### SieveNet: Ternary-Coded Binary Based Approximate Sieve for Neural Network

*Chia-Yu Chou, Tzu-Yi Hung and Tsung-Chu Huang*

Department of Electronics Engineering, National Changhua University of Education, Taiwan ROC

#### Abstract

In approximate computing acceleration of AI neural network, the pruning techniques are usually simple in concept but suffer in efficiency. In this paper we proposes a simple and efficient approximate approach for floating-point multiply-add accumulator during batch forward propagation. During batch forward propagation, the weights of the neural network keep constant and are worthy of approximately converting to ternary-coded binary fixed-point numbers. Then the associated input floating-point numbers are sieved by checking its exponents only. About 20% of inputs are selected for summation. Finally at least 3 folds of the computation can be sieved out. From practicatical evaluation, about 70% of real time can be reduced with an acceptable accuracy loss.

**1. Introduction**

Consumer electronics have driven computing acceleration for AI development by seven times per year [1]. The neural networks (NN) in most consumer electronics require acceleration and approximation due to limited cost. The efficiency of pruning the multiply-add accumulation (MAC) trees becomes the most critical issue.

Numeric precision scaling [3] and truncation [4] are two major technologies in approximate computing. Approximate error control can be pre-computing analyses during compilation [5], design [6] and in-computing adapting [7]. However all of their error adaption requires a lot of efforts and thus limits the efficiency.

The authors of references [8][9] took advantage of the property of constant weights in NNs during forward propagations in a batch for converting all constant-number multiplications to additive accumulation. In a batch, the network weights stay semi-constant, namely, constant during a long period. They proposed an adaptive approximate computing technique for batch leaning. The learning accuracy error was then applied for dynamic error control. However, the fixed-point quantization per sample of input datasets highly reduced the acceleration and limited the applications using floating-point datasets. Moreover, the acceleration will be highly reduced by the sorting complexity in their approach.

In this paper we propose a ternary-coded binary (TCB) based sieving technique, called SieveNet, for improving the efficiency in [8] in floating-point-input applications. In Sec.2 the arithmetic weight (AW) minimization algorithm is reviewed. The proposed SieveNet technique is proposed in Sec.3. In Sec. 4 the improvements are estimated for comparison. Finally, the conclusions are withdrawn in Sec. 5.

**2. Reviews of Related Number Formats**

A n=32-bit IEEE 754 floating-point (FP) is represented as

, (1)

where *s* is sign and exponent . *eg*., π=3.14⋯=0\_10000000\_10010010000111111011011.

An *n*-bit two’s complemented binary integer *x* can represented as

, (2)

where . It can be also represented by a ternary-coded binary (TCB) number,

, (3)

where and signs represent respectively. The AW of *x* is then defined as the minimum count of signs for representing, namely,

. (4)

The uniform-shift binary-to-TCB conversion can be found in Booth encoders [10]. For convenience of explanation, the AW-minimization algorithm is briefly reviewed as follows.

The encoder takes unsigned integer as a sequence feeding from bit by bit and outputs a TCB sequence . Fig.1(a) shows finite state machine (FSM) based algorithm. As shown in Fig.1(b), an example can help understand soon, say, 11111001 will be converted to +000-00+ via states S, 1, S, S, 1, 11, , , 0, and E. The AW is reduced to 4.

