Journal of Engineering Sciences
Vol11,Issue2,FEB/2020
ISSN NO:0377-9254

MITCHELL'S APPROXIMATE MULTIPLICATION FOR CONVOLUTION NEURAL NETWORKS FOR LOW POWER APPLICATIONS

#1N.SOUMYA, M.Tech Student,
#2Dr.Agarala Subba Rami Reddy, Professor,
Dept of ECE,
JYOTHISHMATHI INSTITUTE OF TECHNOLOGY AND SCIENCE, KARIMNAGAR, TELANGANA.

Abstract:

Recent studies have demonstrated the potential for achieving higher area and power saving with approximate computation in error tolerant applications involving signal and image processing. Multiplication is a major mathematical operation in these applications which when performed in logarithmic number system results in faster and energy efficient design. The paper presents a new multiplier enabling achievement of an arbitrary accuracy. It follows the same idea of number representation as the Mitchell's algorithm, but does not use logarithm approximation. The proposed iterative algorithm is simple and efficient and its error percentage is as small as required. As its hardware solution involves adders and shifters, it is not gate and power consuming. Parallel circuits are used for error correction. The error summary for operands ranging from 8-bit to 16-bit operands indicates a very low error percentage with only two parallel correction circuits. The logarithmic of a binary number may be determined approximately from the number itself by simple shifting and counting. Since the logarithms used are approximate there can be errors in the result.

1. Introduction

Signal processing applications use tasks such as Finite Impulse Response filtering, Fast Fourier Transform, and Discrete Cosine Transform, which involve heavy use of arithmetic operations such as addition, multiplication, and division. The arithmetic operations, such as multiplication and division in a binary number system, are computationally complex in terms of area, delay, and power. While the fixed point number system could have the problem of overflow and scaling due to limited precision , the floating-point number system provides better precision and scaling, however, incurring significantly more overhead. Logarithmic Number Systems (LNS) offer a viable alternative combining the simplicity of a fixed point number system and the precision of a floating-point number system. In LNS-based systems, it is possible to implement operations like multiplication and division using addition and subtraction operations on the logarithms of the input data. However, the computations in the LNS domain result in some loss of accuracy and, thus, are mostly limited to those signal processing applications in which a certain amount of error is tolerable. Consequently, an exact result after multiplication is not always required and a rounded product is acceptable in many cases. Hence, LNS arithmetic is preferred in many signal processing systems due to its area, delay, and power advantages in spite of some loss of accuracy. Swartzlander was the first to develop a four function (addition, subtraction, multiplication, and division) arithmetic processor based on the logarithmic number system. Subsequently, the use of LNS has been demonstrated further in other works . Logarithms are usually computed using the Taylor's series or by storing a complete table of logarithms for all the numbers.

These methods are inefficient in terms of the amount of hardware required as well as speed and power. Several approaches have been proposed in the literature for improving the efficiency of logarithm based arithmetic. They can be broadly classified as 1) Look Up Table (LUT)-based interpolation , and 2) Mitchell's algorithm-based logarithm computation . Earlier studies have shown that interpolation-based

methods can implement logarithms with high precision but require a considerable amount of computation and storage . On the other hand, Mitchell's algorithm does not require any ROM and targets efficiency at the loss of some accuracy. Thus, there is a strong interest toward using Mitchell's algorithm to implement logarithmic computations for use in DSP applications. Mitchell's algorithm for multiplying two numbers using logarithms is straightforward. The logarithms of the input numbers are added and the antilogarithm of the sum is determined. The method used to find the logarithm and the antilogarithm impacts the accuracy. Mitchell presented a simple method to approximate the logarithm and antilogarithm calculations using piecewise straight line approximations of the logarithm and the antilogarithm curves. The logarithm error due to Mitchell's approximation is in the range 0 Error 0 : 08639 and attains the maximum value when the fraction part of the logarithm is equal to 0.44 . The maximum possible error for Mitchell's logarithm multiplication is around 11.1 percent and the average error is around 3.8 percent. Mitchell's method has a high amount of error because of the single straight line approximation.

In the same paper, Mitchell discussed the derivation of an analytical correction term that can be added to the final result to reduce the error. However, the computation of the correction term itself required the use of Mitchell's algorithm. Later, several methods based on divided straight line approximation were proposed in the literature for improving the accuracy of the result from Mitchell's algorithm. In, a simulation-based correction strategy was developed for use with Mitchell's algorithm. The method uses a fixed table of correction values for the different fractional regions of the logarithms. In this work, we propose a new method based on operand decomposition (OD) for improving the accuracy of Mitchell's logarithmic multiplication. The OD technique was first proposed in for reducing the amount of switching activity in binary multiplication. The operand decomposition reduces the number of "1" bits in the decomposed operands which, in turn, decreases the chances of a carryover from the mantissa part to the integer part during the logarithm summation step.

