Scaling Up In-Memory-Computing Classifiers via Boosted Feature Subsets in Banked Architectures

Yinqi Tang, *Student Member, IEEE,* Jintao Zhang, *Student Member, IEEE,* and Naveen Verma, *Member, IEEE*

***Abstract*—In-memory computing is an emerging approach for overcoming memory-accessing bottlenecks, by eliminating the costs of explicitly moving data from point of storage to point of computation outside the array. However, computation increases the dynamic range of signals, such that performing it via the existing structure of dense memory substantially squeezes the signal-to-noise ratio (SNR). In this paper, we explore how computations can be scaled up, to jointly optimize energy/latency/bandwidth gains with SNR requirements. We employ algorithmic techniques to decompose computations so that they can be mapped to multiple parallel memory banks operating at chosen optimal points. Specifically focusing on in-memory classification, we consider a custom IC in 130nm CMOS IC [1], and demonstrate an algorithm combining error-adaptive classifier boosting (EACB) and multi-armed bandits (MABs), to enable segmentation of a feature vector into multiple subsets. The measured performance of 10-way MNIST digit classification, using images downsampled to 16×16 pixels (mapped across four separate banks), is 91%, close to that simulated using full unsegmented feature vectors. The energy per classification is 879.7 pJ, 14.3× lower than that of a system based on separated memory and digital accelerator.**

***Index Terms*—Boosting, feature segmentation, in-memory computing, machine learning, multi-armed bandits.**

I. Introduction

WHILE hardware accelerators substantially enhance energy efficiency and performance of logic, the gains do not transfer to memory operations [2]. This is especially problematic in emerging data-centric applications, such as machine-learning workloads, where the data-driven models involved for inference typically do not have compact parametric representations and thus require large amounts of memory and memory operations, which thus dominate in such systems. To address this, recently architectures have emerged integrating *computation in memory* [1], [3], [4]. As opposed to standard memory operations, where raw data is accessed one row at a time, the general principle behind in-memory computing is to access a computation result over many rows of data at once, thus amortizing the energy and latency.

As described below, this enables in-memory computing to achieve significant gains in energy/latency/bandwidth. However, computation over a large amount of data also increases the required dynamic range. Accommodating this within the existing hardware of a standard memory, such as SRAM, leads to reduced signal-to-noise ratio (SNR) within the mixed-signal approaches typically required. With different applications having different SNR requirements, the focus of this paper is on scaling up the computations under such SNR requirements. Previously, [3] introduced a circuit-level knob for addressing application SNR requirements, up to the allowable swing on one set of bit lines. Instead, we take an algorithmic approach for scaling up computations by mapping them across multiple in-memory-computing banks, to exploit the allowable swing across multiple sets of bit lines. In this way, each bank can operate at a design point where the mixed-signal energy/latency/bandwidth and SNR tradeoffs are optimized.

While different in-memory architectures have emerged [1], [3], [4], making use of different compute models and different memory technologies, the tradeoffs concerning energy/latency/bandwidth and SNR generally exist. To concretize these, we focus on a specific SRAM-based architecture for in-memory classification [1], which applies the tradeoffs aggressively by processing feature vectors of up to 128 dimensions in a fully row-parallel operation. We consider the problem of mapping higher-dimensionality feature vectors, noting that increasing feature-vector dimensionality is a common trend in machine learning. Specifically, while [1] demonstrated 81-dimensional feature vectors in one bank, achieving 10-way classification accuracy for MNIST images [5] of 90%, we demonstrate 256-dimensional feature vectors, partitioned into subsets and mapped across four banks (on separate ICs), achieving an accuracy of 91%.

In addition to increasing the dimensionality of feature vectors, composing features vectors (e.g., via layers within deep learning) leads to enhanced performance. Indeed, deep neural networks (NNs) can be readily mapped to in-memory architectures such as [1], since such layering simply requires cascading in-memory-computing banks, unconstrained by the underlying hardware tradeoffs. For instance, high classification accuracy has been demonstrated on MNIST images (e.g., 95% [6]). While such layering provides another algorithmic approach for increasing accuracy [7] and can be readily exploited by in-memory computing, scaling the dimensionality of operations (e.g., neuron synapses) remains an important direction, as it is directly impacted by the energy/latency/bandwidth and SNR tradeoffs underlying in-memory computing.

