Depth–Size Tradeoffs for Neural Computation

Kai-Yeung Siu, Vwani P. Roychowdhury, and Thomas Kailath, *Fellow, IEEE*

Abstract*— We study the tradeoffs between the *depth* (i.e., the time for parallel computation) and the *size* (i.e., the number of threshold gates) in neural networks. This is an important issue in neural computation but far from being resolved in general. In this paper, we focus our study on the neural computations of symmetric Boolean functions and some arithmetic functions. We show that for these functions, a small increase in the depth can significantly decrease the size required in the network. In particular, we show that any symmetric Boolean function (in n variables) can be computed with $O(\sqrt{n})$ threshold gates in a depth-3 network while the best known result required $O(n)$ threshold gates in a depth-2 network; moreover, our depth-3 network is almost optimal in the number of threshold gates and has the minimum size among all the known constructions even if unbounded depth networks are considered. This result answers an open question first posed in 1961 by Kautz [10]. Our novel results about implementing symmetric functions can be also used to improve results about arbitrary Boolean functions. In particular, we show that any Boolean function can be computed in a depth-3 neural network with $O(2^{n/2})$ threshold gates; we also prove that at least $\Omega(2^{n/3})$ threshold gates are required by using a result due to Shannon. Similar tradeoffs are also shown for functions such as *MULTIPLE SUM, ADDITION,* and *COMPARISON.

***Index Terms*—Arithmetic functions, depth–size tradeoffs, neural networks, symmetric Boolean functions, threshold circuits.**

I. Introduction

An artificial neural network, or simply neural network, is now commonly accepted as a general term for any computing architecture that consists of massively parallel interconnection of simple processing units. The initial inspiration for such networks was derived from the biological structure of our brains. The motivation comes mainly from the fact that the human brains seem so much more efficient than the existing digital computers in visual processing and speech recognition. It is therefore believed that there must be some underlying computational principles of the brains differing from those of digital computers.

Neurobiologists and psychologists have been for many years studying the brain from different points of view and trying to understand how our brain works. Much of their work is of a qualitative nature. The mathematical study of neural networks began more than 30 years ago when Rosenblatt proposed a model, called the "perceptron," that is an idealization of each neuron in our brain. He also invented a simple "learning" algorithm for the perceptron. However, as is well known, the perceptron had various limitations that were thoroughly investigated in the 1960's and many negative results were found. Nevertheless, it is also known that many of these limitations do not apply when we consider a network of artificial neural elements. Recently, interest in the study of neural networks has resurged because advances in VLSI technology have made possible the implementation of a massively interconnected network of artificial neural elements, and in many applications, neural networks seem to outperform the traditional methods. While neural networks have found wide application in many areas, the capability and the limitation of such networks are far from being understood. Undoubtedly, any significant progress in the application of neural networks requires a deeper understanding of their fundamental computational properties.

One common model of neural networks is the feedforward multilayer network in which the basic processing unit is a linear threshold gate. A linear threshold gate computes a Boolean function $f(X)$ of its input variables $X = (x_1, \cdots, x_n)$ such that

$$f(X) = \operatorname{sgn}(F(X)) = \begin{cases} 1 & \text{if } F(X) \geq 0 \\ 0 & \text{if } F(X) < 0 \end{cases}$$

where

$$F(X) = \sum_{i=1}^{n} w_i \cdot x_i + w_0.$$

In some literature on neural networks, each of the input variables and outputs of the linear threshold gates is encoded as $\{1, -1\}$ instead of $\{1, 0\}$. These two representations are equivalent and a choice is made based on convenience. For example, the $\{1, -1\}$ encoding can be converted to the $\{1, 0\}$ encoding by the *affine transformation* $x \rightarrow (x + 1)/2$. Thus, given a neural network encoded in the $\{1, -1\}$ representation, there is an equivalent neural network with the same *size* and *depth* (to be defined below). In our analysis, we find it natural and more convenient to adopt the $\{1, 0\}$ convention.

A feedforward network is a network of interconnected functional elements $\in G$: $\{1, 0\}^n \rightarrow \{1, 0\}$ with no feedback. More formally, we define a feedforward network to be an acyclic labeled directed graph, with

1) a list of n_{in} distinguished input nodes with indegree 0.

Manuscript received February 1, 1991; revised June 17, 1991. This work was supported in part by the Joint Services Program at Stanford University (U.S. Army, U.S. Navy, U.S. Air Force) under Contract DAAL03-88-C-0011, the SDIO/IST, managed by the Army Research Office under Contract DAAL03-90-G-0108, and the Department of the Navy, NASA Headquarters, Center for Aeronautics and Space Information Sciences under Grant NAGW-419-S6.

K.-Y. Siu is with the Department of Electrical and Computer Engineering, University of California, Irvine, CA 92717.

V. P. Roychowdhury is with the Department of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

T. Kailath is with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305.

IEEE Log Number 9103508.

2) internal nodes with arbitrary indegree, each of which represents a functional gate $\in G$.
3) a list of n_{out} distinguished output nodes.

The *depth* of a node is defined to be the length of the longest path (each edge is a unit length) from the input nodes to that node. The *depth* of the network is defined to be the maximum depth of all output nodes. If we group all gates with the same depth together, we can consider the network to be arranged in layers, where the depth of the network is equal to the number of layers (excluding the input layer) in the network, and gates of the same layer are computed in parallel. Given an assignment of the input nodes from domain $\{1,0\}^{n_{\text{in}}}$, the value of the network at each output node is obtained by evaluation of the gates in increasing depth order. The network therefore defines a mapping from $\{1,0\}^{n_{\text{in}}}$ to $\{1,0\}^{n_{\text{out}}}$, and the depth of the network can be interpreted as the time for parallel execution of the mapping. The *size* of the network is the number of gates and it measures the required amount of hardware.

We define a *neural network* to be a feedforward network of linear threshold gates. Similarly, a logic circuit is a feedforward network of AND, OR, NOT logic gates. Obviously, any Boolean function can be computed by a logic circuit (without any restriction on its size) and thus by a neural network, since AND, OR, NOT functional gates can be realized by linear threshold gates.

It is well known that threshold gates are much more powerful than unbounded fan-in AND, OR, NOT logic gates. While no logic circuit with polynomial (in n) number (i.e., n^{α}) of unbounded fan-in AND, OR, NOT gates can compute simple symmetric function such as the parity of n variables in *constant depth* [7], it only takes a depth-2 neural network of $O(n)$ threshold gates to compute *any* symmetric function [15], [13], [8]. These results were further extended in [5], [17], [1], [19] and by showing that functions such as multiplication, division, and sorting can be computed by "constant"-depth neural networks but require $\Omega(\log n/\log\log n)$ depth in any polynomial size logic circuit. However, the resulting "constants" in these networks are comparable to $O(\log n)$ for moderately large n. These results were improved in [24], [23], and [26] by giving depth-efficient neural networks for computing *multiple addition, multiplication,* and *sorting.* In particular, it was shown that *multiple addition* can be computed in depth 3, *multiplication* and *sorting* in depth-4 [24], [23], and division and exponentiation in depth-5 [26], with polynomial number of threshold gates. These results have the following implication on their practical significance: *Suppose we can use analog devices to build threshold gates with a cost (in terms of delay and chip area) that is comparable to that of AND, OR, NOT logic elements, then we can compute many basic functions much faster than using traditional logic circuits.*

While all previous results have mainly focused on the depth of the network and allowed the size of the network to grow as a polynomial (in the number of inputs), little is known about the tradeoffs between the depth and the size of the networks. One natural question to ask is the following: *Is it possible to significantly reduce the size of the network by slightly increasing the depth?* In other words, can we reduce the required amount of hardware substantially by slightly increasing the time for the parallel execution of the network? Although this is an important issue, it is almost hopeless to answer such questions in general. In this paper, we attempt to provide some partial answers by focusing on the class of symmetric functions and some arithmetic functions. We shall see that significant tradeoffs between the depth and the size of the networks that compute these functions are indeed possible.

A. Definitions and Summary of Main Results

Definition: A Boolean function $f\colon \{1,0\}^n \to \{1,0\}$ is said to be *symmetric* if

$$f(x_1,\cdots,x_n) = f(x_{(1)},\cdots,x_{(n)})$$

for any permutation $(x_{(1)},\cdots,x_{(n)})$ of $(x_1,\cdots,x_n)$, or equivalently, there exists a set of numbers $\{k_1,\cdots,k_l\}$, $0 \le k_i \le n$ such that

$$f(x_1,\cdots,x_n) = 1 \quad \text{iff} \sum_{i=1}^{n} x_i \in \{k_1,\cdots,k_l\},$$

i.e., a symmetric function depends only on the sum of the input values. A useful symmetric function, for example, is the Parity function. The combinational complexity of symmetric functions has been studied intensively by several researchers and significant results on the size of conventional AND/OR circuits implementing such functions have been reported [14], [28].

