Towards Co-designing Neural Network Function Approximators with In-SRAM Computing

Shamma Nasrin
University of Illinois at Chicago
snasri2@uic.edu

Diaa Badawi
University of Illinois at Chicago
dbadaw2@uic.edu

Ahmet Enis Cetin
University of Illinois at Chicago
aecyy@uic.edu

Wilfred Gomes
Intel Corporation
wilfred.gomes@intel.com

Amit Ranjan Trivedi
University of Illinois at Chicago
amitrt@uic.edu

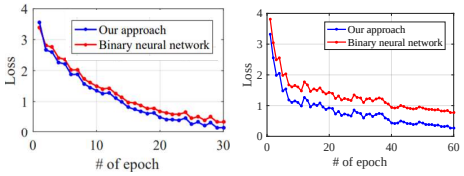
Abstract

We propose a co-design approach for compute-in-memory inference for deep neural networks (DNN). We use multiplication-free function approximators based on $l1$ norm along with a co-adapted processing array and compute flow. Using the approach, we overcame many deficiencies in the current art of in-SRAM DNN processing, such as the need for DACs at each operating SRAM row/column, high precision ADCs, and limited support for multi-bit precision weights, and limited vector-scale parallelism. We also propose an SRAM-immersed successive approximation ADC (SA-ADC). We exploit the parasitic capacitance of bit lines of SRAM array as a capacitive DAC, allowing low area implementation of within-SRAM SA-ADC. Our 8×62 SRAM macro requires a 5-bit ADC, achieves 105TOPS/W with 8-bit input/weight processing at 45 nm CMOS. We evaluated the performance of our proposed network for MNIST, CIFAR10, and CIFAR100 datasets. We chose a network configuration which adaptively mixes multiplication-free and regular operators. The network configurations utilize the multiplication-free operator for more than 85% operations from the total. The selected configurations are 98.6% accurate for MNIST, 90.2% for CIFAR10, and 66.9% for CIFAR100. Since most of the operations in the considered configurations are based on proposed SRAM macros, our compute-in-memory's efficiency benefits broadly translate to the system-level.

1 Introduction

In many practical applications, deep neural networks (DNNs) have shown remarkable prediction accuracy. DNNs in these applications typically utilize thousands to millions of parameters (i.e., weights) and are trained over a huge number of example patterns. Operating over such a large parametric space, which is carefully orchestrated over multiple abstraction levels (i.e., hidden layers), facilitates DNNs with a superior generalization and learning capacity but also presents critical inference constraints, especially when considering real-time and/or low power applications. For instance, when DNNs are mapped on a traditional computing engine, the inference performance is strangled by extensive memory accesses, and the high performance of the processing engine helps little. A radical approach, gaining attention to address this performance challenge of DNN, is to

6th Workshop on Energy Efficient Machine Learning and Cognitive Computing (EMC2 2020), Online (San Jose, California).



(a) (b) (c)

Dataset	Accuracy on test set		
	Conventional	Binary	Multiplication free
MNIST	99.01%	97%	98.6%
CIFAR10	90.9%	85%	90.2%
CIFAR100	67.88%	-	66.9%

(d)

Figure 1: Training curves for the multiplication-free NN operator on (a) MNIST and (b) CIFAR10 data sets. (c) The effect of quantization on multiplication-free operator. (d) Multiplication-free operator vs. conventional DNN and binary neural network (BNN)

design memory units that can not only store DNN weights but also locally process DNN layers. Therefore, using such 'compute-in-memory' high volume data traffic between processor and memory units is obviated, and the critical bottleneck can be alleviated. Moreover, a *mixed-signal* in-memory processing of DNN operands reduces necessary operations for DNN inference. For example, using charge/current-based representation of the operands, the accumulation of products simply reduces to current/charge summation over a wire. In recent years, several in-SRAM DNN implementations have been shown. However, many critical limitations remain, which inhibit the scalability of the processing. Using CONV-SRAM[1] as a motivating example, we highlight these limitations – notably, the challenges are common to most other in-SRAM applications too. To compute the inner product of l-element weight and input vectors $\mathbf{w}$ and $\mathbf{x}$, l-DACs and one ADC are required. Since DACs are concurrently active, they lead to both high area and power. An ADC is needed to digitize the inner product of $\mathbf{w}$ and $\mathbf{x}$ vectors. If $\mathbf{x}$ is n-bit and ADC combines the output of l cells, the minimum necessary precision of the ADC is $\sim n + log_2(l)$ to avoid any quantization loss.