We propose an algorithm based on error-adaptive classifier boosting (EACB) [8] and multi-armed bandits (MABs) [9], to enable iterative training of linear classifiers. The algorithm is capable of both substantially overcoming analog non-idealities and optimally combining computations over the four feature subsets. We also analyze practical hardware requirements to support such an algorithm. Though demonstrated on [1], the algorithmic approach is applicable to a range of in-memory

Manuscript received March 13, 2018; accepted July 4, 2018. Date of publication August XX, 2018; date of current version October XX, 2018. This work was supported by NSF grant CCF-1253670. This brief was recommended by Associate Editor XX. *(Corresponding author: Yinqi Tang.)*

The authors are with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA (e-mail: yinqit@princeton.edu; jintao@princeton.edu; nverma@princeton.edu).

Color versions of one or more of the figures in this brief are available online at http://ieeexplore.ieee.org.

Digital Object Identifier XX

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2018.2854759, IEEE Transactions on Circuits and Systems II: Express Briefs

architectures, given the underlying hardware tradeoffs.

II. In-Memory Classifier Background

A. Hardware Architecture

The architecture of the in-memory classifier [1] is shown in Fig. 1. This consists of a standard 6T SRAM array with 128×128 bit cells and standard periphery for *SRAM Mode*, but also specialized periphery for *Classify Mode*. After bit-line pair (BL/BLB) precharging, the periphery for *Classify Mode* drives feature-vector elements as amplitude-modulated signals on the word lines (WLs) all at once. This is done via 5-b WL digital-to-analog converters (WLDACs), with 5 b chosen based on detailed simulations for the MNIST application [1]. The WL amplitudes set the bit-cell currents $I_{BC(i,j)}$, but whether these discharge BL/BLB is determined by the data stored in the cells. Treating each BL/BLB pair as a differential voltage signal, the feature values can be viewed as multiplying with ±1 data stored in the cells, to give discharge currents, which sum together on BL/BLB to generate a voltage. This multiplication-summation operation amounts to an inner product between a feature vector of 5-b elements, and a weight vector of 1-b elements. A comparator at the base of the column provides sign thresholding, as needed in a linear classifier.

In this way, a weak classifier is implemented in each column. In fact, while a linear classifier itself is weak, the use of 1-b weights and the sensitivity of the multiplication-summation operation to analog non-idealities (nonlinearity, variations) further weakens the classifier. To form a strong classifier, a specialized training algorithm is employed. While a range of algorithmic approaches can be considered for different machine learning models (e.g., non-ideal hardware can be used to derive forward-pass error in NNs training), the training algorithm employed critically incorporates two techniques for linear classifiers. First, to train each column-based weak classifier, an algorithm referred to as constrained resolution regression (CRR) [1] is adopted. In CRR, a constraint, included during training to restrict weights to 1 b, is formulated as a convex mixed-integer program. Second, to overcome analog non-idealities, the EACB algorithm [8] is adopted, which performs iterative training of weak classifiers, and biases the training to overcome the errors from the previous iterations. In this way, optimal fitting to the training set is achieved over the entire ensemble, even in the presence of non-idealities. This enables performance at the level of an ideal classifier based on separated SRAM and digital accelerator, as demonstrated on the MNIST dataset [5], but with 13× lower energy and 175× lower energy-delay product (EDP) [1].

B. Hardware Tradeoffs

While substantial energy and throughput gains are demonstrated, to scale up computations, it is important to recognize the new scaling tradeoffs such an architecture raises. Fig. 2(a) shows the scaling tradeoffs for a traditional architecture (i.e., where computation and memory are separated) and the in-memory architecture [1], assuming all D bits of data, arranged in a $D^{1/2} \times D^{1/2}$ array, are to be accessed for computation (order scaling is shown, to illustrate the general trends). As

