Fast Arithmetic Computing with Neural Networks

Kai-Yeung Siu [1]
Information Systems Laboratory
Stanford University

Jehoshua Bruck
IBM Research Division
Almaden Research Center

Abstract— **The basic processing unit of a neural network is a linear threshold element. It was known that neural networks can be much more powerful than traditional logic circuits, assuming that each threshold element can be built with a cost that is comparable to that of AND, OR, NOT logic elements. Whereas any logic circuit of polynomial size (in n) that computes the product of two n-bit numbers requires unbounded delay, such computations can be done in a neural network with 'constant' delay. We improve some known results by showing that the product of two n-bit numbers and sorting of n n-bit numbers can both be computed by a polynomial size neural network using only 4 unit delays, independent of n. Moreover, the weights of each threshold element in our neural networks require only $O(\log n)$-bit (instead of n-bit) accuracy.**

I. Introduction

Neural networks can be viewed as circuits of highly interconnected parallel processing units called 'neurons'. The most commonly used models of neurons are linear threshold gates or, when continuity or differentiability is required, elements with a sigmoid input-output function. Because of the recent advance in VLSI technology, neural network has also emerged as a new technology and has found wide application in many areas.

Much of the current research in neural networks is in the area of pattern classification and is concerned with developing efficient 'learning' algorithms for adjusting the interconnection weights adaptively to perform the desired classification. Heuristics such as the 'back propagation algorithm' has obtained surprisingly good empirical results [1]. In this paper, we shall look at another area of application of neural networks. Our model of a neuron is the linear threshold gate, and the network architecture considered here is the layered feedforward network. We shall see how common arithmetic functions such as multiplication and sorting can be efficiently computed in a 'shallow' neural network. Whereas the interconnection weights are modified adaptively for different inputs in pattern classification and the desired classification is usually only approximated, in our network the weights are fixed for all inputs and the desired function is computed exactly. We shall confine our attention to operations on numbers represented in binary and we assume the inputs are encoded in $\{1,-1\}$ instead of $\{1,0\}$. Little would change in our analysis if we adopted the conventional $\{1,0\}$ encoding since the transformation $\{1,-1\} \rightarrow \{1,0\}$ can easily be done by $x \rightarrow (x+1)/2$. Before presenting our main results, we first introduce our model of feedforward neural networks.

II. Feedforward Neural Network

The classical model of a neuron [2] is a device which computes a *linear threshold function* $f : \{1,-1\}^n \rightarrow \{1,-1\}$ such that

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

[1]This work was done while the author was a research student associate at IBM Almaden Research Center and was supported in part by the Joint Services Program at Stanford University (US Army, US Navy, US Air Force) under Contract DAAL03-88-C-0011, and the Department of the Navy (NAVELEX) under Contract N00039-84-C-0211, NASA Headquarters, Center for Aeronautics and Space Information Sciences under Grant NAGW-419-S6.

where

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

Without loss of generality, we can assume $F(X) \neq 0$ for all $X \in \{1, -1\}^n$. The real valued coefficients w_i are commonly referred to as the *weights* of the threshold function.

In the definition of the classical model of a neuron, the weights can take on real values. Since we would like to implement a neuron using analog device, from a practical point of view, it is important to see if the assumption of real valued weights is necessary. In other words, can all linear threshold functions be realized if the weights are of finite precision? Actually, it was known [3] that each of the weights in a linear threshold function of n variables can be assumed to be integers of $O(n \log n)$ bits. However, this still allows the weights to grow exponentially fast with the number of input variables. In fact, most linear threshold functions have weights which must grow exponentially fast [4]. This fact can also be interpreted as the necessity of high accuracy and high sensitivity of parameters in actual implementation of artificial neurons. In the rest of this paper, we shall consider a restricted class of neurons which is more practical as a computational model. More formally, each function $f(X) = sgn(\sum_{i=1}^{n} w_i \cdot x_i + w_0)$ computed by this restricted class is characterized by the property that the weights w_i are integers and bounded by a polynomial in n, i.e. $|w_i| \leq n^c$ for some constant $c > 0$. We shall denote this class of functions as $\widehat{LT}_1$ (restricted linear threshold) functions.

