Depth-optimal $O(n)$-node Neural Networks for $n$-bit addition

Piyush Gupta and P. G. Poonacha
Department of Electrical Engineering,
Indian Institute of Technology,
Bombay 400 076, India.

Abstract

In this paper, we study the problem of depth-optimal $n$-bit addition using feed-forward Neural Networks with threshold type nodes. Three two-layer architectures are given requiring respectively $(3n - 1)$, $2n$ and $n$ neurons in the first layer, and $n$ neurons in the second layer. The second and third architectures have been shown to be optimal in the number of neurons for two-bit addition respectively for when inputs are fed to the first layer only and when fed to both the layers. It has been conjectured that they are optimal for any bit size addition. Moreover, architectures proposed have been extended to perform $k$ $n$-bit addition in two layers and multiplication of two $n$-bit numbers in three layers, requiring respectively $O(kn)$ and $O(n^2)$ neurons.

1 Introduction

Conventional adders such as ripple carry adders and carry look ahead adders for adding two $n$-bit numbers require a delay which depends on $n$. In high speed and high precision signal processing and computer applications, it is desirable to have an adder whose delay is independent of the number of input bits. Towards this end multi layer neural networks can be used [1], Hopfield neural networks can also be used [2]. However, they are not constant delay networks.

The approach taken is that of developing application-specific neural net architectures. This becomes essential as training even a three node neural network of linear threshold gates has been shown to be NP-complete [3]. Moreover, the number of training examples increases exponentially with the bit size of the numbers to be added making training more difficult. Thus, in this paper we give three two-layer feedforward Neural Network architectures for binary addition. All the networks consist of linear threshold gates as processing nodes.

\footnote{This work was supported by an MHRD (India) project grant for Laboratory for Artificial Neural Networks.}
The remainder of the paper is divided into four sections. In §2, we derive three architectures to perform binary addition. In §3, we extend these architectures to perform other arithmetic functions. In §4, we describe the approach taken to prove optimality of two of these architectures for two bit addition, and possible generalization for n-bit addition. In §5, we conclude with some open problems and future work.

2 Development of New Architectures

In this section, we propose three architectures for binary addition. Thus, let $a_i$ and $b_i$, $i = 0, 1, \ldots, n - 1$ denote $i^{th}$ bits of two $n$-bit numbers $A$ and $B$ with $a_{n-1}$ and $b_{n-1}$ denoting most significant bits (MSB).

2.1 architecture 1 (A1)

Let us consider generation of $(j + 1)^{th}$ bit $s_j$ of the sum $S$ of two numbers $A$ and $B$. For which, let

$$X_j \overset{\text{def}}{=} \sum_{i=0}^{j} x_i \cdot 2^i \quad \text{for} \quad 0 \leq j \leq (n - 1) \quad (1)$$

Since, $s_j$ depends on $(j + 1)^{th}$ and lower bits of $A$ and $B$, let

$$K_j \overset{\text{def}}{=} A_j + B_j = c_j \cdot 2^{j+1} + S_j = c_j \cdot 2^{j+1} + s_j \cdot 2^j + S_{j-1} \quad (2)$$

where, $c_j$ denotes the carry generated after addition of the first $(j + 1)$ bits. From definition (1), it is easy to see that

$$0 \leq S_{j-1} < 2^j \quad (3)$$

Substituting (2) in (3), we get

$$2^j \cdot (2c_j + s_j) \leq K_j < 2^j \cdot (1 + 2c_j + s_j) \quad (4)$$

Table 1 tabulates (4) for various possible values of $c_j$ and $s_j$. Now, if $x$ is the weighted sum of inputs to a threshold gate with threshold $\theta$, then output of the gate is given by

$$f_0(x) \begin{cases} 1 & \text{if} \quad x \geq \theta \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Thus, $f_{i,2}(K_j)$ for $i = 1, 2, 3$ of Table 1 can be realized using three neurons in the first layer. These neurons determine the range in which $K_j$ lies, therefore, separating out $1^a$ and $0^s$ of $s_j$ as shown in Figure 1. One such separating plane is