In the context of neural networks, it was first shown by Muroga [15] that any symmetric function can be computed in a depth-2 neural network with $(n+1)$ threshold gates and the size was later improved by Minnick [13] to $n/2+1$ (with the same depth). *This has been the best known result on the size even for unbounded-depth neural networks.* Recently, it was proved (see [16]) that any depth-2 neural network computing the PARITY function has size at least $\Omega(n/\log^2 n)$.[1] This shows that increasing the depth of the network beyond 2 is essential in order to reduce the size significantly from $O(n)$. In fact, whether the size of the network can be reduced below $O(n)$ was posed as an open problem as early as 1961 [10].

In Section II, we answer this open question by showing that any symmetric function of n variables can be computed in a depth-3 neural network with at most $2\sqrt{n}+O(1)$ threshold gates. In other words, increasing the depth of the network by 1 enables one to reduce the gate count by a factor of $O(\sqrt{n})$. Moreover, our construction appears to be the best known (with respect to minimizing gate count) for any depth.

Can the size of the networks be further reduced for depth-3 circuits? We can show [27] that the depth-3 networks derived in Section II are almost optimal in size. In particular, the lower bound results state that for any $\epsilon > 0$, the size of depth-3 networks computing arbitrary symmetric functions is $\Omega(n^{1/2-\epsilon})$. These lower bound results were obtained through the use of deeper mathematical tools from the theory of

[1] We use the following standard shorthand notations: when we write $f(n) = O(g(n))$ or $g(n) = \Omega(f(n))$, we shall mean that $f(n) \le c \cdot g(n)$ for some constant $c > 0$, as $n \to \infty$.

rational approximation and the harmonic analysis of Boolean functions [21], [20], [2], [25], [3], [12]. A detailed presentation of such results is beyond the scope of this paper and we shall refer the interested readers to [27].

Whether the size of the neural networks implementing arbitrary symmetric functions can be further reduced (by possibly increasing the depth) remains an important open question. We might mention here that for symmetric functions with structure, one can design neural networks with smaller gate count. For example, in [27] almost optimal depth-d networks that compute the *PARITY* and the *COMPLETE QUADRATIC* functions using $O(n^{1/(d-1)})$ gates are presented.

In Section III we show that the depth–size tradeoff results in Section II can be applied to obtain reduced size networks for arbitrary Boolean functions. In particular we show that any Boolean function of n variables can be computed by a depth-3 neural network with $O(2^{n/2})$ threshold gates. This improves on the straightforward depth-2 circuits with $O(2^n)$ gates. Thus, our techniques lead to an exponential reduction in the network size for arbitrary Boolean functions. Furthermore, using a result due to Shannon [22] we can show that the size of neural networks computing arbitrary Boolean functions is $\Omega(2^{n/3})$; this again appears to be the best known lower bound in this area. Improving the lower bound and obtaining implementations with smaller size than $O(2^{n/2})$ remain important open questions.

While Sections II and III deal with efficient implementations of general classes of functions, in Section IV we explore the depth–size tradeoffs for specific arithmetic functions. For example, in Section IV-A we define MULTIPLE SUM as follows.

Definition: The multioutput function MULTIPLE SUM is defined to be the $(\log n + n)$-bit representation of the sum of n n-bit numbers.

Computing MULTIPLE SUM is a basic step in multiplying two n bit numbers. We show that for any $\epsilon > 0$ (independent of n), MULTIPLE SUM can be computed in a constant depth neural network of size $O(n^{1+\epsilon})$. This shows that the multiplication of two n bit numbers can be performed in constant depth using almost linear number of threshold gates and n^2 two-input AND gates. Making a fair assumption that unbounded fan-in threshold gates are more expensive to implement than two-input AND gates, we claim that our results improve the previous best result due to [9] where it was shown that multiplication can be done with $O(n^2)$ unbounded fan-in threshold gates and n^2 two-input AND gates.

In Section IV-A we discuss depth–size tradeoffs for COMPARISON and ADDITION. To underscore the power of threshold circuits, we show that any depth-2 circuits using AND, OR, NOT gates will require exponentially many gates to implement the COMPARISON (ADDITION) function whereas it has been shown that the same function can be computed by a depth-2 threshold circuit with $O(n^4)\,(O(n^5))$ gates. We also show that the size of the networks required for implementing these functions can be substantially reduced by using depth-3 circuits.

Finally, Section V has some concluding remarks and possible future directions of research.

II. Implementing Symmetric Boolean Functions

Let f: $\{1,0\}^n \rightarrow \{1,0\}$ be an arbitrary symmetric Boolean function of n variables. Since f depends only on the sum of its input values $\sum_{i=1}^{n} x_i$ (where, the input variables $x_i \in \{0,1\}$), we can define f by giving a set of integers s_i and $S_i, i = 1, \cdots, r$, with $s_i \leq S_i < s_{i+1}$ such that

$$f(x_1, \cdots, x_n) = 1 \quad \text{iff } s_i \leq \sum_{j=1}^{n} x_j \leq S_i \text{ for some } i.$$

For example, the PARITY function is defined as follows: $P(X)$: $\{1,0\}^n \rightarrow \{1,0\}$ as follows:

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

then for $n = 5$, we have $r = 3$ and $s_1 = S_1 = 1$, $s_2 = S_2 = 3$, $s_3 = S_3 = 5$.

We first introduce an ingenious technique called the *telescopic technique* that was introduced by Minnick [13]. We shall make repeated use of a similar technique in order to obtain $O(\sqrt{n})$ implementations of symmetric functions.

Lemma 1: Let the integer interval $[0, n]$ be divided into $k+1$ subintervals $[b_0, b_1 - 1], [b_1, b_2 - 1], \cdots, [b_{k-1}, b_k - 1]$, $[b_k, n]$, where $b_0 = 0 < b_1 < b_2 < \cdots < b_k < n$. Let $y_i = \operatorname{sgn}\left(\sum_{j=1}^{n} x_j - b_i\right)$, for all $1 \leq i \leq k$. Then,

$$\sum_{j=1}^{k} (a_j - a_{j-1}) y_j = a_m \quad \text{if } \sum_{j=1}^{n} x_j \in [b_m, b_{m+1} - 1]$$

where $a_0 = 0$, and $a_1, \cdots, a_k$ are arbitrary real numbers.

Proof: Recall that $y_i = \operatorname{sgn}(\sum_{j=1}^{n} x_j - b_i)$ equals 1 if and only if $\sum_{j=1}^{n} x_j \geq b_i$ otherwise it equals 0. Hence, if $\sum_{j=1}^{n} x_j \in [b_m, b_{m+1} - 1]$ (assume for now $m \geq 1$) then $y_1 = \cdots = y_m = 1$, and $y_{m+1} = \cdots = y_k = 0$. Hence,

$$\sum_{j=1}^{k} (a_j - a_{j-1}) y_j = \sum_{j=1}^{m} (a_j - a_{j-1}) = a_m.$$

Note that if $\sum_{j=1}^{n} x_j \in [b_0, b_1 - 1]$, i.e., $m = 0$, then $y_i = 0$, $\forall\, 1 \leq i \leq k$, and hence, $\sum_{j=1}^{n} (a_j - a_{j-1}) y_j = 0 = a_0$. □

Theorem 1: Any symmetric function of n variables can be computed in a depth-3 neural network with $2\sqrt{n} + O(1)$ threshold gates.

Proof: Recall that we can define f by giving a set of integers s_i and S_i, $i = 1, \cdots, r$, with $s_i \leq S_i < s_{i+1}$ such that

$$f(x_1, \cdots, x_n) = 1 \quad \text{iff } s_i \leq \sum_{j=1}^{n} x_j \leq S_i \text{ for some } i.$$

We can always divide the interval $[0, n]$ into d consecutive subintervals, $[s_1, s_{2_1} - 1], [s_{2_1}, s_{3_1} - 1], \cdots, [s_{d_1}, n]$, so that each subinterval (except possibly the last one) contains the same number l of the integers s_i and S_i, where $l \leq n/2d$. The ith subinterval will contain integers $s_{i_1} \leq S_{i_1} < s_{i_2} \leq S_{i_2} < \cdots < s_{i_l} \leq S_{i_l}$. In the following arguments, we shall assume that the last subinterval also contains l s_i's and l S_i's

to avoid the use of cumbersome notations. The readers can easily modify the arguments for the general case.