Addressing the challenges of state-of-the-art, we propose the following core concepts:

1. We use a multiplication-free neural network operator that eliminates high-precision multiplications in input-weight correlation[2]. In the new operator, the correlation of weight $\mathbf{w}$ and input $\mathbf{x}$ is represented as

$$\mathbf{w} \oplus \mathbf{x} = \sum_i \text{sign}(x_i) \cdot \text{abs}(w_i) + \text{sign}(w_i) \cdot \text{abs}(x_i) \tag{1}$$

   Here, $\cdot$ is an element-wise multiplication operator, $+$ is element-wise addition operator, and $\sum$ is vector sum operator. sign() operator is ± 1 and abs() operator produces absolute unsigned value of the operand. Therefore, in Eq. 1, the correlation operator is *inherently* designed to only multiply a one-bit element of sign($\mathbf{x}$) against full precision $\mathbf{w}$, and one-bit sign($\mathbf{w}$) against $\mathbf{x}$. By avoiding direct multiplications between full precision variables, DACs can be avoided in in-memory computing.Additionally, we reformulate Eq. 1 such a way that allows processing with single product port of SRAM cells; thus, it can reduce dynamic energy.

2. We also discuss that parasitic capacitance of bit lines of SRAM array can be exploited as a capacitive digital-to-analog converter (DAC) for successive approximation-based ADC (SA-ADC). In our architecture, when bit lines in one half of the array compute the weight-input correlation, bit lines in the other half implement binary search of SA-ADC to digitize the correlation output. Remarkably, the proposed DNN operator also helps reducing precision constraints on SA-ADC. With the proposed operator, each SRAM cell only performs 1-bit logic operation; thus, to digitize the output of l columns, ADC with $log_2(l)$ precision is needed. By simplifying data converters, our scheme can also achieve higher vector-scale parallelism, i.e., allows processing a higher number of parallel columns (l) with the same ADC complexity than in [1].

Sec. II introduces the co-adapted multiplication-free operators for the in-SRAM deep neural network. Sec. III gives an overview of compute-in-memory macro based on multiplication-free operator. Sec. IV discusses power/performance characterization of the proposed compute-in-SRAM macro. Sec. V concludes.

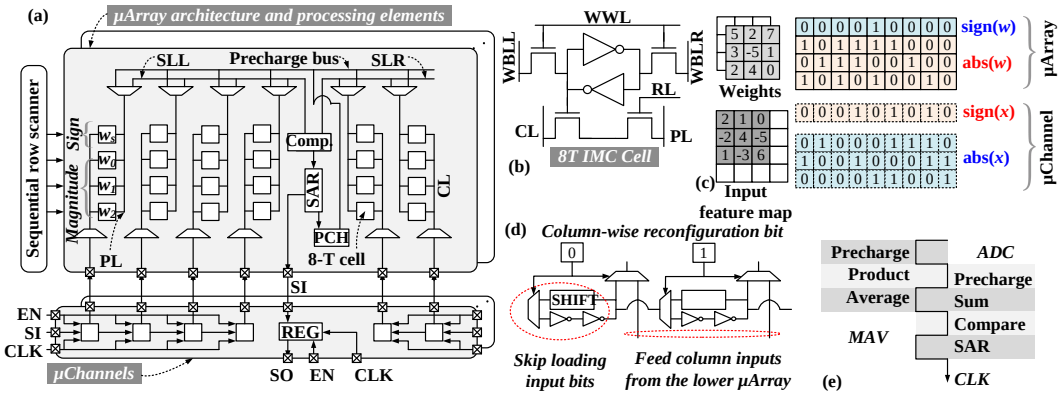


Figure 2: (a) μArray and μChannel architecture for $\mathbf{w} \oplus \mathbf{x}$ processing in a compute-in-SRAM macro. (b) 8-T SRAM cell for in-memory processing. SRAM cell has additional transistors, colored red, for input-weight correlation. (c) Mapping of sign bits and the absolute values of weights and inputs on μArrays and μChannels. (d) Stitching of μArray columns by reconfiguration bits in μChannels. (e) Instruction cycles for in-SRAM processing.

