Neural Logic: Theory and Implementation

Thesis by

Vasken Bohossian

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1998

(Submitted July 17, 1998)

© 1998
Vasken Bohossian
All Rights Reserved

Acknowledgements

I would like to thank my advisor, Professor Jehoshua Bruck, for his guidance and support throughout my stay at Caltech. He helped me develop my skills as a researcher and taught me how to efficiently communicate my ideas. I would also like to thank the students of the Paradise group, at Caltech, for many great moments spent together. Finally, I would like to express my gratitude to Tanya, my grand-mother Alice, and my parents Zaven and Taki for their encouragement and love.

The research presented in this thesis was supported in part by the NSF Young Investigator Award CCR-9457811, by the Sloan Research Fellowship, by a grant from the IBM Almaden Research Center, San Jose, California, and by the center for Neuromorphic Systems Engineering as a part of the National Science Foundation Engineering Research Center Program; and by the California Trade and Commerce Agency, Office of Strategic Technology.

Abstract

Human brains are by far superior to computers in solving hard problems like combinatorial optimization and image and speech recognition, although their basic building blocks are several orders of magnitude slower. This observation has boosted interest in the field of artificial neural networks [20], [37]. The latter are built by interconnecting artificial neurons whose behavior is inspired by that of biological neurons. In this thesis we consider the Boolean version of an artificial neuron, namely, a Linear Threshold (LT) element, which computes a neural-like Boolean function of n binary inputs [32]. An LT element outputs the sign of a weighted sum of its Boolean inputs. The main issues in the study of networks (circuits) consisting of LT elements, called LT circuits, include the estimation of their computational capabilities and limitations and the comparison of their properties with those of traditional Boolean logic circuits based on AND, OR and NOT gates (called AON circuits). For example, there is a strong evidence that LT circuits are more efficient than AON circuits in implementing a number of important functions including the addition, product and division of integers [44], [45].

It is easy to see that an LT element is more powerful than an AON gate, simply because of the freedom one has in selecting the weights. Indeed, different choices of weights produce different Boolean functions. As a matter of fact, the number of n-input Boolean functions that can be implemented by a single LT element is of the order of 2^{n^2}, [42], [22]. That additional power comes at the cost of added complexity. Some LT functions require weights that are very different in magnitude, potentially rendering difficult hardware or software implementations of the correponding LT elements. For that reason, theoretical research in the field of LT circuits has focused on the weights, in particular the power of LT elements with restricted weights. As early as 1971, Muroga, [32], proved that any linear threshold element can be implemented with integer weights. That is, by restricting the magnitudes of the weigths to

natural numbers, one does not lose any of the power of the original LT element. We generalize this result to arbitrary subsets of the set of real numbers. For example, we show that one can restrict the weights to be the squares of integers, and still be able to realize all LT functions. We ask the following question. What are the conditions on the subset $D \in R$ which guarantee that all LT functions can be implemented with weights drawn from it?

Another aspect of the complexity of the weights is their growth as the number of inputs increases. It has been shown [17], [33], [38], [43] that there exist linear threshold functions that can be implemented by a single threshold element with exponentially growing weights, but cannot be implemented by a threshold element with smaller : polynomialy growing weights. In light of that result the above question was dealt with by defining a class, called $\widehat{LT}$, within the set of linear threshold functions : the class of functions with "small" (i.e. polynomialy growing) weights [43]. We focus on a single LT element. Our contribution consists in two novel methods for constructing threshold functions with minimal weights, which allows us to fill up the gap between polynomial and exponential weight growth by further refining the separation. Namely, we prove that the class of linear threshold functions with polynomial-size weights can be divided into subclasses, $\widehat{LT}^{(d)}$, according to the degree, d, of the polynomial. In fact, we prove a more general result—that there exists a linear threshold function for any arbitrary number of inputs and any weight size.

Even though some LT functions require weights that grow exponentially with the number of input variables, it has been shown recently, in [13], [18], that such functions can be replaced by a two-layer circuit composed of LT gates with polynomially growing, i.e., small weights. We improve the best known bound on the size of that circuit, presented in [18] by focusing on a particular function with large coefficients. We also derive explicit two-layer circuits. Two-layer LT circuits are in general composed of different linear threshold elements, but for some useful Boolean functions, such as *parity*, *addition* and *product*, the gates of the first layer are almost identical. To take advantage of this fact we introduce a new Boolean computing element. Instead of the sign function, it computes an arbitrary (with polynomialy many transitions)

Boolean function of the weighted sum of its inputs. We call the new computing element an LTM element, which stands for Linear Threshold with Multiple transitions. The advantages of LTM become apparent in the context of VLSI implementation. Indeed, this new model reduces the layout area of the corresponding symmetric function from $O(n^2)$ to $O(n)$. We present a VLSI implementations of both LT and LTM elements. Two kinds of elements were fabricated, programmable and hardwired. The programmable elements use the charge on a floating gate in order to store the values of the weights.

For many years, the topic of linear threshold logic, has been approached in two different ways, theory, i.e. computational circuit complexity, [38], [56], and hardware implementation, [48], [40]. Surprisingly, there has been very little interaction between those two approaches. As a whole, the present thesis is one step towards establishing a connection between the theory and implementation of threshold circuits. Its constributions are at three levels. At the theoretical level, new classes of functions such as $\widehat{LT}^{(d)}$ and LTM are defined and their computational power is estimated. At the algorithmic level, we show how to convert real weights to weights drawn from an arbitrary subset of the real numbers, e.g., integer weights, we also show how to construct LT functions with minimal weights, and finally we present an algorithm that produces an $\widehat{LT}_2$ circuit (circuit composed of gates with small weights), that computes the comparison function, $COMP$. We also present LTM circuits computing useful functions, such as XOR, ADD, $PRODUCT$. At the implementation level, we show the design, layout and testing of the VLSI implementation of LT and LTM. Establishing a connection between the theoretical and practical aspects of threshold logic will profit both domains by providing solutions for practical problems and by defining new theoretical questions inspired by implementation issues.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

Human brains are by far superior to computers in solving hard problems like combinatorial optimization and image and speech recognition, although their basic building blocks are several orders of magnitude slower. This observation has boosted interest in the field of artificial neural networks [20], [37]. The latter are built by interconnecting artificial neurons whose behavior is inspired by that of biological neurons. In this thesis we consider the Boolean version of an artificial neuron, namely, a Linear Threshold (LT) element, which computes a neural-like Boolean function of n binary inputs [32]. An LT element outputs the sign of a weighted sum of its Boolean inputs. The main issues in the study of networks (circuits) consisting of LT elements, called LT circuits, include the estimation of their computational capabilities and limitations and the comparison of their properties with those of traditional Boolean logic circuits based on AND, OR and NOT gates (called AON circuits). For example, there is a strong evidence that LT circuits are more efficient than AON circuits in implementing a number of important functions including the addition, product and division of integers [44], [45].

Neural or linear threshold logic has been approached from two different directions: theory and implementation. Electronic circuits implementing LT elements have been proposed as early as the sixties, see [4]. Research in that field is still active today, see [40]. On the other hand, the more recent theoretical research related to LT has been conducted within the framework of computational circuit complexity, [38], [56]. It has been shown that certain Boolean functions such as exclusive-OR (XOR), can be implemented by a polynomial size LT circuit of constant depth, but require an exponentially large circuit if one uses the classical AON implementation. XOR being at the base of many useful functions, such as addition, product and division, researchers have been motivated to further investigate the power and limitations of the linear threshold model of computation. That task has proved to be surprisingly

difficult. Indeed, the only strong lower bound in the field is related to $\widehat{LT}_2$ the class of functions that can be implemented by a two-layer, polynomial size circuit of LT elements with small, i.e. polynomial, weights, [14]. In other words, a function was found, that is not in $\widehat{LT}_2$. If one allows the use of LT elements with arbitrary weights, i.e. LT_2, then no lower bound exists, that is no function has been found that cannot be implemented by a two layer, polynomial size circuit of LT elements with arbitrary weights.

There has been very little interaction between the theoretical and practical aspects of neural logic. The goal of the research presented in this thesis is to take one step towards bridging the gap between theory and implementation. In the remainder of this chapter we will address the main ideas presented in the thesis. In Section 1.1 we define LT, the class of linear threshold functions and present examples of common Boolean functions implemented using the LT model. Section 1.2 presents the main ideas related to the study of the weights of LT elements, it introduces the results presented in chapters 2, 3 and 4. Section 1.3 relates to chapters 5 and 6, it presents LTM, a new computing element derived from LT, as well as its VLSI implementation. Finally, in Section 1.4 we summarize the contributions of the thesis.

1.1 LT function: definition and examples

In this section we give a formal definition of the function computed by a linear threshold gate. We show examples of Boolean functions that can be implemented by a single LT element, in particular, we show how to compute AND, OR, MAJ and $COMP$, defined below.

The present thesis focuses on the study of linear threshold circuits, or LT circuits, which are composed of linear threshold gates. Those have binary inputs and output. They are mathematically described by a *linear threshold function.*

Definition 1.1 *(Linear Threshold Function)*
A linear threshold function of n variables is a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$

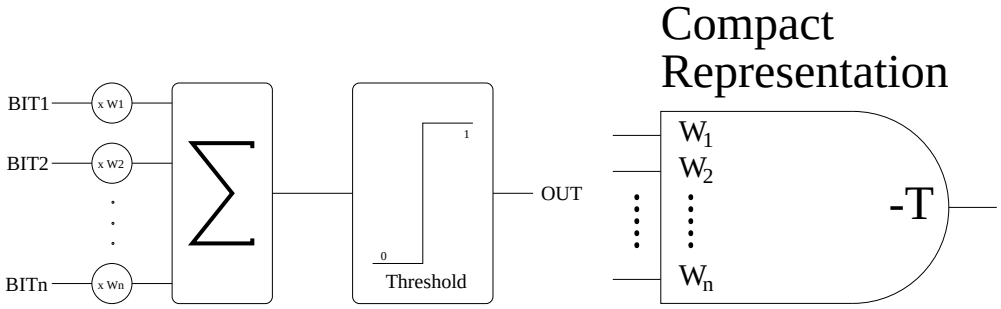


Figure 1.1: Linear Threshold Element $y = sgn(-t + \sum_{i=1}^{n} w_i x_i)$

that can be written, for any $x \in \{0,1\}^n$ *and a fixed* $w \in R^{n+1}$, *as:*

$$f(x) = sgn(F(x)) = \begin{cases} 1 & for\ F(x) \geq 0 \\ 0 & otherwise \end{cases}$$

$$where\ F(x) = w \cdot (1, x) = w_0 + \sum_{i=1}^{n} w_i x_i$$

Figure 1.1 illustrates the idea.

Consider the following examples.

Example 1.1 (LT representation of OR)

A simple Boolean function is the conjunction, OR, of n variables:

$$OR(x_1, ..., x_n) = \begin{cases} 0 & \text{if } (x_1, ..., x_n) = (0, ..., 0) \\ 1 & \text{otherwise} \end{cases}$$

It can be implemented by a threshold gate, i.e., for all n, there exists a weight vector, $(w_0, ..., w_n)$, such that:

$$\forall x \in \{0,1\}^n, \quad OR(x_1, ..., x_n) = sgn(w_0 + \sum_{i=1}^{n} w_i x_i)$$

To implement OR one needs unit weights and a threshold, w_0, of -1.

$$w = (-1, 1, ..., 1)$$

x_1	x_2	$-1+x_1+x_2$	$sgn(-1+x_1+x_2)$	$OR(x_1,x_2)$
0	0	-1	0	0
0	1	0	1	1
1	0	0	1	1
0	0	1	1	1

Table 1.1: 2-variable conjunction, $OR(x_1,x_2) = sgn(-1+x_1+x_2)$

x_1	x_2	$-2+x_1+x_2$	$sgn(-2+x_1+x_2)$	$AND(x_1,x_2)$
0	0	-2	0	0
0	1	-1	0	0
1	0	-1	0	0
0	0	0	1	1

Table 1.2: 2-variable disjunction, $AND(x_1,x_2) = sgn(-2+x_1+x_2)$

$$OR(x_1,...,x_n) = sgn(-1+\sum_{i=1}^{n} x_i)$$

Table 1.1 shows the case of $n = 2$.

Example 1.2 $(AND \in LT)$

The disjunction, AND, is also a linear threshold function:

$$AND(x_1,...,x_n) = sgn(-n+\sum_{i=1}^{n} x_i)$$

Table 1.2 shows the case of $n = 2$.

The majority function, MAJ, is a Boolean function that outputs 1 if half or more than half of its input variables are 1s.

Example 1.3 $(MAJ \in LT)$

Here follows the definition of the majority function:

$$MAJ(x_1,...,x_n) = \begin{cases} 1 & \text{, if } \sum_{i=1}^{n} x_i \geq \left\lceil \frac{n}{2} \right\rceil \\ 0 & \text{, otherwise} \end{cases}$$

x_1	x_2	x_3	$-2+x_1+x_2+x_3$	$sgn(-2+x_1+x_2+x_3)$	$MAJ(x_1,x_2,x_3)$
0	0	0	-2	0	0
0	0	1	-1	0	0
0	1	0	-1	0	0
0	1	1	0	1	1
1	0	0	-1	0	0
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	1	1

Table 1.3: 3-variable majority, $MAJ(x_1,x_2,x_3) = sgn(-2+x_1+x_2+x_3)$

It is a natural candidate for a threshold function; one choice of weights is:

$$w = (-\left\lceil \frac{n}{2} \right\rceil, 1, ..., 1)$$

$$MAJ(x_1, ..., x_n) = sgn(-\left\lceil \frac{n}{2} \right\rceil + \sum_{i=1}^{n} x_i)$$

Table 1.3 shows the case of $n = 3$.

Examples 1.1, 1.2 and 1.3 show Boolean functions that are symmetric. Those are functions for which the output depends on the number of 1s in the input vector irrespective of their position. A well known symmetric function is parity, or XOR.

Example 1.4 $(XOR \notin LT)$
Here follows the definition of the n-variable parity function:

$$XOR(x_1, ..., x_n) = \begin{cases} 1 & \text{, if } \sum_{i=1}^{n} x_i \text{ is odd} \\ 0 & \text{, otherwise} \end{cases}$$

Let $n = 2$ and suppose that there exist some weights which implement XOR:

$$w = (w_0, w_1, w_2)$$

$$XOR(x_1, x_2) = sgn(w_0 + w_1 x_1 + w_2 x_2)$$

x_1	x_2	$-2+x_1+x_2+x_3$	$XOR(x_1,x_2)$	implies that	
0	0	w_0	0	$w_0 < 0$	(1)
0	1	w_0+w_2	1	$w_0+w_2 \geq 0$	(2)
1	0	w_0+w_1	1	$w_0+w_1 \geq 0$	(3)
1	1	w_0+w_1+w2	0	$w_0+w_1+w_2 < 0$	(4)

Table 1.4: 2-variable parity, $XOR(x_1,x_2) \neq sgn(w_0+w_1x_1+w_2x_2)$

Table 1.4 shows the values of $F(x)$ as x varies. The values of the function produce a system of inequalities in w_i which has no solution. Indeed equations $(1)+(4)$ produce $2w_0+w_1+w_2+w_3<0$, while equations $(2)+(3)$ produce $2w_0+w_1+w_2+w_3 \geq 0$. Therefore $XOR \notin LT$ for $n=2$. That is also the case for arbitrary n. Indeed, suppose that $XOR \in LT$ for some n:

$$XOR(x_1,...,x_n) = sgn(w_0+\sum_{i=1}^{n} w_ix_i)$$

Then,

$$XOR(x_1,x_2,0,...,0) = sgn(w_0+w_1x_1+w_2x_2)$$

But $XOR(x_1,x_2,0,...,0) = XOR(x_1,x_2)$ implying that for $n=2$, $XOR \in LT$, which is not true.

For all symmetric functions that are also linear threshold functions, $w_1 = w_2 = ... = w_n$. That is quite convenient, because the weights can then be set all to 1. What happens when the underlying Boolean function is not symmetric? How large are the weights? The following example shows an LT function that requires the weights to be different and therefore requires some of them to be large, in comparison with others. Actually, the weights grow exponentially with the number of input variables.

Example 1.5 $(COMP \in LT)$

The comparison function accepts two integers, X and Y, i.e., their binary represen-

tation, $(x_1, ..., x_n)$ and $(y_1, ..., y_n)$:

$$X = \sum_{i=1}^{n} 2^{i-1} x_i$$

$$Y = \sum_{i=1}^{n} 2^{i-1} y_i$$

and compares them:

$$COMP(x_1, ..., x_n, y_1, ..., y_n) = \begin{cases} 1 & \text{, if } X \geq Y \\ 0 & \text{, otherwise} \end{cases}$$

The LT implementation of $COMP$ is straightforward:

$$COMP(x_1, ..., x_n, y_1, ..., y_n) = sgn(\sum_{i=1}^{n} 2^{i-1} x_i - \sum_{i=1}^{n} 2^{i-1} y_i)$$

Which translates into the following weight vector:

$$w = (0, 1, 2, ..., 2^n, -1, -2, .., -2^n)$$

1.2 Weights of a linear threshold element

In the present section we introduce the main concepts related to the weights of LT elements. We show that different sets of weights can produce the same LT function and define *minimal weights*. The topics of chapters 2, 3 and 4 are introduced, that is restricting the weights, constructing functions with minimal weights and converting a single element with large weights, to a circuit of gates with small weights.

How does one estimate the efficiency of a computing element such as the linear threshold element? A single LT gate can implement a multitude of distinct Boolean functions. That is done by varying its weights. Indeed, to each choice of weights corresponds a function. While some different sets of weights produce the same Boolean function, in general two distinct choices for the weights result in two distinct functions. An LT element with n inputs can implement about 2^{n^2} distinct Boolean functions, as

we will see in Section 2.3.1. This additional power of LT, compared to AON, comes at the cost of added complexity. One may ask the following question: what is the information content of a linear threshold gate, in particular, how many bits does one need in order to store it?

Let us focus on the weights of a single LT element. Note that, given a function f, the weight vector w is not unique. Different weights implement the same function.

Example 1.6 (Reducing the weights)
The following function

$$f(x_1, ..., x_4) = sgn(2 - 4x_1 + 6x_2 - 2x_3 + 4x_4)$$

can be written as

$$f(x_1, ..., x_4) = sgn(1 - 2x_1 + 3x_2 - x_3 + 2x_4)$$

because $sgn(2a) = sgn(a)$ holds for any $a > 0$.

Consider the function

$$f(x_1, ..., x_5) = sgn(2 - 4x_1 + 6x_2 - 2x_3 + 4x_4 + x_5)$$

It does not depend on x_5 and can therefore be written as

$$f(x_1, ..., x_5) = sgn(1 - 2x_1 + 3x_2 - x_3 + 2x_4)$$

because $2 - 4x_1 + 6x_2 - 2x_3 + 4x_4$ is a multiple of 2 it is either ≤ -2 or ≥ 0. In both cases, adding x_5 cannot change its sign.

A similar idea applies to the following two examples:

$$f(x_1, ..., x_4) = sgn(-4 + x_1 + x_2 + x_3 + 4x_4) = x_4$$

$$f(x_1, ..., x_3) = sgn(3 + 2x_1 - x_2 - x_3) = 1$$

But, in general, finding smaller or minimal integer weights is a difficult problem

$$f(x_1, ..., x_3) = sgn(-1 + 2x_1 - 3x_2 + 4x_3)$$

To minimize the above weights, one needs to find the function they implement

$$f(x_1, x_2, x_3) = x_1\bar{x}_2 + x_3$$

and derive the smallest weights that implement that function

$$f(x_1, ..., x_3) = sgn(-1 + x_1 - x_2 + 2x_3)$$

Example 1.6 shows that different weight vectors can be used to implement the same LT function. Here follows a formal definition of that idea.

Definition 1.2 *(Weight Space)*
Given a linear threshold function f, we define $\mathcal{W}$ as the set of all weights that satisfy Definition 1.1, that is

$$\mathcal{W} = \{W \in R^n : \forall x \in \{0,1\}^n, sgn(w_0 + \sum_{i=1}^{n} w_i x_i) = f(x)\}$$

We want to study the weights; in particular, we are interested in the following orthogonal questions:

1. What happens when one restricts the weights to be only integers, or in general restricts the weights to an arbitrary subset of R?

2. Assuming the weights are integers, and given a measure of their size, how to find a minimal weight vector?

3. Suppose that for a given function, f, the minimal weights are large, i.e., grow exponentially with the number of inputs, can we implement that function with a two-layer LT circuit composed of gates with small weights?

In the early 1970's, it was shown that any LT function can be implemented with weights that are signed integers, [32]. That was done using a non-constructive proof, i.e. showing that there exists a set of integer weights, without a method of finding them. In Chapter 2 we ask the following questions. How one can restrict the weights to an arbitrary, given set of numbers, without affecting the power of the resulting element? What are the conditions on that set? Is there an efficient conversion algorithm?

Chapters 3 and 4 deal with the size of integer weights. We use the L_1 norm as a measure of the size of the weights.

Definition 1.3 *(Minimal Weight Size)*
We define the size of a weight vector as the sum of the absolute values of the weights. The minimal weight size of a linear threshold function is defined as

$$S[f] = \min_{w \in \mathcal{W}} (\sum_{i=0}^{n} |w_i|)$$

The particular vector that achieves the minimum is called a minimal weight vector.

Naturally, $S[f]$ is a function of n.