The first layer of our circuit consists of d threshold elements each computing the value z_i where

$$z_i = \operatorname{sgn}\left(\sum_{j=1}^{n} x_j - s_{i_1}\right) \quad \text{for } i = 1, \cdots, d.$$

For each $k = 1, \cdots, l$ we define two telescopic sums as follows:

$$\begin{aligned} T_k &= S_{1_k} z_1 + (S_{2_k} - S_{1_k}) z_2 \\ &\quad + (S_{3_k} - S_{2_k}) z_3 + \cdots + (S_{d_k} - S_{d-1_k}) z_d \\ t_k &= s_{1_k} z_1 + (s_{2_k} - s_{1_k}) z_2 \\ &\quad + (s_{3_k} - s_{2_k}) z_3 + \cdots + (s_{d_k} - s_{d-1_k}) z_d. \end{aligned}$$

Clearly, T_k and t_k are linear combinations of the outputs from the first layer.

The second layer of our circuit will consist of $2l$ threshold elements, each utilizing the integer T_k or t_k as a "threshold" value and computing the value Q_k or q_k where

$$Q_k = \operatorname{sgn}\left(T_k - \sum_{j=1}^{n} x_j\right) \quad \text{and} \quad q_k = \operatorname{sgn}\left(\sum_{j=1}^{n} x_j - t_k\right).$$

The output threshold element in the third layer computes $\operatorname{sgn}\left(\sum_{k=1}^{l} 2(Q_k + q_k) - 2l - 1\right)$.

Now we show that our circuit gives the correct outputs on all inputs $X = (x_1, \cdots, x_n)$. Suppose $\sum_{j=1}^{n} x_j$ lies within the mth interval, i.e., $\sum_{j=1}^{n} x_j \in [s_{m_1}, s_{(m-1)_1} - 1]$. By Lemma 1, the telescopic sums T_k, and t_k $(k = 1, \cdots, l)$ assume the following values:

$$T_k = S_{m_k}, \qquad t_k = s_{m_k}.$$

From our definition of the function $f(X)$,

$$f(X) = 1 \quad \text{iff for some } k,\ s_{m_k} \le \sum_{j=1}^{n} x_j \le S_{m_k}.$$

Let us assume that $f(X) = 1$ for the given input X; this implies that there exists an i such that $s_{m_i} \le \sum_{j=1}^{n} x_j \le S_{m_i}$. In the second layer, $\sum_{j=1}^{n} x_j$ is compared with $T_k = S_{m_k}$ and $-t_k = -s_{m_k}$ for each k, and the outputs are Q_k and q_k, respectively. Since $s_{m_i} \le \sum_{j=1}^{n} x_j \le S_{m_i}$, we can write down the values of the outputs of the second layer (Q_k, q_k) as follows:

$$Q_k + q_k = \begin{cases} 2 & \text{if } k = i \\ 1 & \text{if } k \neq i \end{cases}.$$

Thus, the output threshold element gives $\operatorname{sgn}(\sum_{k=1}^{l} 2(Q_k + q_k) - 2l - 1) = \operatorname{sgn}(2l + 2 - 2l - 1) = 1$.

Similarly, if $f(X) = 0$, i.e., for no $k, s_{m_k} \le \sum_{j=1}^{n} x_j \le S_{m_k}$, then $Q_k + q_k = 1$ for all k and the output threshold element gives $\operatorname{sgn}\left(\sum_{k=1}^{l} 2(Q_k + q_k) - 2l - 1\right) = \operatorname{sgn}(2l - 2l - 1) = 0$.

Hence, our depth-3 neural network gives the correct outputs on all inputs $X = (x_1, \cdots, x_n)$.

To determine the size of our network, note that the first layer consists of d elements, the second layer $2l \le n/d$ elements, and the third layer one output element. So altogether, at most $d + n/d + 1$ threshold elements are required. To complete the proof, take $d = \sqrt{n}$ to minimize the size and thus obtain a depth-3 network of size $2\sqrt{n} + O(1)$. □

Example: We illustrate the above procedure by implementing the PARITY function of 11 variables. Recall that the output of a PARITY function is 1 if $\sum_{i=1}^{11} x_i$ is odd. Hence, $s_1 = S_1 = 1$, $s_2 = S_2 = 3$, $s_3 = S_3 = 5$, $s_4 = S_4 = 7$, $s_5 = S_5 = 9$, and $s_6 = S_6 = 11$. In the constructive proof of Theorem 1, set $s_{1_1} = 1$, $s_{2_1} = 5$, and $s_{3_1} = 9$, i.e., choose $d = 3$. Thus, each subinterval has 2 of the s_i in it. In other words, $s_{1_1} = S_{1_1} = 1 < s_{1_2} = S_{1_2} = 3$; $s_{2_1} = S_{2_1} = 5 < s_{2_2} = S_{2_2} = 7$; and $s_{3_1} = S_{3_1} = 9 < s_{3_2} = S_{3_2} = 11$. Based on these choices, the realization of the function is shown in Fig. 1.

Note that our realization requires eight threshold gates; in comparison, an efficient depth-2 realization, using for example a procedure due to Minnick [13], would require seven gates. However, as the number of variables increase, the efficacy of our procedure becomes apparent. For example, implementing PARITY of 36 variables by Minnick's procedure would require 19 gates, whereas our procedure would require only 13. □

Remark 1: From our proof of Theorem 1, it is clear that the size of the neural network computing a symmetric function f actually depends on r_f, the number of different integers s_i's (or equivalently S_i's) required for defining the function. Recall from our definition of the integers s_i's: $f(X) = 1$ if and only if $s_i \le \sum_{j=1}^{n} x_j \le S_i$. On closer observation, the proof of Theorem 1 shows that our construction of the neural network computing f has size $2\sqrt{rf} + O(1)$. For example, for the PARITY function of n variables $r_f = \lceil n/2 \rceil$, and the gate count is $O(\sqrt{n})$.

A relevant issue is to determine the size of the network for most symmetric functions. The following theorem shows that the depth-3 network construction in Theorem 1 would require $O(\sqrt{n})$ threshold gates to compute most symmetric functions.

Theorem 2: At least $1 - e^{-\Omega(n)}$ portion of the class of symmetric Boolean functions would require $\Omega(\sqrt{n})$ threshold gates to compute in a depth-3 neural network as constructed in Theorem 1.

Proof: Recall from Remark 1 that our construction of the neural network computing a symmetric f has size $2\sqrt{r_f} + O(1)$, where r_f is the number of different integers s_i's corresponding to f: $f(X) = 1$ when $\sum_{j=1}^{n} x_j = s_i$ and $f(X) = 0$ when $\sum_{j=1}^{n} x_j = s_i - 1$. We shall use probabilistic methods to show that if we choose a function f of n-variables uniformly at random from the class of symmetric functions, then with probability $1 - e^{-\Omega(n)}$, we have $r_f = \Omega(n)$. In other words, with high probability, a random symmetric function f would require $\Omega(\sqrt{r_f}) = \Omega(\sqrt{n})$ threshold gates to compute in a depth-3 neural network as constructed in Theorem 1.

Since a symmetric function only depends on the sum of the input variables $\sum_{j=1}^{n} x_j$, and $0 \le \sum_{j=1}^{n} x_j \le n$, there is a one-to-one correspondence between each symmetric function of n variables and each $n+1$-dimensional binary vector. For example, $(0, 1, 0, 1, 1, 0)$ corresponds to a symmetric function

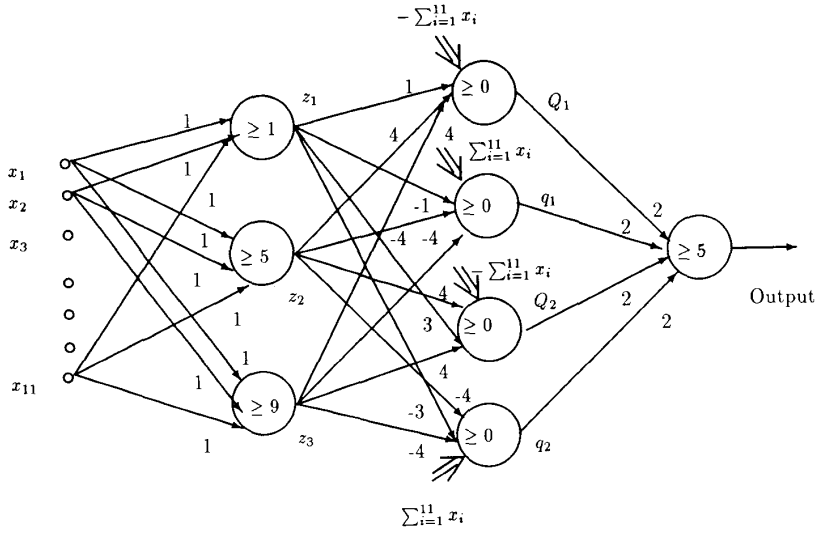


Fig. 1. A depth-3 network for computing the PARITY of 11 variables.

