Circuit Complexity and Feedforward Neural Networks

Ian Parberry*
Department of Computer Sciences
University of North Texas

Abstract

Circuit complexity, a subfield of computational complexity theory, can be used to analyze how the resource usage of neural networks scales with problem size. The computational complexity of discrete feedforward neural networks is surveyed, with a comparison of classical circuits to circuits constructed from gates that compute weighted majority functions.

1 Introduction

Computation consumes resources, including time, memory, hardware, and power. A theory of computation, called *computational complexity theory*[1] has grown from this simple observation, starting with the seminal paper of Hartmanis and Stearns [14]. The prime tenet of this field is that some computational problems intrinsically consume more resources than others. The resource usage of a computation is measured as a function of the size of the problem being solved, that is, the number of bits needed to encode the input. The idea is that as science and technology progresses the amount of data that we must deal with will grow rapidly with time. We not only need to be able to solve today's technological problems, but also to be able to *scale up* to larger problems as our needs grow and larger and faster computers become available. The goal of computational complexity theory is to develop algorithms that are scalable in the sense that the rate of growth in their resource requirements does not outstrip the ability of technology to supply them.

An important contribution of neural networks is their capacity for efficient computation. The first computers were created in rough analogy with the brain, or more correctly, in rough analogy with what was believed about the brain at the time by a certain group of people. Although technology has advanced greatly in recent decades, modern computers are little different from their older counterparts. It is felt by some scientists that in order to produce better computers we must return to the brain for further inspiration.

It is important to determine which features of the brain are crucial to efficient computation, and which features are merely by-products or side-effects. It is unlikely that a computer that is comparable in computing power to the brain can be obtained by merely simulating its observed behaviour, simply because the overhead is too great. The general principles of brain computation must be understood before we try to implement an artificial system that exhibits them.

*Author's address: Department of Computer Sciences, University of North Texas, P.O. Box 13886, Denton, TX 76203-3886, U.S.A. Electronic mail: `ian@ponder.csci.unt.edu`.

[1]Computational complexity theory should not be the confused with the more recent science of complexity studied by physicists.

Another important contribution of neural networks is their capacity for learning from experience. While computational learning theory is outside the scope of this short survey, what we will learn here will have some relevance to learning because neural network computation is a necessary part of the foundations of neural network learning. Just as a child cannot learn to perform a task unless he or she is physically capable of performing it, a neural network cannot learn to compute a function unless it is physically capable of computing it. "Physically capable" in this context means "possessing sufficient resources".

Circuit complexity is a subfield of computational complexity theory that deals with efficient computation by networks of simple processing units. In this paper we survey some results in circuit complexity that cast some light on how neural networks scale. Our main theme is the following: the computational power of neural networks is contingent on (but not limited to) the following features which distinguish them from conventional von Neumann style computers:

- Parallelism: processing elements may compute in parallel.
- Large fan-in: the number of inputs to each gate is not limited to 2 (as is current technology), but may scale with the size of the problem being solved.
- Node function: gates may compute weighted majority functions.

For more details on the subject matter of this paper, and the application of other branches of computational complexity theory to neural networks, see Parberry [29, 30].

The remainder of this paper is divided into four sections. The first considers a simple feedforward neural network model. The second examines computation with a version of this model that has node functions limited to Boolean conjunction, disjunction, and complement. The third considers computation by such networks with the restriction that the number of nodes cannot increase exponentially. The fourth extends the node function set to include weighted majority functions, a popular discrete neural network model that is well-studied in the literature. The fifth sketches some variations on the model.

In the remainder of this paper, $\mathbb{B}$ denotes the Boolean set $\{0,1\}$, $\mathbb{N}$ denotes the natural numbers, $\mathbb{R}$ denotes the real numbers, and $\mathbb{R}^+$ denotes the positive real numbers. If V is a finite set, $\|V\|$ denotes the number of members of V. If $x \in \mathbb{R}^+$, $\lfloor x \rfloor$ denotes the largest natural number not exceeding x, and $\lceil x \rceil$ denotes the smallest natural number not less than x. All logarithms are to base two unless otherwise indicated.

If $f, g\colon \mathbb{N} \to \mathbb{N}$, $f(n)$ is said to be $O(g(n))$ if there exists $c, n_0 \in \mathbb{N}$ such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$. If $f, g\colon \mathbb{N} \to \mathbb{N}$, $f(n)$ is said to be $\Omega(g(n))$ if there exists $c \in \mathbb{N}$ such that for infinitely many values of n, $f(n) \geq c \cdot g(n)$.

If S is a set, S^n denotes the n-fold Cartesian product of S,

$$\underbrace{S \times S \times \cdots \times S}_{n \text{ times}} = \{(s_1, \ldots, s_n) \mid s_i \in S \text{ for } 1 \leq i \leq n\},$$

and S^* denotes $\cup_{n \in \mathbb{N}} S^n$.

A *directed graph* is an ordered pair $G = (V, E)$, where V is a finite set of *nodes* and $E \subseteq V \times V$ is a set of *edges*. An edge $(u, v) \in E$ is said to be directed from u to v. A *cycle* in a graph $G = (V, E)$ is a sequence of vertices $v_1, \ldots, v_n$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i < n$, and $(v_n, v_1) \in E$. A *directed, acyclic graph* is a directed graph that has no cycles.

This paper contains a preliminary version of material to appear in Parberry [30]. Many of the proofs in this paper are truncated or omitted entirely to save space. The reader who wishes to read a more detailed account may refer to [30]. The reader who is not interested in the proofs may skip them entirely and still gain something from the remainder of the paper.

2 Feedforward Neural Networks

The general framework used by neural network researchers is a finite network of simple computational devices wired together so that they interact and cooperate to perform a computation (see, for example, Rumelhart, Hinton, and McClelland [36]).

Suppose $\mathcal{F}$ is a set of functions $f : \mathbb{B}^* \rightarrow \mathbb{B}$. A *feedforward neural network* with node function set $\mathcal{F}$ is a 5-tuple $C = (V, X, Y, E, \ell)$, where

V is a finite ordered set of *gates*
X is a finite ordered set of *inputs*, $X \cap V = \emptyset$
$Y \subseteq V \cup X$ is a set of *outputs*
$(V \cup X, E)$ is a directed, acyclic graph called the the *interconnection graph* of C.
$\ell : V \rightarrow \mathcal{F}$ determines the function computed by each gate.

A gate $g \in V$ will be referred to as an $\ell(g)$-gate. The set of node functions $\mathcal{F}$ is typically one of the following:

1. Binary conjunction, binary disjunction, and unary negation.

   The binary conjunction function is the function $\text{AND} : \mathbb{B}^2 \rightarrow \mathbb{B}$ defined by
   $$\text{AND}(x_1, x_2) = x_1 \wedge x_2,$$
   that is, the function that is 1 iff both of its inputs are 1.

   The binary disjunction function is the function $\text{OR} : \mathbb{B}^2 \rightarrow \mathbb{B}$ defined by
   $$\text{OR}(x_1, x_2) = x_1 \vee x_2,$$
   that is, the function that is 1 iff at least one of its inputs is 1.

   The unary negation function is the function $\text{NOT} : \mathbb{B} \rightarrow \mathbb{B}$ defined by
   $$\text{NOT}(x_1, x_2) = \neg x_1,$$
   that is, the function that is 1 iff its input is 0.

   Circuits with this node function set will be called *classical circuits.*

2. Conjunction and disjunction of an arbitrary number of inputs, and unary negation.

   The conjunction function is the function $\text{AND} : \mathbb{B}^{\text{n}} \rightarrow \mathbb{B}$ defined by
   $$\text{AND}(x_1, \ldots, x_n) = x_1 \wedge x_2 \wedge \cdots \wedge x_n,$$
   that is, the function that is 1 iff all of its inputs are 1.

   The disjunction function is the function $\text{OR} : \mathbb{B}^2 \rightarrow \mathbb{B}$ defined by
   $$\text{OR}(x_1, x_2) = x_1 \vee x_2 \vee \cdots \vee x_n,$$
   that is, the function that is 1 iff at least one of its inputs is 1.

   Circuits with this node function set will be called *AND-OR circuits.*

3. Majority of an arbitrary number of inputs, and unary negation.

   The majority function is the function MAJORITY: $\mathbb{B}^n \rightarrow \mathbb{B}$ defined by

$$\text{MAJORITY}(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i \geq n/2 \\ 0 & \text{otherwise} \end{cases}$$

   that is, the function that is 1 iff at least half of its inputs are 1.

   Circuits with this node function set will be called *threshold circuits.*

4. Weighted majority of an arbitrary number of inputs.

   The weighted majority function[2] is the function WMAJORITY: $\mathbb{B}^n \rightarrow \mathbb{B}$ defined by

$$\text{WMAJORITY}(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq h \\ 0 & \text{otherwise} \end{cases}$$

   for some $w_1, \ldots, w_n, h \in \mathbb{R}$.

   Circuits with this node function set will be called *weighted threshold circuits.*

Note that each of these node function sets includes the previous set.

Let $C = (V, X, Y, E, \ell)$ be a feedforward neural network, where $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$. For each $b_1, \ldots, b_n \in \mathbb{B}$, define the *value* of gate g of circuit C on input $b_1, \ldots, b_n$, denoted $v_C(b_1, \ldots, b_n)(g)$, as follows. If $g = x_i$ for some $1 \leq i \leq n$, define $v_C(b_1, \ldots, b_n)(g) = b_i$. If $g \in V$, and $P = \{g_1, \ldots, g_m\} = \{g' \mid (g', g) \in E\}$, then $v_C(b_1, \ldots, b_n)(g) = \ell(g)(g_1, \ldots, g_m)$. The *output* of C on inputs $b_1, \ldots, b_n \in \mathbb{B}$ is defined to be $v_C(b_1, \ldots, b_n)(y_1), \ldots, v_C(b_1, \ldots, b_n)(y_m)$. An n-input feedforward neural network $C = (V, X, Y, E, \ell)$ is said to *compute* a Boolean function $f: \mathbb{B}^n \rightarrow \mathbb{B}^m$ if for all $b_1, \ldots, b_n \in \mathbb{B}$, the output of C on input $b_1, \ldots, b_n$ is $f(b_1, \ldots, b_n)$.