Many experimental results in the area of neural networks have indicated that the magnitudes of the coefficients in the linear threshold elements grow very fast with the size of the inputs and therefore limit the practical use of the network. One natural question to ask is the following. How limited is the computational power of the network if one limits oneself to threshold elements with only "small" growth in the size of the coefficients? It has been shown [17], [33], [38], [43] that there exists a function that can be implemented by a single threshold element with exponentially growing weights, but cannot be implemented by a threshold element with polynomially growing weights. In fact, that function is $COMP$, the comparison function presented in Example 1.5. In light of that result a subclass of LT was defined, the class of functions with small weights : $\widehat{LT}$ [43]. Large and small stand for exponentially growing in n, and polynomially growing in n, the number of inputs. How dense is $\widehat{LT}$, in other words, are there functions which can be implemented with weights that grow as

a polynomial of degree d, but cannot be realized with smaller weights? That is, can we divide $\widehat{LT}$ into classes of functions indexed by the degree of polynomial growth of their weights? Chapter 3 answers that question by introducing an algorithm for constructing LT functions with a given weight size.

How does one deal with LT functions that require large weights? Siu and Bruck ([43]) proved that $LT_d \subset \widehat{LT}_{2d+1}$. [13] improves the bound to $LT_d \subset \widehat{LT}_{d+1}$ by showing that $LT_1 \subset \widehat{LT}_2$ and generalizing to arbitrary depth. However the method is complicated and the proofs difficult to follow. [18], presents a simplified version of the results in [13]. It focuses on showing that $LT_1 \subset \widehat{LT}_2$. To some degree it supersedes the first by presenting a simpler, more intuitive construction. The idea is to use two operations in order to reduce the weights, divide them by powers of two and divide them modulo a prime. The resulting "small"-weight gates are connected into a circuit that produces the correct output if enough primes are used. One can further simplify the results presented in [13] and [18], by limiting oneself to the simulation of a particular large weight function : $COMP$. As a result one gets bounds on the number of gates in our circuit of the order of $O(n^4 \log n)$, a significant improvement on the general bound of $O(n^{12} \log^{11} n)$ in [18].

1.3 Multiple thresholds and VLSI implementation

In this section we outline the hardware implementation results described in Chapter 6 and present LTM a new class of functions related to LT, see Chapter 5.

Implementations of threshold circuits were proposed already in the 60's and 70's [4], [48], [53], and more recently in [28], [39]. To our knowledge, the theoretical results on threshold circuits have not been linked to any work involving silicon implementations. Programmable neuron-based hardware has been recently proposed [39], [41].

There are essentially two points at which implementations of LT circuits differ. The method used to compute the weighted sum, and the manner in which the weights are stored. We have chosen a sum of currents, each of them corresponding to a weighted input. Given that we use Boolean inputs, no exact multiplication is needed.

One needs only to make sure that a zero current is produced when the input is a logical 0, and the current is w_i for a logical 1. Such a "multiplication" can be done by a single transistor, the input pin being connected to its gate terminal. An advantage of this approach is that the storage of the weight and the scaling of the input are tied together into a single transistor.

But how exactly are the weights stored? There are two approaches to that problem: hardwired and programmable weights. Hardwired weights are defined at the moment the circuit is laid-out, and cannot be changed once it is fabricated. There are many interesting questions related to hardwired weights. Indeed, in most implementations, different weights correspond to different layouts. Such differences make the layout of LT circuits a difficult task, because different elements have different shapes. However, as we saw in the previous section, one can vary the weights of a given LT element without affecting the function it computes. That last fact can help in laying out the elements in such a way that they fit together nicely. Programmable weights, on the other hand, present no such difficulties. All LT elements look the same. There are many ways to implement programmable weights. One can store them in a digital RAM, or feed them in through input lines. In Chapter 6 we show two implementations of the LT element. One using hardwired weights, stored as the width to length ratio of a transistor, and one using programmable weights, placed as a non-volatile charge on a floating gate transistor. In the latter case, the values of the weights can be modified by tunneling or hot-electron injection.

When laying out LT circuits for common LT_2 functions such as parity, one may notice that the LT representation is redundant. While in general LT_2 circuits are composed of distinct threshold elements, in the case of some useful functions, such as parity, addition and multiplication, the gates of the first layer differ only by their thresholds. To take advantage of this fact we introduce, in Chapter 5, a new computing element that we call LTM, *linear threshold element with multiple thresholds.* It performs the following computation. The weighted sum of its Boolean inputs is compared to a set of thresholds, rather than a single threshold. Geometrically it can be viewed as a set of parallel hyperplanes used to dichotomize the hypercube. Here

follows the definition of LTM.

Definition 1.4 *(Linear Threshold Gate with Multiple Transitions – LTM)*
A function f is in LTM if there exists a set of weights $w_i \in Z$, $1 \leq i \leq n$ and a function $h : Z \longrightarrow \{0,1\}$ such that

$$f(X) = h(\sum_{i=1}^{n} w_i x_i) \text{ for all } X \in \{0,1\}^n$$

The only constraint on h is that it undergoes polynomialy many transitions as its input scans $[-\sum_{i=1}^{n} |w_i|, \sum_{i=1}^{n} |w_i|]$.

Notice that without the constraint on the number of transitions, an LTM gate is capable of computing any Boolean function. Indeed, given an arbitrary function f, set $w_i = 2^{i-1}$ and $h(\sum_1^n 2^{i-1} x_i) = f(x_1, ..., x_n)$.

The following example shows how to compute the parity function, XOR, with a single LTM element. In Example 1.4 we showed that a single LT element is not sufficient in order to compute XOR.

Example 1.7 $(XOR \in LTM)$
$XOR(X)$ outputs 1 if $|X|$, the number of 1's in X, is odd. Otherwise it outputs 0. To implement it choose $w_i = 1$ and $h(k) = \frac{1}{2}(1 - (-1)^k)$ for $0 \leq k \leq n$. Note that $h(k)$ needs not be defined for $k < 0$ and $k > n$, and has polynomialy many transitions.

Another useful function is the addition of two integers. It can be computed by a single layer of LTM elements, one per output bit.

The study of LTM was originally motivated by practical considerations, as mentioned above, in order to improve the area required for the layout of the parity function. However, within the theoretical framework of LT, this new computing element poses many challenging problems. It is easy to see that a single LTM element is more powerful than a single LT element. But how powerful is it in comparison with LT_2 or $\widehat{LT_2}$?

1.4 Contributions and organization of the thesis

Our contributions are at three levels:

- At the theoretical level, we define new classes of functions such as $\widehat{LT}^{(d)}$ and LTM and estimate their computational power.
- At the algorithmic level, we show how to convert real weights to weights drawn from an arbitrary subset of the real numbers, e.g., integer weights, we also show how to construct LT functions with minimal weights, and finally we present an algorithm that produces an $\widehat{LT}_2$ circuit (circuit composed of gates with small weights), that computes the comparison function, $COMP$. We also present LTM circuits computing useful functions, such as XOR, ADD, $PRODUCT$.
- At the implementation level, we show the design, layout and testing of the VLSI implementation of LT and LTM. We designed a programmable LT element that uses floating gate technology in order to store the values of the weights.

The thesis is organized as follows. In Chapter 2 we show some well known results in the theory of threshold circuits, in particular, that any linear threshold element can be implemented with integer weights. Our contribution is a generalization of that result, to an arbitrary set of real numbers. The conditions that allow any LT function to be implemented are derived, along with an algorithm for converting the weights. Chapter 3 presents a method for constructing linear threshold functions with minimal weights. It is used to establish the separation between the classes $\widehat{LT}^{(d)}$, indexed by d. Given an integer d, the class $\widehat{LT}^{(d)}$ is defined as the set of functions that can be implemented with weights of $O(n^d)$. In Chapter 4 a well known result is presented, i.e., the fact that a single LT element with large weights can be implemented by a two-layer circuit composed of $\widehat{LT}$ elements, that is, linear threshold elements with small weights. Our contribution is the explicit construction of those circuits in the case of the comparison function, $COMP$. Chapter 5 introduces LTM, or linear threshold element with multiple thresholds. It presents constructions for useful Boolean functions, such as XOR, ADD, $PRODUCT$, along with an estimation of the power of LTM relative

to LT and its derived classes, $\widehat{LT}$, $\widehat{LT_2}$ and LT_2. Finally, Chapter 6 describes the VLSI implementation of LT and LTM. Both hardwired and programmable solutions are presented. Weights are stored as the charge on a floating gate, and modified by tunneling and injection of electrons.

Chapter 2 Restricting the weights

2.1 Introduction

The present chapter focuses on questions related to the weights of a single LT element. Given n, the number of variables of an arbitrary threshold function, how many bits does one need to reserve in order to store its weights? This question has been answered in the early 1970s by Muroga, [32], by showing that any LT function can be implemented with integer weights, and providing a bound to their size. In this chapter we generalize this idea by answering questions such as:

- What happens if we restrict the magnitudes of the weights to be squares of integers rather than integers?
- What if we allow only powers of 2?
- In general, given D, a subset of the positive real numbers, $D \in R^+$, define $LT[D]$ to be the set of LT functions the weights of which have magnitudes drawn from D.

$$LT[D] = \{f : f(x_1, ..., x_n) = sgn(w_0 + \sum_{i=1}^{n} w_i x_i) \text{ where } |w_i| \in D\}$$

  What are the conditions on D under which $LT[D] = LT$ (i.e., weights drawn from D are sufficient in order to implement all LT functions)?

In Section 2.2 we present some motivation for this type of questions. Section 2.3 presents some related work along with proofs and examples. It addresses the following topics:

- What is the number of LT functions of n variables?

- What is the upper bound on the number of bits required to store the weights of an arbitrary threshold function?
- Is there a function that achieves that bound?
- How to convert arbitrary real weights to integers without changing the function they implement.

In Section 2.5 we present the main result: the conditions on the set D which guarantee that it can implement all LT functions.

2.2 Motivation

When dealing with threshold circuits, it is often the case that a particular weight value appears in many locations, either in the same gate or in different gates of the circuit. Given a system in which storing a weight value is expensive, one may want to store the value once, and link the corresponding weights to it rather than store the same value in many locations. This concept can be applied both to hardware and software implementations of threshold circuits.

The above approach is passive, in the sense that the weights are given, either resulting from a learning algorithm or simply pre-computed for a given function; only then does one eliminate duplicate weight values. One more step towards the goal of saving storage space is to modify certain weights without affecting the corresponding threshold function, in order to introduce more duplicate weights. In Section 1.2 we saw that this can be done, since different sets of weights can implement the same threshold function. In that context, one may ask the following question: given a set of real numbers, is it sufficient to represent the weights of all LT functions?

2.3 Preliminaries and related work

Different weight can implement the same LT function. One way to characterize a set of weights is by what we define as their *boundary*.

Definition 2.1 *(Weight Boundary)*

Let $(w_0, ..., w_n)$ be a set of weights, and f the function they implement. Their boundary is the pair (l, h) where:

$$l = \max_{x|f(x)=0} (w_0 + \sum_{i=1}^{n} w_i x_i)$$

$$h = \min_{x|f(x)=1} (w_0 + \sum_{i=1}^{n} w_i x_i)$$

Notice two obvious properties of the boundary (l, h):

- $l < 0$ and $h \geq 0$
- for all $x \in \{0,1\}^n$, $w_0 + \sum_{i=1}^{n} w_i x_i \notin]l, h[$

For the proofs presented below we will need weights with boundary $(-1, 1)$. Given an arbitrary set of weights, the following algorithm converts them to such a set:

Algorithm 2.1 *($(-1,1)$ Boundary)*

Given a set of weights, $(u_0, ..., u_n)$ with boundary (l, h) define:

$$w_0 = \frac{2}{h-l}\left(u_0 - \frac{h+l}{2}\right)$$

$$w_i = \frac{2u_i}{h-l} \text{ for all } i, \quad 1 \leq i \leq n$$

Let us show that the Algorithm 2.1 produces valid weights, i.e., they implement the same function as the original ones, and that their boundary is indeed $(-1, 1)$.

Lemma 2.1 *(Converting the weights)*

Let $(u_0, ..., u_n)$ be an arbitrary set of weights with boundary (h, l), and f the function they implement. The new weights, $(w_0, ..., w_n)$, obtained by Algorithm 2.1 have boundary $(-1, 1)$ and implement the function f.

Proof:

Let $f(x) = sgn(u_0 + \sum_{i=1}^{n} u_i x_i)$, and define g as the function implemented by the new weights $(w_0, ..., w_n)$, namely $g(x) = sgn(w_0 + \sum_{i=1}^{n} w_i x_i)$. We want to show that $g(x) = f(x)$ for all x. We will look at two cases:

- Let x be such that $f(x) = 0$. Then by the definition of the boundary (l, h) (Definition 2.1), and the fact that $h - l > 0$:

$$u_0 + \sum_{i=1}^{n} u_i x_i \leq l$$

$$u_0 - \frac{h+l}{2} + \sum_{i=1}^{n} u_i x_i \leq \frac{l-h}{2}$$

$$\frac{2}{h-l}\left(u_0 - \frac{h+l}{2} + \sum_{i=1}^{n} u_i x_i\right) \leq \frac{2}{h-l}\left(\frac{l-h}{2}\right)$$

$$w_0 + \sum_{i=1}^{n} w_i x_i \leq -1$$

$$g(x) = 0$$

- For x such that $f(x) = 1$.

$$u_0 + \sum_{i=1}^{n} u_i x_i \geq h$$

$$u_0 - \frac{h+l}{2} + \sum_{i=1}^{n} u_i x_i \leq \frac{h-l}{2}$$

$$\frac{2}{h-l}\left(u_0 - \frac{h+l}{2} + \sum_{i=1}^{n} u_i x_i\right) \leq \frac{2}{h-l}\left(\frac{h-l}{2}\right)$$

$$w_0 + \sum_{i=1}^{n} w_i x_i \leq 1$$

$$g(x) = 1$$

We have shown that $g \equiv f$, and that $|w_0 + \sum_{i=1}^{n} w_i x_i| \geq 1$, which becomes an equality at the points for which $u_0 + \sum_{i=1}^{n} u_i x_i$ equals h or l. □

Let us illustrate the above algorithm with an example.

Example 2.1 ($(-1, 1)$ Boundary)

Let us consider the following 2-variable LT function:

$$f(x_1, x_2) = sgn(-1.2 + 0.5x_1 + 1.1x_2)$$

The weight vector is $(-1.2, 0.5, 1.1)$. Let us compute its boundary, (l, h). The weighted sum assumes the following values

x_1	x_2	$-1.2 + 0.5x_1 + 1.1x_2$	$f(x_1, x_2)$
0	0	-1.2	0
0	1	-0.1	0
1	0	-0.7	0
1	1	0.4	1

Referring to Definition 2.1:

$$l = \max_{x|f(x)=0} (-1.2 + 0.5x_1 + 1.1x_2) = -0.1$$

$$h = \min_{x|f(x)=1} (-1.2 + 0.5x_1 + 1.1x_2) = 0.4$$

By Algorithm 2.1, the new weights are:

$$w_0 = \frac{2}{h-l}\left(u_0 - \frac{h+l}{2}\right) = \frac{2}{0.4-(-0.1)}\left(1.2 - \frac{0.4+(-0.1)}{2}\right) = 5.4$$

$$w_1 = \frac{2 \times 0.5}{0.4-(-0.1)} = 2.0$$

$$w_2 = \frac{2 \times 1.1}{0.4-(-0.1)} = 4.4$$

The new weight vector assumes the following values:

x_1	x_2	$-5.4 + 2.0x_1 + 4.4x_2$	$sgn(-5.4 + 2.0x_1 + 4.4x_2)$
0	0	-5.4	0
0	1	-1.0	0
1	0	-2.4	0
1	1	1.0	1

As expected, the new weights implement the same function, and their boundary is $(-1, 1)$.

Let us present a few well known problems related to the study of a single LT element.

2.3.1 There are $O(2^{n^2})$ n-variable LT functions

Given the number of variables, n, it is easy to verify that the total number of Boolean functions is 2^{2^n}. Indeed, a general Boolean function is uniquely specified by its truth table which is composed of 2^n binary entries. How many of those functions are actually threshold functions? This question was considered in the late 1950s by various authors. The following bound was derived in [36]:

$$|LT| < 2^{n^2}$$

where $|LT|$ stands for the number of n-variable threshold functions. Later it was discovered that the best upper bound on $|LT|$ was given by L. Schläfli, [42], in 1850:

$$|LT| < 2\sum_{i=0}^{n} \left(\begin{array}{c} 2^n - 1 \\ i \end{array} \right) = 2^{n^2 - n\log_2 n + O(n)}$$

The first lower bound on $|LT|$ was published in [54], however [46], which appeared in the same journal and presents a similar proof, has precedence by submission date. The lower bound is:

$$|LT| > 2^{\frac{n(n-1)}{2}}$$

It is only in 1989 that this bound was improved by Zuev, see [58], using results from [34] and [57], to:

$$|LT| > 2^{n^2 - \frac{10n^2}{\ln n} - O(n \ln n)}$$

Recently, the above bound was further improved, see [21] and [23], to:

$$|LT| > 2^{n^2 - n\log_2 n - O(n)}$$

And finally Irmatov proved in [22], that asymptotically:

$$|LT| > 2^{n^2 - n\log_2 n + O(n)}$$

thus closing the gap between upper and lower bounds. Since there are around 2^{n^2} n-variable threshold functions, one can use information theoretical arguments to show that the total number of bits needed to represent the weights should be at least n^2; if not, some functions could not be differentiated. Furthermore, knowing that different weight vectors can implement the same function, as we saw in Section 1.2, a good guess is that the LT representation is not optimal in terms of storage, i.e., one needs more than n^2 bits in order to represent the weights.

2.4 Real to integer weights

In this section we show that a function written with an arbitrary set of weights, i.e., real weights, can be written with integer weights. We present two arguments

- a non-constructive, existence based, argument, which provides a bound to the size of the weights,
- a constructive argument, which given a set of weights, converts them to integers, but does not provide a bound on their size.

2.4.1 Threshold functions require $O(n \log_2 n)$ bits per weight

As mentioned above, given the estimate on the number of threshold functions, a single weight would require at least $O(n)$ bits. Because the LT representation is sparse, the actual number is $O(n \log_2 n)$, a result shown by Muroga, [32], in the early 1970s. Notice that even though not optimal, the LT representation is quite dense, and that the difference between $O(n^2)$ and $O(n^2 \log_2 n)$ is rather small and is worth the computational advantage one gets by using a weighted sum followed by a threshold. In order to provide more insight on Muroga's bound, a proof is included below.

Theorem 2.1 *($O(n\log_2 n)$ bits per weight)*

For an arbitrary, n-variable LT function, the weights w_i satisfy:

$$|w_i| < O(n\log_2 n) \quad \forall 0 \leq i \leq n$$

Proof:

Let f be a threshold function, $f \in LT$. Let its weights $(u_0, ..., u_n)$ be unknown. From the truth table of f one derives the following system of 2^n linear inequalities.

$$\begin{cases} u_0 + u_1 x_1^{(1)} + ... + u_n x_n^{(1)} \geq 0 \\ \vdots \qquad\qquad\qquad\qquad\qquad\quad \vdots \\ u_0 + u_1 x_1^{(2^n)} + ... + u_n x_n^{(2^n)} < 0 \end{cases}$$

Where $\{x^{(k)}\}_{k=1}^{2^n}$ are the vertices of the hypercube and the direction of the inequality depends on the value of the function at the corresponding point (the above choice is arbitrary). Call (l, h) the boundary of the weights $(u_0, ..., u_n)$ and apply Algorithm 2.1 in order to get new weights, $(w_0, ..., w_n)$ with boundary $(-1, 1)$. The system of inequalities becomes:

$$\begin{cases} w_0 + w_1 x_1^{(1)} + ... + w_n x_n^{(1)} \geq 1 = 2f(x^{(1)}) - 1 \\ \vdots \qquad\qquad\qquad\qquad\qquad\quad \vdots \\ w_0 + w_1 x_1^{(2^n)} + ... + w_n x_n^{(2^n)} \leq -1 = 2f(x^{(n+1)}) - 1 \end{cases}$$

Using a well known result, in the theory of linear inequalities, [27], [26], we claim there exists a subset of $n+1$ out of those 2^n inequalities, such that if one replaces the inequality sign by an equality, the solution to the resulting system of equations solves the system of inequalities as well. Let $\{z^{(k)}\}_{k=1}^{n+1}$ be the set of points in $\{0,1\}^n$ corresponding to the selected $n+1$ inequalities. We get the following system of

equations:

$$\begin{cases} w_0 & + & w_1 z_1^{(1)} & + & \ldots & + & w_n z_n^{(1)} & = & 2f(z^{(1)}) - 1 \\ \vdots & & & & & & & & \vdots \\ w_0 & + & w_1 z_1^{(n+1)} & + & \ldots & + & w_n z_n^{(n+1)} & = & 2f(z^{(n+1)}) - 1 \end{cases}$$

where the right-hand side of the equations, as mentioned above, depends on the function f. The solution can be found using Cramer's method:

$$w_i = \frac{\Delta_i}{\Delta}$$

Where Δ is the determinant:

$$\Delta = \begin{vmatrix} 1 & z_1^{(1)} & \ldots & z_n^{(1)} \\ \vdots & & & \vdots \\ 1 & z_1^{(n+1)} & \ldots & z_n^{(n+1)} \end{vmatrix}$$

and Δ_i is the determinant with the i^{th} column replaced by the right-hand side of the system of equations:

$$\Delta = \begin{vmatrix} 1 & z_1^{(1)} & \ldots & z_{i-1}^{(1)} & f(z^{(1)}) - 1 & z_{i+1}^{(1)} & \ldots & z_n^{(1)} \\ \vdots & & & & & & & \vdots \\ 1 & z_1^{(n+1)} & \ldots & z_{i-1}^{(n+1)} & f(z^{(n+1)}) - 1 & z_{i+1}^{(n+1)} & \ldots & z_n^{(n+1)} \end{vmatrix}$$

Notice that the above matrices are composed of 1s, 0s and -1s. It is easy to see that the determinant of such a matrix satisfies the following bound:

$$\Delta^{(n \times n)} \leq B_n$$

where the bounding sequence B_n satisfies:

$$B_{n+1} \geq nB_n$$

which in turn implies that:

$$\Delta^{(n\times n)} \leq O(n!)$$

We are interested in the size of the integer weights, so let:

$$w_i' = |\Delta| w_i = \frac{|\Delta|}{\Delta}\Delta_i \leq O(n!) = O(2^{n\log_2 n})$$

implying that one needs $O(n\log_2 n)$ bits to store a single weight. □

2.4.2 Threshold functions require $\Theta(n\log_2 n)$ bits per weight

In Subsection 2.4.1 we proved that at $O(n\log n)$ bits are required to store the weights of an arbitrary LT function, thus establishing an upper bound on their size. As early as 1961, a function was found, [33], that requires weights of size $O(2^{\frac{n}{2}})$, i.e., $\Omega(n)$ bits per weight. Only recently, in [17], Hastad presented a function that requires $\Omega(n\log n)$ bits per weight, establishing that the above bound is tight, i.e., the size is $\Theta(n\log_2 n)$ bits. We saw in Subsection 2.3.1 that there are about 2^{n^2} LT functions, implying that one needs at least n bits of storage per weight. As mentioned, the LT representation is not optimal in terms of storage; it requires a factor of $\log n$ additional bits in order to store a threshold function. Intuitively, that makes sense since different weight vectors (e.g., multiples) can implement the same function. Nevertheless, the LT representation is quite compact, and the extra storage is worth the gain on gets, in terms of computation complexity. The spectral representation of LT functions is optimal in terms of storage. Indeed in [9] the author shows that the first $n+1$ spectral coefficients uniquely specify the function.