$f(X)$ of 5 variables such that $f(X) = 1$ iff $\sum_{j=1}^{5} x_j = 1$, 3, or 4 with $r_f = 2$, $s_1 = 1$, and $s_2 = 3$. Without loss of generality, assume n is odd. Consider each $n+1$ dimensional binary vector as $(n+1)/2$ consecutive pairs of 0's and 1's. In the above example, we consider $(0,1,0,1,1,0)$ as $((0,1),(0,1),(1,0))$. Then the corresponding number r_f is bounded below by p_f, the number of pairs which are equal to $(0,1)$. If we choose each coordinate to be 0 or 1 independently with probability $1/2$, then each pair $= (0,1)$ with probability $1/4$, or we can write p_j as a sum of independent identically distributed Bernoulli random variables as follows:

$$p_f = \sum_{j=1}^{(n+1)/2} u_j, \quad \text{where}$$

$$u_j = \begin{cases} 1 & \text{with probability } 1/4 \\ 0 & \text{with probability } 3/4 \end{cases}.$$

By the Chernoff Bound, for $\epsilon > 0$, we have

$$\Pr\left\{p_f < (1-\epsilon)\frac{(n+1)}{8}\right\} < e^{-\epsilon^2(n+1)/16}. \qquad \square$$

Remark 2: Theorem 2 shows that for most symmetric functions the size of the network by our construction is $O(\sqrt{n})$. Can one do better for random symmetric functions with depth-3 networks? It turns out that one *cannot* do substantially better than our procedure. In particular, it can be shown [27] that for any $\epsilon > 0$, most symmetric functions require $\Omega(n^{1/2-\epsilon})$ threshold gates to be computed by a depth-3 neural network.

III. Realizing Arbitrary Boolean Functions

Note that Theorem 1 only applies to symmetric Boolean functions; can similar depth–size tradeoffs be achieved for *arbitrary* Boolean functions? We would like to extend our results to a more general class of Boolean functions. Recall that a symmetric function only depends on the sum of its input variables $\sum_{j=1}^{n} x_j$. A natural generalization is to consider the class of functions which only depend on a *weighted* sum of its variables $\sum_{j=1}^{n} w_j x_j$, where the w_j's are positive integers. Obviously, we could consider any function in this class to be a symmetric function of $\sum_{j=1}^{n} w_j$ variables by repeating w_j times of each variable x_j. But then a direct application of Theorem 1 would yield a network of size $O\left(\sqrt{\sum_{j=1}^{n} w_j}\right)$, which could be much larger than $O(\sqrt{n})$ in general. It is natural to ask if this is the best we can do. By taking $w_j = 2^{j-1}$ (exponentially large!), $\sum_{j=1}^{n} 2^j x_j$ would be distinct for different values of $X = (x_1, \cdots, x_n)$, thus *any* Boolean function could be considered as a function that only depends on a (exponentially large) weighted sum of its input variables [10]. This observation gives us the motivation to extend our result to arbitrary Boolean function.

By considering the disjunctive or conjunctive normal form of a Boolean function, it is not hard to see that any Boolean function can be implemented by a depth-2 circuit with $O(2^n)$ unbounded fan-in logic gates. In 1949, Shannon [22] showed that almost all Boolean functions of n variables require circuits of size $\Omega(2^n/n)$ with bounded (two) fan-in gates. It is interesting to determine how many threshold gates (allowing unbounded fan-in) are *sufficient* to implement an arbitrary Boolean function. As previous arguments show, every Boolean function of n variables can be considered as a *symmetric* function of 2^n variables (by repeating each variable x_j 2^{j-1} times); we have as a consequence of Theorem 1 the following (similar result was also shown in [18]):

Corollary 1: Any Boolean function of n variables can be computed by a depth-3 neural network with $O(2^{n/2})$ threshold gates.

Thus, Corollary 1 shows that by increasing the depth by 1, one can obtain an exponentially smaller network than a direct depth-2 realization of an arbitrary Boolean function.

The lower bound result of Shannon on the size of network with *bounded fan-in* gates can be used to derive a lower bound on the size of neural network in implementing arbitrary Boolean functions. Before we prove the lower bound result, we need the following lemma, which states that we can simulate a linear threshold gate by a network of two-input AND, OR gates and NOT gates with size $O(n^2)$.

Lemma 2: A linear threshold function of n variables can be realized by a logic circuit with $O(n^2)$ number of two-input AND, OR gates and NOT gates.

Proof: This follows from a well-known result (see [29]) that the sum of n n-bit numbers can be computed by a network of two-input AND, OR gates and NOT gates with size $O(n^2)$. □

Theorem 3: Almost all Boolean functions of n variables require neural networks of size at least $\Omega(2^{n/3})$ to compute.

Proof: Let N be the minimum size of a neural network that can compute an arbitrary Boolean function of n variables. Each threshold gate in the network computes a linear threshold function of at most $(N+n)$ variables and thus can be simulated by a logic circuit of $O((N+n)^2)$ two-input gates by Lemma 2. Now replace every threshold gate in the neural network by the corresponding simulating logic circuit. This yields an overall logic circuit of size $O((N+n)^3)$. By Shannon's lower bound result, $(N+n)^3 = \Omega(2^n/n)$ for almost all Boolean functions. Thus, $N = \Omega(2^{n/3})$. □

There is still an exponential gap in the size between the upper bound $O(2^{n/2})$ in Corollary 1 and the lower bound $\Omega(2^{n/3})$ in Theorem 3. Whereas the network stated in Corollary 1 has only three layers, there is no restriction on the depth of the network stated in Theorem 3. So it might be possible to close such an exponential gap on the size by restricting the depth of the network to be constant.

IV. Implementing Arithmetic Functions

The results presented above apply to Boolean functions belonging to general classes (e.g., Corollary 1 for any Boolean function, Theorem 1 for any symmetric function); it remains an open problem to improve these upper bounds. Another interesting issue is to exploit the structures of specific functions and thereby obtain better depth–size tradeoff results. In the rest of this paper, we address the latter issue and present efficient implementations of several functions of practical interest such as MULTIPLE SUM, COMPARISON, and ADDITION.

A. Multiple Sum and Multiplication

Multiplication of two n-bit numbers can be easily reduced to the computation of the sum of n $O(n)$-bit numbers (multiple sum). It was known [7] that the product of two n-bit numbers cannot be computed in a *constant* depth logic circuit that consists of polynomially many (n^α) unbounded AND, OR, NOT gates. The reason for the "hardness" of multiplication lies in the fact that multiple sum cannot be computed in polynomial size AND, OR logic circuits of constant depth. On the other hand, multiple sum (and thus multiplication) is easily computable in a constant depth neural network that consists of the more powerful threshold gates. In fact, the best known result on the depth of polynomial size neural network for multiple sum and multiplication was first obtained by Siu and Bruck [25]. It was shown in [25] that multiplication can be computed in a neural network of depth 4 and polynomial size, whereas it was shown by Hajnal *et al.* [8] that at least depth-3 is needed. The first layer consists of $O(n^2)$ AND gates (each with fan-in = 2) to reduce multiplication to a multiple sum, and the multiple sum is then computed in the next three layers. The size of the neural networks (with the same depth) for multiple sum was later improved by Hofmeister *et al.* [9] to $O(n^2)$. It seems natural to assume that the cost of implementing (unbounded fan-in) threshold gates should dominate that of the (fan-in = 2) AND gates in the first layer. Therefore, it is interesting to determine if the size of the network for the multiple sum can be reduced further while keeping the depth to be constant. By generalizing our construction of depth-3 neural networks for symmetric functions, we can in fact reduce the size of the network for the multiple sum from $O(n^2)$ to $O(n^{1+\epsilon})$ for any fixed $\epsilon > 0$ and show depth–size tradeoffs for the multiple sum. Without loss of generality, we can assume each of the n numbers has n-bit. This in turn leads to efficient neural network implementations of multiplication of two n-bit numbers.

Definition: The multioutput function MULTIPLE SUM is defined to be the $(\log n + n)$-bit representation of the sum of n n-bit numbers.

Theorem 4: For any $\epsilon > 0$ (independent of n), MULTIPLE SUM can be computed in a constant depth neural network of size $O(n^{1+\epsilon})$.

