Mingu Kang
Sujan Gonugondla
Naresh R. Shanbhag

Deep In-memory Architectures for Machine Learning

Springer

Deep In-memory Architectures for Machine Learning

Mingu Kang • Sujan Gonugondla
Naresh R. Shanbhag

Deep In-memory Architectures for Machine Learning

Springer

Mingu Kang
IBM Thomas J. Watson Research Center
Yorktown Heights, NY, USA

Sujan Gonugondla
University of Illinois at Urbana-Champaign
Urbana, IL, USA

Naresh R. Shanbhag
University of Illinois at Urbana-Champaign
Urbana, IL, USA

ISBN 978-3-030-35970-6 ISBN 978-3-030-35971-3 (eBook)
https://doi.org/10.1007/978-3-030-35971-3

This Springer imprint is published by the registered company Springer Nature Switzerland AG.
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

The concept of deep in-memory architecture (DIMA) described in this book is based on the doctoral dissertation research of the first two coauthors conducted under the supervision of third coauthor. In 2013–2014, we were in search of a circuit fabric that would exhibit a favorable energy vs. SNR trade-off. The rationale underlying our search was the formulation of a Shannon-inspired model of computation, elements of which we had developed gradually over the past decade and which could exploit such a trade-off to realize machine learning accelerators operating at the fundamental limits of energy-latency-accuracy. By embedding analog computations in the periphery of the memory bitcell array, we were not only able to realize such a circuit fabric but also overcome the memory-processor bottleneck inherent in traditional von Neumann digital architectures.

Developing DIMA was a challenging prospect because it broke many traditional assumptions made in the design of circuits, architectures, and algorithms. The justification for breaking those assumptions required us to connect across the compute layers, thereby making DIMA a full-stack technology. This cross-layer connection is hard to comprehend given the siloed nature of modern-day research enterprise. Not surprisingly, we have had our fair share of rejections of our papers by top conferences when we first started. For example, though our first DIMA paper was published in 2014, it was not until 2018 that the general idea of in-memory computing caught the imagination of circuit designers and architects.

Our journey leading to this book would not have been possible without the support and contributions of a number of individuals and institutions. We begin by acknowledging the numerous discussions and brainstorming sessions with Ameya Patil, Charbel Sakr, Sungmin Lim, and Yongjune Kim, who as members of the Shanbhag Research Group at the University of Illinois at Urbana-Champaign directly influenced many of the ideas described in this book. We appreciate the valuable guidance and feedback from Professors Naveen Verma, Pavan Hanumolu, Boris Murmann, and Rob Rutenbar. Their insightful comments and wisdom helped nurture our then nascent concept of deep in-memory architecture (DIMA) into its current form. Specifically, collaborations with Professor Naveen Verma and his students played a critical role in developing DIMA in its various forms. The material

in Chap. 6 is based on a highly productive research collaboration with Prakalp Srivastava, and Professors Vikram Adve and Nam-Sung Kim, who introduced us to the intricacies of programming languages, instruction set architecture, and compiler design.

Our work was initiated under the auspices of the Systems on Nanoscale Information fabriCs (SONIC) Center (2013–2017) at the University of Illinois at Urbana-Champaign (UIUC). We feel extremely fortunate to have been part of the SONIC community—its faculty, students, and sponsors—which provided us with a fertile and vibrant environment that inspired, encouraged us to explore high-risk high-payoff ideas such as DIMA, and supported our efforts throughout the process. We gratefully acknowledge the contributions of Sean Eilert and Ken Curewitz from Micron Inc., who provided encouragement and feedback on our work specially in its early days. Our collaborators from Intel Corporation, Sasi Manipatruni, Dmitri Nikonov, and Ian Young, introduced us to challenging long-term problems which both excited and challenged our creativity. We appreciate their proactive involvement in our work and their contributions to SONIC's overall success.

Our work would not have been possible without the generous and sustained sponsorship from Linton Salmon (DARPA) and Todd Younkin (SRC—Semiconductor Research Corporation). To this day, both DARPA and SRC are playing a critical role in promoting advanced semiconductor research in the United States and thereby enabling the USA to maintain its leadership in this field.