2.4.3 Converting real to integer weights: An algorithm

Before presenting the algorithm along with a proof of its validity, let us look at a few examples:

Example 2.2 (Real to integer weights)
Given the following function:

$$f_1(x_1, x_2) = sgn(-0.5 + 0.2x_1 + 0.3x_2)$$

an obvious way to get integer weights is to multiply them by a factor of 10, using the fact that $sgn(a) = sgn(10a)$, producing:

$$f_1(x_1, x_2) = sgn(-5 + 2x_1 + 3x_2)$$

Let:

$$f_2(x_1, x_2) = sgn(-2.35 + \pi x_1 - \sqrt{2}x_2)$$

In this case scaling does not work since there are irrational weights. One way of dealing with those weights is to use the floor function, $\lfloor\ \rfloor$, producing:

$$f_2(x_1, x_2) = sgn(-3 + 3x_1 - 2x_2)$$

In the case of the following function:

$$f_3(x_1, x_2) = sgn(-0.5 + 0.1x_1 + \frac{\sqrt{2}}{3}x_2)$$

Neither $\times 10$, nor $\lfloor\ \rfloor$ give the right answer. One needs to both multiply and take the floor in order to obtain the correct result, $\lfloor \times 10 \rfloor$:

$$f_3(x_1, x_2) = sgn(-5 + x_1 + 4x_2)$$

The above example shows the main idea behind the algorithm for converting real to integer weights: scale and take the floor of each weight. One needs to prove that for any weight vector there exists a scaling coefficient, large enough, so that the algorithm works.

Algorithm 2.2 *(Real to integer weights)*

Given a set of real weights $(u_0, ..., u_n)$

1. *apply Algorithm 2.1 to get new weights* $(v_0, ..., v_n)$*, with boundary* $(-1, 1)$
2. *set* $w_i = \lfloor (n+2)v_i \rfloor$

Lemma 2.2 *(Real to integer weights)*

The weights produced by Algorithm 2.2 implement the same function as the original weights. Namely,

$$f(x) = sgn(u_0 + \sum_{i=1}^{n} u_i x_i) = sgn(w_0 + \sum_{i=1}^{n} w_i x_i)$$

Proof:

Let $(u_0, ..., u_n)$ be a set of real weights. We apply Algorithm 2.1 obtaining new weights $(v_0, ..., v_n)$. To simplify notation let x denote the extended vector $(1, x_1, x_2, ..., x_n)$ so that $v \cdot x = v_0 + \sum_{i=1}^{n} v_i x_i$. According to Lemma 2.1, the new weights implement the same function and have boundary $(-1, 1)$, i.e.:

$$f(x) = sgn(u \cdot x) = sgn(v \cdot x)$$

$$|v \cdot x| \geq 1 \text{ for all } x \in \{0, 1\}^n$$

We multiply the above inequality by k,

$$|(kw) \cdot x| \geq k \text{ for all } x \in \{0, 1\}^n$$

Let $\lfloor kw \rfloor$ denote the vector $(\lfloor kw_0 \rfloor, ..., \lfloor kw_n \rfloor)$,

$$|(kw) \cdot x - \lfloor kw \rfloor \cdot x + \lfloor kw \rfloor \cdot x| \geq k$$

$$|(kw - \lfloor kw \rfloor) \cdot x + \lfloor kw \rfloor \cdot x| \geq k$$

By the triangle inequality,

$$|(kw - \lfloor kw \rfloor) \cdot x| + |\lfloor kw \rfloor \cdot x| \geq k$$

$$|\lfloor kw \rfloor \cdot x| \geq k - |(kw - \lfloor kw \rfloor) \cdot x|$$

$$|\lfloor kw \rfloor \cdot x| \geq k - \sqrt{n+1}||(kw - \lfloor kw \rfloor)||$$

$$|\lfloor kw \rfloor \cdot x| \geq k - \sqrt{n+1}||(1, ..., 1)||$$

$$|\lfloor kw \rfloor \cdot x| \geq k - \sqrt{n+1}\sqrt{n+1}$$

$$|\lfloor kw \rfloor \cdot x| \geq k - n - 1$$

At this point we set $k = n + 2$ and get:

$$|\lfloor (n+2)w \rfloor \cdot x| \geq 1$$

How does this imply that the new weights implement the original function f? Given an arbitrary input vector x one can repeat the above steps, starting with $u \cdot x \leq l$ for the case $f(x) = 0$, or $u \cdot x \geq h$ for the case $f(x) = 1$, and deriving $\lfloor (n+2)w \rfloor \cdot x \leq -1$ or $\lfloor (n+2)w \rfloor \cdot x \geq 1$ respectively. Using the absolute value in the proof above allows one to treat both cases simultaneously. □

Algorithm 2.2 is illustrated by the following example.

Example 2.3 (Using the algorithm)

Let us use the same 2-variable function we used in Example 2.1:

$$f(x_1, x_2) = sgn(-1.2 + 0.5x_1 + 1.1x_2)$$

We need to apply Algorithm 2.1, which was done in Example 2.1. The new weight vector v is:

$$f(x_1, x_2) = sgn(-5.4 + 2.0x_1 + 4.4x_2)$$

At this point we multiply the weights by $n + 2 = 4$ and take the floor:

$$f(x_1, x_2) = sgn(-22 + 8x_1 + 17x_2)$$

The new integer weights produce the following table:

x_1	x_2	$-22 + 8x_1 + 17x_2$	$sgn(-22 + 8x_1 + 17x_2)$
0	0	-22	0
0	1	-5	0
1	0	-14	0
1	1	3	1

As expected the function has not changed. Notice however that the weights obtained are quite large. In particular, the actual function under consideration is $AND(x_1, x_2)$; it can be implemented with much smaller eights:

$$f(x_1, x_2) = sgn(-2 + x_1 + x_2)$$

In Chapter 3 we address the problem of finding the smallest possible integer weights.

2.5 Converting the weights to an arbitrary set of numbers

We restrict the absolute value of the weights to a set D, $D \subset R$. We call D the *Weight Domain.*

Definition 2.2 *($LT(D)$ – The set of LT functions spawned by D)*
Given D, a subset of R, we define $LT(D)$ as the set of LT functions that can be implemented with weights, the absolute value of which is drawn exclusively from D.

Formally:

$$LT(D) = \{f \subseteq LT : \exists w \in R^{n+1}, \textit{ such that } f(x) = sgn(w_0 + \sum_{i=1}^{n} w_i x_i) \ \ \forall x \in \{0,1\}^n \textit{ and } |w_i| \in D \textit{ for } 0 \leq i \leq n\}$$

Our goal is to study the properties of D and their impact on $LT(D)$. Let us first narrow down the list of candidates for D by eliminating a few obvious cases.

- D is infinite.

  Indeed, if D was finite one can find an n, large enough, so that there exists an n-variable LT function that cannot be implemented with weights drawn from D simply because the number of distinct weights it requires is larger than the cardinality of D. Consider for example a function such as $COMPARISON$, for which half of the weights are distinct and let the number of variables $n > 2|D|$.

- D is countable.

  As mentioned in Section 2.3, $LT(N) = LT$, that is any LT function can be implemented using integer weights. Using a set of higher cardinality than N does not provide any extra functionality. Indeed, if the set contains an interval of the form $]0, \varepsilon[$, then the integer weights can be scaled down to fit in it implying that $LT(D) = LT$. On the other hand, suppose $D = [100, 101]$, the set of functions it spawns is very limited. It includes OR and a few closely related functions. In the general case of an uncountable set D, we will focus on the "best" countable subset of D as the set from which the weights will be drawn.

- D is strictly ordered.

  Since D is countable, it can be ordered. Furthermore, all elements of D must be distinct. It plays the role of an alphabet from which the values of the weights are drawn.

Since D is countable and ordered, it can be indexed:

$$D = \{d_i : i \in N\} \text{ and } d_i < d_{i+1} \quad \forall i$$

We define a "modified floor" function, $d(.)$ as follows:

$$\begin{aligned} d: \quad & R \longrightarrow R \\ d: \quad & x \longrightarrow d(x) = d_i \\ \text{where } i \text{ is such that} \quad & d_i \leq x < d_{i+1} \end{aligned}$$

Example 2.4 (Square and Exponential Weights)
We saw that one can convert real weights to integer ones. Is it possible to convert the weights to perfect squares, that is:

$$d_i = i^2 \text{ for all } i \in N$$

Consider the 5-variable AND, defined as:

$$AND(x_1, ..., x_5) = \begin{cases} 1 & \text{, if } x_1 = ... = x_5 = 1 \\ 0 & \text{, otherwise} \end{cases}$$

We saw that it is an LT function, i.e., can be written as

$$AND(x_1, ..., x_5) = sgn(-5 + x_1 + x_2 + x_3 + x_4 + x_5)$$

Can it be written with weights, the norm of which is a perfect square?

$$AND(x_1, ..., x_5) = sgn(-25 + 4x_1 + 4x_2 + 4x_3 + 4x_4 + 9x_5)$$

What about powers of 2:

$$d_i = 2^i \text{ for all } i \in N$$

$$AND(x_1, ..., x_5) = sgn(-8 + x_1 + x_2 + 2x_3 + 2x_4 + 2x_5)$$

It seems, from Example 2.4, that no matter what the progression of the d_i is, AND can be implemented with weights drawn from D. That is not true for arbitrary LT functions. The following theorem shows that if the d_i grow polynomially, $LT(D) = LT$, but if they are exponential some LT functions cannot be implemented. That is,

if D is $O(i^d)$ it can implement all LT functions, but if it is $\Omega(2^{\alpha n})$ it cannot. In fact, we show a slightly more general result, that is some super-polynomial growths are allowed as well – if d_i are $O(n^{\log n})$, $LT(D) = LT$.

Theorem 2.2 *(Restricting the weights)*
Let the weights be restricted to an ordered set $D = \{d_i, i \in N\}$, $D \subset R$. Then

1. *$LT(D) = LT$ if for any given large constant $C \in R^+$, there exists i_0, such that for any $i \geq i_0$:*

$$C(d_{i+1} - d_i) < d_i$$

2. *$LT(D) \subset LT$ if d_i is $\Omega(2^{\alpha n})$, that is*

$$\exists (K, \alpha, i_0) \text{ such that for all } i \geq i_0, \quad K2^{\alpha i} \leq d_i$$

Proof:
Let us first show part 1. Given the original weight vector, u, with boundary (l, h) we apply Algorithm 2.1 and obtain new weights, v, with boundary $(-1, 1)$:

$$|v \cdot x| \geq 1 \text{ for all } x \in \{0,1\}^n$$

We multiply the above inequality by k,

$$|(kw) \cdot x| \geq k \text{ for all } x \in \{0,1\}^n$$

$$|(kw) \cdot x - d(kw) \cdot x + d(kw) \cdot x| \geq k$$

Where $d(kw)$ denotes the vector $(d(kw_0), ..., d(kw_n))$, the function d, being the generalization of the floor function to the set D, as defined above.

$$|(kw - d(kw)) \cdot x + d(kw) \cdot x| \geq k$$

By the triangle inequality,

$$|(kw - d(kw)) \cdot x| + |d(kw) \cdot x| \geq k$$

$$|d(kw) \cdot x| \geq k - |(kw - d(kw)) \cdot x|$$

$$|d(kw) \cdot x| \geq k - \sum_{j=0}^{n} |kw_i - d(kw_i)|$$

$$|d(kw) \cdot x| \geq k - (n+1)(d_{i+1} - d_i)$$

where i is such that $d(k|w_{max}|) = d_i$, w_{max} being the weight that maximizes $|kw_i - d(kw_i)|$ According to the condition on D, there exists i_0 such that:

$$C(d_{i+1} - d_i) < d_i$$

for all $i \geq i_0$, and for any choice of C, let us choose $C = w_{max}(n+1)$, we get:

$$w_{max}(n+1)(d_{i+1} - d_i) < d_i$$

And since by definition of the function d, $d_i \leq kw_{max}$:

$$w_{max}(n+1)(d_{i+1} - d_i) < kw_{max}$$

$$k - (n+1)(d_{i+1} - d_i) > 0$$

Using the latter inequality produces:

$$|d(kw) \cdot x| > 0$$

which is enough to show that the new weight vector, $d(kw)$, implements the original function. Let us look at two examples before presenting the proof for part 2 of the theorem. □

Example 2.5 (Squares)

Let D be the set defined in Example 2.4 – the set of perfect squares.

$$D = \{1, 4, 9, 16, 25, ...\}$$

According to Theorem 2.2 all LT functions can be implemented because for any $C \in R^+$

$$C(d_{i+1} - d_i) = C(i^2 + 2i + 1 - i^2) = C(2i + 1) < i^2$$

for all $i \geq i_0$, where

$$i_0 = \left\lfloor 2C + \sqrt{C^2 + C} \right\rfloor + 1$$

Indeed, following the proof of Theorem 2.2:

$$|d(kw) \cdot x| \geq k - (n + 1)(d_{i+1} - d_i)$$

becomes

$$|d(kw) \cdot x| \geq k - (n + 1)(2i + 1)$$

$$|d(kw) \cdot x| \geq k - (n + 1)(2\sqrt{kw_{max}} + 1)$$

We choose

$$k = n + 2 + \sqrt{n^2 + 3n + 2}$$

and get

$$|d(kw) \cdot x| > 0$$

which shows that the new weights implement the original function.

Example 2.6 (Numerical example)

Example 2.7 (Powers of 2)

What happens when one tries to use the above proof for powers of 2:

$$D = \{1, 2, 4, 8, 16, ...\}$$

Using the notation of Theorem 2.2:

$$C(d_{i+1} - d_i) = C(2^{i+1} - 2^i) = C2^i = Cd_i$$

The condition:

$$\forall C \in R^+, \quad \exists i_0 / C(d_{i+1} - d_i) < d_i \text{ for all } i \geq i_0$$

cannot be satisfied. We cannot use the theorem. Where does the proof fail?

$$|(kw) \cdot x| \geq k \text{ for all } x \in \{0, 1\}^n$$

$$|(kw) \cdot x - d(kw) \cdot x + d(kw) \cdot x| \geq k$$

$$|(kw - d(kw)) \cdot x + d(kw) \cdot x| \geq k$$

$$|(kw - d(kw)) \cdot x| + |d(kw) \cdot x| \geq k$$

$$|d(kw) \cdot x| \geq k - |(kw - d(kw)) \cdot x|$$

$$|d(kw) \cdot x| \geq k - (n+1)(d_{i+1} - d_i)$$

At this point we use the fact that $d_{i+1} - d_i = d_i \leq k$

$$|d(kw) \cdot x| \geq k - (n+1)k$$

$$|d(kw) \cdot x| \geq -nk$$

which does not tell us anything. In other words, our proof does not work, but there may be another way to prove the result for powers of 2. To guarantee that is not the

case, we need a counter-example. Consider the 5-variable LT function with weight vector $(-5, 1, 1, 2, 3, 4)$:

$$f(x_1, x_2, x_3, x_4, x_5) = sgn(-5 + x_1 + x_2 + 2x_3 + 3x_4 + 4x_5)$$

Suppose we found weights to implement it with, that are powers of 2:

$$f(x_1, x_2, x_3, x_4, x_5) = sgn(-2^{a_0} + 2^{a_1}x_1 + 2^{a_2}x_2 + 2^{a_3}x_3 + 2^{a_4}x_4 + 2^{a_5}x_5)$$

where $a_i \in N$. Given the definition of f, we make the following observations with regards to the relation between the new weights:

$$\begin{aligned} f(11100) = 0 \text{ but } f(11010) = 1 &\implies 2^{a_3} < 2^{a_4} \\ f(10010) = 0 \text{ but } f(10001) = 1 &\implies 2^{a_4} < 2^{a_5} \\ f(00001) = 0 \text{ but } f(00110) = 1 &\implies 2^{a_5} < 2^{a_3} + 2^{a_4} \end{aligned}$$

It is easy to see that the above system of inequalities has no integer solutions.

Proof: (continued)

The proof for claim 2, in Theorem 2.2, follows the idea of Example 2.7 above. We want to find a function which cannot be implemented with weights of the form $K2^{\alpha i}$. The idea is to construct the function in such a way as to require a fine grain definition among the weight values, finer than the sequence $K2^{\alpha i}$ can provide. We choose a function for which any set of weights must satisfy

$$w_l < w_{l+1} < ... < w_{\lfloor l+\frac{1}{\alpha} \rfloor + 1} < ... < w_{\lfloor l+\frac{2}{\alpha} \rfloor + 1}$$

That guarantees the weights to be different. We also set the function so that

$$w_l + w_{\lfloor l+\frac{1}{\alpha} \rfloor} \geq w_{\lfloor l+\frac{2}{\alpha} \rfloor}$$

The first set of inequalities implies:

$$w_l < \frac{1}{2} w_{\lfloor l+\frac{1}{\alpha} \rfloor} < \frac{1}{4} w_{\lfloor l+\frac{2}{\alpha} \rfloor}$$

which toghether with the second provides the needed contradiction. This was done in Example 2.7 for the case $K = \alpha = 1$. How to construct such a function for arbitrary constants K and α? We use the tools developed in Chapter 5, in particular the following lemma.

Lemma 2.3 *(Constructing functions with dense weights)*
Given two constants K and α, there exists a function for which any set of weights must satisfy

$$w_l < w_{l+1} < ... < w_{\lfloor l+\frac{1}{\alpha} \rfloor+1} < ... < w_{\lfloor l+\frac{2}{\alpha} \rfloor+1}$$

and

$$w_l + w_{\lfloor l+\frac{1}{\alpha} \rfloor} \geq w_{\lfloor l+\frac{2}{\alpha} \rfloor}$$

The proof of Lemma 2.3 is given in Section 3.4.3. □

2.6 Conclusion

The main contribution of Chapter 2 is the generalization of a well known result, i.e., the fact that any LT function can be implemented with integer weights. We show that given an arbitrary subset of $D \subset R$, one can implement all LT functions with weights drawn from D, provided that it is dense enough. Roughly, D must have a polynomial, or super-polynomial growth, but if it grows exponentially, some functions can no longer be realized. Theorem 2.2 shows the exact conditions on D. The proofs are constructive, i.e., we show an algorithm for constructing the weights, or construct a counter example, in the case of a sparse D.

An interesting direction for further investigation is to relate the above results to linear decision lists. In [49], the authors introduce a two-layer LT circuit which

implements a linear decision list. The magintudes of the weights in the second layer gate are powers of 2.

Chapter 3 Minimal weights

3.1 Introduction

The present chapter focuses on the study of a single linear threshold gate with binary inputs and output as well as integer weights. Such a gate is mathematically described by a *linear threshold function.*

Definition 3.1 (Linear Threshold Function) *A linear threshold function of n variables is a Boolean function* $f : \{0,1\}^n \rightarrow \{0,1\}$ *that can be written, for any* $x \in \{0,1\}^n$ *and a fixed* $w \in R^{n+1}$*, as :*

$$f(x) = sgn(F(x)) = \begin{cases} 1 & , \text{ for } F(x) \geq 0 \\ 0 & , \text{ otherwise} \end{cases}$$

$$\text{where } F(x) = W \cdot (-1, x) = -w_0 + \sum_{i=1}^{n} w_i x_i$$

As we saw in Chapter 2 any LT function can be realized with integer weights. So in the rest of this chapter, we will assume without loss of generality that all weights are integers. Also, notice that a linear threshold function can be implemented as:

$$f : \{-1,1\}^n \rightarrow \{0,1\}$$

We will address both the $\{0,1\}$ and the $\{-1,1\}$ representations.

