Extended Abstracts of the 2018 International Conference on Solid State Devices and Materials, Tokyo, 2018, pp21-22

Using (emerging) memories for machine learning hardware.

D. Verkest, D. Rodopoulos, B. Verhoef, A. Mallik, J. Constantin, P. Debacker, J. Stuijt[1]. R. Appeltans, D. Garbin, A. Mocuta, G. S. Kar, A. Furnemont,

Imec, Kapeldreef 75, B-3001 Leuven, Belgium and [1]Imec-NL, Eindhoven, The Netherlands
Phone: +32-16-281.211 E-mail: diederik.verkest@imec.be

Abstract

The merit of dot-vector product implementations using (non-volatile) memory fabrics for more efficient machine learning needs to be evaluated at system level. We demonstrate the use of MRAM based cells for in-place binary multiplication with limited accuracy loss at the system-level of 1.4% for MNIST and <10% for CIFAR-10.

1. Introduction

GPUs are the dominant architecture for deep learning applications today. They offer an extremely high performance thanks to their parallel architecture that is well matched to typical convolutional neural networks (CNN), yet they are also very power hungry. To further increase machine learning efficiency, dedicated hardware accelerators are either added to general purpose multi-processor architectures or implemented as stand-alone ASICs. In this paper, we investigate a possible role for even more energy efficient solutions based on the use of memory fabrics for implementing the core computations of neural networks.

2. Deep neural networks

Deep neural networks typically contain multiple convolutional layers followed by a few dense layers. In the convolutional layers, filter kernels are convolved with input tensors, leading to a large level of re-use for the weights of the filters. The dense (fully-connected) layers, in contrast, have lower computational intensity with limited opportunities for data re-use. The dot product (weighted sum) is generally used both for dense layers and convolution layers and has been the core focus of prior art using non-volatile memory (NVM) technologies by leveraging their conceptual similarity with synaptic weights [1, 2, 3].

3. Implementing convolutions using memory elements

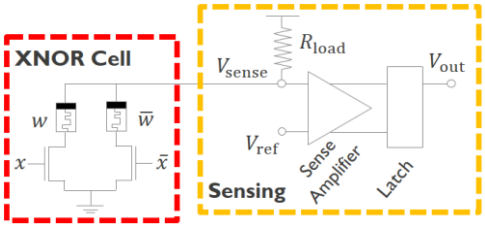
When using NVM for NN implementations, multi-level NVM feature obstacles such as high programming energy or variability, whereas stochastic binary NVM may require redundancy to ensure functionality [4]. Adaptation of NVM for NN implementations has a fundamental requirement: the efficiency improvement should be visible at the system level (from input raw data to classification output) instead of being restricted to a specific operation. Most approaches in the novel NVM technology trend, attempt to emulate the synaptic weight with memristor-like elements. We can sub-categorize this domain with respect to the number of NVM levels that are assumed. In the case of multi-level NVM, we come across technologies like TaOx-based memristors [1] and Phase-Change Memory (PCM) [2]. In both cases, the NVMs are assumed to hold the weight value. Voltage pulses are fed to the rows of the NVM array and the weighted currents are eventually summed at the bottom of each column. Despite attempts to optimize the mapping of the scalar weights on the inherently variable NVMs [1], a fundamental requirement for analog-to-digital conversion arises when that dot-product needs to be sensed out of the NVM array. This may be a limiting factor when interfacing with traditional CMOS logic, raising a major concern for the compatibility of these circuits with established architectures. Conversely, when NVM elements are treated as binary weights, multiple binary NVM elements are required to emulate a scalar synaptic weight [3], with the associated area and energy overheads.

3. Binary Neural Network using MRAM array

Based on the above considerations, we use STT-MRAM to implement an in-place multiplication that can be used for binary neural networks (BNN). We show how this concept needs to be used in different forms to suit different system contexts.

NVM Cell design

The dot-product is generally dominant in MLP/CNN data flows. Considering BNN [5], we propose multiplication between a binary activation and a binary weight using the cell show in Fig. 1, together with its logic description.



w ($\bar{w}$ being the complement)				x ($\bar{x}$ being the complement)			$w \times x$		V_{sense}	V_{out}
Numerical	Logical	Resistance	Magnetization	Numerical	Logical	Full Swing	Numerical	Logical	Half Swing	Full Swing
−1	0	R_{LRS}	0	−1	0	V_{ss}	+1	1	V_H	V_{dd}
−1	0	R_{LRS}	0	+1	1	V_{dd}	−1	0	V_L	V_{ss}
+1	1	R_{HRS}	π	−1	0	V_{ss}	−1	0	V_L	V_{ss}
+1	1	R_{HRS}	π	+1	1	V_{dd}	+1	1	V_H	V_{dd}

Fig. 1 The 2T-2MTJ cell and its truth table

In this illustration, we assume STT-MRAM as the binary NVM technology of choice, however any NVM with two clearly distinguishable resistive states (high resistive state – HRS, and low resistive state – LRS) is also applicable. The proposed binary multiplication cell leverages the equivalence between the numerical values of the BNN software assumptions (-1/+1), the logical values of digital logic (0/1), the resistance values of the NVMs (LRS/HRS) and the angle of the (out-of-plane) magnetization of the STT-MRAM's free layer. The two NVMs of the proposed cell hold the binary weight

value and its complement. The gate nodes of the two nFETs are pulsed according to the activation value and its complement. The XNOR (or multiplication) output appears in the sense node of the voltage divider as a half-swing readout voltage. For the latter value to be used in further digital logic, it must be sensed and translated to an equivalent full-swing voltage. This is a requirement that already exists in MRAM (and generally embedded memory) arrays and can be met using a simple sense amplifier (SA).