$$f_{2^i}(K_j) - f_{2^i-2}(K_j) + f_{2^i-2^i}(K_j) = 0.5 \quad (6)$$

Above architecture is illustrated for two-bit addition in Figure 2. Note that $s_0$ requires only two neurons in the first layer as the fourth possibility of Table 1 does not exist for
<table>
<thead>
<tr>
<th>$c_j$</th>
<th>$s_j$</th>
<th>$K_j = A_j + B_j$</th>
<th>$f_{2^j}(K_j)$</th>
<th>$f_{2 \cdot 2^j}(K_j)$</th>
<th>$f_{3 \cdot 2^j}(K_j)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$0 \leq 2^j$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$2^j \leq 2 \cdot 2^j$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$2 \cdot 2^j \leq 3 \cdot 2^j$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$3 \cdot 2^j \leq 4 \cdot 2^j$</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: Range of $K_j$ for various possible values of $s_j$ and $c_j$.

Figure 1: Separation of $s_j$ from $f_{2^j}(K_j), f_{2 \cdot 2^j}(K_j)$ and $f_{3 \cdot 2^j}(K_j)$. ($X_i$’s denote don’t care combinations).
2.2 Architecture 2 (A2)

Table 1 shows that to separate out four regions of $s_j$, $c_j$ three hyper planes are used. Thus, of the eight regions generated by these hyper planes, only four are used to separate out $1^*$ and $0^*$ of $s_j$. This suggest some redundancy in previous architecture. Hence, to further reduce the size of the first layer, let us consider various possibilities of $s_j$ in terms of $a_j$, $b_j$, and $c_{j-1}$ as shown in Table 2.

From Table 2 it is easy to see that $s_j$ cannot be extracted just from $c_j$ and $c_{j-1}$. However, a threshold function $f(a_j, \ldots, a_1, a_0, b_0, \ldots, b_j, b_0)$ may help $s_j$ to become a threshold function of $c_j, c_{j-1}$ and $f$. Now, of the three variables $a_j$, $b_j$ and $c_{j-1}$ on which $s_j$ is symmetrically dependent, only $c_{j-1}$ is available at the second layer. Hence, the required
function may be symmetric with respect to $a_j$ and $b_j$, but need not be so with $c_{j-1}$. One such function is $f_1(a_j + b_j - c_{j-1})$. We now prove the following theorem.

**Theorem 1** Let $X_j = (a_j + b_j) \cdot 2^j - \sum_{i=0}^{j-1}(a_i + b_i) \cdot 2^i$. Then,

(a) $f_1(a_j + b_j - c_{j-1}) = f_\theta(X_j)$ for $0 < \theta < 1$.

(b) $s_j$ is a threshold function of $c_{j-1}$, $c_j$ and $f_\theta(X_j)$.

**Proof**

(a): Using (2), we can rewrite $X_j$ as

$$X_j = (a_j + b_j) \cdot 2^j - (S_{j-1} + c_{j-1} \cdot 2^j) \quad (7)$$

Substituting (7) in (3), we get

$$0 \leq \{(a_j + b_j - c_{j-1}) \cdot 2^j - X_j\} \leq (2^j - 1) \quad (8)$$

Now, let us consider the two possible values of $f_1(a_j + b_j - c_{j-1})$, i.e.

Case (i) $f_1(a_j + b_j - c_{j-1}) = 1$ :-

From definition (5), we get

$$(a_j + b_j - c_{j-1}) \geq 1 \quad (9)$$

(9) and upper inequality of (8) give

$$X_j \geq 1 \quad (10)$$

Case (ii) $f_1(a_j + b_j - c_{j-1}) = 0$ :-

Again from definition (5), we get

$$(a_j + b_j - c_{j-1}) < 1 \quad (11)$$

(11) and lower inequality of (8) give

$$X_j \leq 0 \quad (12)$$

(10) and (12) indicate that if we choose $\theta$ such that $0 < \theta < 1$, then $f_1(a_j + b_j - c_{j-1})$ and $f_\theta(X_j)$ are equivalent. This completes the proof for (a).