Note that, given a function f, the weight vector w is not unique (see Example 3.1 below).

Definition 3.2 (Weight Space) *Given a linear threshold function* f*, we define* $\mathcal{W}$ *as the set of all weights that satisfy Definition 3.1, that is*

$$\mathcal{W} = \{w \in Z^n : \forall x \in \{0,1\}^n, sgn(W \cdot (-1, x)) = f(x)\}$$

Here follows a measure of the size of the weights.

Definition 3.3 (Minimal Weight Size) *We define the size of a weight vector as the sum of the absolute values of the weights. The minimal weight size of a linear threshold function is defined as*

$$S[f] = \min_{w \in \mathcal{W}} (\sum_{i=0}^{n} |w_i|)$$

The particular vector that achieves the minimum is called a minimal weight vector.

Naturally, $S[f]$ is a function of n.

3.1.1 Motivation

Why do we care about the size of the weights in threshold circuits?

Threshold circuits have been shown to be surprisingly powerful. For example, integer division can be implemented by a polynomial-size threshold circuit of constant depth, [5] [44]. Also it is proved in [1] that any function in AC^0 can be computed by depth 3 majority circuits of quasi-polynomial size, in fact, it is true for all of ACC^0 [56]. Given the foregoing impressive upper bounds, it is not surprising that we face difficulties in obtaining lower bounds. In fact, the best general lower bound for threshold circuits is the result that the Inner-Product-Mod-2 (IP2) requires exponential size for depth 2 [14]. However, this lower bound assumes that the circuits involve small weights, and it is not known whether IP2 can be computed by a depth-2 polynomial size threshold circuit with arbitrary weights. Namely, obtaining progress in lower bounds for threshold circuits seems to be related to the understanding of the role of large weights.

Hence, it is natural to ask how limited is the computational power of the circuit if one limits oneself to threshold elements with only "small" growth in the size of the coefficients? It has been shown [17], [33], [38], [43] that there exist linear threshold functions that can be implemented by a single threshold element with exponentially growing weights, $S[f] \sim 2^n$, but cannot be implemented by a threshold element

with smaller: polynomialy growing weights, $S[f] \sim n^d$, d constant. In light of that result the above question was dealt with by defining a class within the set of linear threshold functions: the class of functions with "small" (i.e., polynomialy growing) weights [43]. Most of the recent research focused on the power of circuits with small weights, relative to circuits with arbitrary weights [12], [13]. In particular, it showed that increasing the depth of the circuit by one is sufficient to reduce all the weights to be of polynomial size. However, these impressive upper bounds still were not helpful in improving the lower bounds.

In this chapter we take a different approach. Rather than dealing with circuits we focus on the modest task of studying a single threshold gate. The main contribution of the present chapter is to further refine the division of small versus arbitrary weights. We separate the set of functions with small weights into classes indexed by d, the degree of polynomial growth, and show that all of them are non-empty. In particular, we develop a technique for proving that a weight vector is minimal. We use that technique to construct a function of size $S[f] = s$ for an arbitrary s. The natural future direction is to extend our techniques for constructing minimal weight threshold functions to circuits of depth 2. This might help in defining explicit functions that cannot be computed by depth-2, polynomial size threshold circuits with specific weight size.

3.1.2 Organization

Here follows a brief outline of the rest of the chapter. In Section 3.2 we show some of the difficulties one faces when minimizing the weights as well as how the latter are affected by the choice of input domain. In Section 3.3 we consider functions defined over $\{-1, 1\}$. We limit ourselves to functions with no threshold (generalized majority function) and we show how to construct such functions with minimal weights. In Section 3.4 we present another way of constructing minimal functions that allows us to deal with any threshold function defined over $\{0, 1\}$.

3.2 Preliminaries and examples

In this section we illustrate some of the difficulties one faces when trying to minimize the weights of a threshold function. We also show how the input domain (i.e., $\{0,1\}$ versus $\{-1,1\}$) affects the size of the weights. See [25] for related results.

3.2.1 Minimizing the weights

The main difficulty in analyzing the size of the weights of a threshold element is due to the fact that a single linear threshold function can be implemented by different sets of weights as shown in the following example.

Example 3.1 (A Threshold Function with Minimal Weights) *Let us consider the following two sets of weights (weight vectors).*

$$w = (4 \;\; 1 \;\; 2 \;\; 5),\; F_1(x) = -4 + x_1 + 2x_2 + 5x_3$$

$$w' = (8 \;\; 2 \;\; 4 \;\; 10),\; F_2(x) = -6 + 2x_1 + 4x_2 + 10x_3$$

They both implement the same threshold function

$$f(x) = sgn(F_2(x)) = sgn(2F_1(x)) = sgn(F_1(x))$$

A closer look reveals that $f(X) = sgn(-1 + x_3)$*, implying that none of the above weight vectors has minimal size. Indeed, the minimal one is* $w'' = (1 \;\; 0 \;\; 0 \;\; 1)$ *and* $S[f] = 2$.

To determine if a given set of weights is minimal is in general a difficult problem, [3], [52]. Our technique consists of constructing weight vectors whose minimality is easily established. We then show how to modify them, while keeping them minimal, in order to get to a larger set of functions.

3.2.2 $\{0,1\}$ versus $\{-1,1\}$

Suppose we implement the same function over $\{0,1\}$ and over $\{-1,1\}$. How are the weights affected? Let us look at an example.

Example 3.2 (The OR function)

1. *Let $x_i \in \{0,1\}$,*

$$OR(x_1,...,x_n) = sgn(-1 + x_1 + ...x_n)$$

The size of the weights is $S = n+1$. Those weights are minimal.

Proof: *The weights are integers. Reducing their size implies reseting one or more of them to 0, which will violate the definition of OR.* □

2. *Now, let $x_i \in \{-1,1\}$,*

$$OR(x_1,...,x_n) = sgn(n - 2 + x_1 + ... + x_n)$$

The size of the weights is $S = 2n-2$. Those weights are minimal as well.

Proof: *Any weights that implement OR have to be positive. Suppose there exist weights of size $S' < 2n-2$. No weight can be 0, so $\sum_1^n w' \geq n$, implying that the threshold $-w_0 < (2n-2) - n = n-2$. Let w_i' be the smallest weight. Set $x_i = 1$ and all other inputs to -1. $\sum_1^n w' < -w_i(n-2)$ so that $F(X) < 0$ violating the definition of OR.* □

It appears from this example that the $\{0,1\}$ implementation has smaller weight size than the $\{-1,1\}$ representation. Is that true in general?

Example 3.3 (The Majority (MAJ) function) *Let the number of variables, n, be odd. The majority function outputs true if more than half of its inputs are true.*

- *Let $x_i \in \{0,1\}$,*

$$MAJ(x_1,...,x_n) = sgn(-\frac{n+1}{2} + x_1 + ... + x_n)$$

The size of the weights is $S = \frac{3n+1}{2}$. *We show they are minimal by a proof similar to case 2, above.*

- *Now, let* $x_i \in \{-1, 1\}$,

$$MAJ(x_1, ..., x_n) = sgn(x_1 + ...x_n)$$

Those weights are minimal since reducing them would imply reseting one or more of them to 0, which will violate the definition of MAJ. *The size of the weights is* $S = n$.

This second example shows that in general we cannot tell which implementation $\{0, 1\}$ or $\{-1, 1\}$ will produce a function with smaller weights.

3.3 Generalized majority function over $\{-1, 1\}$

In this section we study the following model:

$$f : \{-1, 1\} \to \{0, 1\}$$

$$f(X) = sgn(\sum_{1}^{n} w_i x_i)$$

Notice that there is no threshold; we are looking at a majority function with arbitrary weights. We address the problem of constructing functions with minimal weights. In particular, our goal is that for a given number of inputs n and size S, we find a function.

3.3.1 Mathematical setting

We are interested in constructing functions for which the minimal weight is easily determined. Finding the minimal weight involves a search; we are therefore interested in finding functions with a constrained weight spaces. The following tools allows us

to put constraints on $\vec{w}$. (In the remainder of this section we will explicitly denote vectors, in order to avoid confusion.)

Definition 3.4 (Root Space of a Boolean Function) *A vector $\vec{v} \in \{-1,1\}^n$ such that $f(\vec{v}) = f(-\vec{v})$ is called a root of f. We define the root space, R, as the set of all roots of f.*

Definition 3.5 (Root Generator Matrix) *For a given weight vector $\vec{w} \in W$ and a root $\vec{v} \in R$, the root generator matrix, $G = (g_{ij})$, is a $(n \times k)$-matrix, with entries in $\{-1, 0, 1\}$, whose rows $\vec{g}$ are orthogonal to $\vec{w}$ and equal to $\vec{v}$ at all non-zero coordinates, namely,*

1. *$G\vec{w} = \vec{0}$*

2. *$g_{ij} = 0$ or $g_{ij} = v_j$ for all i and j.*

Example 3.4 (Root Generator Matrix) *Suppose that we are given a linear threshold function specified by a weight vector $\vec{w} = (1,1,2,4,1,1,2,4)$. By inspection we determine one root $\vec{v} = (1,1,1,1,-1,-1,-1,-1)$. Notice that $w_1 + w_2 - w_7 = 0$ which can be written as $\vec{g} \cdot \vec{w} = 0$, where $\vec{g} = (1,1,0,0,0,0,-1,0)$ is a row of G. Set $\vec{r} = \vec{v} - 2\vec{g}$. Since $\vec{g}$ is equal to $\vec{v}$ at all non-zero coordinates, $\vec{r} \in \{-1,1\}^n$. Also $\vec{r}\cdot\vec{w} = \vec{v}\cdot\vec{w} + \vec{g}\cdot\vec{w} = 0$. We have generated a new root: $\vec{r} = (-1,-1,1,1,-1,-1,1,-1)$.*

Lemma 3.1 (Orthogonality of G and W) *For a given weight vector $\vec{w} \in W$ and a root $\vec{v} \in R$, $\vec{u}G^T = \vec{0}$ holds for any weight vector $\vec{u} \in W$.*

Proof: For an arbitrary $\vec{u} \in W$ and an arbitrary row, $\vec{g}_i$, of G, let $\vec{v}' = \vec{v} - 2\vec{g}_i$. By definition of $\vec{g}_i$, $\vec{v}' \in \{-1,1\}^n$ and $\vec{v}' \cdot \vec{w} = 0$. That implies $f(\vec{v}') = f(-\vec{v}')$: $\vec{v}'$ is a root of f. For any weight vector $\vec{u} \in W$, $sgn(\vec{u} \cdot \vec{v}') = sgn(-\vec{u} \cdot \vec{v}')$. Therefore, $\vec{u} \cdot (\vec{v} - 2\vec{g}_i) = 0$ and finally, since $\vec{v} \cdot \vec{u} = 0$, we get $\vec{u} \cdot \vec{g}_i = 0$. □

Lemma 3.2 (Minimality) *For a given weight vector $\vec{w} \in W$ and a root $\vec{v} \in R$ if $rank(G) = n - 1$ (i.e., G has $n-1$ independent rows) and $|w_i| = 1$ for some i, then $\vec{w}$ is the minimal weight vector.*

Proof: From Lemma 3.1 any weight vector $\vec{u}$ satisfies $\vec{u}G^T = \vec{0}$. $rank(G) = n - 1$ implies that $dim(W) = 1$, i.e., all possible weight vectors are integer multiples of each other. Since $|w_i| = 1$, all vectors are of the form $\vec{u} = k\vec{w}$, for $k \geq 1$. Therefore, $\vec{w}$ has the smallest size. □

We complete Example 3.4 with an application of Lemma 3.2.

Example 3.5 (Minimality) *Given :*

$$\vec{w} = (1, 1, 2, 4, 1, 1, 2, 4)$$

$$\vec{v} = (1, 1, 1, 1, -1, -1, -1, -1)$$

we can construct:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

It is easy to verify that $rank(G) = n - 1 = 7$ *and therefore, by Lemma 3.2,* $\vec{w}$ *is minimal and* $S[f] = 16$.

3.3.2 Weight vectors

In Example 3.5 we saw how, given a weight vector, one can show that it is minimal. In this section we present an example of a linear threshold function with minimal weight size, with an arbitrary number of input variables.

We would like to construct a weight vector and show that it is minimal. Let the number of inputs, n, be even. Let $\vec{w}$ consist of two identical blocks :

$$(w_1, w_2, ..., w_{n/2}, w_1, w_2, ..., w_{n/2})$$

Clearly, $\vec{v} = (1, 1, ..., 1, -1, -1, ..., -1)$ is a root and G is the corresponding generator matrix.

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & ... & 0 & 0 & 0 & -1 & 0 & 0 & 0 & ... & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & ... & 0 & 0 & 0 & 0 & -1 & 0 & 0 & ... & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & ... & 0 & 0 & 0 & 0 & 0 & -1 & 0 & ... & 0 & 0 & 0 \\ \vdots & & & & & & & & & & & & & & & \vdots \\ 0 & 0 & 0 & 0 & ... & 0 & 1 & 0 & 0 & 0 & 0 & 0 & ... & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & ... & 0 & 0 & 1 & 0 & 0 & 0 & 0 & ... & 0 & 0 & -1 \end{pmatrix}$$

3.3.3 Construction

The following theorem states that given an integer s and a number of variables n, there exists a function of n variables and minimal weight size s.

Theorem 3.1 (Main Result) *For any pair (s, n) that satisfies*

1. $n \leq s \leq \begin{cases} 2^{\frac{n}{2}} & \text{, for } n \text{ even} \\ 2^{\frac{n-1}{2}} + 2^{\frac{n-3}{2}} & \text{, for } n \text{ odd} \end{cases}$
2. *s even*

there exists a linear threshold function of n variables, f, with minimal weight size $S[f] = s$.

Proof: Given a pair (s, n), that satisfies the above conditions, we first construct a weight vector $\vec{w}$ that satisfies $\sum_{i=1}^{n} |w_i| = s$, then show that it is the minimal weight vector of the function $f(x) = sgn(\vec{w} \cdot \vec{x})$. The proof is shown only for n even.

CONSTRUCTION.

1. Define $(a_1, a_2, ..., a_{n/2}) = (1, 1, ..., 1)$.
2. If $\sum_{i=1}^{n/2} a_i < s/2$, then increase by one the smallest a_i such that $a_i < 2^{i-2}$. (In the case of a tie, take the w_i with smallest index i).
3. Repeat the previous step until $\sum_{i=1}^{n/2} a_i = s/2$ or $(a_1, a_2, ..., a_N) = (1, 1, 2, 4, ..., 2^{\frac{n}{2}-2})$.

4. Set $\vec{w} = (a_1, a_2, ..., a_{n/2}, a_1, a_2, ..., a_{n/2}.)$

Because we increase the size by one unit at a time, the algorithm will converge to the desired result for any integer s that satisfies $n \leq s \leq 2^{\frac{n}{2}}$. We have a construction for any valid (s, n) pair. Let us show that $\vec{w}$ is minimal.

MINIMALITY. Given that $\vec{w} = (a_1, a_2, ..., a_{n/2}, a_1, a_2, ..., a_{a/2})$ we find a root $\vec{v} = (1, 1, ..., 1, -1, -1, ..., -1)$ and $n/2$ rows of the generator matrix G corresponding to the equations $w_i = w_{i+\frac{n}{2}}$. To form additional rows note that the first k a_i's are powers of two (where k depends on s and n). Those can be written as $a_i = \sum_{j=1}^{i-1} a_j$ and generate $k - 1$ rows. And finally note that all other a_i, $i > k$, are smaller than 2^{k+1}. Hence, they can be written as a binary expansion $a_i = \sum_{j=1}^{k} \alpha_{ij} a_j$ where $\alpha_{ij} \in \{0, 1\}$. There are $\frac{n}{2} - k$ such weights. G has a total of $n - 1$ independent rows. $rank(G) = n - 1$ and $w_1 = 1$; therefore, by Lemma 3.2, $\vec{w}$ is minimal and $S[f] = s$. □

Example 3.6 (A Function of 10 variables and size 26) *We start with*

$$\vec{a} = (1, 1, 1, 1, 1)$$

We iterate:

$$(1, 1, 2, 1, 1)$$

$$(1, 1, 2, 2, 1)$$

$$(1, 1, 2, 2, 2)$$

$$(1, 1, 2, 3, 2)$$

$$(1, 1, 2, 3, 3)$$

$$(1, 1, 2, 4, 3)$$

$$(1, 1, 2, 4, 4)$$

and finally the algorithm converges to

$$\vec{a} = (1, 1, 2, 4, 5)$$

We claim that

$$\vec{w} = (\vec{a}, \vec{a}) = (1, 1, 2, 4, 5, 1, 1, 2, 4, 5)$$

is minimal. Indeed, $\vec{v} = (1, 1, 1, 1, 1, -1, -1, -1, -1, -1)$ *and*

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

is a matrix of rank 9.

Example 3.7 (Functions with Polynomial Size) *This example shows an application of Theorem 3.1. We define* $\widehat{LT}^{(d)}$ *as the set of linear threshold functions for which* $S[f] \leq n^d$*. The Theorem states that for any even* n *there exists a function* f *of* n *variables and minimum weight* $S[f] = n^d$*. The implication is that for all* d*,* $\widehat{LT}^{(d-1)}$ *is a proper subset of* $\widehat{LT}^{(d)}$*.*

3.4 Arbitrary threshold function over $\{0, 1\}$

In this section we present a different technique for constructing threshold functions with minimal weights. It allows us to construct functions with any weight size and number of variables. We consider functions with input domain $\{0, 1\}$, but as men-

tioned below, the argument holds for an arbitrary input space $\{a, b\}$. In the rest of this section we will use capital letters to denote vectors, in order to avoid confusion and differentiate with the previous section.

3.4.1 Approach

The method we use is based on a result from [52]. We assume, without loss of generality, that the weights are strictly positive integers. Our goal is to minimize $S = \sum_0^n |w_i| = \sum_0^n w_i$. We know from [32] that any other weights, U, implementing the same function have to be strictly positive. We will show that under certain conditions on W, $\sum_0^n w_i \geq \sum_0^n u_i$ for any U.

Consider input vectors X and Y for which the following equations hold:

$$F(X) = -w_0 + \sum_1^n w_i x_i = 0 \qquad\qquad F(Y) = -w_0 + \sum_1^n w_i y_i = -1$$

Let them define the rows of a matrix that we call A:

$$A = \begin{pmatrix} -1 & X^{(1)} \\ -1 & X^{(2)} \\ \vdots & \vdots \\ -1 & X^{(p)} \\ 1 & -Y^{(1)} \\ 1 & -Y^{(2)} \\ \vdots & \vdots \\ 1 & -Y^{(q)} \end{pmatrix} = \begin{pmatrix} -1 & x_1^{(1)} & x_2^{(1)} & ... & x_n^{(1)} \\ -1 & x_1^{(2)} & x_2^{(2)} & ... & x_n^{(2)} \\ \vdots & & & & \vdots \\ -1 & x_1^{(p)} & x_2^{(p)} & ... & x_n^{(p)} \\ 1 & -y_1^{(1)} & -y_2^{(1)} & ... & -y_n^{(1)} \\ 1 & -y_1^{(2)} & -y_2^{(2)} & ... & -y_n^{(2)} \\ \vdots & & & & \vdots \\ 1 & -y_1^{(q)} & -y_2^{(q)} & ... & -y_n^{(q)} \end{pmatrix}$$

We allow repetition of rows: we may have $X^{(i)} = X^{(j)} = ... = X^{(k)}$.

Example 3.8 (The matrix A) *Suppose we are given the following weights:*

$$W = (13 \;\; 6 \;\; 6 \;\; 3 \;\; 3 \;\; 2 \;\; 2 \;\; 1 \;\; 1)$$

Our goal is to show they are minimal. We need to first construct the matrix A. Here

follows a candidate:

$$A = \begin{pmatrix} -1 & X^{(1)} \\ -1 & X^{(2)} \\ 1 & -Y^{(1)} \\ 1 & -Y^{(2)} \end{pmatrix} = \begin{pmatrix} -1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ -1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & -1 \\ 1 & -1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 \end{pmatrix}$$

There are many possible choices for A. The one shown above is not a good one as we will see. □

Theorem 3.2 (Condition for Minimality) *Given a weight vector W, we construct A as described above. If there exists $a > 0$, such that A satisfies:*

$$(1 \ldots 1)A = (a \ldots a)$$

the weight vector W is minimal.