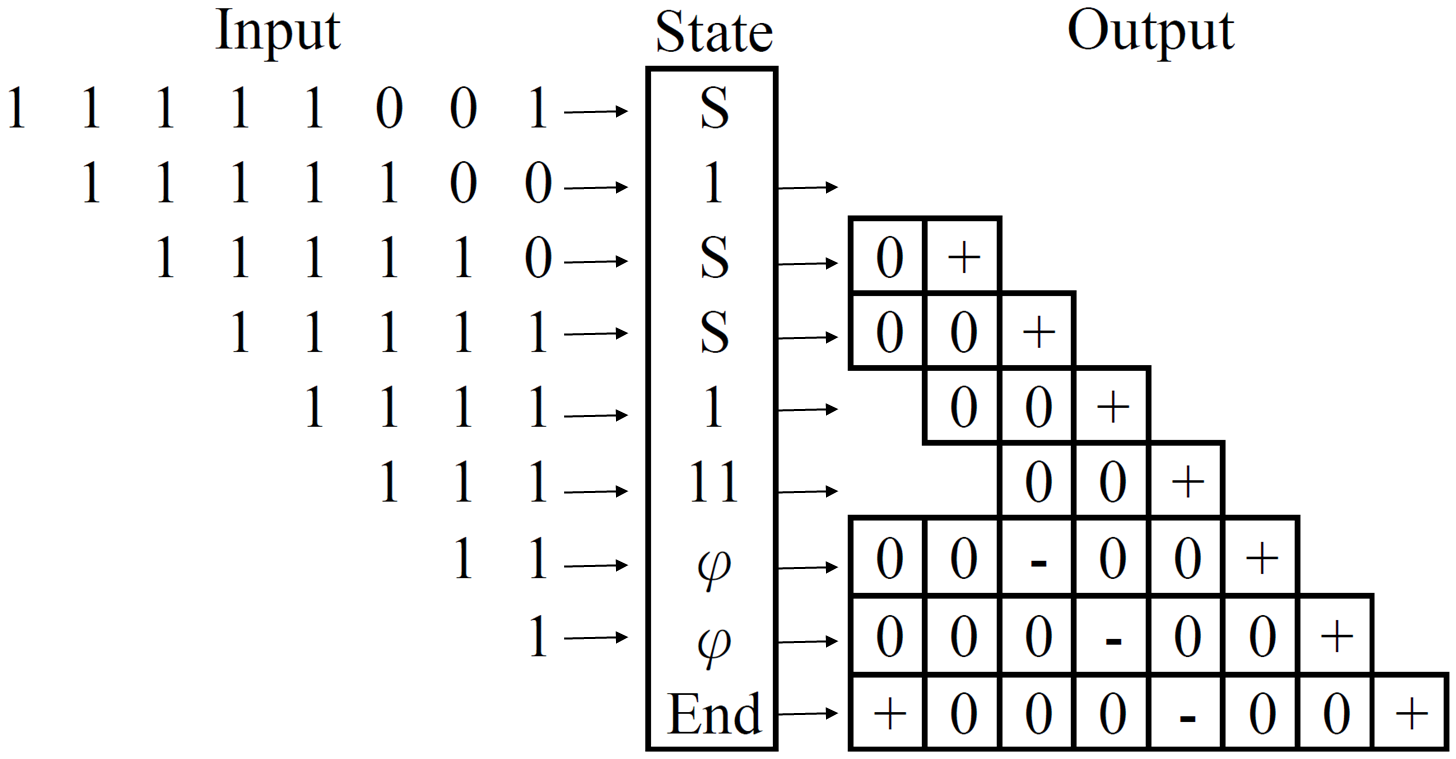
(a)(b)

Fig. 1 (a) Proposed TCB conversion, and (b) an example.

**3. proposed Approximate Computing**

Fig.2(a) shows a typical NN with layers composed of a bunch of MACs as shown in (b) represented as

, where and . (5)

(a)(b)

Fig. 2 (a) a typical NN with (b) MACs, and (b) our object.

Without loss of generality, only a MAC accumulated by a FP adder is considered and the the layer and output indices l and j in Eq.(4) are implied as a sum, , where and are FP numbers in most applications. In the related works [8][9], they are approximated to *BA* and *BW* bits of fixed-point TCB numbers, and , where the fixed-point positions *p* and *q* are usually implied by users. Allitems are sorted in advance in [8] and take a high time complexity.

In this paper we propose a direct sieving approach without sorting as shown in the python-style algorithm in Fig.3. An error rate *2r* is initialized as *2ro* and and adaptively reduced for increasing accuracy of next epoches. Line 1 of SieveNet algorithm defines given data during a forward propagation batch.. Since *W*[*n*] are constant during a batch they are converted to TCBs in Line 2. In Line 3, the sum S is initialized as 0 and its exponent in IEEE 754 is easily fetched as 0. To make the sievingThreshold a constant for as many as possible operations, the bit-index-first sieving is selected for loops in Lines 5 and 6. Then scanning all inputs, the numbers with exponents approximately less than the sieveThreshold are directly sieved. Otherwise the input is accumulated as illustrated in Fig.4, where the gray-colored FP numbers are skipped out. Although modern high-level syntheses (HLS) techniques can convert a sequential algorithm to a parallel hardware, only the sequential algorithm is explained without loss of generality.

1 def Sieve(float A[n], float W[n], int B, int p, r):

2 TCB(W[n], W[n][B])

3 S = 0

4 for each k in range(B):

5 sievingThreshold = r + exponent(S) - p - k

5 for each i in range(n):

6 if the kth bit of W[n][B] != 0: // 1st layer sieving

7 if exponent(A[n]) sievingThreshold: // 2nd sieving

8 S += A[n]

Fig. 4 Proposed SieveNet algorithm.



Fig. 4 (a) a typical NN with (b) MACs, and (b) our object.

In Fig.3 and 4, we can find that all multiplications are converted to additions. Although the multiplications can be improved by Vedic maths or Toom-Cook algorithm, the worst critical path still takes double time of additions. As a result the theoretical contributions are retrieved from multiply-to-add conversion, TCB sieving and exponent sieving.

Finally, an approximate error controllable learning flow is established as Fig.5 shows. In Line 9, the error level (EL) in dB-inaccuracy is proposed as an accuracy () metrics in deep learning as

. (6)

Accuracy deceleration between consecutive epochs is measured to modify the error-rate exponent *r* and thus sievingThreshold. Finally the approximation error can be converged and well controlled. Eventually almost the same precision and accuracy can be achieved in the end.