For example, Figure 1 shows a classical circuit $C = (V, X, Y, E, \ell)$, where

$$\begin{aligned} V &= \{g_1, g_2, g_3, g_4, g_5, g_6, g_7, g_8, g_9\} \\ X &= \{x_1, x_2, x_3, x_4\} \\ Y &= \{g_9\} \\ E &= \{(x_1, g_1), (x_2, g_1), (x_2, g_2), (x_3, g_2), (x_3, g_3), (x_4, g_4), \\ &\quad (g_1, g_4), (g_2, g_6), (g_2, g_5), (g_3, g_7), (g_4, g_6), (g_5, g_7), (g_6, g_8), (g_7, g_8), (g_8, g_9)\} \end{aligned}$$

$$\begin{array}{lll} \ell(g_1) = \text{AND} & \ell(g_4) = \text{NOT} & \ell(g_7) = \text{OR} \\ \ell(g_2) = \text{AND} & \ell(g_5) = \text{NOT} & \ell(g_8) = \text{AND} \\ \ell(g_3) = \text{OR} & \ell(g_6) = \text{OR} & \ell(g_9) = \text{NOT} \end{array}$$

C computes the function

$$\neg((\neg(x_1 \wedge x_2) \vee (x_2 \wedge x_3)) \wedge (\neg(x_2 \wedge x_3) \vee (x_3 \vee x_4))).$$

The *size* of a feedforward neural network $C = (V, X, Y, E, \ell)$ is defined to be $\|V\|$, the number of gates. The *depth* is defined to be the maximum number of gates in V on any path from an input to an output. The gates in a circuit of depth d can be partitioned in d *levels* or *layers.* The inputs X are said to be at *level* 0. A gate $v \in V$ is said to be at *level* $i > 0$ if

[2]Weighted majority functions are often called the *linear threshold functions.*

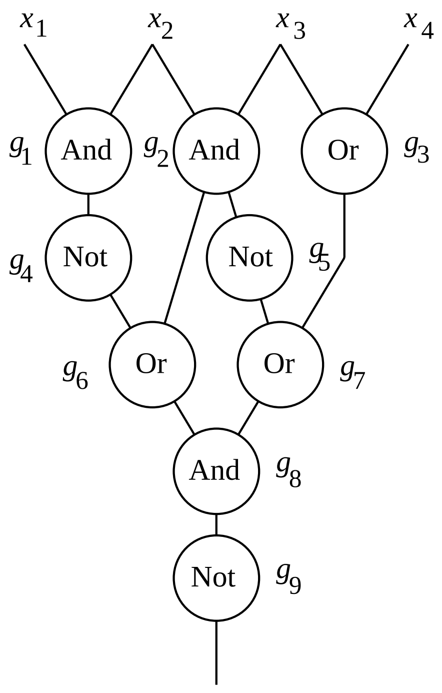


Figure 1: A classical circuit.

1. for all $u \in V \cup X$ such that $(u, v) \in E$, u is at a level less than i, and
2. there exists $u \in V \cup X$ at level $i - 1$ such that $(u, v) \in E$.

There is little attention paid in the literature to how these finite feedforward neural networks scale to larger problems. When one designs a circuit to solve a given task, such as performing pattern recognition on an array of pixels, one typically starts with a small number of inputs and eventually hopes to scale up the solution to real life situations. How the resources of the circuit scale as the number of inputs increases is of prime importance. A good abstraction of this process is to imagine a potentially infinite series of circuits, $C = (C_1, C_2, \ldots)$, where for $n \in \mathbb{N}$, C_n has n inputs, and to measure the increase in resources from one circuit in the series to the next. The *size* of a circuit family $C = (C_1, C_2, \ldots)$ is said to be $Z(n)$ if for all $n \in \mathbb{N}$, the size of C_n is at most $Z(n)$. The *depth* of C is said to be $D(n)$ if for all $n \in \mathbb{N}$, the depth of C_n is at most $D(n)$.

There is an apparent flaw in our abstraction. Since for every natural number n, every Boolean function with n inputs can be computed by a finite classical circuit, our infinite-family-of-finite-circuits model can compute any Boolean function:

Theorem 2.1 *Every Boolean function $f : \mathbb{B}^* \rightarrow \mathbb{B}$ can be computed by a classical circuit family.*

PROOF: It is sufficient to show that for all $n \in \mathbb{N}$ and every Boolean function $f : \mathbb{B}^{\mathrm{n}} \rightarrow \mathbb{B}$, there is a classical circuit that computes f. The proof is by induction on n. The Theorem is certainly true for $n = 1$ (in which case there are only four functions to consider, the always `true` function, the always `false` function, the identity function, and the complement function, each of which can be realized with at most one gate).

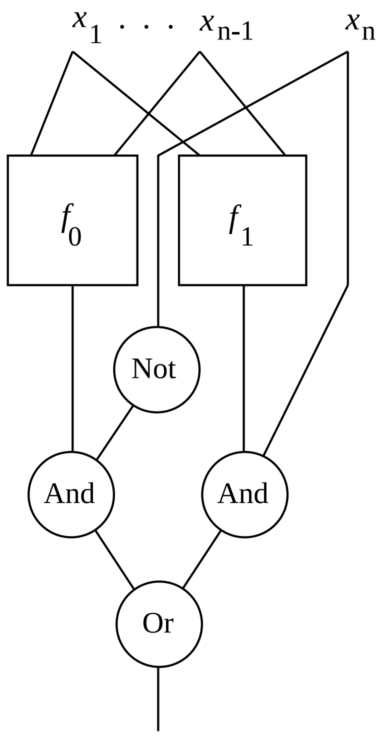


Figure 2: Circuit for computing f in the proof of Theorem 2.1.

Suppose that every function on $n-1$ inputs can be computed by a classical circuit. Define $f_0, f_1 : \mathbb{B}^{n-1} \rightarrow \mathbb{B}$ as follows:

$$\begin{aligned} f_0(x_1, \ldots, x_{n-1}) &= f(x_1, \ldots, x_{n-1}, 0) \\ f_1(x_1, \ldots, x_{n-1}) &= f(x_1, \ldots, x_{n-1}, 1) \end{aligned}$$

Then

$$\begin{aligned} f(x_1, \ldots, x_n) &= (f_0(x_1, \ldots, x_{n-1}) \wedge (x_n = 0)) \vee (f_1(x_1, \ldots, x_{n-1}) \wedge (x_n = 1)) \\ &= (f_0(x_1, \ldots, x_{n-1}) \wedge \neg x_n) \vee (f_1(x_1, \ldots, x_{n-1}) \wedge x_n). \end{aligned}$$

Since f_0 and f_1 can be computed by a classical circuit (by the induction hypothesis), f can be computed by the classical circuit shown in Figure 2. $\square$

By Theorem 2.1, our feedforward neural network model can compute the halting problem (given the Gödel number of a Turing machine and an input, determine whether the Turing machine halts on that input). Hence, our model can compute more functions than a Turing machine. But the Church-Turing thesis states that all reasonable computational models compute the same class of functions. What is so unreasonable about our model? It is the fact that the interconnection pattern (V, E) and the gate assignment function ℓ can be noncomputable functions. Let us call the interconnection pattern and gate assignment functions collectively the *architecture* of a neural network. In practice we will probably use computers to manufacture neural networks, and thus is would be reasonable to insist that the architectures be computable. That is, we could insist that the architecture of each finite circuit in the infinite series be similar to its predecessor in the series in the sense that a Church-Turing style computer can compute the differences between the two. This type of circuit is called a *uniform* circuit (whereas the unrestricted model is called a *nonuniform* circuit).

However, from a theoretical point of view, a circuit family with a noncomputable interconnection pattern might still be interesting. Knowing that a circuit families for a given function exists may be important information, even though there may be no algorithm for constructing it. However, Theorem 2.1 is of no practical use since the circuits it constructs have exponential size, and therefore can only be used for very small values of n. Can we hope to reduce the size bound in Theorem 2.1 from an exponential to a polynomial? The answer is no. This is due to the fact that some Boolean functions *require* exponential size (see Theorem 3.5). It is more interesting to ask instead which Boolean functions can be computed with nonuniform circuits of only polynomial size. Sometimes it may be useful to know that polynomial size circuits exist even if it is not known how to construct them.

3 Alternating Circuits

The philosophy behind classical circuit families is the following. It assumes that as technology improves, we will be able to construct circuits of increasingly larger size. At the same time, it assumes that gate technology will remain fixed, and in particular, that the number of inputs to each gate (called the *fan-in*) will not increase as time passes. However, in the human brain the fan-in is extremely large compared to the number of neurons (approximately 10^{10} neurons and fan-in of up to 10^5). It may be that the incredible computing power of the brain comes from this high fan-in. If we are to eventually reach fan-ins this large, we will probably do it by scaling the fan-in as well as the size. The AND-OR circuit models a scenario in which technological advances allow the construction of larger circuits as time passes, and that the technology is developed in such a way that fan-in keeps pace with size. AND-OR circuits are very similar to classical circuits, but the fan-in may grow with input size instead of being limited to 2. In each finite circuit it is clearly bounded above by the number of gates, but it may increase from one circuit in the family to the next.