Our work was conducted at the Coordinated Science Laboratory and the Department of Electrical Engineering at the University of Illinois at Urbana-Champaign. We gratefully acknowledge access to their world-class facilities from office space, to computing and laboratory infrastructure, to administrative and editorial support that made this book possible. We thank Jenny Applequist for her careful reading of our manuscript which improved its quality immensely.

Finally, we would like to acknowledge the critical role played by the members of our respective families: (first author) Steven, Brandon, Stella, and Yiseul; (second author) Surya, Sunitha, and Meghana; (third author) Vinay, Hailan, and Lei. This book would not have been possible without their strong and sustained support that saw us through many difficulties along the way.

Old Tappan, NJ, USA — Mingu Kang
Urbana, IL, USA — Sujan Gonugondla
Urbana, IL, USA — Naresh R. Shanbhag

The original version of this book was revised: Affiliation of the Author "Mingu Kang" has been updated. A correction to this book is available at https://doi.org/10.1007/978-3-030-35971-3_8

Contents

Chapter 1
Introduction

There is much interest in incorporating artificial intelligence (AI) capabilities into various products and services in both the commercial and defense industries today. Though machine learning (ML) algorithms have begun to exceed human performance in cognitive and decision-making tasks [1, 2], they tend to be computationally complex and require processing of large volumes of data. Today, such tasks are realized in the Cloud [3] (see Fig. 1.1a) due to the availability of sufficient computational resources. However, there is growing interest in embedding data analytics into sensor-rich platforms at the Edge including wearables, autonomous vehicles, personal biomedical devices, Internet of Things (IoT) devices and others to provide them with local decision-making capabilities as shown in Fig. 1.1b. Such platforms, though (sensory) data-rich, are heavily constrained in terms of computational resources (storage, processing, and communications), energy, latency, and form factor. Therefore, there is much interest in exploring the implementation of machine learning systems on resource-constrained Edge platforms. This book describes a unique approach to realizing that objective—the *Deep In-memory Architecture* (DIMA).

This chapter begins with description of the energy problem in realizing machine learning workloads on the traditional (digital) von Neumann architecture, followed by a comparative overview of various architectures for implementing machine learning systems.

1.1 The Energy Problem in Machine Learning

Primary metrics for the design of machine learning systems on resource-constrained Edge platforms are: (1) energy-efficiency; (2) decision latency and throughput; and (3) decision(-making) accuracy. Energy efficiency is critical for embedded battery-

