Neural Network Binary Multiplier

Han Huang, Fangyue Chen and Luxia Xu
School of Science, Hangzhou Dianzi University, Hangzhou, Zhejiang, P. R. China

Abstract—*Almost all signal processing applications demand a considerable number of multiplications. As addition, multiplication is also the speed-limiting element in the core of arithmetic logic unit (ALU) in a microprocessor. In this paper, the perceptron of artificial neural network (ANN) is used to build a binary multiplier, named neural network binary multiplier(NNBM). It is shown that many advantages of ANN such as synchronous, parallel and fast speed on processing information are taken by the multiplier.*

Keywords: Arithmetic logic unit, multiplication, Perceptron, neural network binary multiplier

1. Introduction

As addition, multiplication is also an essential arithmetic operation for common DSP applications [1–4]. Almost all signal processing applications demand a considerable number of multiplications, and every microprocessor design company always wants a fast multiplier design that will allow their processor to be the fastest on the market. The only problem with this is the fact that the faster the multiplier, the smaller the chip area becomes for other important parts of the microprocessors [5,6].

Usually, a binary multiplier is an electronic circuit used in digital electronics to multiply two binary numbers, for example, a circuit logic diagram of a 2-bit by 2-bit binary multiplier is shown in **Fig. 1**.

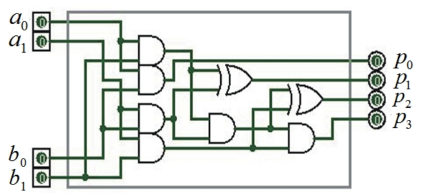

Fig. 1: Logic diagram of a 2-bit by 2-bit binary multiplier.

In order to allow for a faster design that takes up less space, one must find an innovative way to design the circuitry. Fortunately, perceptron of feedforward neural networks, which was inspired by the threshold logic unit neuron model of McCulloch and Pitts, can be used to design a binary multiplier [7,8]. However, it can be very difficult for a designer to find the best design when the number of inputs of the perceptron is a large number [5,6].

This paper proposes a design of neural network binary multiplier (NNBM) under the framework of multi-layer perceptron (MLP) of binary feedforward neural networks (BFNNs). The DNA-like learning algorithm proposed by the present authors is successfully used for training the weight-threshold values of NNBM [9-11].

2. Design of NNBM

2.1 Partial products of multiplication

For two integers n and m ($n \geq m$), let n-bit unsigned binary number $A = (a_{n-1}, a_{n-2}, \cdots, a_0)$ is multiplicand, and m-bit unsigned binary number $B = (b_{m-1}, b_{m-2}, \cdots, b_0)$ is multiplier, then $P = A \cdot B = (p_{n+m-1}, p_{n+m-2}, \cdots, p_0)$ is the $(n+m)$-bit binary product, where

$$\begin{cases} p_i = \text{mod}(\sum_{j+k=i} a_j b_k + c_i, 2) \\ c_{i+1} = [\sum_{j+k=i} a_j b_k / 2] \\ \quad (i = 0, 1, \cdots, m+n-2; \\ \quad j = 0, 1, \cdots, n-1; \\ \quad k = 0, 1, \cdots, m-1) \\ p_{m+n-1} = c_{m+n-1} \end{cases} \tag{1}$$

where $c_0 = 0$ is the initial carry, and $[\,x\,]$ represents the first integer larger than x. The multiplication process is shown in **Fig. 2**.

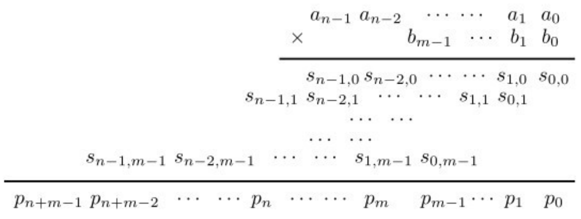
$a_{n-1}\ a_{n-2}\ \cdots\cdots\ a_1\ a_0$
$\times\ b_{m-1}\ \cdots\ b_1\ b_0$
$s_{n-1,0}\ s_{n-2,0}\ \cdots\cdots\ s_{1,0}\ s_{0,0}$
$s_{n-1,1}\ s_{n-2,1}\ \cdots\cdots\ s_{1,1}\ s_{0,1}$
$s_{n-1,m-1}\ s_{n-2,m-1}\ \cdots\cdots\ s_{1,m-1}\ s_{0,m-1}$
$p_{n+m-1}\ p_{n+m-2}\ \cdots\cdots\ p_n\ \cdots\cdots\ p_m\ p_{m-1}\cdots\ p_1\ p_0$