2 Co-designed Multiplication-free(MF) NN Operator and Accuracy results

Multiplication-free DNN operator was presented in[2]. In this work, we expand on the potential of multiplication-free operators to considerably reduce the complexity of SRAM-based compute-in-memory design. Compared to[2], we adjusted the operator with abs() operation on operands w and x in Eq. (1) to further simplify compute-in-memory processing steps. Our later discussion will show that the adjusted operator also achieves high prediction accuracy on various benchmark data sets. Note that a multiplication-free operator in Eq. (1) is based on the ℓ_1 norm, since $\mathbf{x} \oplus \mathbf{x} = 2||x||_1$. In traditional neural networks, neurons perform inner products to compute the correlation between the input vector with the weights of the neuron. We define a new neuron by replacing the affine transform of a traditional neuron using co-designed NN operator as $\phi(\alpha(\mathbf{x} \oplus \mathbf{w}) + b)$ where $\mathbf{w} \in R^d$, $\alpha, b \in R$ are weights, the scaling coefficient and the bias, respectively. Moreover, since the proposed NN operator is nonlinear itself, an additional nonlinear activation layer (e.g., ReLU) is not needed, i.e., $\phi()$ can be an identity function. Most neural network structures, including multi-layer perceptrons (MLP), recurrent neural networks (RNN), and convolutional neural networks (CNN), can be easily converted into such an *compute-in-memory compatible* network structures by just replacing ordinary neurons with the activation functions defined using $\oplus$ operations without modification of the topology and the general structure. We characterize the prediction accuracy of the proposed NN operator for MNIST, CIFAR10, and CIFAR100 data-sets. For MNIST, we simulate the LeNet-5 network. For CIFAR10, we use a convolutional neural network consisting of five convolution layers and two fully connected layers. For CIFAR100, we choose MobileNetV2. Figures 1(a-b) show the training curve for the multiplication-free operator compared to BNN under identical training conditions for MNIST and CIFAR10. In Figure 1(c), the prediction accuracy of multiplication-free operator is also amenable to input/weight quantization. Figure 1(d) summarizes test-set accuracy for the data sets with conventional, multiplication-free, and binarized network operators. For CIFAR10 results, the convolution layers are binarized in BNN and made multiplication-free in the proposed network, respectively, while the fully-connected layers are implemented using regular multiplications. The accuracy of the multiplication-free operator is also quite competitive to the conventional operator on various datasets.

3 Overview of Compute-in-SRAM Macro based on MF Operator

Figure 2(a) shows the proposed design of compute-in-SRAM macro for multiplication-free operator-based DNN inference. In the proposed design, an SRAM macro consists of μArrays and μChannels, as shown in the figure. Each μArray is dedicated to storing one weight channel. DNN weights are arranged across columns in a μArray where each bit plane of weights is arranged in a row. Figure 2(b) shows the proposed 8T SRAM cell used for the in-SRAM processing of the operator. The added transistors are selected by the row and column select lines (RL and CL) and operate on the product bit line (PL). Each μArray is augmented with a μChannel. μChannels convey digital inputs/outputs to/from μArrays. μChannels are essentially low overhead serial-in serial-out digital paths based on

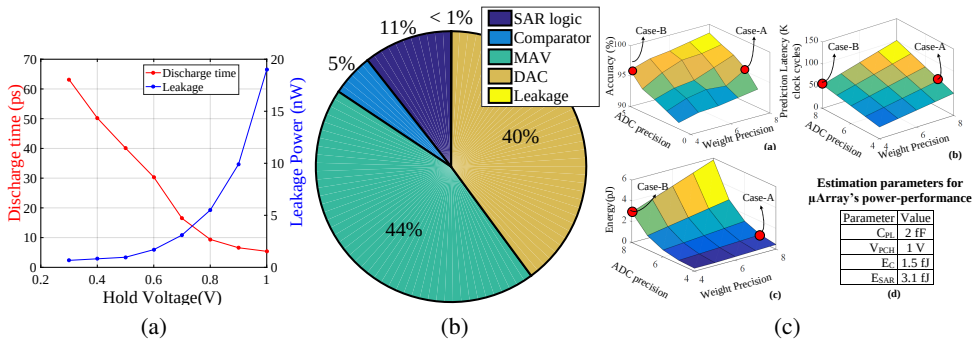


Figure 3: Power performance of SRAM array. (a) The product line discharge time and SRAM leakage power at varying hold voltage. (b) Distribution of dynamic and leakage energy for a μArray for performing MAV and within-SRAM digitization. (c) Surface plot for a. accuracy for MNIST characterization, and b. latency and c. energy of compute-in-SRAM macros at varying precision of weights and ADC.