M. Kang et al., *Deep In-memory Architectures for Machine Learning*,
https://doi.org/10.1007/978-3-030-35971-3_1

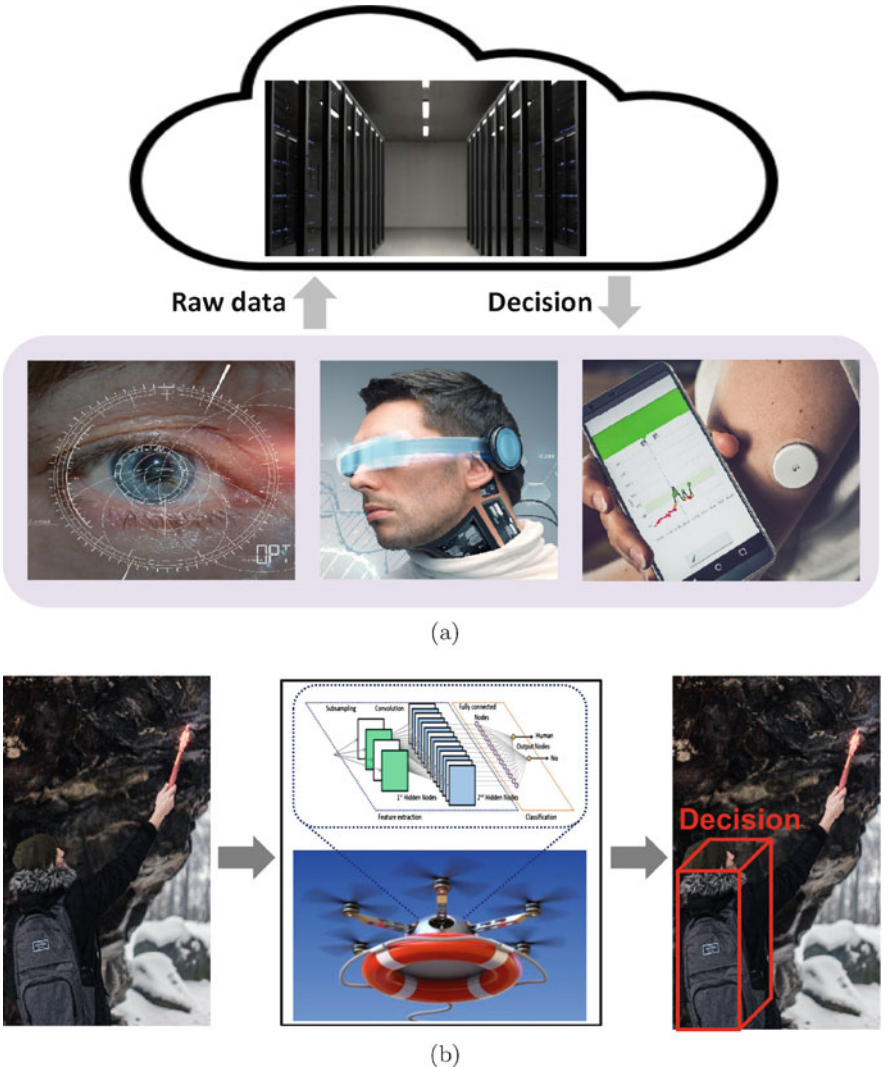


Fig. 1.1 Machine learning (ML) platforms: (**a**) the cloud, and (**b**) at the Edge

powered and autonomous platforms. As a result, a number of integrated circuit (IC) implementations of ML kernels and algorithms have appeared recently [4–9].

For the implementation of ML systems, the mainstream architecture today is the digital (von Neumann) architecture Fig. 1.2a. A key feature of this architecture is the intrinsic separation between memory and processor, i.e., the memory-processor interface. The energy and latency of an ML system realized on a digital architecture include the energy and latency costs of memory accesses via the memory interface and those of the arithmetic operations executed in the processor. It has been well-

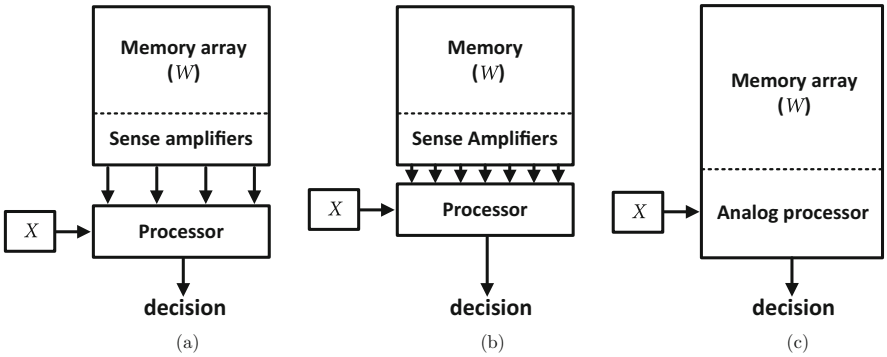


Fig. 1.2 Inference architectures: (**a**) digital (von Neumann) architecture, (**b**) near-memory, architecture, and (**c**) in-memory architecture, where X is an input pattern and W is a stored weight parameter