Fig. 2: Multiplication process

Here, $s_{j,k} = a_j b_k (j = 0, 1, \cdots, n-1;\ k = 0, 1, \cdots,\ m-1)$ are mn partial products in the multiplication process. In fact, in order to obtain the products, each row of the process should be extended into $(m+n)$-bit binary number, and the supplementary numbers are zero. For example, the k-th ($k = 0, 1, \cdots, m-1$) row of the partial products is,

$$s_{j,k} = \begin{cases} a_j b_k & \text{if } 0 \leq j \leq n-1\,, \\ 0 & \text{if } -k \leq j \leq -1 \\ & \text{and } n \leq j \leq m+n-k-1 \end{cases} \tag{2}$$

A network with inputs $A=(a_{n-1},a_{n-2},\cdots,a_0)$ and $B=(b_{m-1},b_{m-2},\cdots,b_0)$, and produces outputs $P=A\cdot B=(p_{n+m-1},p_{n+m-2},\cdots,p_0)$ is named binary multiplier. If the network is the neural network, then the corresponding multiplier ia called neural network binary multiplier (NNBM).

2.2 Truth table of NNBM

Based on the multiplication process and the partial products of the binary multiplication, the array representation of the truth table of NNBM can be easily obtained as shown in Table 1.

Table 1: Truth table of NNBM

$(a_{n-1},\cdots,a_0)$	$(b_{m-1},\cdots,b_0)$	$(p_{n+m-1},\cdots,p_0)$
$A^{(0)}$	B	$P^{(0)}$
$A^{(1)}$	B	$P^{(1)}$
$\vdots$	$\vdots$	$\vdots$
$A^{(2^n-2)}$	B	$P^{(2^n-2)}$
$A^{(2^n-1)}$	B	$P^{(2^n-1)}$

In Table 1, B is a $(2^m\times m)$-order array:

$$B=\begin{pmatrix}0\ 0 & \cdots & 0\ 0\\ 0\ 0 & \cdots & 0\ 1\\ \vdots\ \vdots & \ldots & \vdots\ \vdots\\ 1\ 1 & \cdots & 1\ 0\\ 1\ 1 & \cdots & 1\ 1\end{pmatrix},$$

the decimal code of the i-th row ($b_{i,0},\ b_{i,1},\ \cdots,\ b_{i,m-1}$) of B, $\sum_{j=0}^{m-1}b_{i,j}\cdot 2^{m-1-j}$, is $i(i=0,1,\cdots,2^m-1)$. $A^{(0)}$ is a ($2^m\times n$)-order zero array, and $A^{(i)}$ is the array with all identical rows, and the decimal code of every row is i ($i=1,2,\cdots,2^n-1$). At the same time, $P^{(0)}$ is a $(2^m\times(n+m))$-order zero array, and

$$P^{(i+1)}=P^{(i)}+D. \tag{3}$$

Where D is a $2^m\times(n+m)$-order array:

$$D=\begin{pmatrix}0\cdots 0 & 0\ 0\cdots 0\ 0\ 0\\ 0\cdots 0 & 0\ 0\cdots 0\ 0\ 1\\ 0\cdots 0 & 0\ 0\cdots 0\ 1\ 0\\ \vdots\ \vdots\ \vdots & \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\\ 0\cdots 0 & 1\ 1\cdots 1\ 1\ 0\\ 0\cdots 0 & 1\ 1\cdots 1\ 1\ 1\end{pmatrix}.$$

In D, its first column to the n-th column is $A^{(0)}$, the $(n{+}1)$ -th column to the last column is B, and the i-th row of $P^{(i)}$ adds to the i-th row of D is the i-th row of $P^{(i+1)}$, the "+" is binary addition. So $P^{(i)}(i=1,2,\cdots,2^n-1)$ can be obtained by the recursion formula (3). For example,$P^{(1)}=D,\ P^{(2)}=P^{(1)}+D=D+D$, i,e