Proof: By definition of the X's and the Y's, the matrix A satisfies:

$$A \cdot (w_0\ w_1\ w_2 \ldots w_n)^T = (\overbrace{0\ 0 \ldots 0\ 0}^{p}\ \overbrace{1\ 1 \ldots 1\ 1}^{q})^T \tag{3.1}$$

Because $sgn(0) = 1$ and $sgn(-1) = 0$ any other weight vector, U, implementing the same function has to verify the above equalities with "$\geq$" instead of "=":

$$A \cdot (u_0\ u_1\ u_2 \ldots u_n)^T \geq (\overbrace{0\ 0 \ldots 0\ 0}^{p}\ \overbrace{1\ 1 \ldots 1\ 1}^{q})^T \tag{3.2}$$

Let $V = U - W$, and subtract Equations (3.1) from Inequalities (3.2), we get:

$$A \cdot (v_0\ v_1\ v_2 \ldots v_n)^T \geq (\overbrace{0\ 0 \ldots 0\ 0}^{p+q})^T \tag{3.3}$$

Now suppose A is such that:

$$(\overbrace{1\ 1 \ldots 1\ 1}^{p+q}) \cdot A = (\overbrace{a\ a \ldots a\ a}^{n}) \tag{3.4}$$

where a is a strictly positive integer. We multiply Inequalities (3.3) by the all 1 vector from the left and get:

$$(\overbrace{1\ 1\ ...\ 1\ 1}^{p+q}) \cdot A \cdot (v_0\ v_1\ v_2\ ...\ v_n)^T \geq (\overbrace{1\ 1\ ...\ 1\ 1}^{p+q}) \cdot (\overbrace{0\ 0\ ...\ 0\ 0}^{p+q})^T$$

$$(\overbrace{a\ a\ ...\ a\ a}^{n}) \cdot (v_0\ v_1\ v_2\ ...\ v_n)^T \geq 0$$

$$a \sum_{0}^{n} v_i \geq 0$$

And since $a > 0$, $w_i \geq 0$, $u_i \geq 0$ for all $i = 0, ..., n$, we know that: $\sum_0^n u_i \geq \sum_0^n w_i$ □

Notice that nowhere in the proof did we use the fact that the input domain is $\{0, 1\}$. Indeed, the above proof is valid for any input domain $\{a, b\}$. As you can see the proof relies on constructing A so that Equation (3.4) holds. To construct A we need appropriate X's and Y's which in turn depend on the choice W.

3.4.2 Basic construction

In this section we introduce W, the weight vector for the general construction, and prove it is minimal by finding an appropriate matrix A. Let the threshold, w_0, be arbitrary. We choose $w_1 = \lfloor \frac{w_0}{2} \rfloor$, $w_3 = \lfloor \frac{w_0 - w_1}{2} \rfloor$, $w_5 = \lfloor \frac{w_0 - w_1 - w_3}{2} \rfloor$, ..., $w_{n-1} = 1$, and $w_{2i} = w_{2i-1}$ for $i = 1...n$. We choose n so that $\sum_{i=1}^{n} w_{2i-1} = w_0 - 1$. Let us look at an example:

Example 3.9 ($w_0 = 13$) *Applying the above recursive definition, we get the weight vector of Example 3.8:* $W = (13\ \ 6\ \ 6\ \ 3\ \ 3\ \ 2\ \ 2\ \ 1\ \ 1)$ *Here follow the X and Y-type*

rows for A.

$$\left\{\begin{array}{ccccccccc} -1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ -1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right\} \quad sumX_1 = (-2\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 2\ \ 2)$$

$$\left\{\begin{array}{ccccccccc} -1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \end{array}\right\} \quad sumX_2 = (-2\ \ 1\ \ 1\ \ 1\ \ 1\ \ 2\ \ 2\ \ 0\ \ 0)$$

$$\left\{\begin{array}{ccccccccc} -1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ -1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{array}\right\} \quad sumX_3 = (-2\ \ 1\ \ 1\ \ 2\ \ 2\ \ 0\ \ 0\ \ 1\ \ 1)$$

$$\left\{\begin{array}{ccccccccc} -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right\} \quad sumX_4 = (-2\ \ 2\ \ 2\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1)$$

$$\underbrace{\begin{array}{ccccccccc} 1 & -1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & -1 \end{array}}_{sumY_1=(2\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ -1)}$$

We replicate rows and add them in order to get to the all 1 vector. Only odd numbered columns are shown.

$$\left(\begin{array}{ccccc} -2 & 1 & 1 & 1 & 2 \\ -2 & 1 & 1 & 2 & 0 \\ -2 & 1 & 2 & 0 & 1 \\ -2 & 2 & 0 & 0 & 1 \\ -2 & -1 & -1 & -1 & -1 \end{array}\right) \quad \left(\begin{array}{ccccc} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 1 & -1 & -1 & 0 \\ -2 & -1 & -1 & -1 & -1 \end{array}\right) \quad \left(\begin{array}{ccccc} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 2 & -1 & -1 & -1 & -1 \end{array}\right)$$

The latter of which add up to the all 2 vector. □

Theorem 3.3 (Minimality of the Construction) *For any w_0 we can construct a threshold function with minimal weights of size $S = 3 * w_0 - 2$ and number of variables $n = \lceil \log_2 S \rceil$.*

Proof: We are going to construct A, show that it satisfies $\underline{1}A = a\underline{1}$, and apply Theorem 3.2. Only two Y-type vectors are needed for the construction of A:

$$\begin{pmatrix} 1 & -1 & 0 & -1 & 0 & ... & -1 & 0 \\ 1 & 0 & -1 & 0 & -1 & ... & 0 & -1 \end{pmatrix}$$

They add up to $(2 \ -1 \ ... \ -1)$. The X-type vectors, summed two by two, add up to two possible forms:

$$-2 \ \ 1 \ \ ... \ \ 1 \ \ 2 \ \ 0 \ \ ... \ \ 0 \ \ 0$$

or

$$-2 \ \ 1 \ \ ... \ \ 1 \ \ 2 \ \ 0 \ \ ... \ \ 0 \ \ 1$$

By repeating and adding those partial sums one can get to the all 1 vector. How do we do that? We produce the $(0, ..., 0, 1)$ vector by adding two Y and two X-type vectors.

$$\begin{pmatrix} 2 & -1 & ... & -1 & -1 \\ -2 & 1 & ... & 1 & 2 \end{pmatrix}$$

Let us denote by S_i, $i = 1..n$, the singleton vector $(0, ...0, 1, 0, ..., 0)$, where the 1 is in the i^{th} position. We use induction to show that we can get to all S_i by adding up X and Y-type vectors. Indeed, suppose we have obtained all S_j for $j = 1, ..., i-1$. We can produce S_i by adding two X and two Y-type vectors:

$$\begin{pmatrix} 2 & -1 & -1 & -1 & -1 & .. & -1 & -1 & -1 & -1 \\ -2 & 1 & .. & 1 & 2 & 0 & .. & 0 & 0 & 0 \\ 0 & 0 & .. & 0 & 0 & 1 & 0 & .. & 0 & 0 \\ 0 & 0 & .. & 0 & 0 & 0 & 1 & 0 & .. & 0 \\ \vdots & & & & & & & & & \vdots \\ 0 & 0 & .. & 0 & 0 & 0 & 0 & .. & 0 & 1 \end{pmatrix}$$

Once we have all S_i vectors, we add them up three times to $(2 \ -1 \ ... \ -1)$ in order to get to the all 2 vector. □

3.4.3 Construction for arbitrary size and number of variables

In this section we show how to split a weight in order to get an additional variable. We also prove that adding one or two variables with unit weight results in a minimal function as well.

Lemma 3.3 (Splitting a Weight) *Let* $W = (w_0, w_1, ..., w_n)$ *be minimal. Then* $\tilde{W} = (w_0, a, b, w_2, w_3, ..., w_{n+1})$ *where* $a + b = w_1$ *is also minimal.*

Proof: Construct A while duplicating the second column. □

Lemma 3.4 (Adding an input with unit weight) *Let* $W = (w_0, w_1, ..., w_n)$ *be minimal. Then* $\tilde{W} = (w_0, w_1, w_2, w_3, ..., w_{n+1})$ *where* $w_{n+1} = 1$ *is also minimal.*

Proof: Suppose it is not minimal, implying there exists a better choice for $\tilde{W}$; let us call it W'. There are two possibilities. Either $w'_{n+1} = 0$ or some of the w'_i for $i < n + 1$ is smaller than the corresponding w_i. In the latter case, we set $x_{n+1} = 0$ and obtain the original function implemented with smaller weights, contradicting the hypothesis. Now suppose $w'_{n+1} = 0$, implying that $\tilde{f}$ does not depend on x_{n+1}. That in turn implies $\sum_0^n w_i x_i \geq 0$ or $\sum_0^n w_i x_i \leq -2$ for all inputs X. We can reduce w_0 by 1, implying the original function was not minimal. □

Using those two lemmas, the construction of functions with arbitrary size and number of variables is straightforward. In fact, we can do more than that: we can implement functions with rigid weight structure. Let us illustrate that idea by proving Lemma 2.3.

Proof of Lemma 2.3:

Given two strictly positive constants K and α, we need to construct a weight vector for which

$$w_l < w_{l+1} < ... < w_{\lfloor l+\frac{1}{\alpha} \rfloor + 1} < ... < w_{\lfloor l+\frac{2}{\alpha} \rfloor + 1}$$

$$w_l + w_{\lfloor l+\frac{1}{\alpha} \rfloor} \geq w_{\lfloor l+\frac{2}{\alpha} \rfloor}$$

The case of $\alpha = 1$ was dealth with in Example 2.7. Let the weight vector be of the form

$$(-2^p, 1, 1, 2, 4, ..., 2^p, 1, 1, 2, 4, ..., 2^p, w, w+1, ..., w+\left\lfloor\frac{1}{\alpha}\right\rfloor+1, ..., w+\left\lfloor\frac{2}{\alpha}\right\rfloor+1$$

where p is chosen so that $2^p \geq w_{\lfloor l+\frac{2}{\alpha}\rfloor+1}$ and w is an integer large enough. The same way as we prove this vector to be minimal, we show that for any other weights the above two requirements are met. □

3.5 Conclusions

We presented two techniques for constructing minimal weight threshold functions of arbitrary weight size and number of inputs. We considered both the $\{0, 1\}$ and $\{-1, 1\}$ input domains. Using these techniques we further refined the separation between polynomialy and exponentially growing weights. The natural open problem is to find out if these new techniques are useful in extending the existing lower bounds [14] on circuit size to functions with arbitrary weights.

Chapter 4 Trading weight size for circuit depth

4.1 Introduction

Many experimental results in the area of neural networks have indicated that the magnitudes of the coefficients in the linear threshold elements grow very fast with the size of the inputs and therefore limit the practical use of the network. One natural question to ask is the following. How limited is the computational power of the network if one limits oneself to threshold elements with only "small" growth in the size of the coefficients? The present chapter focuses on the implementation of LT functions that require large weights. Instead of using a single LT gate with large weights, we use a two layer circuit composed of LT gates with small weights. Large and small stand for exponentially growing in n, and polynomially growing in n, respectively, in the number of inputs.

It has been shown [17], [33], [38], [43] that there exists a function that can be implemented by a single threshold element with exponentially growing weights, but cannot be implemented by a threshold element with polynomially growing weights. In light of that result a subclass of LT was defined, the class of functions with small weights: $\widehat{LT}$ [43]. Siu and Bruck ([43]) proved that $LT_d \subset \widehat{LT}_{2d+1}$. [13] improves the bound to $LT_d \subset \widehat{LT}_{d+1}$ by showing that $LT_1 \subset \widehat{LT}_2$ and generalizing to arbitrary depth. However, the method is complicated and the proofs difficult to follow. [18] presents a simplified version of the results in [13]. It focuses on showing that $LT_1 \subset \widehat{LT}_2$. To some degree it supersedes the first by presenting a simpler, more intuitive construction. The idea is to use two operations in order to reduce the weights, divide them by powers of two and divide them modulo a prime. The resulting "small"-weight gates are connected into a circuit that produces the correct output if enough primes

are used.

We have further simplified the results presented in [13] and [18], by limiting ourselves to the simulation of a particular large weight function: $COMPARISON$. As a result we get bounds on the number of gates in our circuit of the order of $O(n^4 \log n)$, a significant improvement on the general bound of $O(n^{12} \log^{11} n)$ in [18]. We also ran a computer simulation of the circuit and determined the minimal circuits for up to 22 variables. We show the results of the simulation and finally mention applications and directions for further research.

4.2 $\widehat{LT}_2$ circuit for comparison

Consider the COMPARISON function of two n-bit numbers. Let $X_1 = (x_1, x_3, \ldots, x_{2n-1})$, $X_2 = (x_2, x_4, \ldots, x_{2n}) \in \{0,1\}^n$. The integer values represented by X_1 and X_2 are equal to $\sum_{i=1}^{n} x_{2i-1} 2^{i-1}$ and $\sum_{i=1}^{n} x_{2i} 2^{i-1}$, respectively. The COMPARISON function is defined as

$$C(X_1, X_2) = \begin{cases} 1 & X_1 > X_2 \\ 0 & \text{otherwise.} \end{cases}$$

In other words,

$$\begin{aligned} C(X_1, X_2) &= \operatorname{sgn}[X_1 - X_2] \\ &= \operatorname{sgn}\left[\sum_{i=1}^{n} 2^{i-1}(x_{2i-1} - x_{2i})\right]. \end{aligned}$$

The COMPARISON function has the interesting property that it belongs to LT_1, but not to $\widehat{LT}_1$. Using tools from harmonic analysis, it is shown in [43] that COMPARISON is in $\widehat{LT}_2$. We provide an explicit construction of an $\widehat{LT}_2$ circuit for COMPARISON using the method described in [18]. An explicit construction for COMPARISON was presented in [2].

Note that the value of COMPARISON can be determined by examining the highest-order bit position in which X_1 and X_2 differ. If this bit is 1 in X_1 and 0 in X_2, then $C(X_1, X_2) = 1$. Else, $C(X_1, X_2) = 0$. (In the event that $X_1 = X_2$, by

definition $C(X_1, X_2) = 0$.)

Let $F(X) = \sum_{i=1}^{n} 2^{i-1}(x_{2i-1} - x_{2i})$. The first step is to form a sequence of functions $F_l(X)$ by repeatedly dividing all weights of magnitude greater than 1 in half and setting weights of magnitude 1 to zero. We begin with $F_0(X) = F(X)$. After n steps, the division process yields a function which is identically zero.

$$\begin{aligned} F_0 &= x_1 - x_2 + 2x_3 - 2x_4 + \cdots + 2^{n-1}x_{2n-1} - 2^{n-1}x_{2n} \\ F_1 &= x_3 - x_4 + \cdots + 2^{n-2}x_{2n-1} - 2^{n-2}x_{2n} \\ \vdots &= \vdots \\ F_{n-1} &= x_{2n-1} - x_{2n} \\ F_n &= 0. \end{aligned}$$

Note that each division is equivalent to left-shifting both X_1 and X_2.

Lemma 1. For the linear combination $F(X)$ and the corresponding sequence of left-shifted linear combinations $F_l(X)$, $0 \leq l \leq n$, as defined above,

$$F(X) > 0 \;\Leftrightarrow\; \exists\, l : F_l(X) = 1.$$

Proof. For each $X \in \{0,1\}^{2n}$, there exists some t, such that for all $l \geq t$, $F_l(X) = 0$. Now consider the maximum error introduced by shifting. We see that

$$\max_{X \in \{0,1\}^{2n}} \max_{l \geq 0} |F_l(X) - 2 \cdot F_{l+1}(X)| \;=\; 1,$$

since at most two variables (representing the "low-order bits" with weights 1 and -1) are eliminated with each shift. Hence if $|F(X)|$ is non-zero, then there exists some l such that $|F_l(X)| = 1$. Note that if $F(X)$ is positive then all $F_l(X)$ are positive or zero, and if $F(X)$ is negative then all $F_l(X)$ are negative or zero. The result follows. □

Let $X_1 = (x_1, x_3, \ldots, x_{2n-1})$, $X_2 = (x_2, x_4, \ldots, x_{2n}) \in \{0,1\}^n$. Define the "test"

function for each $0 \leq l \leq n$ as follows.

$$T_l(X) = \begin{cases} 1 & \text{if } F_l(X) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 1 may be expressed as

$$F(X) > 0 \Leftrightarrow \bigvee_{l=0}^{n} T_l(X) = 1$$

Although trivial in itself, Lemma 1 becomes useful when we introduce the idea of computing modulo prime numbers. For what follows, we define the modulus operation to return values in a symmetric interval centered at zero: i.e., for an integer Z and a positive integer k , let $Z \bmod (2k-1) = t$, where $t \in [-k, k]$ and $t \equiv Z \pmod{2k-1}$.

Given a prime p, define a "test" operation modulo p for each $0 \leq l \leq n$ as follows.

$$T_{p,l}(X) = \begin{cases} 1 & \text{if } F_l(X) \bmod p = 1 \\ 0 & \text{otherwise.} \end{cases}$$

For a given $X \in \{0,1\}^{2n}$ and a given prime p, suppose that we compute $T_{p,l}(X)$ for all functions $F_l(X)$ in the sequence. This would not be sufficient, since the test operation modulo a prime p does not necessarily give the correct answer. However, the following lemma tells us that if we repeat the process for enough prime numbers, say r many, then we will obtain the correct answer most of the time.

Lemma 2. Let $p_1 < p_2 < \cdots$ be consecutive primes greater than 3. Let s be the minimum integer which satisfies $p_1 p_2 \cdots p_s \geq 2^{n+1} - 1$. Then for every integer Z, where $|Z| \leq 2^n - 1$,

$$Z \in [-1,1] \;\Rightarrow\; Z \bmod p_i \in [-1,1] \text{ for all primes} > 3,$$
$$Z \notin [-1,1] \;\Rightarrow\; Z \bmod p_i \in [-1,1] \text{ for less than } 3 \cdot s \text{ many primes} > 3.$$

Proof. The first statement is trivial. The second follows directly from the Chinese Remainder Theorem (see [18]). Note that $s = O(n \log n)$. □

For a given $X \in \{0,1\}^{2n}$ and a set of primes $p_1, p_2, \ldots, p_r$, suppose that we have an array of elements which compute $T_{p_i,l}(X)$ for $1 \leq i \leq r$ and $0 \leq l \leq n$, i.e.,

$T_{p_1,0}(X)$	$T_{p_1,1}(X)$	$\cdots$	$T_{p_1,n}(X)$
$T_{p_2,0}(X)$	$T_{p_2,1}(X)$	$\cdots$	$T_{p_2,n}(X)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$T_{p_r,0}(X)$	$T_{p_r,1}(X)$	$\cdots$	$T_{p_r,n}(X)$

Define a "false" positive to be the event that an element returns a 1 when $F(X) < 0$. Define a "true" positive to be the event that an element returns a 1 when $F(X) > 0$.

- When $F(X) > 0$, Lemma 1 tells us that there is at least one true positive per row. Hence in total there are at least r true positives in the array.
- When $F(X) < 0$, Lemma 2 tells us that there are always fewer than $3 \cdot s$ false positives per column. Hence in total, there are less than $3 \cdot s \cdot n$ false positives in the array.

If we choose $r = 3 \cdot s \cdot n$, then the number of elements returning 1's in the case where $F(X) < 0$ will always be less than r, whereas the number returning 1's in the case where $F(X) > 0$ will always be greater than or equal to r. The key here is that the upper bound on the number of false positives is independent of the number of rows, whereas the lower bound on the number of true positives is independent of the number of columns.

To obtain a circuit for COMPARISON, we can simply connect the test elements as inputs to an LT gate and set the threshold of the gate to r. The question remaining is how to realize the test elements using a single layer of thresholds gates with small weights.

The approach is a standard one in threshold circuit theory. For $1 \leq i \leq r$ and $0 \leq l \leq n$, define $F_{p_i,l}(X)$ to be the linear combination obtained by reducing the weights of $F_l(X)$ modulo p_i. Note that for each $X \in \{0,1\}^{2n}$,

$$T_{p_i,l}(X) = 1 \;\Leftrightarrow\; F_{p_i,l}(X) \bmod p = 1.$$

Now $F_{p_i,l}(X)$ assumes at most $n \cdot p_i$ different values. At most n of these are equal to 1 when taken modulo p_i. Denote the values of $F_{p_i,l}(X)$ which when reduced modulo p_i equal 1 as $v_1, v_2, \ldots, v_n$. For each v_j, $1 \leq j \leq n$, we place two LT gates in the first layer, and call them $G_j^{(1)}(X)$ and $G_j^{(2)}(X)$.

- The weights on the input wires of both $G_j^{(1)}$ and $G_j^{(2)}$ are set equal to the corresponding weights of $F_{p_i,l}(X)$.
- The thresholds of $G_j^{(1)}$ and $G_j^{(2)}$ are set to v_j and v_{j+1}, respectively.
- The weights on the output wires of $G_j^{(1)}$ and $G_j^{(2)}$ are set to 1 and -1, respectively.

Clearly,

$$\sum_{j=1}^{n}(G_j^{(1)}(X) + G_j^{(2)}(X)) = T_{p_i,l}(X)$$

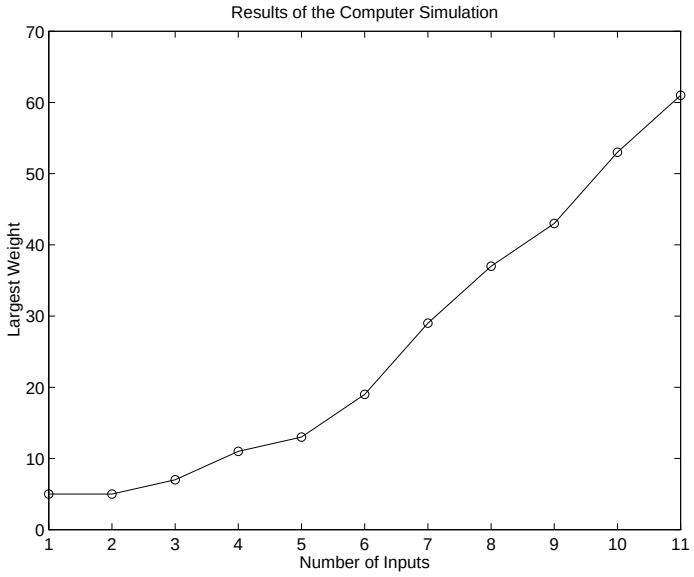
To summarize, we have a total of $3 \cdot s \cdot n^2$ test elements, each of which requires $2 \cdot n$ LT gates to realize. So, in total we require $6 \cdot s \cdot n^3$ LT gates. Since $s = O(n \log n)$, we conclude that our construction has size $O(n^4 \log n)$.