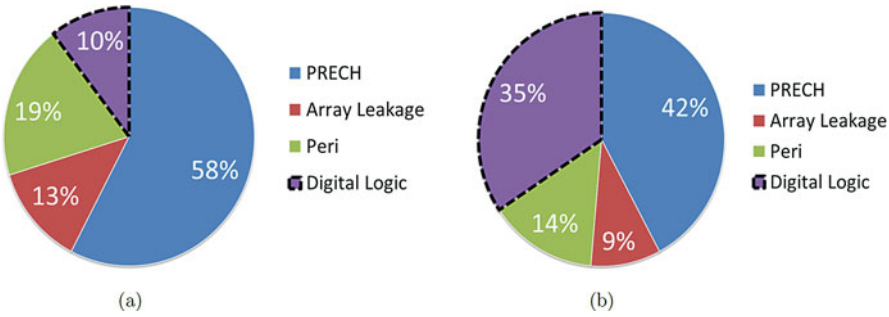


Fig. 1.3 Energy breakdown for two simple pattern matching tasks in a 65 nm process using (**a**) Manhattan distance, and (**b**) cross correlation to find the closest image to a query from 64 candidate images. A 512 × 256 SRAM with synthesized 8-b digital logic is simulated post-layout. The dotted region is from the digital computational block including the clock tree network. The peripheral circuitry (Peri) includes the energy of the sense amplifiers (SAs), decoders, 4:1 column muxes, and WL drivers

established that, the energy and latency costs are dominated by memory accesses [10], e.g., it takes about 20–100 pJ per access of a 16-b word from a 32 kB to 1 MB static random access memory (SRAM) versus 1 pJ per multiply in a 45 nm process [10]. As an additional confirmation of this energy gap, the energy breakdown for two pattern matching tasks realized in a 65 nm process is shown in Fig. 1.3, which shows that memory access energy dominates by taking up to 90% of the total system energy. In addition, recent implementations of deep neural networks (DNN) [4, 9] also report that memory accesses account for the largest portion (between 35% and 45%) of the total energy cost.

Therefore, it is clear that one needs to address the high energy costs of memory accesses even if at the expense of computational energy, if AI capabilities are to be realized on Edge devices.

1.2 Digital ML Architectures

Recently, a number of energy-efficient digital architectures and IC implementations [4–9] for ML systems have been proposed that strive to reduce the number of memory accesses via techniques such as data reuse, minimizing computations, and efficient data-flow.

Several research efforts have tried to minimize the data access cost through architectural optimization. Processor-in-memory (PIM) architectures such as Smart Memory [11–13] and Intelligent RAM [14] locate frequently used logic (e.g., pointer logic [13] or MAC [14]) close to memory by using a wide crossbar. However, physical proximity does not reduce memory read and processing costs.

An effective approach in reducing memory access energy and enhancing throughput for ML algorithms is to reuse data once read from memory. DianNao [9] first identified and exploited the opportunity for massive data reuse across the fetched tile of input and output feature maps in a convolutional neural network (CNN), achieving 21× energy savings. Eyeriss [4, 15] extended the data reuse opportunities at multiple levels (convolutional, filter, and input feature map reuse) to achieve up to 2.5× energy savings for AlexNet.

Low-power circuit techniques have also been explored to achieve energy-efficient processing for ML algorithms. ENVISION [5] implemented the CNN with a dynamic voltage-accuracy-frequency scaling technique given a bit-precision requirement. A speech recognizer with a deep neural network (DNN) [16] has also been introduced that has a voice-activated power-gating technique to enhance energy efficiency during standby mode. The RAZOR technique [17] applied to a sparse DNN engine [7] allows minimizing the supply voltage by tolerating timing errors. These approaches achieve significant energy efficiency, but without exploiting the opportunities afforded by analog processing.

Low-voltage SRAM techniques have been proposed [18, 19] to reduce the energy of memory read accesses. These techniques involve operating the bitcell array (BCA) at voltages in the range of a few hundred mVs, which reduces the throughput significantly into the kHz regime. Low-voltage operation also degrades SRAM's read and write static margins causing catastrophic failure of inference applications when MSB errors occur. Therefore, SRAMs tailored for inference algorithms have been proposed as in [20, 21]. In these works, selective bitline (BL) negative boosting is employed to improve write-ability and protect MSBs during the read operation by the means of selective error correcting code (ECC). However, those techniques suffer from large BL-toggling energy and dropping out of LSBs to accommodate ECC check bits. In [22], a filter approximation technique was employed and accelerated by a 7T SRAM to fetch convolution filter coefficients efficiently by enabling two read modes: row-access and column-access, but at the cost of degraded storage density due to the use of an additional transistor in the bitcell.