In this model it is not strictly necessary to have the NOT-gates scattered arbitrarily throughout the circuit if a constant factor increase in size is permitted:

Theorem 3.1 *For every n-input AND-OR circuit of depth d and size s there exists an equivalent n-input AND-OR circuit of depth at most d and size at most $2s + n$ in which all of the* NOT-*gates are at level 1.*

Theorem 3.1 allows us to put all AND-OR circuits into a useful kind of normal form. An *alternating circuit* is an AND-OR circuit in which all of the gates in any given layer compute the same function, and the layers (apart from the first) alternate between gates computing AND and gates computing OR as we go down the circuit (that is, the even numbered layers compute AND and the odd numbered layers compute OR, or vice-versa):

Corollary 3.2 *For every n-input AND-OR circuit of size s and depth d there is an equivalent alternating circuit of size at most $2s + n$ and depth at most d.*

We will use alternating circuits in preference to AND-OR circuits from this point onwards. Let us redefine the depth and size of an alternating circuit to exclude layer 1 (which consists of NOT-gates). It is convenient to think of an alternating circuit as being a function of a set of *literals*, where a literal is either an input or its complement. Our motivation is based primarily on the desire for a cleaner model, but we are not totally divorced from reality, since NOT-gates are relatively cheap compared to AND and OR-gates (particularly since we have placed no bound on the fan-in of the latter). Omitting the NOT-gates can only have a relatively small effect on the size of alternating circuits, since they can have most n NOT-gates.

One can prove that any function with output dependent on all inputs requires a classical circuit of depth at least $\log_2 n$. In contrast, alternating circuits can compute any function in constant depth.

Theorem 3.3 *For all $f : \mathbb{B}^n \to \mathbb{B}$, there is an alternating circuit of size $2^{n-1}+1$ and depth 2 that computes f.*

PROOF: (Outline) Express f in either disjunctive normal form (a disjunction of conjunctions of literals) or CNF (a conjunction of disjunctions of literals). It can be shown that if the former has c conjunctions and the latter has d disjunctions, then $c + d \leq 2^n$. Therefore, one of c, d is at most 2^{n-1}, and therefore the circuit constructed from the appropriate formula has size $2^{n-1}+1$ and depth 2. □

Theorem 3.3 compares favourably with Theorem 2.1, which gives classical circuits of size $O(2^n)$ and depth $O(n)$. Unfortunately, the circuits constructed in both Theorems have exponential size (that is, size that grows exponentially with n), and hence cannot be considered a practical method for constructing circuits for all but the smallest values of n. It is interesting to ask whether exponential size is necessary. It is certainly necessary if we wish to maintain depth 2. In fact, Theorem 3.3 has optimal size for circuits of depth 2. Consider the following problem.

PARITY
INSTANCE: $x_1, \ldots, x_n \in \mathbb{B}$.
QUESTION: Is $\|\{i \mid x_i = 1\}\|$ odd?

Theorem 3.4 *Any depth 2 alternating circuit for computing* PARITY *must have size at least $2^{n-1}+1$.*

Theorem 3.4 is due to Lupanov [18, 19]. An obvious question to ask is whether we can reduce the size and trade it for increased depth. The answer is that this is not possible beyond a certain size: some functions intrinsically require exponential size circuits.

Theorem 3.5 *There exists a function that requires an alternating circuit of size $\Omega(2^{n/2})$.*

PROOF: (Outline) Count the number of circuits of a given size, and the the number of Boolean functions. The former must be no smaller than the latter. □

Can this size lower bound, which is smaller by a polynomial amount (actually, a square-root) than the upper bound of Theorem 3.3, be met? Surprisingly it can be met with a circuit of depth 3, a result that can be traced to Redkin [35].

Theorem 3.6 *If $f : \mathbb{B}^n \to \mathbb{B}$, then there is an alternating circuit of size $O(2^{n/2})$ and depth 3 that computes f.*

PROOF: Let $f : \mathbb{B}^n \to \mathbb{B}$. Without loss of generality, assume n is even (a similar approach will work when n is odd). We will construct a circuit for f using a standard divide-and-conquer technique.

For each $x_1, \ldots, x_{n/2} \in \mathbb{B}$, define $g(x_1, \ldots, x_{n/2}) : \mathbb{B}^{n/2} \to \mathbb{B}$ by

$$g(x_1, \ldots, x_{n/2})(x_{n/2+1}, \ldots, x_n) = f(x_1, \ldots, x_n).$$

Each of the $2^{n/2}$ functions $g(x_1, \ldots, x_{n/2})$ for $x_1, \ldots, x_{n/2} \in \mathbb{B}$ can be computed by a single multi-output circuit of depth 2 and size $2^{n/2}+1$ with the first layer consisting of OR gates, and the second layer consisting of AND gates, using the technique of Theorem 3.3. Note that the resulting circuits each have size $2^{n/2} + 1$ giving a combined size of $2^n + 2^{n/2}$. However, there are only $2^{n/2}$ different OR functions of $n/2$ inputs. Therefore, the first layers of these circuits can be combined into a single layer that has at most $2^{n/2}$ OR gates. The resulting circuit $\mathcal{C}$ computes the $2^{n/2}$ functions in size $2^{n/2+1}$ by using $2^{n/2}$ OR gates in the first layer and $2^{n/2}$ AND gates in the second layer.

For each $b_1, \ldots, b_{n/2} \in \mathbb{B}$, define $h(b_1, \ldots, b_{n/2}) : \mathbb{B}^n \to \mathbb{B}$ by

$$\begin{aligned} h(b_1, \ldots, b_{n/2})(x_1, \ldots, x_n) &= (x_i = b_i \text{ for } 1 \leq i \leq n/2) \wedge \\ &\quad\; g(x_1, \ldots, x_{n/2})(x_{n/2+1}, \ldots, x_n). \end{aligned}$$

The circuit $\mathcal{C}$ constructed above can easily be modified to compute the $2^{n/2}$ functions $h(b_1, \ldots, b_{n/2})$ for $b_1, \ldots, b_{n/2} \in \mathbb{B}$ by simply taking the AND gate that computes

$$g(b_1, \ldots, b_{n/2})(x_{n/2}, \ldots, x_n),$$

and giving it extra inputs from $x_1[b_1], \ldots, x_{n/2}[b_{n/2}]$ (where $x_i[0]$ denotes $\overline{x}_i$, $x_i[1]$ denotes x_i, $\overline{x}_i[0]$ denotes x_i, and $\overline{x}_i[1]$ denotes $\overline{x}_i$). The resulting circuit still has depth 2 and size $2^{n/2+1}$.

Finally, we note that

$$f(x_1, \ldots, x_n) = h(\underbrace{0, \ldots, 0}_{n/2})(x_1, \ldots, x_n) \vee \cdots \vee h(\underbrace{1, \ldots, 1}_{n/2})(x_1, \ldots, x_n),$$

and f can therefore be computed by a circuit of depth 3 and size $2^{n/2+1} + 1$. □

For example, Figures 3 and 4 show the construction of a depth 3 circuit for a function f with 4 inputs and the input-output behaviour shown in Table 1. Figure 3 shows the two steps in the construction of the circuit for computing the functions $g(0, 0)$, $g(0, 1)$, $g(1, 0)$, $g(1, 1)$. Figure 4 shows the resulting circuit for f.

4 Polynomial Size Alternating Circuits

The problem with the methods for circuit design presented in the previous section is that they produce circuits whose size grows exponentially with n. Unfortunately, as we saw in Theorem 3.5, not all functions have polynomial size circuits. It is interesting to consider those that do.

Let $\mathcal{P}$ denote the set of decision problems which can be solved by an alternating circuit of polynomial size, that is, a circuit $C = (C_1, C_2, \ldots)$, where for some $c \in \mathbb{N}$ and all $n \geq 1$, the size of

x_1	x_2	x_3	x_4	$f(x_1,x_2,x_3,x_4)$	x_1	x_2	x_3	x_4	$f(x_1,x_2,x_3,x_4)$
0	0	0	0	0	1	0	0	0	0
0	0	0	1	1	1	0	0	1	0
0	0	1	0	0	1	0	1	0	0
0	0	1	1	0	1	0	1	1	1
0	1	0	0	0	1	1	0	0	1
0	1	0	1	1	1	1	0	1	1
0	1	1	0	0	1	1	1	0	0
0	1	1	1	1	1	1	1	1	0

Table 1: Truth table for a 4-input Boolean function f.

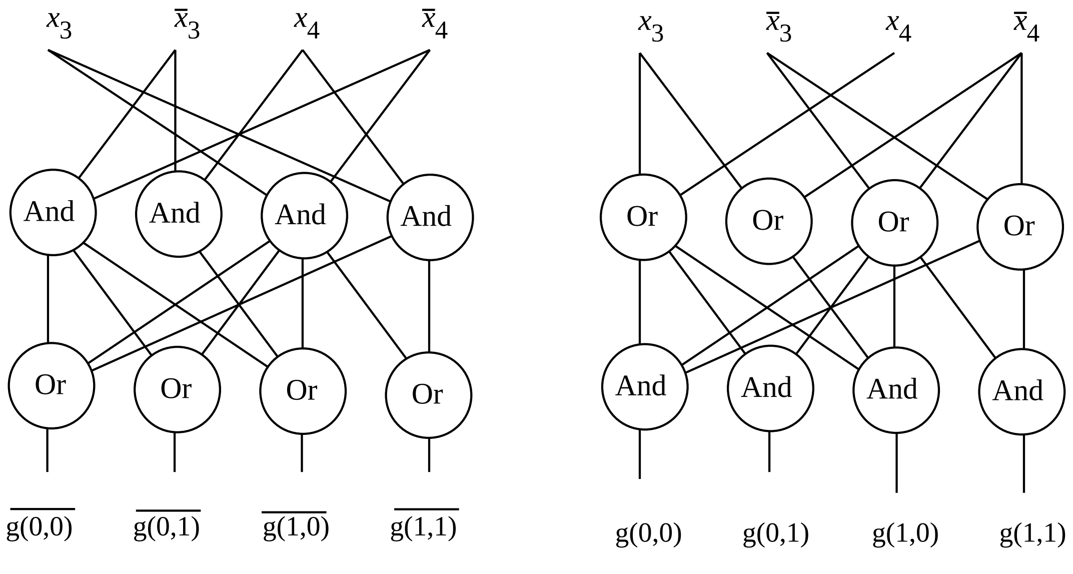


Figure 3: Construction of alternating circuit using Theorem 3.6 for the function f defined in Table 1. Left: Circuit for complement of $g(0,0)$, $g(0,1)$, $g(1,0)$, $g(1,1)$. Right: Circuit for $g(0,0)$, $g(0,1)$, $g(1,0)$, $g(1,1)$.