(b): In order to check linear separability of $s_j$, we construct a binary cube with $c_{j-1}$, $c_j$, and $f_1(a_j + b_j - c_{j-1})$ as the three axes and $s_j$ as the output variable, as it was done for A1. If don’t care combinations are chosen appropriately, then $1^a$ and $0^a$ of $s_j$ are linearly separable. One such separating plane is

$$c_{j-1} + f_1(a_j + b_j - c_{j-1}) - c_j = 0.5 \quad (13)$$

From (a) we can rewrite (13) as

$$c_{j-1} + f_1(X_j) - c_j = 0.5 \quad (14)$$

Above procedure is illustrated for two-bit addition in Figure 3. It follows that only $2n$ neurons are needed in the first layer for $n$-bit addition.
2.3 architecture 3 (A3)

In previous architectures, inputs were fed to the first layer only. Now question arises that if inputs are fed to both first and second layers, can a less complex network be obtained?. The answer is in affirmative as it follows from the following equation obtained using Table 2

\[ s_j + 2c_j = a_j + b_j + c_{j-1} \]  

(15)

where, \( c_{j-1} \) and \( c_j \) can be obtained using one neuron for each of them as in previous architectures. Moreover, \( s_j \) can be obtained just by proper summation of \( a_j, b_j, c_{j-1} \) and \( c_j \), thus obviating the need of thresholding in the second layer. Hence, \( n \) hidden neurons are enough to perform \( n \)-bit addition. The architecture has been illustrated for two-bit addition in Figure 4.

3 Extension of the proposed Architectures to other arithmetic functions

Architectures A1 and A3 proposed in §1 can be generalized to perform other arithmetic functions namely two’s complement, binary subtraction, addition of \( k \) \( n \)-bit numbers and multiplication. Extension for two’s complement and subtraction is quite obvious, only point worth mentioning is that architectures for two’s complement do not require exponential weights. Now, extension of A1 to addition of \( k \) \( n \)-bit numbers requires \((2k - 1)n\) neurons in the first layer, while extension of A3 requires \((k - 1)n\) neurons. Both the extensions require \((n + \log_2(k - 1))\) neurons in the second layer.

Multiplication of two \( n \)-bit numbers can be carried out by generating \( n^2 \) product terms in the first layer, and summing up appropriate product terms in the next two
layers to obtain the output bits. Hence, extension of $A_1$ requires $n^2,$ $(2n^2 - n - 2)$ and $(2n - 1)$ neurons in the first, second and third layer respectively. While extension of $A_3$ requires $n^2,$ $(n^2 - n - 1)$ and $(2n - 1)$ neurons respectively in the first, second and third layer respectively. Details of the extensions can be found in [4].

4 On optimality of $A_2$ and $A_3$

Architectures $A_2$ and $A_3$ appear to be optimal in the number of neurons used, for when inputs are fed to the first layer only and otherwise respectively. This conjecture is based on the fact that $n$ independent EXclusive ORs require same number of neurons as required in the proposed architectures for the respective network configurations. Moreover, above conjecture is supported by an exhaustive proof for two-bit addition. The procedure taken for the same is as follows:

- **STEP 1:** To generate all linearly separable (l.s.) functions of four binary variables. Now all l.s. functions of $n$ input variables are determined by computing a set of $(n + 1)$ Chow parameters for every Boolean function of $n$ variables. If the parameter set of a function matches with a set among those listed for linearly separable functions, then the function is linearly separable [5]. For 4 variables, there are 1882 l.s. functions. However, only first 941 l.s. functions need be considered as the rest can be obtained as complements.

- **STEP 2:**
  
  (i) **Architecture 2:** $A_2$ would not be optimal for two bit addition if there exists a three hidden neuron architecture for the same. To prove that is not possible, we can consider all possible combinations of three l.s functions and can show that none
of these combinations can linearly separate all the bits of the sum. This can be
achieved by transforming present problem to that of Linear Programming for each
of the sum bits $s_i$ (for $i = 0, 1, 2$). Thus, a set of inequalities is obtained for each
$s_i$ corresponding to different possible values of the input vector. Inequality for the
present input vector is checked for consistency with the inequalities for previous
input vectors. This is essential as three $l.s.$ functions will give rise to only eight
possible values for first layer output vector as against sixteen possible input vectors.