To sum up, previous approaches have addressed the energy cost by either co-locating the processor and memory, or by minimizing data accesses (via data reuse) or by employing low-power digital techniques for processing and low-voltage

memories. In contrast, with associative memories [23, 24], simple logic operations are embedded into the BCA to determine a data vector with the minimum Hamming or Euclidean distances from a reference data vector. This embedding is done at the expense of storage density due to the presence of logic circuits in the bitcell. *Kerneltron* [25] also embeds computation (bit-wise multiplication) into the BCA to process and read simultaneously in the charge domain. However, this requires the use of charge injection devices in the BCA and a massive array of ADCs to interface analog and digital processing. Moreover, the need for special devices makes the Kerneltron incompatible with mainstream memory topologies such as SRAM or DRAM.

Near-memory architectures [26–28] (see Fig. 1.2b) distribute computations near memory banks or replace the digital processor with an analog processor in order to save both computational and memory access energy. However, the near-memory architectures preserve the intrinsic separation between the memory and processor.

1.3 In-Memory ML Architectures

Recently, in-memory architectures [29–37] were proposed in which memory access costs in ML systems have been directly addressed by embedding analog computations in close proximity to the memory bitcell array (BCA). Such proximity of computation to the BCA makes in-memory architectures inherently and massively parallel, and well-matched to the data-flow of ML algorithms. However, their intrinsic analog nature makes in-memory architectures susceptible to process, voltage, and temperature (PVT) variations. The use of BL swings and stringent bitcell pitch-matching constraints results in lowered signal-to-noise (SNR) of in-memory computations. Those robustness issues were addressed by a combination of (1) restricting one or both operands to be binary (1-b) [35, 36], (2) using larger (8T or 10T) bitcells [34], and (3) partitioning the array into sub-banks [34] or using separate right and left WLs [36]. Specifically, the restriction on 1-b operands makes it difficult for such in-memory architectures to execute multi-bit operations without sacrificing their energy-latency benefits. Furthermore, the use of binary nets [38], the default network realized by these precision restricted in-memory architectures, leads to increased memory requirements for a fixed accuracy as shown theoretically in [39] and experimentally in [40].

In contrast, the deep in-memory architecture (DIMA) [29–33, 41, 42] embeds *multi-bit mixed-signal computations* in the periphery of a conventional 6T BCA to preserve its storage density, and conventional SRAM read-write functionality. DIMA accesses multiple rows of a standard SRAM BCA per precharge to generate a BL discharge ΔV_{BL} that is proportional to a linear combination of column bits. DIMA then performs scalar operations on the BL outputs via column pitch-matched mixed-signal circuitry in the periphery of the SRAM, followed by a step that aggregates the BL outputs across all the columns of the BCA. Thus, DIMA computations are intrinsically multi-bit, e.g., 8-b in [32, 33] and 5-b in [31], and

strives, to delay the conversion from analog to digital to allow for SNR budgeting across the different stages of analog processing. It can be shown that this *delayed decision property* of DIMA, combined with its cross BL aggregation step effectively compensates for the low-SNR problem inherent to in-memory architectures and thereby leads to accurate inferences [32]. Algorithmic approaches, such as boosting (e.g., AdaBoost) [31] and ensemble methods (e.g., random forest (RF)) [33, 42], have also been leveraged to enhance DIMA's robustness to PVT variations.

1.4 Book Organization

This book is organized as follows. Chapter 2 provides an overview of DIMA. Since DIMA is analog, the chapter provides a systems rationale to quantify the intrinsic robustness of in-memory architectures to analog non-idealities. The rest of the chapter (and the book) focuses on DIMA, since it represents the most aggressive form of in-memory architectures. DIMA design techniques and guidelines are provided to address the circuit and architectural implementation challenges. In addition, this chapter provides energy, delay, and behavioral models with key design parameters.

Chapter 3 presents two DIMA prototype ICs in a 65 nm CMOS process: (1) a multi-functional DIMA that realizes four algorithms, and (2) the random forest (RF) DIMA IC. This chapter also describes techniques for achieving reconfigurability with analog circuitry.

Chapter 4 presents a DIMA IC with an on-chip training loop whose intent is to push the limits of decision-level EDP gains by adapting the model parameters to variations in process parameters, temperature, and voltage as well as data statistics. This chapter shows that to operate at the limits, some form of adaptation and error compensation is required.