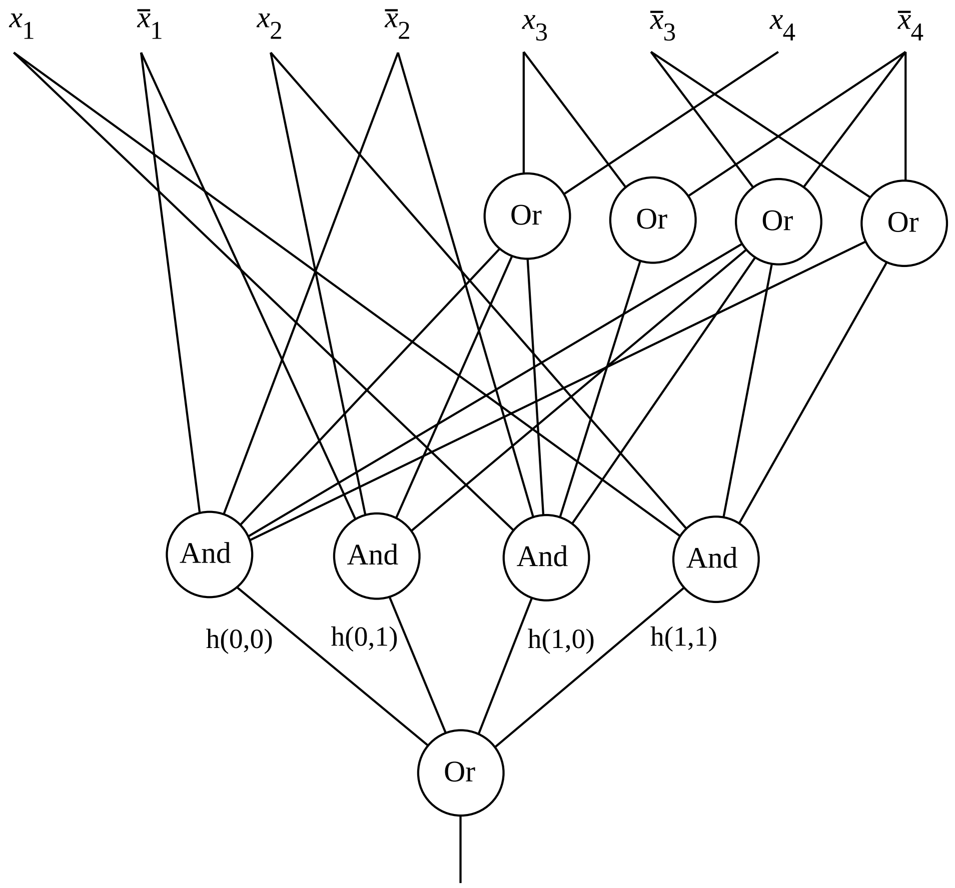


Figure 4: Construction of alternating circuit using Theorem 3.6 for the function f defined in Table 1. The top two layers of the circuit compute $h(0,0)$, $h(0,1)$, $h(1,0)$, $h(1,1)$, and should be compared with Figure 3. The third layer computes f.

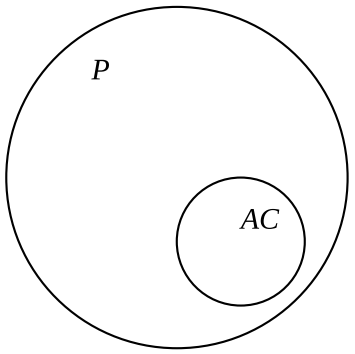


Figure 5: The classes $\mathcal{AC}$ and $\mathcal{P}$ (conjectured).

C_n is bounded above by n^c. It is easy to show that PARITY is a member of $\mathcal{P}$(see, for example, Theorem thm.parity.nc). We will call the problems in $\mathcal{P}$ *tractable* and those not in $\mathcal{P}$ *intractable.*

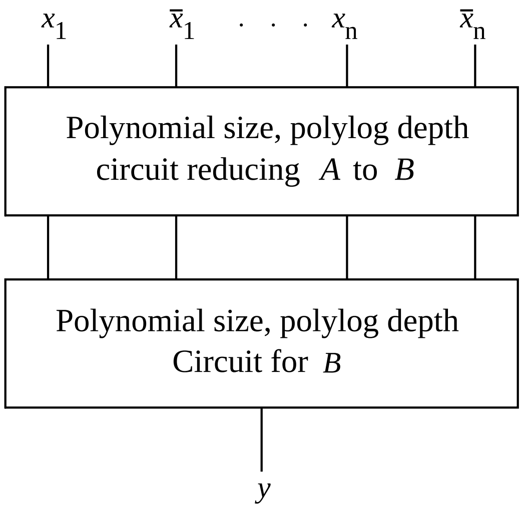
To enhance readability, we will describe decision problems not as being functions of a sequence of bits, but as functions of mathematical objects, wherever appropriate. This is reasonable, since all finite mathematical objects can be encoded as a sequence of bits. For example, an integer can be encoded in binary, a sequence of integers can be encoded by repeating each bit in the numbers (replacing 0 with 00, and 1 with 11 wherever it occurs) and separating each pair of integers by 01, and a set can be represented as a sequence of members. This approach is useful in that it adds a level of abstraction that insulates the reader from messy details at the bit level. However, there is a pitfall to be avoided here. If the encoding scheme is suitably sparse, then every function that is computable by an alternating circuit can be made a member of $\mathcal{P}$. For example, if a function $f:\mathbb{N}\to\mathbb{B}$ is computable in size 2^n when the input is encoded in binary, then simply encode it in unary. The number of gates will then be linear in the number of inputs since the number of inputs has been exponentially bloated. However, such trickery will not enable us to compute useful functions with a modest amount of hardware. It is reasonable to insist that inputs encode a sufficiently large amount of information about the mathematical objects in question. We will insist that the input encoding is sufficiently dense, that is, it is not more than polynomially larger than the tersest description of the input.

Whilst all problems in $\mathcal{P}$ have polynomial depth circuits (if the depth were greater than a polynomial, then so would the size be), it is interesting to consider which of them have depth exponentially smaller than size, that is, growing polynomially with $\log n$. Depth is of particular interest since it corresponds to the notion of time, assuming that gates are allowed to compute in parallel. We will use the notation $\log^c n$ to denote the function $(\log n)^c$, and use the term *polylog* to denote a function of the form $\log^c n$ for some $c \in \mathbb{R}$.

Let $\mathcal{AC}$ denote the set of decision problems which can be solved by an alternating circuit of polynomial size and polylog depth. The polynomial size condition ensures that $\mathcal{AC} \subseteq \mathcal{P}$, but it is unknown whether this containment is proper. It is widely conjectured that $\mathcal{P} \neq \mathcal{AC}$. Figure 5 shows the conjectured relationship between $\mathcal{AC}$ and $\mathcal{P}$.

Although it is not known for sure whether there is a problem in $\mathcal{P}$ that is not in $\mathcal{AC}$, there is a good candidate, in the sense that if any problems are in of $\mathcal{P}$ but not $\mathcal{NC}$, this is one of them:

CIRCUIT VALUE (CVP)
INSTANCE: An n-input alternating circuit C, and $x_1, \ldots, x_n \in \mathbb{B}$.



x_1
$\bar{x}_1$
. . .
x_n
$\bar{x}_n$
Polynomial size, polylog depth circuit reducing A to B
Polynomial size, polylog depth Circuit for B
y

Figure 6: The polynomial size, polylog depth circuit for A, given $A \leq_l B$ and $B \in \mathcal{AC}$.

QUESTION: What is the output of C on input $x_1, \ldots, x_n$?

We say that a problem A is *$\mathcal{AC}$-reducible* to problem B, written $A \leq_l B$, if there exists a function f computable by an alternating circuit of polynomial size and polylog depth such that for every x, $x \in A$ iff $f(x) \in B$.

Lemma 4.1 *If $A \leq_l B$, and $B \in \mathcal{AC}$, then $A \in \mathcal{AC}$.*

PROOF: Suppose $B \in \mathcal{AC}$, that is, there is a circuit for B of size n^b and depth $\log^{b'} n$, for some $b, b' \in \mathbb{N}$. Further suppose there is a circuit C of size n^c and depth $\log^{c'} n$, for some $c, c' \in \mathbb{N}$, which reduces A to B. A circuit for A can be obtained by combining the polynomial size, polylog depth circuit for B and the polynomial size, polylog depth circuit C which reduces A to B, as shown in Figure 6. Since C has size n^c, C has at most n^c outputs. Therefore the circuit for B has size n^{bc} and depth $c \log^{b'} n$, and so the entire circuit has polynomial size and polylog depth. $\Box$

We will say that a problem is *$\mathcal{P}$-hard* if every problem in $\mathcal{P}$ is $\mathcal{AC}$-reducible to it, and that it is *$\mathcal{P}$-complete* if it is $\mathcal{P}$-hard and a member of $\mathcal{P}$.