1 def ErrorControl(dataset, network, Ep, Ba):

2 Initializing model(dataset, network), B, p, r and intial error

3 for each epoch:

4 randomize, batching and normalization

5 for each batch:

6 Sieve(A[*n*], W[*n*], *B, p, r*)

7 backpropagation   
 8 adapting learning rate

9 *calculate EL and for reducing error rate exponent r*

10 statistics and report

Fig. 5 Proposed approximate error controllable learning flow.

From our preliminary work, the average and maximum AWs of an -bit number is minimized to and . For a neuron in the th layer, the screened addend size will be far less than about , so a scalable carry save adder (CSA) [11] is designed to substitute -bit multipliers.

**4. Simulation results**

In the preliminary experiments, only the FP multiplication/addition counts, and normalized execution time are estimated in Python although some FPGA evaluation using Pynq-z2/Zynq7020 has been verified for several small MACs in hardware most recently.

*A. TCB Sieving Rate*

Fig.6(a) shows the TCB sieving rate can be improved from 50% using binary codes to 66% using TCB with an improvement of 33%. Fig.6(2) shows comparison of the accuracy curves of MNIST NN784-16-10-Softmax respectively for usual work, previous work [8] and proposed SieveNet.

(a) (b)

TBD

TBD

Fig. 6 (a) Sieving rate due to TCB, and (b) Cross-validation accuracy.

an evaluation result for the MNIST 784-16-Softmax NN for hand-written digit perception. The green middle curve shows the learning curve without approximation. When the multiplicity thresholds are set to 8, 16, 24 and 32, the required batch count is postponed for about 25-30% as the right red curve shows. However, because the batches using approximate computing can be highly accelerated, the real time for total learning can be reduced. The pink arrow and the green area separately show the time saved and energy saved.

Table 1 lists a set of dense NNs for evaluations. Dropout and other pruning skills are not considered here. All NNs are executed until the same accuracies are achieved. We find that on average more than one half of real time and energy can be reduced although additional iterations are required.

Table 1. Experimental Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Neural Network | Traditional | [8] | Ours | Reduction |
| #Batches |  |  |  |  |
| Batch Size |  |  |  |  |
| #Epochs |  |  |  |  |
| #Additions |  |  |  |  |
| #Multiplications |  |  |  |  |
| A/W Format |  |  |  |  |
| Sieving Rate |  |  |  |  |
| Execution Time |  |  |  |  |

**5. Conclusions and Future Work**

In the paper we have proposed an efficient pruning technique called SieveNet for improving the sieving rate in terms of operation counts for more than 300% compared with sorting in [8]. All the evaluation will be varified using Pynq-z2/Zynq7020 system as soon in the future.

**References**

1. OpenAI. AI and Compute. https://openai.com/blog/ai-and-compute. May 16, 2018.
2. Sparsh Mittal. A Survey of Techniques for Approximate Computing. March 2016ACM Computing Surveys 48(4).
3. M. Gao and G. Qu, "Estimate and Recompute: A Novel Paradigm for Approximate Computing on Data Flow Graphs," IEEE Trans. CAD of ICs and Sys. Dec. 2018.
4. R. R. Osorio and G. Rodríguez, "Truncated SIMD Multiplier Architecture for Approximate Computing in Low-Power Programmable Processors," IEEE Access, vol. 7, pp. 56353-56366, 2019.
5. K. Nepal *et.al*., "Automated High-Level Generation of Low-Power Approximate Computing Circuits," IEEE Trans. on Emerging Topics in Computing, 7(1):18-30, 2019.
6. W. Liu et al., "Design and Analysis of Approximate Redundant Binary Multipliers," IEEE Trans. Computers, 68(6):804-819, 1 June 2019.
7. D. Gadioli, E. Vitali, G. Palermo and C. Silvano, "mARGOt: A Dynamic Autotuning Framework for Self-Aware Approximate Computing," IEEE Trans. Computers, 68(5):713-728, 1 May 2019.
8. L.-Y. Lin, J. Schroff, C.-C. Liang and T.-C. Huang. "Approximate Computing for Batch Learning in Neural Network," EEE Int’l Conf. on Consumer Electronics - Taiwan, Sep. 29, 2020.
9. C.-D. Tsai, T.-Y. Chen, H.-W. Fu and T.-C. Huang. "TCBNN: Error-Correctable Ternary-Coded Binarized Neural Network," 2021 IEEE Int’l Conf. on AI Circuits and Systems (AICAS2021), June 7, 2021.
10. L.P. Rubinfield, "A Proof of the Modified Booth's Algo-rithm for Multiplication," IEEE Trans. Computers, C-24(10):1014-1015, Oct. 1975.
11. T. Kim, W. Jao and S. Tjiang, "Circuit optimization using carry-save-adder cells," Trans. CAD of ICs & Sys. 17(10): 974-984, Oct. 1998.