It is shown mathematically in that the accuracy of Mitchell's algorithm can be improved by avoiding the carryover during the summation step. We show that applying operand decomposition to the inputs as a preprocessing step to Mitchell's algorithm significantly improves the accuracy. Further, since the OD method does not require the addition of a correction term, we show that it can be easily combined with other methods to obtain even better accuracy. The remainder of the paper is as follows. LNS-based multiplication methods are of two categories: 1) methods that use lookup tables and interpolation and 2) methods based on Mitchell's algorithm that computes the logarithms through shift and count operations. While the lookup table-based methods involve memory overhead, Mitchell's algorithm does not require memory; however, it incurs some loss of accuracy. Thus, several researchers have proposed methods to improve the accuracy in Mitchell's algorithm. The various related works in logarithm-based multiplication are categorized in the taxonomy diagram given in Fig. 2. These methods are briefly explained in the rest of the section. The traditional approach to logarithmic multiplication is to use complete tables of logarithm values stored in memory, which is inefficient in terms of memory requirements. Thus, several researchers investigated the idea of using smaller lookup tables and compensating with interpolation methods . The objective was to identify LUT sizes corresponding to varying logarithmic accuracy levels, trading off accuracy versus memory overhead. In the multidimensional logarithmic number system (MDLNS) is investigated in which the memory requirements are reduced at the cost of additional computations.

In spite of the attempts at reducing the amount of memory to store lookup tables, there is strong interest in Mitchell's algorithm, which completely eliminates the use of lookup tables. In this section, we briefly describe Mitchell's algorithm for logarithm-based multiplication. Mitchell's logarithm and antilogarithm calculations require only shifting and counting operations. The zero detector is to ensure a zero output if any of the inputs is zero. Mitchell's algorithm for multiplying two inputs A and B is given in Table 1. The position of the leading 1 bit in each input is identified by shifting the input bits left

until the most significant bit is a "1," decrementing a counter each time, which is initially loaded with the word size. The final values of these counters, k A and k B , form the integer part of the logarithms of A and B. The remaining sequences of bits form the corresponding mantissa parts, m A and m B . The corresponding integer and mantissa parts can be appended to form the logarithm values, f A and f B , of the given inputs A and B. These logarithm values are added and the antilogarithm of the sum, f AB , is determined as follows: The value of the characteristic part of the sum, k AB , is decoded to determine the position in the final product, where a leading 1-bit is inserted.

The mantissa part, m AB , of the sum is simply appended immediately to the right of the inserted leading 1-bit padded by 0-bits in the remaining positions to form the final result, which is the product A ? B . A step-by-step example illustrating the above procedure for using the logarithmic multiplication of decimal numbers 18 and 58 It can be observed that Mitchell's logarithm multiplication algorithm is simple to perform and requires only shifting and counting operations. The example shown has an error of 5 percent. Mitchell shows in that the error in the multiplication is due to the fraction part of the logarithms and provides a detailed error analysis.

2. Mitchell's Logarithmic Multiplication

One of the most significant multiplication methods in logarithm number system is Mitchell's algorithm. In MA two operands are multiplied by finding their logarithm, adding them and taking the antilogarithm of the sum.

$$N = \sum_{i=j}^{k} 2^i * z_i \qquad \text{.....(1)}$$

Where z_i =0,1. Since z_k is the most significant bit.

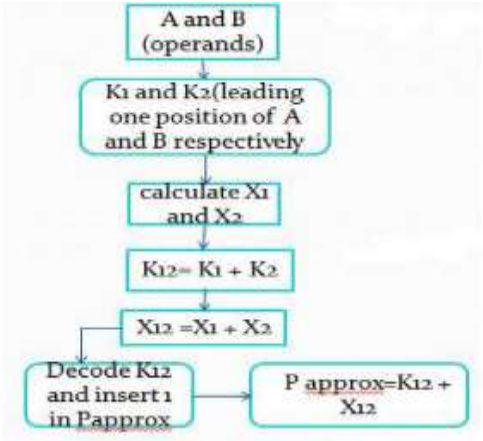

Figure 1 : Mitchell's based algorithm

Approximation of the logarithm and antilogarithm is derived from binary representation of numbers.

$$N = 2^k\left(1+\sum_{i=j}^{k-1} 2^{i-k} * z_i\right)$$
$$N = 2^k(1+x) \qquad \text{.....(2)}$$

Where k is the characteristic number or place of the most significant bit with the value of 1,Zi is the bit value at ith position is the fraction depends on the number precision. MA produces an significant error .this error increases with the number of bits with the value of 1' in the mantissa.