Theorem 4.2 *CVP is $\mathcal{P}$-complete.*

PROOF: It can be shown that CVP $\in \mathcal{P}$. It remains to show that CVP is $\mathcal{P}$-hard, that is, for all $A \in \mathcal{P}$, $A \leq_l$ CVP. Suppose $A \in \mathcal{P}$. Then there is a polynomial size circuit which recognizes A. It is easy to construct a circuit of polynomial size and constant depth (it consists purely of constant-gates which output either 0 or 1 regardless of their input) which inputs $x_1, \ldots, x_n$ and outputs a description of A with a copy of the input $x_1, \ldots, x_n$. The output is an instance of CVP which is a member of CVP iff $x \in A$. Therefore, $A \leq_l$ CVP. $\Box$

The uniform version of Theorem 4.2 is due to Ladner [17]. The proof in that reference is somewhat sketchy; a more detailed proof appears in Parberry [28]. $\mathcal{P}$-complete problems are

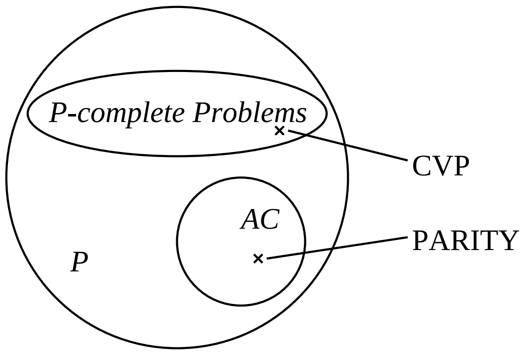


Figure 7: $\mathcal{P}$-complete problems (conjectured).

interesting since, by Lemma 4.1, if one of them is in $\mathcal{AC}$, then $\mathcal{AC} = \mathcal{P}$. If the conjecture that $\mathcal{AC} \neq \mathcal{P}$ is correct, then by Theorem 4.2, the circuit value problem requires polynomial depth if it is to be solved by polynomial size circuits, and Figure 7 reflects the true state of affairs.

Define $\mathcal{AC}^k$ to be the set of problems that can be solved in polynomial size, and depth $O(\log^k n)$, for $k \geq 0$. Clearly $\mathcal{AC}^k \subseteq \mathcal{AC}^{k+1}$ for $k \geq 0$, and

$$\mathcal{AC} = \cup_{k\geq 0}\mathcal{AC}^k.$$

The classes $\mathcal{AC}^k$ for $k \geq 1$ first appeared in Cook [8].

The relationship between classical circuits and AND-OR circuits is obvious.

Theorem 4.3 *For every finite alternating circuit of size s and depth d there is a finite classical circuit of size $s(s+n)$ and depth $d\lceil\log(s+n)\rceil$.*

PROOF: Let C be a finite alternating circuit of size s, depth d, and fan-in f. The new circuit C' is constructed by replacing every AND-gate in C with a tree of fan-in 2 AND gates of size f and depth $\lceil\log f\rceil$ in the obvious fashion, and similarly for OR-gates. Figure 8 shows the construction for $f = 8$. Since $f \leq s+n$, the result follows. □

There are bounded fan-in analogs of the complexity classes studied so far in this section. Define $\mathcal{NC}^k$ to be the set of problems that can be solved by classical circuits in polynomial size and depth $O(\log^k n)$, for $k \geq 1$. Clearly $\mathcal{NC}^k \subseteq \mathcal{NC}^{k+1}$ for $k \geq 1$. Define

$$\mathcal{NC} = \cup_{k\geq 0}\mathcal{NC}^k.$$

$\mathcal{NC}$ is an abbreviation of "Nick's Class", named by Cook [9] after Pippenger, who discovered an important relationship between $\mathcal{NC}$ and conventional Turing machine based computation [33].

Corollary 4.4 *1. For $k \geq 0$, $\mathcal{NC}^k \subseteq \mathcal{AC}^k$.*

2. For $k \geq 0$, $\mathcal{AC}^k \subseteq \mathcal{NC}^{k+1}$.

3. $\mathcal{NC} = \mathcal{AC}$.

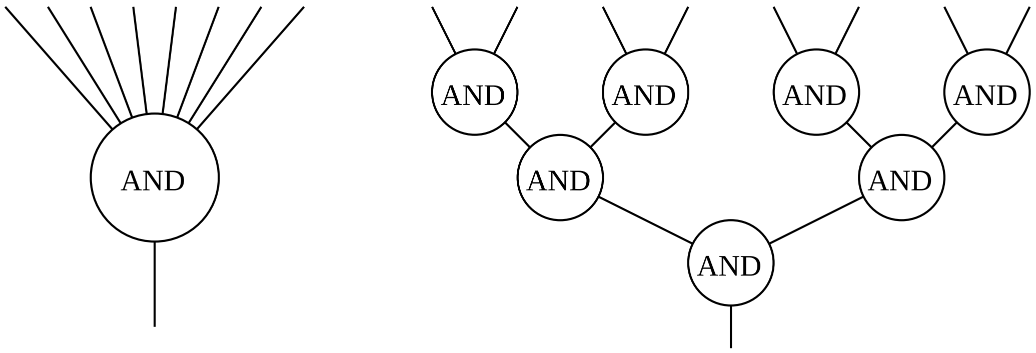


Figure 8: Replacing a fan-in 8 AND-gate with a tree of fan-in 2 gates.

Proof: Part (1) is obvious, since fan-in 2 circuits are a special case of unbounded fan-in circuits. Part (2) is a corollary of Theorem 4.3. Part (3) follows immediately from part (2). □

Figure 11 shows the relationships between $\mathcal{NC}^k$ and $\mathcal{AC}^k$.

Theorem 4.5 PARITY $\in \mathcal{NC}^1$*:*

Proof: A depth 2, size 6 alternating circuit computing the parity of two inputs is shown in Figure 9. It can be used as a sub-circuit to compute the n-input parity function $x_1 \oplus \cdots \oplus x_n$ in depth $2\lceil \log n \rceil$ and size $6(n-1)$ using the binary tree construction illustrated in Figure 10. □

It is interesting to note that the unbounded fan-in of the $\mathcal{AC}$ circuits allows a significant reduction in depth without reducing size.

Theorem 4.6 *Any function computed by a classical circuit of depth d and size z can be computed by an alternating circuit of depth $\lceil d/\delta \rceil$ and size $O(z2^{2^\delta})$, for any $\delta \in \mathbb{N}$.*

Proof: Divide the classical circuit into horizontal strips of depth δ. Each gate on the bottom of a strip can depend on at most $2^{\delta-1}$ gates at the top of the strip, and thus its role can be played by an alternating circuit of size $O(2^{2^\delta})$ and constant depth (by Theorem 3.3). □

Corollary 4.7 *Any function that can be computed by a classical circuit of depth $D(n)$ and size $S(n)$ can be computed by an alternating circuit of depth $O(D(n)/\log\log S(n))$ and size less than $S(n)^{1+\epsilon}$ for any $\epsilon \in \mathbb{R}^+$.*

Proof: The claimed result follows immediately from Theorem 4.6 by taking

$$\delta = \lfloor 0.5 \log\log S(n) \rfloor.$$

The resulting circuit has size $S(n)2^{\sqrt{\log S(n)}}$, which for any $\epsilon \in \mathbb{R}^+$ and large enough n is less than $S(n)^{1+\epsilon}$. □

A weaker form of Corollary 4.7 is due to Chandra, Stockmeyer, and Vishkin [7]. Our result is the obvious generalization, and tightens the sloppy analysis of Theorem 5.2.8 of Parberry [29].

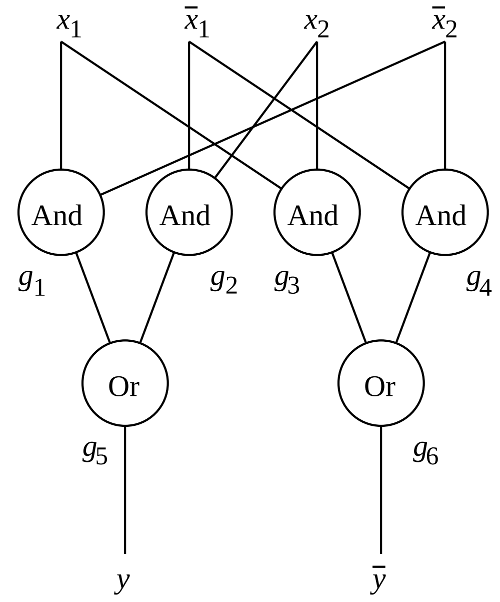


Figure 9: An alternating circuit computing $y = x_1 \oplus x_2$ and its complement.

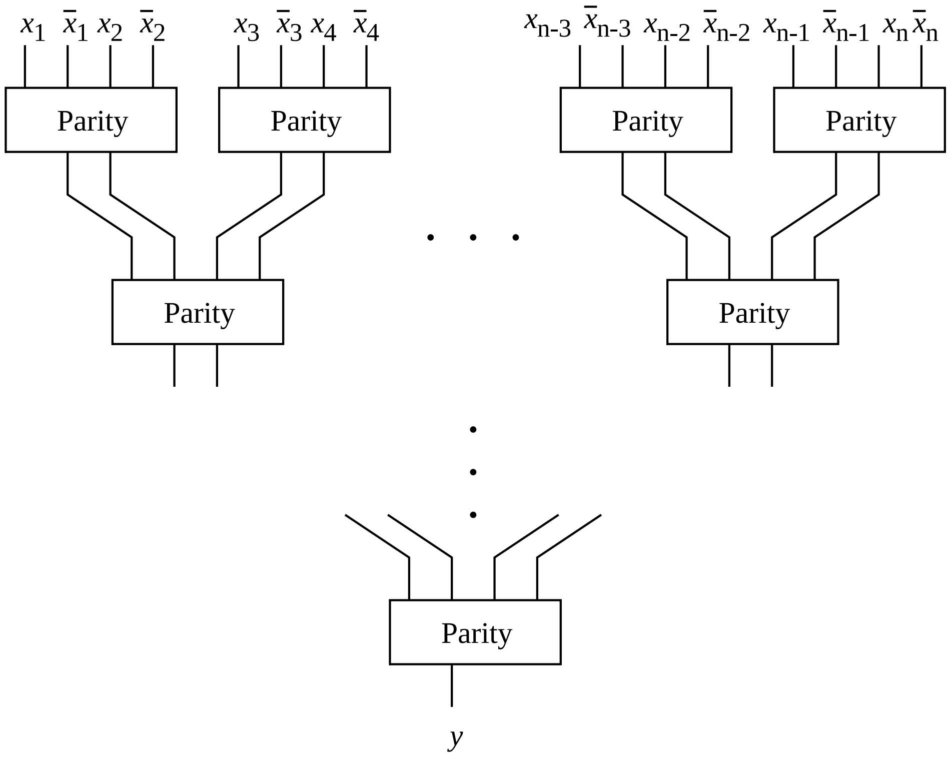

x_1 $\bar{x}_1$ x_2 $\bar{x}_2$
x_3 $\bar{x}_3$ x_4 $\bar{x}_4$
x_{n-3} $\bar{x}_{n-3}$ x_{n-2} $\bar{x}_{n-2}$ x_{n-1} $\bar{x}_{n-1}$ x_n $\bar{x}_n$
Parity
Parity
Parity
Parity
Parity
Parity
Parity
y

Figure 10: An alternating circuit computing $y = x_1 \oplus \cdots \oplus x_n$. Each box is a copy of the circuit in Figure 9.

