critical path**, in fact by adding some registers in between the computation of the bits will reduce the logic chains.
- **Activation Function Circuit**: As seen during the laboratory class, this part will be implemented by exploiting a Look-Up-Table. In order to do so, could be necessary a **truncation** of the result of the former computation in order to limit the size of the LUT. With 12 bits are necessary $2^{12} = 4096$ entries, which could be even reduced by performing some optimization by exploiting the sigmoid function symmetry.

2 — Architecture

In this chapter will be discussed deeply the architecture of the three main parts of the perceptron. The general structure could be summarized by the following schema:

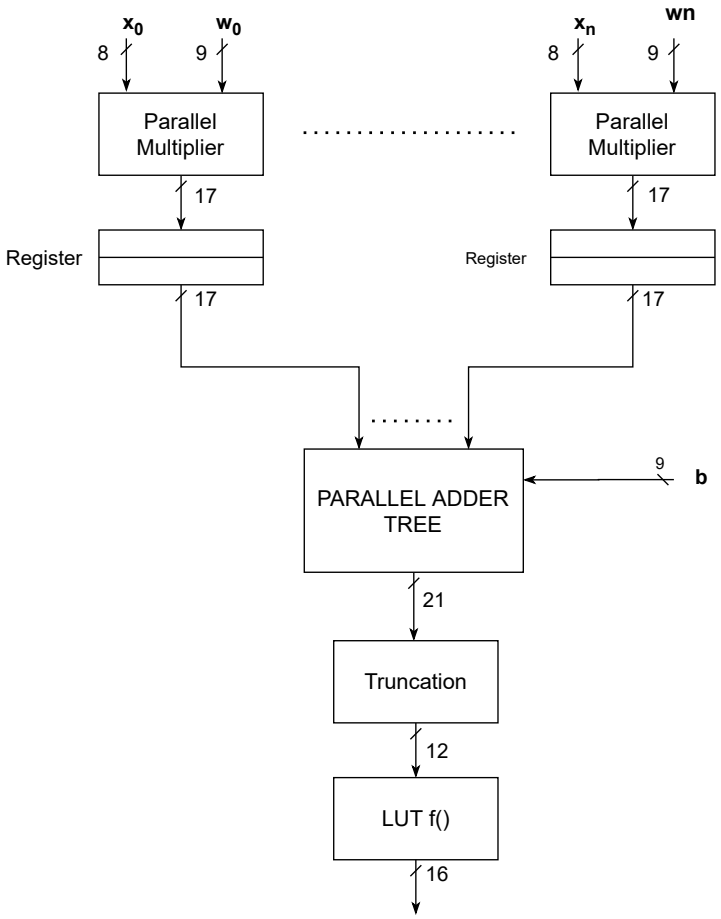


Figure 4: General Schema

2.1 Multiplication Circuit Architecture

The Multiplication Circuit, as said before, will be implemented through a Parallel Multiplier. The inputs b_i and w_i are composed respectively by $b_x = 8$ bits and $b_w = 9$ bits. In order to compute the multiplication in the correct way, the inputs need to be translated in the **unsigned form** and then is

possible to perform the multiplication with the parallel multiplier. In the following image is presented the architecture with these particular inputs.