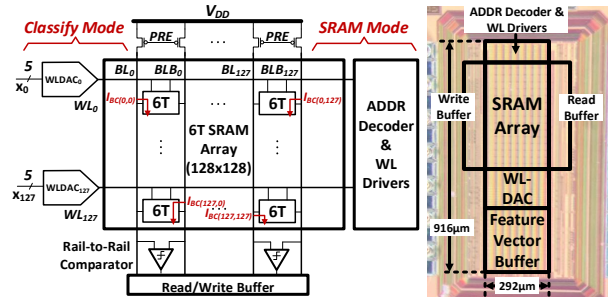
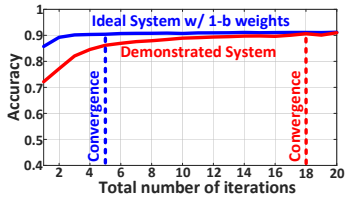


Fig. 1. Hardware architecture and die photo of the in-memory classifier [1].

Metric	Traditional Architecture	In-Memory Architecture
Energy	$D^{3/2}$	D
Latency	D	1
Bandwidth	$1/D^{1/2}$	1
SNR	1	$1/D^{1/2}$

(a)



(b)

Fig. 2. (a) Scaling tradeoffs between traditional and in-memory architectures. (b) Need for increased boosting iterations, due to reduced SNR, in the in-memory classifier [1].

shown, the usual metrics of interest, energy, latency, and bandwidth, all improve compared to the traditional architecture, since accessing all D bits of data requires only one BL/BLB discharge cycle in the in-memory architecture.

However, SNR gets worse. This is because the traditional architecture assumes a constant BL/BLB discharge, set by the sense-amplifier input requirement. On the other hand, in the in-memory architecture, the worst-case SNR occurs when half of the bit cells in a column discharge BL, half of them discharge BLB, and one additional bit cell discharges either BL or BLB. In this case, the differential voltage is set by the one bit cell's current integrating on a capacitance that scales with the number of rows as $D^{1/2}$. While reduced SNR, limited in practice by analog non-idealities, is handled by the EACB training algorithm, excessive SNR degradation will lead to an excessive increase in the number of iterations for performance convergence. For instance, for the 128×128 design in [1], Fig. 2(b) shows that the in-memory architecture requires 18 iterations for convergence, compared to 5 iterations for the traditional architecture. Though the simultaneous accessing of 128 rows still leads to substantial energy benefit, this will degrade with reducing SNR.

III. Algorithm for Scaling-Up

To scale up computations in terms of feature-vector dimensionality, our approach is to decompose the feature vector into several feature subsets. These subsets can then be mapped to separate banks, whose number of rows is set by the SNR requirements. Such decomposition necessitates a specialized algorithm for the in-memory architecture.

A. Extending EACB to AdaBoost.MH

Our algorithm for managing SNR degradation is integrated with EACB and CRR training techniques employed in the in-memory classifier [1]. Since the original EACB algorithm was derived from adaptive boosting (AdaBoost) [10], we present

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2018.2854759, IEEE Transactions on Circuits and Systems II: Express Briefs

Algorithm 1 AdaBoost.MH

Input: $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{D}^{(1)}$, T, η, *WeakLearner*

for t in 1 to T **do**
  $\mathrm{h}^{(t)} \leftarrow$*WeakLearner*$(\mathbf{X}, \mathbf{Y}, \mathbf{D}^{(t)})$
  Compute $\gamma^{(t)}$ and $\alpha^{(t)}$
  Update to $\mathbf{D}^{(t+1)}$: $d_{i,j}^{(t+1)} \leftarrow d_{i,j}^{(t)} \exp(-\alpha^{(t)} \hat{y}_{i,j}^{(t)} y_{i,j})$
  Normalize $\mathbf{D}^{(t+1)}$ to make it a distribution
return $\mathrm{f}^{(T)} = \sum_{t=1}^{T} \alpha^{(t)} \mathrm{h}^{(t)}$