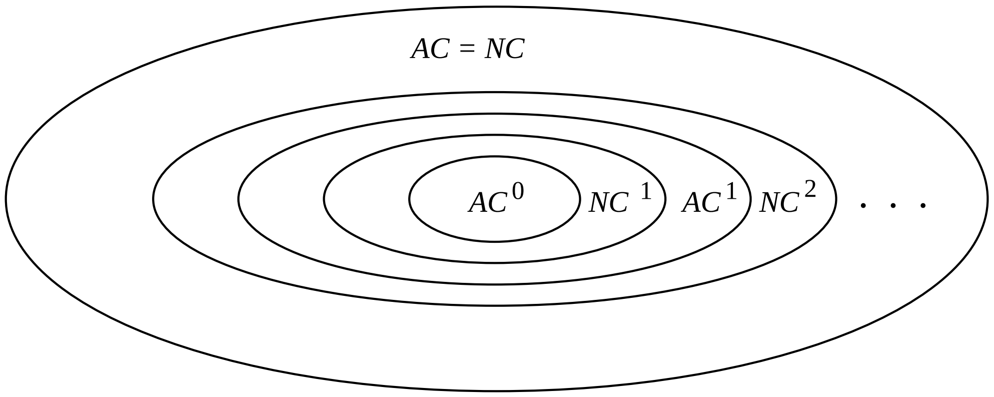


Figure 11: The classes $\mathcal{NC}^k$ and $\mathcal{AC}^k$.

Corollary 4.7 is particularly interesting when $S(n)$ is a polynomial in n, in which case it states that the depth of a classical circuit can be reduced by a factor of $\log \log n$ in return for a very small increase in size.

$\mathcal{AC}^0$, the set of problems that can be solved in polynomial size and *constant* depth, is of particular interest. The class $\mathcal{AC}^0$ was first studied by Furst, Saxe, and Sipser [11], and was named by Barrington [4]. We saw in Theorem 3.3 that every Boolean function can be computed in constant depth with *exponential* size, but this cannot be considered practical for any but the very smallest values of n. Unfortunately, as we saw in Theorem 3.5, some Boolean functions intrinsically require exponential size (regardless of depth). However, some interesting functions can be computed in constant depth with only polynomial size. For example, the sum of two n-bit natural numbers can be computed by an alternating circuit of size $O(n^2)$ and depth 4 using the standard carry-lookahead algorithm (see, for example, Wegener [40]). Chandra, Fortune and Lipton have shown by a sophisticated argument that the size bound can be reduced from $O(n^2)$ to an almost linear function. Define $f^{(1)}(x) = f(x)$, and for $i > 1$, $f^{(i)}(x) = f(f^{(i-1)}(x))$. Define $f_1(n) = 2^n$, and for $i > 1$, $f_i(n) = f_{i-1}^{(n)}(2)$. Chandra, Fortune and Lipton [6] have shown that there is an alternating circuit for computing the carry of two n-bit numbers in depth $6d+3$ and size $nf_d^{-1}(n)^2$. Surprisingly, they also found a matching lower bound [5]. However, there are languages in $\mathcal{NC}^1$ that are not in $\mathcal{AC}^0$:

Theorem 4.8 *Every constant depth AND-OR circuit for* PARITY *requires exponential size.*

PROOF: This result was originally proved by Furst, Saxe, and Sipser [11] and Ajtai [2], and improved lower bounds on the size required for constant depth alternating circuit for PARITY were successively obtained by Yao [41] and Hastad [38]. $\Box$

More strongly, for every $k \in \mathbb{N}$ there are functions that can be computed by alternating circuits in polynomial size and depth k, but require exponential size alternating circuits of depth $k - 1$ (Sipser [37]). This is often called the *depth hierarchy theorem* for $\mathcal{AC}^0$.

5 Threshold Circuits

The node function set for a weighted threshold circuit is the set of *weighted majority* functions $f : \mathbb{B}^n \to \mathbb{B}$. These are functions of the form:

$$f(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq h \\ 0 & \text{otherwise} \end{cases}$$

for some $w_1, \ldots, w_n, h \in \mathbb{R}$. The values $w_1, \ldots, w_n$ that define f are called *weights*, and the value h is called the *threshold*. The sequence $(w_1, \ldots, w_n, h)$ is called a *presentation* of f. The *weight* of a presentation is the maximum of the magnitude of its weights. Define

$$\sigma_n(w_1, \ldots, w_n)(x_1, \ldots, x_n) = \sum_{i=1}^{n} w_i x_i,$$

and further define $\theta_n(w_1, \ldots, w_n, h)$ to be the weighted majority function with presentation $(w_1, \ldots, w_n, h)$, that is,

$$\theta_n(w_1, \ldots, w_n, h)(x_1, \ldots, x_n) = 1 \text{ iff } \sigma_n(w_1, \ldots, w_n)(x_1, \ldots, x_n) \geq h.$$

It is clear that every weighted majority function has an infinite number of presentations. It is not too difficult to show that every weighted majority function has an infinite number of *integer* presentations, that is, presentations whose weights and thresholds are integers (Minsky and Papert [21]). Furthermore, every n-input weighted majority function has an integer presentation of weight bounded above by a function of n.

Theorem 5.1 *Every weighted majority function has an integer presentation of weight at most*

$$(n+1)^{(n+1)/2}/2^n.$$

PROOF: Suppose f is a weighted majority function. Assume without loss of generality that f is nondegenerate, that is, it depends on all of its inputs. Consider the following inequalities in unknowns $w_1, \ldots, w_n, h$, one for each $s = (s_1, \ldots, s_n) \in \mathbb{B}^n$. If $f(s) = 0$, the inequality corresponding to s is

$$\sigma_n(w_1, \ldots, w_n)(s_1, \ldots, s_n) \leq h - 1. \tag{1}$$

If $f(s) = 1$, the inequality corresponding to s is

$$\sigma_n(w_1, \ldots, w_n)(s_1, \ldots, s_n) \geq h. \tag{2}$$

The inequalities of the form (1) and (2) define a convex polytope in $\mathbb{R}^{n+1}$ whose interior and surface points are presentations of f. Since f has at least one presentation, this polytope is nontrivial.

Since f is nondegenerate, there is a point on the hypersurface of the polytope which meets exactly $n+1$ hyperfaces. This point satisfies $n+1$ of the inequalities (1), (2) in exact equality. Therefore there are $n+1$ equations in $w_1, \ldots, w_n, h$,

$$\begin{array}{ccccccccccc} s_{1,1}w_1 & + & s_{1,2}w_2 & + & \cdots & + & s_{1,n}w_n & - & h & = & a_1 \\ s_{2,1}w_1 & + & s_{2,2}w_2 & + & \cdots & + & s_{2,n}w_n & - & h & = & a_2 \\ & & & & & & & & & \vdots & \\ s_{n+1,1}w_1 & + & s_{n+1,2}w_2 & + & \cdots & + & s_{n+1,n}w_n & - & h & = & a_{n+1}, \end{array}$$

where $s_{i,j} \in \{0,1\}$ and $a_i \in \{0,-1\}$ for $1 \le i \le n+1$, whose solution is a presentation of f.

By Cramer's rule, the solution to these simultaneous equations is given by $w_i = \Delta_i/\Delta$ for $1 \le i \le n$, and $h = \Delta_{n+1}/\Delta$, where

$$\Delta = \begin{vmatrix} s_{1,1} & s_{1,2} & \cdots & s_{1,n} & -1 \\ s_{2,1} & s_{2,2} & \cdots & s_{2,n} & -1 \\ & & & \vdots & \\ s_{n+1,1} & s_{n+1,2} & \cdots & s_{n+1,n} & -1 \end{vmatrix},$$

and

$$\Delta_i = \begin{vmatrix} s_{1,1} & s_{1,2} & \cdots & s_{1,i-1} & a_1 & s_{1,i+1} & \cdots & s_{1,n} & -1 \\ s_{2,1} & s_{2,2} & \cdots & s_{2,i-1} & a_2 & s_{2,i+1} & \cdots & s_{2,n} & -1 \\ & & & & \vdots & & & & \\ s_{n+1,1} & s_{n+1,2} & \cdots & s_{n+1,i-1} & a_{n+1} & s_{n+1,i+1} & \cdots & s_{n+1,n} & -1 \end{vmatrix}$$

for $1 \le i \le n$, where $s_{j,k} \in \{0,1\}$, $a_j \in \{0,-1\}$, for $1 \le j \le n+1$, $1 \le k \le n$.

Clearly, by construction $(w_1, \ldots, w_n, h)$ is a presentation of f. Therefore (since multiplying all weights and thresholds by the same value gives another valid presentation), $(\Delta_1, \ldots, \Delta_n, g)$ is a presentation of f. Furthermore, since an integer determinant is always an integer, it is an integer presentation of f. It remains to show that $|\Delta_i| \le (n+1)^{(n+1)/2}/2^n$.