The depth-3 neural network for MULTIPLE SUM in [25] consists of two major steps: the sum of n n-bit numbers is first reduced to the sum of two $(n + \log n)$-bit numbers using one layer, then the final sum is computed in two layers. We shall use this technique as our basis and utilize our result in Theorem 1 to reduce the size of each step while keeping the depth to be constant. First, let us prove two lemmas.

Lemma 3: Let $S_1(X) = \sum_i w_i x_i \in [0, 2^l]$ and $0 \le k < l$. For any positive integer d, the $(k+1)$th (least significant) bit of $S_1(X)$ can be computed in a depth-$(d+1)$ neural network of size $O(2^{(l-k)/d})$.

Proof: Note that the $(k+1)$th bit of $S_1(X)$ is 1 if $S_1(X) \in [j2^k, (j+1)2^k - 1]$ for $j = 1, 3, 5, \cdots, 2^{l-k} - 1$ and 0 otherwise.

The underlying idea is as follows: Each layer of our network has $2^{(l-k)/d}$ threshold gates. We use each layer to recursively remove the most significant $(l-k)/d$ bits of the sum computed from the previous layer. Since the original sum is $(l+1)$-bit long, the most significant bit of the reduced sum after d steps is the $(k+1)$th bit of the original sum.

To show this more formally, the first layer of our network consists of $2^{(l-k)/d}$ threshold gates each computing the value y_j where

$$y_j = \operatorname{sgn}\left(\sum_i w_i x_i - j2^{l-(l-k)/d}\right)$$

$$\text{for } j = 0, 1, \cdots, 2^{(1-k)/d} - 1.$$

Let

$$t = \sum_{j=0}^{2^{(1-k)/d}-1} 2^{l-(l-k)/d} y_j.$$

If $j^* 2^{l-(l-k)/d} \le \sum_i w_i x_i < (j^*+1)2^{l-(l-k)/d}$, then

$$y_j = \begin{cases} 1 & \text{if } j \le j^* \\ 0 & \text{if } j > j^* \end{cases} \quad \text{and} \quad t = j^* 2^{l(1-1/d)+k/d}$$

Thus, $0 \le (S_1(X) - t) < 2^{l-(l-k)/d}$. Let $S_2(X) = S_1(X) - t$. Note that $S_1(X) \in [j2^k, (j+1)2^k - 1]$ iff $S_2(X) \in [j2^k, (j+1)2^k - 1]$ for $j = 1, 3, 5, \cdots, 2^{l-k} - 1$. In other words, the $(k+1)$th bit of $S_1(X)$ is that of $S_2(X)$. Now apply the above reduction recursively. In general, for $2 \le m \le d$, the mth layer of our neural network consists of $2^{(l-k)/d}$ threshold gates each computing the value y_j^m where

$$y_j^m = \operatorname{sgn}(S_{m-1}(X) - j2^{l-m(l-k)/d})$$
$$\text{for } j = 0, 1, \cdots, 2^{(1-k)/d} - 1 \quad \text{and}$$
$$S_m(X) = S_{m-1}(X) - \sum_{j=0}^{2^{(1-k)/d}-1} 2^{l-m(l-k)/d} y_j.$$

The reduction is applied d times, each time increasing the depth by 1 and the size of the network by an additional $2^{(l-k)/d}$ threshold gates. At the $(d+1)$th layer, the output gate computes $\operatorname{sgn}(S_d(X) - 2^k)$ which is the $(k+1)$th significant bit of $S_1(X)$. □

Corollary 2: Given n log n-bit numbers and $\epsilon > 0$, each bit of their 2 log n-bit sum can be computed in constant depth neural network of size $O(n^\epsilon)$.

Proof: Let $\epsilon > 0$ be arbitrary and let the sum be represented by $s_{2\log n-1} s_{2\log n-2} \cdots s_0$ with s_0 being the least significant bit. We shall use the above lemma to show each s_k can be computed in a constant depth neural network of size $O(n^\epsilon)$. Choose d such that $\epsilon > 1/d$. We consider two cases. For $k = 0, \cdots, \log n - 2$, s_k depends only on a sum $\in [0, \cdots, 2^{\log n + k}]$. In this case, put $l = (\log n + k)$ in Lemma 2, it follows that each s_k can be computed in constant depth with $O(2^{(l-k)/d}) = O(n^\epsilon)$ number of threshold gates. Now consider $k = \log n - 1, \cdots, 2\log n$. Since the total sum $\in [0, 2^{2\log n}]$, put $l = 2\log n$ in Lemma 2, and in this case $(l-k) \le \log n$ and thus $O(2^{(l-k)/d}) = O(n^\epsilon)$ threshold gates suffice. □

The depth-3 neural network stated in Theorem 7 has size $O(n^2)$. In order to show that MULTIPLE SUM can be computed in constant depth neural network of size $O(n^{1+\epsilon})$ for arbitrary constant $\epsilon > 0$, we shall first show how to reduce the size of the neural network for ADDITION (of two n-bit numbers). This technique is based on ideas from parallel prefix computation [11], [6] and our proof is adapted from [4].

Lemma 4: ADDITION can be computed in a constant depth network of (unbounded fan-in) AND, OR gates and size $O(n \log n)$.

Proof: Before we prove this lemma, we need to introduce some notions and terminologies. Let $X = (x_{n-1}, \cdots, x_0)$, $Y = (y_{n-1}, \cdots, y_0) \in \{1, 0\}^n$ be the two n-bit numbers with x_0 and y_0 being the least significant bits. Given the proof of Theorem 7, it suffices to show that each carry bit c_k can be computed in a constant depth network of (unbounded fan-in) AND, OR gates and size $O(n \log n)$. Define an *interval* of indexes $1, \cdots, n, I$, to be an ordered subset of consecutive indexes and represent I by concatenating its indexes in an increasing order. Without loss of generality, assume $n = 2^m$. A *principal interval* I_j^k is an interval such that for some $k = 0, \cdots, m$ and $j = 1, \cdots, 2^{m-k}$, $I_j^k = (j-1)2^k \cdots (j2^k - 1)$. Another way of visualizing the principal intervals is as follows: construct a complete binary tree whose leaves are from left to right the indexes $1, \cdots, n$; then each principal interval corresponds to an interior node v by concatenating the indexes that are the leaves of the subtree rooted at v. Since there are $O(n)$ interior nodes in a complete binary tree, there are $O(n)$ principal intervals. The principal interval I_j^k corresponds to the jth node (from left to right) on the kth level (the leaves are on level 0). The following observation is crucial to our proof: *every prefix of* $1, \cdots, n$ *is a concatenation of at most* $\log n$ *principal intervals.* The reduction in the size (at the expense of a small constant increase in the depth) comes from the fact that we only have to compute the carry generated from each of the corresponding $\log n$ principal intervals. The construction can be divided into several steps. Each step takes as inputs the computed values from previous steps.

Step 1: For $i = 0, 1, \cdots, (n-1)$, compute $z_i = (x_i \vee y_i)$ with one layer of $O(n)$ threshold gates.

Step 2: For each principal interval I, compute $f_p(I) = \bigwedge z_i = \bigwedge_{i \in I}(x_i \vee y_i)$ with another layer of $O(n)$ gates. f_p is to be thought of "propagate carry if it exists."

Step 3: For each principal interval $I = i_1 \cdots i_k$, compute the following:

$$f_s(I) = x_{i_k} y_{i_k} \vee \bigvee_{j=1}^{k-1} \left(x_{i_j} y_{i_j} \wedge \bigwedge_{l=j+1}^{k} z_l \right).$$

This requires two layers and $O(n \log n)$ gates. f_s is to be thought of "set carry."

Step 4: For $k = 1, \cdots, (n+1)$, the prefix $1 \cdots k$ is the concatenation of at most $\log n$ principal intervals $I_1, \cdots, I_l$. The kth carry bit c_k can be computed as

$$c_k = f_s(I_l) \vee \bigvee_{i=1}^{l-1} \left[f_s(I_i) \wedge \bigwedge_{j=i+1}^{l} f_p(I_j) \right]$$

with two layers and $O(n \log n)$ gates.

Clearly, the depth of our network is constant and the size is $O(n \log n)$. □

With the above two lemmas, we are now ready to show how to obtain a constant depth $O(n^{1+\epsilon})$ size neural network for MULTIPLE SUM.

Proof of Theorem 4: Suppose we are given n n-bit binary numbers: $x_i = 2^{n-1} x_{i_{n-1}} + 2^{n-2} x_{i_{n-2}} + \cdots + x_{i_0}$, $i = 1, \cdots, n$ and we want to compute their sum. We first show how to reduce this multiple sum to the sum of two numbers by a "block save" technique. This method is a generalization of the classical "carry save" technique in parallel addition and is first presented in [25]. By using Corollary 2, we show that

this reduction only needs a neural network of size $O(n^{1+\epsilon})$ and constant depth, for any $\epsilon > 0$.