our extension of EACB by starting with AdaBoost. AdaBoost is an ensemble-learning algorithm, based on training simple (weak) classifiers at each iteration and using weighted voting over these classifiers to construct a final strong classifier. To support multi-class classification (as in the case of 10-way MNIST digit recognition), we employ a variant called the AdaBoost.MH algorithm [11], summarized in Algorithm 1. For formal description of the algorithm, let $\mathbf{X} = (\vec{x_1}, \cdots, \vec{x_N})$ be a $d \times N$ *training-data matrix*, where $\vec{x_i}$ is a training sample with dimensionality d. Suppose we are dealing with a K-class problem, let $\mathbf{Y} = (\vec{y_1}, \cdots, \vec{y_N})$ be the corresponding $K \times N$ *training-label matrix*, where $\vec{y_i} \in \{+1, -1\}^K$ is the one-hot encoded label of $\vec{x_i}$. AdaBoost.MH also maintains a $K \times N$ *training-data weighting matrix* as a distribution $\mathbf{D}$ over training-data instances and labels, which is updated at each iteration to increase the emphasis on misclassified instances. In AdaBoost.MH, such updates are based solely on fitting errors, while in EACB they are based on all error sources of potentially non-ideal weak-classifier implementations.

The K-class problem is converted to $\frac{K(K-1)}{2}$ binary one-versus-one problems, and we set the initial distribution $\mathbf{D}^{(1)} = [d_{i,j}^{(1)}]$ uniformly to $1/(KN)$. As mentioned in Section II, the *WeakLearner* in the in-memory classifier is a linear classifier with 1-b weights, trained using CRR. At each iteration t, in total $\frac{K(K-1)}{2}$ *WeakLearners* are combined to build a vector-valued classifier $\mathrm{h}^{(t)}$, which maps the original instance space to $\{+1, -1\}^K$. The final strong classifier $\mathrm{f}^{(T)}$ after T boosting iterations is the weighted sum of all $\mathrm{h}^{(t)}$. The weight $\alpha^{(t)}$ associated with $\mathrm{h}^{(t)}$ is set to $\eta \ln[(1+\gamma^{(t)})/(1-\gamma^{(t)})]$. Here, η, originally set fixed to 0.5 [11], is left as a parameter, which we can set appropriately for the 1-b weight constraint within the column-based linear classifier. $\gamma^{(t)}$ is a metric of the classification performance of $\mathrm{h}^{(t)}$, computed as:

$$\gamma^{(t)} = \sum_{i=1}^{K} \sum_{j=1}^{N} d_{i,j}^{(t)} \hat{y}_{i,j}^{(t)} y_{i,j}, \tag{1}$$

where $\hat{y}_{i,j}^{(t)}$ is the label predicted by $\mathrm{h}^{(t)}$, which in the case of EACB is taken directly from the in-memory classifier to enable adaptation to analog non-idealities. We thus refer to the *WeakLearner* as *NonIdealLearner* in the EACB framing.

B. Combining EACB with MABs

As mentioned, to manage the SNR constraints in the in-memory classifier, we would like to restrict the *NonIdeal-Learners* to utilize a specific fixed subset of the features from the feature vector, instead of the full feature set. Several previous works have explored subset-based ensemble-learning algorithms, with an overview provided in [12]. However, for

Algorithm 2 EACB.MABs

Input: $\{\mathbf{X}_1, \cdots, \mathbf{X}_M\}$, $\mathbf{Y}$, $\mathbf{D}^{(1)}$, T, η, *NonIdealLearner*

Initialize $\vec{\mathrm{p}}^{(1)}$ uniformly
for t in 1 to T **do**
  $j^{(t)} \leftarrow$ *Random*$(\mathrm{p}_1^{(t)}, \cdots, \mathrm{p}_M^{(t)})$
  $\mathrm{h}^{(t)} \leftarrow$*NonIdealLearner*$(\mathbf{X}_j, \mathbf{Y}, \mathbf{D}^{(t)})$
  Compute $\gamma_j^{(t)}$ and $\alpha^{(t)}$
  Update to $\mathbf{D}^{(t+1)}$ and normalize it
  Compute reward $r_j^{(t)}$ for the selected *arm* $j^{(t)}$
  Update to $\vec{\mathrm{p}}^{(t+1)}$ based on $r_j^{(t)}$
return $\mathrm{f}^{(T)} = \sum_{t=1}^{T} \alpha^{(t)} \mathrm{h}^{(t)}$

the in-memory classifier, there are two critical considerations. First, few of the previous algorithms are compatible with boosting, making them difficult to incorporate with EACB, which is required for overcoming analog non-idealities. Second, the algorithms that are based on boosting, use different, randomly-constructed subsets, making them unsuitable for efficient mapping to a small number of in-memory-computing banks, where the feature subset of each bank should be fixed.