Negating column i of Δ_i, multiplying each of the first n columns by 2 and adding column $n+1$ to each of them, we find that

$$2^n|v_i| = \begin{vmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,i-1} & b_1 & t_{1,i+1} & \cdots & t_{1,n} & -1 \\ t_{2,1} & t_{2,2} & \cdots & t_{2,i-1} & b_2 & t_{2,i+1} & \cdots & t_{2,n} & -1 \\ & & & & \vdots & & & & \\ t_{n+1,1} & t_{n+1,2} & \cdots & t_{n+1,i-1} & b_{n+1} & t_{n+1,i+1} & \cdots & t_{n+1,n} & -1 \end{vmatrix}$$

for $1 \le i \le n$, where $t_{j,k} = 2s_{j,k} - 1 \in \{-1,1\}$, $b_j = -2a_j - 1 \in \{-1,1\}$, for $1 \le j \le n+1$, $1 \le k \le n$.

By the Hadamard inequality, the determinant of an $(n+1) \times (n+1)$ matrix over $\{-1,1\}$ is bounded above in magnitude by $(n+1)^{(n+1)/2}$. Thus we deduce that $|\Delta_i| \le (n+1)^{(n+1)/2}/2^n$. □

Theorem 5.1 is due to Muroga, Toda, and Takasu [23], and appears in more detail in Muroga [22]. Weaker versions of this result were more recently rediscovered by Hong [15], Raghavan [34], and Natarajan [25].

It is unknown whether the upper bound of Theorem 5.1 is tight. For obtaining lower bounds, it is useful to count the number of n-input weighted threshold functions.

Theorem 5.2 *There are at least $2^{n(n-1)/2}$ weighted threshold functions with n inputs.*

PROOF: Let $C(n)$ be the number of n-input weighted threshold functions with zero threshold. Then $C(1) = 2$, and we claim that for $n > 1$,

$$C(n) \ge (2^{n-1} + 1)C(n-1).$$

Let $f = \theta_n(w_1, \ldots, w_n, 0)$ be a weighted threshold function. We will count the number of ways that the weights $w_1, \ldots, w_n$ can be chosen to give different functions f. Partition the domain $\mathrm{I\!B^n}$ of f into two sets

$$\mathrm{I\!B}^{\mathrm{n}}_0 = \{(x_1, \ldots, x_{n-1}, 0) \mid x_i \in \mathrm{I\!B} \text{ for } 1 \le \mathrm{i} < \mathrm{n}\}$$

and

$$\mathbb{B}_1^{\mathrm{n}} = \{(x_1, \ldots, x_{n-1}, 1) \mid x_i \in \mathbb{B} \text{ for } 1 \leq \mathrm{i} < \mathrm{n}\}$$

The weights $w_1, \ldots, w_{n-1}$ can be chosen in $C(n-1)$ different ways, each of which makes f have a different output for some $x \in \mathbb{B}_0^{\mathrm{n}}$, since there are exactly as many choices for f restricted to domain $\mathbb{B}_0^{\mathrm{n}}$ as there are $(n-1)$-input weighted threshold functions with zero threshold. This choice of the first $n-1$ weights fixes the relative order of $\sigma_n(w_1, \ldots, w_n)(x)$ for all $x \in \mathbb{B}_1^{\mathrm{n}}$ (which values we can easily make distinct), regardless of the choice of w_n. Then w_n can be chosen to make $\sigma_n(w_1, \ldots, w_n)(x) < 0$ for exactly the first i of the $x \in \mathbb{B}_1^{\mathrm{n}}$ in this order, for $0 \leq i \leq 2^{n-1}$. Therefore $C(n) \geq (2^{n-1}+1)C(n-1)$, as claimed.

Therefore, by induction on n,

$$C(n) \geq \prod_{i=0}^{n-1} (2^i + 1) > 2^{n(n-1)/2}.$$

□

This result is attributed to Dahlin by Muroga [22]. The lower-bound can be improved to $C(n) > 2^{n(n-1)/2\ +16}$ by observing that $C(8) > 2^{44}$ (Muroga, Tsuboi, and Baugh [24]).

Define the *weight* of a weighted threshold function f to be the smallest $w \in \mathbb{N}$ such that f has an integer presentation with all weights no greater than w in absolute value. We can deduce from Theorem 5.2 that there are n-input weighted threshold functions with weight at least $2^{(n-1)/2}$, since if all weighted threshold functions have weights strictly less than this value, then there would be less than $2^{n(n-1)/2}$ weighted threshold functions. A similar lower bound can be found in Hampson and Volper [13]. This nonconstructive counting argument is a little unsatisfactory, since it does not give any specific weighted threshold functions with weights this large. Fortunately, a specific example is known:

Theorem 5.3 *If n is odd, $w_i = 2^{\lfloor (i-1)/2 \rfloor}$ for $1 \leq i \leq n$, and $h = 2^{(n-1)/2}$, the weighted threshold function $\theta_n(w_1, \ldots, w_n, h)$ has weight $2^{(n-1)/2}$.*

Proof: Suppose n is odd. Let $k = (n-1)/2$. It is easier to permute the inputs and consider instead the weighted threshold function

$$f = \theta_n(1, 2, 4, \ldots, 2^{k-1}, 2^k, 2^{k-1}, \ldots, 4, 2, 1, 2^k).$$

We are required to prove that there is no integer presentation of f with weights smaller in magnitude than 2^k. Suppose $(w_1, \ldots, w_n, h)$ is an arbitrary presentation of f. We will without loss of generality assume that all of the weights are non-negative. It is sufficient to prove that $w_{k+1} \geq 2^k$.

For $1 \leq i \leq k$, define $u_i, v_i \in \mathbb{B}^{\mathrm{n}}$ as follows:

$$u_i = (\overbrace{\underbrace{0, \ldots, 0}_{i}, 1, 0, \ldots, 0}^{k+1}, \overbrace{1, \ldots, 1, \underbrace{0, \ldots, 0}_{i-1}}^{k})$$

$$v_i = (\overbrace{\underbrace{1, \ldots, 1}_{i}, 0, 0, \ldots, 0}^{k+1}, \overbrace{1, \ldots, 1, \underbrace{0, \ldots, 0}_{i-1}}^{k}).$$

Then

$$\begin{aligned}\sigma_n(1,2,4,\ldots,2^{k-1},2^k,2^{k-1},\ldots,4,2,1,2^k)(u_i) &= 2^i + (2^k - 1 - \sum_{j=0}^{i-1} 2^j) \\ &= 2^k,\end{aligned}$$

and

$$\begin{aligned}\sigma_n(1,2,4,\ldots,2^{k-1},2^k,2^{k-1},\ldots,4,2,1,2^k)(v_i) &= \sum_{j=0}^{i-1} 2^j + (2^k - 1 - \sum_{j=0}^{i-1} 2^j) \\ &= 2^k - 1,\end{aligned}$$

so $f(u_i) = 1$ and $f(v_i) = 0$ for $1 \leq i \leq k$. Therefore $\sigma_n(w_1,\ldots,w_n)(u_i) \geq h$, that is,

$$w_{i+1} + \sum_{j=k+1}^{2k+2-i} w_j \geq h, \tag{3}$$

and $\sigma_n(w_1,\ldots,w_n)(v_i) < h$, that is,

$$\sum_{j=1}^{i} w_j + \sum_{j=k+1}^{2k+2-i} w_j < h. \tag{4}$$

Inequalities (3) and (4) imply that

$$w_{i+1} \geq \sum_{j=1}^{i} w_j,$$

for $1 \leq i \leq k$. Therefore, by induction on i, $w_{i+1} \geq 2^i$ for $1 \leq i \leq k$. In particular $w_{k+1} \geq 2^k$. □

The above result can easily be modified to give an improved lower bound of $\lfloor \phi^n/\sqrt{5} \rfloor$, where $\phi = (1+\sqrt{5})/2$. Håstad [39] has found a weighted threshold function that requires weights at least $n^{n/2-O(n)}$ (see Parberry [30] for a slightly better bound on this function).

Theorem 5.1 implies that the weights used in any polynomial size weighted threshold circuit can be described using only $O(n \log n)$ bits. Of particular interest are weighted threshold circuits that have weights ± 1. Since every gate can be padded out with extra inputs that are fixed to 0 or 1 (to bring the threshold to exactly half of the number of inputs), these are identical in theory to threshold circuits. (Recall that threshold circuits are built from majority gates, which are true when the majority of their inputs are true, and unary negation gates).

It can be shown that polynomial size weighted threshold circuits compute exactly the decision problems in $\mathcal{P}$, and polynomial size, polylog depth weighted threshold circuits recognize exactly the decision problems in $\mathcal{AC}$. Things become far more interesting, however, with constant depth. Let $\mathcal{TC}^0$ be the class of decision problems computed by threshold circuits of constant depth and polynomial size. It is clear that $\mathcal{TC}^0 \neq \mathcal{AC}^0$ since PARITY can be computed in depth 2 and linear size, as follows.

A function $f : \mathbb{B}^n \rightarrow \mathbb{B}$ is called *symmetric* if its output remains the same regardless of the order of the input bits.