$$P^{(2)}=\begin{pmatrix}0\cdots 0 & 0\,0\cdots 0\,0\,0\\ 0\cdots 0 & 0\,0\cdots 0\,0\,1\\ 0\cdots 0 & 0\,0\cdots 0\,1\,0\\ \vdots\ \vdots\ \vdots & \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\\ 0\cdots 0 & 1\,1\cdots 1\,1\,0\\ 0\cdots 0 & 1\,1\cdots 1\,1\,1\end{pmatrix}+\begin{pmatrix}0\cdots 0 & 0\,0\cdots 0\,0\,0\\ 0\cdots 0 & 0\,0\cdots 0\,0\,1\\ 0\cdots 0 & 0\,0\cdots 0\,1\,0\\ \vdots\ \vdots\ \vdots & \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\\ 0\cdots 0 & 1\,1\cdots 1\,1\,0\\ 0\cdots 0 & 1\,1\cdots 1\,1\,1\end{pmatrix}$$
$$=\begin{pmatrix}0\cdots 0 & 0\,0\cdots 0\,0\,0\\ 0\cdots 0 & 0\,0\cdots 0\,1\,0\\ 0\cdots 0 & 0\,0\cdots 1\,0\,0\\ \vdots\ \vdots\ \vdots & \vdots\ \vdots\ \vdots\ \vdots\ \vdots\ \vdots\\ 0\cdots 1 & 1\,1\cdots 1\,0\,0\\ 0\cdots 1 & 1\,1\cdots 1\,1\,0\end{pmatrix}$$

Furthermore, the $2^{(m+n)}$-bit binary outputs of the $n+m$ Boolean functions implementing the $(n+m)$-bit multiplier, say $p_{n+m-1},p_{n+m-2},\cdots,p_2,p_1,p_0$ respectively are the last $n+m$ columns in Table 1. For example, the $2^{(m+n)}$-bit binary outputs of p_0 is: $p_0=(p^{(0)},p^{(1)},\cdots,p^{(2^n-1)})$, where $p^{(0)}=(0,0,\cdots,0),p^{(1)}=(0,1,0,1,\cdots,0,1)$, and if i is an even number, $p^{(i)}=p^{(0)}$, if i is an odd number, $p^{(i)}=p^{(1)}(i=0,1,\cdots,2^n-1)$. Since $p^{(i+1)}$ can be obtained from $p^{(i)}$ by the recursive formula (3), therefore $p_{n+m-1},p_{n+m-2},\cdots,p_2,p_1,p_0$ are called multiplier Boolean functions (MBFs).

2.3 Truth table and Boolean functions of 3-bit by 2-bit NNBM

For the specific input number of a NNBM, its truth table and Boolean functions are easily determined. The truth table of the 3-bit by 2-bit NNBM is shown in Table 2.

It generates five Boolean functions p_0,p_1,p_2,p_3 and p_4, each holds 2^5 binary output symbols.

$$\begin{aligned}p_0&=0000\ 0101\ 0000\ 0101\ 0000\ 0101\ 0000\ 0101,\\ p_1&=0000\ 0011\ 0101\ 0110\ 0000\ 0011\ 0101\ 0110,\\ p_2&=0000\ 0000\ 0011\ 0010\ 0101\ 0101\ 0110\ 0111,\\ p_3&=0000\ 0000\ 0000\ 0001\ 0011\ 0011\ 0010\ 0010,\\ p_4&=0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001\ 0001.\end{aligned}$$

3. Decomposition of MBF and Perceptron Implementing NNBM

It was known that all Boolean functions can be divided into two classes: linearly separable Boolean function (LSBF) and non-linearly separable Boolean function (non-LSBF). Based on the criterion for LSBF [9-11], the MBF p_0 and p_4 are linearly separable, and the MBF p_1, p_2 and p_3 are non-linearly separable. Furthermore, any non-LSBF can be decomposed as the logic $\oplus$ (XOR) operation of a sequence of LSBFs. By using the DNA-like algorithm in [9,10], it is easily to know that p_1, p_2 and p_3 can be decomposed as:

Table 2: Truth table of 3-bit by 2-bit NNBM

a_2, a_1, a_0	b_1, b_0	p_4, p_3, p_2, p_1, p_0
0 0 0	0 0	0 0 0 0 0
0 0 0	0 1	0 0 0 0 0
0 0 0	1 0	0 0 0 0 0
0 0 0	1 1	0 0 0 0 0
0 0 1	0 0	0 0 0 0 0
0 0 1	0 1	0 0 0 0 1
0 0 1	1 0	0 0 0 1 0
0 0 1	1 1	0 0 0 1 1
0 1 0	0 0	0 0 0 0 0
0 1 0	0 1	0 0 0 1 0
0 1 0	1 0	0 0 1 0 0
0 1 0	1 1	0 0 1 1 0
0 1 1	0 0	0 0 0 0 0
0 1 1	0 1	0 0 0 1 1
0 1 1	1 0	0 0 1 1 0
0 1 1	1 1	0 1 0 0 1
1 0 0	0 0	0 0 0 0 0
1 0 0	0 1	0 0 1 0 0
1 0 0	1 0	0 1 0 0 0
1 0 0	1 1	0 1 1 0 0
1 0 1	0 0	0 0 0 0 0
1 0 1	0 1	0 0 1 0 1
1 0 1	1 0	0 1 0 1 0
1 0 1	1 1	0 1 1 1 1
1 1 0	0 0	0 0 0 0 0
1 1 0	0 1	0 0 1 1 0
1 1 0	1 0	0 1 1 0 0
1 1 0	1 1	1 0 0 1 0
1 1 1	0 0	0 0 0 0 0
1 1 1	0 1	0 0 1 1 1
1 1 1	1 0	0 1 1 1 0
1 1 1	1 1	1 0 1 0 1