4.3 Computer simulation

We used a short Matlab program, shown in the Appendix, in order to simulate the $\widehat{LT}_2$ construction of COMPARISON. For each n (half the number of variables) we found the smallest number of primes and the smallest threshold that produce a correct circuit. The following table shows the results:

n	# of primes	threshold
1	1	0
2	1	0
3	2	1
4	3	2
5	4	3
6	6	5
7	8	7
8	10	9
9	12	11
10	14	13
11	16	15

We plot the largest weight in the circuit as a function of the number of inputs.

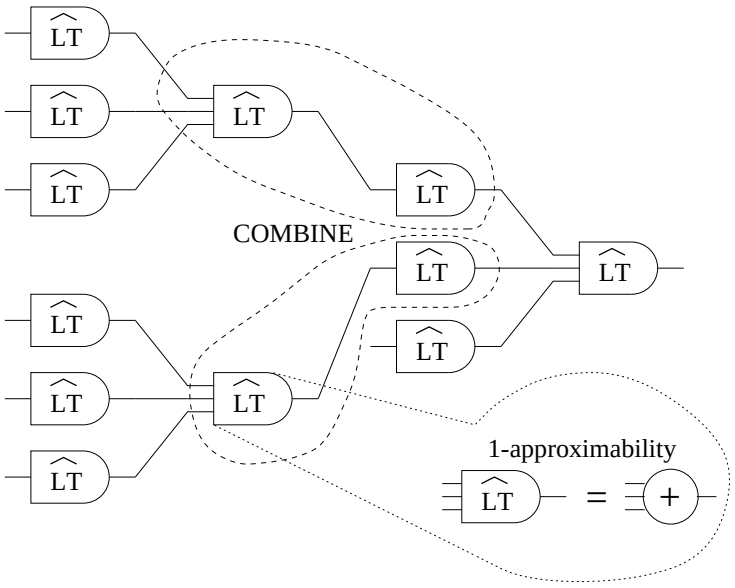


4.4 Generalization to $LT_d \subset \widehat{LT}_{d+1}$

An important property of the above construction, called "1-approximability," is the fact that one can replace the output threshold gate by a weighted sum (i.e., eliminate

the $sgn(.)$ function). The output of such a circuit will be very close to 0 or 1. That fact is used in [13] to show that $LT_d \subset \widehat{LT}_{d+1}$. The idea is the following:

1. Take the output gate, G_{last} of the LT_d circuit. It is an LT gate so we can substitute it with an $\widehat{LT}_2$ circuit as shown in Section 4.2.
2. Take all gates, $G^{(i)}_{last-1}$ that connect to it, and substitute them as well.
3. Use the "1-approximability" property to combine the output layer of $G^{(i)}_{last-1}$ with the input layer of $G^{(i)}_{last}$.
4. Continue the same procedure until all LT gates are replaced by $\widehat{LT}_2$ circuits.



Combine layers in order to simulate an LT_d function
by an $\widehat{LT}_{d+1}$ circuit.

4.5 Conclusion

$LT_d \subset \widehat{LT}_{d+1}$ is a very useful result. For example, we can use it show that $LTM \subset \widehat{LT}_2$. By construction $MADD \in LTM$ which then implies that $MADD \in \widehat{LT}_2$.

An interesting direction of research is to derive the exact size of the $\widehat{LT}_2$ implementation of $COMPARISON$ using the approach presented in Section 4.2.

Chapter 5 LTM: linear threshold element with multiple thresholds

5.1 Introduction

Motivated by our work on the VLSI implementation of LT elements [8], see Chapter 6, we introduce in this paper a more powerful computing element, a multiple threshold neuron, which we call LTM, which stands for Linear Threshold with Multiple transitions; see [15] and [35]. Instead of the sign function in the LT element, it computes an arbitrary (with polynomialy many transitions) Boolean function of the weighted sum of its inputs.

The main issues in the study of LTM circuits (circuits consisting of LTM elements) include the estimation of their computational capabilities and limitations and the comparison of their properties to those of AON circuits. A natural approach in this study is first to understand the relation between LT circuits and LTM circuits. Our main contributions in this chapter are:

- We demonstrate the power of LTM by deriving efficient designs of LTM circuits for the addition of m integers and the product of two integers.
- We show that LTM circuits are more amenable in implementation than LT circuits. In particular, the area of the VLSI layout is reduced from $O(n^2)$ in LT circuits to $O(n)$ in LTM circuits, for n input symmetric Boolean functions.
- We characterize the computing power of LTM relative to LT circuits.

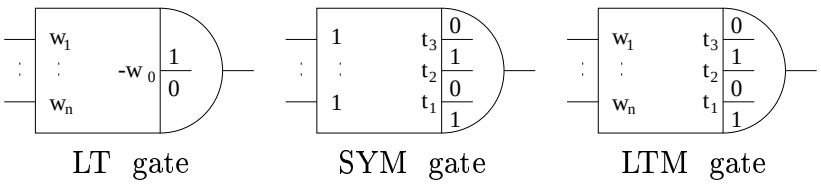


Figure 5.1: Schematic representation of LT, SYM and LTM computing elements.

Next we describe the formal definitions of LT and LTM elements.

5.1.1 Definitions and examples

Definition 5.1 *(Linear Threshold Gate – LT)*
A linear threshold gate computes a Boolean function of its binary inputs:

$$f(X) = sgn(w_0 + \sum_{i=1}^{n} w_i x_i)$$

where the w_i are integers and $sgn(.)$ outputs 1 if its argument is greater or equal to 0, and 0 otherwise.

Figure 5.1.1 shows an n-input LT element; if $\sum_1^n w_i x_i \geq -w_0$ the element outputs 1, otherwise it outputs 0. A single LT gate is unable to compute parity. The latter belongs to the general class of symmetric functions – SYM.

Definition 5.2 *(Symmetric Functions – SYM)*
A Boolean function f is symmetric if its value depends only on the number of ones in the input denoted by $|X|$,

$$|X| = \sum_{1}^{n} x_i$$

Figure 5.1.1 shows an example of a symmetric function; it has three transitions, it outputs 1 for $|X| < t_1$ and for $t_2 \leq |X| < t_3$, and 0 otherwise. AND, OR and parity are examples of symmetric functions. A single LT element can implement only a

limited subset of symmetric functions. We define LTM as a generalization of SYM. That is, we allow the weights to be arbitrary as in the case of LT, rather than fixed to 1 (see Figure 5.1.1).

Definition 5.3 *(Linear Threshold Gate with Multiple Transitions – LTM)*
A function f is in LTM if there exists a set of weights $w_i \in Z$, $1 \leq i \leq n$ and a function $h : Z \longrightarrow \{0,1\}$ such that

$$f(X) = h(\sum_{i=1}^{n} w_i x_i) \text{ for all } X \in \{0,1\}^n$$

The only constraint on h is that it undergoes polynomialy many transitions as its input scans $[-\sum_{i=1}^{n} |w_i|, \sum_{i=1}^{n} |w_i|]$.

Notice that without the constraint on the number of transitions, an LTM gate is capable of computing any Boolean function. Indeed, given an arbitrary function f, let $w_i = 2^{i-1}$ and $h(\sum_1^n 2^{i-1} x_i) = f(x_1, ..., x_n)$.

Example 5.1 $(XOR \in LTM)$
$XOR(X)$ outputs 1 if $|X|$, the number of 1's in X, is odd. Otherwise it outputs 0. To implement it choose $w_i = 1$ and $h(k) = \frac{1}{2}(1 - (-1)^k)$ for $0 \leq k \leq n$. Note that $h(k)$ needs not be defined for $k < 0$ and $k > n$, and has polynomialy many transitions.

Another useful function that LTM can compute is $ADD(X, Y)$, the sum of two n-bit integers X and Y.

Example 5.2 $(ADD \in LTM)$
To implement addition we set

$$f_l(X, Y) = h_l(\sum_{i=1}^{l} 2^i (x_i + y_i))$$

where $h_l(k) = 1$ for $k \in [2^l, 2 \times 2^l - 1] \cup [3 \times 2^l, +\infty)$. Defined thus, f_l computes the m-th bit of $X + Y$. Figure 5.2 shows an example for $n = 4$.

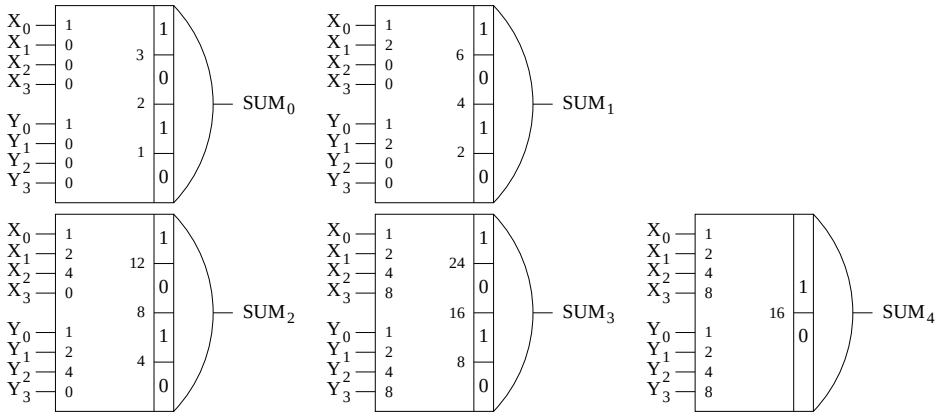


Figure 5.2: Addition of two 4-bit integers using a single LTM gate per output bit.

5.1.2 Organization

This chapter is organized as follows. In Section 5.2, we study a number of applications of LTM circuits. In particular, we show how to compute the addition of m integers with a single layer of LTM elements. In Section 5.3, we prove the characterization results of LTM – inclusion relations, in particular $LTM \subseteq \widehat{LT}_2$. In addition, we indicate which inclusions are proper and exhibit functions to demonstrate the separations.

5.2 LTM constructions

The theoretical results about LTM can be applied to the VLSI implementation of Boolean functions. The idea of a gate with multiple thresholds came to us as we were looking for an efficient VLSI implementation of symmetric Boolean functions. Even though a single LT gate is not powerful enough to implement any symmetric function, a 2-layer LT circuit is, Figure 5.2. Furthermore, it is well known that such a circuit performs much better than the traditional logic circuit based on AND, OR and NOT gates. The latter has exponential size (or unbounded depth) [51].

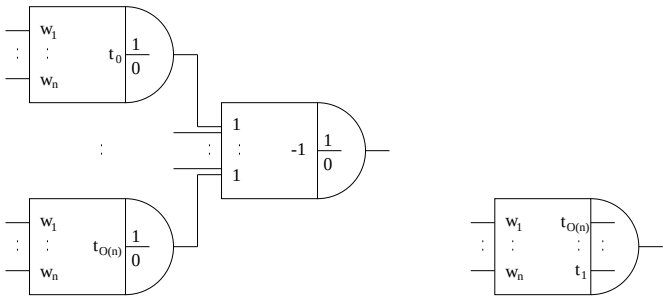


Figure 5.3: LT circuit of size $O(n)$ versus a single LTM gate.

Proposition 5.1 *(LT_2 versus LTM for symmetric function implementation)*
The LT_2 layout of a symmetric function requires area of $O(n^2)$; while using LTM one needs only area of $O(n)$.

PROOF:
Implementing a generalized symmetric function in LT_2 requires up to n LT gates in the first layer. Those have the same weights w_i except for the threshold w_0. Instead of laying out n times the same linear sum $\sum_1^n w_i x_i$, we do it once and compare the result to n different thresholds. The resulting circuit corresponds to a single LTM gate. □

Figure 5.2 shows the advantage of LTM over LT for the implementation of a generalized symmetric function. Indeed, the LT_2 layout is redundant; it has n copies of each weight, requiring area of at least $O(n^2)$. On the other hand, LTM performs a single weighted sum. Its area requirement is $O(n)$.

A single LTM gate can compute the addition of m n-bit integers $MADD$. The only constraint is that m be polynomial in n.

Theorem 5.1 *($MADD \in LTM$)*
A single layer of LTM gates can compute the sum of m n-bit integers, provided that

m is at most polynomial in n.

PROOF:

$MADD$ returns an integer of at most $n + \log m$ bits. We need one LTM gate per bit. The least significant bit is computed by a simple m-bit XOR. For all other bits we use

$$f_l(X^{(1)}, .., X^{(m)}) = h_l(\sum_{i=1}^{l} 2^i \sum_{j=1}^{m} x_i^{(j)})$$

to compute the l-th bit of the sum. □

Example 5.3 (Addition of three 3-bit integers)
We apply the above construction to the case $m = 3$, $n = 3$. The result is shown in Figure 5.2. Notice that the result is in the range $\{0, ..., 21\}$; therefore, the LTM gate computing output bit 3 requires only 2 thresholds.

Corollary 5.1 *($PRODUCT \in PTM$) A single layer of PTM (which is defined below) gates can compute the product of m n-bit integers, provided that m is at most polynomial in n.*

PROOF:

By analogy with PT_1, defined in [9], in PTM_1 (or simply PTM) we allow a polynomial rather than a linear sum:

$$f(X) = h(w_1x_1 + ... + w_nx_n + w_{(1,2)}x_1x_2 + ...)$$

However, we restrict the sum to have polynomialy many terms (else, any Boolean function could be realized with a single gate). The product of two n-bit integers X and Y can be written as $PRODUCT(X, Y) = \sum_{i=1}^{n} x_iY$. We use the construction of $MADD$ in order to implement $PRODUCT$.

$$PRODUCT(X, Y) = MADD(x_1Y, x_2Y, ..., x_nY)$$

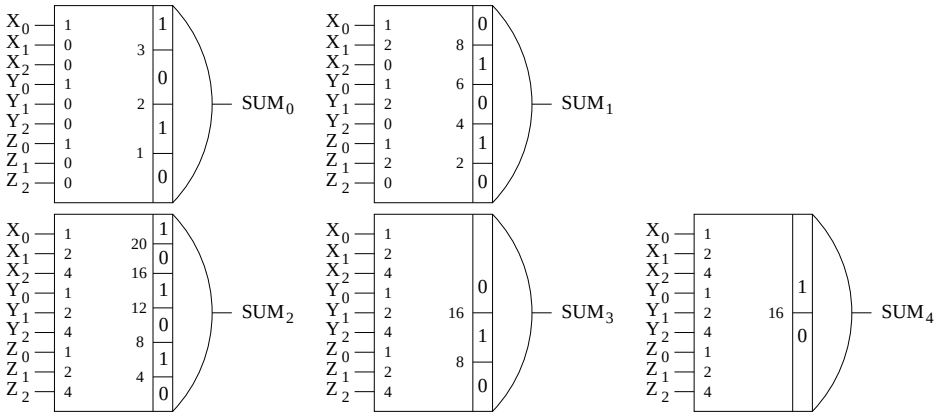


Figure 5.4: MADD: addition of three 3-bit integers – X, Y and Z – using a layer of LTM elemets.

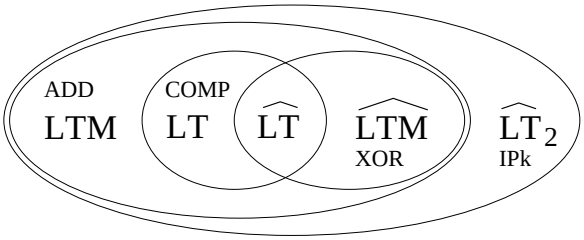


Figure 5.5: Relationship between classes.

$$f_l(X,Y) = h_l(\sum_{j=1}^{n}\sum_{i=1}^{l} 2^i x_j y_i)$$

f_l outputs the l-th bit of the product. □

5.3 Classification of LTM

We use a hat to indicate small (polynomialy growing) weights, e.g., $\widehat{LT}$, $\widehat{LTM}$ [6], [43], and a subscript to indicate the depth (number of layers) of the circuit of more than a single layer. All the circuits we consider in this paper are of polynomial size (number of elements) in n (number of inputs). For example, the class $\widehat{LT}_2$ consists of those Boolean functions that can be implemented by a depth-2 polynomial size circuit of $\widehat{LT}$ elements.

Figure 5.3 depicts the membership relations between five classes of Boolean functions, including LT, $\widehat{LT}$, LTM, $\widehat{LTM}$ and $\widehat{LT}_2$, along with the functions used to establish the separations.

In this section we will prove the relations illustrated by Figure 5.3.

Theorem 5.2 *(Classification of LTM)*
The inclusions and separations shown in Figure 5.3 hold. That is,

1. $\widehat{LT} \subseteq LT \subseteq LTM$
2. $\widehat{LT} \subseteq \widehat{LTM} \subseteq LTM$
3. $LTM \subseteq \widehat{LT}_2$
4. $XOR \in \widehat{LTM}$ *but* $XOR \notin LT$
5. $COMP \in LT$ *but* $COMP \notin \widehat{LTM}$
6. $ADD \in LTM$ *but* $ADD \notin LT \cup \widehat{LTM}$
7. $IP_k \in \widehat{LT}_2$ *but* $IP_k \notin LTM$

5.4 Proof of the classification theorem

Let us prove the relations illustrated by Figure 5.3. We first show the inclusion relations. Then, we provide functions that demonstrate the separation between classes.

5.4.1 Inclusions

Most inclusion relations follow from the definitions: $\widehat{LT} \subseteq LT \subseteq LTM$ and $\widehat{LT} \subseteq \widehat{LTM} \subseteq LTM$. Only one requires a proof:

$\underline{LTM \subseteq \widehat{LT}_2}$

To show the above statement we use a result from [13]: a single LT gate with arbitrary weights can be realized by an $\widehat{LT}_2$ circuit. Furthermore, the non-linearity in the second layer can be removed without affecting the output of the circuit (a property called "1-approximability," [18]). So, given $f \in LT$, $f(X) = \sum_{i_1}^{p} w_i f_i(X)$ where p is polynomial in n and $f_i \in \widehat{LT}$ for all i.

Now, consider the LT_2 implementation of a function in LTM. It consists of a layer of identical LT gates followed by a single gate with 1 and -1 weights and a -1 threshold. We substitute each LT gate of the first layer by its equivalent layer of $\widehat{LT}$ gates and weighted sum. We combine the weighted sums, i.e., collapse the second and the third level. The resulting circuit is in $\widehat{LT}_2$.

5.4.2 Separation

In Example 1 we saw that $XOR \in \widehat{LTM}$ and it is well known that $XOR \notin LT$. On the other hand $COMP(X, Y)$, the comparison of two n-bit integers is in LT [43].

$$COMP(X,Y) = sgn(\sum_{i=1}^{n} 2^i(x_i - y_i)) = \begin{cases} 1 & \text{if } Y \leq X \\ 0 & \text{otherwise} \end{cases}$$

Let us show that $COMP \notin \widehat{LTM}$. For that we introduce the notion of *entropy* of a Boolean function. An equivalent definition based on communication complexity is developed in [47].

Definition 5.4 *(Entropy)*
Given an n-variable Boolean function, S a subset of those variables and $s \in \{0,1\}^{|S|}$, we call $f_s(x_1, .., x_{n-|S|})$ the function obtained by assigning the value s to S in f. The entropy of f is defined as:

$$E[f] = \max_S |\{f_s : s \in \{0,1\}^{|S|}\}|$$

In words, the entropy is the maximum number of sub-functions over $n-|S|$ variables one can produce by assigning to a set S of its n variables all possible $2^{|S|}$ values. The maximum is taken over S.

Lemma 5.1 *(Exponential Entropy implies Exponential Weights)*
Given a function f such that $E[f]$ is exponential in n, its LTM implementation requires exponential weights, i.e., $\sum_1^n |w_i|$ exponential in n.

Proof: A subfunction can be written as

$$f_s(x_1,..,x_{n-|S|}) = f(X, S=s) = h(\sum_{i\in X-S} w_i x_i + W_s)$$

where $W_s = \sum_{i\in S} w_i s_i$. By the pigeonhole principle, and given that W_s is an integer, $|\{W_s : s\}|$ must be greater than $E[f]$. If it is not, there will not be enough distinct values of W_s to map to all $E[f]$ distinct sub-functions. That in turn implies

$$E[f] \leq \sum_{i\in S} |w_i| \leq \sum_{i=1}^{n} |w_i|$$

□

$\underline{COMP \notin L\widehat{TM}}$

Proof: We show that $E[COMP]$ is exponential and use Lemma 5.1. Let

$$f_s(x_1,..,x_n) = COMP(X, Y=s)$$

There are 2^n such functions; let us show that they are all distinct. Given two distinct integers s_1 and s_2, choose X_0 such that $s_1 \leq X_0 < s_2$, then $f_{s_1}(X_0) \neq f_{s_2}(X_0)$. □

$\underline{ADD \in LTM \text{ but } ADD \notin LT \cup L\widehat{TM}}$

Proof: We already saw that $ADD \in LTM$. The least significant bit of the sum is XOR which is not in LT. On the other hand, $E[ADD]$ is exponential by a proof similar to the one for $COMP$, implying that $ADD \notin L\widehat{TM}$. □

$\underline{IP_k \in \widehat{LT}_2 \text{ but } IP_k \notin LTM}$

Proof: Let $IP(X,Y) = \sum_1^n x_i y_i$. Define the function $IP_k(X,Y) = 1$ iff $IP \geq k$, else $IP_k = 0$. We claim that $IP_k \notin LTM$. Indeed, if IP_k was in LTM, then it could be implemented by a layer of $\widehat{LT}$ gates followed by a weighted sum [13]. We could then combine the circuits for $k = 1..n$ to implement $IP2$ (Inner Product mod 2) in $\widehat{LT}_2$ which is known to be false [14]. □

What remains to be shown in order to complete the classification picture is $\widehat{LT} = LT \cap \widehat{LTM}$. We conjecture that this is true.