Theorem 5.4 *Any symmetric function can be computed by a threshold circuit with size* $2n+1$ *and depth 2.*

PROOF: A symmetric function can be uniquely defined by the set

$$S_f = \{m \in \mathbb{N} \mid \mathrm{f(x) = 1 \text{ for all } x \in \mathbb{B}^n \text{ with exactly m ones}}\}.$$

The circuit uses $\|S_f\|$ pairs of gates. The ith pair has one gate active when the number of ones in the input is at least the ith member of S_f, and the other gate active when the number of ones in the input is at most the ith member of S_f. When given an input x such that $f(x) = 1$, exactly $\|S_f\| + 1$ of these gates are active, and when given an input x such that $f(x) = 0$, exactly $\|S_f\|$ of these gates are active. The output gate can therefore be a gate with threshold value $\|S_f\| + 1$ whose inputs come from these gates. $\square$

The exact relationship between $\mathcal{AC}^0$ and $\mathcal{TC}^0$ is not known. It has been conjectured that $\mathcal{AC}^0$ is contained in $\mathcal{TC}^0$ depth 3 (Immerman and Landau [16]); however, all that is known is that every function in $\mathcal{AC}^0$ can be computed by threshold circuits of depth 3 and size $n^{\log^c n}$ (Allender [3]). There is no depth hierarchy theorem for $\mathcal{TC}^0$, although there is a depth hierarchy theorem for *monotone* $\mathcal{TC}^0$ (that is, $\mathcal{TC}^0$ without Boolean negations, Yao [42]). In the general (nonmonotone) case, it is trivial to separate $\mathcal{TC}^0$ depth 1 from depth 2, and depth 2 has been separated from depth 3 (Hajnal *et al.* [12]), but beyond that nothing is known.

It is known that the sum of n polynomial-bit numbers can be computed by a threshold circuit in constant depth and polynomial size (Chandra, Stockmeyer and Vishkin[7]). It can be deduced from this result and Theorem 5.1 (Parberry and Schnitger [31]) that any problem that can be solved using a polynomial size, constant depth weighted threshold circuit can be solved using a polynomial size, constant depth threshold circuit. That is, $\mathcal{TC}^0$ is the same even if weighted threshold gates are used instead of majority gates.

6 Variations

The tools and techniques that we have surveyed in this paper are not limited to our simple feedforward model that we have used so far. Variations that appear in the literature include:

1. *Networks with cycles.* These can be unwound into feedforward neural networks in the obvious manner (see, for example, Parberry and Schnitger [31, 32]).

2. *Probabilistic neural networks.* Allowing computers access to a random source appears to make them more efficient than a plain deterministic computer in some circumstances (see, for example, Cormen, Leiserson, and Rivest [10, Section 33.8]). In this case, it is sufficient for the algorithm to compute the correct result with high probability, say 0.999. Surprisingly, such a randomized algorithm can be replaced with a nonuniform one with only a small increase in resources (Adleman [1]). This principle can even be applied to probabilistic neural networks such as Boltzmann machines (Parberry and Schnitger [32]). More specifically, uniform probabilistic $\mathcal{TC}^0$ is the same as nonuniform $\mathcal{TC}^0$.

3. *Analog node functions.* Many neural network researchers use a continuous model (i.e. one in which the neurons compute a continuous value). It can be shown that if one assumes that neuron outputs are robust to small errors in precision, then their model is essentially the same as a discrete one within $\mathcal{TC}^0$ (Obradovic and Parberry [26, 27]). More importantly, the same is true *even without the assumption of robustness* (Maass, Schnitger, and Sontag [20]). More specifically, any problem that can be solved by an analog neural network of polynomial size and constant depth can be approximated by a $\mathcal{TC}^0$ circuit.

7 Conclusion

Circuit complexity is a useful tool for analyzing the scalability of neural networks, but its usefulness should not be exaggerated. Whilst it tells us that, for example, analog and probabilistic neural networks are not *much* faster or hardware efficient than discrete neural networks, they may well *be* faster or more hardware efficient in practice (within the limits set down by the theory). The proper interpretation is not that we should abandon analog neural networks and concentrate on building discrete ones, but that it is only possible to build a polynomial size, constant depth analog neural network for problems that approximate decision problems in $\mathcal{TC}^0$. This alone gives us a powerful tool that enables us to decide exactly which functions can be evaluated efficiently by analog neural networks.

Computational complexity theory can sometimes (but not always) distinguish between problems that require exponential resource usage and those for which polynomial resources suffice, but often even quadratic resource usage is too large. It remains an open problem to develop a complexity theory of neural networks that is realistic, useful, and deep.

References

[1] L. Adleman. Two theorems on random polynomial time. In *19th Annual Symposium on Foundations of Computer Science*, pages 75–83. IEEE Computer Society Press, 1978.

[2] M. Ajtai. Σ_1^1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24:1–48, 1983.

[3] E. Allender. A note on the power of threshold circuits. In *30th Annual Symposium on Foundations of Computer Science*, pages 580–584. IEEE Computer Society Press, 1989.

[4] D. A. Barrington. Bounded width polynomial size branching programs recognize exactly those languages in NC^1. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 1–5. ACM Press, 1986.

[5] A. K. Chandra, S. Fortune, and R. Lipton. Lower bounds for constant depth circuits for prefix problems. In *Proc. 10th International Colloquium on Automata, Languages, and Programming,* in Series *Lecture Notes in Computer Science*, volume 154, pages 109–117. Springer-Verlag, 1983.

[6] A. K. Chandra, S. J. Fortune, and R. Lipton. Unbounded fan-in circuits and associative functions. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 52–60. ACM Press, 1983.

[7] A. K. Chandra, L. J. Stockmeyer, and U. Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13(2):423–439, May 1984.

[8] S. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1–3):2–22, 1985.

[9] S. A. Cook. Towards a complexity theory of synchronous parallel computation. *L'Enseignement Mathématique*, XXVII(1–2):75–100, 1980.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[11] M. Furst, J. B. Saxe, and M. Sipser. Parity, circuits and the polynomial time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

[12] A. Hajnal, W. Maass, P. Pudlák, M. Szegedy, and G. Turán. Threshold circuits of bounded depth. In *28th Annual Symposium on Foundations of Computer Science*, pages 99–110. IEEE Computer Society Press, October 1987.

[13] S. E. Hampson and D. J. Volper. Linear function neurons: Structure and training. *Biological Cybernetics*, 53:203–217, 1986.

[14] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117(5):285–306, 1965.

[15] J. Hong. On connectionist models. Technical Report 87-012, Dept. of Computer Science, Univ. of Chicago, June 1987.

[16] N. Immerman and S. Landau. The complexity of iterated multiplication. *Proc. 4th IEEE Structure in Complexity Theory Conference*, pages 104–111, 1989.

[17] R. E. Ladner. The circuit value problem is log space complete for P. *SIGACT News*, 7(1):18–20, 1975.

[18] O. Lupanov. Implementing the algebra of logic functions in terms of bounded depth formulas in the basis $+$, $*$, $-$. *Soviet Physics Doklady*, 6(2), 1961.

[19] O. Lupanov. Implementing the algebra of logic functions in terms of bounded depth formulas in the basis $+$, $*$, $-$. *Doklady Akad. Nauk SSR*, 166(5), 1961.

[20] W. Maass, G. Schnitger, and E. D. Sontag. On the computational power of sigmoid versus Boolean threshold circuits. In *32nd Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, 1991, To Appear.

[21] M. Minsky and S. Papert. *Perceptrons.* MIT Press, 1969.

[22] S. Muroga. *Threshold Logic and its Applications.* Wiley-Interscience, New York, 1971.

[23] S. Muroga, I. Toda, and S. Takasu. Theory of majority decision elements. *J. Franklin Inst.*, 271:376–418, May 1961.

[24] S. Muroga, T. Tsuboi, and C. R. Baugh. Enumeration of threshold functions of eight variables. *IEEE Transactions on Computers*, C-19(9):818–825, September 1970.

[25] B. K. Natarajan. *Machine Learning: A Theoretical Approach.* Morgan Kaufmann, 1991.

[26] Z. Obradović and I. Parberry. Analog neural networks of limited precision I: Computing with multilinear threshold functions. In *Advances in Neural Information Processing Systems 2*, pages 702–709. Morgan Kaufmann, 1990.

[27] Z. Obradović and I. Parberry. Learning with discrete multi-valued neurons. *Proceedings of the Seventh Annual Machine Learning Conference*, pages 392–399, 1990.

[28] I. Parberry. *Parallel Complexity Theory.* Research Notes in Theoretical Computer Science. Pitman Publishing, London, 1987.

[29] I. Parberry. A primer on the complexity theory of neural networks. In R. Banerji, editor, *Formal Techniques in Artificial Intelligence: A Sourcebook*, volume 6 of *Studies in Computer Science and Artificial Intelligence*, pages 217–268. North-Holland, 1990.

[30] I. Parberry. *Circuit Complexity and Neural Networks.* MIT Press, 1994.

[31] I. Parberry and G. Schnitger. Parallel computation with threshold functions. *Journal of Computer and System Sciences*, 36(3):278–302, 1988.

[32] I. Parberry and G. Schnitger. Relating Boltzmann machines to conventional models of computation. *Neural Networks*, 2(1):59–67, 1989.

[33] N. Pippenger. On simultaneous resource bounds. In *20th Annual Symposium on Foundations of Computer Science*, pages 307–311. IEEE Computer Society Press, 1979.

[34] P. Raghavan. Learning in threshold networks. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 19–27, Cambridge, MA, August 1988.

[35] N. P. Redkin. Synthesis of threshold element networks for certain classes of Boolean functions. *Kibernetika*, (5):6–9, 1970.

[36] D. E. Rumelhart, G. E. Hinton, and J. L. McClelland. A general framework for parallel distributed processing. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 282–317. MIT Press, 1986.

[37] M. Sipser. Borel sets and circuit complexity. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 61–69. ACM Press, 1983.

[38] J. Håstad. Improved lower bounds for small depth circuits. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 6–20. ACM Press, 1986.

[39] J. Håstad. On the size of weights for threshold gates. Unpublished Manuscript, 1992.

[40] I. Wegener. *The Complexity of Boolean Functions.* Wiley-Teubner, 1987.

[41] A. C. Yao. Separating the polynomial-time hierarchy by oracles. In *26th Annual Symposium on Foundations of Computer Science*, pages 1–10. IEEE Computer Society Press, 1985.

[42] A. C. Yao. Circuits and local computation. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 186–196. ACM Press, 1989.