scan-registers. Figure 2(d) shows μChannel-based column merging between two μArrays. Figure 2(c) illustrates input/weight mapping to SRAM macro and operation sequence. For step$(\mathbf{x}) \cdot$ abs$(\mathbf{w})$ step in $\mathbf{w} \oplus \mathbf{x}$, step$(\mathbf{x})$ vector is loaded on the μChannel and operated against abs$(\mathbf{w})$ rows of μArray. For step$(\mathbf{w}) \cdot$ abs$(\mathbf{x})$ step, bitplanes of abs$(\mathbf{x})$ vector are sequentially loaded on the μChannel and operated against step$(\mathbf{w})$ row of the μArray. In a μArray, to compute $\mathbf{x} \oplus \mathbf{w}$, the operation proceeds by bit planes. If the left half computes the weight-input product, the right half digitizes. Both halves subsequently exchange their operating mode to process weights stored in the right half. Figure 2(e) shows the instruction sequence for the left half to compute MAV consisting of precharge, product, and average stages.

Since MAV output at the sum line (SL) is charge-based, an analog-to-digital converter (ADC) is necessary to convert the output into digital bits. In Figure 2(a), the right half of the array implements an SRAM-immersed successive approximation (SA) data converter to digitize the output at the left sum line (SLL). Reference voltages for SA-based data conversion are generated by exploiting PL parasitic in the right half. The product lines of the right half are charged and discharged according to the SAR logic to produce the reference voltage at the right sum line (SLR). Figure 2(e) also shows the instruction cycles for the data conversion consisting of precharge, average, compare, and SAR steps.

4 Power–Performance Characteristics of Compute-in-SRAM Macro

In this section, we discuss power–performance characteristics of compute-in-SRAM macro presented earlier. We use LeNET-5 for MNIST characterization as a running use-case to discuss various design and power optimization opportunities. The network weights for LeNET-5 are trained using TensorFlow. Data transfer among SRAM arrays and post-processing of array's output, such as applying max-pooling, is simulated functionally. Compute-in-SRAM operations are simulated using HSPICE in 45 nm CMOS technology using predictive technology models. Each μArray in SRAM-macro has 8 rows and 62 columns. A μArray is split into halves, where each half stores a weighted channel by flattening it to one-dimensional. μArrays process 8-bit weights against 8-bit inputs. In each SA cycle in μArrays, MAV is digitized to 5-bit output. If the flattened filter width is more than 31, it is partitioned and mapped on more μArrays.

In Figure 3(a), reducing the hold voltage of SRAM cells reduces leakage current through the cells; however, also increases the PL discharge time during product computation. The average leakage power of a μArray is 0.97 nW at the typical corner. Figure 3(b) shows the distribution of power among various operations in a μArray. A μArray consumes $\sim$7.6 μW of active power to perform the MAV operation while operating at 1 GHz. In Figure 3(c), we explore the design space of varying weight precision (W_P) and ADC precision (A_P) for predictions on MNIST dataset. The following determines the clock cycles ($\mathcal{T}$) and average energy ($\mathcal{E}$) for the unit operation.

$$\mathcal{T} = W_P \times (1 + 2A_P) \tag{2a}$$

$$\mathcal{E} = W_P \times \left(MC_{PL}V_{PCH}^2 + \sum_{i=0}^{A_P-1} E_C + E_{SAR} + 2^i C_{PL} V_{PCH}^2 \right) \tag{2b}$$

Metric	Our	[3]	[4]	[5]	[6]
Tech (nm)	45	28	65	7	65
Weight bits	8-bit	8-bit	8-bit	4-bit	1-bit
Input bits	8-bit	8-bit	8-bit	4-bit	1-bit
Output bits	16-bit	20-bit	8-bit	4-bit	5-bit
Efficiency (TOPs/W)	105 (8×62 μArray) 84 (8×30 μArray)	7	2.96	321	671.5
MNIST	98.6%	-	99.28%	98.2%	98.3%
CIFAR10	90.2%	91.9%	86.62%	-	85.5%
CIFAR100	66.9%	67.8%	-	-	-