A feedforward network is a network of interconnected functional gates with no feedback. The *depth* of a gate is defined to be the length of the longest path (each connection wire between gates is a unit length) from the inputs to that gate. The *depth* of the network is defined to be the maximum depth of all output gates. 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. Given an assignment of the inputs, the output values of the network are obtained by evaluation of the gates in increasing depth order. The depth of the network can therefore be interpreted as the time for its parallel execution of the function to be computed.

We define a *neural network* to be a feedforward network of $\widehat{LT}_1$ elements. We denote $\widehat{LT}_k$ to be the class of neural networks of *depth k* with the *size bounded by a polynomial* in the number of inputs. 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 gates can be simulated by $\widehat{LT}_1$ elements.

Loosely speaking, a network is 'shallow' if it has small depth. In the rest of this paper, we shall only be interested in network whose size is bounded by a polynomial in the number of inputs. Before we see how shallow neural network can compute arithmetic functions, we first review the classical implementation using AND, OR, NOT logic elements and the limitation of constant depth circuits of such elements. This is the subject of next section.

III. Classical Implementation of Arithmetic Functions

It is well known among experienced circuit designers that they cannot implement some common functions such as parity and multiplication with small programmable logic arrays (PLA's), a type of integrated circuit used inside microprocessors to compactly represent many functions. Thus it is of both practical and theoretical interest to see if such difficulty is unavoidable. It turns out that PLA's are well modelled by bounded-depth circuits of AND, OR, NOT logic elements with arbitrary fanin. However, it was shown in [5], [6] that an exponential number of gates is necessary to implement the parity function in a bounded-depth logic circuit. All these results can be interpreted as proofs that any PLA implementing parity must need an exponential amount of chip area and thus establish a basis for the common belief among circuit designers.

Another way of interpreting these results is that any parity circuit which uses polynomial amount of chip area must have delay which increases with the size of the inputs. By introducing the notion of constant-depth reduction [7], similar results can be shown for other common functions such as multiplication and division. In fact, multipliers of current use require $O(\log n)$ delays for input numbers of n-bits. The lower bound results also imply that the minimum possible delay for multipliers of polynomial size is $\Omega(\log n / \log \log n)$.

We can explain the above negative results by the fact

that the basic processing logic elements AND, OR, NOT of the circuits are not powerful enough. In practical implementation, these logic elements are built using analog devices; perhaps we can build a more powerful gate out of analog devices to increase the computational power of the circuit? In the previous section, because of practical issues of implementation, we have introduced a new model of a neuron called $\widehat{LT_1}$ element as the basic building block in our neural network. In fact, the main theme of this paper is to see how a 'shallow' neural network of polynomial size can compute common functions such as multiplication and sorting with small delay. The fact that these two functions can be computed in a 'constant-depth' neural network was shown in [7]; however, their construction is not 'depth'-efficient and it is not explicitly stated how many constant layers are needed in each step of their construction. The novelty in our ideas is the application of a 'depth'-efficient algorithm and the results in [4] to greatly reduce the constant to obtain a 'shallow' network.

IV. Main Results

In this section, we shall focus on the computational capability of the feedforward neural network model introduced in section II. We assume that each neuron takes a unit delay to compute and we shall only consider neural networks of polynomial size. Because of space limitation, all results are stated without proofs.

A. Addition and Multiplication

Using the carry-look-ahead method, it was known that the sum of 2 n-bit numbers can be computed in a bounded-depth logic circuits of polynomial size with arbitrary fanin. In fact, we showed in [4] that a 2-layer neural network suffices. Since the least significant bit of the sum is the EXCLUSIVE-OR function of the least significant bits of the 2 numbers, which is not a linear threshold function, it follows that the sum cannot be computed using only one layer. Thus a 2-layer neural network is depth-optimal.

Whereas the sum of 2 n-bit numbers can be computed in bounded-depth logic circuits, the sum of n n-bit numbers cannot be computed in fixed delay. Surprisingly, such multiple sum can be computed in a neural network with only 3 layers. We state this result as the following theorem:

Theorem 1 *The sum of n n-bit numbers is computable in $\widehat{LT_3}$.*

It is not hard to show that computing the product of 2 n-bit numbers can be reduced to computing the sum of n $2n$-bit numbers, using one more layer. So we have the following:

Theorem 2 *The product of two n-bit numbers is computable in $\widehat{LT_4}$.*

B. Sorting

In sorting, we assume that the input is a list of n n-bit binary numbers and the output will be the same list sorted in nondecreasing order. A number wich appears m times in the input list will be duplicated m times in the output list.