Without loss of generality, we assume that $N = n/\log n$ and $\log n$ are integers, where log denotes logarithm to the base 2. Consider the following scheme: Partition each binary number x_i into N consecutive blocks $\tilde{x}_{i_0}, \tilde{x}_{i_1}, \cdots, \tilde{x}_{i_{N-1}}$ of $\log n$ bits each so that

$$x_i = \sum_{j=0}^{N-1} \tilde{x}_{i_j} \cdot 2^{\log n \cdot j}$$

where $0 \le \tilde{x}_{i_j} < 2^{\log n}$. Note that in binary representation, $\tilde{x}_{i_0} = x_{i_{\log n - 1}} x_{i_{\log n - 2}} \cdots x_{i_0}$ and $\tilde{x}_{i_{N-1}} = x_{i_{n-1}} \cdots x_{i_{n-2}} \cdots x_{i_{n-\log n}}$. We call a block $\tilde{x}_{i_j}$ "odd" or "even" if j is odd or even, respectively.

Let s_{odd} denote the sum of the n numbers when the "even" blocks are set to zero and s_{even} denote the sum when the "odd" blocks are set to zero. In other words,

$$s_{\text{odd}} = \sum_{j\text{odd}} \tilde{s}_j \cdot 2^{\log n \cdot j}; \qquad s_{\text{even}} = \sum_{j\text{even}} \tilde{s}_j \cdot 2^{\log n \cdot j}.$$

Then the sum of the original n numbers will be the sum of s_{odd} and s_{even}. We now show how to compute s_{odd} and s_{even} in parallel with a constant depth neural network of size $O(n^{1+\epsilon})$.

Observe that for each $j = 0, \cdots, N-1$, the block sum

$$\tilde{s}_j = \sum_{i=0}^{n-1} \tilde{x}_{i_j} < \sum_{i=0}^{n-1} 2^{\log n} = 2^{2\log n}$$

and thus can be represented in $2\log n$ bits. Since $\tilde{s}_j$ can be represented in $2\log n$ bits, there is no overlapping in the binary representation between

$$\tilde{s}_j \cdot 2^{\log n \cdot j} \quad \text{and}$$
$$\tilde{s}_{j+2} \cdot 2^{\log n \cdot (j+2)} = 2^{2\log n}\left(\tilde{s}_{j+2} \cdot 2^{\log n \cdot j}\right).$$

Therefore, we can sum each "odd" block $\tilde{s}_j$ in parallel and concatenate the resulting bits of each sum together to obtain s_{odd}. We can obtain s_{even} in a similar fashion in parallel. Now by Corollary 2, each bit of $\tilde{s}_j$ can be computed in depth $(d+1)$ and size $O(n^{\epsilon})$ with $\epsilon < 1/d$. Thus, the resulting two $O(n)$-bit numbers s_{odd} and s_{even} can be computed in a constant depth neural network of size $O(n^{1+\epsilon})$. To compute the final sum of s_{odd} and s_{even}, we only need another constant depth neural network of size $O(n\log n)$ by Lemma 4. So altogether the neural network has constant depth and size $O(n^{1+\epsilon})$. □

B. Comparison and Addition

Let $X = (x_1, \cdots, x_n)$, $Y = (y_1, \cdots, y_n) \in \{1,0\}^n$. We consider X and Y as two n-bit numbers representing $\sum_{i=1}^{n} x_i \cdot 2^i$ and $\sum_{i=1}^{n} y_i \cdot 2^i$, respectively.

Definition: The COMPARISON function is a Boolean function of $2n$ variables defined as

$$C_n(X,Y) = 1 \quad \text{iff } X \ge Y.$$

In other words,

$$C_n(X,Y) = \text{sgn}\left\{\sum_{i=1}^{n} 2^i(x_i - y_i) + 1\right\}.$$

The COMPARISON function was used in [23] to show that sorting of n n-bit numbers can be computed with a polynomial size depth-4 neural network. Notice that although the COMPARISON function can be expressed as a linear threshold function, some of the weights w_i's are exponentially large (in n). While it was shown in [23] that the COMPARISON function cannot be computed by a single linear threshold gate with *polynomially bounded* integer weights (i.e., $|w_i| \le n^{\alpha}$), it can be computed in a depth-2 network with $O(n^4)$ threshold gates and polynomially bounded weights. This is the best known result on the size of a depth-2 network (with polynomially large weights) computing the COMPARISON function. By increasing the depth by 1, we can indeed compute this function using only $O(n)$ gates.

We shall first show that COMPARISON cannot be computed in depth-2 (polynomial size) logic circuits of AND, OR gates. Since COMPARISON can be computed by polynomial-size depth-2 neural networks, the above result underscores the power of threshold gates. We shall then present a straightforward $O(n)$ gate realization using depth-3 networks.

Theorem 5: Any depth-2 logic circuit with only (unbounded fan-in) AND, OR gates that computes the COMPARISON function must have exponential size.

Proof: First consider any depth-2 logic circuit computing COMPARISON with AND gates in the first layer and an OR output gate in the second layer. Then the logic circuit is equivalent to a "sum of product" form (i.e., OR of AND's):

$$C_n(X,Y) = P_1 \vee P_2 \vee \cdots \vee P_m$$

where each "product" term P_i corresponds to the AND function of some subset of the variables (possibly negated), and the number of "products," m, corresponds to the number of AND gates in the first layer. There are many different "sum of product" forms for the COMPARISON function. We shall show that each of these forms must have exponentially many "product" terms, and thus the corresponding depth-2 logic circuit must have exponential size.

For convenience, we shall use $C_n(X,Y)$ to mean both the COMPARISON function and any of its "sum of product" forms (in the variables $x_1, \cdots, x_n, y_1, \cdots, y_n$ and their negations). Moreover, let $|C_n|$ and $|C_{n-1}|$ denote the number of product terms in any "sum of product" form of $C_n(X,Y)$ and $C_{n-1}(X,Y)$, respectively. Our idea in this proof is to show that $|C_n| \ge 2|C_{n-1}|$, thereby implying an exponential lower bound on $|C_n|$.

Using standard logic factorization, we can always write

$$\begin{aligned} C_n(X,Y) = {} & x_n y_n P^1_{n-1} \vee x_n \overline{y_n} P^2_{n-1} \vee \overline{x_n} y_n P^3_{n-1} \\ & \vee \overline{x_n}\,\overline{y_n} P^4_{n-1} \vee x_n P^5_{n-1} \vee \overline{x_n} P^6_{n-1} \vee y_n P^7_{n-1} \\ & \vee \overline{y_n} P^8_{n-1} \vee P^9_{n-1} \end{aligned}$$

where P^j_{n-1}'s are "sum of product" forms independent of the variables x_n and y_n.

Now, if $x_n = 0$ and $y_n = 1$, then $C_n(X,Y) = 0$, irrespective of the assignments to the other variables, $x_{n-1}, \cdots, x_1$, and $y_{n-1}, \cdots, y_1$. Hence, substituting $x_n = 0$ and $y_n = 1$ in the above expression for $C_n(X,Y)$, we obtain that $P^3_{n-1} \equiv$

$P^6_{n-1} \equiv P^7_{n-1} \equiv P^9_{n-1} \equiv 0$. Thus, the expression for $C_n(X,Y)$ simplifies to

$$C_n(X,Y) = x_n y_n P^1_{n-1} \vee x_n P^5_{n-1} \vee \overline{x_n}\,\overline{y_n} P^4_{n-1} \vee \overline{y_n} P^8_{n-1} \vee x_n \overline{y_n} P^2_{n-1}.$$

Next, we observe that if $x_n = 1$ and $y_n = 1$, or $x_n = 0$ and $y_n = 0$, then $C_n(X,Y) \equiv C_{n-1}(X,Y)$. Hence, substituting these values of x_n and y_n in the above expression of $C_n(X,Y)$, we get: $P^1_{n-1} \vee P^5_{n-1} \equiv C_{n-1}(X,Y)$ and $P^4_{n-1} \vee P^8_{n-1} \equiv C_{n-1}(X,Y)$. It follows then from the above expression of $C_n(X,Y)$ that the number of "product" terms in $C_n(X,Y)$ is at least twice the number of "product" terms in $C_n(X,Y)$, i.e.,

$$|C_n| \geq 2|C_{n-1}|.$$

Thus, $C_n(X,Y)$ must have exponential number of product terms in any "sum of product" form and the corresponding depth-2 logic circuit must have exponential size.