While the earlier chapters focus on realization of simple ML kernels based on efficient matrix-vector multiplication (MVM) on DIMA, Chap. 5 describes the mapping of complex ML algorithms such as deep neural networks (DNNs), convolutional neural networks (CNN), and sparse distributed memory (SDM). In particular, algorithmic reformulations such as error-aware retraining (for CNNs) and hierarchical decision-making (for SDM) are shown to enhance the benefits of using DIMA.

Chapter 6 describes a DIMA-based accelerator core with an instruction set architecture (ISA) that is aligned well with the common functional flow of multiple ML benchmarks.

Chapter 7 concludes this book and provides perspectives on future work and trends.

Chapter 2
The Deep In-Memory Architecture (DIMA)

This chapter describes the Deep In-memory Architecture (DIMA). First, the algorithmic data-flow of commonly used ML algorithms is described. DIMA's architectural data-flow is shown to be well-matched to the algorithmic data-flow of those algorithms since it implements a highly efficient matrix-vector multiply (MVM) operation. The rationale underlying DIMA's robustness to PVT variations is presented next. The various stages of analog computations in DIMA are described in detail along with practical circuit and architectural design guidelines. Finally, circuit-aware system models of DIMA's energy, delay, and functionality are presented and employed to evaluate DIMA's decision-making accuracy, energy, and latency trade-offs.

2.1 Data-Flow of Machine Learning Algorithms

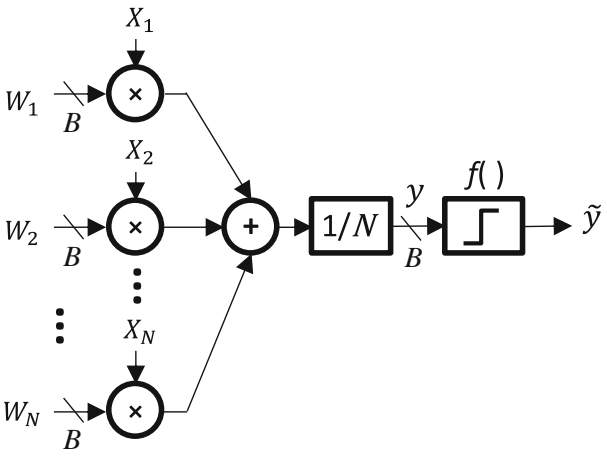
Current-day machine learning algorithms require the implementation of massive numbers of dot-products (DPs). A DP is a measure of the Euclidean distance between two N-dimensional vectors, W and X, and is typically followed by a non-linearity as shown below:

$$y = \sum_{i=1}^{N} \mathbf{w}^{\mathsf{T}}\mathbf{x} \tag{2.1}$$

$$\tilde{y} = f(y) \tag{2.2}$$

where y is the DP output (a scalar), $\mathbf{w} = [W_1, W_2, \ldots, W_N]^{\mathsf{T}}$ and $\mathbf{x} = [X_1, X_2, \ldots, X_N]^{\mathsf{T}}$, and $f(\ )$ is a non-linear function, e.g., a rectified linear unit (ReLU) or sigmoid. In the context of DIMA, the $\mathbf{w}$ is the stored weight vector and $\mathbf{x}$ is the input vector. The algorithmic data flow of (2.1) in Fig. 2.1a shows that the

M. Kang et al., *Deep In-memory Architectures for Machine Learning*,
https://doi.org/10.1007/978-3-030-35971-3_2



(a)

$f\left(D\left(\mathbf{w},\mathbf{x}\right)\right)$	Inner loop kernel $D(\mathbf{w},\mathbf{x})=\sum_{i=1}^{N} d(W_i,X_i)$	$f(\ )$
SVM	$\sum_{i=1}^{N} W_i X_i$	sign
Temp. Match. (L1)	$\sum_{i=1}^{N} \lvert W_i - X_i \rvert$	min
Temp. Match. (L2)	$\sum_{i=1}^{N} (W_i - X_i)^2$	min
DNN	$\sum_{i=1}^{N} W_i X_i$	sigmoid
Feature extraction (PCA)	$\sum_{i=1}^{N} W_i X_i$	--
k-NN (L1)	$\sum_{i=1}^{N} \lvert W_i - X_i \rvert$	majority vote
k-NN (L2)	$\sum_{i=1}^{N} (W_i - X_i)^2$	majority vote
Matched filter	$\sum_{i=1}^{N} W_i X_i$	min
Linear Regression	$\sum_{i=1}^{N} W_i$	accumulate
	$\sum_{i=1}^{N} W_i^2$	accumulate
	$\sum_{i=1}^{N} W_i X_i$	accumulate