* CIFAR100 accuracy extracted using MobileNetV2

(a)

Network configuration for CIFAR10

Layer	% of Params	% of Operations	Ops Per param	Digital	In-mem	Mixed
Conv1	<1	67.56	900	R	MF	MF
Conv2	2.09	16.92	784	R	MF	MF
Conv3	4.18	8.43	144	R	MF	MF
Conv4	8.35	2.74	100	R	MF	MF
Conv5	16.7	1.38	49	R	MF	MF
FC1	67	2.43	1	R	MF	R
FC2	<1	<1	1	R	MF	R
Test Accuracy (%)				91	79	90.2
Avg. TOPs/W				2.8	105	100.91

(b)

Figure 4: (a) Comparison against state-of-the-art, and (b) Different network combination for CIFAR10 data set.

Here, C_{PL} is the product line (PL) capacitance. V_{PCH} is the precharge voltage for processing. E_C and E_{SAR} are the average energy of unit operation for the comparator and SA register logic. M is the number of columns in each half of μArray. From our simulations in 45 nm CMOS, Figure 3(c) shows various parameters in the equation. For C_{PL}, we add a 20% overhead from the transistor capacitance to account for the interconnect parasitic.

Figure 4(a) compares our results against the state-of-art on different data sets. For various datasets, our network configuration was discussed in Sec. II. We achieve better prediction accuracy than the competitive approaches by processing with multibit precision weights and inputs, whereas many prior approaches were limited to binarized weights. Our 8×62 SRAM macro, which requires a 5-bit ADC, achieves ∼105 tera operations per second per Watt (TOPS/W) with 8-bit input/weight processing at 45 nm CMOS. In Figure 4(b), we explore the effect of combining typical operator and multiplication free operator on accuracy and energy efficiency for CIFAR10 data set.

5 Conclusion

We presented a compute-in-SRAM macro based on a multiplication-free learning operator. The macro comprises low area/power overhead μArrays and μChannels. Operations in the macro are DAC-free. μArrays exploit bit line parasitic for low overhead memory-immersed data conversion. We characterized the accuracy of our scheme on MNIST, CIFAR10, and CIFAR100 data sets. On an equivalent network configuration, our framework has 1.8× lower error on MNIST and 1.5× lower error on CIFAR10 compared to the binarized neural network. At 8-bit precision, our 8×62 compute-in-SRAM μArray achieves ∼105 TOPS/W, which is significantly better than the current compute-in-SRAM designs at matching precision.

References

[1] A. Biswas and A. P. Chandrakasan, "Conv-sram: An energy-efficient sram with in-memory dot-product computation for low-power convolutional neural networks," 2019.

[2] C. E. Akbas¸, A. Bozkurt, A. E. C¸ etin, R. C¸ etin-Atalay, and A. U¨ ner, "Multiplication-free neural networks," 2015.

[3] J. Su, X. Si, Y. Chou, T. Chang, W. Huang, Y. Tu, R. Liu, P. Lu, T. Liu, J. Wang, Z. Zhang, "15.2 a 28nm 64kb inference-training two-way transpose multibit 6t sram computein-memory macro for ai edge chips," in ISSCC.

[4] J. Yue, Z. Yuan, X. Feng, Y. He, Z. Zhang, X. Si, R. Liu, M. Chang, X. Li, H. Yang, and Y. Liu, "14.3 a 65nm computing-in-memory-based cnn processor with 2.9-to-35.8tops/w system energy efficiency using dynamic-sparsity performance-scaling architecture and energy-efficient inter/intra-macro data reuse," in ISSCC, 2020, pp. 234–236.

[5] Q. Dong, M. E. Sinangil, B. Erbagci, D. Sun, W. Khwa, H. Liao, Y. Wang, and J. Chang, "15.3 a 351tops/w and 372.4gops computein-memory sram macro in 7nm finfet cmos for machine-learning applications,ISSCC.

[6] Z. Jiang, S. Yin, J. Seo, and M. Seok, "C3sram: An in-memorycomputing sram macro based on robust capacitive coupling computing mechanism," IEEE Journal of Solid-State Circuits.