A similar dual argument applies to depth-2 logic circuit with OR gates in the first layer and an output AND gate in the second layer, and easily shows that there must be exponentially many OR gates for the COMPARISON function. □

Theorem 6: The COMPARISON function can be computed in a depth-3 neural network of size $3n$ with polynomially bounded integer weights.

Proof: We first write a Boolean expression for the COMPARISON function $C_n(X,Y)$ as a recursion on n.

$$C_1(X,Y) = x_1 \vee \overline{y_1} = \begin{cases} 1 & \text{if } x_1 \geq y_1 \\ 0 & \text{otherwise} \end{cases}$$

$$C_n(X,Y) = (x_n \wedge \overline{y_n}) \bigvee [(x_n \vee \overline{y_n}) \wedge C_{n-1}(X,Y)]$$

$$= \begin{cases} 1 & \text{if } x_n > y_n \text{ or } (x_n \geq y_n \text{ and } \sum_{i=1}^{n-1} 2^i x_i \geq \sum_{i=1}^{n-1} 2^i y_i) \\ 0 & \text{otherwise.} \end{cases}$$

Define Boolean expressions

$$B_n = x_n \wedge \overline{y_n}$$

$$B_k = (x_k \vee \overline{y_k}) \bigwedge_{j=k+1}^{n} (x_j \wedge \overline{y_n}) \quad \text{for } k = 2, \cdots, n-1$$

$$\text{and} \quad B_1 = \bigwedge_{j=1}^{n} (x_j \wedge \overline{y_j})$$

Using the recursive Boolean expressions of $C_n(X,Y)$, it is easy to show by induction on n that $C_n(X,Y) = \bigvee_{j=1}^{n} B_j$.

The first layer of our network for the COMPARISON function has $2n-1$ gates computing the $(x_j \wedge \overline{y_j})$ and $(x_j \vee \overline{y_j})$. With these computed values as inputs, the second layer has n gates each computing the B_j. Finally the output gate computes the OR $(\bigvee)$ of all the B_j's. The total number of gates is $3n$ as claimed. □

Note that we have not made use of the full power of threshold gates in the above depth-3 neural network for COMPARISON. We simply use the threshold gates to simulate the AND $(\bigwedge)$, OR $(\bigvee)$ logic operations.

Definition: The multioutput ADDITION function is defined to be the $n+1$ bit representation of the sum of two n-bit numbers X and Y.

Each bit of the sum was known to be computable in a depth-2 neural network. The best known result on the size of neural network for ADDITION is $O(n^4)$ [25] (with depth-2). The size can be significantly reduced to $O(n)$ if we increase the depth by 1. Since there are $n+1$ bits in the final sum, the total size of the network computing ADDITION is $O(n^2)$.

Theorem 7: The multioutput function ADDITION of two n-bit numbers can be computed in a depth-3 network of (unbounded fan-in) AND, OR gates and size $O(n^2)$.

Proof: Let $X = (x_{n-1}, \cdots, x_0)$, $Y = (y_{n-1}, \cdots, y_0) \in \{1,0\}^n$ be the two n-bit numbers with x_0 and y_0 being the least significant bits, and $S = (s_n, \cdots, s_0)$ be the $n+1$-bit sum. Let c_k be the kth carry bit, i.e., c_k is the most significant bit of the sum $\sum_{i=0}^{k-1} 2^i (x_i + y_i)$ with $c_0 = 0$. The main idea is to compute the carry bits c_k's and their negations $\overline{c_k}$'s in parallel using depth-2 network. Then the sum bits s_k can be computed as follows:

$$\begin{aligned} s_k &= c_k \oplus x_k \oplus y_k \\ &= (\overline{c_k} x_k \overline{y_k}) \vee (\overline{c_k}\,\overline{x_k} y_k) \vee (c_k x_k y_k) \vee (c_k \overline{x_k}\,\overline{y_k}). \end{aligned}$$

The network computing the carry bit c_k is very similar to the network computing the COMPARISON function. Note that $c_k = 1$ iff there is a $j \leq k-1$ such that $x_j = y_j = 1$ but for all $j < i \leq k-1$, we do not have $x_i = y_i = 0$. Thus,

$$\begin{aligned} c_k = x_{k-1} y_{k-1} &\bigvee [(x_{k-1} \vee y_{k-1}) x_{k-2} y_{k-2}] \\ &\bigvee [(x_{k-1} \vee y_{k-1})(x_{k-2} \vee y_{k-2}) x_{k-3} y_{k-3}] \cdots \\ &\bigvee [(x_{k-1} \vee y_{k-1})(x_{k-2} \vee y_{k-2})(x_{k-3} \vee y_{k-3}) \cdots x_0 y_0] \end{aligned}$$

$$\begin{aligned} \overline{c_k} = \overline{x_{k-1}}\,\overline{y_{k-1}} &\bigvee [(\overline{x_{k-1}} \vee \overline{y_{k-1}}) \overline{x_{k-2}}\,\overline{y_{k-2}}] \\ &\bigvee [(\overline{x_{k-1}} \vee \overline{y_{k-1}})(\overline{x_{k-2}} \vee \overline{y_{k-2}}) \overline{x_{k-3}}\,\overline{y_{k-3}}] \cdots \\ &\bigvee [(\overline{x_{k-1}} \vee \overline{y_{k-1}})(\overline{x_{k-2}} \vee \overline{y_{k-2}})(\overline{x_{k-3}} \vee \overline{y_{k-3}}) \cdots (\overline{x_0} \vee \overline{y_0})]. \end{aligned}$$

As in the case of the COMPARISON network, define Boolean expressions

$$B^k_k = x_k y_k$$

$$B^k_j = (x_j y_j) \bigwedge_{i=j+1}^{k} (x_i \vee y_i) \quad \text{for } j = 0, \cdots, k-1$$

$$\tilde{B}^k_k = \overline{x_k}\,\overline{y_k}$$

$$\tilde{B}^k_j = (\overline{x_j}\,\overline{y_j}) \bigwedge_{i=j+1}^{k} (\overline{x_i} \vee \overline{y_i}) \quad \text{for } j = 1, \cdots, k-1 \quad \text{and}$$

$$\tilde{B}^k_0 = \bigwedge_{i=0}^{k} (\overline{x_i} \vee \overline{y_i}).$$

The first layer of our network consists of $2n$ gates computing the $(x_i \vee y_i)$ and $(\overline{x_i} \vee \overline{y_i})$ for $i = 1, \cdots, n$. With these computed values, the second layer, which consists of $O(n^2)$ gates, computes for each $k = 1, \cdots, n$, $0 \leq j \leq k$, $(\tilde{B}^{k-1}_j x_k \overline{y_k})$, $(\tilde{B}^{k-1}_j \overline{x_k} y_k)$, $(B^{k-1}_j x_k y_k)$, and $(B^{k-1}_j \overline{x_k}\,\overline{y_k})$. The third layer has $n+1$ output gates and computes each sum

bit s_k that is the OR of all these values:

$$s_k = \bigvee_{j=0}^{k-1} (\tilde{B}_j^{k-1} x_k \overline{y_k}) \vee (\tilde{B}_j^{k-1} \overline{x_k} y_k) \vee (B_j^{k-1} x_k y_k) \vee (B_j^{k-1} \overline{x_k}\,\overline{y_k}).$$

Clearly the size is dominated by the number of gates in the second layer, which is $O(n^2)$. □

As for the COMPARISON function, it is interesting to note that ADDITION cannot be computed in depth-2 (polynomial size) logic circuits of AND, OR gates.

Theorem 8: Any depth-2 logic circuit with only (unbounded fan-in) AND, OR gates that computes the ADDITION of two n-bit numbers must have exponential size.

Proof: We use the same notations as in the proof of Theorem 6 and show that the nth sum bit s_n requires an exponential size depth-2 logic circuit (with AND, OR gates only). Since $s_n = c_n \oplus x_n \oplus y_n$ and $s_n = c_n$ with $x_n = y_n = 0$, it suffices to show an exponential lower bound on the size of the circuit for c_n. The proof can be carried out almost exactly as in the case of the COMPARISON function (simply by renaming the variables). First derive a recursion in the "sum of product" or "product of sum" forms for c_n in terms of c_{n-1}. Then show that the number of terms $|c_n| \geq 2|c_{n-1}|$ and thus $|c_n|$ grows exponentially in n. □

V. Concluding Remarks

We have studied the tradeoffs between the depth and the size in neural networks. We show that for symmetric functions and some arithmetic functions, a significant reduction in the size is possible at the expense of a small constant increase in depth. In the process, we have developed several neural networks which have the minimum size among all the known constructions. For example, our construction of $O(\sqrt{n})$ size networks for symmetric functions appears to be the best known even for unbounded depth networks. Similarly, our results on the size of constant depth networks for MULTIPLE SUM and multiplication is the best known.