$$\begin{aligned}
P_1 &= 00000011010101100000001101010110\\
&= 00111111011111110011111101111111\\
&\oplus 00111111001111110011111100111111\\
&\oplus 00000011000101110000001100010111\\
&\oplus 00000000000000010000000000000001,\\
P_2 &= 00000000001100100101010101100111\\
&= 00000000001100110111011111111111\\
&\oplus 00000000000100010011001111111111\\
&\oplus 00000000000100000001000101110111\\
&\oplus 00000000000000000000000000010000,\\
P_3 &= 00000000000000010011001100100010\\
&= 00000000000000010011001100110011\\
&\oplus 00000000000000000000000000010001.
\end{aligned}$$

Perceptron, inspired by the threshold logic unit neuron model of McCulloch and Pitts, introduced by Rosenblatt, is one of the most important and significant aspects of ANN [7,8].

It was also known that LSBF can be realized by an single-layer perceptron (SLP), and non-LSBF needs MLP to realize it. Furthermore, let $(u_1, u_2, u_3, u_4, u_5) = (a_2, a_1, a_0, b_1, b_0)$ and by using the DNA-like learning algorithm [9-11], the weights-threshold values of a perceptron is easy to be determined. Thus, all SLP and MLP implementing p_0, p_4, p_1, p_2 and p_3 can be obtained as follows.

The SLP implementing p_0 is

$$y = f(2u_3 + 2u_5 - 3) \tag{4}$$

The SLP implementing p_4 is

$$y = f(2u_1 + 2u_2 + 2u_4 + 2u_5 - 7) \tag{5}$$

The MLP implementing p_1 is

$$\begin{cases} y_1^{(1)} = f(2u_2 + 4u_3 + 4u_4 + 2u_5 - 3) \\ y_2^{(1)} = f(2u_3 + 2u_4 - 1) \\ y_3^{(1)} = f(2u_2 + 4u_3 + 4u_4 + 2u_5 - 7) \\ y_4^{(1)} = f(2u_2 + 2u_3 + 2u_4 + 2u_5 - 7) \\ \quad y = f(2y_1^{(1)} - 2y_2^{(1)} + 2y_3^{(1)} - 2y_4^{(1)} - 1) \end{cases} \tag{6}$$

The MLP implementing p_2 is

$$\begin{cases} y_1^{(2)} = f(6u_1 + 4u_2 + 4u_4 + 2u_5 - 7) \\ y_2^{(2)} = f(6u_1 + 4u_2 + 4u_4 + 2u_5 - 9) \\ y_3^{(2)} = f(6u_1 + 4u_2 - 2u_3 + 4u_4 + 4u_5 - 11) \\ y_4^{(2)} = f(2u_1 + 2u_2 - 2u_3 + 2u_4 + 2u_5 - 7) \\ \quad y = f(2y_1^{(2)} - 2y_2^{(2)} + 2y_3^{(2)} - 2y_4^{(2)} - 1) \end{cases} \tag{7}$$

The MLP implementing p_3 is

$$\begin{cases} y_1^{(3)} = f(6u_1 + 2u_2 + 2u_3 + 8u_4 + 2u_5 - 13) \\ y_2^{(3)} = f(2u_1 + 2u_2 + 2u_4 + 2u_5 - 7) \\ \quad y = f(2y_1^{(2)} - 2y_2^{(2)} - 1) \end{cases} \tag{8}$$

Where f is the first-order jump function defined by

$$f(x) = sign(x) = \begin{cases} 1 & \text{if } x > 0\,, \\ 0 & \text{if } x \le 0 \end{cases} \tag{9}$$

Summarizing (4) to (8), the networks of 3-bit by 2-bit NNBM is obtained **Fig. 3**.

