B. M. Wilamowski, J. J. Cupal, R. S. Sandige, "A Fractional Powers-of-Two Number System for Digital Neural Networks", proceedings of the IASTED International Conference on Modeling and Simulation, Pittsburgh, Pennsylvania, pp. 620-622, May 10-12, 1993.

A Fractional Powers-of-Two Number System for Digital Neural Networks

Bogdan M. Wilamowski, Jerry J. Cupal, and Richard S. Sandige
University of Wyoming
Department of Electrical Engineering
Laramie, WY 82071

Abstract

A new arithmetic number system based on fractional powers of two is proposed. In this system, the weight of each bit position is a power of $2^{1/n}$, where n=2,3,4 etc. Multiplication or division by $2^{1/n}$ can be accomplished by simple shifts of the input data. Although not as simple as in conventional twos complement binary, addition and substraction can be accomplished using simple ALU structures. Details of how this is accomplished are given, as well as an implementation of a hardware structure to perform multiply-accumulate operations commonly found in neural nets and FIR filters.

Key words: Neural Networks, Hardware, Arithmetics

Introduction

Many recent achievements have been recorded in the relatively new area of research in neural networks. New learning algorithms, new system structures and many useful applications have been found. Neural network hardware implementations are also of current focus, but this area of research is not as advanced. This is obvious that better hardware implementation could enhance neural network development. A neuron must perform a series of multiply-accumulate operations. This can be done in analog hardware or in digital computer systems. For many applications, the input signals are in digital format and the analog technique of implementing a neural network is prohibitive. Therefore, an all-digital approach must be taken [1][2][3]. Because of the need to perform the multiply operation, neural networks implemented in a digital computer operate relatively slowly. The signal at each neural connection has to be multiplied by an appropriate weight. One way to overcame this problem is to use logarithmic arithmetic [4][5]. In this case multiplications are fast (amounting to simple addition) but the addition and subtraction process creates a problem. Another approach is to use the shift operation instead of multiplication [6]. In the case of binary systems, such multiplication is coarse. Multiplicands with a value of powers of two are only possible; i.e. 2, 4, 8, 16 ... or 0.5, 0.25, 0.125 ... The purpose of this paper is to introduce a new number system to perform the arithmetic operations required in sum-of-products algorithms. This system seems ideally suited for applications where fast, but not necessarily accurate, multiplications are required, as is the case for a neuron in a neural network. Instead of binary numbers in base 2 numbers in base $2^{1/n}$ are introduced. Using this approach fast multiplications and division are possible by simple shifting of the input data.

Concept of the arithmetic

Consider a neuron that performs arithmetic using a different number system. This number system is to be based upon weights of $2^{1/n}$ (where n = 2,3,4, etc.) instead of 2^1. The use of such a number system has lead to the design of simple neurons with interesting features. For example, if n=2, this new number system is based upon powers of $\sqrt{2}$. That is, the least significant bit has a value of 1, the second $\sqrt{2}$, etc. In this case, the representation for the number 7.4142_{10} = $\sqrt{2}$ + 6 or $0110_{\sqrt{2}} + 0001_{\sqrt{2}}$ is as shown in Figure 1.

8√2	8	4√2	4	2√2	2	√2	1	==	value
0	0	0	1	0	1	1	0	==	√2+6=7.4142

Figure 1. The representation of 7.4142_{10} = $\sqrt{2}$ + 6 in base $\sqrt{2}$ number system.

In this numbering system, multiplication or division by powers of $\sqrt{2}$ corresponds to the shift left or shift right operation by one position as shown in Figure 2. Multiplication by $(\sqrt{2})^3 = 2\sqrt{2}=2.8284$ of the number 7.4142 corresponds to the shift left by three places.

8√2	8	4√2	4	2√2	2	√2	1	==	value
0	0	1	0	1	1	0	0	==	6√2+2=10.4853
0	0	0	0	1	0	1	1	==	3√2+1=5.2426

Figure 2. Multiplication and division by $\sqrt{2}$ as done with shifts in base $\sqrt{2}$ number system. The original value of the number was 7.4142_{10} = $\sqrt{2}$ + 6, as in Figure 1.

Addition (and substraction) of two numbers in base $\sqrt{2}$ is similar to binary addition (and substraction), with the difference that the carry coming out of the addition of any two bits is added in two bit positions to the left as shown in Figure 3. If the base is $2^{1/n}$, the carry is shifted by n.