Conjecture 5.1 *($\widehat{LT} = LT \cap \widehat{LTM}$)*
Let LT denote the class of linear threshold functions with arbitrary weights, and $\widehat{LT}$ the class of functions with polynomial growth in the weights, and let LTM be as described in Definition 5.1, then

$$\widehat{LT} = LT \cap \widehat{LTM}$$

5.5 Conclusions

Our original goal was to use theoretical results in order to efficiently lay out a generalized symmetric function. During that process we came to the conclusion that the LT_2 implementation is partially redundant, which lead to the definition of LTM, a new, more powerful computing element. We characterized the power of LTM relative to LT. We showed how it can be used to reduce the area of VLSI layouts from $O(n^2)$ to $O(n)$ and derive efficient designs for multiple addition and product. Interesting directions for future investigation are (i) to prove the conjecture: $\widehat{LT} = LT \cap \widehat{LTM}$, (ii) to apply spectral techniques ([9]) to the analysis of LTM, in particular show how PTM fits into the classification picture (Figure 5.3).

Linear decision lists, LDL, [49], were mentioned in the concluding section of

Chapter 2, Section 2.6. It is easy to see that LTM is an instance of an LDL, implying that $LTM \subseteq LDL$. An interesting problem is to establish whether $LTM \in LDL$ or $LTM = LDL$. To prove the former, which is the more probable answer, one needs to show an LDL construction of IP_k, the function shown not to be in LTM.

Another direction for future research, described in Chapter 6, consists in introducing the ideas described above in the domain of VLSI. We have fabricated a programmable generalized symmetric function on a 2μ, analog chip using the model described above. Floating gate technology is used to program the weights. We store a weight on a single transistor by injecting and tunneling electrons on the floating gate [16].

Chapter 6 VLSI implementation: programmable neural logic

6.1 Introduction

In the field of neuromorphic analog VLSI, most research deals with implementing neurons that in some way learn or adapt, [11], [16], [19]. That is because it is believed that the power of neural systems comes from their adaptive behavior. In fact it has been shown that the function performed by a neuron – the sum of weighted inputs followed by a threshold – is by itself (without learning) a powerful building block. For many years, theoretical computer science has studied the power of such neurons, in issues related to polynomial versus exponential size circuits and the general problem of NP completeness. The basic problem – build Boolean input Boolean output threshold circuits, to compute useful Boolean functions efficiently. Threshold circuits have been shown to be surprisingly powerful [1]. For example, integer division can be implemented by a polynomial-size threshold circuit of constant depth, [5], [44]. In other words, if one is to implement a threshold circuit to compute the division of two n-bit integers, one needs polynomially many, in n, threshold elements. On the other hand, using the traditional logic circuits, composed of AND, OR and NOT gates, requires exponentially many gates. That is also the case with simpler functions such as exclusive-OR and integer addition.

Many results from the theory of threshold circuits could be applied to the implementation of circuits on silicon. Results such as the relationship between the maximal size allowed for the weights and the power of the resulting element or circuit [6], [13], not to mention efficient designs for XOR, ADD, $MULTIPLY$ and other useful func-

tions; see [24], [28], [31]. For example, a simple application of the theory led us to the introduction of a *multiple threshold element*; see Chapter 5. The latter reduces the area of the layout from $O(n^2)$ to $O(n)$ for certain Boolean functions, in particular symmetric functions, such as PARITY.

Our research has three distinct goals:

1. The implementation aspect. To design and implement efficient threshold elements on silicon.

2. The theoretical aspect. To leverage the work done in theoretical computer science in order to design high performance threshold circuits in a systematic way.

3. The programmable aspect. To introduce threshold elements as building blocks in FPGA's.

Implementations of threshold circuits were proposed already in the 60's and 70's [4], [48], [53], and more recently in [28], [39]. To our knowledge, the theoretical results on threshold circuits have not been linked to any work involving silicon implementations. Programmable neuron-based hardware has been recently proposed [39], [41]. In the implementation section below, we show how those relate to our work. For a short overview of FPGA's, see [50]. In Section 6.2 we compare threshold circuits to traditional logic circuits. In Section 6.3 we discuss the programmable aspect of the design. Section 6.4 shows the VLSI implementation and testing results. The LTM element was presented in Chapter 5 and in [7] from the theoretical point of view. It was compared to traditional threshold circuits and (AND, OR, NOT) circuits. Section 6.5 presents an implementation of LTM on a 2μ-technology $2mm \times 2mm$ chip.

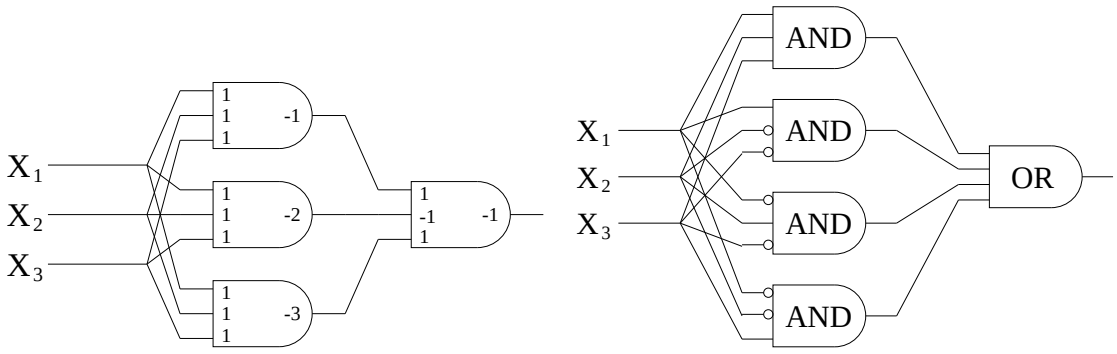


Figure 6.1: Neural vs. conventional logic. Two circuits computing XOR.

6.2 Neural logic versus conventional logic

Why bother use threshold elements given that any Boolean functions can be implemented, in a systematic way, by a circuit of AND, OR and NOT gates (AON circuit). The reason is that for some functions, such as exclusive-OR (XOR), the number of elements in the AON circuit will grow exponentially with the number of bits in the input, [51]. On the other hand, if one uses linear threshold elements, the number of gates is linear in the number of input bits. This is shown in Figure 6.2 for a 3-bit input. In general, a depth-2, AON circuit computing XOR of n bits requires at least $2^{n-1}+1$ gates. Using LT, one needs only $n+1$ gates.

It is easy to see that LT circuits are more powerful than AON circuits. The reason is that for any single AON gate there is an equivalent LT gate, computing the same function. On the contrary, most LT gates do not have equivalents in AON.

Example 6.1 (MAJORITY)
Consider the function defined by the weight vector $(w_0, ..., w_5) = (-3, 1, 1, 1, 1, 1)$:

$$f(x_1, ..., x_5) = sgn(-3 + x_1 + x_2 + x_3 + x_4 + x_5)$$

It outputs 1 only when three or more of the inputs are 1. It cannot be implemented

by a single AND or OR gate, even if we allow some inputs to be negated (NOT). □

One may argue that even though LT circuits are more powerful, their building blocks are more complex and therefore will require a larger area in the circuit layout. This argument is correct to some extent. However, the exponential to polynomial decrease in the number of required gates dominates the penalty introduced by an increase in their size. The following section addresses the issue.

6.3 Programmable versus hardwired weights

One can look at FPGA's as circuits of elements in which the function that each element computes can be programmed, that is it can be chosen among a set of available functions. In traditional FPGA's that set consists of AND, OR and NOT. We propose a larger collection of functions, namely the set of Linear Threshold Functions, LT.

All the information about an LT gate is contained in the weights and threshold. We consider two ways of implementing the weights.

- Hardwired weights are encoded in the width to length ratio of a transistor.
- Programmable weights are stored as non volatile charge on a floating gate.

Hardwired weights cannot be changed once the circuit has been fabricated, while programmable ones can. Hardwired weights present an interesting problem in terms of automated layout. Some functions such as the comparison function, $COMP$, require weights ranging from 1 to $2^{n/2}$. Figure 6.3 shows an 8-bit $COMP$ function. AND, OR and all symmetric functions can be implemented with small weights. This difference implies that using hardwired weights, some LT gates are larger than others.

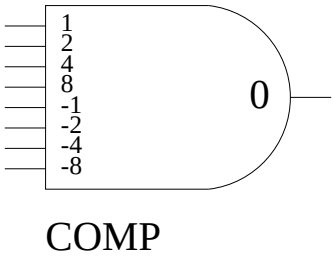


Figure 6.2: Comparison of two 4-bit integers.

Using programmable weights simplifies the layout, and allows one to modify the function that the LT element computes. In the next section we describe the details of the implementation.

6.4 Implementation and results

In [41] the authors have fabricated a neuron-based circuit that implements an arbitrary Boolean function. We implement an arbitrary threshold element (a limited set of Boolean functions). The actual function is selected by modifying the weights. Figure 6.4 shows the schematic implementation. A 16-input threshold element was fabricated using the standard 2 μm double - poly analog process available from MOSIS. See Figure 6.4 for the layout. The 16 inputs are fed to all four gates via metal 2 (purple); such layout allows one to build dense arrays of threshold elements.

We store the weights on polysilicon floating gates, using a single transistor per weight, providing long-term retention without refresh. The multiplication relies on the fact that the inputs are boolean, 0 Volts for a logical 0, and X volts for a logical 1, where X can vary from 1 to 5 Volts. An input generates current proportional to the corresponding weight. The sum, $\sum_{i=1}^{n} w_i x_i$, comes naturally as we connect all transistors to the same node. That is another difference with the approach of [39] where a capacitive sum of voltages is used, rather than a sum of currents. Finally two inverters provide hard thresholding pulling the output to logical 0, or logical 1.

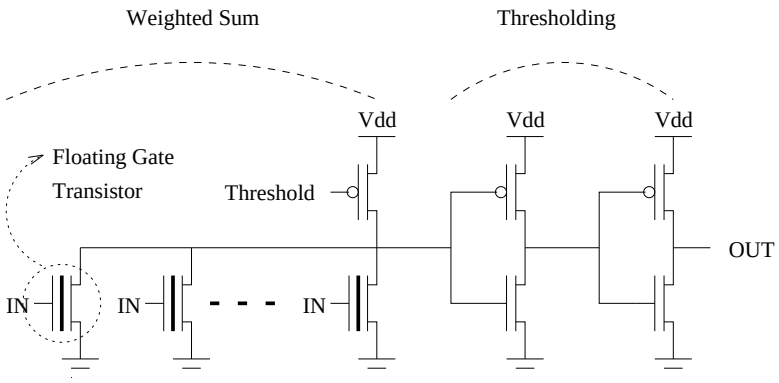


Figure 6.3: Schematic of a Programmable Linear Threshold Element.

To program in a new function one modifies the weights via tunneling (increasing) and hot electron injection (decreasing), see [16]; [19], [55] for similar applications of floating gates. As shown in [10] an analog memory cell, which is slightly more complex than the single transistor storage used here, can store up to 14 bits of information, an amount largely sufficient for most practical threshold functions.

We tested the linearity of our threshold element by detecting the value of the threshold, w_0, at which $w_0 + \sum_{i=0}^{16} x_i = 0$, while varying the number of 1's in the input vector. 1 Volt was used as the value of logical 1. Figure 6.5 shows the result.

Notice the square root shape of the data. This illustrates an important point, the voltage one needs to apply in order to get a certain value of T is not linear in T. For an $nFET$, operating above or below threshold the contributions of a single input are respectively:

$$I = \frac{\beta}{2}(V_g - V_T)^2$$

$$I = I_0 e^{\frac{\kappa V_g}{V_T}}$$

where V_T is the thermal voltage and β, I_0 and κ are constants. Hardwired weights are encoded as the W/L ratio of the transistor to which both β and I_0 are proportional [29]. That in turn makes the values of the weights linear in W/L irrespective of the region of operation of the transistor. In the case of programmable weights, the value

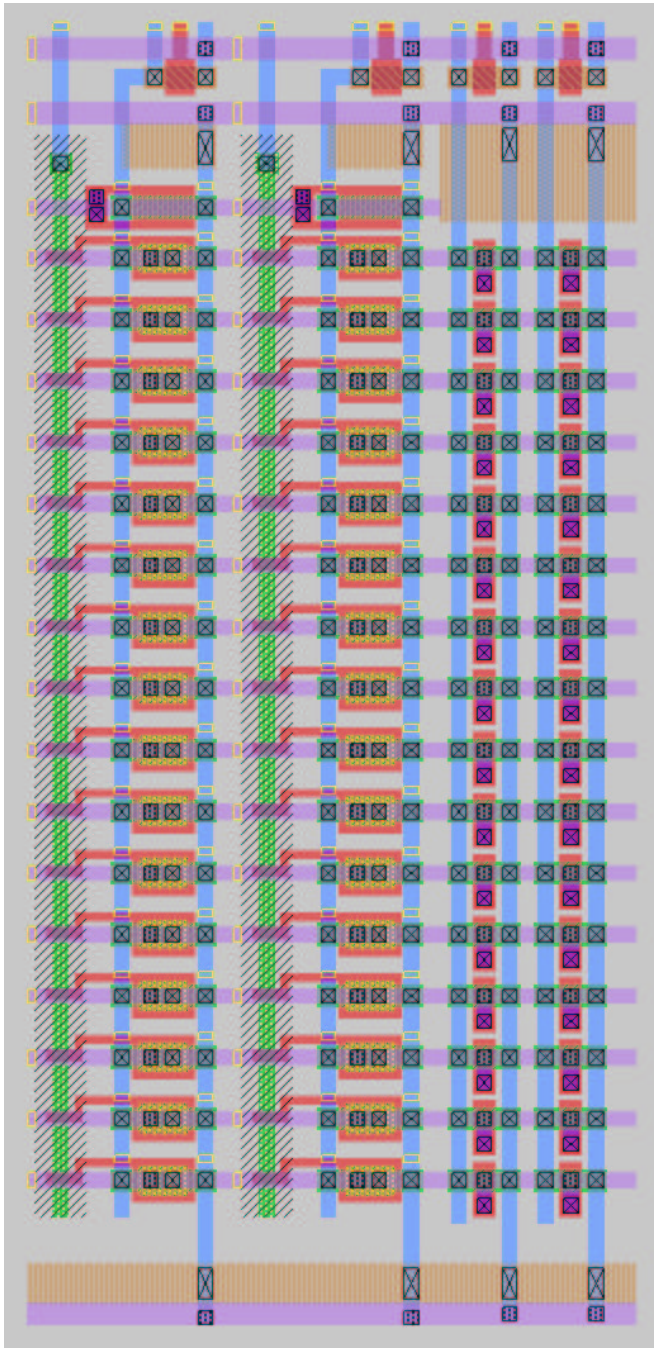


Figure 6.4: Layout of the linear sum $-w_0 + \sum_{i=1}^{16} w_i x_i$. Four threshold elements are shown, two programmable and two non programmable, the latter having unit weights. The area shown is $168\mu \times 360\mu$. The chip was fabricated using the 2μ technology available from MOSIS.

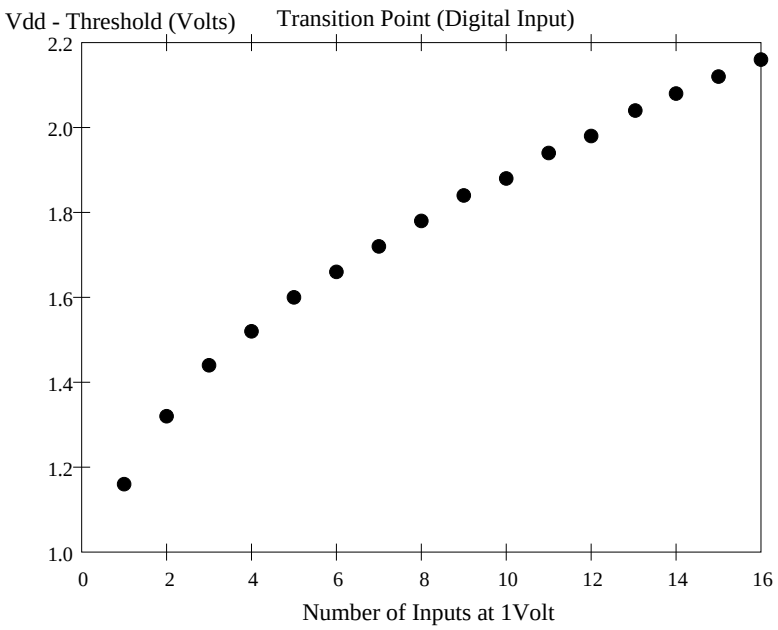


Figure 6.5: $Vdd - Threshold$ versus the number of 1's in the input.

of the weights can be quadratic or exponential in the voltage stored on the floating gate; see Figure 6.4. Such non-linearities result in a large dynamic range.

6.5 LTM:VLSI layout

The theoretical results about LTM can be applied to the VLSI implementation of Boolean functions. The idea of a gate with multiple thresholds came to us as we were looking for an efficient VLSI implementation of symmetric Boolean functions. Even though a single LT gate is not powerful enough to implement any symmetric function, a 2-layer LT circuit is. Furthermore, it is well known that such a circuit performs much better than the traditional logic circuit based on AND, OR and NOT gates. The latter has exponential size (or unbounded depth) [51].

Proposition 6.1 *(LT_2 versus LTM for symmetric function implementation)*
The LT_2 layout of a symmetric function requires area of $O(n^2)$, while using LTM

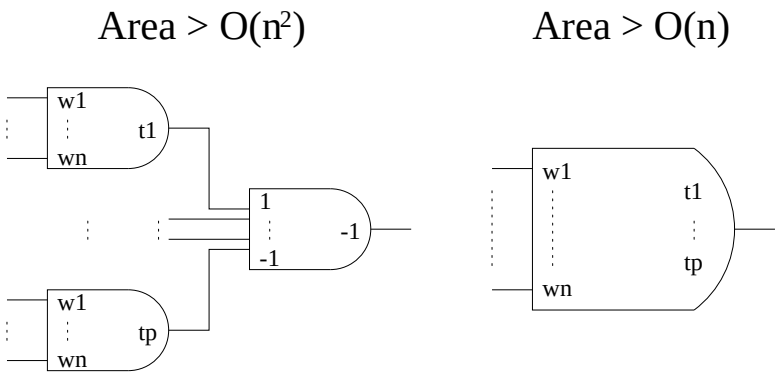


Figure 6.6: Advantage of LTM (right) over LT (left) for symmetric functions. The weighted sum is implemented only once rather than in each gate of the first layer.

one needs only area of $O(n)$.

PROOF:

Implementing a generalized symmetric function in LT_2 requires up to n LT gates in the first layer. Those have the same weights w_i except for the threshold w_0. Instead of laying out n times the same linear sum $\sum_1^n w_i x_i$, we do it once and compare the result to n different thresholds. The resulting circuit corresponds to a single LTM gate. □

The above proposition is illustrated in Figure 12. The LT_2 layout is redundant; it has n copies of each weight, requiring area of at least $O(n^2)$. On the other hand, LTM performs a single weighted sum, and its area requirement is $O(n)$. Figure 6.5 shows a high level schematic of the LTM element.