In our algorithm, summarized in Algorithm 2, we generate a small number of fixed, mutually-exclusive feature subsets by randomly partitioning the original feature vector. At each iteration, all $\frac{K(K-1)}{2}$ *NonIdealLearners* choose one subset for evaluation. This subset selection procedure can be modeled as a sequential decision problem, which poses high combinatorial complexity to optimize, especially in our case where the impacts of analog non-idealities in the *NonIdealLearners* are not known a priori. To manage this, we employ the construct of MABs [9]. Our algorithm builds on previous works integrating AdaBoost with MABs [13], [14]. The key difference is that these works apply MABs to reduce the search space of tree-based classifiers to speed up AdaBoost. Instead, we exploit MABs for choosing the optimal subset at each iteration, in the presence of analog non-idealities in the *NonIdealLearners*.

In the MABs setting, each *arm* represents a feature subset from the original feature vector. Assume there are in total M *arms*, and the M corresponding $\frac{d}{M} \times N$ *training-data matrices* of feature subsets are denoted as $\{\mathbf{X}_1, \cdots, \mathbf{X}_M\}$. At each boosting iteration t, selecting an *arm* $j^{(t)}$ (i.e., the feature subset $\mathbf{X}_j$) results in a reward $r_j^{(t)}$, and the goal is to maximize the total reward received after T iterations, thus improving the performance of the final strong classifier $\mathrm{f}^{(T)}$. This sets up an *exploration*-vs.-*exploitation* tradeoff, where *exploration* refers to using *arms* that have been scarcely used, and *exploitation* refers to using *arms* that have yielded large reward in the past. The specific MABs algorithm we use is referred to as Exp3.P [9], which maintains a *probability-distribution vector* $\vec{\mathrm{p}}^{(t)}$ over all *arms*. $\vec{\mathrm{p}}^{(t)}$ is initialized uniformly and iteratively updated after receiving a reward for the selected *arm*. The next *arm* is then chosen randomly according to this distribution. As suggested by [14], the *arm* reward is computed as:

$$r_j^{(t)} = \min\left(1, -\log\sqrt{1 - \left(\gamma_j^{(t)}\right)^2}\right), \tag{2}$$

where $\gamma_j^{(t)}$ is computed using (1) based on the chosen feature subset $\mathbf{X}_j$, again using predictions taken from the in-memory

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2018.2854759, IEEE Transactions on Circuits and Systems II: Express Briefs

classifier to overcome analog non-idealities. In this way, the MABs algorithm gradually learns the utility of all feature subsets by evaluating their empirical performance during the boosting iterations, and helps boost the performance close to that based on the full unsegmented feature set. We refer to the proposed algorithm combining EACB and MABs as EACB.MABs. To initialize $\vec{\mathrm{p}}^{(1)}$ and update $\vec{\mathrm{p}}^{(t)}$ based on $r_j^{(t)}$, a *weighting vector* $\vec{\mathrm{w}}^{(t)}$ is also maintained over all *arms* during iterations. The relationship between $\vec{\mathrm{p}}^{(t)}$ and $\vec{\mathrm{w}}^{(t)}$ is:

$$\mathrm{p}_j^{(t)} = (1-\lambda)\frac{\mathrm{w}_j^{(t)}}{\sum_{j=1}^{M}\mathrm{w}_j^{(t)}} + \frac{\lambda}{M}, \tag{3}$$