A natural continuation of this work would be to investigate the depth–size tradeoffs in neural computation for larger classes of functions. Our work has also resulted in several interesting theoretical open questions. For example, can one derive less than $O(\sqrt{n})$ size neural networks for computing arbitrary symmetric functions? Lower bound results in [27] suggest that a substantial reduction in size is not possible without increasing the depth beyond 3. Similar questions remain open for the other functions studied in this paper.

References

[1] P. W. Beame, S. A. Cook, and H. J. Hoover, "Log depth circuits for division and related problems," *SIAM J. Comput.*, vol. 15, pp. 994–1003, 1986.

[2] J. Bruck, "Harmonic analysis of polynomial threshold functions," *SIAM J. Discrete Math.*, May 1990.

[3] J. Bruck and R. Smolensky, "Polynomial threshold functions, AC^0 functions and spectral norms," Tech. Rep. RJ 7140, IBM Research, Nov. 1989. Appeared in *Proc. IEEE Symp. Foundations Comput. Sci.* Oct. 1990.

[4] A. K. Chandra, L. Stockmeyer, and R. Lipton, "Unbounded fan-in circuits and associative functions," in *Proc. 15th ACM Symp. Theory Comput.*, 1983, pp. 52–60.

[5] A. K. Chandra, L. Stockmeyer, and U. Vishkin, "Constant depth reducibility," *SIAM J. Comput.*, vol. 13, pp. 423–439, 1984.

[6] F. Fich, "Two problems in concrete complexity: Cycle detection and parallel prefix computation," Ph.D. dissertation, Univ. of California, Berkeley, 1982.

[7] M. Furst, J. B. Saxe, and M. Sipser, "Parity, circuits and the polynomial-time hierarchy," in *Proc. IEEE Symp. Foundations Comput. Sci.*, vol. 22, 1981, pp. 260–270.

[8] A. Hajnal, W. Maass, P. Pudlak, M. Szegedy, and G. Turan, "Threshold circuits of bounded depth," *IEEE Symp. Foundations Comput. Sci.*, vol. 28, 1987, pp. 99–110.

[9] T. Hofmeister, W. Hohberg, and S. Kohling, "Some notes on threshold circuits, and multiplication in depth 4," unpublished manuscript, 1990.

[10] W. Kautz, "The realization of symmetric switching functions with linear-input logical elements," *IRE Trans. Electron. Comput.*, vol. EC-10, 1961.

[11] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *J. ACM*, vol. 27, no. 4, pp. 831–838, Oct. 1980.

[12] N. Linial, Y. Mansour, and N. Nisan, "Constant depth circuits, Fourier transforms, and learnability," in *Proc. 30th IEEE Symp. Foundations Comput. Sci.*, 1989.

[13] R. Minnick, "Linear-input logic," *IEEE Trans. Electron. Comput.*, vol. EC-10, 1961.

[14] D. E. Muller and F. P. Preparata, "Bounds to complexities of networks for sorting and for switching," *J. ACM*, vol. 22, pp. 195–201, 1975.

[15] S. Muroga, "The principle of majority decision logic elements and the complexity of their circuits," in *Proc. Int. Conf. Inform. Processing*, Paris, France, June 1959.

[16] R. Paturi and M. Saks, "On threshold circuits for parity," in *Proc. IEEE Symp. Foundations Comput. Sci.*, Oct. 1990.

[17] N. Pippenger, "The complexity of computations by networks," *IBM J. Res. Develop.*, vol. 31, no. 2, Mar. 1987.

[18] N. P. Redkin, "Synthesis of threshold circuits for certain classes of Boolean functions," *Kibernetika*, no. 5, pp. 6–9, 1970.

[19] J. Reif, "On threshold circuits and polynomial computation," in *Proc. 2nd Structure in Complexity Theory Conf.*, 1987, pp. 118–123.

[20] V. P. Roychowdhury, K. Y. Siu, A. Orlitsky, and T. Kailath, "On the circuit complexity of neural networks," in *Advances in Neural Information Processing Systems 3*, D. S. Touretzky and R. Lippman Eds., Apr. 1991.

[21] V. Roychowdhury, K. Y. Siu, A. Orlitsky, and T. Kailath, "A geometric approach to threshold circuit complexity," in *Proc. 4th Annu. Workshop Computational Learning Theory (COLT)*, Aug. 1991.

[22] C. Shannon, "The synthesis of two-terminal switching circuits," *Bell Syst. Tech. J.*, vol. 28, pp. 59–98, 1949.

[23] K. Y. Siu and J. Bruck, "On the power of threshold circuits with small weights," *SIAM J. Disc. Math.*, Aug. 1991.

[24] ——, "Neural computation of arithmetic functions," *Proc. IEEE*, vol. 78, pp. 1669–1675, Oct. 1990.

[25] ——, "On the dynamic range of linear threshold elements," Tech. Rep. RJ 7237, IBM Research, Jan. 1990.

[26] K. Y. Siu, J. Bruck, and T. Kailath, "Depth-efficient neural networks for division and related problems," *IEEE Trans. Inform. Theory*, submitted for publication.

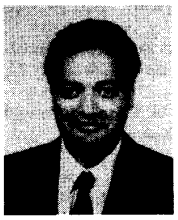
[27] K. Y. Siu, V. P. Roychowdhury, and T. Kailath, "Computing with almost optimal size threshold circuits," in *Proc. Int. Symp. Inform. Theory*, Budapest, Hungary, June 1991.

[28] L. Stockmeyer, "On the combinational complexity of certain symmetric Boolean functions," *Math. Syst. Theory*, vol. 10, pp. 323–336, 1977.

[29] I. Wegener, *The Complexity of Boolean Functions*. New York: Wiley, 1987.

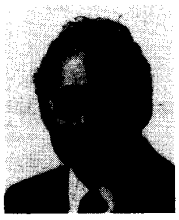
Kai-Yeung Siu received the B.Sc. degree in mathematics and computer science from New York University, New York, NY, and the B.Eng. degree in electrical engineering from The Cooper Union, NY, both in 1987. He pursued his graduate studies at Stanford University, Stanford, CA, and received the M.Sc. degree and the Ph.D. degree in electrical engineering in 1988 and 1991, respectively.

He is currently an Assistant Professor in the Department of Electrical and Computer Engineering, University of California, Irvine. His research interests include artificial neural networks, fault-tolerant computation, and parallel algorithms.



Vwani P. Roychowdhury received the B.Tech degree from the Indian Institute of Technology, Kanpur, the M.S. degree from University of Rochester, Rochester, NY, and the Ph.D. degree from Stanford University, Stanford, CA, in 1982, 1983, and 1988, respectively, all in electrical engineering.

He is currently an Assistant Professor in the Department of Electrical Engineering, Purdue University. His research interests include parallel algorithms and architectures, design and analysis of neural networks, special purpose computing arrays and VLSI design, and fault-tolerant computation.



Thomas Kailath (S'57–M'62–F'70) was educated in Poona, India, and at the Massachusetts Institute of Technology (S.M., 1959; Sc.D., 1961).

From October 1961 to December 1962, he worked at the Jet Propulsion Laboratories, Pasadena, CA, where he also taught part-time at the California Institute of Technology. He joined Stanford University as an Associate Professor of Electrical Engineering in 1963. He has been a Full Professor since 1968, served as Director of the Information Systems Laboratory from 1971 to 1980, as Associate Department Chairman from 1981 to 1987, and currently holds the Hitachi America Professorship in Engineering. He has held short term appointments at several institutions around the world. He has worked in a number of areas including information theory, communications, computation, control, linear systems, statistical signal processing, stochastic processes, linear algebra and operator theory; his recent research interests include array processing, fast algorithms for nonstationary signal processing, and the design of special purpose computing systems. He is the author of *Linear Systems* (Englewood Cliffs, NJ: Prentice-Hall, 1980) and *Lectures on Wiener and Kalman Filtering* (Berlin, Germany: Springer-Verlag, 1981).

Dr. Kailath has held Guggenheim, Churchill, and Royal Society fellowships, among others, and received awards from the IEEE Information Theory Society and the American Control Council, in addition to the Technical Achievement and Society Awards of the IEEE Signal Processing Society in 1989 and 1991. He served as President of the IEEE Information Theory Society in 1975, and was awarded an honorary doctorate from Linköping University, Sweden, in 1990. He is a Fellow of the Institute of Mathematical Statistics and is a member of the National Academy of Engineering.