One such element was fabricated on a $2mm \times 2mm$ chip, using 2μ technology from MOSIS. Figure 14 shows its layout. It has 16 inputs. The output consists of a 4-bit bus addressing a 4-bit memory cell (not shown). The weighted sum is implemented in the Neuron MOS fashion, as a capacitive sum of voltages, see [30], [39], as opposed to a sum of currents used in the layout of the LT gate, Figure 6.4. The values of the weights and thresholds are stored on floating gates. They can be programmed globally (e.g., increasing all weights, or thresholds in parallel), or individually by using

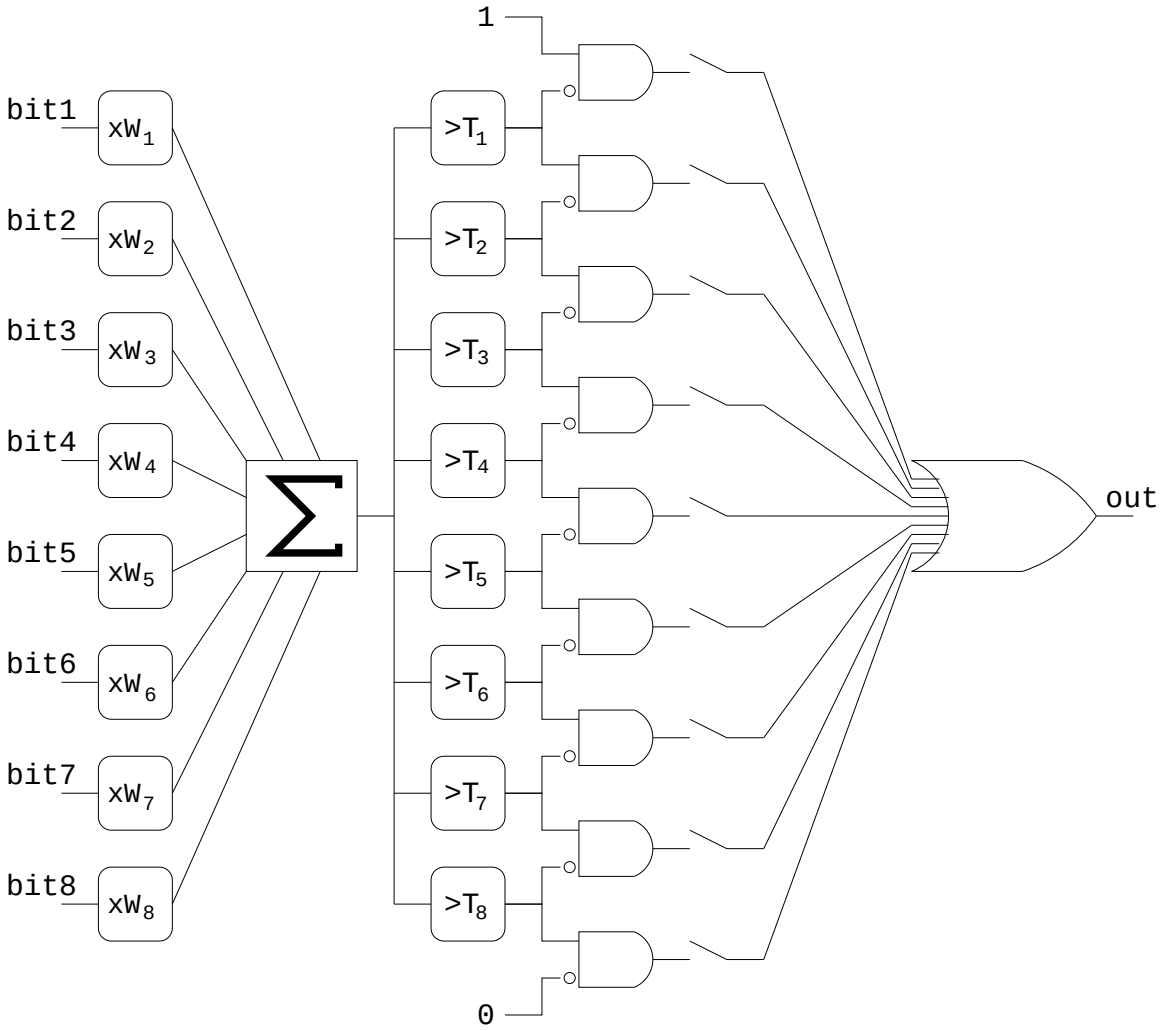


Figure 6.7: High level schematic of an LTM gate.

the input lines to select a particular weight/threshold. Assuming the thresholds are increasing, one knows that only a single line is at logical 1 at the output of the AND layer; see Figure 13. Taking advantage of this fact, we multiplex the 16 lines into a 4-bit bus addressing a memory cell, which stores the values of the function. In general, we get a bus of $\log_2 t$ bits, where t is the number of transitions (thresholds) of the LTM element. In the case of symmetric functions $t = n$, the number of inputs. Alternatively, one can look at the circuit of Figure 6.5 as a 16-bit input, 4-bit output programmable computing element.

6.6 Conclusion

We have fabricated and tested a 16-input programmable linear threshold element using floating gates to store the weights. Such storage requires no refresh and allows the weights to be modified via tunneling and injection. We have fabricated a second chip implementing a 16-input multi-threshold element. A single multi-threshold element can implement XOR and integer addition. It takes advantage of the fact that some useful Boolean functions can be implemented by a 2-layer LT circuit in which all gates of the first layer have the same weights. That allows us to reduce the area from n^2 to n, by implementing the weighted sum only once.

From the practical point of view, one possible extension of this research is to devise a systematic (maybe automated) way of generating the layout of threshold circuits with hardwired weights. Another direction of research is to incorporate programmable threshold elements as building blocks in FPGA's.

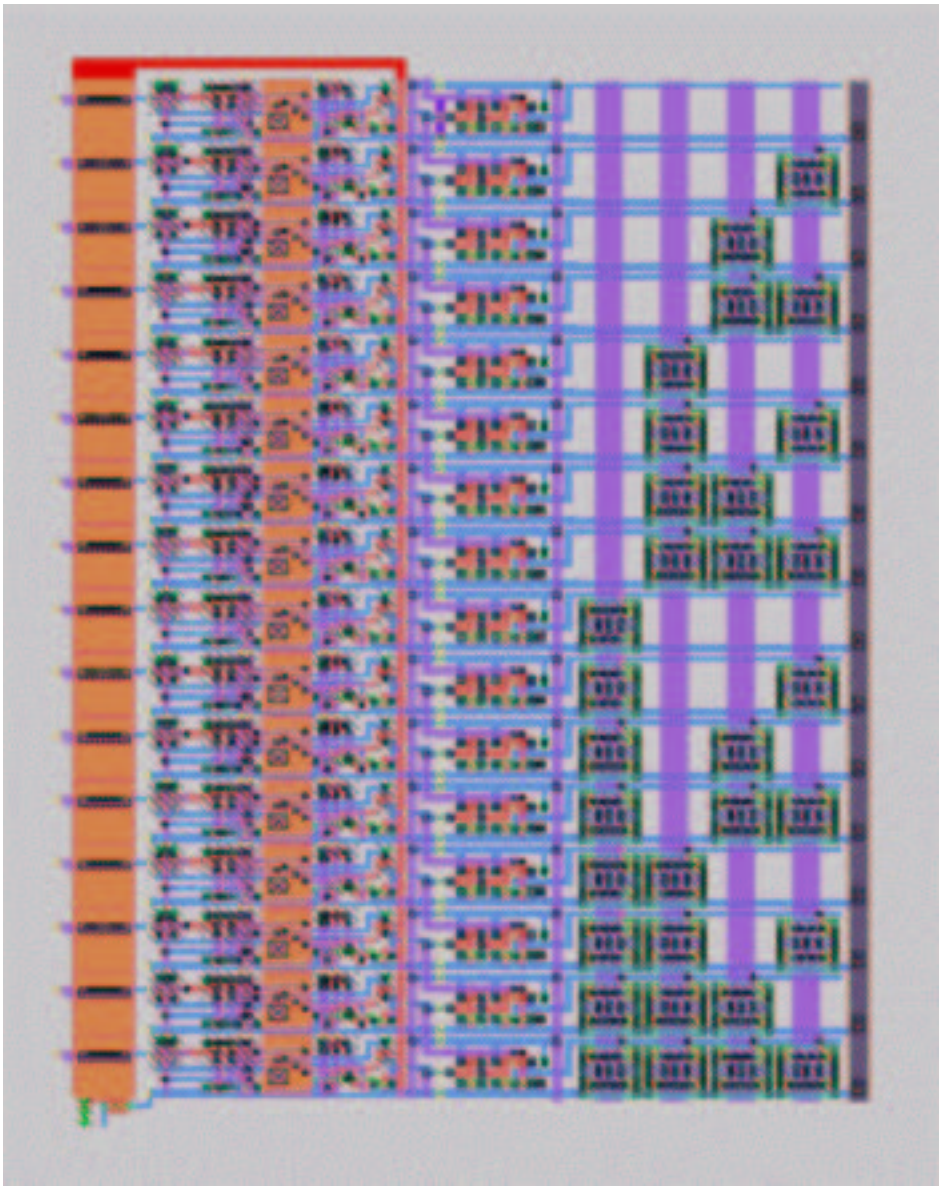


Figure 6.8: Layout of a 16-input LTM element. The output consists of a 4-bit bus addressing a 4-bit memory cell (not shown). The weighted sum is implemented in the Neuron MOS fashion, as a capacitive sum of voltages. The chip was fabricated using the 2μ technology available from MOSIS.

Chapter 7 Conclusions

The present thesis studied the properties of linear threshold elements. Those can be viewed as Boolean inputs, Boolean output artificial neurons, computing the sign of a weighted sum of their inputs. Our contributions are three levels

- At the theoretical level, we defined new classes of functions such as $\widehat{LT}^{(d)}$ and LTM and classified their computational power.
- At the algorithmic level, we showed how to convert real weights to weights drawn from an arbitrary subset of the real numbers, e.g., integer weights, we also showed how to construct LT functions with minimal weights, and finally we presented an algorithm that produces the $\widehat{LT}_2$ circuit that computes $COMP$. We also presented LTM circuits computing useful functions, such as XOR, ADD, $PRODUCT$.
- At the implementation level, we showed the design, layout and testing of the VLSI implementation of LT and LTM. We designed a programmable LT element that uses floating gate technology in order to store the values of the weights.

In Chapter 2 we showed some well known results in the theory of threshold circuits, in particular, that any linear threshold element can be implemented with integer weights. Our contribution is a generalization of that result, to an arbitrary set of real numbers. The conditions that allow any LT function to be implemented were derived, along with an algorithm for converting the weights. Chapter 3 presented a method for constructing linear threshold functions with minimal weights. It was used to establish the separation between the classes $\widehat{LT}^{(d)}$, indexed by d. Given an integer

d, the class $\widehat{LT}^{(d)}$ is defined as the set of functions that can be implemented with weights of $O(n^d)$. In Chapter 4 a well known result was presented, i.e., the fact that a single LT element with large weights can be implemented by a two-layer circuit composed of $\widehat{LT}$ elements, that is, linear threshold elements with small weights. Our contribution is the explicit construction of those circuits for the comparison function, $COMP$. Chapter 5 introduces, LTM, or linear threshold element with multiple thresholds. It presents constructions for useful Boolean functions, such as XOR, ADD, $PRODUCT$, along with an estimation of the power of LTM relative to LT and its derived classes, $\widehat{LT}$, $\widehat{LT_2}$ and LT_2. Finally Chapter 6 describes the VLSI implementation of LT and LTM. Both hardwired and programmable solutions are presented. Weights are stored as the charge on a floating gate, and modified by tunneling and injection of electrons.

There remain many open questions and interesting directions for future investigation. For example, the relationship between the classes of functions described in this thesis and linear decision lists, [49]. The proof of Conjecture 5.1 is needed to complete the picture relating LTM to LT. From the algorithmic point of view, to develop efficient algorithms for converting or minimizing the weights appears to be a challenging problem. As far as hardware implementation, a long term goal is the integration of threshold elements as building blocks of logic design libraries, in particular field programmable gate arrays.

Appendix

```
function correct = test(n, r, t)

%% function correct = test(n, r, t)
%  Simulation of COMPARISON(X,Y)
%  correct = 1 if the construction works
%  n = number of bits in X (Y)
%  r = number of primes used
%  t = threshold used
%  V.Bohossian May, 96

BIG = 2^n;
correct = 1;
load primes.txt;          % The first 1000 primes
p = primes(3:r + 2);      % Remove 2 and 3
hp = fix(p / 2) + 0.1;    % hp : half p
p = p * ones(1, n);       % Duplicate columns
hp = hp * ones(1, n);
for i=1:n, L(i) = 2 ^ (i - 1);end;

for x = 0 : 2 ^ n - 1,
for y = 0 : 2 ^ n - 1,
     Ax = fix((x * ones(1, n)) ./ L);
     Ay = fix((y * ones(1, n)) ./ L);
     A = Ax - Ay;
     A = rem(ones(r, 1) * A + BIG * p, p);
     A = A + ((sign(hp - A) - 1) / 2) .* p;
     positives = size(find(~(A - 1)),1);
     bit = (positives > t);
```

```
     correct = correct & (bit == (x > y));
end;
end;

return;
```

Bibliography

[1] E. Allender. A note on the power of threshold circuits. *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pages 580–584, 1989.

[2] N. Alon and J. Bruck. Explicit constructions of depth-2 majority circuits for comparison and addition. *SIAM Journal of Discrete Mathematics*, 7(1):1–8, February 1994.

[3] E. Amaldi and V. Kann. The complexity and approximability of finding maximum feasible subsystems of linear relations. *Ecole Polytechnique Federale De Lausanne Technical Report*, ORWP 93/11, August 1993.

[4] J.J. Amodei, R.O. Winder, D. Hampel, and T.R. Mayhew. Digital circuit techniques. *International Solid-State Circuits Conference*, February 1967.

[5] P.W. Beame, S.A. Cook, and H.J. Hoover. Log depth circuits for division and related problems. *Proceedings of the 25th IEEE Symposium on Foundations of Computer Science*, pages 1–6, 1984.

[6] V. Bohossian and J. Bruck. Algebraic techniques for constructing minimal weight threshold functions. *Submitted to SIAM Journal of Discrete Mathematics.* Available at http://paradise.caltech.edu/ETR.html.

[7] V. Bohossian and J. Bruck. Multiple threshold neural logic. *Advances in Neural Information Processing Systems*, 10, 1998. Available at http://paradise.caltech.edu/ETR.html.

[8] V. Bohossian, P. Hasler, and J. Bruck. Programmable neural logic. *Proceedings of the second annual IEEE International Conference on Innovative Systems in Silicon*, pages 13–21, 1997. Available at http://paradise.caltech.edu/ETR.html.

[9] J. Bruck. Harmonic analysis of polynomial threshold functions. *SIAM Journal of Discrete Mathematics*, 3(2):168–177, May 1990.

[10] C. Diorio, S. Mahajan, P. Hasler, B.A. Minch, and C. Mead. A high resolution non-volatile analog memory cell. *Proceedings of the International Conference of Circuits and Systems*, 3:2233–2236, 1995.

[11] R. Douglas, M. Mahowald, and C. Mead. Neuromorphic analogue VLSI. *Annual Reviews in Neuroscience*, 18:255–281, 1995.

[12] M. Goldmann, J. Hastad, and A. Razborov. Majority gates vs. general weighted threshold gates. *Computational Complexity*, 2:277–300, 1992.

[13] M. Goldmann and M. Karpinski. Simulating threshold circuits by majority circuits. *Proceedings of the 25th ACM Symposium on the Theory of Computing*, pages 551–56, 1993.

[14] A. Hajnal, W. Maass, P. Pudlak, M. Szegedy, and G. Turan. Threshold circuits of bounded depth. *Journal of Computer and System Sciences*, 46(2):129–154, April 1993.

[15] D.R. Haring. Multi-threshold threshold elements. *IEEE Transactions on Electronic Computers*, 15(1), February 1966.

[16] P. Hasler, C. Diorio, B.A. Minch, and C. Mead. Single transistor learning synapses. *Advances in Neural Information Processing Systems*, pages 817–824, 1995.

[17] J. Hastad. On the size of weights for threshold gates. *SIAM Journal of Discrete Mathematics*, 7:484–492, 1994.

[18] T. Hofmeister. A note on the simulation of exponential threshold weights. *COCOON conference*, 1996.

[19] M. Holler, S. Tam, H. Castro, and R. Benson. An electrically trainable artificial neural network with 10240 'floating gate' synapses. *International Joint Conference on Neural Networks*, II:191–196, June 1989.

[20] J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the USA National Academy of Sciences*, 79:2554–2558, 1982.

[21] A.A. Irmatov. On the number of threshold functions. *Diskretnaya Matematika (Russian)*, 5(3):40–43, 1993.

[22] A.A. Irmatov. Estimations of the number of threshold functions. *Discrete Mathematics and Applications*, 6(6):569–583, 1996.

[23] J. Kahn, J. Komlós, and E. Szemerédi. On the probability that a random $\{\pm 1\}$-matrix is singular. *Journal of the American Mathematical Society*, 8(1):223–240, 1995.

[24] W.H. Kautz. The realization of symmetric switching functions with linear–input logical elements. *IRE Transactions on Electronic Computers*, March 1961.

[25] M. Krause and P. Pudlak. On computing boolean functions by sparse real polynomials. *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 682–691, October 1995.

[26] H.W. Kuhn and A.W. Tucker. Linear inequalities and related systems. *Annals of Mathematics Studies*, 38, 1956. Princeton University Press, Princeton, NJ.

[27] H.W. Kuhn and A.W. Tucker. On systems of linear inequalities. *Linear Inequalities and Related Systems, Annals of Mathematics Studies*, 38:99–156, 1957. Princeton University Press, Princeton, NJ.

[28] R. Lauwereins and J. Bruck. Efficient implementation of a neural multiplier. *IBM Research Report*, RJ 8138 (74551), May 30, 1991.

[29] C. Mead. Analog VLSI and neural systems. *Addison-Wesley*, 1989.

[30] B.A. Minch, C. Diorio, P. Hasler, and C. Mead. Translinear circuits using subthreshold floating-gate MOS-transistors. *Analog integrated circuits and signal processing*, 9(2):167–179, March 1996.

[31] R.C. Minnick. Linear - input logic. *IRE Transactions on Electronic Computers*, March 1961.

[32] M. Muroga. Threshold logic and its applications. *Wiley-Interscience*, 1971.

[33] J. Myhill and W.H. Kautz. On the size of weights required for linear-input switching functions. *IRE Transactions on Electronic Computers*, 10:288–290, 1961.

[34] A.M. Odlyzko. On subspaces spanned by random selections of ± 1 vectors. *Journal of Combinatorial Theory*, Series A(47):124–133, 1988.

[35] S. Olafsson and Y.S. Abu-Mostafa. The capacity of multilevel threshold functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(2), March 1988.

[36] D.T. Perkins, D.G. Willis, and E.A. Whitmore. Division of space by concurrent hyperplanes. *Internal Report, Missile and Space Division*, 1959. Lockheed Aircraft Corporation, Sunnyvale, California.

[37] D. Rumelhart and J. McClelland. Parallel distributed processing: Explorations in the microstructure of cognition. *MIT Press*, 1982.

[38] J.S. Shawe-Taylor, M.H.G. Anthony, and W. Kern. Classes of feedforward neural networks and their circuit complexity. *Neural Networks*, 5:971–977, 1992.

[39] T. Shibata, K. Kotania, and T. Ohmi. Real-time reconfigurable logic circuits using neuron MOS transistors. *International Solid-State Circuits Conference*, 1993.

[40] T. Shibata and T. Ohmi. A functional MOS transistor featuring gate-level weighted sum and threshold operations. *IEEE Transactions on Electron Devices*, 39(6), June 1992.

[41] T. Shibata and T. Ohmi. Neuron MOS binary-logic integrated circuits – part I: Design fundamentals and soft-hardware-logic circuit implementation. *IEEE Transactions on Electron Devices*, 40(3), March 1993.

[42] L. Shläfli. Gesamelte mathematische abhandlugen. *Band 1*, 1850. Basel: Verlag Birkhäuzer.

[43] K. Siu and J. Bruck. On the power of threshold circuits with small weights. *SIAM Journal of Discrete Mathematics*, 4(3):423–435, August 1991.

[44] K. Siu, J. Bruck, T. Kailath, and T. Hofmeister. Depth efficient neural networks for division and related problems. *IEEE Transactions on Information Theory*, 39(3):423–435, May 1993.

[45] K. Siu and V.P. Roychowdhury. On optimal depth threshold circuits for multiplication and related problems. *SIAM Journal of Discrete Mathematics*, 7(2):284–292, May 1994.

[46] D.R. Smith. Bounds on the number of threshold functions. *IEEE Transactions on Electronic Computers*, June 1966.

[47] M. Szegedy. Algebraic methods in lower bounds for computational models with limited communication. *PhD Thesis*, 1989. Chicago, Illinois.

[48] T. Tich-Dao. Threshold I^2L and its applications to binary symmetric functions and multivalued logic. *IEEE Journal of Solid-State Circuits*, 12(5), October 1977.

[49] G. Turán and F. Vatan. Linear decision lists and partitioning algorithms for the construction of neural networks. 1997.

[50] J. Villasenor and W.H. Mangione-Smith. Configurable computing. *Scientific American*, pages 66–71, June 1997.

[51] I. Wegener. The complexity of the parity function in unbounded fan-in unbounded depth circuits. *Theoretical Computer Science*, 85:155–170, 1991.

[52] D.G. Willis. Minimum weights for threshold switches. *Switching Theory in Space Techniques*, 1963. Stanford University Press.

[53] B.A. Wooley and C.R. Baugh. An integrated m-out-of-n detection circuit using threshold logic. *IEEE Journal of Solid-State Circuits*, 9(5), October 1974.

[54] S. Yajima and T. Ibaraki. A lower bound on the number of threshold functions. *IEEE Transactions on Electronic Computers*, 14(6):926–929, December 1965.

[55] K. Yang and A.G. Andreou. The multiple input floating gate MOS differential amplifier: An analog computational building block. *IEEE ISCAS*, 5, 1994.

[56] A.C. Yao. On ACC and threshold circuits. *Proceedings of the 31th IEEE Symposium on Foundations of Computer Science*, pages 619–627, 1990.

[57] T. Zaslavsky. Facing up to arrangements: Face-count formulas for partitions of space by hyperplanes. *Journal of the American Mathematical Society*, 154, Providence, RI, 1975.

[58] Y.A. Zuev. Methods of geometry and probabilistic combinatorics in threshold logics. *Discrete Mathematics*, pages 427–438, Appl. 2, 1992.