where $\mathrm{p}_j^{(t)}$ and $\mathrm{w}_j^{(t)}$ are the jth element of $\vec{\mathrm{p}}^{(t)}$ and $\vec{\mathrm{w}}^{(t)}$, respectively. $\lambda \in (0,1]$ is a smoothing parameter: a larger λ leads to a more uniform $\vec{\mathrm{p}}^{(t)}$. $\vec{\mathrm{w}}^{(1)}$ is initialized uniformly to $\exp(\frac{\beta\lambda}{3}\sqrt{\frac{T}{M}})$, where $\beta > 0$ is another smoothing parameter. Based on our experiments, we set $\beta = 0.3$ and $\lambda = 0.2$. After receiving the reward $r_j^{(t)}$, $\vec{\mathrm{w}}^{(t)}$ is updated as follows:

$$\mathrm{w}_j^{(t+1)} = \mathrm{w}_j^{(t)} \exp\left(\frac{\lambda}{3M}\left(\mathrm{q}_j^{(t)} + \frac{\beta}{\mathrm{p}_j^{(t)}\sqrt{MT}}\right)\right), \tag{4}$$

where the *arm-reward vector* $\vec{\mathrm{q}}^{(t)}$ is given as follows (having only one non-zero element at the index $j^{(t)}$, corresponding to the selected *arm*):

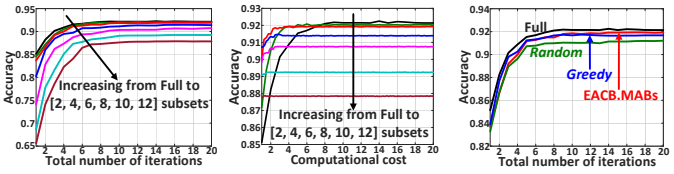
$$\mathrm{q}_j^{(t)} = \begin{cases} r_j^{(t)}/\mathrm{p}_j^{(t)} & \text{if } j = j^{(t)}, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

IV. Simulations and Hardware Experiments

A. Simulation Results

The MNIST dataset [5] is used to evaluate our algorithm, with features corresponding to the raw pixel values. All images are downsampled from 28×28 pixels to 16×16 pixels (required for mapping feature subsets to four in-memory-computing ICs [1]). Simulations correspond to ideal MATLAB implementation of linear classifiers with 1-b weights. 5-fold cross-validation of training and testing is performed to evaluate the classification performance. Also, as the classification accuracy based on feature subsets depends on how we decompose the full feature set, 20 different segmentations are considered in each experiment, and we report the averaged performance results among all segmentations.

We first explore how the total number of feature subsets affects the classification accuracy in the EACB.MABs algorithm. For each experiment, all subsets are mutually exclusive and have roughly the same number of features. The results in Fig. 3(a) show a baseline accuracy of 92% using the full feature vector (i.e., all 256 pixel values), which can also be achieved by using up to 4 subsets. The accuracy using 12 subsets shows 4.5% lower performance. In general, fewer subsets are expected to result in higher accuracy and require fewer iterations, since more features are available at each iteration. However, we note that the higher dimensionality due to fewer subsets results in more computations per iteration, in addition to lower SNR. Thus, Fig. 3(b) shows the computational cost, normalized to one iteration with 256 features, also exposing a tradeoff between computational complexity and accuracy.



(a) (b) (c)

Fig. 3. (a) Accuracy comparison using different numbers of feature subsets. (b) Tradeoff between computational complexity and accuracy. (c) Accuracy comparison using different feature-subset selection approaches at each boosting iteration during training.

Next, we fix to four feature subsets, to compare the performance of EACB.MABs with two other approaches: 1) *Random*, which randomly selects a subset for boosting at each iteration; and 2) *Greedy*, which assesses all subsets at each iteration, and selects the *arm* that maximizes the reward at that iteration. The results are shown in Fig. 3(c). As seen, EACB.MABs performs the best, restoring accuracy to 92%, close to baseline using the full feature vector. While *Random* has the lowest accuracy, *Greedy* achieves performance close to EACB.MABs. However, it requires evaluating all four subsets during training, resulting in 4× more computational cost and energy when mapping subsets to separate ICs.