	8√2	8	4√2	4	2√2	2	√2	1	value
A	0	1	1	0	1	0	1	1	7√2+9
B	0	0	0	0	0	1	1	1	√2+3
carry	1	-	1	1	1	1	-	-	
A+B	1	1	0	1	0	0	0	0	8√2+12

Figure 3. Addition of two numbers in base √2 number system

Because of this somewhat unique addition feature, it is possible to separate a number in the √2 number system into two halves, one representing the square root terms and the other representing the integer terms. Addition (or substraction) can be done on each half in parallel, making this a faster process than adding the whole value, as in conventual binary adders. Figure 4 shows an example of the addition operation.

A		B		A + B
0 1 1 1 1	+	0 0 0 1	=	1 0 0 0
1 0 0 1		0 0 1 1		1 1 1 0 0
7√2 + 9 = 18.8995		3 + √2 = 4.4142		8√2 + 12 = 23.3137

Figure 4. Addition of two numbers in base √2 number system. The top row represents the fractional terms, the bottom the integer terms. Addition is accomplished by adding the separate representations.

In the of example of Figure 5 showing the subtraction of 00000111 (4.4242) from 01101011 (18.8995), each 8 bit number can be split into two 4 bit components. Substraction is accomplished by taking the twos complement of both components of the subtrahend and adding these to the corresponding components of the minuend. Note that the number 00000111 has two components: 0001 and 0011 with the respective two's complements 1111 and 1101. Therefore the two's complement in base √2 arithmetic is 11111011, which is differ from the two's complement in binary logic (11111001).

A		B		A + B
0 1 1 1 1	-	0 0 0 1	=	
1 0 0 1		0 0 1 1		
corresponds to				
0 1 1 1 1	+	1 1 1 1 1	=	0 1 1 1 0
1 0 0 1		1 1 1 0 1		0 1 1 1 0
7√2 + 9 = 18.8995		- √2 - 3 = - 4.4142		6√2 + 6 = 14.4853

Figure 5. Subtraction of two numbers in base √2 number system

Because multiplication or division is done by shifts, the weights of a neuron could be stored as some number that represents the number of left or right shifts. This would reduce the memory requirements to store the weights of any given structure. This, plus the fact that fast multiplication can be performed by simple shifts of the input values, make this new arithmetic an interesting technique for applications in neural networks. The examples show arithmetics based upon $2^{1/n}$ with n = 2. By using larger values of n, the "coarseness" of the weights can be reduced from the examples given.

Implementation of the arithmetic

With the proposed arithmetic, multiplication or division by a constant can be replaced with shift left or shift right operations. Likewise, when arithmetics base √2 are used, addition and subtraction operations can be performed in a similar way as in binary arithmetic. The only difference is the necessity of independent operations on two components of half length. These computations are best performed in parallel using two separate ALU units. Thus arithmetics in base $2^{1/n}$ require n parallel ALU units.

There are several hardware configurations that could implement the $2^{1/n}$ arithmetic discussed above. It is obvious that one shift register and at least two simple ALUs that perform addition or substraction (using twos complement) are required for the √2 arithmetic shown in the examples. One such hardware model is shown in figure 6. It is intended to perform a multiply-accumulate operation such as would be required in a neural net or a FIR filter. The input values are twos complement binary numbers, in this case 16 bits wide. The weights can be represented in a 6 bit number, in a form of sign-magnitude format. Actually, one bit is for the sign, another for the shift direction (multiplication or division), and 4 bits for the shift count.

The input value is loaded into a 16 bit barrel shifter with the shift direction controlled by the weight's direction bit and the shift amount controlled by the most significant 3 bits of the shift count. Shifting is done as in normal twos complement data shifts-- left shifts fill with zeros and right shifts fill by sign extension. The result of this shifting process multiplies the input value by the unsigned fixed weight.

The output of the shift register is feed into one of two add/accumulate structures by the action of two 16 bit multiplexers controlled by the least significant bit of the shift count. If this bit is a one, indicating multiplication/division by √2, then the value is added to the accumulator holding the √2 terms (represented as result_sq_rt in Figure 6). If this bit is a zero, indicating multiplication/division by integer values, then the value is added to the result_int accumulator. Actually, the ALUs in the figure perform either addition or twos complement substraction, as controlled by the sign of the weight.

The hardware controller continues the multiply-accumulate operations until all the input values are processed. At the conclusion, the result accumulators hold the √2 and integer sums. Since it is desirable to have standard binary outputs, it is necessary to multiply the value in the result_sq_rt accumulator by √2 and add these to the integer accumulator. This is done by passing the value in the result_sq_rt accumulator back through the multiply-accumulate structure with appropriate weights to approximate multiplication by √2 (√2 = 1 + 1/4 +1/8 + 1/32).