Inequality for an input vector is formulated so that if for this input vector $s_i$ is
one, then the weighted sum of the three $l.s.$ functions considered (weights are to
be determined) should be greater than or equal to some variable say $u$; but if $s_i$
is zero, then the weighted sum should be less than or equal to say $l$. $u$ and $l$ give
respectively the upper and lower bounds on the value of threshold of the threshold
function which separates $s_i$. Now, using linear programming $(u - l)$ is maximized.
If the value obtained is nonzero positive, then $s_i$ is linearly separable from this
combination of three functions. This procedure is repeated for other values of $i$
and for all possible combinations of three $l.s.$ functions.

Computer simulation of above showed that no combination of three $l.s.$ functions
can separate all the bits of the sum. Hence, $A2$ is optimal for two-bit addition.

(ii) Architecture $3$: The procedure taken is same as that for $A2$ except that one $l.s.$
function is considered along with the four inputs for separability of $s_i$. Simulation
showed that only $l.s.$ functions which can separate $s_o$ are $a_0 \cdot \overline{b_0}, a_0 \cdot \overline{b_0}, a_0 \cdot b_0$ and

\[ a_0 \cdot \overline{b_0}, \text{ and their complements. Now, clearly none of these functions can separate} \]
\[ s_1. \text{ Hence, } A3 \text{ is optimal for two-bit addition.} \]

Above simulation and our experience with various methods to prove optimality of $A3$
[4] has prompted us to put forward the following conjecture.

**Conjecture 1** A non-threshold function of $n$ variables $x_1, x_2, \ldots, x_n$ cannot be expressed
as a threshold function of $f_i$ and $x_1, x_2, \ldots, x_n, x_{n+1}$, where $f_i$ is a threshold function and
has at least one prime implicant dependant upon $x_{n+1}$.

From above simulation it follows that Conjecture 1 holds for $n = 2$. Moreover, based
on this conjecture and mathematical induction, we can argue that $A3$ is optimal for any
$k$-bit addition. Now $A3$ is optimal for $k = 2$. Hence, if given that $A3$ is optimal for $k = n$, then we argue that in an optimal realization of $(n + 1)$-bit addition, block used for $n$-bit addition need not be modified. This follows from Conjecture 1 as $\{s_0, s_1, \ldots, s_{n-1}\}$ are independent of $(n + 1)^{th}$ bit of the two numbers. Now since $s_n$ is not linearly separable using $\{a_0, a_1, \ldots, a_n, b_0, \ldots, b_n\}$, therefore at least one additional neuron is needed in the hidden layer to separate it. But that is the case with $A3$, hence $A3$ should be optimal for all $n$.

5 Conclusions

We have proposed three architectures of two layers each to perform binary addition. Two
of these architectures have been shown to be optimal in the number of neurons required for
two-bit addition. It has been conjectured that they are optimal for any bit size addition. Moreover, proposed architectures have been extended to perform addition of $k$ $n$-bit numbers in two layers requiring $O(k \cdot n)$ neurons as against a bound of $O(k^{1+\epsilon} \cdot n)$ neurons with depth $O(\frac{1}{\epsilon})$ (for $0 < \epsilon \leq 1$) suggested in [6]. Also, extension to multiplication of two $n$-bit numbers requires $O(n^2)$ neurons with a depth of three as against a depth four network in [1] or $O(\frac{1}{\epsilon} \cdot n^{1+\epsilon})$ neurons with a depth of $O(\frac{1}{\epsilon})$ (for $0 < \epsilon \leq 0.6$) in [6].

The problem of immediate concern is to obtain proofs for the optimality of the proposed architectures and Conjecture 1. Next step of our study would be to obtain a polynomial size constant depth architecture for division problem and to improve upon the architectures proposed for binary multiplication. A major drawback of our architectures is the requirement of exponential weights which limits the VLSI implementability to lower values of $n$. Hence, another problem to work on is to overcome above limitation with no or little additional requirement on the number of neurons used.

References