3. Proposed multiplier

The logarithmic multiplier was presented by mitchelle . In general it has two operands A and B converting them to logarithmic number system (LNS) and computing the final result Z.

$$Z = \sum_{i=0}^{n-1} 2^i z_i = 2^k\left(1+\sum_{i=j}^{k-1} 2^{i-k} z_i\right), \qquad k \geq j \geq 0. \qquad \text{.....(3)}$$

Where k is the position corresponding to the lead one , Z_i is a value of i^{th} position and j depends on precision .

$$\log_2(Z) = \log_2\left(2^k\left(1+\sum_{i=j}^{k-1} 2^{i-k} z_i\right)\right)$$
$$= \log_2(2^k(1+x))$$
$$= k + \log_2(1+x). \qquad \text{.....(4)}$$

3.1 algorithm

A, *B*: *n*-bits, *P*: *2n*-bits approximate product
1. $k_A = \text{LOD}(A)$, $k_B = \text{LOD}(B)$
2. $x_A = A << (n - k_A - 1)$, $x_B = B << (n - k_B - 1)$
3. $L = (\text{`0'} \,\&\, k_A \,\&\, x_B[n\text{-}2..0]) + (\text{`0'} \,\&\, k_B \,\&\, x_B[n\text{-}2..0])$
4. $charac = L[n+log(n)\text{-}1..n\text{-}1]$, $mant = L[n\text{-}2..0]$, $s = charac[log(n)]$
5. IF s == '1' THEN // Large characteristic
   $D = (\text{`1'} \,\&\, mant) << ((\text{`0'} \,\&\, charac[log(n)\text{-}1..0]) + 1)$
   ELSE // Small characteristic
   $D = (\text{`1'} \,\&\, mant) >> (\sim charac[log(n)\text{-}1..0])$
6. IF *Is-Zero(A,B)* THEN // *A* or *B* are zero
   $P = 0$
   ELSE
   $P = D$

Example 1 (Mitchell's multiplication of 234 and 198).

N1 =234 =11101010; N2 = 198 =11000110

k1 = 0111; x1 = 11010100k2 =0111; x2 =10001100

k1 + k2 = 1110

x1 + x2 =101100000 P 28 k12 = 1110 + 1 = 1111

PMA = 101100000000000 = 45056
Ptrue = 46332

The proposed solution simplifies logarithm approximation introduced in (5) and introduces an iterative algorithm with various possibilities for achieving the multiplication error as small as required and the possibility of achieving the exact result.

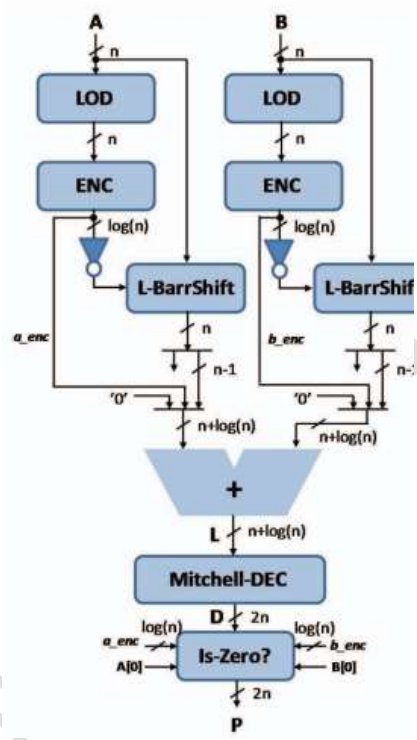

Figure 2: Mitchell block diagram

By simplifying the logarithm approximation introduced in (5), the correction terms could be calculated almost immediately after thecalculation of the approximate product has been started. In such a way, the high level of parallelism can be achieved by the principle of pipelining, thus reducing the complexity of the logic required by (5) and increasing the speed of the multiplier with error correction circuits

4. Conclusion

We have presented optimized implementation of the logarithmic multiplication algorithm and demonstrated that it can reduce significant amount of power for CNN's. Our design has significant power reduction while compared to conventional logarithmic multiplication

References

[1] Z. Babic', A. Avramovic', P. Bulic' (2010) An iterative logarithmic multiplier.

[2] DurgeshNandan, JitendraKanungo, Anurag Mahajan (2017) An efficient VLSI architecture for Iterative Logarithmic Multiplier.

[3] John n. Mitchell, jr. t associate, ire (1962) Computer Multiplication and Division Using Binary Logarithms.

[4] UrosLoric, PatricioBulic (2012) Applicability of approximate multipliers in hardware neural networks.

[5] ZdenkaBabic, AleksejAvramovic, PatricioBulic (2008) An iterative Mitchell's Algorithm Based Multiplier.