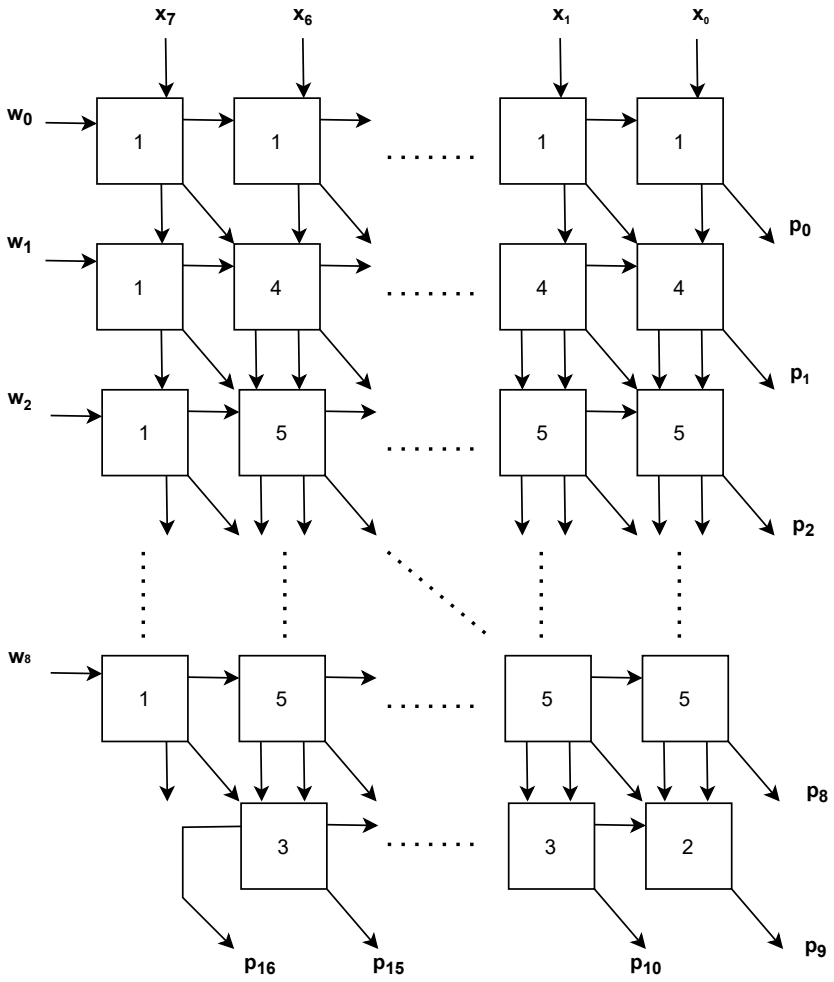


Figure 5: Parallel Multiplier Architecture

2.2 Adder Circuit Architecture

In order to compute the equation (1.1) different sums need to be computed. The building block of this part will be the **Parallel Adder with Pipeline**: as said before, by adding some registers in between the Carry chains, the critical path impact can be reduced. Furthermore, by exploiting the parallel architecture, a single sum can be computed in a single clock cycle. In the next figure will be presented the Parallel Adder:

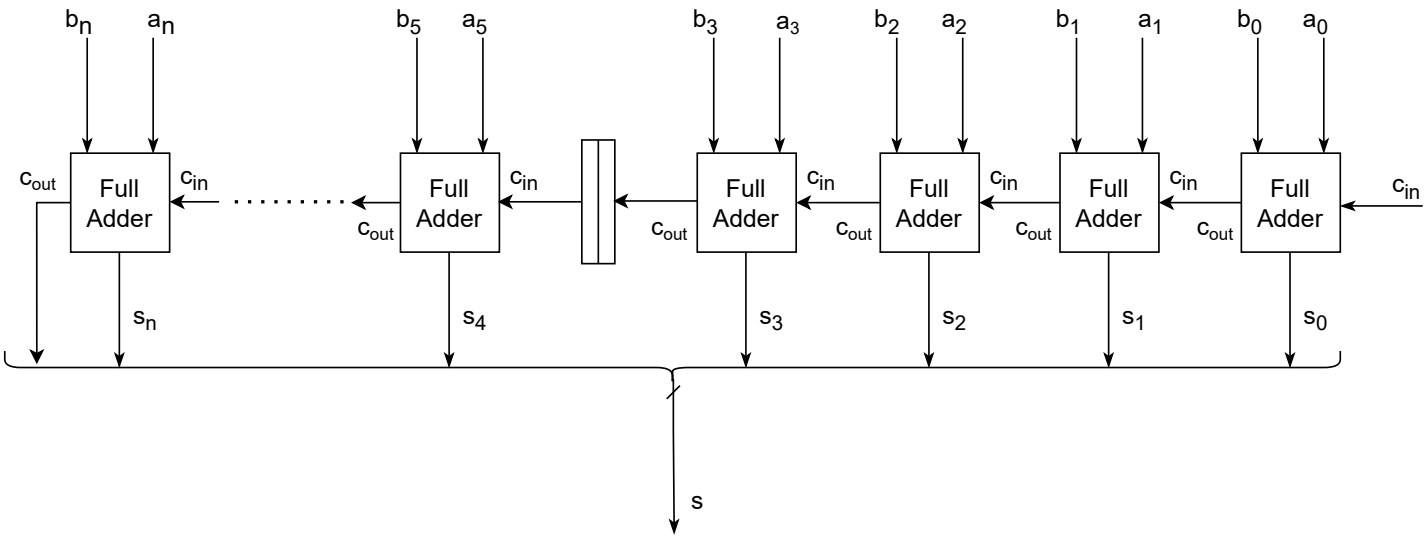


Figure 6: Parallel Adder Architecture

To implement the whole sum of 11 terms, in order to decrease the number of cycles needed to compute the whole sum and to reduce the number of bits needed, a tree approach has been chosen. The schema of the tree parallel adder is the following:

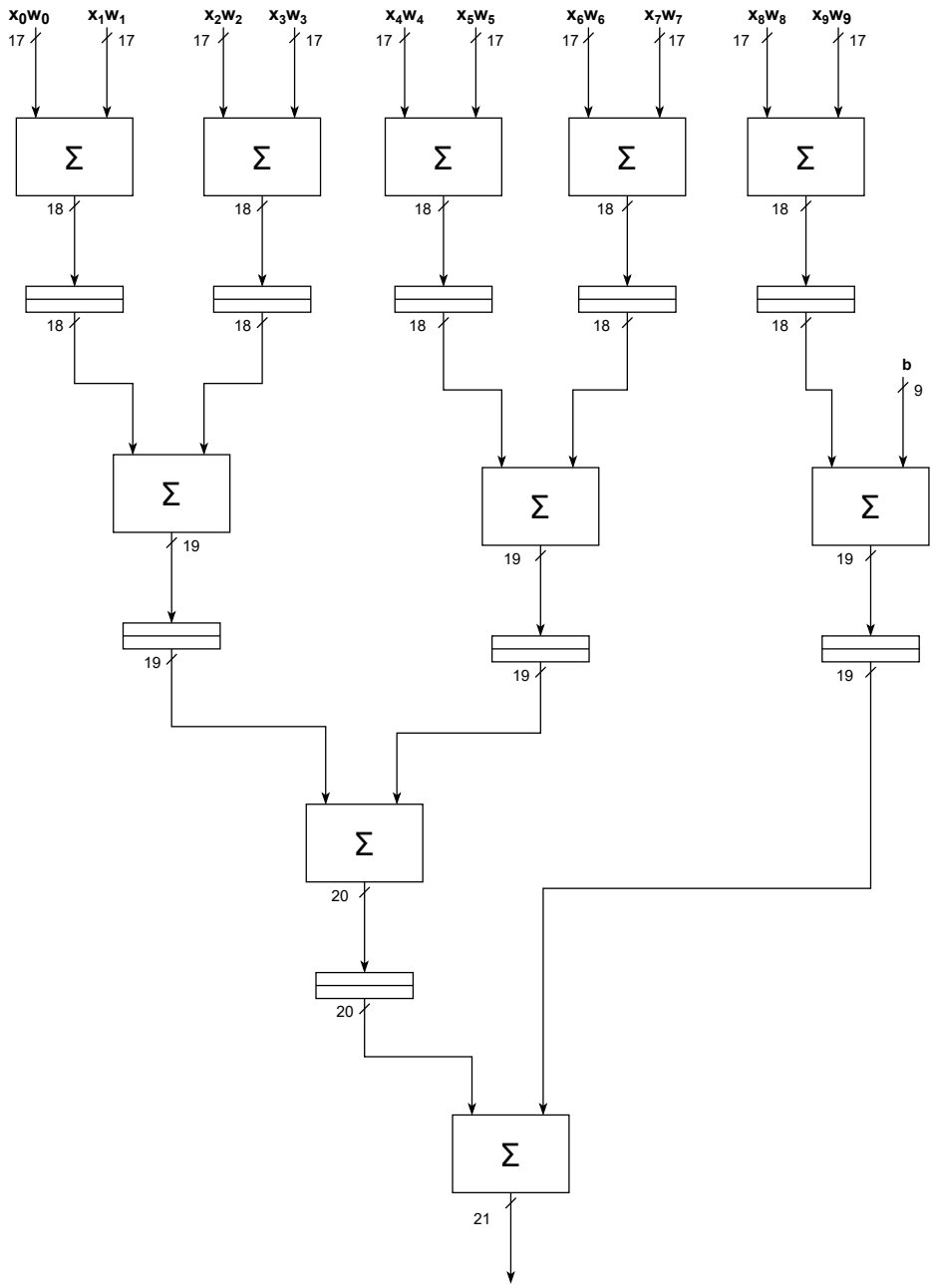


Figure 7: Parallel Multiplier Architecture

Some register has been put in between the sum to limit the critical path impact on the performances and clock period limit.

2.3 Activation Function Circuit Architecture

At the end of the computation of the latter phase the output is composed by 21 bits. The computation of the sigmoid function will be done through a **Look-Up-Table**, which will need 2^{21} =**2097152 entries** of different outputs with 16 bits. In order to reduce the size of the Look-Up-Table a truncation is needed: from 21 bits to 12 bits. In this case the Look-Up Table will be composed by 2^{12} =**4096 entries**, but, by exploiting the **odd symmetry** of the sigmoid, only 4096/2 = **2048 entries** are needed.

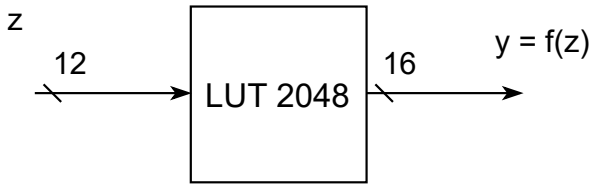


Figure 8: Look-Up Table Architecture

3 — VHDL CODE

4 — Test Plan

5 — XILINX VIVADO Report

6 — Conclusion