NVM array and system consideration

The binary product can be read out of an array that holds weight values in NVM elements. Overall, NN layers are envisioned as a data flow occupying different XNOR cell arrays, each layer followed by normalization and non-linearity layers as shown in Fig. 2. Assuming a densely connected layer receiving input activations from N neurons and being composed of M neurons, we can store all related binary weights in a NxM XNOR cell array. Each dense/convolution addend is created by signaling the appropriate word line. A pop-counter is needed to implement the sideways sum of the +1 or -1 addends: Starting from an initial value of 0, it increments if the read-out value is +1 or decrements if the read-out value is -1. Once the outputs of all XNOR cells of a column have been read out and counted, the pop-counter will contain the final value (signed integer) corresponding to that column.

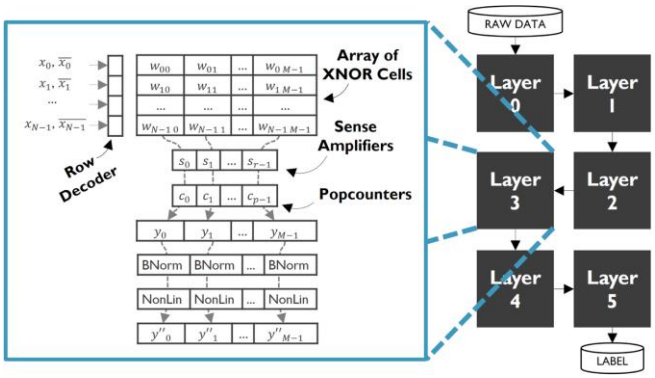


Fig. 2 System view including normalization and non-linearity layers.

With BNN assumptions, the data type of the dot-product output opens an interesting space for the optimization of the normalization and non-linearity operations that typically follow a dense or convolutional layer. Typical floating-point representations can be simplified to integer or even Boolean. The accuracy implications of these data-type refinements (DTR) are verified on the full test sets of the MNIST [6] and CIFAR10 [7] benchmarks.

Input representation

For BNN, inputs need to be recast to the (-1,+1) range. For MNIST input values can be directly binarized (see Fig. 3). However, for CIFAR input pixels have a wide range of values, so direct binarization would incur large errors. Instead we use a binary representation with Q bits for each pixel and use these Q bits as separate inputs to the NN.

4. Benchmarking

Accuracy is reported on the test set of each benchmark, without cropping/augmentation, assuming end-to-end binary weights/activations and the proposed DTRs. Fig. 3 shows MNIST accuracy for 3 hidden layers and different numbers of neurons (N) per hidden layer. The reference MLP [5] provides default accuracy, where inputs to the NN receive values in the (-1, +1) interval and normalization data structures retain float precision. Input binarization and DTR lead, on average, to less than 1.4% accuracy degradation for the same MLP topologies. For the CNN running CIFAR10 we sweep pixel quantization (Q). Default (reference) accuracy is derived from the public software binary CNN version [5], where inputs to the NN are not binarized (solid line in Fig. 3) and normalization parameters retain float precision. On average (across Q values), we observe less than 10% accuracy decrease due to DTR and input quantization.

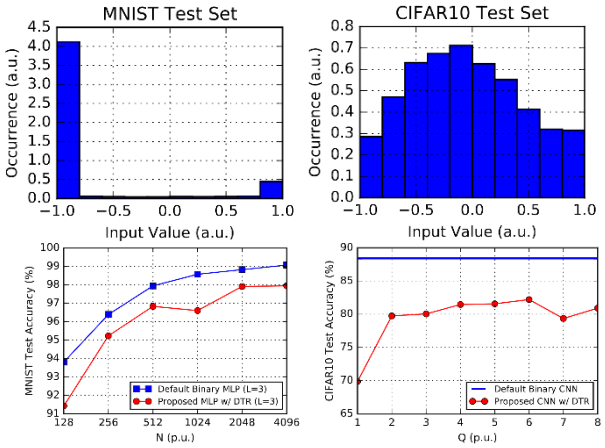


Fig. 3 MNIST and CIFAR 10 benchmark: input value histograms (top) and accuracy assessment (bottom).

5. Conclusions

In this work we discuss the enablement of NNs with NVMs, starting from Binary NNs. We opt for binary NVMs (STT-MRAMs) and design a cell for in-place multiplication using binary weights (stored in the NVM) and activations (pulsed as a word line). The proposed NVM cells perform an XNOR and can be organized with standard memory design practices. Data type refinement is applied to NN normalization, reducing the overall digital complexity. These optimizations lead to competitive accuracy on MNIST and CIFAR10, enabling end-to-end binary, NVM-based NN designs.

Acknowledgements

This research or part of this research is conducted within the imec "Machine Learning" IIAP.

References

[1] M. Hu et al., Proc. of *DAC,* (2016).

[2] G.W. Burr et al. *IEEE TED*, 62(11):3498-3507, 2015

[3]. D. Garbin et al, *IEEE TED*, 62(8), 2015.

[4] D. Querlioz et al., *Proc. of the IEEE*, 103(8)1398-1416, 2015.

[5] M. Courbariaux, *CoRR*, abs/1602.02830, 2016.

[6] Y. Lecun et al., http://yann.lecun.com/exdb/mnist/.

[7] A. Krizhevsky, Learning multiple layers of features from tiny images, Tech. Rep. Univ of Toronto, 2009