B. Chip Measurement

To demonstrate our algorithm for adapting to analog non-idealities in the in-memory classifier, we implement a new system combining four banks from separate ICs, as shown in Fig. 4. The in-memory classifiers compute the inner product between a feature vector and a weight vector, which dominates for classification computations. Other pre-processing and post-processing steps necessary for the algorithm (e.g., feature-vector normalization, weighted voting) are performed off-chip via a PC. Communication between the PC and ICs is via a data acquisition (DAQ) system. The DAQ also triggers a 50 MHz clock for in-memory classifiers to operate in *Classify Mode*. Computation mapping to row/column hardware is as follows. Using four feature subsets, 64 features are mapped to the rows of each bank, and 32 rows are used for offset compensation [1] (the remaining 32 rows are disabled). For boosting, a weak classifier (i.e., *NonIdealLearner*) selected at each iteration is mapped to a column of the corresponding bank, and if more than 128 columns are required, weak-classifier computations are performed serially on the bank hardware.

Compared with simulations, there is one additional pre-processing step we need to perform for chip measurement. In the in-memory classifier, the dynamic-range constraint set by the full-swing range of BL/BLB must be considered. As shown in Fig. 5(a), if the common-mode discharge currents on BL/BLB are too small, the voltage difference between BL and BLB is reduced and more prone to comparator error. On the other hand, if the discharge currents on BL/BLB are too large, both BL and BLB can saturate before the comparator is triggered, again leading to comparator error. Thus, normalization of the input data is beneficial for addressing the BL/BLB dynamic-range constraint (note that BL/BLB discharge does not cause bit-cell upsets, due to the low WL voltages employed in *Classify Mode* [1]). Specifically, in our implementation, each MNIST image is normalized so that the sum of all

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCSII.2018.2854759, IEEE Transactions on Circuits and Systems II: Express Briefs

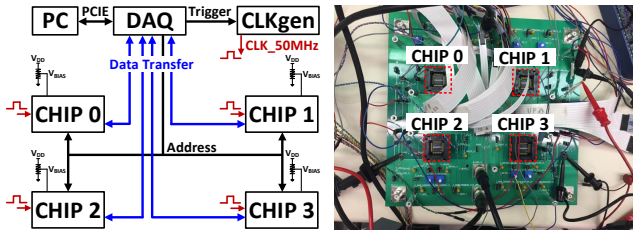
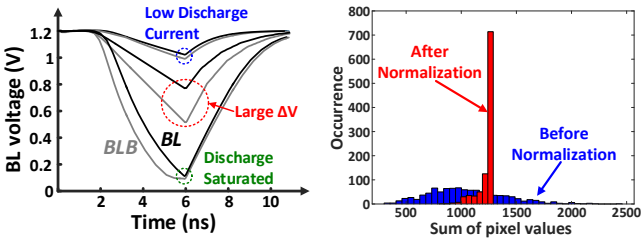


Fig. 4. Four-chip system used for demonstration.



(a) (b)

Fig. 5. (a) BL/BLB discharge in different scenarios. (b) Distributions of pixel-value sum per image (5-b quantization) before and after normalization.

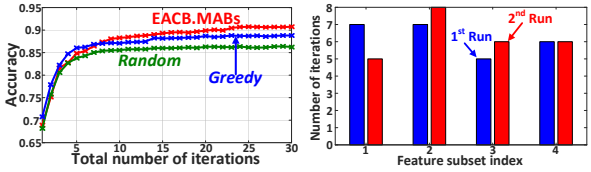
pixel values is roughly the same (about 1250 LSB after 5-b quantization, resulting in 600 mV voltage drop on BL/BLB on average) for all images. Fig. 5(b) shows the distributions of pixel-value sum per image before and after normalization, generated using 100 sample images of each digit. As seen, normalization makes the distribution more concentrated, which maximizes the comparator sensing margin.

Fig. 6(a) shows the measured accuracy for the three approaches. As seen, EACB.MABs achieves the highest accuracy of 91%, close to the ideal expected accuracy from simulations (*Random* achieves 86%, and *Greedy* achieves 89%). 25 EACB boosting iterations are required for performance convergence, resulting in the energy of 879.7 pJ per 10-way classification. This is 14.3× lower than the energy estimated for an ideal system separating memory and computations, which requires 10 iterations for convergence from Fig. 3(c) (assuming memory-accessing energy of 14.7 pJ/cycle measured from *SRAM Mode* of the in-memory classifier [1]). Further, measurement across several runs shows that the utilization of the four subsets is nearly uniform, as seen from Fig. 6(b). This is because the algorithm based on MABs explicitly introduces an *exploration*-vs.-*exploitation* consideration during feature subset selection at each iteration, thus making less frequently used subsets more likely to be selected. The chip measurement results are summarized in Fig. 6(c).