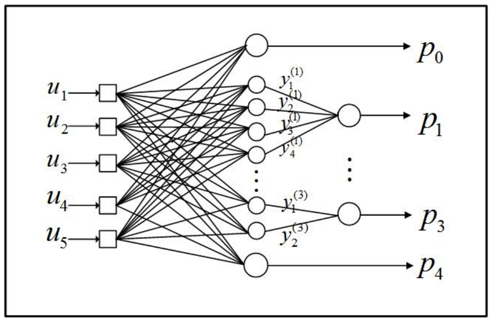

Fig. 3: MLPs implementing NNRM

4. Remarks and Conclusions

As stated in the previous section, an NNBM with n-bit by m-bit inputs will receives $n + m$ inputs and generates $n + m$ MBFs as outputs, and the serial MBFs is called to have implemented the $(n + m)$-bit multiplier.

The efficiency of NNBM and digital multiplier can be evaluated by comparing the number of hidden neurons of the MLPs with the number of product terms required to implement the MBFs [12].

According to formula (1), it is easy to know that $p_i = mod(\sum_{j+k=i} a_j b_k + c_i, 2)$, and $c_{i+1} = [\sum_{j+k=i} a_j b_k / 2]$ ($i = 0, 1, 2, 3$; $j = 0, 1, 2$; $k = 0, 1$), then the number of product terms of 3-bit by 2-bit digital multiplier is $2 \times 3 \times 2 \times 4 = 48$. However, the number of hidden layer neurons of the MLPs is only 10.

The successful design of computational systems is often predicated on the realization of fast multiplication in digital or analog hardware. A key design issue is the tradeoff between speed, complexity, and chip area. With this in mind and some advantages of ANN such as synchronous, parallel and fast speed, an innovative fast neural network multiplier has been designed in this paper. Similarly, in the future, other combinational logic circuits such as selector, code translator, and so on, can be considered by using ANN. They will be able to perform various real-world applications in the microprocessor technology.

Acknowledgment

This research was jointly supported by NSFC (Grants 11171084 and 60872093) and the Hong Kong Research Grants Council (Grant No. CityU117/10).

References

[1] R. Guanasekaran, "A Fast Serial-Parallel Binary Multiplier," *IEEE Trans. Computers*, vol. 34, pp. 741–744, 1985.

[2] H. M. A. Andree, G. T. Barkema, W. Lourens, and A. Taal, "A Comparison Study of Binary Feedforward Neural Networks and Digital Circuits", *Neural Network*, vol. 6, pp. 785–790, 1993.

[3] M. C. Wen, S. J. Wang, Y. N. Lin, "Low-power Parallel Multiplier with Column Bypassing", *Electronics Letters*, vol. 41, pp. 581–583, 2005.

[4] J. D. Moni, A. K. Priyadharsini, *Design of Low-Power and High Performance Radix-4 Multiplier*, New York, U.S.A.: IEEE Press, 2012.

[5] D. C. Biederman, E. Ososanya, "Capacity of Several Neural Networks with Respect to Digital Adder and Multiplier", in *Proc. Southeastern Symposium on System Theory'27*, 1995, p. 305–308.

[6] D. C. Biederman, E. Ososanya, "Design of a Neural Network-based Digital Multiplier", in *Proc. Southeastern Symposium on System Theory'29*, 1997, p. 320–326.

[7] W. S. McCulloch, W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity", *Bulletin of Math, Biophysics*, vol. 5, pp. 115–133, 1943.

[8] F. Rosenblatt, "The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain", *Cornell Aeronautical Laboratory, Psychological Review*, vol. 65, pp. 386–408, 1958.

[9] F. Y. Chen, G. R. Chen, G. L. He, X. B. Xu and Q. B. He, "Universal Perceptron and DNA-like Learning Algorithm of Binary Neural Networks: LSBF and PBF Implementation", *IEEE Trans. Neural Netw.*, vol. 20, pp. 1645–1658, 2009.

[10] F. Y. Chen, G. R. Chen, Q. B. He, G. L. He, and X. B. Xu, "Universal Perceptron and DNA-like Learning Algorithm of Binary Neural Networks: Non-LSBF Implementation", *IEEE Trans. Neural Netw.*, vol. 20, pp. 1293–1301, 2009.

[11] F. Y. Chen, G. L. He, G. R. Chen, "Realization of Boolean Functions via CNN: Mathematical Theory, LSBF and Template Design", *IEEE Trans. Circ. Syst. I*, vol. 53, pp. 2203–2213, 2006.

[12] H. M. A. Andree, G. T. Barkema, W. Lourens, and A. Taal, "A Comparison Study of Binary Feedforward Neural Networks and Digital Circuits", *IEEE Trans, Neural Netw.*, vol. 6, pp. 785–790, 1993.