(b)

Fig. 2.1 Typical inference algorithms: (**a**) algorithmic data-flow, and (**b**) specifics of a set of commonly used algorithms

DP is obtained by aggregating element-wise scalar products. ML algorithms may employ other notions of VD, as shown in Fig. 2.1b.

ML algorithms have inherent error resiliency for the following reasons: (1) the final thresholding operation to obtain class labels is immune to small-magnitude errors at its input, and (2) the aggregation of a large number of elements in computing a VD enhances the SNR if the computational noise in the constituent SDs is uncorrelated. Those sources of the intrinsic error-tolerance of ML algorithms can be exploited by both the digital architecture and DIMA to reduce the energy and

throughput costs of inference. However, as shown later, DIMA is able to exploit the algorithmic error-tolerance much more effectively.

2.2 DIMA Overview

A high-level view of DIMA (Fig. 2.2a) [29, 41, 43, 44] shows that the data stored in an $N_{\mathrm{ROW}} \times N_{\mathrm{COL}}$ bitcell array (BCA) are processed in four sequentially executed stages:

- the **multi-row functional read** (FR) stage fetches an N-dimensional data vector W consisting of column-major stored B-bit elements by reading B rows (*word-row*) per BL precharge (read cycle). This stage includes the precharge circuitry, the FR WL drivers and the BCA;
- the **BL processing** (BLP) stage computes scalar distances (SDs) between the elements of W and X. The N_{COL} BLP blocks operate in parallel in a single-instruction multiple-data (SIMD) manner;
- the **cross BL processing** (CBLP) stage aggregates the SDs computed by the N_{COL} analog BLP blocks to generate a vector distance (VD); and
- in the final stage an **analog-to-digital converter** (ADC) and **residual digital logic** (RDL) are used to realize for realizing any digital computations that may be necessary such as thresholding/decision function $f(\,)$ and others.

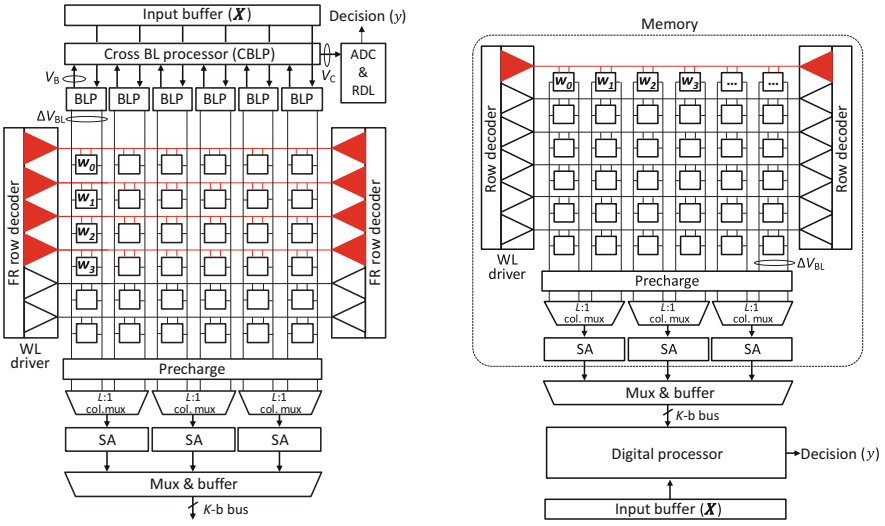


Fig. 2.2 Inference architectures: (**a**) DIMA, and (**b**) the digital (von Neumann) architecture. The blocks marked in red are the wordline drivers that are turned on