Though the comparison of the demonstrated 4-bank classifier is hard to make with the original 1-bank classifier [1], we point out that the 25 EACB iterations required for convergence, correspond to 64-element feature subsets. This achieves higher accuracy (91%) than that with the 81-element feature vector (90%), which required 18 iterations for convergence. Thus, the accuracy gain by mapping a feature vector of 3× higher dimensionality across four banks is achieved at a modest additional computational cost (<10%).

V. Conclusion

While in-memory-computing classifiers present gains in energy/latency/bandwidth by accessing multiple rows at once,



(a) (b)

Technology	*130 nm*	SRAM Size	*128 × 128 bits*
Number of Chips	*4*	Classification Rate	*50 MHz*
Speed (SRAM Mode)	*300 MHz*	MNIST Perf. (10-way)	*91%*
Energy / 10-way Class.	*879.7 pJ*	Energy Saving	*14.3 ×*

(c)

Fig. 6. (a) Measured accuracy comparison. (b) Utilization of the four subsets from two representative runs. (c) Measurement summary.

they degrade the SNR. To enable design-point optimization, while enabling computations to be scaled up, we present an algorithm based on EACB and MABs to partition features into smaller subsets, which can be mapped to separate banks. Using a four-chip system, performance at the level of an ideal system (separating memory and computation) is achieved for MNIST image classification, yet at 14.3× lower energy.

References

[1] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6T SRAM array," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, Apr. 2017.

[2] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.

[3] M. Kang, M. S. Keel, N. R. Shanbhag, S. Eilert, and K. Curewitz, "An energy-efficient VLSI architecture for pattern recognition via deep embedding of computation in SRAM," in *IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, May 2014, pp. 8326–8330.

[4] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw, "A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6T bit cell enabling logic-in-memory," *IEEE J. Solid-State Circuits*, vol. 51, no. 4, pp. 1009–1021, Apr. 2016.

[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[6] W. S. Khwa, J. J. Chen, J. F. Li, X. Si, E. Y. Yang, X. Sun, R. Liu, P. Y. Chen, Q. Li, S. Yu, and M. F. Chang, "A 65nm 4Kb algorithm-dependent computing-in-memory SRAM unit-macro with 2.3ns and 55.8TOPS/W fully parallel product-sum operation for binary DNN edge processors," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 496–498.

[7] Y. Ji, Y. Zhang, S. Li, P. Chi, C. Jiang, P. Qu, Y. Xie, and W. Chen, "NEUTRAMS: Neural network transformation and co-design under neuromorphic hardware constraints," in *IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–13.

[8] Z. Wang, R. E. Schapire, and N. Verma, "Error adaptive classifier boosting (EACB): Leveraging data-driven training towards hardware resilience for signal inference," *IEEE Trans. Circuits Syst. I*, vol. 62, no. 4, pp. 1136–1145, Apr. 2015.

[9] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The non-stochastic multiarmed bandit problem," *SIAM J. Comput.*, vol. 32, no. 1, pp. 48–77, Feb. 2002.

[10] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997.

[11] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine Learning*, vol. 37, no. 3, pp. 297–336, Dec. 1999.

[12] M. A. Aly and A. F. Atiya, "Novel methods for the feature subset ensembles approach," *Int. J. Artificial Intelligence and Machine Learning*, vol. 6, no. 4, pp. 1–7, 2006.

[13] R. Busa-Fekete and B. Kégl, "Accelerating adaboost using UCB," in *Proc. KDD-Cup 2009 Competition*, Jun. 2009, pp. 111–122.

[14] R. Busa-Fekete and B. Kégl, "Fast boosting using adversarial bandits," in *Int. Conf. on Machine Learning (ICML)*, Jun. 2010, pp. 143–150.