The basic operation in sorting is the comparison of 2 numbers, i.e. given 2 n-bit binary numbers $x = x_n x_{n-1} \cdots x_1$, $y = y_n y_{n-1} \cdots y_1$, we want to compute whether $x \geq y$ or not. It is tempting to conclude that comparison can be computed in a single layer since

$$x \geq y \quad \text{iff} \quad \text{sgn}\{\sum_{i=1}^{n} 2^i \cdot (x_i - y_i)\} = +1$$

However, notice that the weights chosen above are exponential in n and thus do not satisfy the conditions in our definition of a $\widehat{LT_1}$ element. In fact, it was shown in [4] that the comparison function cannot be computed using a single $\widehat{LT_1}$ element, but it can be computed in 2 layers of $\widehat{LT_1}$ elements. Using this result and the techniques in [7], we obtain the following:

Theorem 3 *Sorting of n n-bit numbers is computable in $\widehat{LT_4}$.*

C. Extensions to other functions

It is natural to continue our study of neural network on computation of more complicated arithmetic functions such as exponentiation, division, and extraction of square roots. In fact, it can be shown that multiplication of n n-bit numbers and division of 2 n-bit numbers can also be computed by a constant-depth neural network using the results in [8]. However, the constant obtained by direct application of the algorithm in [8] will be too large for the

resulting neural network to be considered as 'shallow', and therefore the difference between $\log n$ and the constant in the delay is not significant unless the input numbers is astronomically large. These results are of theoretical importance, however, because they describe the fundamental difference in computation between neural networks and logic circuits. At this time, we are not able to reduce the constant to obtain a 'shallow' neural network for division and multiple product. We refer the more interested readers to [8, 9] for other details.

V. Conclusion

In this paper, we have introduced a restricted model of a 'neuron' which is more practical as a model of computation than the classical model of a 'neuron'. We define our model of neural networks as a feedforward network of such neurons. We have shown how some common arithmetic functions such as multiple addition, multiplication and sorting can be computed by a polynomial size 'shallow' neural network with 3, 4, 5 unit delays respectively, whereas it was known that these functions cannot be computed in constant-depth logic circuits. Applying the techniques in [8], these results can be extended to more complicated functions such as multiple products, divisions, and rational functions.

The moral of our study is to see that parallel computer based on threshold elements could be immensely powerful. Of course, all these results are based on the assumption that a linear threshold ($\widehat{LT_1}$) element can be implemented using analog devices whose unit cost is small. This would justify research in device technology to investigate the feasibility of building such elements with small cost. In fact, there has already been some research progress in this area as mentioned in [9].

Acknowledgement

The first author would like to thank Prof. Thomas Kailath for his guidance, constant encouragement, and financial support.

References

[1] J. L. McClelland D. E. Rumelhardt and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1.* MIT Press, 1986.

[2] M. Minsky and S. Papert. *Perceptrons.* The MIT Press, expanded edition, 1988.

[3] P. Raghavan. Learning in threshold networks: A computation model and applications. Technical Report RC 13859, IBM Research, July 1988.

[4] K. Y. Siu and J. Bruck. On the Dynamic Range of Linear Threshold Elements. Technical Report RJ 7237, IBM Research, January 1990.

[5] J. Hastad. Almost optimal lower bounds for small depth circuits. *ACM Symp. Theor. Computing*, 18:6–20, 1986.

[6] R. Smolensky. Algebaic methods in the theory of lower bounds for Boolean circuit complexity. *ACM Symp. Theor. Computing*, 19:77–82, 1987.

[7] A. K. Chandra, L. Stockmeyer, and U. Vishkin. Constant depth reducibility. *Siam J. Comput.*, 13:423–439, 1984.

[8] P. W. Beame, S. A. Cook, and H. J. Hoover. Log depth circuits for division and related problems. *Siam J. Comput.*, 15:994–1003, 1986.

[9] J. Reif. On Threshold Circuits and Polynomial Computation. *Second Annual Structure in Complexity Theorey Symp.*, pages 118–123, 1987.