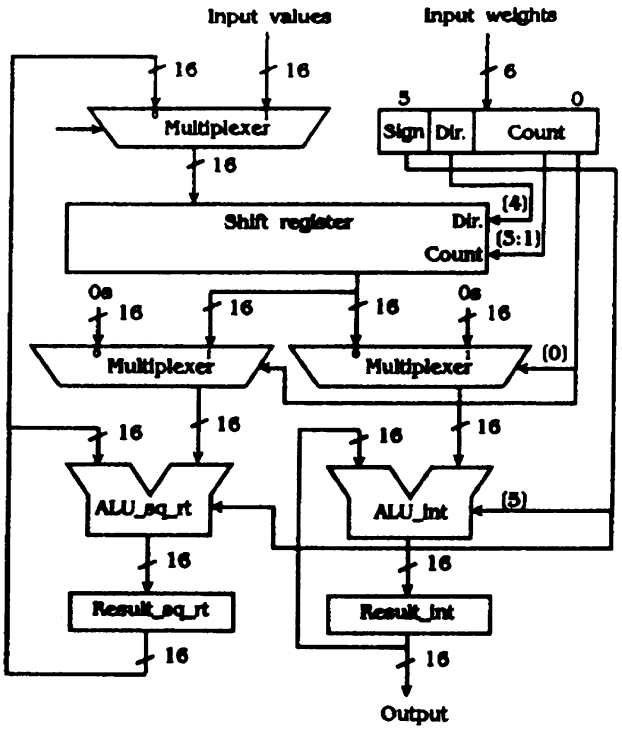

Figure 6. A hardware model of an arithmetic logic circuit to perform multiply-accumulate algorithms in the √2 number system.

The hardware structure shown in Figure 6 has been implemented in Verilog HDL. A 20 tap FIR filter structure was demonstrated using all fractional coefficients. That is, the coefficients were not formed into integers as is often the case when using integer multiplications within a microprocessor. Since the coefficients are always fractional, the direction bit in each weight is not required. There was an average error of about 10% in the actual value of the coefficient compared to the desired. This is approximately half that of a filter based upon shifted binary weights (ref 6). Further reduction in the coefficient error could be obtained with structures based upon $2^{1/n}$, but these would require additional multiplexer/ALU/accumulator structures.

Parallel processing which is natural for addition or subtraction operations also can be extended for an entire layer. The nature of a neural network is such that all signals in one neural layer can be processed at the same time, thus allowing for parallel computation of all neurons within a layer.

Conclusion

The proposed arithmetic has several advantages: (1) Multiple ALU units provide parallel processing, resulting in fast operation. (2) Since multiplication is done by data shifting, a serial communication technique may be used between neurons to further enhance multiplication speed and reduce number of required interconnections. (3) For most practical neuron applications, only four to six bits of memory are required to store each weight value.

The disadvantages of the proposed arithmetics are that in most cases longer registers would be required since numbers in base $2^{1/n}$ are longer. Through the introduction base $2^{1/n}$ arithmetic the coarsity of weights can be significantly reduced in comparison to normal base 2 arithmetic in which multiplication by the power 2 is substituted by shift operations.

Reference

1 *A. F. Murray, A. W. Smith, "Asynchronous VLSI Neural Networks Using Pulse-Stream Arithmetic", IEEE Journal of Solid-State Circuits, Vol. 23, No. 3, June 1988, pp. 688-697.*

2. *M. Mumford, D. Andes, L. Kern, "The Mod 2 Meurocomputer System Design," IEEE Trans. on Neural Networks, Vol. 3, No. 3, May 1992, pp. 423-430.*

3. *G. Pechanek, S. Vassiliadis, J. Delgago-Frias, "Digital Neural Emulators Using Tree Accumulation and Communication Structures," IEEE Trans. on Neural Networks, Vol. 3, No. 6, November 1992, pp. 934-940*

4. *M. Arnold, T. Bailey, J. Cowles, J. Cupal, "Implementing Back Propagation Neural Nets with Logarithmic Arithmetic", Proceedings Intern. AMSE Conference on Neural Networks, San Diego, May 29-31, 1991, Vol. 1, pp. 75-86*

5. *M. Arnold, T. Bailey, J. Cowles, J. Cupal, "Redundant Logarithmic Arithmetic", IEEE Trans. on Computers, Vol. 39, No. 8. August 1990, pp. 1077-1086.*

6. *M. Marchesi, G. Orlandi, F. Piazza, A. Uncini, "Fast Neural Networks Without Multipliers". IEEE Trans. on Neural Networks, Vol. 4, No. 1, January 1993, pp. 53-62*